



MORGAN & CLAYPOOL PUBLISHERS

Embedded Systems Design with the Texas Instruments MSP432 32-bit Processor

Dung Dang
Daniel J. Pack
Steven F. Barrett

*SYNTHESIS LECTURES ON
DIGITAL CIRCUITS AND SYSTEMS*

Mitchell A. Thornton, *Series Editor*

**Embedded Systems Design
with the Texas Instruments
MSP432 32-bit Processor**



Synthesis Lectures on Digital Circuits and Systems

Editor

Mitchell A. Thornton, *Southern Methodist University*

The *Synthesis Lectures on Digital Circuits and Systems* series is comprised of 50- to 100-page books targeted for audience members with a wide-ranging background. The *Lectures* include topics that are of interest to students, professionals, and researchers in the area of design and analysis of digital circuits and systems. Each *Lecture* is self-contained and focuses on the background information required to understand the subject matter and practical case studies that illustrate applications. The format of a *Lecture* is structured such that each will be devoted to a specific topic in digital circuits and systems rather than a larger overview of several topics such as that found in a comprehensive handbook. The *Lectures* cover both well-established areas as well as newly developed or emerging material in digital circuits and systems design and analysis.

Embedded Systems Design with the Texas Instruments MSP432 32-bit Processor

Dung Dang, Daniel J. Pack, and Steven F. Barrett
2017

Fundamentals of Electronics: Book 4 Oscillators and Advanced Electronics Topics

Thomas F. Schubert, Jr., and Ernest M. Kim
2016

Fundamentals of Electronics: Book 3 Active Filters and Amplifier Frequency Response

Thomas F. Schubert, Jr., and Ernest M. Kim
2016

Bad to the Bone: Crafting Electronic Systems with BeagleBone Black, Second Edition

Steven Barrett and Jason Kridner
2016

Fundamentals of Electronics: Book 2

Thomas F. Schubert, Jr., and Ernest M. Kim
2015

Fundamentals of Electronics: Book 1 Electronic Devices and Circuit Applications

Thomas F. Schubert and Ernest M. Kim
2015

Applications of Zero-Suppressed Decision Diagrams

Tsutomu Sasao and Jon T. Butler

2014

Modeling Digital Switching Circuits with Linear Algebra

Mitchell A. Thornton

2014

Arduino Microcontroller Processing for Everyone! Third Edition

Steven F. Barrett

2013

Boolean Differential Equations

Bernd Steinbach and Christian Posthoff

2013

Bad to the Bone: Crafting Electronic Systems with BeagleBone and BeagleBone Black

Steven F. Barrett and Jason Kridner

2013

Introduction to Noise-Resilient Computing

S.N. Yanushkevich, S. Kasai, G. Tangim, A.H. Tran, T. Mohamed, and V.P. Shmerko

2013

Atmel AVR Microcontroller Primer: Programming and Interfacing, Second Edition

Steven F. Barrett and Daniel J. Pack

2012

Representation of Multiple-Valued Logic Functions

Radomir S. Stankovic, Jaakko T. Astola, and Claudio Moraga

2012

Arduino Microcontroller: Processing for Everyone! Second Edition

Steven F. Barrett

2012

Advanced Circuit Simulation Using Multisim Workbench

David Báez-López, Félix E. Guerrero-Castro, and Ofelia Delfina Cervantes-Villagómez

2012

Circuit Analysis with Multisim

David Báez-López and Félix E. Guerrero-Castro

2011

Microcontroller Programming and Interfacing Texas Instruments MSP430, Part I

Steven F. Barrett and Daniel J. Pack

2011

Microcontroller Programming and Interfacing Texas Instruments MSP430, Part II
Steven F. Barrett and Daniel J. Pack
2011

Pragmatic Electrical Engineering: Systems and Instruments
William Eccles
2011

Pragmatic Electrical Engineering: Fundamentals
William Eccles
2011

Introduction to Embedded Systems: Using ANSI C and the Arduino Development Environment
David J. Russell
2010

Arduino Microcontroller: Processing for Everyone! Part II
Steven F. Barrett
2010

Arduino Microcontroller Processing for Everyone! Part I
Steven F. Barrett
2010

Digital System Verification: A Combined Formal Methods and Simulation Framework
Lun Li and Mitchell A. Thornton
2010

Progress in Applications of Boolean Functions
Tsutomu Sasao and Jon T. Butler
2009

Embedded Systems Design with the Atmel AVR Microcontroller: Part II
Steven F. Barrett
2009

Embedded Systems Design with the Atmel AVR Microcontroller: Part I
Steven F. Barrett
2009

Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller II: Digital and Analog Hardware Interfacing
Douglas H. Summerville
2009

[Designing Asynchronous Circuits using NULL Convention Logic \(NCL\)](#)

Scott C. Smith and JiaDi
2009

[Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller I: Assembly Language Programming](#)

Douglas H. Summerville
2009

[Developing Embedded Software using DaVinci & OMAP Technology](#)

B.I. (Raj) Pawate
2009

[Mismatch and Noise in Modern IC Processes](#)

Andrew Marshall
2009

[Asynchronous Sequential Machine Design and Analysis: A Comprehensive Development of the Design and Analysis of Clock-Independent State Machines and Systems](#)

Richard F. Tinder
2009

[An Introduction to Logic Circuit Testing](#)

Parag K. Lala
2008

[Pragmatic Power](#)

William J. Eccles
2008

[Multiple Valued Logic: Concepts and Representations](#)

D. Michael Miller and Mitchell A. Thornton
2007

[Finite State Machine Datapath Design, Optimization, and Implementation](#)

Justin Davis and Robert Reese
2007

[Atmel AVR Microcontroller Primer: Programming and Interfacing](#)

Steven F. Barrett and Daniel J. Pack
2007

[Pragmatic Logic](#)

William J. Eccles
2007

PSpice for Filters and Transmission Lines

Paul Tobin

2007

PSpice for Digital Signal Processing

Paul Tobin

2007

PSpice for Analog Communications Engineering

Paul Tobin

2007

PSpice for Digital Communications Engineering

Paul Tobin

2007

PSpice for Circuit Theory and Electronic Devices

Paul Tobin

2007

Pragmatic Circuits: DC and Time Domain

William J. Eccles

2006

Pragmatic Circuits: Frequency Domain

William J. Eccles

2006

Pragmatic Circuits: Signals and Filters

William J. Eccles

2006

High-Speed Digital System Design

Justin Davis

2006

Introduction to Logic Synthesis using Verilog HDL

Robert B. Reese and Mitchell A. Thornton

2006

Microcontrollers Fundamentals for Engineers and Scientists

Steven F. Barrett and Daniel J. Pack

2006

Copyright © 2017 by Morgan & Claypool

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

Embedded Systems Design with the Texas Instruments MSP432 32-bit Processor

Dung Dang, Daniel J. Pack, and Steven F. Barrett

www.morganclaypool.com

ISBN: 9781627054959 paperback

ISBN: 9781627059756 ebook

DOI 10.2200/S00728ED1V01Y201608DCS051

A Publication in the Morgan & Claypool Publishers series

SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS

Lecture #51

Series Editor: Mitchell A. Thornton, *Southern Methodist University*

Series ISSN

Print 1932-3166 Electronic 1932-3174

Embedded Systems Design with the Texas Instruments MSP432 32-bit Processor

Dung Dang
Texas Instruments, TX

Daniel J. Pack
The University of Tennessee, Chattanooga, TN

Steven F. Barrett
University of Wyoming, Laramie, WY

SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS #51



MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book provides a thorough introduction to the Texas Instruments MSP432 TM microcontroller. The MSP432 is a 32-bit processor with the ARM Cortex M4F architecture and a built-in floating point unit. At the core, the MSP432 features a 32-bit ARM Cortex-M4F CPU, a RISC-architecture processing unit that includes a built-in DSP engine and a floating point unit. As an extension of the ultra-low-power MSP microcontroller family, the MSP432 features ultra-low power consumption and integrated digital and analog hardware peripherals. The MSP432 is a new member to the MSP family. It provides for a seamless transition to applications requiring 32-bit processing at an operating frequency of up to 48 MHz. The processor may be programmed at a variety of levels with different programming languages including the user-friendly Energia rapid prototyping platform, in assembly language, and in C. A number of C programming options are also available to developers, starting with register-level access code where developers can directly configure the device's registers, to Driver Library, which provides a standardized set of application program interfaces (APIs) that enable software developers to quickly manipulate various peripherals available on the device. Even higher abstraction layers are also available, such as the extremely user-friendly Energia platform, that enables even beginners to quickly prototype an application on MSP432. The MSP432 LaunchPad is supported by a host of technical data, application notes, training modules, and software examples. All are encapsulated inside one handy package called MSPWare, available as both a stand-alone download package as well as on the TI Cloud development site: dev.ti.com The features of the MSP432 may be extended with a full line of BoosterPack plug-in modules. The MSP432 is also supported by a variety of third party modular sensors and software compiler companies. In the back, a thorough introduction to the MSP432 line of microcontrollers, programming techniques, and interface concepts are provided along with considerable tutorial information with many illustrated examples. Each chapter provides laboratory exercises to apply what has been presented in the chapter. The book is intended for an upper level undergraduate course in microcontrollers or mechatronics but may also be used as a reference for capstone design projects. Practicing engineers already familiar with another microcontroller, who require a quick tutorial on the microcontroller, will also find this book very useful. Finally, middle school and high school students will find the MSP432 highly approachable via the Energia rapid prototyping system.

KEYWORDS

MSP432 microcontroller, microcontroller interfacing, embedded systems design, Texas Instruments

To our families.

Contents

	Preface	xxiii
	Acknowledgments	xxvii
1	Introduction to Microcontrollers and the MSP432	1
	1.1 Overview	1
	1.2 Background Theory: A Brief History and Terminology	2
	1.3 Microcontroller Systems	3
	1.4 Why the Texas Instruments MSP432?	4
	1.4.1 MSP432 part numbering system	11
	1.5 MSP-EXP432P401R LaunchPad	12
	1.6 BoosterPacks	12
	1.7 Software Development Tools	15
	1.8 Laboratory Exercise: Getting Acquainted with Hardware and Software Development Tools	16
	1.9 Summary	19
	1.10 References and Further Reading	19
	1.11 Chapter Problems	20
2	A Brief Introduction to Programming	23
	2.1 Overview	23
	2.2 Energia	24
	2.3 Energia Quickstart	24
	2.4 Energia Development Environment	25
	2.4.1 Energia IDE Overview	25
	2.4.2 Sketchbook Concept	26
	2.4.3 Energia Software, Libraries, and Language References	26
	2.5 Energia Pin Assignments	27
	2.6 Writing an Energia Sketch	30
	2.6.1 Control Algorithm for the Dagu Magician Robot	50
	2.7 Some Additional Comments on Energia	63

2.8	Programming in C	63
2.9	Anatomy of a Program	65
2.9.1	Comments	66
2.9.2	Include Files	67
2.9.3	Functions	67
2.9.4	Port Configuration	69
2.9.5	Program Constants	74
2.9.6	Interrupt Handler Definitions	74
2.9.7	Variables	75
2.9.8	Main Program	76
2.10	Fundamental Programming Concepts	77
2.10.1	Operators	77
2.10.2	Programming Constructs	81
2.10.3	Decision Processing	83
2.11	Laboratory Exercise: Getting Acquainted with Energia and C	89
2.12	Summary	90
2.13	References and Further Reading	90
2.14	Chapter Problems	91
3	MSP432 Operating Parameters and Interfacing	93
3.1	Overview	93
3.2	Operating Parameters	94
3.2.1	MSP432 3.3 VDC Operation	94
3.2.2	Compatible 3.3 VDC Logic Families	95
3.2.3	Microcontroller Operation at 5.0 VDC	95
3.2.4	Interfacing 3.3 VDC Logic Devices with 5.0 VDC Logic Families ...	98
3.3	Input Devices	99
3.3.1	Switches	99
3.3.2	Switch Debouncing	101
3.3.3	Keypads	101
3.3.4	Sensors	107
3.3.5	Transducer Interface Design (TID) Circuit	115
3.3.6	Operational Amplifiers	118
3.4	Output Devices	121
3.4.1	Light Emitting Diodes (LEDs)	121
3.4.2	Seven Segment LED Displays	123
3.4.3	Tri-state LED Indicator	125

3.4.4	Dot Matrix Display	127
3.4.5	Liquid Crystal Display (LCD)	133
3.5	High Power DC Interfaces	135
3.5.1	DC Motor Interface, Speed, and Direction Control	139
3.5.2	DC Solenoid Control	150
3.5.3	Stepper Motor Control	150
3.5.4	Optical Isolation	157
3.6	Interfacing to Miscellaneous DC Devices	157
3.6.1	Sonalerts, Beepers, Buzzers	157
3.6.2	Vibrating Motor	158
3.6.3	DC Fan	158
3.6.4	Bilge Pump	158
3.7	AC Devices	159
3.8	Educational Booster Pack MkII	162
3.9	Grove Starter Kit for LaunchPad	164
3.10	Application: Special Effects LED Cube	165
3.10.1	Construction Hints	166
3.10.2	LED Cube MSP432 Energia Code	169
3.11	Laboratory Exercise: Introduction to the Educational Booster Pack MkII and the Grove Starter Kit	185
3.12	Summary	186
3.13	References and Further Reading	186
3.14	Chapter Problems	188
4	MSP432 Memory System	191
4.1	Overview	191
4.2	Basic Memory Concepts	192
4.2.1	Memory Buses	192
4.2.2	Memory Operations	194
4.2.3	Binary and Hexadecimal Numbering Systems	194
4.2.4	Memory Architectures	196
4.2.5	Memory Types	196
4.3	Memory Operations in C Using Pointers	198
4.4	Memory Map	200
4.5	Flash Memory	200
4.5.1	FLCTL Drivelib Support	204

4.6	Direct Memory Access (DMA)	208
4.6.1	DMA Specifications	208
4.6.2	DMA Transfer Types	209
4.6.3	DMA Registers	209
4.6.4	DMA Drivelib Support	211
4.6.5	DMA Example	213
4.7	External Memory: Bulk Storage with an MMC/SD Card	217
4.8	Laboratory Exercise: MMC/SD Card	217
4.9	Summary	221
4.10	References and Further Reading	222
4.11	Chapter Problems	222
5	MSP432 Power Systems	225
5.1	Overview	225
5.2	Background Theory	226
5.3	Operating Modes and Speed of Operation	227
5.4	Power Supply System	227
5.5	The Power Control Module	228
5.6	Operating Modes	229
5.7	Operating Mode Summary	230
5.8	Operating Mode Transitions	230
5.9	PSS and PCM Registers	230
5.10	Battery Operation	233
5.11	DriverLib Support	235
5.12	Programming in C	237
5.13	Laboratory Exercise: Operating Modes	256
5.14	Summary	266
5.15	References and Further Reading	267
5.16	Chapter Problems	267
6	Time-Related Systems	269
6.1	Overview	269
6.2	Background	270
6.3	Time-related Signal Parameters	270
6.3.1	Frequency	270
6.3.2	Period	271

6.3.3	Duty Cycle	271
6.3.4	Pulse Width Modulation	271
6.3.5	Input Capture and Output Compare	272
6.4	MSP432 Clock System	274
6.4.1	Clock Source Registers	276
6.4.2	DriverLib APIs	277
6.4.3	Timer Applications in C	279
6.5	Energia-related Time Functions	286
6.6	Watchdog Timer	289
6.6.1	WDT Modes of Operation	289
6.6.2	WDT System	289
6.6.3	Watchdog DriverLib APIs	290
6.7	Timer32	300
6.7.1	Registers	300
6.7.2	DriverLib APIs	301
6.8	Timer_A	309
6.8.1	Registers	311
6.8.2	DriverLib APIs	312
6.9	Real-Time Clock, RTC_C	325
6.9.1	RTC Registers	325
6.9.2	RTC DriverLib API Support	328
6.10	Laboratory Exercise: Generation of Varying Pulse Width Modulated Signals to Control DC Motors	335
6.11	Summary	335
6.12	References and Further Reading	335
6.13	Chapter Problems	335
7	Resets and Interrupts	337
7.1	Overview	337
7.2	Background	338
7.3	MSP432 Resets	338
7.4	Interrupts	339
7.4.1	Interrupt Handling Process	339
7.5	MSP432 Interrupt System	341
7.5.1	Interrupt Service Routine (ISR)	343
7.6	Energia Interrupt Support	343

7.7	DriverLib	347
7.8	Programming Interrupts in C	348
7.9	Laboratory Exercise: Autonomous Robot	356
7.10	Summary	357
7.11	References and Further Reading	357
7.12	Chapter Problems	357
8	Analog Peripherals	359
8.1	Overview	359
8.2	Background	360
8.3	Analog-to-Digital Conversion	360
8.3.1	Sampling	361
8.3.2	Quantization	363
8.3.3	Encoding	366
8.4	Digital-to-Analog Converter	367
8.5	MSP432 Analog-to-Digital Converter	367
8.5.1	Features	368
8.5.2	Operation	369
8.5.3	ADC Registers	371
8.5.4	Analysis of Results	371
8.6	Programming the MSP432 ADC14 System	372
8.6.1	Energia Programming	372
8.6.2	MSP432 Driver Library	376
8.6.3	Programming ADC14 in C	382
8.7	Voltage Reference	387
8.8	Comparator	392
8.9	Laboratory Exercise: Educational BoosterPack Mk II	396
8.10	Summary	396
8.11	References and Further Reading	396
8.12	Chapter Problems	397
9	Communication Systems	399
9.1	Overview	399
9.2	Background	400
9.3	Serial Communication Concepts	401
9.4	MSP432 UART	403

9.4.1	UART Features	403
9.4.2	UART Overview	404
9.4.3	Character Format	406
9.4.4	Baud Rate Selection	407
9.4.5	UART Associated Interrupts	407
9.4.6	UART Registers	408
9.4.7	API Support	408
9.5	Code Examples	410
9.5.1	Energia	410
9.5.2	UART DriverLib API Example	412
9.5.3	UART C Example	413
9.6	Serial Peripheral Interface-SPI	417
9.6.1	SPI Operation	417
9.6.2	MSP432 SPI Features	418
9.6.3	MSP432 SPI Hardware Configuration	419
9.6.4	SPI Registers	421
9.6.5	SPI Data Structures API Support	422
9.6.6	SPI Code Examples	425
9.7	Inter-Integrated Communication - I ² C Module	432
9.7.1	Overview	432
9.7.2	Programming	432
9.7.3	MSP432 as a Slave Device	433
9.7.4	MSP432 as a Master Device	434
9.7.5	I ² C Registers	435
9.7.6	I ² C API Support	436
9.7.7	I ² C Code Examples	438
9.8	Laboratory Exercise: UART and SPI Communications	445
9.9	Summary	445
9.10	References and Further Reading	446
9.11	Chapter Problems	446
10	MSP432 System Integrity	449
10.1	Overview	449
10.2	Electromagnetic Interference	450
10.2.1	EMI Reduction Strategies	450
10.3	Cyclic Redundancy Check	452
10.3.1	MSP432 CRC32 Module	453

10.3.2	CRC32 Registers	453
10.3.3	API Support	454
10.4	AES256 Accelerator Module	461
10.4.1	Registers	462
10.4.2	API Support	463
10.5	Laboratory Exercise: AES256	473
10.6	Summary	473
10.7	References and Further Reading	473
10.8	Chapter Problems	474
11	System Level Design	475
11.1	Overview	475
11.2	What is an Embedded System?	476
11.3	Embedded System Design Process	476
11.3.1	Project Description	476
11.3.2	Background Research	476
11.3.3	Pre-Design	477
11.3.4	Design	478
11.3.5	Implement Prototype	480
11.3.6	Preliminary Testing	480
11.3.7	Complete and Accurate Documentation	481
11.4	Weather Station	481
11.4.1	Requirements	482
11.4.2	Structure Chart	482
11.4.3	Circuit Diagram	482
11.4.4	UML Activity Diagrams	483
11.4.5	Microcontroller Code	483
11.4.6	Project Extensions	491
11.5	Submersible Robot	491
11.5.1	Approach	492
11.5.2	Requirements	493
11.5.3	ROV Structure	494
11.5.4	Structure Chart	496
11.5.5	Circuit Diagram	499
11.5.6	UML Activity Diagram	499
11.5.7	MSP432 Code	499
11.5.8	Control Housing Layout	514

11.5.9	Final Assembly Testing	514
11.5.10	Final Assembly	516
11.5.11	Project Extensions	516
11.6	Mountain Maze Navigating Robot	516
11.6.1	Description	518
11.6.2	Requirements	518
11.6.3	Circuit Diagram	518
11.6.4	Structure Chart	518
11.6.5	UML Activity Diagrams	521
11.6.6	4WD Robot Algorithm Code	521
11.6.7	Mountain Maze	531
11.6.8	Project Extensions	532
11.7	Laboratory Exercise: Project Extensions	532
11.8	Summary	534
11.9	References and Further Reading	534
11.10	Chapter Exercises	535
	Authors' Biographies	539
	Index	541

Preface

Texas Instruments is well known for its analog and digital devices, in particular, Digital Signal Processing (DSP) chips. Unknown to many, the company quietly developed its microcontroller division in the early 1990s and started producing a family of controllers aimed mainly for embedded meter applications for power companies, which require an extended operating time without recharging. It was not until the mid 2000s, the company began to put serious efforts to present the MSP430 microcontroller family, its flagship microcontrollers, to the academic community and future engineers. Their efforts have been attracting many educators and students due to the MSP430's cost and the suitability of the controller for capstone design projects requiring microcontrollers. The MSP432 is a natural extension to the MSP430 family. It provides 32-bit operation at operating frequencies of up to 48 MHz. In addition, Texas Instruments offers a large number of compatible analog and digital devices that can expand the range of the possible embedded applications of the microcontroller.

We have four goals writing this book. The first is to introduce readers to microcontroller programming. The MSP432 microcontrollers can be programmed using the user-friendly Energia rapid prototype system, assembly language, DriverLib APIs, or a high-level programming language such as C. The second goal of the book is to teach students how computers work. After all, a microcontroller is a computer within a single integrated circuit (chip). Third, we present the microcontroller's input/output interface capabilities, one of the main reasons for developing embedded systems with microcontrollers. Finally, we illustrate how a microcontroller is a component within embedded systems controlling the interaction of the environment with system hardware and software.

Background

This book provides a thorough introduction to the Texas Instrument MSP432 microcontroller. The MSP432 is a 32-bit reduced instruction set (RISC) processor that features ultra-low power consumption and integrated digital and analog hardware.

This book is intentionally tutorial in nature with many worked examples, illustrations, and laboratory exercises. An emphasis is placed on real-world applications such as smart homes, mobile robots, unmanned underwater vehicle, and DC motor controllers to name a few.

Intended Readers

The book is intended to be used in an upper-level undergraduate course in microcontrollers or mechatronics but may also be used as a reference for capstone design projects. Also, practicing engineers who are already familiar with another line of microcontrollers, but require a quick tutorial on the MSP432 microcontroller, will find this book beneficial. Finally, middle school and

high school students can effectively use this book as the MSP432 is highly approachable via the Energia rapid prototyping system.

Approach and Organization

This book provides is designed to give a comprehensive introduction to the MSP432 line of microcontrollers, programming techniques, and interface concepts. Each chapter contains a list of objectives, background tutorial information, and detailed materials on the operation of the MSP432 system under study. Each chapter provides laboratory exercises to give readers an opportunity to apply what has been presented in the chapter and how the concepts are employed in real applications. Each chapter concludes with a series of homework exercises divided into Fundamental, Advanced, and Challenging categories. Because of the tight family connection between the MSP430 16-bit microcontroller and the MSP432 32-bit microcontroller, this book is almost a second edition of our book, “Microcontroller Programming and Interfacing: Texas Instruments MSP430.” With Morgan and Claypool permission much of the common material between the MSP430 and the MSP432 has been carried forward to this book. If you are a seasoned MSP430 practitioner, you will find the transition to the MSP432 seamless.

Chapter 1 provides a brief review of microcontroller terminology and history followed by an overview of the MSP432 microcontroller. The chapter reviews systems onboard the microcontroller and also the MSP432-EXP432P401R evaluation board (MSP432 LaunchPad). The information provided can be readily adapted to other MSP432-based hardware platforms. It also provides a brief introduction to the MSP432 hardware architecture, software organization, programming model and overview of the ARM 32-bit Cortex M4F central processing unit. The chapter concludes with an introduction to the hardware and software development tools that will be used for the remainder of the book. It provides readers a quickstart guide to get the controller up and operate it quickly with the Energia rapid prototyping system. The chapter provides a number of easy-to-follow examples for the novice programmer.

Chapter 2 provides a brief introduction to programming in C. The chapter contains multiple examples for a new programmer. It also serves as a good review for seasoned programmers.

Chapter 3 describes a wide variety of input and output devices and how to properly interface them to the MSP432 microcontroller. The chapter begins with a review of the MSP432 electrical operating parameters followed by a discussion of the port system. The chapter describes a variety of input device concepts including switches, interfacing, debouncing, and sensors. Output devices are discussed including light emitting diodes (LEDs), tri-state LED indicators, liquid crystal displays (LCDs), high power DC and AC devices, motors, and annunciator devices. Also, complete sensor modules available for the MSP432 are also discussed.

Chapter 4 is dedicated to descriptions and use of the different memory components onboard the MSP432 including Flash, RAM, ROM, and the associated memory controllers. The Direct Memory Access (DMA) controller is also discussed.

Chapter 5 provides an in depth discussion of the MSP432 Power Control Manager (PCM). The PCM enables developers to configure various operating modes depending on the application's requirements, and provides for ultra-low power operation and practices.

Chapter 6 presents the clock and timer systems aboard the MSP432. The chapter begins with a detailed description of the flexible clocking features, followed by a discussion of the timer system architecture. The timer architecture discussion includes clocking system sources, the watchdog timer, the real time clock, the flexible 16-bit timer A, the general-purpose 32-bit timers, and pulse width modulation (PWM) features.

Chapter 7 presents an introduction to the concepts of resets and interrupts. The various interrupt systems powered by the Nested Vector Interrupt Controller (NVIC) associated with the MSP432 are discussed, followed by detailed instructions on how to properly configure and program them.

Chapter 8 is dedicated to outline the analog systems aboard the MSP432. The chapter discusses the analog-to-digital converters (ADCs), the internal reference module, and the analog comparators. The chapter also provides pointers to a wide variety of compatible Texas Instrument analog devices.

Chapter 9 is designed for a detailed review of the complement of serial communication systems resident on the MSP432, including the universal asynchronous receiver transmitter (UART), the serial peripheral interface (SPI), the Inter-Integrated Circuit (I2C) system, and the Infrared Data Association (IrDA) link.

Chapter 10 provides a detailed introduction to the data integrity features aboard the MSP432 including a discussion of noise and its sources and suppression, an Advanced Encryption Standard (AES) 256 accelerator module, and a 32-bit cyclic redundancy check (CRC) engine.

Chapter 11 discusses the system design process followed by system level examples. The examples employ a wide variety of MSP432 systems discussed throughout the book.

Dung Dang, Daniel J. Pack, and Steven F. Barrett
September 2016

Acknowledgments

There have been many people involved in the conception and production of this book. We especially want to thank Paul Nossaman and Cathy Wicks of Texas Instruments. The future of Texas Instruments is bright with such helpful, dedicated engineering and staff members. Joel Claypool of Morgan & Claypool Publishers has provided his publishing expertise to convert our final draft into a finished product. We thank him for his support on this project and many others. We also thank him for his permission to port common information from our MSP430 text forward to this book. His vision and expertise in the publishing world made this book possible. We also thank Dr. C.L. Tondo of T&T TechWorks, Inc. and his staff for working their magic to convert our final draft into a beautiful book.

Finally, we thank our families who have provided their ongoing support and understanding while we worked on the book.

Dung Dang, Daniel J. Pack, and Steven F. Barrett
September 2016

Introduction to Microcontrollers and the MSP432

Objectives: After reading this chapter, the reader should be able to:

- briefly describe the key technological accomplishments leading to the development of the microcontroller;
- define the terms microprocessor, microcontroller, and microcomputer;
- identify multiple examples of microcontroller applications in daily life;
- list key attributes of the MSP432 microcontroller;
- list the subsystems onboard the microcontroller and briefly describe subsystem operation;
- provide an example application for each subsystem onboard the MSP432 microcontroller;
- describe the hardware, software, and emulation tools available for the MSP432 microcontroller; and
- employ the development tools to load and execute a sample program on the MSP432 LaunchPad (MSP-EXP432P401R).

1.1 OVERVIEW

This book is all about microcontrollers! A microcontroller is a self-contained processor system in a single integrated circuit (chip) that provides local computational or control resources to a number of products. These computational or control tasks can be as simple as turning on or off a switch, a light, sensing a touch, or can be as complex as controlling a motor, operating in a wireless network, or driving a complex graphical interface. They are used when a product requires a “limited” amount of processing power with a small form factor to perform its mission. They are everywhere! In the routine of daily life, we use multiple microcontrollers. Take a few minutes and jot down a list of microcontroller equipped products you have used today.

This chapter provides an introduction to the Texas Instruments MSP432 microcontroller. We begin with a brief history of computer technology followed by an introduction to the MSP432 microcontroller. We also introduce readers to the powerful and user-friendly development tools in developing embedded system applications.

1.2 BACKGROUND THEORY: A BRIEF HISTORY AND TERMINOLOGY

The development of microcontrollers can be traced back to the roots of early computing with the first generation of computers. The generations of computer development are marked by breakthroughs in hardware and architecture innovation. The first generation of computers employed vacuum tubes as the main switching element. Mauchly and Eckert developed the Electronic Numerical Integrator and Calculator (ENIAC) in the mid 1940's. This computer was large and consumed considerable power due to its use of 18,000 vacuum tubes. The computer, funded by the U.S. Army, was employed to calculate ordnance trajectories in World War II. The first commercially available computer of this era was the UNIVAC I [Bartee, 1972].

The second generation of computers employed transistors as the main switching element. The transistor was developed in 1947 by John Bardeen and Walter Brattain at Bell Telephone Laboratories. Bardeen, Brattain, and William Schockley were awarded the 1956 Nobel Prize in Physics for development of the transistor [Nobel.org, 1980]. The transistor reduced the cost, size, and power consumption of computers.

The third generation of processors started with the development of the integrated circuit. The integrated circuit was developed by Jack Kilby at Texas Instruments in 1958. The integrated circuit revolutionized the production of computers, greatly reducing their size and power consumption. Computers employing integrated circuits were first launched in 1965 [Bartee, 1972]. Kilby was awarded the Nobel Prize in Physics in 2002 for his work on the integrated circuit [Nobel.org, 1980]. The first commercially available minicomputer of this generation was the Digital Equipment Corporation's (DEC) PDP-8 [Osborne, 1980].

The fourth generation of computers was marked by the advancement of levels of integration, leading to very large scale integration (VLSI) and ultra large scale integration (ULSI) production techniques. In 1969, the Data Point Corporation of San Antonio, Texas had designed an elementary central processing unit (CPU). The CPU provides the arithmetic and control for a computer. Data Point contracted with Intel and Texas Instruments to place the design on a single integrated circuit. Intel was able to complete the task, but Data Point rejected the processor as being too slow for their intended application [Osborne, 1980].

Intel used the project as the basis for their first general purpose 8-bit microprocessor, the Intel 8008. The microprocessor chip housed the arithmetic and control unit for the computer. Other related components such as read only memory (ROM), random access memory (RAM), input/output components, and interface hardware were contained in external chips. From 1971 to 1977, Intel released the 8008, 8080, and 8085 microprocessors which significantly reduced

the number of system components and improved upon the number of power supply voltages required for the chips. Some of the high visibility products of this generation were the Apple II personal computer, developed by Steve Jobs and Steve Wozniak and released in 1977, and the IBM personal computer, released in 1981 [MCS 85, 1977, Osborne, 1980].

The first single chip microcontroller was developed by Gary Boone of Texas Instruments in the early 1970's. A microcontroller contains all key elements of a computer system within a single integrated circuit. Boone's first microcontroller, the TMS 1000, contained the CPU, ROM, RAM, and I/O and featured a 400 kHz clock [Boone, 1971, 1977]. From this early launch of microcontrollers, an entire industry was born. There are now over 35 plus companies manufacturing microcontrollers worldwide offering over 250 different product lines. The 16-bit MSP430 line of microcontrollers was first developed in 1992 and became available for worldwide release in 1997. The 32-bit MSP432 microcontroller was formally released by Texas Instruments in 2015.

1.3 MICROCONTROLLER SYSTEMS

Although today's microcontrollers physically bear no resemblance to their earlier computer predecessors, they all have similar architecture. All computers share the basic systems shown in Figure 1.1. The processor or central processing unit (CPU) contains both datapath and control hardware. The datapath is often referred to as the arithmetic logic unit (ALU). As its name implies, the ALU provides hardware to perform the mathematical and logic operations for the computer. The control unit provides an interface between the computer's hardware and software. It generates control signals to the datapath and other system components such that operations occur in the correct order and within an appropriate time to execute the actions of a software program.

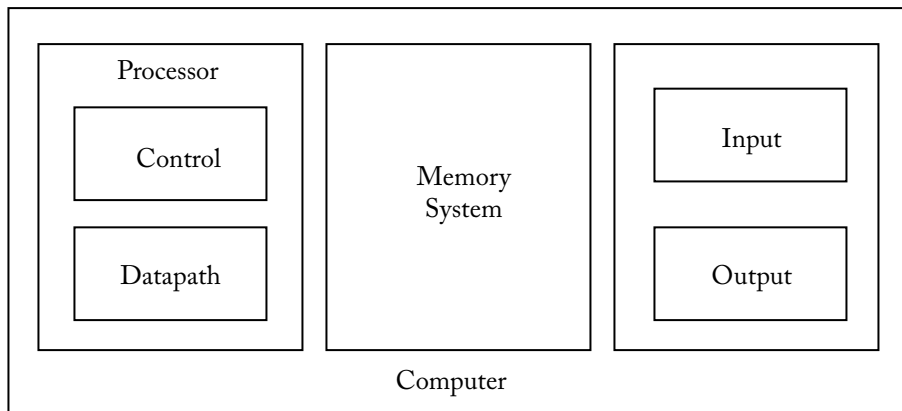


Figure 1.1: Basic computer architecture. (Adapted from Patterson and Hennessy [1994].)

The memory system contains a variety of memory components to support the operation of the computer. Typical memory systems aboard microcontrollers contain Random Access Mem-

4 1. INTRODUCTION TO MICROCONTROLLERS AND THE MSP432

ory (RAM), Read Only Memory (ROM), non-volatile memory such as Flash or Ferro-Random Access Memory (FRAM),¹ and Electrically Erasable Programmable Read Only Memory (EEPROM) components. RAM is volatile. When power is unavailable, the contents of RAM memory is lost. RAM is typically used in microcontroller operations for storing global variables, local variables, which are required during execution of a function, and to support heap operations during dynamic allocation activities. In contrast, ROM memory is nonvolatile. That is, it retains its contents even when power is not available. ROM memory is used to store system constants and parameters. If a microcontroller is going to be mass produced for an application, the resident application program may be written into ROM memory at the manufacturer.

EEPROM is available in two variants: byte-addressable and flash programmable. Byte-addressable memory EEPROM, as its name implies, allows variables to be stored, read, and written during program execution. The access time for byte-addressable EEPROM is much slower than RAM memory; however, when power is lost, the EEPROM memory retains its contents. Byte-addressable EEPROM may be used to store system passwords and constants. For example, if a microcontroller-based algorithm has been developed to control the operation of a wide range of industrial doors, system constants for a specific door type can be programmed into the microcontroller onsite when the door is installed. Flash EEPROM can be erased or programmed in bulk. It is typically used to store an entire program.

The input and output system (I/O) of a microcontroller usually consists of a complement of ports. Ports are fixed sized hardware registers that allow for the orderly transfer of data in and out of the microcontroller. In most microcontroller systems, ports are equipped for dual use. That is, they may be used for general purpose digital input/output operations or may have alternate functions such as input access for the analog-to-digital (ADC) system.

Our discussion thus far has been about microcontrollers in general. For the remainder of this chapter and the rest of the book, we concentrate on the Texas Instruments MSP432 microcontroller.

1.4 WHY THE TEXAS INSTRUMENTS MSP432?

The 16-bit MSP430 line of microcontrollers began development in 1992. Since this initial start, there have been multiple families of the microcontroller developed and produced with a wide range of features. This allows one to choose an appropriate microcontroller for a specific application. As a natural new member to the MSP family, the MSP432 32-bit microcontroller was released in 2015. To support the use of the MSP43x processor family, Texas Instruments invests considerable resources in providing support documentation, development tools, and instructional aids.

A block diagram of the MSP432P401x is provided in Figure 1.2. The MSP432 has the following features:

¹FRAM technology is only available on MSP430 MCU today.

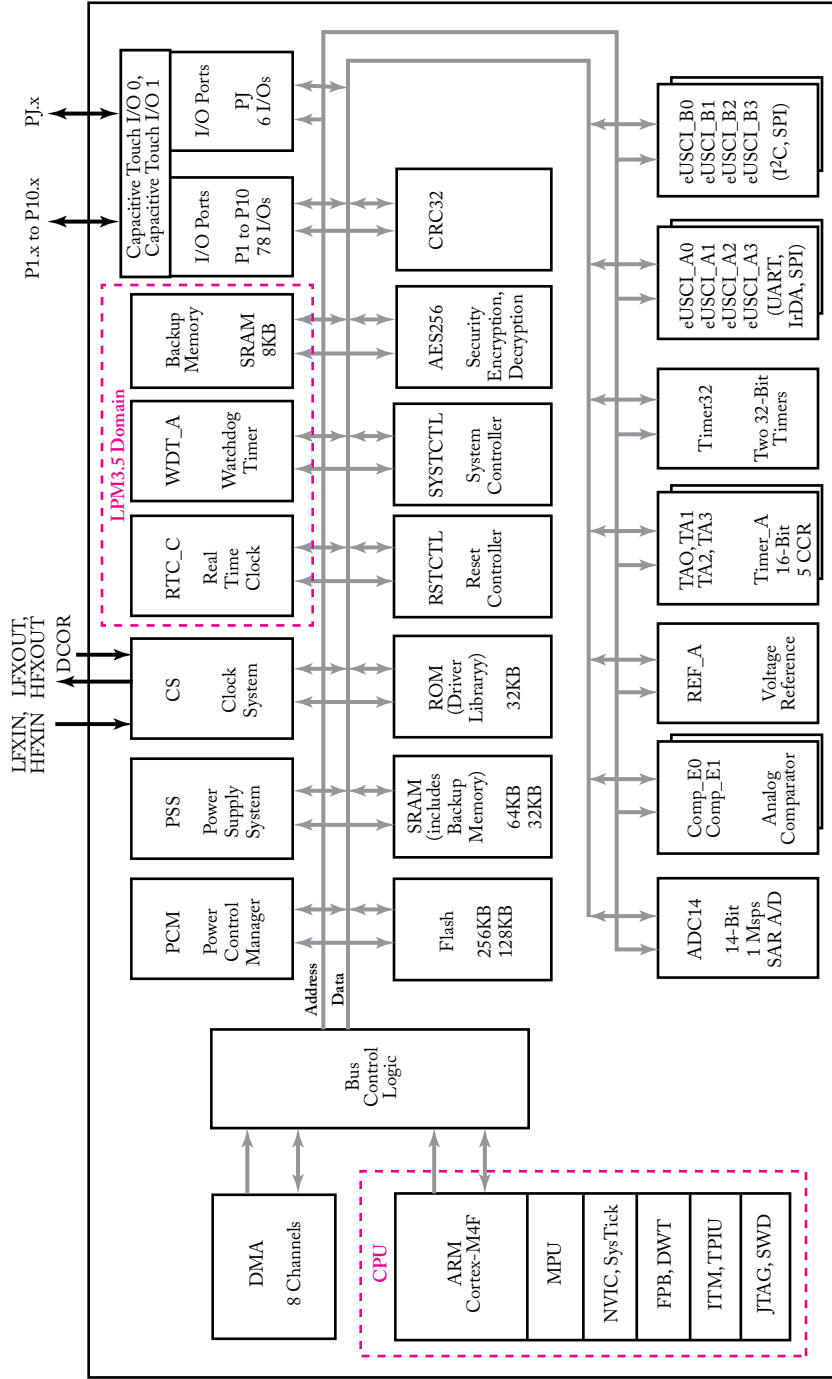


Figure 1.2: Basic MSP432P401x block diagram [SLAS826A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

6 1. INTRODUCTION TO MICROCONTROLLERS AND THE MSP432

- mechanism to operate in a wide supply voltage range (1.62–3.7 VDC);
- ultra-low power consumption functional units;
- clock system that allows operating frequency of up to 48 MHz;
- 32-bit Reduced Instruction Set (RISC) architecture;
- onboard floating point unit (FPU);
- wide variety of memory assets with direct memory access (DMA) capability;
- flexible clocking features;
- wide variety of timing features;
- capability to integrate digital and analog components;
- serial communication subsystems including UART, SPI, IrDA, and I2C. Compatibility to work with peripheral components to provide wireless communications;
- onboard analog-to-digital converters (ADC) and analog comparators;
- encryption and data integrity accelerators;
- support for the full ARM Cortex-M4F instruction set; and
- full range of documentation and support for the student, design engineer, and instructor.

The importance of most of these features is self-evident. A few need to be briefly described. Feature information provided below is from [Barrett and Pack, 2006, SLAS826A, 2015].

32-bit RISC architecture. Reduced Instruction Set Computer (RISC) architecture is based on the premise of designing a processor that is very efficient in executing a basic set of building block instructions. From this set of basic instructions more complex instructions may be constructed. The 32-bit data width establishes the range of numerical arguments that may be used within the processor. For example, a 32-bit processor can easily handle 32-bit unsigned integers. This provides a range of unsigned integer arguments from 0 to $(2^{32} - 1)$ or approximately 4.29G. Larger arguments may be handled, but additional software manipulation is required for processing, which consumes precious execution time.

Integrated digital and analog components. Microcontrollers are digital processors. However, they are used to measure physical parameters in an analog world. Most physical parameters such as temperature, pressure, displacement, and velocity are analog in nature. A microcontroller is typically used in applications where these physical parameters are measured. Based on the measurements, a control algorithm hosted on the microcontroller will issue control signals to control a physical process that will affect the physical world. Often these control signals may be digital,

analog, or a combination of both. The MSP432 family is equipped with a large complement of analog-to-digital converters to process analog input signals. The MSP432 is also equipped with two analog comparators. The MSP432 is compatible with a large complement of analog components.

RF connectivity. Microcontrollers are often used in remote or portable applications. For example, they may be used to monitor the energy consumption of a home, power (kW) consumed over time (kWh). The power utility company charges customers by the kWh based on their monthly power consumption. A person may be employed by the power company to read the energy meters. Alternately, a microcontroller integrated within the energy meter may track monthly consumption and report the value when polled via an RF link. The MSP432 has several RF compatible transmitters and receivers that may be easily interfaced to the microcontroller for such applications.

Low supply voltage range: 1.62–3.7 VDC. The MSP432 operates at very low voltages. Some operating voltage of interest include:

- 3.7V: close to Li-Ion battery supply range (rechargeable electronic battery);
- 1.8–3.3V: 2x AA or AAA batteries, coin-cell applications, and energy harvesting applications. In energy harvesting techniques, energy is derived from sources external to the microcontroller; and
- 1.62V: $1.8 \pm 10\%$: many modern sensors/consumer electronics operate at 1.8V, being able to run the microcontroller down to this range means the whole system can operate natively in the ideal $VCC = 1.8V$.

Ultra-low power consumption. The MSP432 has an active mode (AM) and multiple low power modes (LPM). In the active mode, the MSP432 draws 90 microamps of current per MHz of clock speed. Similarly, in standby mode (LPM3), the MSP432 consumes down to 850nA. In many low-power or energy-harvesting applications where the microcontroller can sleep most of the time and only periodically wake up to perform a task for a short period of time, this low LPM3 current can translate to 10–20 years of operation on a single battery charge.

Onboard flash memory: 256 kbytes. Flash memory is used to store programs and, also, system constants that must be retained when system power is lost (non-volatile memory). The 256 kbytes of flash memory allow a memory space for substantial program development.

Onboard RAM memory: 64 kbytes. The MSP432 also hosts a large RAM memory. RAM memory is used for global variables, local variables, and the dynamic allocation of user-defined data types during program execution.

Power Control Manager (PCM). Texas Instruments has spent considerable effort minimizing the power consumption of the MSP432 microcontroller under varying conditions. Flexibility is provided to the designer with multiple Low Power Modes (LPM): 0, 3, 3.5, 4, and 4.5. Also, additional power saving features have been implemented including turning off clocks to peripheral devices when not in use, removing power from portions of circuitry when not in use,

and providing flexible “wake-up” prompts while in the CPU sleep modes. Also, a power policy manager has been implemented to optimize low power performance with little input needed from the designer [SLAA668, SPRY282].

Large complement of input/output ports. The MSP432P401R is a 100-pin integrated circuit. Most of the pins have multiple functions as shown in Figure 1.3. The general purpose input/output (I/O) pins have been subdivided into ports (P10[5:0], P9[7:0] through P1[7:0], and PJ[5:0]). The I/O pins are all equipped with capacitive touch capability. Up to 48 pins are equipped with interrupt and wake-up capability.

Clock system. Microcontrollers are synchronous circuits. That is, all microcontroller operations are synchronized with a clock circuit. There are a number of clock source options available for the MSP432, including a 32 kHz crystal (LFXT), an internal very low frequency oscillator (VLO), an internal trimmed low-frequency oscillator (REFO), a module oscillator (MODOSC), a system oscillator (SYSOSC), and a high-frequency crystal oscillator up to 48 MHz (HFXT).

Four 16-bit timers and two 32-bit timers. The MSP432 has four separate, 16-bit timer channels (TA0, TA1, TA2, and TA3) and two 32-bit timers. These timers are employed for capturing the parameters of an incoming signal (period, frequency, duty cycle, pulse length), generating an internal interval, generating a precision output digital signal, or generating a pulse width modulated (PWM) signal.

Eight universal serial communication interfaces (eUSCI_A0 to A3 and eUSCI_B0 to B3). The eUSCI system is used to provide serial communications. The MSP432 is equipped with eight separate eUSCI channels designated eUSCI_A0 to A3 and eUSCI_B0 to B3. The eUSCI channels are quite versatile. They provide many different types of serial communications including the following.

- Synchronous serial peripheral interface (SPI). The SPI provides relatively high-speed serial, two-way communication between a transmitter and a receiver. Synchronization is maintained between the transmitter and receiver using a common clock signal provided by the master designated device.
- Inter-Integrated Circuit (I2C) (also known as inter IC, IIC, or I^2C). I2C was developed by Philips to provide for a two-wire serial communications between components on the same circuit board or those within close proximity of one another. It is especially useful in microcontroller-based systems employing multiple peripheral components such as sensors, input devices, and displays.
- Universal asynchronous receiver transmitter (UART). The UART is used to communicate with RS232 compatible devices. The RS232 serial communications standard has been around for some time but it is still popular in many applications. Synchronization is maintained between a transmitter and a receiver using start and stop bits to frame the data.

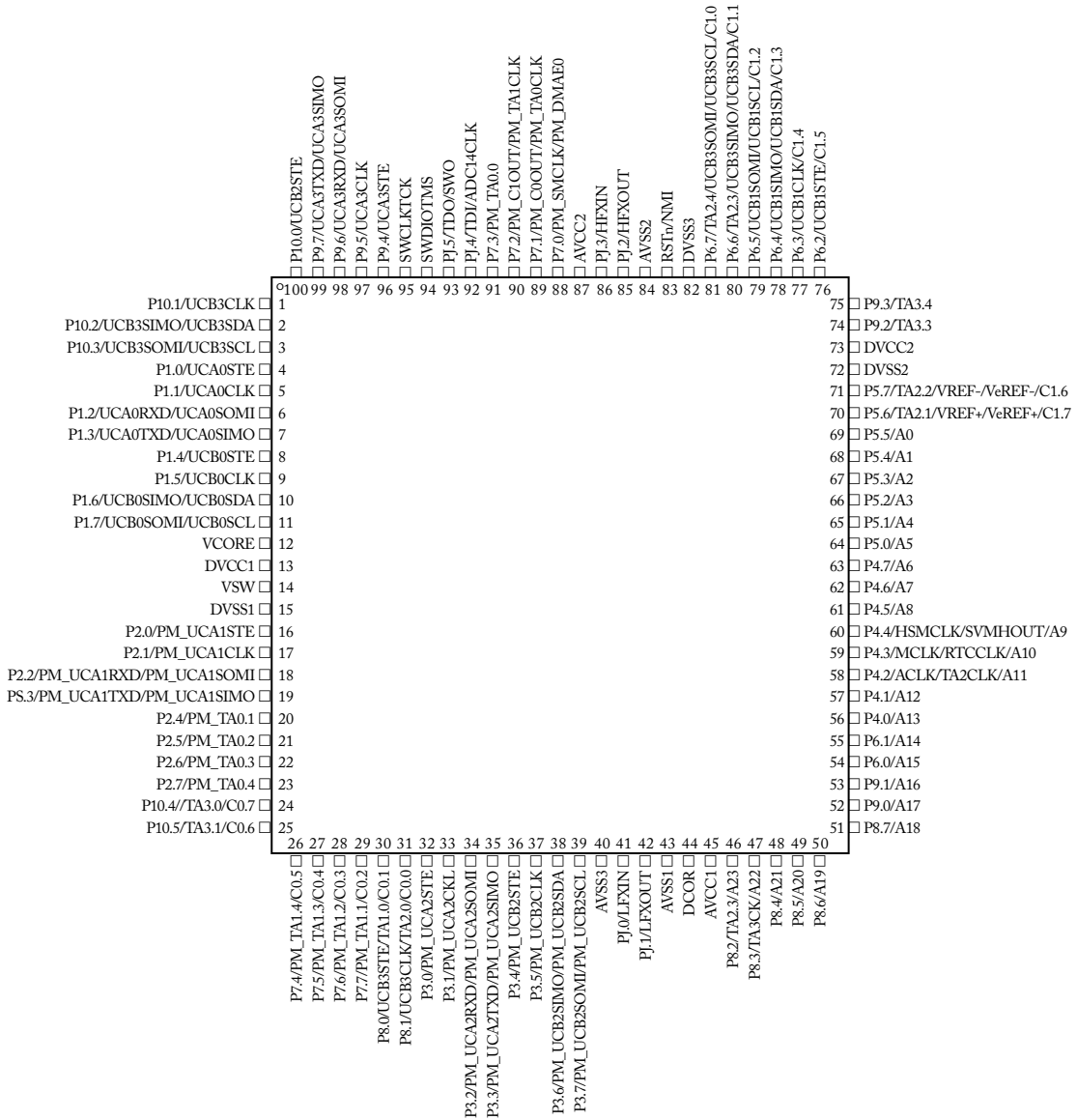


Figure 1.3: MSP432P401R in 100-pin QFP package [SLAS826A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

- Infrared Data Association (IrDA). The IrDA provides for the short range exchange of data using an infrared link. It is useful in hazardous environments where RF communications may not be possible.

Thirty two input channel 14-bit analog-to-digital converter. The MSP432 is equipped with the ADC14 analog-to-digital converter system. The system has 32 channels of 14-bit ADC. This capability provides for a large number of conversion channels and very good resolution of analog signals converted.

High-performance arithmetic. Instead of a hardware multiplier (MPY) like MSP430, the MSP432 includes the following high-performance arithmetic processing features:

- Digital Signal Processing (DSP) instruction set that can be extremely powerful for vector-based math or filtering operations;
- hardware multiply and divide; and
- built-in floating point engine that outperforms a software implementation of a floating point operation.

Many microcontrollers can perform mathematical multiplication operations. However, many perform these calculations using a long sequence of instructions that can consume multiple clock cycles. That is, it takes a relatively long period of time to perform a multiplication operation. The MSP432 Cortex-M4F CPU is equipped with a dedicated hardware multiplier and a dedicated hardware divider that can perform signed and unsigned multiplication and division operations. The hardware multiplier and divider can also perform the signed and unsigned multiply and accumulate or divide operations. Extending beyond just multiplication and division operations, the Cortex-M4F also includes digital signal processing (DSP) instruction set that enables DSP or vector-based math operations. This operation is used extensively in digital signal processing (DSP) operations.

Floating Point Unit. The MSP432 is equipped with a hardware floating point unit (FPU). It provides single-precision arithmetic based on the IEEE 754 standard for adding, subtracting, multiplying, dividing, and performing square root. The FPU can also perform the multiply and accumulate (MAC) function common in digital signal processing.

Eight channels of internal direct memory access (DMA). Memory transfer operations from one location to another typically requires many clock cycles involving the central processing unit (CPU). The MSP432 is equipped with eight DMA channels that allow memory-to-memory location transfers without involving the CPU, freeing the CPU to perform other instructions.

Real-time clock (RTC). Microcontrollers keep time based on elapsed clock ticks. They do not “understand” the concepts of elapsed time in seconds, hours, etc. The RTC provides a general purpose 32-bit counter while in the Counter Mode or an RTC in the Calendar Mode. Both timer modes can be used to read or write counters using software.

Cyclic redundancy check (CRC) generator. Data stored aboard a microcontroller may be corrupted by noise sources flipping 1's to 0's and vice versa. The MSP432 is equipped with the

CRC32 subsystem, which employs the CRC-CCITT standard to calculate a checksum for a block of data. The checksum is also stored with the data. When the data is used, a checksum is calculated again and compared to the stored value. If the values agree, the data is considered good for use. Alternately, if the checksums do not agree, the data is considered corrupted and not available for use.

AES256 Encryption Accelerator. The MSP432 is also equipped with a hardware-based Advanced Encryption Standard (AES) accelerator. The accelerator speeds up the encryption and decryption of data by one to two orders of magnitude over a software-based implementation.

1.4.1 MSP432 PART NUMBERING SYSTEM

The MSP432 part numbering system employed by Texas Instruments is shown in Figure 1.4. The MSP designator indicates the “432” line of processors that employ mixed signal processing (MSP) techniques. That is, it can process digital and analog signals. The “432” indicates a 32-bit, low power platform. The next letter following MSP432 “P” indicates the processor is a member of the performance and low-power series. The “401R” indicates a flash-based memory device operating up to 48 MHz equipped with the ADC14 converter, 256-Kbytes of flash memory, and 64-Kbytes of RAM (www.ti.com).

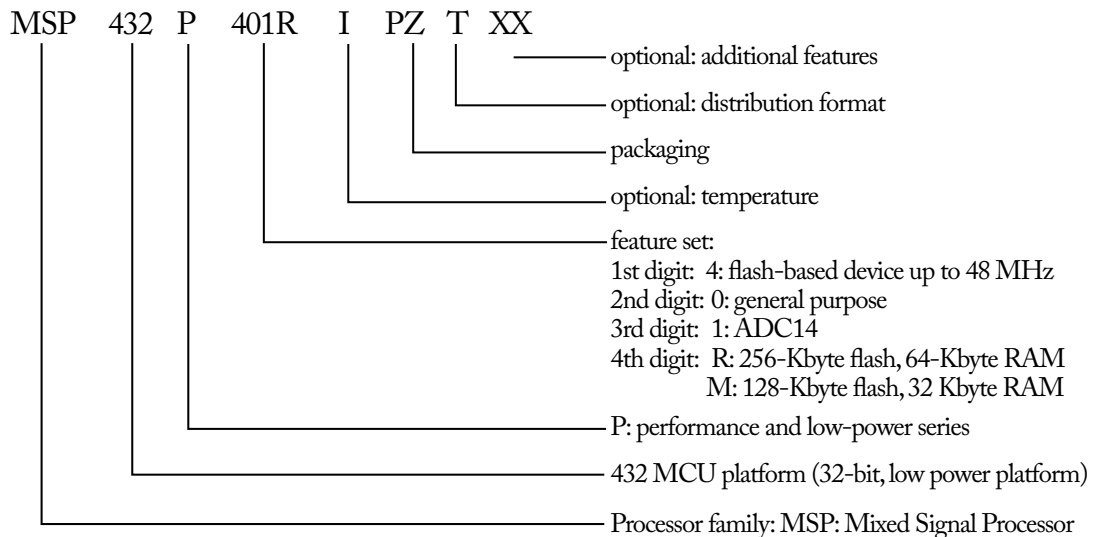


Figure 1.4: Texas Instruments MSP432 numbering system. (Adapted from Texas Instruments.)

As previously mentioned, the MSP-EXP432P401R LaunchPad is equipped with the MSP432P401R microcontroller.

1.5 MSP–EXP432P401R LAUNCHPAD

Throughout the book we employ the MSP–EXP432P401R LaunchPad. This easy-to-use evaluation module (EVM) hosts the MSP432P401R processor, as shown in Figure 1.5.

The LaunchPad is divided into two sections. The top portion hosts the XDS110–ET on-board emulator, featuring the energy-aware debugging technology called EnergyTrace. The “XDS” provides a communication link between the host personal computer (PC) and the LaunchPad, allowing remote programming and debugging of the MSP432P401R processor. The XDS also provides power to the entire LaunchPad via the USB connection from the host PC [SLAU597A, 2015].

The LaunchPad’s top section also has the EnergyTrace Technology (ETT). The ETT provides real-time monitoring of power consumption. The power consumption is monitored via the host PC-based graphical user interface (GUI). The ETT is essential in developing ultra-low power applications [SLAU597A, 2015].

The lower section of the LaunchPad hosts the MSP432P401R microcontroller and associated interface. Interface components include two switches (S1 and S2) and two light-emitting diodes (LEDs), designated LED1 and LED2. At the bottom of the LaunchPad is a breakout section for user available pins. Also, the LaunchPad is equipped with a 40-pin standard interface for MSP432 compatible BoosterPacks [SLAU597A, 2015].

The link between the XDS and the MSP432P401R microcontroller portions of the LaunchPad is provided by the Jumper Isolation block. The block consists of header pins with removable jumpers to allow connection/disconnection between the two sections as a specific application dictates. The J6 header at the lower corner of the LaunchPad allows for external power. The LaunchPad may be powered from 1.62–3.7 VDC [SLAU597A, 2015].

A wide variety of BoosterPacks are available to extend the features of the LaunchPad. BoosterPacks are connected to the LaunchPad via the standard 40-pin interface consisting of header pins J1–J4. The standard configuration is shown in Figure 1.6.

1.6 BOOSTERPACKS

BoosterPacks extend the features of the MSP432. There are a wide variety of MSP432 BoosterPacks available including (www.ti.com):

- stepper motor drivers,
- RFID transponders,
- motor drivers,
- data converters,
- display development tools,

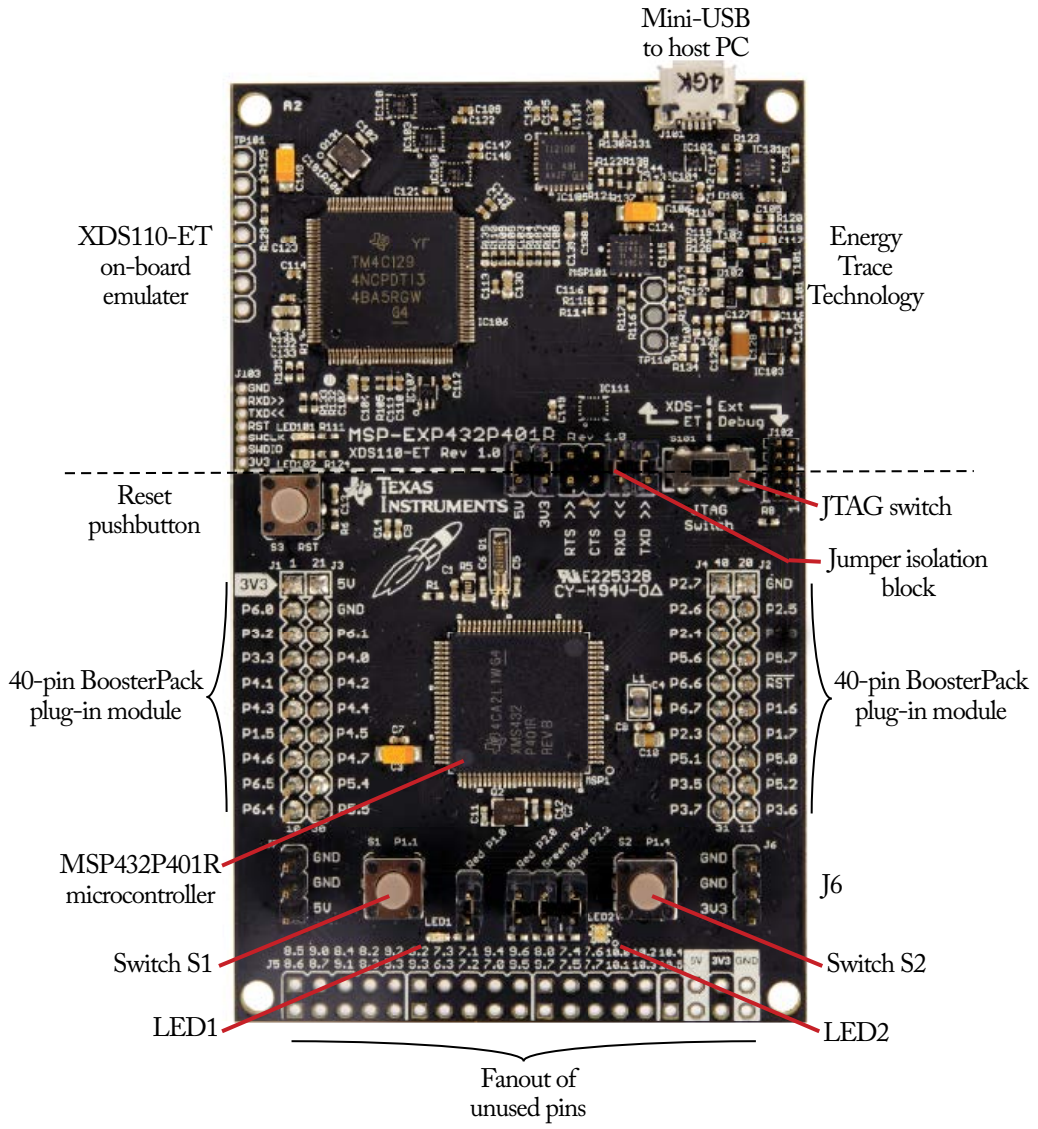


Figure 1.5: MSP432 LaunchPad. (Adapted from Texas Instruments [SLAU597A, 2015].) Illustration used with permission of Texas Instruments www.ti.com.

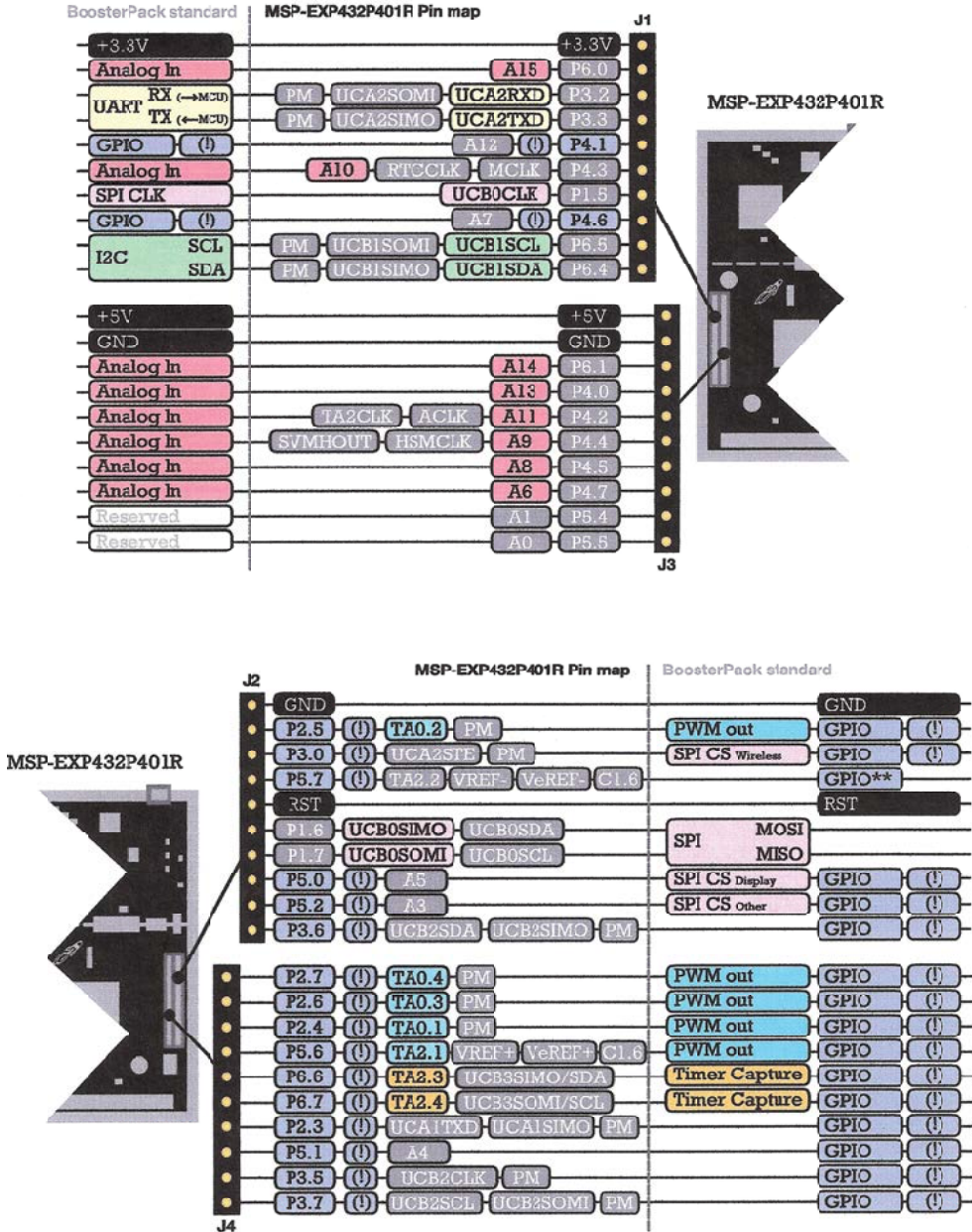


Figure 1.6: MSP432 BoosterPack standard interface [SLAU597A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

- ethernet development tools,
- capacitive touch development kits,
- WiFi development kits,
- temperature sensing kits,
- a multi-sensor platform, and a
- multifunction educational development kit, the Educational BoosterPack MK-II, containing multiple sensors, an LCD display, and output drivers.

Additional details about specific BoosterPacks will be provided in the book. It is important to note new BoosterPacks are under development. Also, you may develop your own BoosterPack. For more information on the LaunchPad and BoosterPack ecosystem from TI, go to ti.com/launchpad.

1.7 SOFTWARE DEVELOPMENT TOOLS

There are many software tools and resources available to support the MSP432 line of microcontrollers from Texas Instruments and third party producers. Here is a partial listing (www.ti.com).

- **Code Composer Studio:** Throughout the book, we use Texas Instruments Code Composer Studio (CCS) Integrated Development Environment (IDE). CCS is used to develop code for all of TI's digital processors including digital signal processors (DSPs), microcontrollers, and application processors. In addition to code development, CCS may be used for debugging and simulation (www.TI.com).
- **CCS Cloud:** In addition to CCS Desktop, CCS Cloud is a development environment for MSP432 that can be accessible from a web browser. While CCS Cloud only provides a subset of programming or debugging capabilities compared to CCS, it can be a convenient platform to quickly develop and prototype an MSP432 application.
- **Other IDEs:** In addition to CCS, the MSP432 may also be programmed using the Keil and IAR Systems IDEs. Throughout the book, we use the CCS IDE.
- **TI Resource Explorer:** The TI Resource Explorer is a “one-stop” location for documentation, code examples, libraries, and data sheets. It may be accessed from within CCS.
- **Energia:** Energia is a user-friendly development environment to allow for quick development of software applications. Similar to the Arduino sketchbook approach, Energia allows MSP microcontroller access to budding engineers and scientists and those without an extensive programming background. Energia allows several activities to be executed (run) in parallel.

- **Driver Library:** Driver Library is an application programming interface (API) library providing rapid software development and prototyping. It encapsulates many of the lower level details in easy-to-use functions. The MSP432 Driver Library provides application functions for virtually every system aboard the MSP432. Also, the source code for each function is available to the user.
- **MSPWare:** MSPWare is a complete collection of development and learning resources for MSP microcontrollers. It is the one-stop shop for all MSP developers to gain access to any design resources they might need during the entire learning or design cycle of their MSP application. MSPWare includes documentation such as device technical reference manuals and datasheets, code examples, libraries of various abstraction layers such as the peripheral Driver Library, USB, real-time operating systems, the Graphical Library, Application Notes, and practical training material on various aspects of the MSP microcontrollers.

1.8 LABORATORY EXERCISE: GETTING ACQUAINTED WITH HARDWARE AND SOFTWARE DEVELOPMENT TOOLS

Introduction. In this first laboratory exercise, we get acquainted with the Texas Instruments MSP-EXP432P401R LaunchPad and Code Composer Studio.

Procedure 1: Install Code Composer Studio. In this first procedure we install Code Composer Studio (CCS) version 6.1 or higher. This procedure was adapted from “Code Composer Studio 6.1 for MSP432” [SLAU575B, 2015].

1. Download CCS from www.ti.com/tool/ccstudio.
2. Check for software updates.
3. Start up CCS. From within CCS install MSPWare and Energia.

Procedure 2: Out-of-box Demo In this procedure we complete the “Out-of-box Demo.” This procedure has been adapted from “Meet the MSP432P401R LaunchPad Development Kit” [SLAU596, 2015]. In the procedure the Red-Green-Blue (RGB) light emitting diode (LED) aboard the MSP-EXP432P401R is varied using a user-friendly graphical user interface (GUI) run on the host PC. The GUI is shown in Figure 1.7.

1. Download software from www.ti.com/beginMSP432launchpad.
2. Connect the LaunchPad to the host PC using the USB cable. With the USB cable connected, the green LED should illuminate and the RGB LED (LED2) should flash during startup.

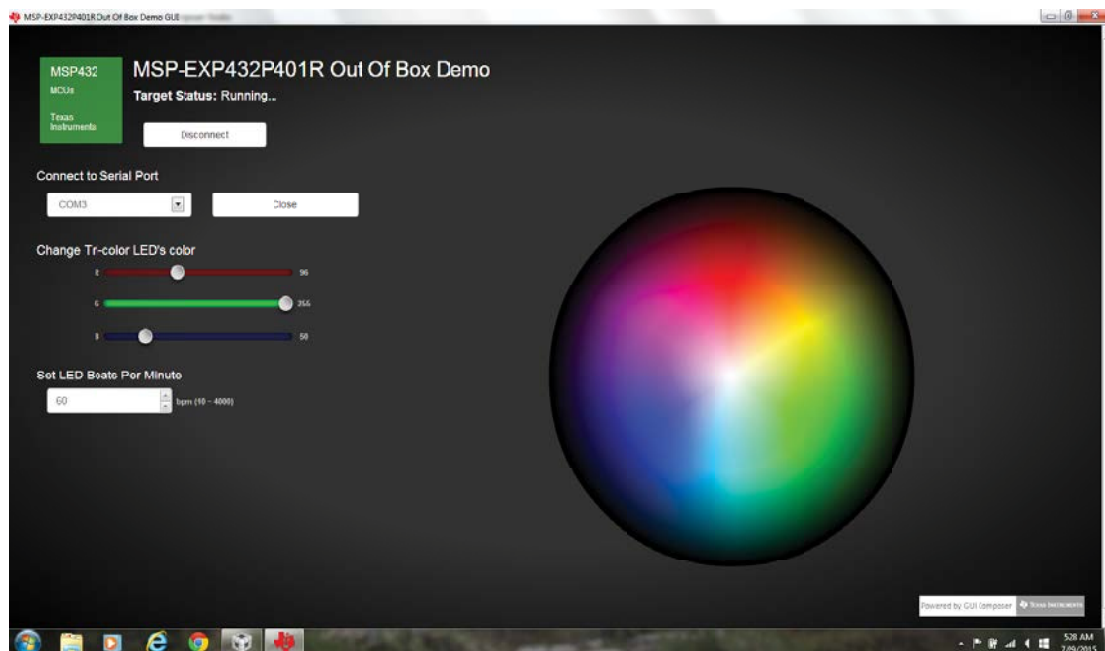


Figure 1.7: RGB LED interface. Illustration used with permission of Texas Instruments www.ti.com.

3. Open the GUI at MSPWare -> Development Tools -> MSP-EXP432P401R -> Examples -> Out of Box Experience.
4. From the GUI change the color of the LED and the speed of the LED flash.

Procedure 3: Blink LED In this procedure we blink LED1 on the MSP-EXP432P401R using the code segment below. The code segment is accessible from within CCS. The code does the following.

- Turns off the MSP432P401R watchdog timer.
- Configures the P1.0 port pin of the MSP432P401R as an output pin [SLAU597A, 2015].
- Enters a while loop to toggle LED1 connected to the P1.0 pin.

```
//*****
// MSP432 main.c template - P1.0 port toggle
//
//Copyright Texas Instruments [www.ti.com]
//*****
```

```

#include "msp.h"

void main(void)
{
    volatile uint32_t i;

    WDT_A->rCTL.r = WDTPW | WDTLHOLD;    // Stop watchdog timer

    // The following code toggles P1.0 port
    DIO->rPADIR.b.bP1DIR |= BIT0;       // Configure P1.0 as output

    while(1)
    {
        DIO->rPAOUT.b.bP1OUT ^= BIT0;   // Toggle P1.0
        for(i=10000; i>0; i--);        // Delay
    }
}
//*****

```

To execute the code snippet, perform the following steps to generate a CCS project. The project contains source, include, and configuration files for a specific application [SLAU575B, 2015].

1. Start CCS.
2. Select a workspace location or select “OK” to use the default location.
3. When CCS launches, select File → New → Project → Code Composer Studio → CCS Project
4. Select MSP432 as the target family and MSP432P401R as the device.
5. Select the XDS110 as the debug probe. Recall this debugger is part of the MSP-EXP432P401R LaunchPad.
6. Select a new project name for your application.
7. Select “Blink the LED” from the “Project templates and examples” list.
8. Click “Finish” to complete your project.
9. To launch the debug session, select Run → Debug. The project will then be built.
10. To run the program, select Run → Go Main

11. LED1 will be blinking!

1.9 SUMMARY

In this chapter, we introduced microcontrollers and an overview of related technologies. We began with a brief review of computer development leading up to the release of microcontrollers, reviewed microcontroller related terminology and provided an overview of systems associated with the MPS432P401R microcontroller. The remainder of the chapter was spent getting better acquainted with the MPS432P401R microcontroller and its associated support hardware and software.

1.10 REFERENCES AND FURTHER READING

Barrett, S. and Pack, D. *Microcontroller Fundamentals for Engineers and Scientists*, San Rafael, CA, Morgan & Claypool Publishers, 2006. DOI: [10.2200/s00025ed1v01y200605dcs001](https://doi.org/10.2200/s00025ed1v01y200605dcs001). 6

Barrett, S. and Pack, D. *Microcontroller Programming and Interfacing, Texas Instruments MSP430*, San Rafael, CA, Morgan & Claypool Publishers, 2011. DOI: [10.2200/s00340ed1v01y201105dcs033](https://doi.org/10.2200/s00340ed1v01y201105dcs033).

Bartee, T. *Digital Computer Fundamentals*, 3rd ed., New York, McGraw-Hill, 1972. 2

Boone, G. *Computing System CPU*. United States Patent 3,757,306 filed August 31, 1971, and issued September 4, 1973. 3

Boone, G. *Variable Function Programmable Calculator*. United States Patent 4,074,351 filed February 24, 1977 and issued February 15, 1978. 3

Dale, N. and Lilly, S.C. *Pascal Plus Data Structures*, 4th ed. Englewood Cliffs, NJ, Jones and Bartlett, 1995.

Fowler, M. and Scott, K. *UML Distilled—A Brief Guide to the Standard Object Modeling Language*, 2nd ed., Boston, Addison-Wesley, 2000.

MCS 85 User's Manual, Intel Corporation: 1977. DOI: [10.1057/9781137002419.0006](https://doi.org/10.1057/9781137002419.0006). 3

Nobelprize.org The Official Web Site of the Nobel Prize. <http://www.nobelprize.org>. 2

Osborne, A. *An Introduction to Microcomputers Volume 1 Basic Concepts*, 2nd ed., Berkeley, Osborne/McGraw-Hill, 1980. 2, 3

Patterson, D. and Hennessy, J. *Computer Organization and Design the Hardware/Software Interface*, San Francisco, Morgan Kaufman, 1994. 3

Texas Instruments Code Composer Studio 6.1 for MSP432 (SLAU575B), Texas Instruments, 2015. 16, 18

Texas Instruments Designing an Ultra-Low-Power (ULP) Application With MSP432 Microcontrollers (SLAA668), Texas Instruments, 2015. 8

Texas Instruments Designing RTOS Power Management Emerges as a Key for MCU-based IoT Nodes (SPRY282), Texas Instruments, 2015. 8

Texas Instruments MSP432P401R LaunchPad Development Kit (MSP-EXP432P401R) (SLAU597A), Texas Instruments, 2015. 12, 13, 14, 17

Texas Instruments Meet the MSP432P401R LaunchPad Development Kit (SLAU596), Texas Instruments, 2015. 16

Texas Instruments MSP432P401x Mixed-Signal Microcontrollers (SLAS826A), Texas Instruments, 2015. 5, 6, 9

1.11 CHAPTER PROBLEMS

Fundamental

1. Define the terms microprocessor, microcontroller, and microcomputer. Provide an example of each.
2. What were the catalysts that led to the multiple generations of computer processors?
3. What are the five main components of a computer architecture? Briefly define each.
4. Distinguish between RAM, ROM, and EEPROM memory. Provide an example application of how each can be employed within a microcontroller.
5. What is RISC architecture? What is its fundamental premise?
6. What is an API? Why is it important?
7. What is the function of an emulator?
8. Why is a floating point unit (FPU) an important onboard feature of a microcontroller?

Advanced

1. List the key features of the MSP432 families of microcontrollers.
2. What is the importance of integrated digital and analog hardware within a single microcontroller?

3. What is the advantage of a 32-bit microcontroller vs. a 16-bit microcontroller?
4. Why are ULP features important?
5. What is DMA? Why is it an important feature on a microcontroller?

Challenging

1. Write one-page paper on a specific generation of computers.
2. Research the difference between CISC and RISC computer architectures. Provide the main features of each approach. Which approach is better suited for microcontroller applications?
3. Research IrDA infrared communication standards. Write one-page paper on the topic.
4. Research the I2C standard and write one-page paper on the topic.
5. Research the CRC-CCITT standard used to calculate a checksum. Write one-page paper on the topic.

CHAPTER 2

A Brief Introduction to Programming

Objectives: After reading this chapter, the reader should be able to:

- successfully download and execute a program using Energia;
- describe the key features of the Energia Integrated Development Environment;
- write programs using Energia for the MSP-EXP432P401R LaunchPad;
- describe how to launch multiple programs simultaneously with Energia multitasking features;
- list the programming support information available at the Energia home page (energia.nu);
- describe key components of a C program;
- specify the size of different variables within the C programming language;
- define the purpose of the main program;
- explain the importance of using functions within a program;
- write functions that pass parameters and return variables;
- describe the function of a header file;
- discuss different programming constructs used for program control and decision processing; and
- write programs in C for execution on the MSP-EXP432P401R LaunchPad.

2.1 OVERVIEW

The goal of this chapter is to provide a tutorial on how to begin programming on the MSP432 microcontroller.¹ We begin with an introduction to programming using the Energia Integrated

¹This chapter was adapted with permission from S. Barret, *Arduino Microcontroller Processing for Everyone*, 3rd ed., San Rafael, CA, Morgan & Claypool Publishers, 2013.

Development Environment (IDE), followed by an introduction to programming in C. Throughout the chapter, we provide examples and pointers to a number of excellent references.

2.2 ENERGIA

Energia is an open-source IDE modeled after the Arduino Sketchbook concept. It allows for rapid prototyping of a wide range of Texas Instruments microcontroller products. We use it to rapidly prototype programs and embedded systems using the MSP-EXP432P401R LaunchPad. Energia MT (multitasking) allows multiple tasks to be run seemingly simultaneously (energia.nu). In space lore, the Energia was a Soviet heavy lift rocket. Similarly, the Energia IDE performs heavy lifting when learning software programming for the first time.

2.3 ENERGIA QUICKSTART

To quickly get up and operating with Energia, follow these steps (energia.nu).

- Download and install Energia MT, version 16 or newer from the energia.nu website to the host computer. It is available for different operating systems including: Windows, Mac OS X, and Linux.
- If using the Windows operating system, download and install drivers for the LaunchPad from the Energia website. The drivers allow communication between the LaunchPad and the host computer's serial com ports.
- Launch Energia on the host computer by going to the Energia folder and clicking on the Energia icon. The icon is a red ball with a rocket silhouette.
- Connect the LaunchPad to the host computer via the USB cable provided with the LaunchPad.
- With Energia launched, go to Tools -> Board -> and select the LaunchPad w/msp432 EMT (48MHz).
- Check the comm port setting using Tools -> Serial Port.
- To load the first example use File -> Examples -> Basics -> Blink.
- To compile, upload, and run the program, use the Upload icon (right facing arrow).
- The red LED on the LaunchPad will blink!

With our first program launched, let's take a closer look at the Energia IDE.

2.4 ENERGIA DEVELOPMENT ENVIRONMENT

In this section, we provide an overview of the Energia IDE. We begin with some background information about the IDE and then review its user-friendly features. We then introduce the sketchbook concept and provide a brief overview of the built-in software features within the IDE. Our goal is to provide readers with a brief introduction to Energia features. All Energia-related features are well-documented on the Energia homepage (energia.nu). We will not duplicate this excellent source of material; but merely provide a brief introduction with pointers to advanced features.

2.4.1 ENERGIA IDE OVERVIEW

At its most fundamental level, the Energia IDE is a user-friendly interface to allow one to quickly write, load, and execute code on a microcontroller. A barebones program needs only a `setup()` and a `loop()` function. The Energia IDE adds the other required pieces such as header files and the main program constructs (energia.nu).

The Energia IDE is illustrated in Figure 2.1. The IDE contains a text editor, a message area for displaying status, a text console, a tool bar of common functions, and an extensive menu system. The IDE also provides a user-friendly interface to the LaunchPad which allows for the quick compiling and uploading of code.

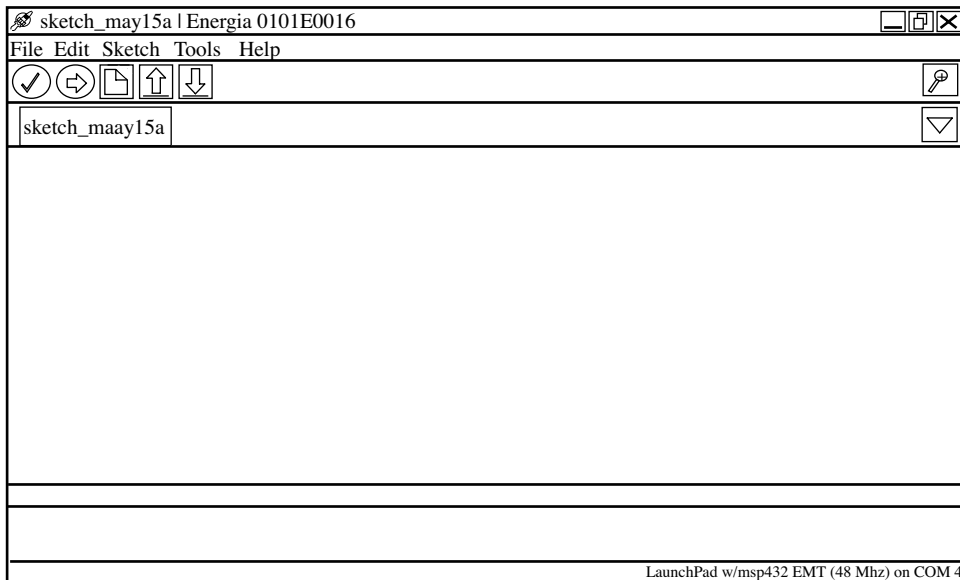


Figure 2.1: Energia IDE (energia.nu).

A close-up of the Energia toolbar is provided in Figure 2.2. The toolbar provides single button access to the more commonly used menu features. Most of the features are self-explanatory. The “Upload” button compiles the program and uploads it to the LaunchPad. The “Serial Monitor” button opens a serial monitor to allow text data to be sent to and received from the LaunchPad. The tab feature allows multiple tabs to be opened simultaneously for program multitasking.

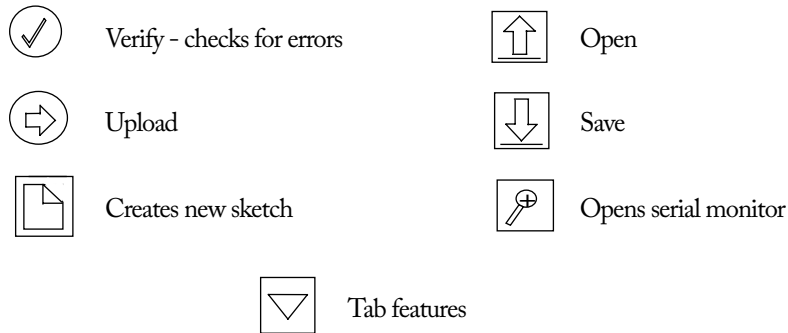


Figure 2.2: Energia IDE buttons.

2.4.2 SKETCHBOOK CONCEPT

In keeping with a hardware and software platform for students of the arts, the Energia environment employs the concept of a sketchbook. Artists maintain their works in progress in a sketchbook. Similarly, we maintain our programs within a sketchbook in the Energia environment. Furthermore, we refer to individual programs as sketches. An individual sketch within the sketchbook may be accessed via the Sketchbook entry under the file tab.

2.4.3 ENERGIA SOFTWARE, LIBRARIES, AND LANGUAGE REFERENCES

The Energia IDE has a number of built-in features. Some of the features may be directly accessed via the Energia IDE drop down toolbar illustrated in Figure 2.1. Provided in Figure 2.3 is a handy reference to show all of the available features. The toolbar provides a wide variety of features to compose, compile, load, and execute a sketch.

Aside from the toolbar accessible features, the Energia IDE contains a number of built-in functions that allow the user to quickly construct a sketch. These built-in functions are summarized in Figure 2.4. Complete documentation for these built-in function is available at the Energia homepage (energia.nu). This documentation is easily accessible via the Help tab on the Energia IDE toolbar. We refer to these features at appropriate places throughout the remainder of the book and provide additional background information as needed.

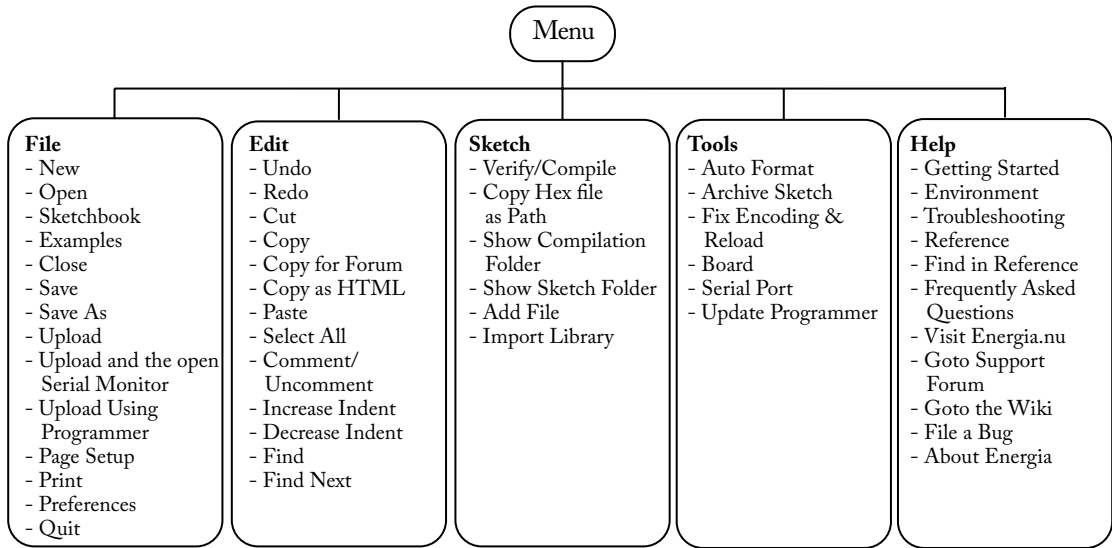


Figure 2.3: Energia IDE menu (energia.nu).

2.5 ENERGIA PIN ASSIGNMENTS

Hardware features onboard the LaunchPad (LEDs, switches, etc.) are accessed via Energia using pin numbers. Pin numbers range from 1–72 as shown in Figure 2.5. This information is also contained in a header file within Energia. It is provided here for easy reference (energia.nu). We refer to this frequently in the upcoming examples to obtain pin number access for MSP-EXP432P401R LaunchPad features and systems.

```

//*****
//Copyright (c) 2015, Texas Instruments Incorporated
//All rights reserved.
//*****

#ifndef Pins_Energia_h
#define Pins_Energia_h

#include <stdbool.h>
#include <stdint.h>

static const uint8_t RED_LED = 75;    //RGB LED - red component
static const uint8_t GREEN_LED = 76; //RGB LED - green component

```

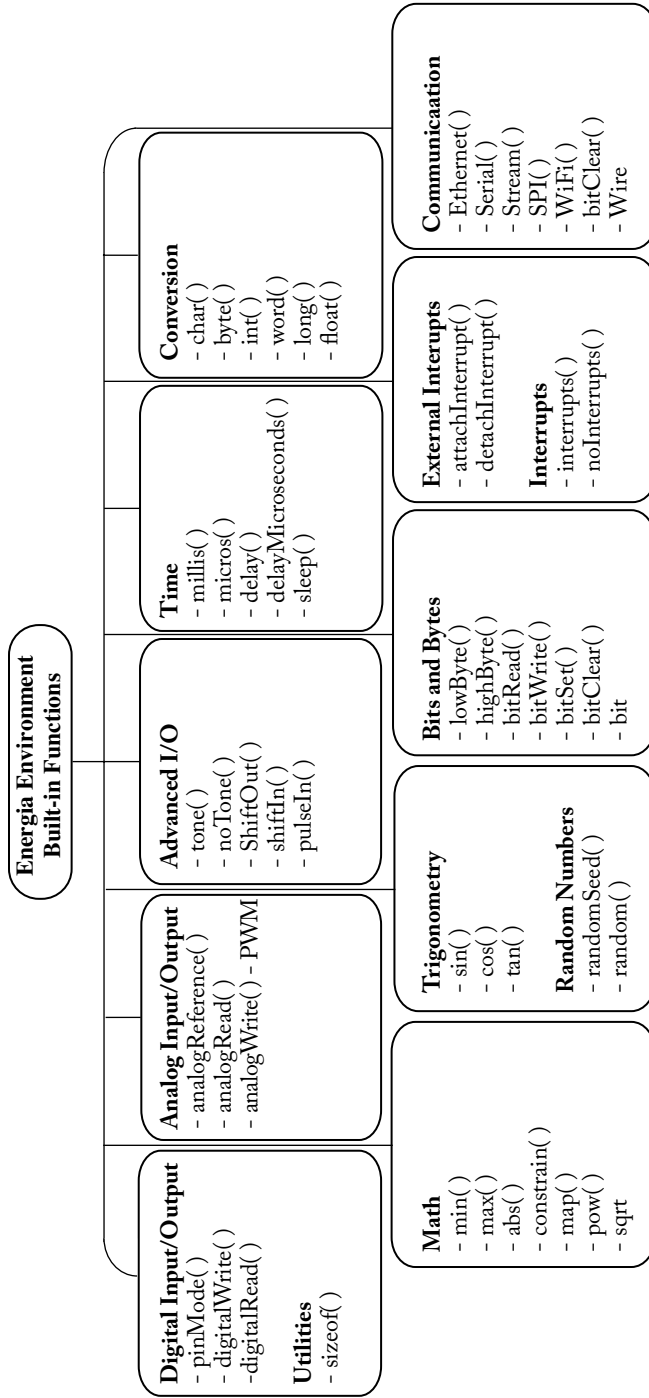



Figure 2.4: Energia IDE built-in features (energia.nu).

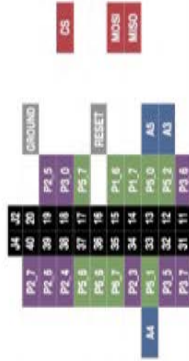
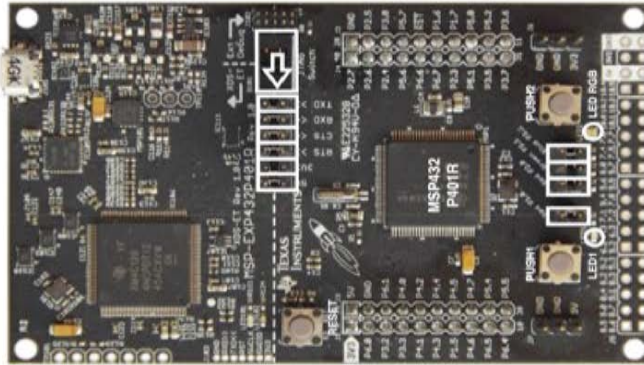


LaunchPad with MSP432

Revision 3.2

SRAM 64 KB
Flash 256 KB

Serial 115200 bps
ADC 14 bits
Use pins numbers only!



TI MSP432P401R LaunchPad with Energia
www.ti.com
© 2015-2016
ENERGIA

Figure 2.5: MSP432P401R pin map (energia.nu). Illustration used with permission of Texas Instruments www.ti.com.

```

static const uint8_t BLUE_LED = 77;    //RGB LED - blue component
static const uint8_t YELLOW_LED = 78; //Mapped to the other RED LED
static const uint8_t PUSH1 = 73;      //Switch 1
static const uint8_t PUSH2 = 74;      //Switch 2

static const uint8_t A0 = 30;          //analog-to-digital converter
static const uint8_t A1 = 29;          //channels 0 to 23
//static const uint8_t A2 = n/a;
static const uint8_t A3 = 12;
static const uint8_t A4 = 33;
static const uint8_t A5 = 13;
static const uint8_t A6 = 28;
static const uint8_t A7 = 8;
static const uint8_t A8 = 27;
static const uint8_t A9 = 27;
static const uint8_t A10 = 6;
static const uint8_t A11 = 25;
static const uint8_t A12 = 5;
static const uint8_t A13 = 24;
static const uint8_t A14 = 23;
static const uint8_t A15 = 2;
static const uint8_t A16 = 59;
static const uint8_t A17 = 42;
static const uint8_t A18 = 58;
static const uint8_t A19 = 57;
static const uint8_t A20 = 41;
static const uint8_t A21 = 43;
static const uint8_t A22 = 69;
static const uint8_t A23 = 44;

#endif
//*****

```

2.6 WRITING AN ENERGIA SKETCH

The basic format of the Energia sketch consists of a “setup” and a “loop” function. The setup function is executed once at the beginning of the program. It is used to configure pins, declare variables and constants, etc. The loop function will execute sequentially step-by-step. When the end of the loop function is reached, it will automatically return to the first step of the loop function and execute the function again. This goes on continuously until the program is stopped.

```

//*****

void setup()
{
  //place setup code here
}

void loop()
{
  //main code steps are provided here
  :
  :
}

```

```

//*****

```

Example 1: Blink. Let's examine the sketch used to blink the LED (energia.nu).

```

//*****
//Blink: Turns on an LED on for one second, then off for
//one second, repeatedly.
//
//Change the LED color using the #define statement to select
//another LED number.
//
// This example code is in the public domain.
//*****

//the red LED is connected to Energia pin 75
#define LED 75

//the setup routine runs once when you press reset:
void setup()
{
  //Configure pin 75 as a digital output pin
  pinMode(LED, OUTPUT);
}

//the loop routine runs continuously
void loop()

```

```

{
digitalWrite(LED, HIGH); //turn the LED on
delay(1000); //wait a second
digitalWrite(LED, LOW); //turn the LED off
delay(1000); //wait for a second
}

//*****

```

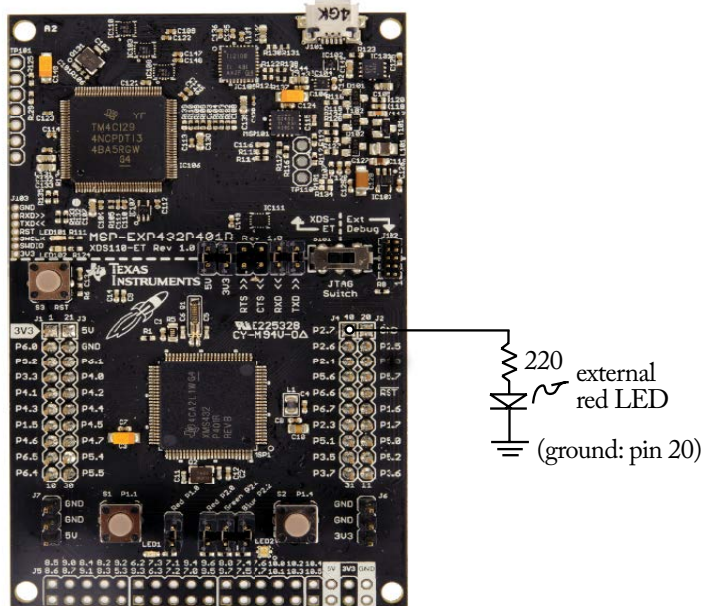
In the first line the #define statement links the designator “LED” to pin 75 on the LaunchPad. In the setup function, LED is designated as an output pin. Recall the setup function is only executed once. The program then enters the loop function that is executed sequentially step-by-step and continuously repeated. In this example, the LED is first set to logic high to illuminate the LED onboard the LaunchPad. A 1000 ms delay then occurs. The LED is then set low. A 1000 ms delay then occurs. The sequence then repeats.

Aside from the Blink example, there are also a number of program examples available to allow a user to quickly construct a sketch. They are useful to understand the interaction between the Energia IDE and the LaunchPad. They may also be used as a starting point to write new applications. The program examples are available via the File → Examples tab within Energia. The examples fall within these categories:

1. Basics
2. Digital
3. Analog
4. Communication
5. Control
6. Strings
7. Sensors
8. Display
9. Educational BP Mk II—a multifunction educational development kit containing multiple sensors, an LCD display, and output drivers
10. MultiTasking—allows multiple tasks to be executed simultaneously

We now examine several more Energia based examples.

Example 2: External LED. In this example we connect an external LED to LaunchPad pin 40. The onboard green LED (pin 76) will blink alternately with the external LED. The external LED is connected to the LaunchPad, as shown in Figure 2.6.



(a) schematic.

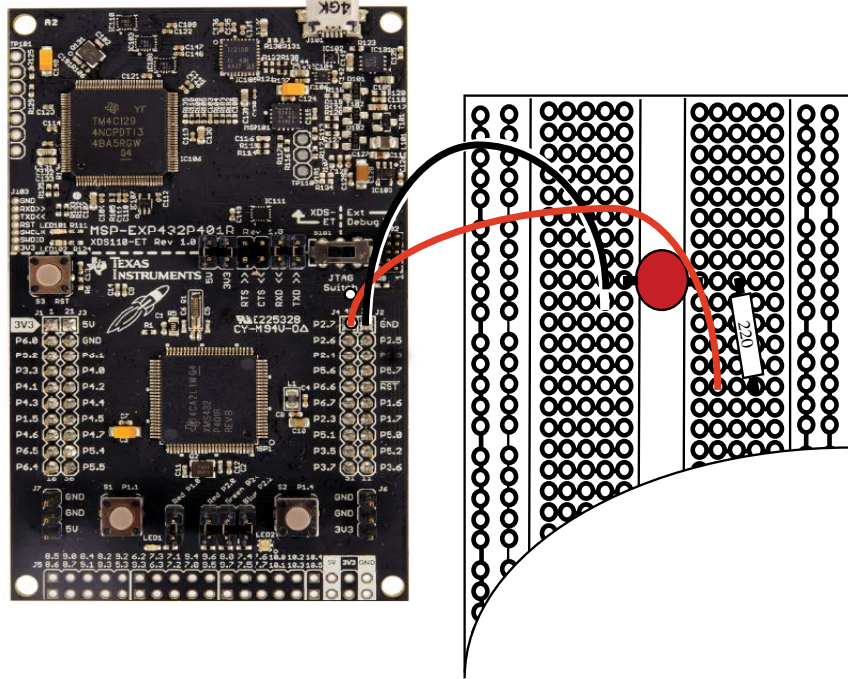
Figure 2.6: LaunchPad with an external LED. (*Continues.*)

```
//*****

#define int_LED 76
#define ext_LED 40

void setup()
{
  pinMode(int_LED, OUTPUT);
  pinMode(ext_LED, OUTPUT);
}

void loop()
{
  digitalWrite(int_LED, HIGH);
  digitalWrite(ext_LED, LOW);
  delay(500);          //delay specified in ms
}
```



(b) circuit layout.

Figure 2.6: (Continued.) LaunchPad with an external LED.

```
digitalWrite(int_LED, LOW);
digitalWrite(ext_LED, HIGH);
delay(500);
}
```

//*****

Example 3: External LED and switch. In this example we connect an external LED to LaunchPad pin 40 and an external switch attached to pin 31. The onboard green LED will blink alternately with the external LED when the switch is depressed. The external LED and switch is connected to the LaunchPad, as shown in Figure 2.7.

//*****

```
#define int_LED 76
#define ext_LED 40
```

```

#define ext_sw 31

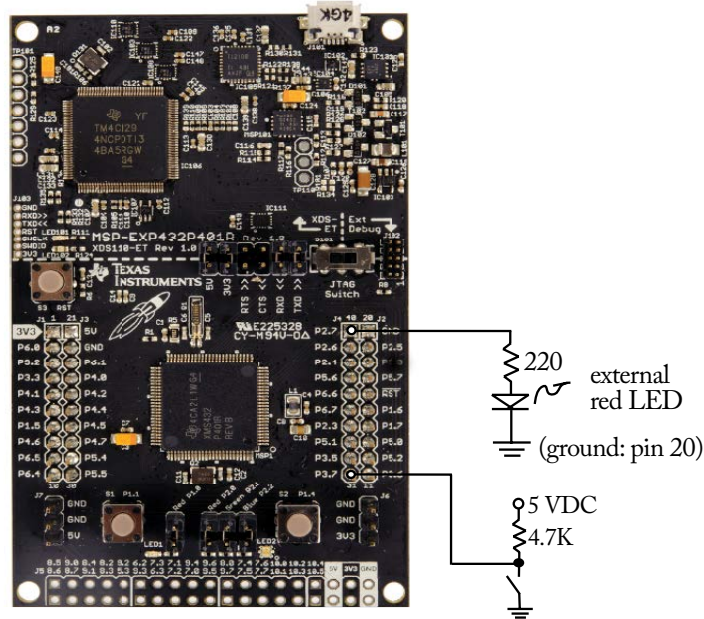
int switch_value;

void setup()
{
  pinMode(int_LED, OUTPUT);
  pinMode(ext_LED, OUTPUT);
  pinMode(ext_sw, INPUT);
}

void loop()
{
  switch_value = digitalRead(ext_sw);
  if(switch_value == LOW)
  {
    digitalWrite(int_LED, HIGH);
    digitalWrite(ext_LED, LOW);
    delay(50);
    digitalWrite(int_LED, LOW);
    digitalWrite(ext_LED, HIGH);
    delay(50);
  }
  else
  {
    digitalWrite(int_LED, LOW);
    digitalWrite(ext_LED, LOW);
  }
}
//*****

```

Example 4: MultiTasking (MT). Energia MT provides for multitasking on the MSP432-EXPP401R LaunchPad. This feature allows for multiple tasks to be executed simultaneously. To configure an application for multitasking, independent tasks are provided unique “setup” and “loop” names. The tasks may be placed in the same Energia sketch or may be placed in different tabs. The example below was modified from the example found within Energia (File -> Examples -> MultiTasking -> MultiBlink). Note three independent tasks have been placed in the same sketchbook. Each task controls the blink rate on a different LED. In more advanced applications, information may be exchanged between the tasks using global variables (energia.nu).



(a) schematic.

Figure 2.7: LaunchPad with an external LED and switch. (*Continues.*)

To place the tasks under independent tabs, launch a new tab for each task using the new tab icon (downward arrow) on the right side of the Energia IDE.

```
//*****
```

```
#define LED BLUE_LED
```

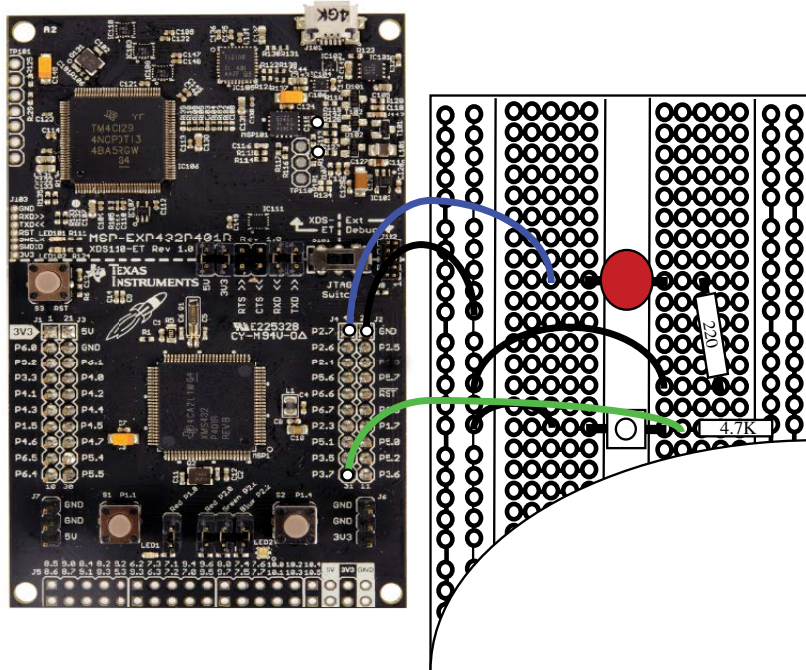
```
void setupBlueLed()
```

```
{
  //initialize the digital pin as an output.
  pinMode(LED, OUTPUT);
}
```

```
//the loop routine runs over and over again forever as a task.
```

```
void loopBlueLed()
```

```
{
  digitalWrite(LED, HIGH); //turn the LED on (HIGH is the voltage level)
```



(b) circuit layout.

Figure 2.7: (Continued.) LaunchPad with an external LED and switch.

```

delay(100);           //wait for 100 ms
digitalWrite(LED, LOW); //turn the LED off by making the voltage LOW
delay(100);           //wait for 100 ms
}

//*****
#define LED GREEN_LED

void setupGreenLed()
{
  //initialize the digital pin as an output.
  pinMode(LED, OUTPUT);
}

//the loop routine runs over and over again forever as a task.

```

```

void loopGreenLed()
{
  digitalWrite(LED, HIGH); //turn the LED on (HIGH is the voltage level)
  delay(500);              //wait for half a second
  digitalWrite(LED, LOW);  //turn the LED off by making the voltage LOW
  delay(500);              //wait for half a second
}

//*****
#define LED RED_LED

void setupRedLed()
{
  // initialize the digital pin as an output.
  pinMode(LED, OUTPUT);
}

//the loop routine runs over and over again forever as a task.
void loopRedLed()
{
  digitalWrite(LED, HIGH); //turn the LED on (HIGH is the voltage level)
  delay(1000);             //wait for a second
  digitalWrite(LED, LOW);  //turn the LED off by making the voltage LOW
  delay(1000);             //wait for a second
}

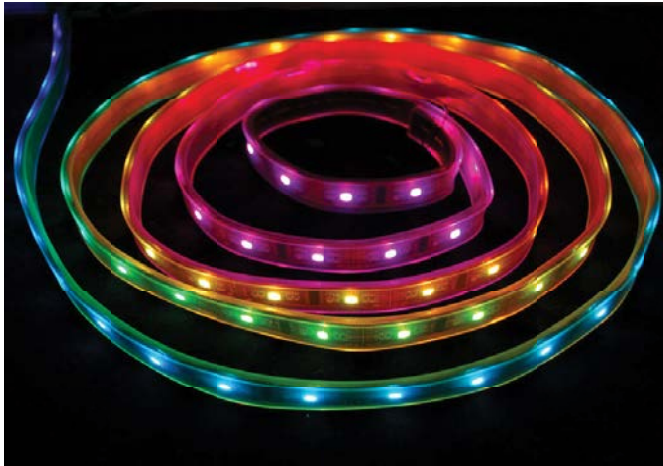
//*****

```

Example 5: LED strip. LED strips may be used for motivational (fun) optical displays, games, or for instrumentation-based applications. In this example we control an LPD8806-based LED strip using Energia. We use a 1-m, 32-RGB LED strip available from Adafruit (#306) for approximately \$30 USD (www.adafruit.com).

The red, blue, and green component of each RGB LED is independently set using an eight-bit code. The most significant bit (MSB) is logic one followed by seven bits to set the LED intensity (0–127). The component values are sequentially shifted out of the MSP432–EXP432P401R LaunchPad using the Serial Peripheral Interface (SPI) features. The first component value shifted out corresponds to the LED nearest the microcontroller. Each shifted component value is latched to the corresponding R, G, and B component of the LED. As a new component value is received, the previous value is latched and held constant. An extra byte is required to latch the final parameter value. A zero byte (00)₁₆ is used to complete the data sequence and reset back to the first LED (www.adafruit.com).

Only four connections are required between the MSP432-EXP432P401R LaunchPad and the LED strip as shown in Figure 2.8. The connections are color coded: red-power, black-ground, yellow-data, and green-clock. It is important to note the LED strip requires a supply of 5 VDC and a current rating of 2 amps per meter of LED strip. In this example we use the Adafruit #276 5V 2A (2000mA) switching power supply (www.adafruit.com).



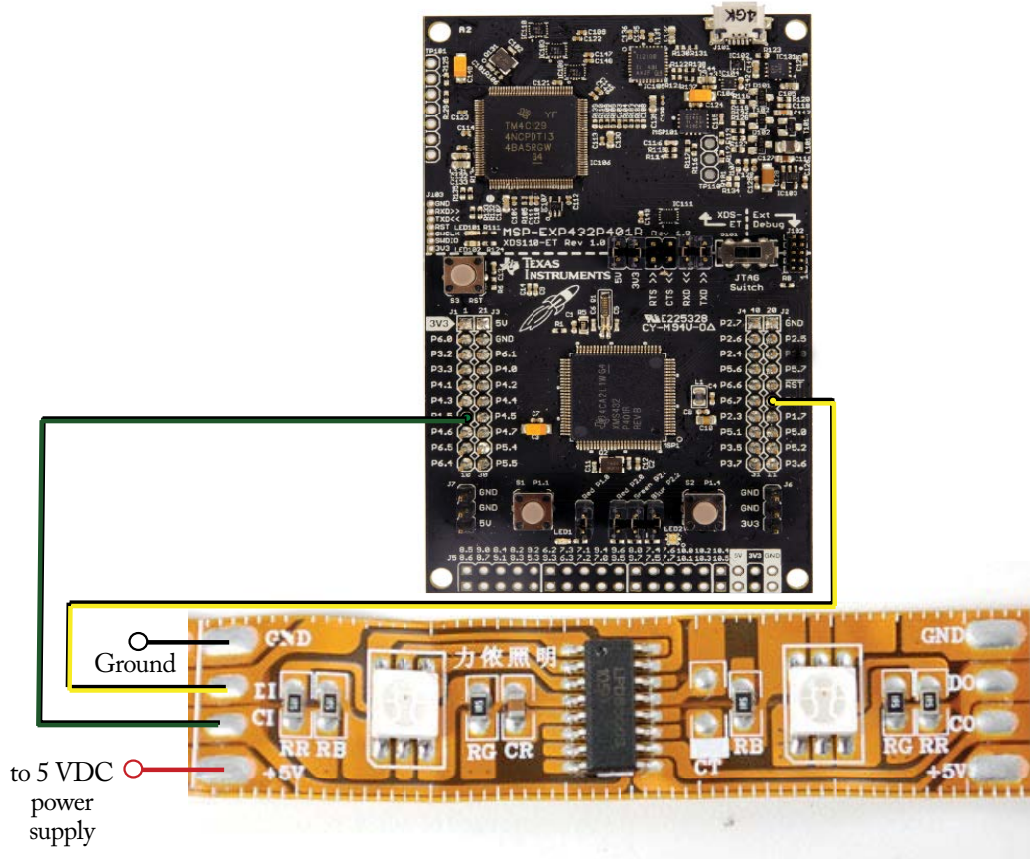
(a) LED strip by the meter (www.adafruit.com).

Figure 2.8: LaunchPad controlling LED strip (www.adafruit.com). (*Continues.*)

In this example each RGB component is sent separately to the strip. The example illustrates how each variable in the program controls a specific aspect of the LED strip. Here are some important implementation notes.

- SPI must be configured for most significant bit (MSB) first.
- LED brightness is seven bits. Most significant bit (MSB) must be set to logic one.
- Each LED requires a separate R-G-B intensity component. The order of data is G-R-B.
- After sending data for all LEDs. A byte of (0x00) must be sent to return strip to first LED.
- Data stream for each LED is: 1-G6-G5-G4-G3-G2-G1-G0-1-R6-R5-R4-R3-R2-R1-R0-1-B6-B5-B4-B3-B2-B1-B0

```
//*****
//RGB_led_strip_tutorial: illustrates different variables within
//RGB LED strip
```



(b) MSP432-EXP432P401R to LED strip connection (www.adafruit.com).

Figure 2.8: (Continued.) LaunchPad controlling LED strip (www.adafruit.com).

```
//
//LED strip LDP8806 - available from www.adafruit.com (#306)
//
//Connections:
// - External 5 VDC supply - Adafruit 5 VDC, 2A (#276) - red
// - Ground - black
// - Serial Data In - LaunchPad pin 14 (MOSI pin USI) P1.6 - yellow
// - CLK - LaunchPad pin 7 (SCK pin) P1.5 - green
//
```

```

//Variables:
// - LED_brightness - set intensity from 0 to 127
// - segment_delay - delay between LED RGB segments
// - strip_delay - delay between LED strip update
//
//Notes:
// - SPI must be configured for Most significant bit (MSB) first
// - LED brightness is seven bits.
Most significant bit (MSB)
// must be set to logic one
// - Each LED requires a separate R-G-B intensity component.
The order
// of data is G-R-B.
// - After sending data for all strip LEDs.
A byte of (0x00) must
// be sent to reutrn strip to first LED.
// - Data stream for each LED is:
//1-G6-G5-G4-G3-G2-G1-G0-1-R6-R5-R4-R3-R2-R1-R0-1-B6-B5-B4-B3-B2-B1-B0
//
//This example code is in the public domain.
//*****

#include <SPI.h>

#define LED_strip_latch 0x00

const byte strip_length = 32; //number of RGB LEDs in strip
const byte segment_delay = 100; //delay in milliseconds
const byte strip_delay = 500; //delay in milliseconds
unsigned char LED_brightness; //0 to 127
unsigned char position; //LED position in strip
unsigned char troubleshooting = 0; //allows printouts to serial
//monitor

void setup()
{
  SPI.begin(); //SPI support functions
  SPI.setBitOrder(MSBFIRST); //SPI bit order
  SPI.setDataMode(SPI_MODE3); //SPI mode

```

42 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
SPI.setClockDivider(SPI_CLOCK_DIV32); //SPI data clock rate
Serial.begin(9600);                    //serial comm at 9600 bps
}

void loop()
{
  SPI.transfer(LED_strip_latch);       //reset to first segment
  clear_strip();                       //all strip LEDs to black
  delay(500);

  //increment the green intensity of the strip LEDs
  for(LED_brightness = 0; LED_brightness <= 60;
      LED_brightness = LED_brightness + 10)
  {
    for(position = 0; position < strip_length; position = position + 1)
    {
      SPI.transfer(0x80 | LED_brightness); //Green - MSB 1
      SPI.transfer(0x80 | 0x00);          //Red   - none
      SPI.transfer(0x80 | 0x00);          //Blue  - none

      if(troubleshooting)
      {
        Serial.println(LED_brightness, DEC);
        Serial.println(position, DEC);
      }
      delay(segment_delay);
    }
    SPI.transfer(LED_strip_latch);       //reset to first segment
    delay(strip_delay);
    if(troubleshooting)
    {
      Serial.println(" ");
    }
  }

  clear_strip();                       //all strip LEDs to black
  delay(500);

  //increment the red intensity of the strip LEDs
```

```

for(LED_brightness = 0; LED_brightness <= 60;
  LED_brightness = LED_brightness + 10)
{
for(position = 0; position<strip_length; position = position+1)
  {
  SPI.transfer(0x80 | 0x00);           //Green - none
  SPI.transfer(0x80 | LED_brightness); //Red   - MSB1
  SPI.transfer(0x80 | 0x00);           //Blue  - none

  if(troubleshooting)
    {
    Serial.println(LED_brightness, DEC);
    Serial.println(position, DEC);
    }
  delay(segment_delay);
  }
SPI.transfer(LED_strip_latch);        //reset to first segment
delay(strip_delay);
if(troubleshooting)
  {
  Serial.println(" ");
  }
}

clear_strip();                        //all strip LEDs to black
delay(500);

//increment the blue intensity of the strip LEDs
for(LED_brightness = 0; LED_brightness <= 60;
  LED_brightness = LED_brightness + 10)
{
for(position = 0; position<strip_length; position = position+1)
  {
  SPI.transfer(0x80 | 0x00);           //Green - none
  SPI.transfer(0x80 | 0x00);           //Red   - none
  SPI.transfer(0x80 | LED_brightness); //Blue  - MSB1

  if(troubleshooting)
    {

```


44 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
        Serial.println(LED_brightness, DEC);
        Serial.println(position, DEC);
    }
    delay(segment_delay);
}
SPI.transfer(LED_strip_latch);    //reset to first segment
delay(strip_delay);
if(troubleshooting)
{
    Serial.println(" ");
}
}

clear_strip();                    //all strip LEDs to black
delay(500);
}

//*****

void clear_strip(void)
{
    //clear strip
    for(position = 0; position < strip_length; position = position+1)
    {
        SPI.transfer(0x80 | 0x00);    //Green - none
        SPI.transfer(0x80 | 0x00);    //Red - none
        SPI.transfer(0x80 | 0x00);    //Blue - none

        if(troubleshooting)
        {
            Serial.println(LED_brightness, DEC);
            Serial.println(position, DEC);
        }
    }
    SPI.transfer(LED_strip_latch);    //Latch with zero
    if(troubleshooting)
    {
        Serial.println(" ");
    }
}
```

```

    delay(2000);
}

//clear delay

//*****

```

Example 6: Analog In–Analog Out–Serial Out. This example is modified from the example Analog In–Analog Out–Serial Out provided with Energia. It illustrates several Energia built-in functions.

- **Serial.begin(baud_rate):** Sets baud rate in bits per second to communicate with the host computer.
- **Serial.print(text):** Prints text to Energia serial monitor.
- **AnalogRead(analog_channel):** Reads the analog value at the designated analog channel and returns a value from 0 (0 VDC) to 1023 (3.3 VDC).
- **map(test_value, input_low, input_high, output_low, output_high):** Remaps test_value from a value between input_low and input_high to a corresponding value between output_low and output_high.
- **analogWrite(analogOutPin, outputValue):** Sends an output value from 0–255 to designated analogOutPin.

```

//*****
//Analog input, analog output, serial output - Reads an analog input pin,
//maps the result to a range from 0 to 255 and uses the result to set the
//pulsewidth modulation (PWM) of an output pin.
The PWM value is sent to
//the red LED pin to modulate its intensity.
Also prints the results to
//the serial monitor. Open a serial monitor using the serial monitor
//button in Energia to view the results.
//
//
//The circuit:
// - Potentiometer connected to analog pin 0 (30). The center wiper
//   pin of the potentiometer goes to the analog pin.
The side pins of
//   the potentiometer go to +3.3 VDC and ground.
// - The analog output is designated as the onboard red LED.
//
//Created: Dec 29, 2008

```

46 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
//Modified: Aug 30, 2011
//Author: Tom Igoe
//
//This example code is in the public domain.
//*****

const int analogInPin = 30; //Energia analog input pin A0
const int analogOutPin = 75; //Energia onboard red LED pin

int sensorValue = 0; //value read from potentiometer
int outputValue = 0; //value output to the PWM (red LED)

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop()
{
  //read the analog in value:
  sensorValue = analogRead(analogInPin);

  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);

  // change the analog out value:
  analogWrite(analogOutPin, outputValue);

  // print the results to the serial monitor:
  Serial.print("sensor = ");
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // wait 10 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(10);
}
```

}

```
//*****
```

Example 7: Dagu Magician Autonomous Maze Navigating Robot. In this example, an autonomous, maze navigating robot is equipped with infrared (IR) sensors to detect the presence of maze walls and navigate about the maze. The robot has no prior knowledge about the maze configuration. It uses the IR sensors and an onboard algorithm to determine the robot's next move. The overall goal is to navigate from the starting point of the maze to the end point as quickly as possible without bumping into maze walls, as shown in Figure 2.9. Maze walls are usually painted white to provide a good, light reflective surface, whereas, the maze floor is painted matte black to minimize light reflections.

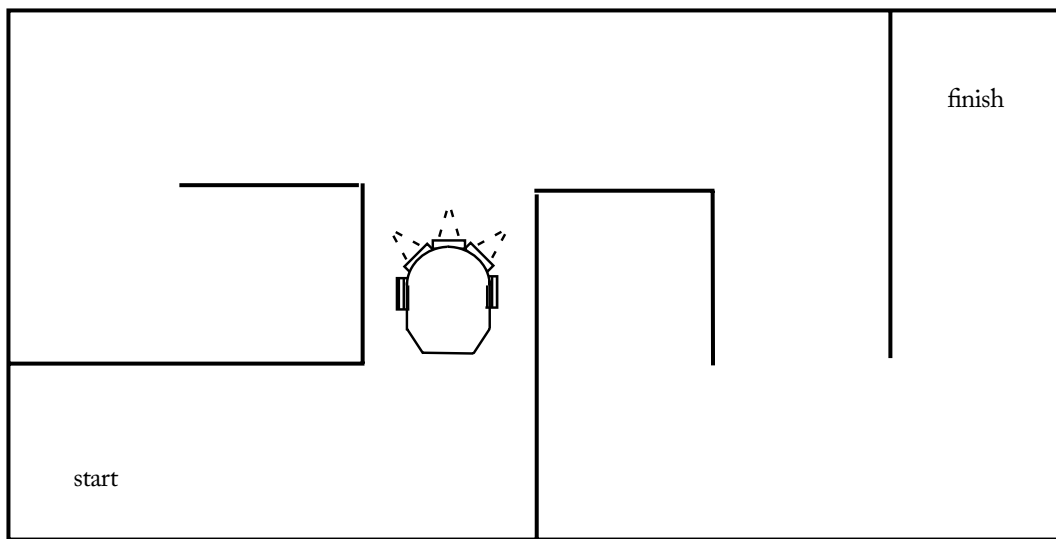


Figure 2.9: Autonomous robot within maze.

Before delving into the design, it would be helpful to review the fundamentals of robot steering and motor control. Figure 2.10 illustrates the fundamental concepts. Robot steering is dependent upon the number of powered wheels and whether the wheels are equipped with unidirectional or bidirectional control. Additional robot steering configurations are possible. An H-bridge is typically required for bidirectional control of a DC motor. We discuss the H-bridge in greater detail in an upcoming chapter.

In this application project, we equip the Dagu Magician robot for control by the LaunchPad as a maze navigating robot; see Figure 2.12. The Magician kit may be purchased from SparkFun Electronics (www.sparkfun.com). The robot is controlled by two 7.2 VDC motors which inde-

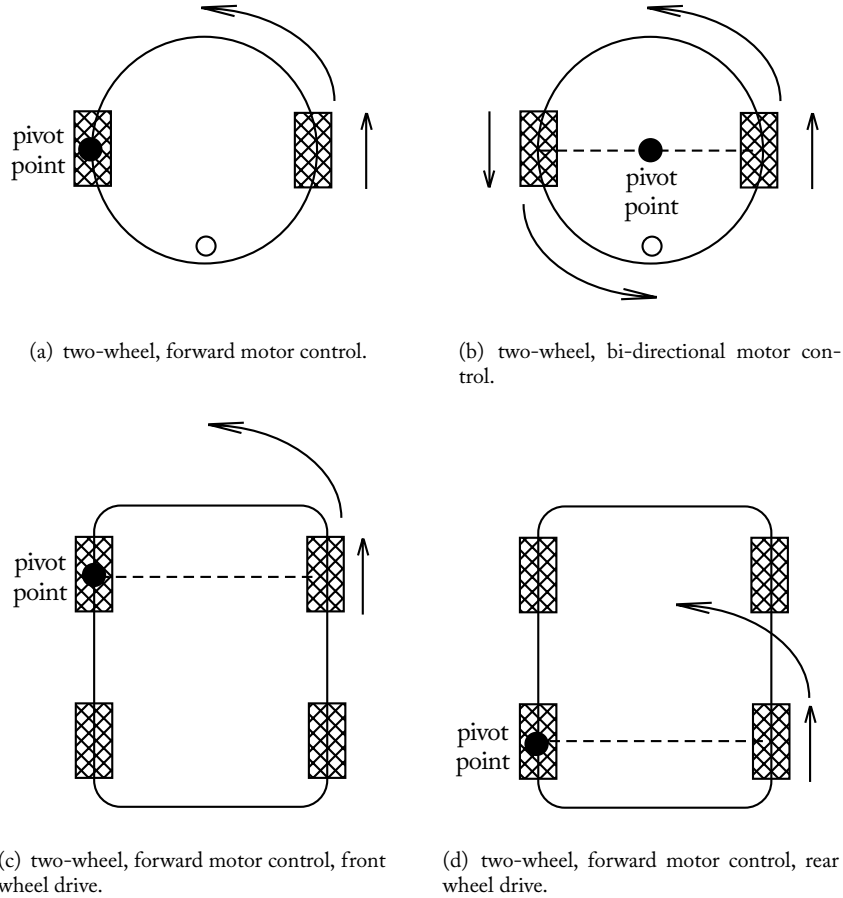


Figure 2.10: Robot control configurations. (*Continues.*)

pendently drive a left and right wheel. A third non-powered drag ball provides tripod stability for the robot.

We equip the Dagu Magician robot platform with three Sharp GP2Y0A21YKOF IR sensors as shown in Figure 2.13. The sensors are available from SparkFun Electronics (www.sparkfun.com). We mount the sensors on a bracket constructed from thin aluminum. Dimensions for the bracket are provided in the figure. Alternatively, the IR sensors may be mounted to the robot platform using “L” brackets available from a local hardware store. The characteristics of the sensor are provided in Figure 2.11. The robot is placed in a maze with reflective walls. The project goal is for the robot to detect wall placement and navigate through the maze. It is important to note the robot does not have any *a priori* information about the maze. The control algorithm for the

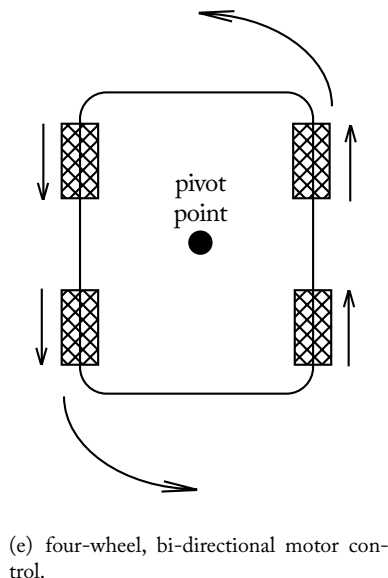


Figure 2.10: (Continued.) Robot control configurations.

robot is hosted on the LaunchPad. The requirements for this project are simple, the robot must autonomously navigate through the maze without touching maze walls.

The circuit diagram for the robot is provided in Figure 2.14. The three IR sensors (left, middle, and right) are mounted on the leading edge of the robot to detect maze walls. The output from the sensors is fed to three ADC channels (analog in 3–5). The robot motors will be driven by PWM channels (PWM: DIGITAL 11 and PWM: DIGITAL 10).

To save on battery expense, a 9 VDC, 2A rated inexpensive, wall-mount power supply is used to provide power to the robot. A power umbilical of flexible, braided wire may be used to link the power supply to the robot while navigating about the maze. The robot motors are rated at 7.2 VDC. Therefore, three 1N4001 diodes are placed in series with the motor to reduce the supply voltage to be approximately 6.9 VDC. The LaunchPad is interfaced to the motors via a Darlington NPN transistor (TIP120) with enough drive capability to handle the maximum current requirements of the motor. A 3.3 VDC voltage regulator is used to supply power to the LaunchPad.

Warning: It is important **not** to have the LaunchPad connected to the host computer via the USB cable and an external 3.3 VDC supply at the same time. It is recommended to download the program to the LaunchPad, disconnect the USB cable, remove the 3.3 VDC header jumper on the Jumper Isolation Block, and then connect the 3.3 VDC external supply to the J6 connector. Alternatively, a double throw double pole (DPDT) switch may be used, as shown in Figure 2.14.

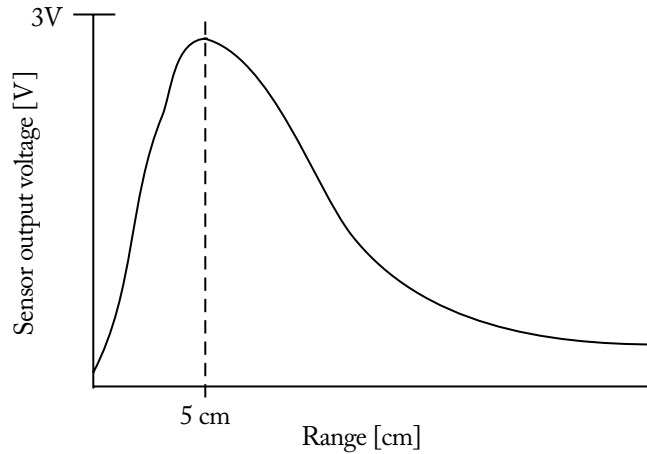


Figure 2.11: Sharp GP2Y0A21YKOF IR sensor profile.

Structure chart: A structure chart is a visual tool used to partition a large project into “doable” smaller parts. It also helps to visualize what systems will be used to control different features of the robot. The arrows within the structure chart indicate the data flow between different portions of the program controlling the robot. The structure chart for the robot project is provided in Figure 2.15. As you can see, the robot has three main systems: the motor control system, the sensor system, and the digital input/output system. These three systems interact with the main control algorithm to allow the robot to autonomously (by itself) navigate through the maze by sensing and avoiding walls.

UML activity diagrams: A Unified Modeling Language (UML) activity diagram, or flow chart, is a tool to help visualize the different steps required for a control algorithm. The UML activity diagram for the robot is provided in Figure 2.16. As you can see, after robot systems are initialized, the robot control system enters a continuous loop to gather data and issue outputs to steer the robot through the maze.

2.6.1 CONTROL ALGORITHM FOR THE DAGU MAGICIAN ROBOT

In this section, we provide the basic framework for the robot control algorithm. The control algorithm will read the IR sensors attached to the LaunchPad analog in (pins 3–5). In response to the wall placement detected, it will render signals to turn the robot to avoid the maze walls. Provided in Figure 2.17 is a truth table that shows all possibilities of maze placement that the robot might encounter. A detected wall is represented with a logic one. An asserted motor action is also represented with a logic one.

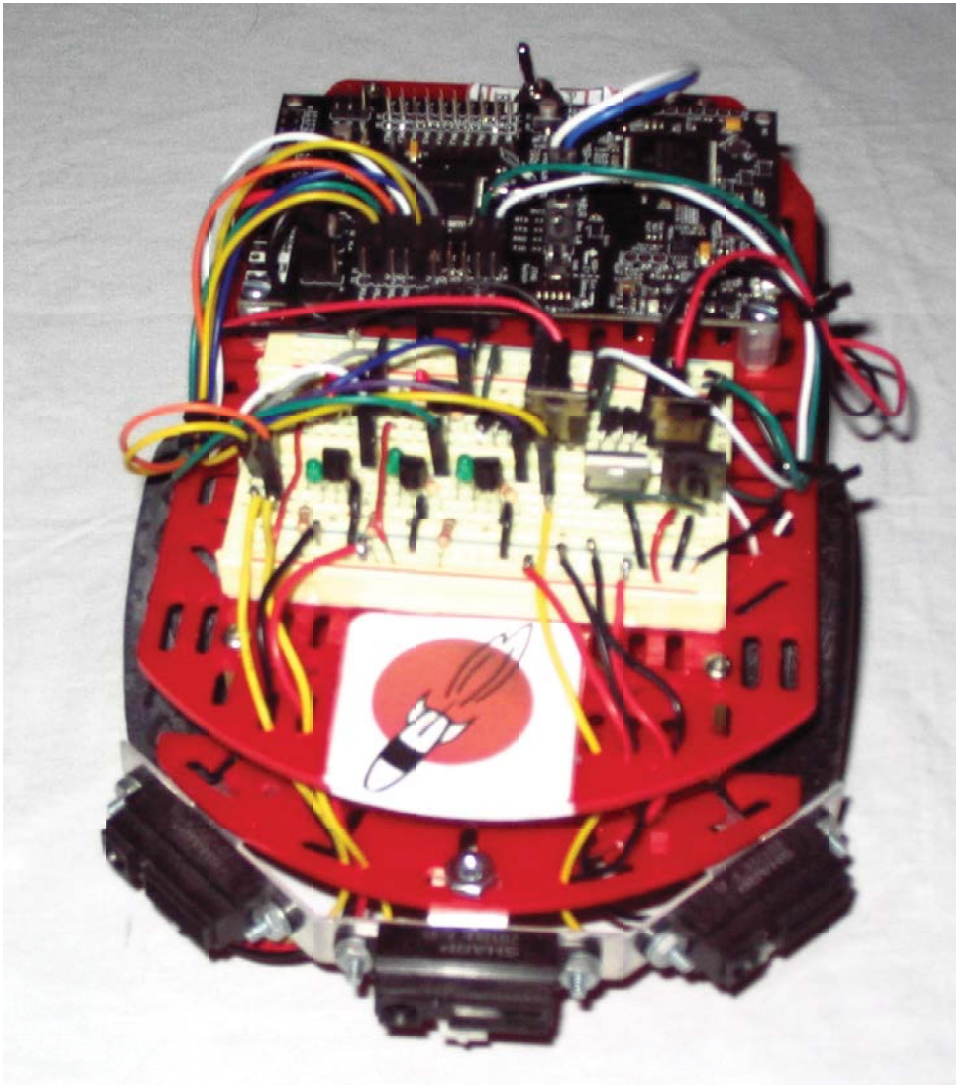
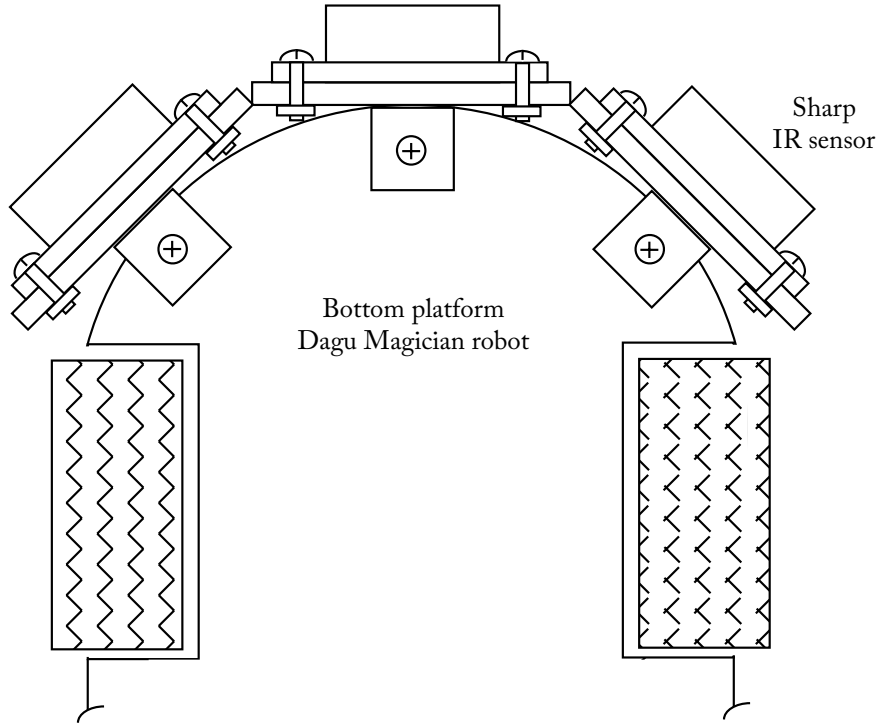
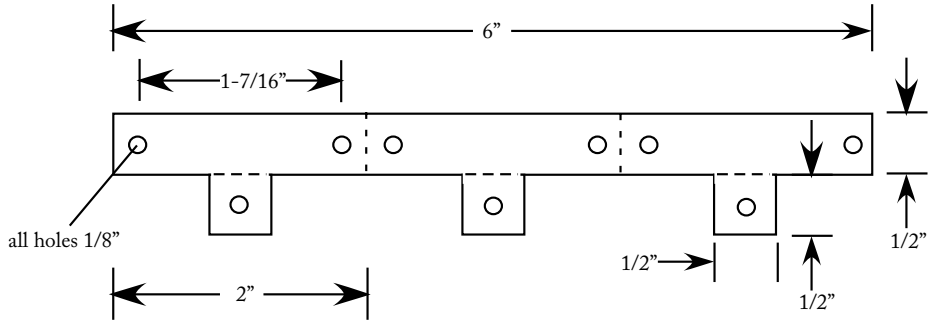


Figure 2.12: Dagu Magician robot.

The robot motors may only be moved in the forward direction. We review techniques to provide bi-directional motor control in an upcoming chapter. To render a left turn, the left motor is stopped and the right motor is asserted until the robot completes the turn. To render a right turn, the opposite action is required.



(a) Top view of robot platform.



(b) Construction details for sensor bracket.

Figure 2.13: Dagu Magician robot platform modified with three IR sensors.

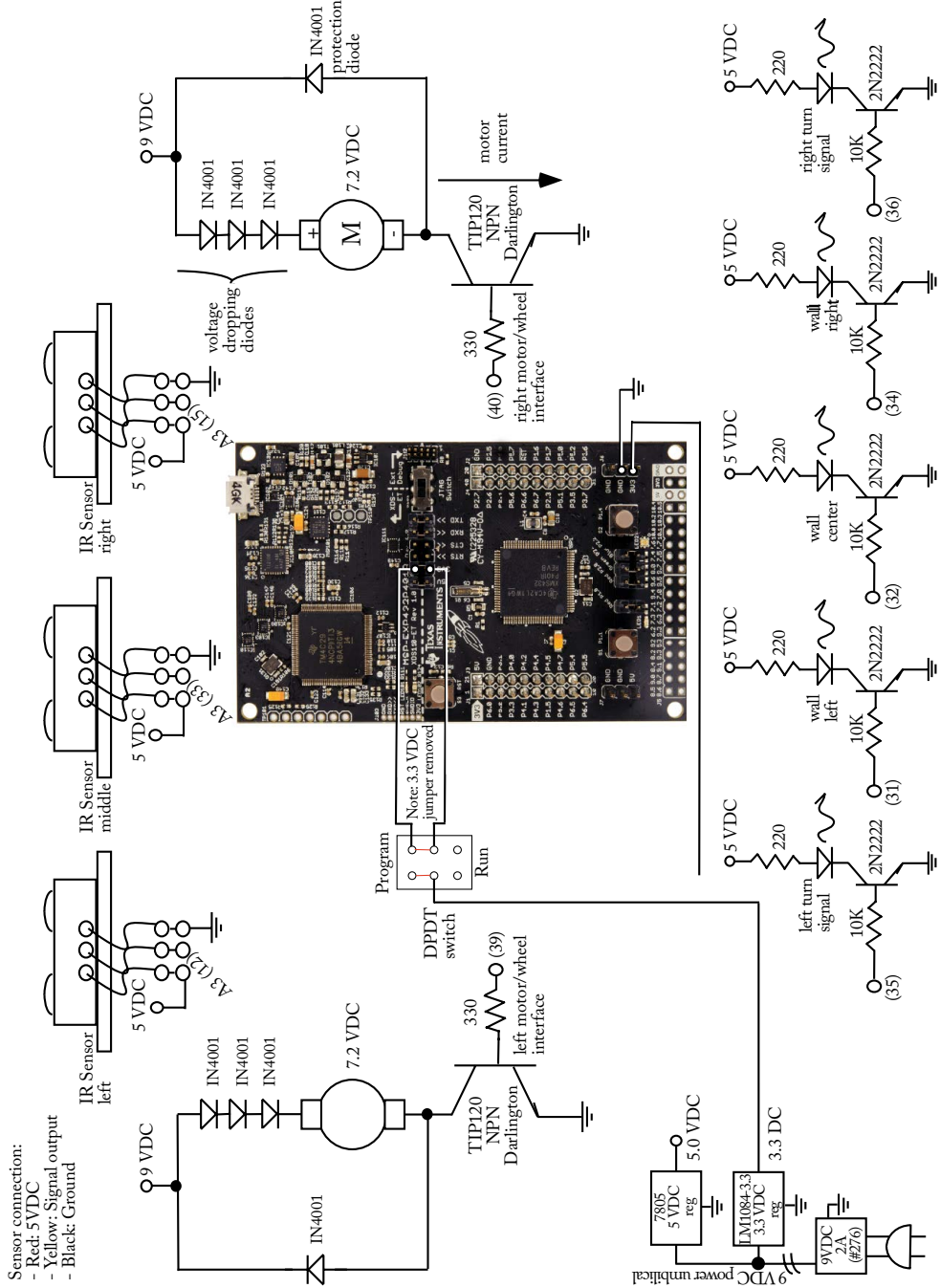


Figure 2.14: Robot circuit diagram.

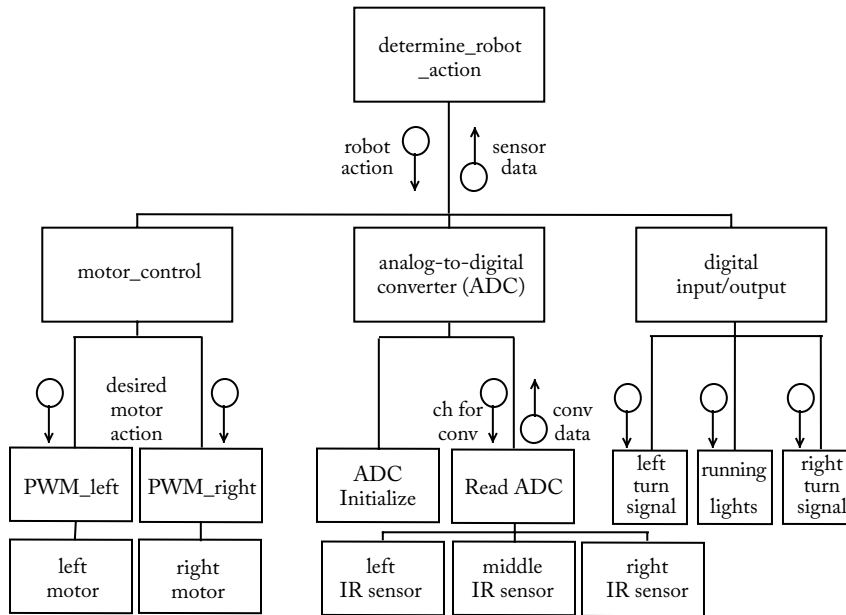


Figure 2.15: Dagu robot structure diagram.

The task in writing the control algorithm is to take the UML activity diagram provided in Figure 2.16 and the actions specified in the robot action truth table (Figure 2.17) and transform both into an Energia sketch. This may seem formidable but we take it a step at a time.

The control algorithm begins with Energia pin definitions. Variables are then declared for the readings from the three IR sensors. The two required Energia functions follow: `setup()` and `loop()`. In the `setup()` function, Energia pins are declared as output. The `loop()` begins by reading the current value of the three IR sensors.

The `analogRead` function reports a value between 0 and 1023. The 0 corresponds to 0 VDC while the value 1023 corresponds to 3.3 VDC. A specific value corresponds to a particular IR sensor range. The threshold detection value may be adjusted to change the range at which the maze wall is detected.

The read of the IR sensors is followed by an eight part if–else if statement. The statement contains a part for each row of the truth table provided in Figure 2.17. For a given configuration of sensed walls, the appropriate wall detection LEDs are illuminated followed by commands to activate the motors (`analogWrite`) and illuminate the appropriate turn signals.

The `analogWrite` command issues a signal from 0–3.3 VDC by sending a constant from 0–255 using pulse width modulation (PWM) techniques. PWM techniques will be discussed in an upcoming chapter. The turn signal commands provide to actions: the appropriate turns signals

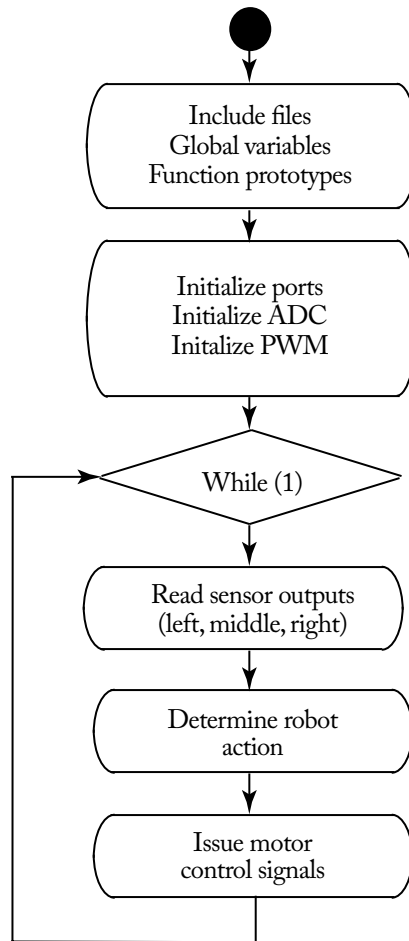


Figure 2.16: Robot UML activity diagram.

are flashed and a 1.5 s total delay is provided. This provides the robot 1.5 s to render a turn. This delay may need to be adjusted during the testing phase.

```

//*****
//robot
//
////This example code is in the public domain.
//*****
//analog input pins

```

	Left Sensor	Middle Sensor	Right Sensor	Wall Left	Wall Middle	Wall Right	Left Motor	Right Motor	Left Signal	Right Signal	Comments
0	0	0	0	0	0	0	1	1	0	0	Forward
1	0	0	1	0	0	1	1	1	0	0	Forward
2	0	1	0	0	1	0	1	0	0	1	Right
3	0	1	1	0	1	1	0	1	1	0	Left
4	1	0	0	1	0	0	1	1	0	0	Forward
5	1	0	1	1	0	1	1	1	0	0	Forward
6	1	1	0	1	1	0	1	0	0	1	Right
7	1	1	1	1	1	1	1	0	0	1	Right

Figure 2.17: Truth table for robot action.

```

#define left_IR_sensor    12           //analog pin - left IR sensor
#define center_IR_sensor  33           //analog pin - center IR sensor
#define right_IR_sensor   13           //analog pin - right IR sensor

                                     //digital output pins
                                     //LED indicators - wall detectors
#define wall_left         31           //digital pin - wall_left
#define wall_center       32           //digital pin - wall_center
#define wall_right        34           //digital pin - wall_right

                                     //LED indicators - turn signals
#define left_turn_signal  35           //digital pin - left_turn_signal
#define right_turn_signal 36           //digital pin - right_turn_signal

                                     //motor outputs
#define left_motor        39           //digital pin - left_motor
#define right_motor       40           //digital pin - right_motor

int left_IR_sensor_value;           //declare variable for left IR sensor

```

```

int center_IR_sensor_value; //declare variable for center IR sensor
int right_IR_sensor_value; //declare variable for right IR sensor

void setup()
{
    //LED indicators - wall detectors
    pinMode(wall_left, OUTPUT); //configure pin for digital output
    pinMode(wall_center, OUTPUT); //configure pin for digital output
    pinMode(wall_right, OUTPUT); //configure pin for digital output

    //LED indicators - turn signals
    pinMode(left_turn_signal,OUTPUT); //configure pin for digital output
    pinMode(right_turn_signal,OUTPUT); //configure pin for digital output

    //motor outputs - PWM
    pinMode(left_motor, OUTPUT); //configure pin for digital output
    pinMode(right_motor, OUTPUT); //configure pin for digital output
}

void loop()
{
    //read analog output from IR sensors
    left_IR_sensor_value = analogRead(left_IR_sensor);
    center_IR_sensor_value = analogRead(center_IR_sensor);
    right_IR_sensor_value = analogRead(right_IR_sensor);

    //robot action table row 0 - robot forward
    if((left_IR_sensor_value < 512)&&(center_IR_sensor_value < 512)&&
        (right_IR_sensor_value < 512))
    {
        //wall detection LEDs
        digitalWrite(wall_left, LOW); //turn LED off
        digitalWrite(wall_center, LOW); //turn LED off
        digitalWrite(wall_right, LOW); //turn LED off

        //motor control
        analogWrite(left_motor, 128); //0(off)-255(full speed)
        analogWrite(right_motor, 128); //0(off)-255(full speed)

        //turn signals
        digitalWrite(left_turn_signal, LOW); //turn LED off
    }
}

```

```

digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
analogWrite(left_motor, 0); //turn motor off
analogWrite(right_motor,0); //turn motor off
}

//robot action table row 1 - robot forward
else if((left_IR_sensor_value < 512)&&(center_IR_sensor_value < 512)&&
(right_IR_sensor_value > 512))
{
//wall detection LEDs
digitalWrite(wall_left, LOW); //turn LED off
digitalWrite(wall_center, LOW); //turn LED off
digitalWrite(wall_right, HIGH); //turn LED on
//motor control
analogWrite(left_motor, 128); //0(off)-255(full speed)
analogWrite(right_motor, 128); //0(off)-255(full speed)
//turn signals
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
analogWrite(left_motor, 0); //turn motor off
analogWrite(right_motor,0); //turn motor off
}

```

```

}

//robot action table row 2 - robot right
else if((left_IR_sensor_value < 512)&&(center_IR_sensor_value > 512)&&
        (right_IR_sensor_value < 512))
{
    //wall detection LEDs
    digitalWrite(wall_left, LOW); //turn LED off
    digitalWrite(wall_center, HIGH); //turn LED on
    digitalWrite(wall_right, LOW); //turn LED off
    //motor control
    analogWrite(left_motor, 128); //0(off)-255(full speed)
    analogWrite(right_motor, 0); //0(off)-255 (full speed)
    //turn signals
    digitalWrite(left_turn_signal, LOW); //turn LED off
    digitalWrite(right_turn_signal, HIGH); //turn LED on
    delay(500); //delay 500 ms
    digitalWrite(left_turn_signal, LOW); //turn LED off
    digitalWrite(right_turn_signal, LOW); //turn LED off
    delay(500); //delay 500 ms
    digitalWrite(left_turn_signal, LOW); //turn LED off
    digitalWrite(right_turn_signal, HIGH); //turn LED on
    delay(500); //delay 500 ms
    digitalWrite(left_turn_signal, LOW); //turn LED off
    digitalWrite(right_turn_signal, LOW); //turn LED off
    analogWrite(left_motor, 0); //turn motor off
    analogWrite(right_motor,0); //turn motor off
}

//robot action table row 3 - robot left
else if((left_IR_sensor_value < 512)&&(center_IR_sensor_value > 512)&&
        (right_IR_sensor_value > 512))
{
    //wall detection LEDs
    digitalWrite(wall_left, LOW); //turn LED off
    digitalWrite(wall_center, HIGH); //turn LED on
    digitalWrite(wall_right, HIGH); //turn LED on
    //motor control
    analogWrite(left_motor, 0); //0(off)-255 (full speed)

```



```

    analogWrite(right_motor, 128);           //0(off)-255 (full speed)
                                           //turn signals
    digitalWrite(left_turn_signal, HIGH);   //turn LED on
    digitalWrite(right_turn_signal, LOW);   //turn LED off
    delay(500);                             //delay 500 ms
    digitalWrite(left_turn_signal, LOW);    //turn LED off
    digitalWrite(right_turn_signal, LOW);   //turn LED off
    delay(500);                             //delay 500 ms
    digitalWrite(left_turn_signal, HIGH);   //turn LED on
    digitalWrite(right_turn_signal, LOW);   //turn LED off
    delay(500);                             //delay 500 ms
    digitalWrite(left_turn_signal, LOW);    //turn LED off
    digitalWrite(right_turn_signal, LOW);   //turn LED off
    analogWrite(left_motor, 0);             //turn motor off
    analogWrite(right_motor,0);             //turn motor off
}

//robot action table row 4 - robot forward
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value < 512)&&
        (right_IR_sensor_value < 512))
{
                                           //wall detection LEDs
    digitalWrite(wall_left, HIGH);         //turn LED on
    digitalWrite(wall_center, LOW);        //turn LED off
    digitalWrite(wall_right, LOW);        //turn LED off
                                           //motor control
    analogWrite(left_motor, 128);          //0(off)-255 (full speed)
    analogWrite(right_motor, 128);         //0(off)-255 (full speed)
                                           //turn signals
    digitalWrite(left_turn_signal, LOW);   //turn LED off
    digitalWrite(right_turn_signal, LOW);  //turn LED off
    delay(500);                           //delay 500 ms
    digitalWrite(left_turn_signal, LOW);   //turn LED off
    digitalWrite(right_turn_signal, LOW);  //turn LED off
    delay(500);                           //delay 500 ms
    digitalWrite(left_turn_signal, LOW);   //turn LED off
    digitalWrite(right_turn_signal, LOW);  //turn LED off
    delay(500);                           //delay 500 ms
    digitalWrite(left_turn_signal, LOW);   //turn LED off

```

```

digitalWrite(right_turn_signal, LOW); //turn LED off
analogWrite(left_motor, 0); //turn motor off
analogWrite(right_motor,0); //turn motor off
}

//robot action table row 5 - robot forward
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value < 512)&&
        (right_IR_sensor_value > 512))
{
//wall detection LEDs
digitalWrite(wall_left, HIGH); //turn LED on
digitalWrite(wall_center, LOW); //turn LED off
digitalWrite(wall_right, HIGH); //turn LED on
//motor control
analogWrite(left_motor, 128); //0(off)-255 (full speed)
analogWrite(right_motor, 128); //0(off)-255 (full speed)
//turn signals
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
delay(500); //delay 500 ms
digitalWrite(left_turn_signal, LOW); //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
analogWrite(left_motor, 0); //turn motor off
analogWrite(right_motor,0); //turn motor off
}

//robot action table row 6 - robot right
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value > 512)&&
        (right_IR_sensor_value < 512))
{
//wall detection LEDs
digitalWrite(wall_left, HIGH); //turn LED on
digitalWrite(wall_center, HIGH); //turn LED on

```

62 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
digitalWrite(wall_right, LOW);           //turn LED off
                                         //motor control
analogWrite(left_motor, 128);           //0(off)-255 (full speed)
analogWrite(right_motor, 0);            //0(off)-255 (full speed)
                                         //turn signals
digitalWrite(left_turn_signal, LOW);    //turn LED off
digitalWrite(right_turn_signal, HIGH);  //turn LED on
delay(500);                             //delay 500 ms
digitalWrite(left_turn_signal, LOW);    //turn LED off
digitalWrite(right_turn_signal, LOW);   //turn LED off
delay(500);                             //delay 500 ms
digitalWrite(left_turn_signal, LOW);    //turn LED off
digitalWrite(right_turn_signal, HIGH);  //turn LED off
delay(500);                             //delay 500 ms
digitalWrite(left_turn_signal, LOW);    //turn LED OFF
digitalWrite(right_turn_signal, LOW);   //turn LED OFF
analogWrite(left_motor, 0);             //turn motor off
analogWrite(right_motor,0);            //turn motor off
}

//robot action table row 7 - robot right
else if((left_IR_sensor_value > 512)&&(center_IR_sensor_value > 512)&&
(right_IR_sensor_value > 512))
{
                                         //wall detection LEDs
digitalWrite(wall_left, HIGH);          //turn LED on
digitalWrite(wall_center, HIGH);        //turn LED on
digitalWrite(wall_right, HIGH);         //turn LED on
                                         //motor control
analogWrite(left_motor, 128);           //0(off)-255 (full speed)
analogWrite(right_motor, 0);            //0(off)-255 (full speed)
                                         //turn signals
digitalWrite(left_turn_signal, LOW);    //turn LED off
digitalWrite(right_turn_signal, HIGH);  //turn LED on
delay(500);                             //delay 500 ms
digitalWrite(left_turn_signal, LOW);    //turn LED off
digitalWrite(right_turn_signal, LOW);   //turn LED off
delay(500);                             //delay 500 ms
digitalWrite(left_turn_signal, LOW);    //turn LED off
```

```

digitalWrite(right_turn_signal, HIGH); //turn LED on
delay(500);                          //delay 500 ms
digitalWrite(left_turn_signal, LOW);  //turn LED off
digitalWrite(right_turn_signal, LOW); //turn LED off
analogWrite(left_motor, 0);           //turn motor off
analogWrite(right_motor,0);          //turn motor off
}
}
//*****

```

Testing the control algorithm: It is recommended that the algorithm be first tested without the entire robot platform. This may be accomplished by connecting the three IR sensors and LEDs to the appropriate pins on the LaunchPad as specified in Figure 2.14. In place of the two motors and their interface circuits, two LEDs with the required interface circuitry may be used. The LEDs will illuminate to indicate the motors would be on during different test scenarios. Once this algorithm is fully tested in this fashion, the LaunchPad may be mounted to the robot platform and connected to the motors. Full up testing in the maze may commence. Enjoy!

2.7 SOME ADDITIONAL COMMENTS ON ENERGIA

Keep in mind the Energia is based on the open source concept. Users throughout the world are constantly adding new built-in features. As new features are added, they will be released in future Energia IDE versions. As an Energia user, you too may add to this collection of useful tools. In the next section we investigate programming in C.

2.8 PROGRAMMING IN C

Most microcontrollers are programmed with some variant of the C programming language. The C programming language provides a nice balance between the programmer's control of the microcontroller hardware and time efficiency in programming writing.

As you can see in Figure 2.18, the compiler software is hosted on a computer separate from the LaunchPad. The job of the compiler is to transform the program provided by the program writer (filename.c and filename.h) into machine code suitable for loading into the processor.

Once the source files (filename.c and filename.h) are provided to the compiler, the compiler executes two steps to render the machine code. The first step is the compilation process. Here the program source files are transformed into assembly code (filename.asm). If the program source files contains syntax errors, the compiler reports these to the user. Syntax errors are reported for incorrect use of the C programming language. An assembly language program is not generated until the syntax errors have been corrected. The assembly language source file is then passed to the

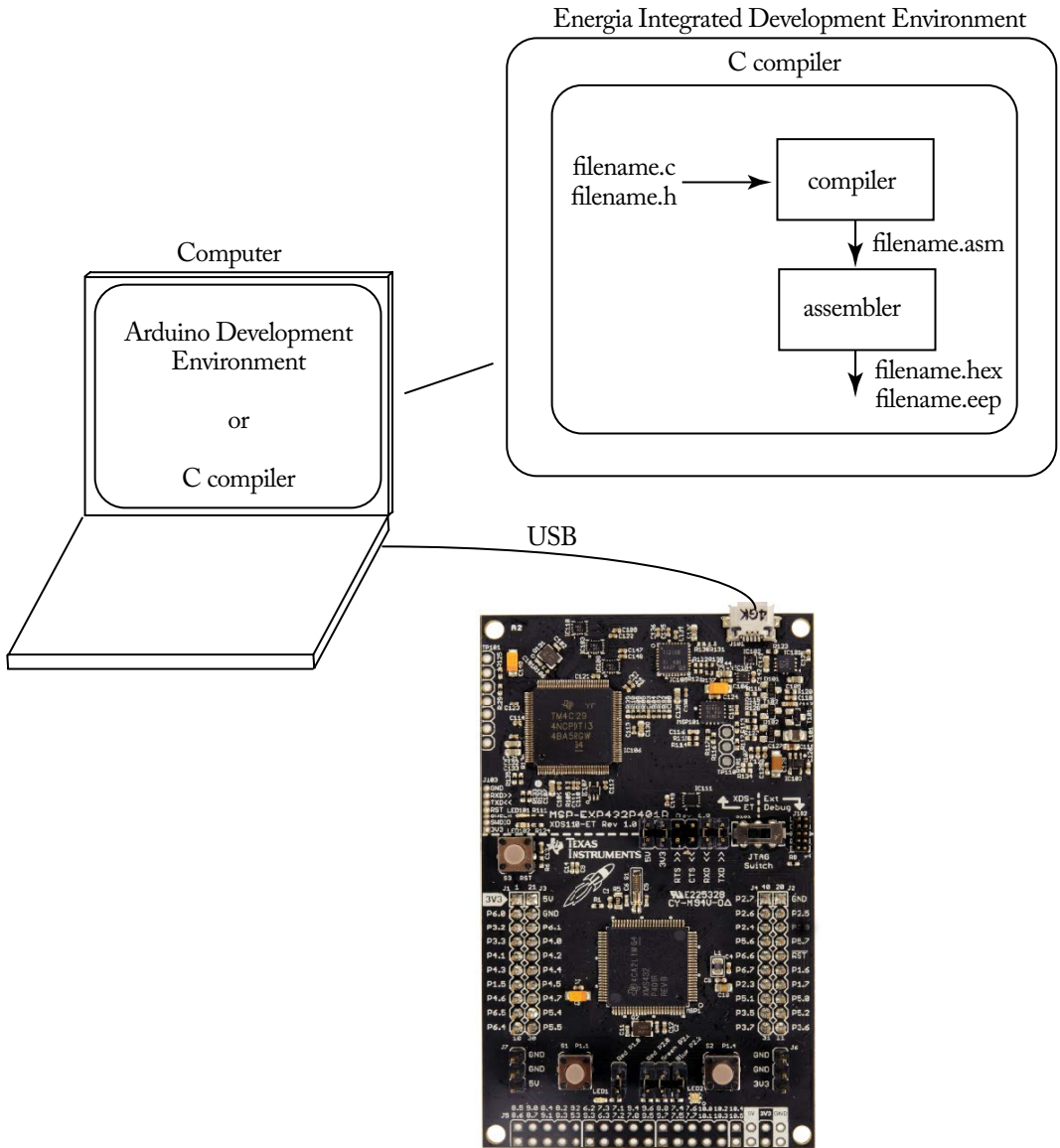


Figure 2.18: Programming the LaunchPad.

assembler. The assembler transforms the assembly language source file to machine code suitable for loading to the LaunchPad.

During the compilation process, warnings may also be generated. Warnings do not prevent the creation of an assembly language version of the C program. However, they should be resolved since flagged incorrect usage of the C language may result in unexpected program run time errors.

As seen earlier in the chapter, the Energia Integrated Development Environment provides a user-friendly interface to aid in program development, transformation to machine code, and loading into the LaunchPad. As described in Chapter 1, the LaunchPad may also be programmed using Code Composer Studio, Keil, and IAR Systems software. We use Code Composer Studio throughout the book.

For the remaining portion of the chapter we present a brief introduction to C. Many examples are provided. We encourage the reader to modify, load, and run the examples on the LaunchPad.

Example 8: If not already done, complete Lab 1: Getting Acquainted with Hardware and Software Development Tools in Chapter 1.

In the next section, we will discuss the components of a C program.

2.9 ANATOMY OF A PROGRAM

Programs written for a microcontroller have a fairly repeatable format. Slight variations exist but many follow the format provided.

```
//*****
//Comments containing program information
// - file name:
// - author:
// - revision history:
// - compiler setting information:
// - hardware connection description to microcontroller pins
// - program description
//*****

//include files
#include<file_name.h>

//function prototypes
A list of functions and their format used within the program

//program constants
#define TRUE 1
#define FALSE 0
```

```

#define    ON      1
#define    OFF     0

//interrupt handler definitions
Used to link the software to hardware interrupt features

//global variables
Listing of variables used throughout the program

//main program

void main(void)
{

body of the main program

}

//*****
//function definitions: A detailed function body and definition
//for each function used within the program.
For larger
//programs, function definitions may be placed in accompanying
//header files.
//*****

```

Let's take a closer look at each part of the program.

2.9.1 COMMENTS

Comments are used throughout the program to document what and how things were accomplished within a program. The comments help you and others to reconstruct your work at a later time. Imagine that you wrote a program a year ago for a project. You now want to modify that program for a new project. The comments will help you remember the key details of the program.

Comments are not compiled into machine code for loading into the microcontroller. Therefore, the comments will not fill up the memory of your microcontroller. Comments are indicated using double slashes (//). Anything from the double slashes to the end of a line is then considered a comment. A multi-line comment can be constructed using a /* at the beginning of the comment and a */ at the end of the comment. These are handy to block out portions of code during troubleshooting or providing multi-line comments.

At the beginning of the program, comments may be extensive. Comments may include some of the following information:

- file name,
- program author and dates of creation,
- revision history or a listing of the key changes made to the program,
- compiler setting information,
- hardware connection description to microcontroller pins, and
- program description.

2.9.2 INCLUDE FILES

Often you need to add extra files to your project besides the main program. For example, most compilers require a “personality file” on the specific microcontroller that you are using. This file is provided with the compiler and provides the name of each register used within the microcontroller. It also provides the link between a specific register’s name within software and the actual register location within hardware. These files are typically called header files and their name ends with a “.h”. Within the C compiler there will also be other header files to include in your program such as the “math.h” file when programming with advanced math functions.

To include header files within a program, the following syntax is used:

```
//C programming: include files
#include<file_name1.h> //searches for file in a standard list
#include<file_name2.h>
#include 'file_name3.h' //searches for file in current directory
```

2.9.3 FUNCTIONS

Later in the book we discuss in detail the top-down design, bottom-up implementation approach to designing microcontroller based systems. In this approach, a microcontroller based project including both hardware and software is partitioned into systems, subsystems, etc. The idea is to take a complex project and break it into doable pieces with a defined action.

We use the same approach when writing computer programs. At the highest level is the main program which calls functions that have a defined action. When a function is called, program control is released from the main program to the function. Once the function is complete, program control reverts back to the main program.

Functions may in turn call other functions as shown in Figure 2.19. This approach results in a collection of functions that may be reused over and over again in various projects. Most importantly, the program is now subdivided into doable pieces, each with a defined action. This

makes writing the program easier but also makes it convenient to modify the program since every action is in a known location.

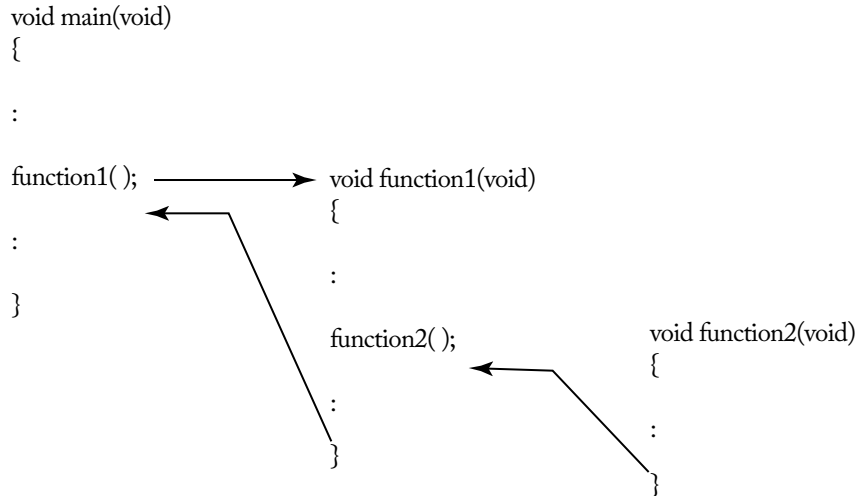


Figure 2.19: Function calling.

There are three different pieces of code required to properly configure and call a function:

- function prototype,
- function call, and
- function body.

Function prototypes are provided early in the program as previously shown in the program template. The function prototype provides the name of the function and any variables required by the function and any variable returned by the function.

The function prototype follows this format:

```
return_variable function_name(required_variable1, required_variable2);
```

If the function does not require variables or sends back a variable the word “void” is placed in the variable’s position.

The **function call** is the code statement used within a program to execute the function. The function call consists of the function name and the actual arguments required by the function. If the function does not require arguments to be delivered to it for processing, the parenthesis containing the variable list is left empty.

The function call follows this format:

```
function_name(required_variable1, required_variable2);
```

A function that requires no variables is called by:

```
function_name( );
```

When the function call is executed by the program, program control is transferred to the function, the function is executed, and program control is then returned to the portion of the program that called it.

The **function body** is a self-contained “mini-program.” The first line of the function body contains the same information as the function prototype: the name of the function, any variables required by the function, and any variable returned by the function. The last line of the function contains a “return” statement. Here a variable may be sent back to the portion of the program that called the function. The processing action of the function is contained within the open ({} and close brackets (}). If the function requires any variables within the confines of the function, they are declared next. These variable are referred to as local variables. A local variable is known only within the confines of a specific function. The actions required by the function follow.

The function prototype follows this format:

```
return_variable function_name(required_variable1, required_variable2)
{
//local variables required by the function
unsigned int variable1;
unsigned char variable2;

//program statements required by the function

//return variable
return return_variable;
}
```

2.9.4 PORT CONFIGURATION

The MSP432 is equipped with a complement of input/output (I/O) ports designated P1 through P10 and PJ. Ports P1–P10 may be read and written as 8-bit ports or may be grouped into pairs and designated PA, PB, etc. The half-word (16-bit) letter designated ports may be read or written. To provide for complete flexibility, each individual port pin may be separately addressed. The specific configuration of digital I/O pins as individual pins, 8-bit ports, or 16-bit ports is determined by the specific application.

Configuration and access to digital I/O pins is provided by a complement of registers. These registers include the following.

- **Input Registers (PxIN):** Allows input logic value of pin to be read (1: High, 0: Low).
- **Output Registers (PxOUT):** Value of output register is provided to corresponding output pin (1: High, 0: Low).
- **Direction Registers (PxDIR):** Bit in PxDIR selects corresponding digital I/O pin as (1: output, 0: input).
- **Pullup or Pulldown Resistor Enable Registers (PxREN):** Each bit determines if an internal pulled up (or pulled down) resistor is enabled at the corresponding pin. The value of the corresponding PxOUT register determines if pulled up (1) or pulled down (0) is selected. In summary, use the following PxDIR, PxREN, and PxOUT settings:
 - 00x: input
 - 010: input with pulldown resistor
 - 011: input with pullup resistor
 - 1xx: output
- **Output Drive Strength Selection Registers (PxDS):** The value of the register determines the drive strength for specific pins (1: high drive strength, 0: regular drive strength).
- **Function Select Registers (PxSEL0, PxSEL1):** Allows specific function of multi-function pins to have access to I/O pin.

Throughout the book we use two approaches to configure MSP432 subsystems via their complement of control registers. The registers may be configured directly using C programming techniques (the “bare metal” approach) or via higher level application program interface (APIs). The MSP432 has an extensive complement of APIs for all MSP432 subsystems within the MSPWare library.

For example, MSPWare has the following APIs available for general purpose input/output:

- void GPIO_setAsOutputPin(uint_fast8_t selectedPort, uint_fast16_t selectedPins)
- void GPIO_setAsInputPin(uint_fast8_t selectedPort, uint_fast16_t selectedPins)
- void GPIO_setAsPeripheralModuleFunctionOutputPin(uint_fast8_t selectedPort, uint_fast16_t selectedPins, uint_fast8_t mode)
- void GPIO_setAsPeripheralModuleFunctionInputPin(uint_fast8_t selectedPort, uint_fast16_t selectedPins, uint_fast8_t mode)
- void GPIO_setOutputHighOnPin(uint_fast8_t selectedPort, uint_fast16_t selectedPins)
- void GPIO_setOutputLowOnPin(uint_fast8_t selectedPort, uint_fast16_t selectedPins)

- void GPIO_setAsInputPinWithPullDownResistor(uint_fast8_t selectedPort, uint_fast16_t selectedPins)
- void GPIO_setAsInputPinWithPullUpResistor(uint_fast8_t selectedPort, uint_fast16_t selectedPins)
- uint8_t GPIO_getInputPinValue(uint_fast8_t selectedPort, uint_fast16_t selectedPins)
- void GPIO_setDriveStrengthHigh(uint_fast8_t selectedPort, uint_fast8_t selectedPins)
- void GPIO_setDriveStrengthLow(uint_fast8_t selectedPort, uint_fast8_t selectedPins)

In the following examples, we revisit the blink LED example using the “bare metal” approach and the MSPWare API approach.

Register configuration: The “msp.h” header file within the CCS compiler provides the link between software and the MSP432 hardware. Definitions for each user-accessible MSP432 register and pin is provided here. It is recommended the reader take a few minutes and examine the contents of the “msp.h” header file. Provided below is a snapshot of the file containing memory locations for the Port A associated registers.

```

//*****
// DIO Registers
//
//Copyright, Texas Instruments, [www.TI.com]
//*****
#define PAIN    (HWREG16(0x40004C00))    //Port A Input
#define PAOUT   (HWREG16(0x40004C02))    //Port A Output
#define PADIR   (HWREG16(0x40004C04))    //Port A Direction
#define PAREN   (HWREG16(0x40004C06))    //Port A Resistor Enable
#define PADS    (HWREG16(0x40004C08))    //Port A Drive Strength
#define PASELO  (HWREG16(0x40004C0A))    //Port A Select 0
#define PASEL1  (HWREG16(0x40004C0C))    //Port A Select 1
//*****

```

Recall the 16-bit PORTA is comprised of two 8-bit ports P1 and P2. Provided below is a snapshot of the file containing memory locations for the Port P1 and P2 associated registers.

```

//*****
// DIO Registers
//
//Copyright, Texas Instruments, [www.TI.com]
//*****
#define P1IN    (HWREG8(0x40004C00))    //Port 1 Input

```

72 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
#define P2IN    (HWREG8(0x40004C01))    //Port 2 Input
#define P2OUT   (HWREG8(0x40004C03))    //Port 2 Output
#define P1OUT   (HWREG8(0x40004C02))    //Port 1 Output
#define P1DIR   (HWREG8(0x40004C04))    //Port 1 Direction
#define P2DIR   (HWREG8(0x40004C05))    //Port 2 Direction
#define P1REN   (HWREG8(0x40004C06))    //Port 1 Resistor Enable
#define P2REN   (HWREG8(0x40004C07))    //Port 2 Resistor Enable
#define P1DS    (HWREG8(0x40004C08))    //Port 1 Drive Strength
#define P2DS    (HWREG8(0x40004C09))    //Port 2 Drive Strength
#define P1SEL0  (HWREG8(0x40004C0A))    //Port 1 Select 0
#define P2SEL0  (HWREG8(0x40004C0B))    //Port 2 Select 0
#define P1SEL1  (HWREG8(0x40004C0C))    //Port 1 Select 1
#define P2SEL1  (HWREG8(0x40004C0D))    //Port 2 Select 1
//*****
```

Also included in the header file are definitions for each register. For example, provided below is the definition for the PADIR register referred to as “rPADIR.”

```
//*****
union
{
    //PADIR Register
    __IO uint16_t r;
    struct
    {
        //PADIR Bits
        __IO uint16_t bP1DIR : 8;    //Port 1 Direction
        __IO uint16_t bP2DIR : 8;    //Port 2 Direction
    }b;
}rPADIR;
//*****
```

Definitions are also provided for the constituent bP1DIR and bP2DIR bits:

```
//*****
//PADIR[P1DIR] Bits
#define P1DIR_OFS    ( 0)    //P1DIR Offset
#define P1DIR_M      (0x00ff) //Port 1 Direction
//PADIR[P2DIR] Bits
#define P2DIR_OFS    ( 8)    //P2DIR Offset
#define P2DIR_M      (0xff00) //Port 2 Direction
//*****
```

The following example toggles the LED at pin P1.0. The program begins by disabling the watchdog timer. This timer is discussed in a later chapter. The next line configures pin P1.0 as an output pin. Note how the specific bit related to pin P1.0 is accessed within the PADIR register. This technique will be used throughout the book to configure registers.

```
//*****
// MSP432 main.c - P1.0 port toggle
//
//Copyright, Texas Instruments, [www.TI.com]
//*****

#include "msp.h"

void main(void)
{
volatile uint32_t i;

WDT_A->rCTL.r = WDTPW | WDTHOLD;           //Stop watchdog timer

DIO->rPADIR.b.bP1DIR |= BIT0;              //Configure P1.0 as output

                                           //Code toggles P1.0 port
while(1)
{
    DIO->rPAOUT.b.bP1OUT ^= BIT0;          //Toggle P1.0
    for(i=10000; i>0; i--);               //Delay
}
}
//*****
```

MSPWare API approach: The “driverlib.h” header file pulls in other multiple header files containing API definitions for the subsystems aboard the MSP432. The following example toggles the LED connected to P1.0 using APIs.

```
//*****
//MSP432 main.c - P1.0 port toggle
//copyright: Texas Instruments, Inc
//Created by: E. Chen, March 2015
//Built with Code Composer Studio v6
//*****
```

```

#include <driverlib.h>

void main(void)
{
volatile uint32_t i;

WDT_A_hold(WDT_A_BASE);                //Stop watchdog timer

                                        //Set P1.0 to output
GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

while(1)
{
                                        //Toggle P1.0 output
GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

for(i=10000; i>0; i--);                //Delay
}
}
//*****

```

2.9.5 PROGRAM CONSTANTS

The `#define` statement is used to associate a constant name with a numerical value in a program. It can be used to define common constants such as `pi`. It may also be used to give terms used within a program a numerical value. This makes the code easier to read. For example, the following constants may be defined within a program:

```

//program constants
#define TRUE 1
#define FALSE 0
#define ON 1
#define OFF 0

```

2.9.6 INTERRUPT HANDLER DEFINITIONS

Interrupts are functions that are written by the programmer but usually called by a specific hardware event during system operation. We discuss interrupts and how to properly configure them in an upcoming chapter.

2.9.7 VARIABLES

There are two types of variables used within a program: global variables and local variables. A global variable is available and accessible to all portions of the program, whereas a local variable is only known and accessible within the function where it is declared.

When declaring a variable in C, the number of bits used to store the variable is also specified. Variable specifications may vary by compiler. For code portability among different platforms fixed formats may be used.

Type	Size	Range
unsigned char	1	0..255
signed char	1	-128..127
unsigned int	2	0..65535
signed int	2	-32768..32767
float	4	+/-1.175e-38..+/-3.40e+38
double	4 - 8	compiler dependent

Figure 2.20: C variable sizes.

Fixed format variable are defined within the “stdint.h” header file [stdint.h]. Provided below is a small extract from this header file.

```
//*****
typedef signed char int8_t;
typedef unsigned char uint8_t;
typedef int int16_t;
typedef unsigned int uint16_t;
typedef long int32_t;
typedef unsigned long uint32_t;
typedef long long int64_t;
typedef unsigned long long uint64_t;

//*****
```

When programming microcontrollers, it is important to know the number of bits and the memory location used to store the variable. For example, assigning the contents of an unsigned char variable, which is stored in 8-bits, to an 8-bit output port will have a predictable result. However, assigning an unsigned int variable, which is stored in 32-bits, to an 8-bit output port does not provide predictable results. It is wise to ensure your assignment statements are balanced

for accurate and predictable results. The modifier “unsigned” indicates all bits will be used to specify the magnitude of the argument. Signed variables will use the left most bit to indicate the polarity (\pm) of the argument.

Variables may be read (scanned) into a program using the “scanf” statement. The general format of the scanf statement is provided below. The format of the variable and the variable name are specified. Similarly, the variables may be printed using the “printf” statement. The backslash n specifies start a new line.

```
//*****
#include<stdio.h>

int main( )
{
int input_variable;

scanf("%d", &input_variable);

printf("%d\n", input_variable);

}

//*****
```

A global variable is declared using the following format provided below. The type of the variable is specified, followed by its name, and an initial value if desired.

```
//*****
//global variables
unsigned int loop_iterations = 6;

//*****
```

2.9.8 MAIN PROGRAM

The main program is the hub of activity for the entire program. The main program typically consists of program steps and function calls to initialize the processor followed by program steps to collect data from the environment external to the microcontroller, process the data and make decisions, and provide external control signals back to the environment based on the data collected.

2.10 FUNDAMENTAL PROGRAMMING CONCEPTS

In the previous section, we covered many fundamental concepts. In this section we discuss operators, programming constructs, and decision processing constructs to complete our fundamental overview of programming concepts.

2.10.1 OPERATORS

There are a wide variety of operators provided in the C language. An abbreviated list of common operators are provided in Figures 2.21 and 2.22. The operators have been grouped by general category. The symbol, precedence, and brief description of each operator are provided. The precedence column indicates the priority of the operator in a program statement containing multiple operators. Only the fundamental operators are provided.

General		
Symbol	Precedence	Description
{ }	1	Brackets, used to group program statements
()	1	Parenthesis, used to establish precedence
=	12	Assignment
Arithmetic Operations		
Symbol	Precedence	Description
*	3	Multiplication
/	3	Division
+	4	Addition
-	4	Subtraction
Logical Operations		
Symbol	Precedence	Description
<	6	Less than
<=	6	Less than or equal to
>	6	Greater than
>=	6	Greater than or equal to
==	7	Equal to
!=	7	Not equal to
&&	8	Logical AND
	10	Logical OR

Figure 2.21: C operators. (Adapted from Barrett and Pack [2005].)

Bit Manipulation Operations		
Symbol	Precedence	Description
<<	5	Shift left
>>	5	Shift right
&	8	Bitwise AND
^	8	Bitwise exclusive OR
	8	Bitwise OR
Unary Operations		
Symbol	Precedence	Description
!	2	Unary negative
~	2	One's complement (bit-by-bit inversion)
++	2	Increment
--	2	Decrement
type (argument)	2	Casting operator (data type conversion)

Figure 2.22: C operators. (Adapted from Barrett and Pack [2005].)

General Operations

Within the general operations category are brackets, parenthesis, and the assignment operator. We have seen in an earlier example how bracket pairs are used to indicate the beginning and end of the main program or a function. They are also used to group statements in programming constructs and decision processing constructs. This is discussed in the next several sections.

The parenthesis is used to boost the priority of an operator. For example, in the mathematical expression $7 \times 3 + 10$, the multiplication operation is performed before the addition since it has a higher precedence. Parenthesis may be used to boost the precedence of the addition operation. If we contain the addition operation within parenthesis $7 \times (3 + 10)$, the addition will be performed before the multiplication operation and yield a different result from the earlier expression.

The assignment operator ($=$) is used to assign the argument(s) on the right-hand side of an equation to the left-hand side variable. It is important to insure that the left and the right-hand side of the equation have the same type of arguments. If not, unpredictable results may occur.

Arithmetic Operations

The arithmetic operations provide for basic math operations using the various variables described in the previous section. As described in the previous section, the assignment operator ($=$) is used to assign the argument(s) on the right-hand side of an equation to the left-hand side variable.

In this example, a function returns the sum of two unsigned int variables passed to the function.

```
//*****
unsigned int  sum_two(unsigned int variable1, unsigned int variable2)
{
unsigned int  sum;

sum = variable1 + variable2;

return sum;
}

//*****
```

Logical Operations

The logical operators provide Boolean logic operations. They can be viewed as comparison operators. One argument is compared against another using the logical operator provided. The result is returned as a logic value of one (1, true, high) or zero (0 false, low). The logical operators are used extensively in program constructs and decision processing operations to be discussed in the next several sections.

Bit Manipulation Operations

There are two general types of operations in the bit manipulation category: shifting operations and bitwise operations. Let's examine several examples.

Given the following code segment, what will the value of variable2 be after execution?

```
//*****

unsigned char  variable1 = 0x73;
unsigned char  variable2;

variable2 = variable1 << 2;

//*****
```

Answer: Variable "variable1" is declared as an eight bit unsigned char and assigned the hexadecimal value of $(73)_{16}$. In binary this is $(0111_0011)_2$. The $<< 2$ operator provides a left shift of the argument by two places. After two left shifts of $(73)_{16}$, the result is $(cc)_{16}$ and will be assigned to the variable "variable2." Note that the left and right shift operation is equivalent to multiplying and dividing the variable by a power of two.

The bitwise operators perform the desired operation on a bit-by-bit basis. That is, the least significant bit of the first argument is bit-wise operated with the least significant bit of the second argument and so on.

Given the following code segment, what will the value of variable3 be after execution?

```
//*****
unsigned char   variable1 = 0x73;
unsigned char   variable2 = 0xfa;
unsigned char   variable3;

variable3 = variable1 & variable2;

//*****
```

Answer: Variable “variable1” is declared as an eight bit unsigned char and assigned the hexadecimal value of $(73)_{16}$. In binary, this is $(0111_0011)_2$. Variable “variable2” is declared as an eight bit unsigned char and assigned the hexadecimal value of $(fa)_{16}$. In binary, this is $(1111_1010)_2$. The bitwise AND operator is specified. After execution variable “variable3,” declared as an eight bit unsigned char, contains the hexadecimal value of $(72)_{16}$.

Unary Operations

The unary operators, as their name implies, require only a single argument. For example, in the following code segment, the value of the variable “i” is incremented. This is a shorthand method of executing the operation “ $i = i + 1$,”

```
//*****
unsigned int    i;

i++;

//*****
```

Example 9: It is not uncommon in embedded system design projects to have every pin on a microcontroller employed. Furthermore, it is not uncommon to have multiple inputs and outputs assigned to the same port but on different port input/output pins. Some compilers support specific pin reference. Another technique that is not compiler specific is **bit twiddling**. Figure 2.23 provides bit twiddling examples on how individual bits may be manipulated without affecting other bits using bitwise and unary operators. The information provided here was extracted from the ImageCraft ICC AVR compiler documentation [ImageCraft].

Syntax	Description	Example
<code>a b</code>	Bitwise or	<code>P2OUT = 0x80; // turn on bit 7 (msb)</code>
<code>a & b</code>	Bitwise and	<code>if ((P2IN & 0x81) == 0) // check bit 7 and bit 0</code>
<code>a ^ b</code>	Bitwise exclusive or	<code>P2OUT ^= 0x80; // flip bit 7</code>
<code>~a</code>	Bitwise complement	<code>P2OUT &= ~0x80; // turn off bit 7</code>

Figure 2.23: Bit twiddling [ImageCraft].

2.10.2 PROGRAMMING CONSTRUCTS

In this section, we discuss several methods of looping through a piece of code. We will examine the “for” and the “while” looping constructs.

The **for** loop provides a mechanism for looping through the same portion of code a fixed number of times. The for loop consists of three main parts:

- loop initiation,
- loop termination testing, and
- the loop increment.

In the following code fragment the for loop is executed ten times.

```

//*****
unsigned int  loop_ctr;

for(loop_ctr = 0; loop_ctr < 10; loop_ctr++)
{
    //loop body
}

//*****

```

The for loop begins with the variable “loop_ctr” equal to 0. During the first pass through the loop, the variable retains this value. During the next pass through the loop, the variable “loop_ctr” is incremented by one. This action continues until the “loop_ctr” variable reaches the value of ten. Since the argument to continue the loop is no longer true, program execution continues with the next instruction after the close bracket of the for loop.

In the previous example, the for loop counter was incremented by one. The “loop_ctr” variable can be updated by any amount. For example, in the following code fragment the “loop_ctr” variable is increased by three for every pass of the loop.

```
//*****
unsigned int  loop_ctr;

for(loop_ctr = 0; loop_ctr < 10; loop_ctr=loop_ctr+3)
{
    //loop body

}

//*****
```

The “loop_ctr” variable may also be initialized at a high value and then decremented at the beginning of each pass of the loop as shown below.

```
//*****
unsigned int  loop_ctr;

for(loop_ctr = 10; loop_ctr > 0; loop_ctr--)
{
    //loop body

}

//*****
```

As before, the “loop_ctr” variable may be decreased by any numerical value as appropriate for the application at hand.

The **while** loop is another programming construct that allows multiple passes through a portion of code. The while loop will continue to execute the statements within the open and close brackets while the condition at the beginning of the loop remains logically true. The code snapshot below will implement a ten iteration loop. Note how the “loop_ctr” variable is initialized outside of the loop and incremented within the body of the loop. As before, the variable may be initialized to a greater value and then decremented within the loop body.

```
//*****
```

```

unsigned int  loop_ctr;

loop_ctr = 0;
while(loop_ctr < 10)
{
    //loop body
    loop_ctr++;
}

//*****

```

Frequently, within a microcontroller application, the program begins with system initialization actions. Once initialization activities are completed, the processor enters a continuous loop. This may be accomplished using the following code fragment.

```

//*****

while(1)
{

}

//*****

```

2.10.3 DECISION PROCESSING

There are a variety of constructs that allow decision making. These include the following:

- the **if** statement,
- the **if-else** construct,
- the **if-else if-else** construct, and the
- **switch** statement.

The **if** statement will execute the code between an open and close bracket set should the condition within the if statement be logically true. The **if-else** statement will execute the code between an open and close bracket set should the condition within the if statement be logically true. If the statement is not true, the code after the else is executed.

Example 10: In this example switch S1 is connected to P1.1 (S1 on the MSP-EXP432P401R LaunchPad). If the switch is at logic one (switch not pressed), the red LED

84 2. A BRIEF INTRODUCTION TO PROGRAMMING

at pin P1.0 is illuminated. If the switch is pressed taking P1.1 to logic low, the LED goes out. In the example, pay close attention to how P1.0 is configured for output and P1.1 is configured for input with the corresponding pullup resistor activated using bit-twiddling techniques.

```

//*****
// if_example.c
//*****

#include "msp.h"

void main(void)
{
    unsigned char switch_value;

    WDT_A->rCTL.r = WDTPW | WDT_HOLD;    //Stop watchdog timer

    //Configure P1.0 as output (1) for LED
    //0x used to designate hexadecimal number
    DIO->rPADIR.b.bP1DIR |= 0x01;        //Bit 0 to logic 1

    //Configure P1.1 as input (0) for switch with pullup
    //resistor enabled:
    // PxDIR = 0, PxREN = 1, PxOUT = 1
    DIO->rPADIR.b.bP1DIR &= 0xfd;        //Bit 1 to logic 0
    DIO->rPAREN.b.bP1REN |= 0x02;        //Bit 1 to logic 1
    DIO->rPAOUT.b.bP1OUT |= 0x02;        //Bit 1 to logic 1

    while(1)
    {
        switch_value = DIO->rPAIN.b.bP1IN;

        if((switch_value & 0x02) == 0x02)
        {
            DIO->rPAOUT.b.bP1OUT |= 0x01;    //P1.0 to logic one
            printf("high\n");
        }
        else
        {
            DIO->rPAOUT.b.bP1OUT &= 0xFE;    //P1.0 to logic zero
            printf("low\n");
        }
    }
}

```

```

    }
}
}

```

```

//*****

```

The **if-else if-else** construct may be used to implement a three LED system. In this example, the individual component colors of the RGB LED are illuminated depending on the integer input by the user.

```

//*****

```

```

//if_RGB.c

```

```

//

```

```

//User is prompted for an integer between 0 and 100. Based on the number
//provided the red (0 to 33), green (34 to 66), or blue (67 to 100) will
//be illuminated.

```

```

//

```

```

//Copyright, Texas Instruments, [www.TI.com]

```

```

//*****

```

```

#include "msp.h"

```

```

#include <stdio.h>

```

```

int integer_value;

```

```

void main(void)

```

```

{

```

```

WDT_A->rCTL.r = WDTPW | WDTHOLD;    //Stop watchdog timer

```

```

//Configure P2.0 as output (1) for red LED

```

```

DIO->rPADIR.b.bP2DIR |= 0x01;      //Bit 0 to logic 1

```

```

//Configure P2.1 as output (1) for green LED

```

```

DIO->rPADIR.b.bP2DIR |= 0x02;      //Bit 1 to logic 1

```

```

//Configure P2.2 as output (1) for blue LED

```

```

DIO->rPADIR.b.bP2DIR |= 0x04;      //Bit 2 to logic 1

```

86 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
while(1)
{
printf("Insert an integer between 0 and 100 and press [Enter].\n");

scanf("%d", &integer_value);    //retrieve integer value
printf("%d\n\n", integer_value); //echo integer value

if((integer_value >= 0) && (integer_value <= 33))
{
DIO->rPAOUT.b.bP2OUT |= 0x01;    //P2.0 to logic one
DIO->rPAOUT.b.bP2OUT &= 0xFD;    //P2.1 to logic zero
DIO->rPAOUT.b.bP2OUT &= 0xFB;    //P2.2 to logic zero
printf("RED\n\n");
}

else if((integer_value >= 34) && (integer_value <= 66))
{
DIO->rPAOUT.b.bP2OUT &= 0xFE;    //P2.0 to logic zero
DIO->rPAOUT.b.bP2OUT |= 0x02;    //P2.1 to logic one
DIO->rPAOUT.b.bP2OUT &= 0xFB;    //P2.2 to logic zero
printf("GREEN\n\n");
}

else if((integer_value >= 67) && (integer_value <= 100))
{
DIO->rPAOUT.b.bP2OUT &= 0xFE;    //P2.0 to logic zero
DIO->rPAOUT.b.bP2OUT &= 0xFD;    //P2.1 to logic zero
DIO->rPAOUT.b.bP2OUT |= 0x04;    //P2.2 to logic one
printf("BLUE\n\n");
}

else
{
DIO->rPAOUT.b.bP2OUT &= 0xFE;    //P2.0 to logic zero
DIO->rPAOUT.b.bP2OUT &= 0xFD;    //P2.1 to logic zero
DIO->rPAOUT.b.bP2OUT &= 0xFB;    //P2.2 to logic zero
printf("RGB off\n\n");
}
}
```

```
}
//*****
```

The **switch** statement is used when multiple if–else conditions exist. Each possible condition is specified by a case statement. When a match is found between the switch variable and a specific case entry, the statements associated with the case are executed until a **break** statement is encountered. When a case match is not found, the default case is executed.

Example 11: In this example the user is prompted for an integer between 1 and 70 evenly divisible by 10. Using a switch statement the appropriate LED combination is illuminated.

```
//*****
//switch_RGB.c
//User prompted for an integer between 1 and 70 evenly divisible by 10.
//
//Copyright, Texas Instruments, [www.TI.com]
//*****

#include "msp.h"
#include <stdio.h>

int integer_value;

void main(void)
{
    WDT_A->rCTL.r = WDTPW | WDTHOLD;    //Stop watchdog timer

    //Configure P2.0 as output (1) for red LED
    DIO->rPADIR.b.bP2DIR |= 0x01;      //Bit 0 to logic 1

    //Configure P2.1 as output (1) for green LED
    DIO->rPADIR.b.bP2DIR |= 0x02;      //Bit 1 to logic 1

    //Configure P2.2 as output (1) for blue LED
    DIO->rPADIR.b.bP2DIR |= 0x04;      //Bit 2 to logic 1

    while(1)
    {
        printf("Insert an integer between 1 and 70 and "
               "evenly divisible by 10 and press [Enter].\n");
```

88 2. A BRIEF INTRODUCTION TO PROGRAMMING

```
scanf("%d", &integer_value); //retrieve integer value
printf("%d\n\n", integer_value); //echo integer value

switch(integer_value)
{
    case 10: //RGB - 100
        DIO->rPAOUT.b.bP2OUT |= 0x01; //P2.0 to logic one
        DIO->rPAOUT.b.bP2OUT &= 0xFD; //P2.1 to logic zero
        DIO->rPAOUT.b.bP2OUT &= 0xFB; //P2.2 to logic zero
        printf("RED\n");
        break;

    case 20: //RGB - 010
        DIO->rPAOUT.b.bP2OUT &= 0xFE; //P2.0 to logic zero
        DIO->rPAOUT.b.bP2OUT |= 0x02; //P2.1 to logic one
        DIO->rPAOUT.b.bP2OUT &= 0xFB; //P2.2 to logic zero
        printf("GREEN\n");
        break;

    case 30: //RGB - 110
        DIO->rPAOUT.b.bP2OUT |= 0x01; //P2.0 to logic one
        DIO->rPAOUT.b.bP2OUT |= 0x02; //P2.1 to logic one
        DIO->rPAOUT.b.bP2OUT &= 0xFB; //P2.2 to logic zero
        printf("RED-GREEN\n");
        break;

    case 40: //RGB - 001
        DIO->rPAOUT.b.bP2OUT &= 0xFE; //P2.0 to logic zero
        DIO->rPAOUT.b.bP2OUT &= 0xFD; //P2.1 to logic zero
        DIO->rPAOUT.b.bP2OUT |= 0x04; //P2.2 to logic one
        printf("BLUE\n");
        break;

    case 50: //RGB - 101
        DIO->rPAOUT.b.bP2OUT |= 0x01; //P2.0 to logic one
        DIO->rPAOUT.b.bP2OUT &= 0xFD; //P2.1 to logic zero
        DIO->rPAOUT.b.bP2OUT |= 0x04; //P2.2 to logic one
        printf("RED-BLUE\n");
        break;
```

```

case 60: //RGB - 110
    DIO->rPAOUT.b.bP2OUT |= 0x01;    //P2.0 to logic one
    DIO->rPAOUT.b.bP2OUT |= 0x02;    //P2.1 to logic one
    DIO->rPAOUT.b.bP2OUT &= 0xFB;    //P2.2 to logic zero
    printf("RED-GREEN\n");
    break;

case 70: //RGB - 111
    DIO->rPAOUT.b.bP2OUT |= 0x01;    //P2.0 to logic one
    DIO->rPAOUT.b.bP2OUT |= 0x02;    //P2.1 to logic one
    DIO->rPAOUT.b.bP2OUT |= 0x04;    //P2.2 to logic one
    printf("RED-GREEN-BLUE\n");
    break;

default: //RGB - 000
    DIO->rPAOUT.b.bP2OUT &= 0xFE;    //P2.0 to logic zero
    DIO->rPAOUT.b.bP2OUT &= 0xFD;    //P2.1 to logic zero
    DIO->rPAOUT.b.bP2OUT &= 0xFB;    //P2.2 to logic zero
    printf("RGB off\n\n");
    break;

} //end switch
} //end while
}
//*****

```

That completes our brief overview of Energia and the C programming language.

2.11 LABORATORY EXERCISE: GETTING ACQUAINTED WITH ENERGIA AND C

Introduction. In this laboratory exercise, you will become familiar with Energia and the C programming language through a variety of programming exercises.

Procedure 1: Energia.

1. Create a counter that counts continuously from 1–100 and repeats with a 50 ms delay between counts. The onboard red LED should illuminate for odd numbers and the onboard green LED for even numbers.

2. Take the last three numbers of your identification card (e.g., driver license, student ID, etc.). Blink the red, green, and blue components of the onboard RGB at different intervals. For example, if the last three digits of your ID is 732, blink the red LED at 70 ms, the green LED at 30 ms, and the blue LED at 20 ms intervals.

Procedure 2: C.

1. Develop a program that prompts the user for two integer numbers. If the first number is less than the second, the program should count up continuously from the lower to the higher number with a 50 ms delay between counts. The onboard red LED should illuminate for odd numbers and the onboard green LED for even numbers. If the first number is higher than the lower, the program should count down continuously from the lower to the higher number with a 50 ms delay between counts. The onboard red LED should illuminate for odd numbers and the onboard green LED for even numbers.
2. Develop a program that prompts the user for an integer number. If the number is evenly divisible by 2 the red LED illuminates, evenly divisible by 3 the green LED, evenly divisible by 5 the blue LED, and evenly divisible by 7 LED1. Note: More than one LED may illuminate depending on the number provided.

2.12 SUMMARY

The goal of this chapter was to provide a tutorial on how to begin programming. We began with a discussion on the Energia Development Environment and how it may be used to develop a program for the MSP-EXP432P401R LaunchPad. For C, we used a top-down design approach. We began with the “big picture” of the program of interest followed by an overview of the major pieces of the program. We then discussed the basics of the C programming language. Only the most fundamental concepts were covered. Throughout the chapter, we provided examples and a number of excellent references.

2.13 REFERENCES AND FURTHER READING

Arduino homepage, www.arduino.cc.

Barrett, J. Closer to the Sun International,
www.closetothetunfineartndesign.com.

Barrett, S. (2010) *Embedded Systems Design with the Atmel AVR Microcontroller*, San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/S00225ED1V01Y200910DCS025](https://doi.org/10.2200/S00225ED1V01Y200910DCS025).

Barrett, S. and Pack, D. (2008) *Atmel AVR Microcontroller Primer Programming and Interfacing*, San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/S00100ED1V01Y200712DCS015](https://doi.org/10.2200/S00100ED1V01Y200712DCS015).

Barrett, S.F. and Pack, D.J. *Embedded Systems Design and Applications with the 68HC12 and HCS12*, Pearson Prentice Hall, 2005. 77, 78

Barrett, S. and Pack, D. (2006) *Microcontrollers Fundamentals for Engineers and Scientists*, San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/S00025ED1V01Y200605DCS001](https://doi.org/10.2200/S00025ED1V01Y200605DCS001).

ImageCraft Embedded Systems C Development Tools, 706 Colorado Avenue, #10-88, Palo Alto, CA, 94303, www.imagecraft.com. 80, 81

2.14 CHAPTER PROBLEMS

Fundamental

1. Describe the steps in writing a sketch and executing it on an MSP-EXP432P401R LaunchPad processing board.
2. Describe the key components of any C program.
3. Describe two different methods to program an MSP-EXP432P401R LaunchPad processing board.
4. What is an include file?
5. What are the three pieces of code required for a program function?
6. Describe how a program constant is defined in C.
7. What is the difference between a for and while loop?
8. When should a switch statement be used vs. the if-then statement construct?
9. What is the serial monitor feature used for in the Energia Development Environment?
10. Describe what variables are required and returned and the basic function of the following built-in Energia functions: Blink, Analog Input.

Advanced

1. Provide the C program statement to set PORT 1 pins 1 and 7 to logic one. Use bit-twiddling techniques.
2. Provide the C program statement to reset PORT 1 pins 1 and 7 to logic zero. Use bit-twiddling techniques.

3. Using MSP-EXP432P401R LaunchPad, write a program in Energia that takes an integer input from the user. If negative, the red LED is illuminated. If odd the green LED is illuminated or the blue LED for an even number.
4. Repeat the program above using C.

Challenging

1. Create a counter that counts continuously from 1–100 and repeats with a 50 ms delay between counts. The onboard red LED should illuminate for odd numbers and the onboard green LED for even numbers.
2. Take the last three numbers of your identification card (e.g., driver license, student ID, etc.). Blink the red, green, and blue components of the onboard RGB at different intervals. For example, if the last three digits of your ID is 732, blink the red LED at 70 ms, the green LED at 30 ms, and the blue LED at 20 ms intervals.
3. Develop a program that prompts the user for two integer numbers. If the first number is less than the second, the program should count up continuously from the lower to the higher number with a 50 ms delay between counts. The onboard red LED should illuminate for odd numbers and the onboard green LED for even numbers. If the first number is greater than the second, the program should count down continuously from the lower to the higher number with a 50 ms delay between counts. The onboard red LED should illuminate for odd numbers and the onboard green LED for even numbers.
4. Develop a program that prompts the user for an integer number. If the number is evenly divisible by 2 the red LED illuminates, evenly divisible by 3 the green LED, evenly divisible by 5 the blue LED, and evenly divisible by 7 LED¹. Note: More than one LED may illuminate depending on the number provided.

MSP432 Operating Parameters and Interfacing

Objectives: After reading this chapter, the reader should be able to:

- describe the voltage and current parameters for the Texas Instrument MSP432 microcontroller;
- apply the voltage and current parameters toward properly interfacing input and output devices to the MSP432 microcontroller;
- interface the MSP432 microcontroller operating at 3.3 VDC with a peripheral device operating at 5.0 VDC;
- interface a wide variety of input and output devices to the MSP432 microcontroller;
- describe the special concerns that must be followed when the MSP432 microcontroller is used to interface to a high power DC or AC device;
- describe how to control the speed and direction of a DC motor;
- describe how to control several types of AC loads;
- describe the peripheral components available aboard the Educational Booster Pack MkII (MkII);
- describe the peripheral components accessible via the Grove starter kit for the LaunchPad; and
- write programs using Energia to interact with the MkII and the Grove Starter Kit.

3.1 OVERVIEW

In this chapter,¹ we introduce the important concepts of the operating envelope for a microcontroller. By operating envelope, we mean the parameters and conditions for a microcontroller to

¹This chapter was adapted with permission from S. Barret and D. Pack, *Microcontroller Programming and Interfacing Texas Instruments MSP430*, San Rafael, CA, Morgan & Claypool Publishers, 2011.

function successfully, as it operates alone or interfaces with external devices. We begin by reviewing the voltage and current electrical parameters for the MSP432 microcontroller. We use this information to properly interface input and output devices to the microcontroller. The MSP432 operates at a low voltage (1.62–3.7 VDC) by design. Since the MSP-EXP432P401R LaunchPad operates at 3.3 VDC, we concentrate our discussions at this supply voltage. Although there are many compatible low voltage peripheral devices, many conventional peripheral devices operate at 5.0 VDC. In this chapter, we discuss how to interface a 3.3 VDC microcontroller to 5.0 VDC peripherals, a variety of other devices to the MSP432, and a high power DC or AC load such as a motor. Throughout the chapter, we provide a number of detailed examples to illustrate concepts.

3.2 OPERATING PARAMETERS

A microcontroller is an electronic device with precisely defined operating conditions. As long as the microcontroller is used within its defined operating parameter limits or envelope, it should continue to operate correctly. However, if the allowable conditions are violated, spurious results or damage to the processor may result.

3.2.1 MSP432 3.3 VDC OPERATION

Anytime a device is connected to a microcontroller, careful interface analysis must be performed. The MSP432 is a low operating voltage microcontroller with a supply voltage between 1.62 and 3.7 VDC. To perform the interface analysis, there are eight different electrical specifications we must consider. The electrical parameters are:

- V_{OH} : the lowest guaranteed output voltage for a logic high;
- V_{OL} : the highest guaranteed output voltage for a logic low;
- I_{OH} : the output current for a V_{OH} logic high;
- I_{OL} : the output current for a V_{OL} logic low;
- V_{IH} : the lowest input voltage guaranteed to be recognized as a logic high;
- V_{IL} : the highest input voltage guaranteed to be recognized as a logic low;
- I_{IH} : the input current for a V_{IH} logic high; and
- I_{IL} : the input current for a V_{IL} logic low.

Additionally, the MSP432 microcontroller has two different general purpose input/output drive strengths [SLAS826A]:

- full drive strength with I_{OH} and I_{OL} ratings of 10–20 mA maximum when operating at 3.0 VDC, and

- reduced drive strength with I_{OH} and I_{OL} ratings of 2–6 mA maximum when operating at 3.0 VDC.

Furthermore, it is also important to note the MSP432 has a maximum current limit for all outputs combined. It is important to realize that these parameters are static values taken under very specific operating conditions. If external circuitry is connected such that the microcontroller acts as a current source (current leaving microcontroller) or current sink (current entering microcontroller), the voltage parameters listed above will also be affected.

In the current source case, an output voltage V_{OH} is provided at the output pin of the microcontroller when the load connected to this pin draws a current of I_{OH} . If a load draws more current from the output pin than the I_{OH} specification, the value of V_{OH} is reduced. If the load current becomes too high, the value of V_{OH} falls below the value of V_{IH} for the subsequent logic circuit stage, and it will not be recognized as an acceptable logic high signal. When this situation occurs, erratic and unpredictable circuit behavior may result.

In the sink case, an output voltage V_{OL} is provided at the output pin of the microcontroller when the load connected to this pin delivers a current of I_{OL} to this logic pin. If a load delivers more current to the output pin of the microcontroller than the I_{OL} specification, the value of V_{OL} increases. If the load current becomes too high, the value of V_{OL} rises above the value of V_{IL} for the subsequent logic circuit stage, and it will not be recognized as an acceptable logic low signal. As before, when this situation occurs, erratic and unpredictable circuit behavior may result.

You must also ensure that total current limit for an entire microcontroller port and overall bulk port specifications are met. For planning purposes with the MSP432, the sum of current sourced or sunk from a port should not exceed 100 mA for the full drive strength setting or 48 mA for the reduced drive current setting. As before, if these guidelines are not complied with, erratic microcontroller behavior may result [SLAS826A, 2015]. A summary of MSP432 digital input/output parameters are shown in Figure 3.1.

3.2.2 COMPATIBLE 3.3 VDC LOGIC FAMILIES

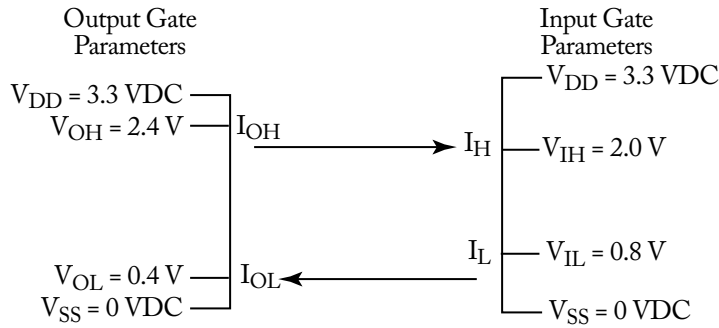
Since the MSP-EXP432P401R LaunchPad operates at 3.3 VDC, we concentrate our discussions at this supply voltage for the rest of this chapter. There are several compatible logic families that operate at 3.3 VDC. These families include the LVC, LVA, and the LVT logic families. Key parameters for the low voltage compatible families are provided in Figure 3.2. A wide range of logic devices are available within these logic families.

3.2.3 MICROCONTROLLER OPERATION AT 5.0 VDC

Many HC CMOS microcontroller families and peripherals operate at a supply voltage of 5.0 VDC. For completeness, we provide operating parameters for these type of devices. This information is essential should the MSP432 be interfaced to a 5 VDC CMOS device or peripheral.

MSP432 Digital Input/Output Parameters				
Digital Inputs (applies to both normal and high drive)				
Parameter	V _{CC}	Min	Max	Test Conditions
V _{IT+} Positive-going input threshold voltage	2.2 V	0.99 V	1.65 V	
	3.0 V	1.35 V	2.25 V	
V _{IT-} Negative-going input threshold voltage	2.2 V	0.55 V	1.21 V	
	3.0 V	0.75 V	1.65 V	
Digital Outputs (applies to normal drive)				
Parameter	V _{CC}	Min	Max	Test Conditions
V _{OH} High-level output voltage	2.2 V	1.95 V	2.2 V	I _{OHmax} = -1 mA
	2.2 V	1.60 V	2.2 V	I _{OHmax} = -3 mA
	3.0 V	2.75 V	3.0 V	I _{OHmax} = -2 mA
	3.0 V	2.40 V	3.0 V	I _{OHmax} = -6 mA
V _{OL} Low-level output voltage	2.2 V	0.00 V	0.25 V	I _{OLmax} = 1 mA
	2.2 V	0.00 V	0.60 V	I _{OLmax} = 3 mA
	3.0 V	0.00 V	0.25 V	I _{OLmax} = 2 mA
	3.0 V	0.00 V	0.60 V	I _{OLmax} = 6 mA
Digital Outputs (applies to high drive)				
Parameter	V _{CC}	Min	Max	Test Conditions
V _{OH} High-level output voltage	2.2 V	1.95 V	2.2 V	I _{OHmax} = -5 mA
	2.2 V	1.60 V	2.2 V	I _{OHmax} = -15 mA
	3.0 V	2.75 V	3.0 V	I _{OHmax} = -10 mA
	3.0 V	2.70 V	3.0 V	I _{OHmax} = -20 mA
V _{OL} Low-level output voltage	2.2 V	0.00 V	0.25 V	I _{OLmax} = 5 mA
	2.2 V	0.00 V	0.60 V	I _{OLmax} = 15 mA
	3.0 V	0.00 V	0.25 V	I _{OLmax} = 10 mA
	3.0 V	0.00 V	0.30 V	I _{OLmax} = 20 mA

Figure 3.1: MSP432 digital input/output parameters [SLAS826A, 2015].



(a) Voltage and current electrical parameters.

	LVC	LVA	LVT
V_{CC}	1.65–3.6 V	2.0–5.5 V	2.7–3.6V
tpd	5.5 ns	14 ns	3.5 ns
loc	10 μA	20 μA	190 μA

(b) LV parameters.

Figure 3.2: Low voltage compatible logic families.

Typical values for a microcontroller in the HC CMOS family, assuming $V_{DD} = 5.0$ volts and $V_{SS} = 0$ volts, are provided below. The minus sign on several of the currents indicates a current flow out of the device. A positive current indicates current flow into the device.

- $V_{OH} = 4.2$ volts,
- $V_{OL} = 0.4$ volts,
- $I_{OH} = -0.8$ milliamps,
- $I_{OL} = 1.6$ milliamps,
- $V_{IH} = 3.5$ volts,
- $V_{IL} = 1.0$ volt,
- $I_{IH} = 10$ microamps, and
- $I_{IL} = -10$ microamps.

3.2.4 INTERFACING 3.3 VDC LOGIC DEVICES WITH 5.0 VDC LOGIC FAMILIES

Although there are a wide variety of available 3.3 VDC peripheral devices available for the MSP432, you may find a need to interface the controller with 5.0 VDC devices. If bidirectional information exchange is required between the microcontroller and a peripheral device, a bidirectional level shifter should be used. The level shifter translates the 3.3 VDC signal up to 5 VDC for the peripheral device and back down to 3.3 VDC for the microcontroller. There are a wide variety of unidirectional and bidirectional level shifting devices available. Texas Instruments level shifting options include: unidirectional, bidirectional, and direction controlled level shifters. For example, the LSF0101, LSF0102, LSF0204, and LSF0108 level shifters are available in the LSF010XEVM-001 Bi-Directional Multi-Voltage Level Translator Evaluation Module (LSFEVM) (www.ti.com). Later in the chapter we show how the LSF010XEVM module is used to interface the MSP432 with a LED special effects cube.



Figure 3.3: LSF010XEVM-001 Bi-Directional Multi-Voltage Level Translator Evaluation Module (LSFEVM). Image used with permission of Texas Instruments (www.ti.com).

Example 1: Large LED Displays. Large seven-segments displays with character heights of 6.5 inches are available from SparkFun Electronics (www.sparkfun.com). Multiple display characters may be daisy chained together to form a display panel of desired character length. Only four lines from the MSP432 are required to control the display panel (ground, latch, clock, and serial data). Each character is controlled by a Large Digit Driver Board (#WIG-13279) equipped with the Texas Instrument TPIC6C596 IC Program Logic 8-bit Shifter Register. The shift register requires a 5 VDC supply and has a V_{IH} value of 4.25 VDC. The MSP432 when supplied at 3.3 VDC has a maximum V_{OH} value of 3.3 VDC. Since the output signal levels from the MSP432 are not high enough to control the TPIC6C596, a level shifter (e.g., LSF010XEVM module) is required to up convert the MSP432 signals to be compatible to the ones for the TPIC6C596 [SLIS093D].

3.3 INPUT DEVICES

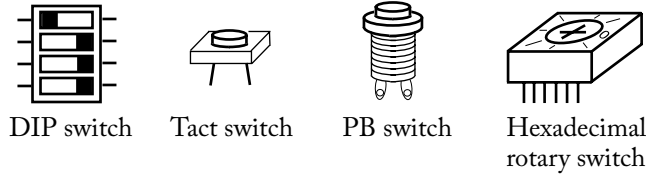
In this section, we present techniques to properly interface a variety of input devices to a microcontroller. We start with the most basic input component, a simple on/off switch.

3.3.1 SWITCHES

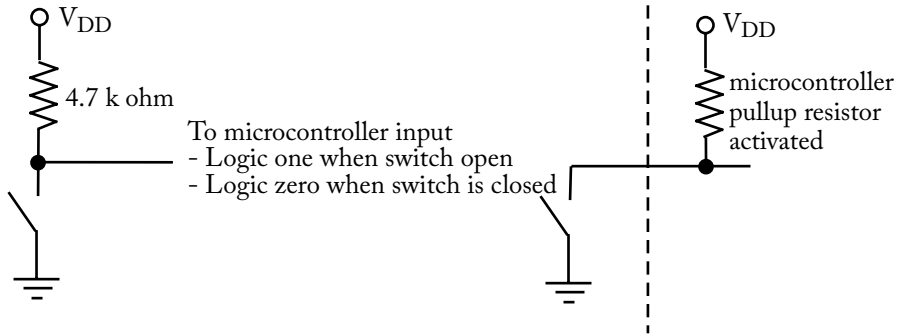
Switches come in all sizes and types. The system designer must choose the appropriate switch for a specific application. Switch varieties commonly used in microcontroller applications are illustrated in Figure 3.4a. Here is a brief summary of the different types.

- **Slide switch:** A slide switch has two different positions: on and off. The switch is manually moved to one position or the other. For microcontroller applications, slide switches are available that fit in the profile of a common integrated circuit size dual inline package (DIP). A bank of four or eight DIP switches in a single package is commonly available. Slide switches are often used to read application specific settings at system startup.
- **Momentary contact pushbutton switch:** A momentary contact pushbutton switch comes in two varieties: normally closed (NC) and normally open (NO). A normally open switch, as its name implies, does not provide an electrical connection between its contacts. When the switch is depressed, the connection between the two switch contacts is made. The connection is held as long as the switch is depressed. When the switch is released, the connection is opened. The converse is true for a normally closed switch. For microcontroller applications, pushbutton switches are available in a small tactile (tact) type switch configuration. The MSP-EXP432P401R LaunchPad is equipped with two pushbutton tactile (tact) switches designated S1 (P1.1) and S2 (P1.4).
- **Push on/push off switches:** These type of switches are also available in a normally open or normally closed configuration. For the normally open configuration, the switch is depressed to make connection between the two switch contacts. The pushbutton must be depressed again to release the connection.
- **Hexadecimal rotary switches:** Small profile rotary switches are available for microcontroller applications. These switches commonly have sixteen rotary switch positions. As the switch is rotated to each position, a unique four bit binary code is provided at the switch contacts. Hexadecimal switches are often used to read application specific settings at system startup.

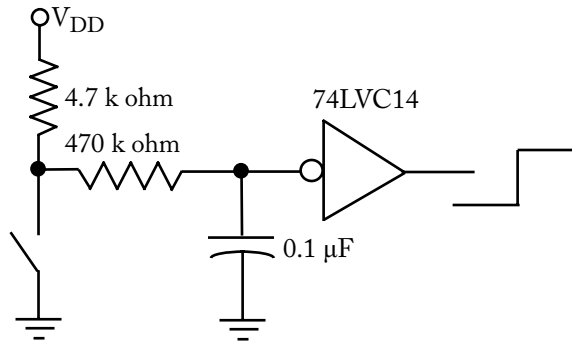
A common switch interface is shown in Figure 3.4b. This interface allows a logic one or zero to be properly introduced to a microcontroller input port pin. The basic interface consists of the switch in series with a current limiting resistor. The node between the switch and the resistor is connected to the microcontroller input pin. In the configuration shown, the resistor pulls the microcontroller input up to the supply voltage V_{DD} . When the switch is closed, the node is grounded and a logic zero is detected by the microcontroller input pin. To reverse the



(a) Switch varieties.



(b) Switch interface.



(c) Switch interface equipped with debouncing circuitry.

Figure 3.4: Switches and switch interfaces.

logic of the switch configuration, the position of the resistor and the switch is simply reversed. As discussed in Chapter 2, the MSP432 is equipped with code configurable pullup or pulldown resistors, removing the need for an external resistor, if the internal resistors are asserted.

3.3.2 SWITCH DEBOUNCING

Mechanical switches do not make a clean transition from one position (on) to another (off). When a switch is moved from one position to another, it makes and breaks contact multiple times. This activity may go on for tens of milliseconds. A microcontroller is relatively fast as compared to the action of the switch. Therefore, the microcontroller is able to recognize each switch bounce as a separate and erroneous transition.

To correct the switch bounce phenomena, additional external hardware components may be used or software techniques may be employed. A hardware debounce circuit is shown in Figure 3.4c. The node between the switch and the limiting resistor of the basic switch circuit is connected to a low pass filter (LPF), formed by the 470 kOhm resistor and the capacitor. The LPF isolates abrupt changes (bounces) in the input signal from reaching the microcontroller. The LPF is followed by a 74LVC14 Schmitt Trigger, an inverter equipped with hysteresis. Hysteresis provides different threshold points when transitioning from logic high to low and low to high. This provides a lockout window where switch transitions are not allowed. This further limits the switch bouncing.

Switches may also be debounced using software techniques. This is accomplished by inserting a 30–50 ms lockout delay in the function responding to port pin changes. The delay prevents the microcontroller from responding to the multiple switch transitions related to bouncing.

You must carefully analyze a given design to determine if hardware or software switch debouncing techniques should be used. It is important to remember that all switches exhibit bounce phenomena and therefore must be debounced.

3.3.3 KEYPADS

A keypad is an extension of the simple switch configuration. A typical keypad configuration and interface are shown in Figure 3.5. As you can see, the keypad contains multiple switches in a two-dimensional array configuration. The switches in the array share common row and column connections. The common column connections are pulled up to Vcc by external 10 K resistors or by pullup resistors within the MSP432.

To determine if a switch has been depressed, a single row of keypad switches are first asserted by the microcontroller, followed by a reading of the host keypad column inputs. If a switch has been depressed, the keypad pin corresponding to the column the switch is in will also be asserted. The combination of a row and a column assertion can be decoded to determine which key has been pressed. The keypad rows are sequentially asserted. Since the keypad is a collection of switches, debounce techniques must also be employed. In the example code provided, a 200 ms delay is provided to mitigate switch bounce. In the keypad shown, the rows are sequentially asserted active low (0).

The keypad is typically used to capture user requests to a microcontroller. A standard keypad with alphanumeric characters may be used to provide alphanumeric values to the microcontroller such as providing your personal identification number (PIN) for a financial transaction. However,

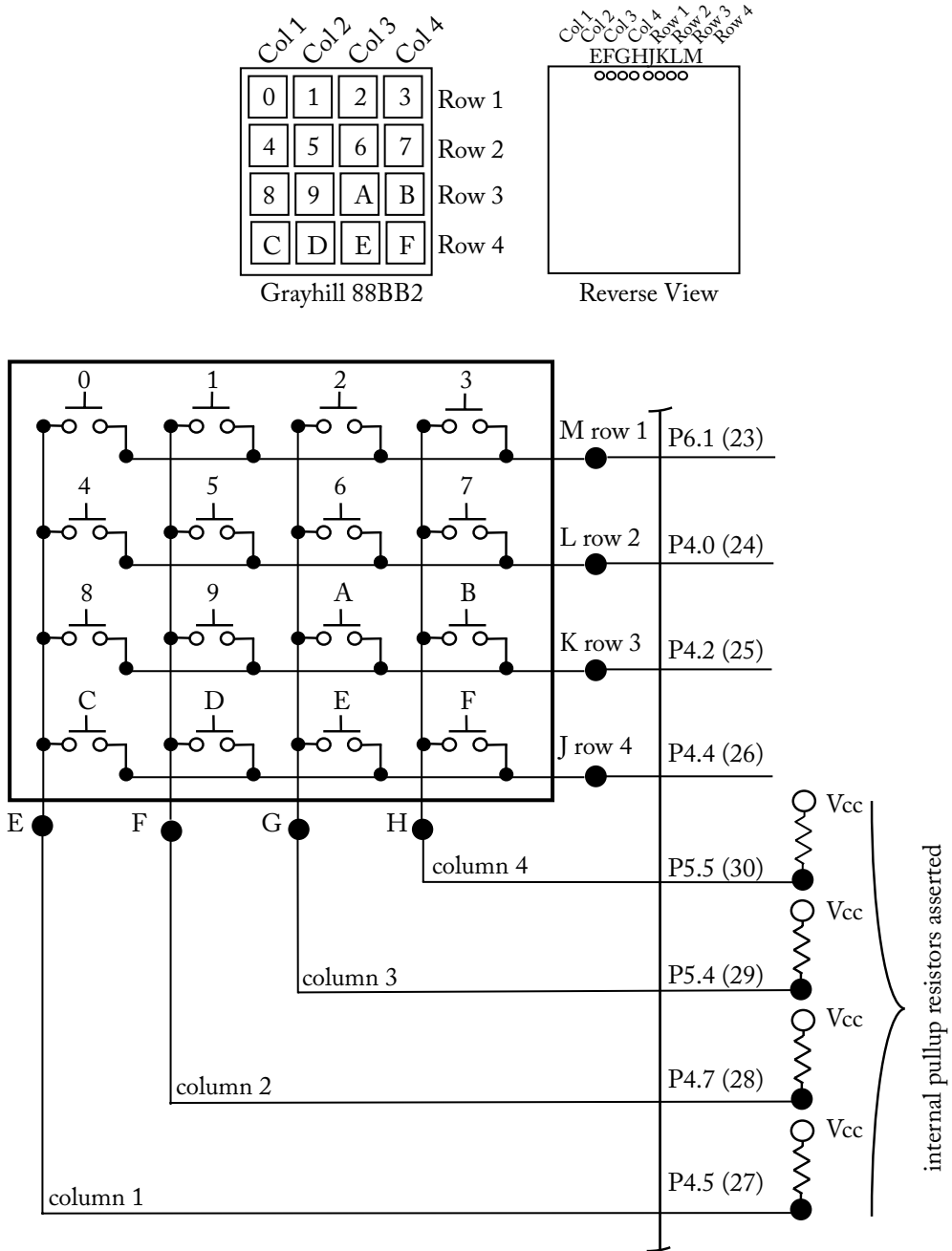


Figure 3.5: Keypad interface.

some keypads are equipped with removable switch covers such that any activity can be associated with a key press.

Example 2: Keypad. In this example a Grayhill 88BB2 4-by-4 matrix keypad is interfaced to the MSP432. The example shows how a specific switch depression can be associated with different activities by using a “switch” statement.

```
//*****
//keypad_4X4
//
//This code is in the public domain.
//*****

#define row1  23
#define row2  24
#define row3  25
#define row4  26

#define col1  27
#define col2  28
#define col3  29
#define col4  30

unsigned char  key_depressed = '*';

void setup()
{
  //start serial connection to monitor
  Serial.begin(9600);

  //configure row pins as ouput
  pinMode(row1, OUTPUT);
  pinMode(row2, OUTPUT);
  pinMode(row3, OUTPUT);
  pinMode(row4, OUTPUT);

  //configure column pins as input and assert pullup resistors
  pinMode(col1, INPUT_PULLUP);
  pinMode(col2, INPUT_PULLUP);
  pinMode(col3, INPUT_PULLUP);
  pinMode(col4, INPUT_PULLUP);
}
```

```
}

void loop()
{
  //Assert row1, deassert row 2,3,4
  digitalWrite(row1, LOW);  digitalWrite(row2, HIGH);
  digitalWrite(row3, HIGH); digitalWrite(row4, HIGH);

  //Read columns
  if (digitalRead(col1) == LOW)
    key_depressed = '0';
  else if (digitalRead(col2) == LOW)
    key_depressed = '1';
  else if (digitalRead(col3) == LOW)
    key_depressed = '2';
  else if (digitalRead(col4) == LOW)
    key_depressed = '3';
  else
    key_depressed = '*';

  if (key_depressed == '*')
  {
    //Assert row2, deassert row 1,3,4
    digitalWrite(row1, HIGH);  digitalWrite(row2, LOW);
    digitalWrite(row3, HIGH);  digitalWrite(row4, HIGH);

    //Read columns
    if (digitalRead(col1) == LOW)
      key_depressed = '4';
    else if (digitalRead(col2) == LOW)
      key_depressed = '5';
    else if (digitalRead(col3) == LOW)
      key_depressed = '6';
    else if (digitalRead(col4) == LOW)
      key_depressed = '7';
    else
      key_depressed = '*';
  }
}
```

```
}

if (key_depressed == '*')
{
//Assert row3, deassert row 1,2,4
digitalWrite(row1, HIGH);  digitalWrite(row2, HIGH);
digitalWrite(row3, LOW);  digitalWrite(row4, HIGH);

//Read columns
if (digitalRead(col1) == LOW)
    key_depressed = '8';
else if (digitalRead(col2) == LOW)
    key_depressed = '9';
else if (digitalRead(col3) == LOW)
    key_depressed = 'A';
else if (digitalRead(col4) == LOW)
    key_depressed = 'B';
else
    key_depressed = '*';
}

if (key_depressed == '*')
{
//Assert row4, deassert row 1,2,3
digitalWrite(row1, HIGH);  digitalWrite(row2, HIGH);
digitalWrite(row3, HIGH);  digitalWrite(row4, LOW);

//Read columns
if (digitalRead(col1) == LOW)
    key_depressed = 'C';
else if (digitalRead(col2) == LOW)
    key_depressed = 'D';
else if (digitalRead(col3) == LOW)
    key_depressed = 'E';
else if (digitalRead(col4) == LOW)
    key_depressed = 'F';
else
    key_depressed = '*';
}
```

```
if(key_depressed != '*')
{
  Serial.write(key_depressed);
  Serial.write(' ');

  switch(key_depressed)
  {
    case '0' :   Serial.println("Do something associated with case 0");
                 break;

    case '1' :   Serial.println("Do something associated with case 1");
                 break;

    case '2' :   Serial.println("Do something associated with case 2");
                 break;

    case '3' :   Serial.println("Do something associated with case 3");
                 break;

    case '4' :   Serial.println("Do something associated with case 4");
                 break;

    case '5' :   Serial.println("Do something associated with case 5");
                 break;

    case '6' :   Serial.println("Do something associated with case 6");
                 break;

    case '7' :   Serial.println("Do something associated with case 7");
                 break;

    case '8' :   Serial.println("Do something associated with case 8");
                 break;

    case '9' :   Serial.println("Do something associated with case 9");
                 break;

    case 'A' :   Serial.println("Do something associated with case A");
```

```

        break;

    case 'B' : Serial.println("Do something associated with case B");
              break;

    case 'C' : Serial.println("Do something associated with case C");
              break;

    case 'D' : Serial.println("Do something associated with case D");
              break;

    case 'E' : Serial.println("Do something associated with case E");
              break;

    case 'F' : Serial.println("Do something associated with case F");
              break;

    }
}
//limit switch bounce
delay(200);
}
//*****

```

3.3.4 SENSORS

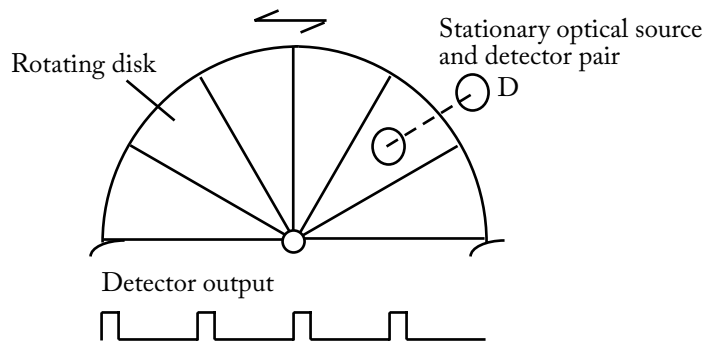
A microcontroller is typically used in control applications where data is collected, assimilated, and processed by the host algorithm, and a control decision and accompanying signals are generated by the microcontroller. Input data for the microcontroller is collected by a complement of input sensors. These sensors are either digital or analog in nature.

Digital Sensors

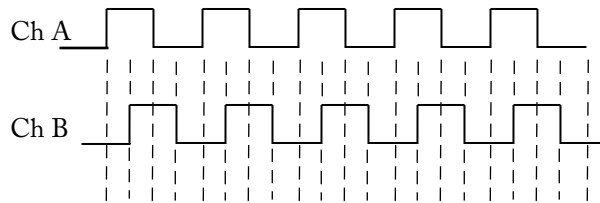
Digital sensors provide a series of digital logic pulses with sensor data encoded. The sensor data may be encoded in any of the parameters associated with the digital pulse train such as duty cycle, frequency, period, or pulse rate. The input portion of the timing system may be configured to measure these parameters.

An example of a digital sensor is the optical encoder. An optical encoder consists of a small plastic transparent disk with opaque lines etched into the disk surface. A stationary optical emitter and detector source are placed on either side of the disk. As the disk rotates, the opaque

lines break the continuity between the optical source and detector. The signal from the optical detector is monitored to determine disk rotation, as shown in Figure 3.6.



(a) Incremental tachometer encoder.



(b) Incremental quadrature encoder.

Figure 3.6: Optical encoder.

There are two major types of optical encoders: incremental encoders and absolute encoders. An absolute encoder is used when it is required to retain position information when power is lost. For example, if you were using an optical encoder in a security gate control system, an absolute encoder would be used to monitor the gate position. An incremental encoder is used in applications where a velocity or a velocity and direction information is required.

The incremental encoder types may be further subdivided into tachometers and quadrature encoders. An incremental tachometer encoder consists of a single track of etched opaque lines as shown in Figure 3.6a. It is used when the velocity of a rotating device is required. To calculate velocity, the number of detector pulses is counted in a fixed amount of time. Since the number of pulses per encoder revolution is known, velocity may be calculated.

Example 3: Optical Encoder. An optical encoder provides 200 pulses per revolution. The encoder is connected to a rotating motor shaft. If 80 pulses are counted in a 100 ms span, what is the speed of the motor in revolutions per minute (RPM)?

Answer:

$$(1\text{ rev}/200\text{ pulses}) * (80\text{ pulses}/0.100\text{ s}) * (60\text{ s}/\text{min}) = 240\text{ RPM.}$$

The quadrature encoder contains two tracks shifted in relationship to one another by 90°. This allows the calculation of both velocity and direction. To determine direction, one would monitor the phase relationship between Channel A and Channel B as shown in Figure 3.6b. The absolute encoder is equipped with multiple data tracks to determine the precise location of the encoder disk [Sick Stegmann].

Analog Sensors and Transducers

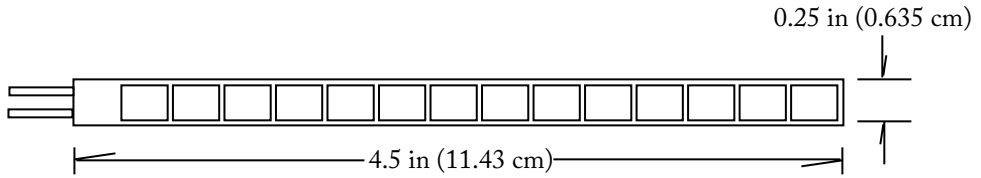
Analog sensors or transducers provide a DC voltage that is proportional to the physical parameter being measured. The analog signal may be first preprocessed by external analog hardware such that it falls within the voltage references of the conversion subsystem. In the case of the MSP432 microcontroller, the transducer output must fall between 0 and 3.3 VDC when operated at a supply voltage of 3.3 VDC. The analog voltage is then converted to a corresponding binary representation.

Example 4: Flex Sensor. An example of an analog sensor is the flex sensor shown in Figure 3.7a. The flex sensor provides a change in resistance for a change in sensor flexure. At 0° flex, the sensor provides 10 k ohms of resistance. For 90° flex, the sensor provides 30-40 k ohms of resistance. Since the microcontroller cannot measure resistance directly, the change in flex sensor resistance must be converted to a change in a DC voltage. This is accomplished using the voltage divider network shown in Figure 3.7c. For increased flex, the DC voltage will increase. The voltage can be measured using the MSP432's analog-to-digital converter (ADC) subsystem.

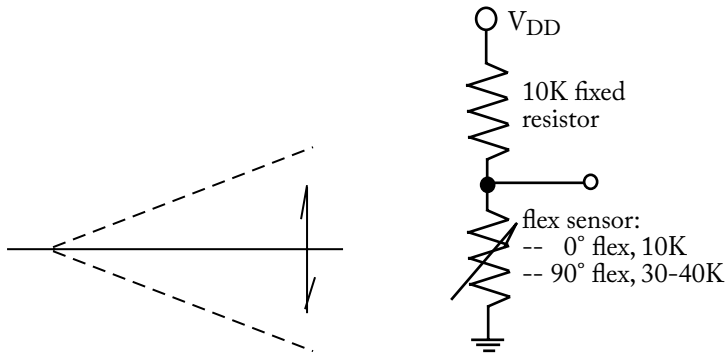
The flex sensor may be used in applications such as virtual reality data gloves, robotic sensors, biometric sensors, and in science and engineering experiments [Images Company]. One of the co-authors used the circuit provided in Figure 3.7 to help a colleague in Zoology monitor the movement of a newt salamander during a scientific experiment.

Example 5: Joystick. The thumb joystick is used to select desired direction in an X-Y plane, as shown in Figure 3.9. The thumb joystick contains two built-in potentiometers (horizontal and vertical). A reference voltage of 3.3 VDC is applied to the VCC input of the joystick. As the joystick is moved, the horizontal (HORZ) and vertical (VERT) analog output voltages will change to indicate the joystick position. The joystick is also equipped with a digital select (SEL) button.

Example 6: IR Sensor. In Chapter 2, a Sharp IR sensor is used to sense the presence of maze walls. In this example, we use the Sharp GP2Y0A21YKOF IR sensor to control the intensity of an LED. The profile of the Sharp IR sensor is provided in Figure 3.10.



(a) Flex sensor physical dimensions.



(b) Flex action.

(c) Equivalent circuit.

Figure 3.7: Flex sensor.

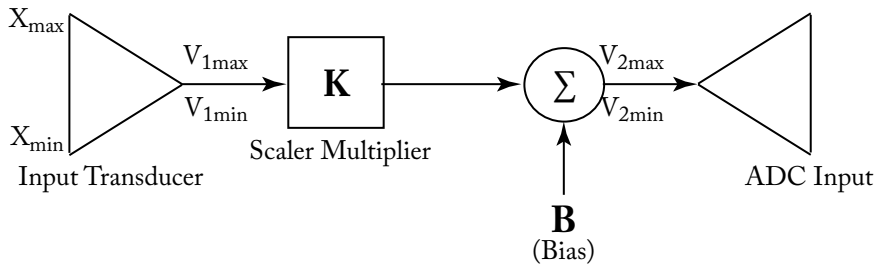
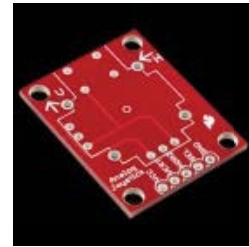
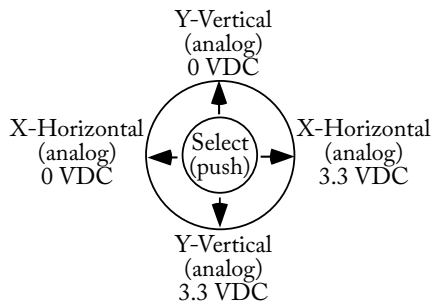
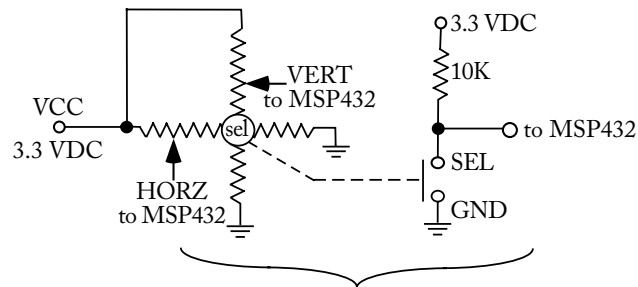


Figure 3.8: A block diagram of the signal conditioning for an ADC. The range of the sensor voltage output is mapped to the ADC input voltage range. The scalar multiplier maps the magnitudes of the two ranges and the bias voltage is used to align two limits.

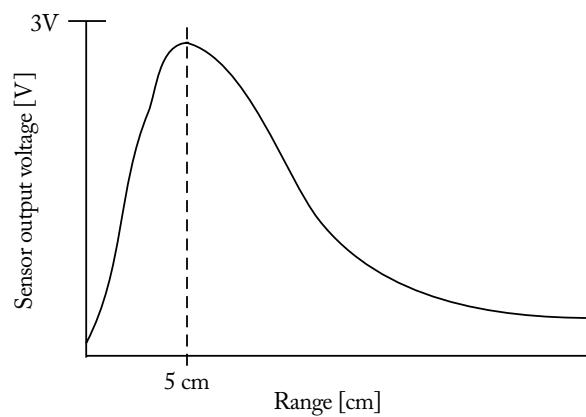


(a) Joystick operation.

(b) Sparkfun joystick (COM-09032) and breakout board (BOB-09110).



(c) Thumb joystick circuit.

Figure 3.9: Thumb joystick. Images used with permission of Sparkfun (www.sparkfun.com).**Figure 3.10:** Sharp GP2Y0A21YKOF IR sensor profile.

112 3. MSP432 OPERATING PARAMETERS AND INTERFACING

```

//*****
//IR_sensor
//
//The circuit:
// - The IR sensor signal pin is connected to analog pin 0 (30).
//   The sensor power and ground pins are connected to 5 VDC and
//   ground respectively.
// - The analog output is designated as the onboard red LED.
//
//Created: Dec 29, 2008
//Modified: Aug 30, 2011
//Author: Tom Igoe
//
//This example code is in the public domain.
//*****

const int analogInPin = 30; //Energia analog input pin A0
const int analogOutPin = 75; //Energia onboard red LED pin

int sensorValue = 0; //value read from the OR sensor
int outputValue = 0; //value output to the PWM (red LED)

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop()
{
  //read the analog in value:
  sensorValue = analogRead(analogInPin);

  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);

  // change the analog out value:
  analogWrite(analogOutPin, outputValue);
}

```

```

// print the results to the serial monitor:
Serial.print("sensor = " );
Serial.print(sensorValue);
Serial.print("\t output = ");
Serial.println(outputValue);

// wait 10 milliseconds before the next loop
// for the analog-to-digital converter to settle
// after the last reading:
delay(10);
}

//*****

```

Example 7: Ultrasonic Sensor. The ultrasonic sensor pictured in Figure 3.11 is an example of an analog based sensor. The sensor is based on the concept of ultrasound or sound waves that are at a frequency above the human range of hearing (20 Hz–20 kHz). The ultrasonic sensor pictured in Figure 3.11c emits a sound wave at 42 kHz. The sound wave reflects from a solid surface and returns back to the sensor. The amount of time for the sound wave to transit from the surface and back to the sensor may be used to determine the range from the sensor to the wall. Figure 3.11c and d show an ultrasonic sensor manufactured by Maxbotix (LV-EZ3). The sensor provides an output that is linearly related to range in three different formats: (a) a serial RS-232 compatible output at 9600 bits per second, (b) a pulse output which corresponds to 147 μ s/inch width, and (c) an analog output at a resolution of 10 mV/inch. The sensor is powered from a 2.5–5.5 VDC source (www.sparkfun.com).

Example 8: Inertial Measurement Unit. Pictured in Figure 3.12 is an inertial measurement unit (IMU) which consists of an IDG5000 dual-axis gyroscope and an ADXL335 triple axis accelerometer. This sensor may be used in unmanned aerial vehicles (UAVs), autonomous helicopters and robots. For robotic applications the robot tilt may be measured in the X and Y directions as shown in Figures 3.12c and d (www.sparkfun.com).

Example 9: Level Sensor. Milone Technologies manufacture a line of continuous fluid level sensors. The sensor resembles a ruler and provides a near liner response as shown in Figure 3.13. The sensor reports a change in resistance to indicate the distance from sensor top to the fluid surface. A wide resistance change occurs from 700 ohms at a one inch fluid level to 50 ohms at a 12.5 inch fluid level (www.milonetech.com). To covert the resistance change to a voltage change measurable by the MSP432, a voltage divider circuit as shown in Figure 3.13 may be used. With a supply voltage (V_{DD}) of 3.3 VDC, a V_{TAP} voltage of 0.855 VDC results for a 1 inch fluid level. Whereas, a fluid of 12.5 inches provides a V_{TAP} voltage level of 0.080 VDC.

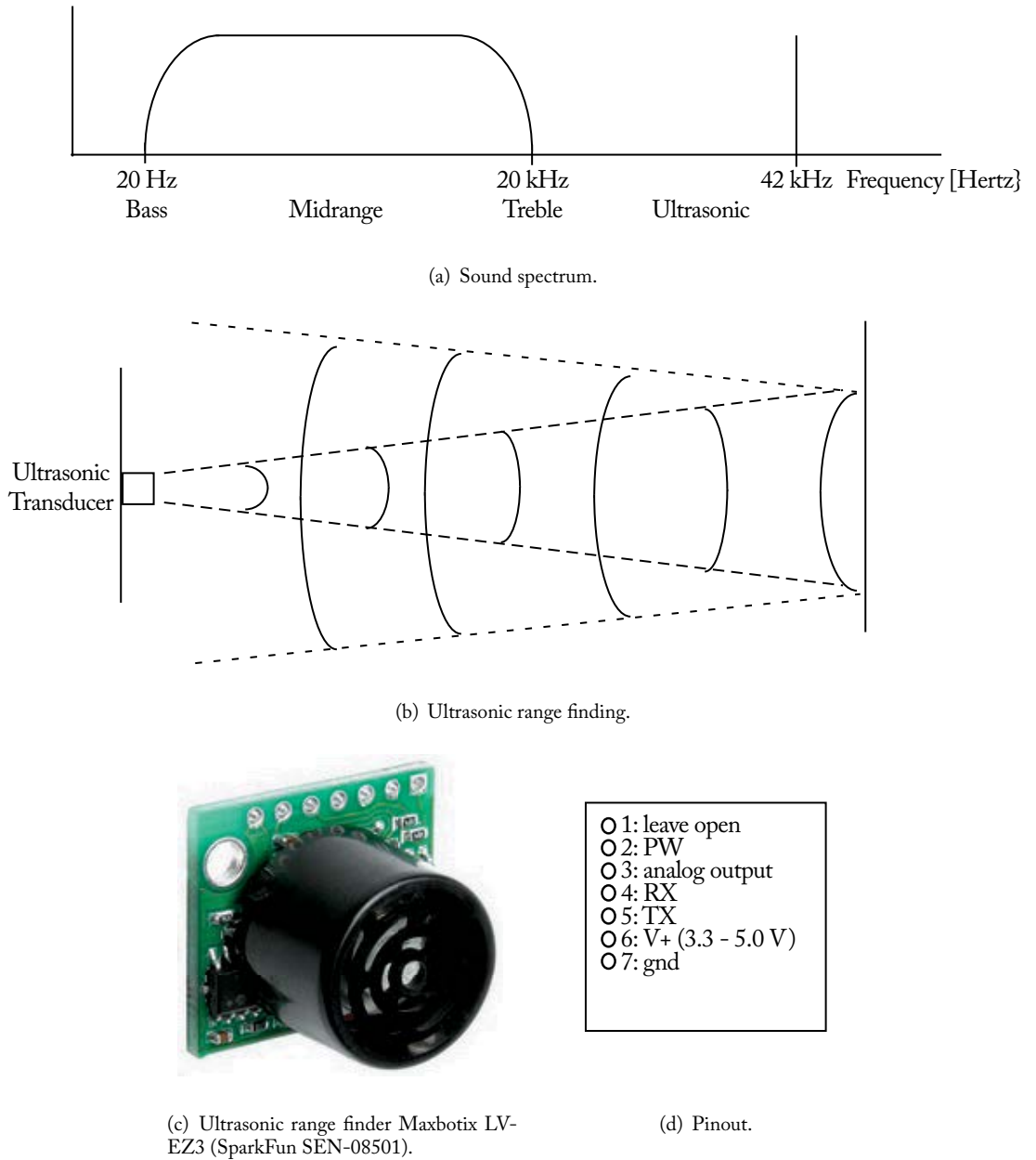
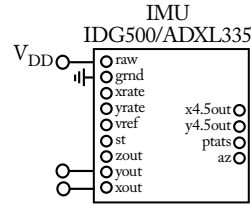


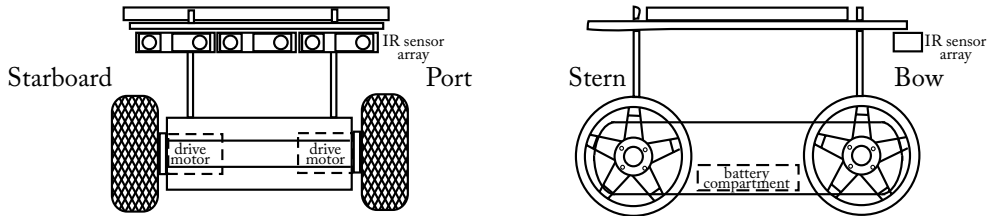
Figure 3.11: Ultrasonic sensor. (Sensor image used courtesy of SparkFun (www.sparkfun.com), Electronics.)



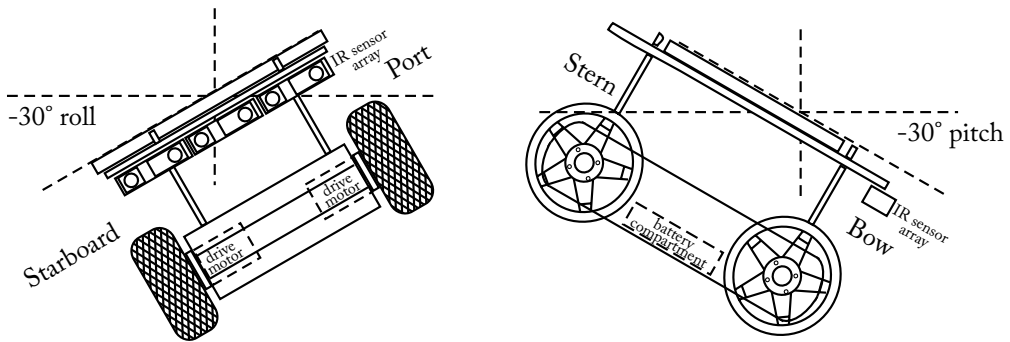
(a) SparkFun IMU Analog Combo Board 5° of Freedom IDG500/ADXL335 SEN.



(b) IDG500/ADXL335 pinout.



(c) (left) Robot front view and (right) side view.

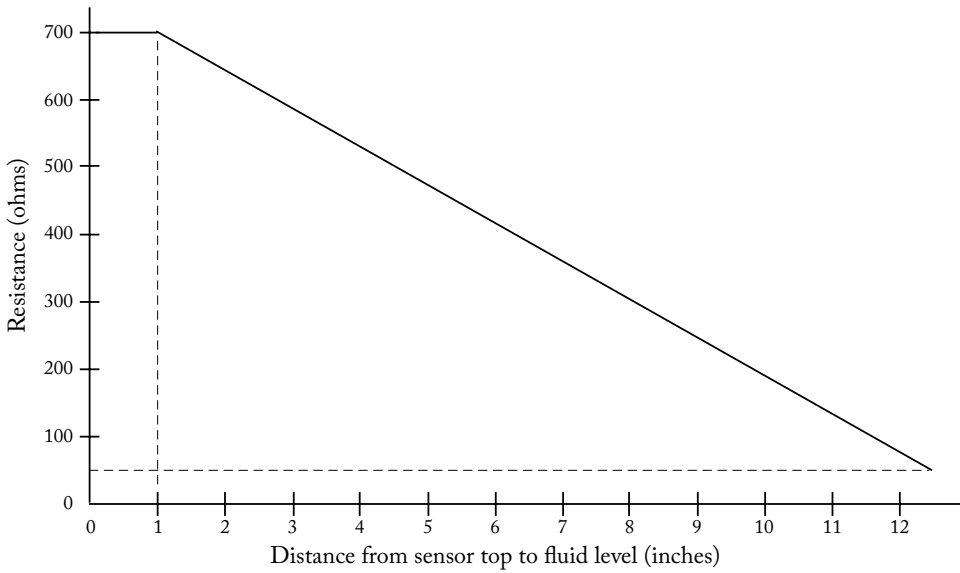


(d) (left) Roll and (right) pitch.

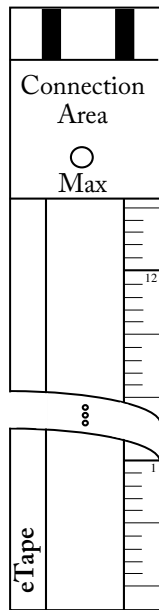
Figure 3.12: Inertial measurement unit. (IMU image used courtesy of SparkFun (www.sparkfun.com), Electronics.)

3.3.5 TRANSDUCER INTERFACE DESIGN (TID) CIRCUIT

In addition to a transducer, interface circuitry is required to match the output from the transducer to the ADC system. The objective of the transducer interface circuit is to scale and shift the

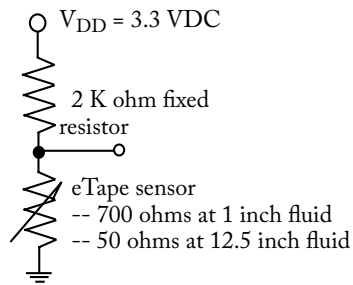


(a) Characteristics for Milone Technologies eTape™ fluid level sensor.



(b) eTape sensor.

Sensor Lead Connections



(c) Equivalent circuit.

Figure 3.13: Milone Technologies fluid level sensor. (www.milonetech.com)

transducer output signal range to the input range of ADC, which is typically 0–3.3 VDC for the MSP432. Figure 3.8 shows the transducer interface circuit using an input transducer.

The transducer interface consists of two steps: scaling and then shifting via a DC bias. The scale step allows the span of the transducer output to match the span of ADC system input range. The bias step shifts the output of the scale step to align with the input of the ADC system. In general, the scaling and bias process may be described by two equations:

$$\begin{aligned}V_{2\max} &= (V_{1\max} \times K) + B \\V_{2\min} &= (V_{1\min} \times K) + B.\end{aligned}$$

The variable $V_{1\max}$ represents the maximum output voltage from the input transducer. This voltage occurs when the maximum physical variable (X_{\max}) being measured is presented to the input transducer. This voltage must be scaled by the scalar multiplier (K) and then have a DC offset bias voltage (B) added to provide the voltage $V_{2\max}$ to the input of the ADC converter.

Similarly, the variable $V_{1\min}$ represents the minimum output voltage from the input transducer. This voltage occurs when the minimum physical variable (X_{\min}) being measured is presented to the input transducer. This voltage must be scaled by the scalar multiplier (K) and then have a DC offset bias voltage (B) added to produce voltage $V_{2\min}$, the input of the ADC converter.

Usually the values of $V_{1\max}$ and $V_{1\min}$ are provided with the documentation for the transducer. Also, the values of $V_{2\max}$ and $V_{2\min}$ are known. They are the high and low reference voltages for the ADC system (usually 3.3 VDC and 0 VDC for the MPS432 microcontroller). We thus have two equations and two unknowns to solve for K and B . The circuits to scale by K and add the offset B are usually implemented with operational amplifiers. This transducer interface design technique assumes the transducer has a linear response between $X_{1,2\max}$ and $X_{1,2\min}$.

Example 10: Transducer Interface Design with Photodiode. A photodiode is a semiconductor device that provides an output current, corresponding to the amount of light impinging on its active surface. The photodiode is used with transimpedance amplifier to convert the output current to an output voltage. A photodiode/transimpedance amplifier provides an output voltage of 0 volts for maximum rated light intensity and -2.50 VDC output voltage for the minimum rated light intensity. Calculate the required values of gain (K) and bias (B) for this light transducer to be interfaced with a microcontroller's ADC system. Assume the ADC is operating at 3.3 VDC.

$$\begin{aligned}V_{2\max} &= (V_{1\max} \times K) + B \\V_{2\min} &= (V_{1\min} \times K) + B \\3.3V &= (0V \times K) + B \\0V &= (-2.50V \times K) + B.\end{aligned}$$

The values of K and B are determined to be 1.3 and 3.3 VDC, respectively.

3.3.6 OPERATIONAL AMPLIFIERS

In the previous section, we discussed the transducer interface design (TID) process. This design process yields a required value of gain (K) and DC bias (B). Operational amplifiers (op-amps) are typically used to implement a TID interface. In this section, we briefly introduce operational amplifiers including ideal op-amp characteristics, classic op-amp circuit configurations, and an example to illustrate how to implement a TID with op-amps. Op-amps are also used in a wide variety of other applications, including analog computing, analog filter design, and a myriad of other applications. The interested reader is referred to the References section at the end of the chapter for pointers to some excellent texts on this topic.

The Ideal Operational Amplifier

A generic ideal operational amplifier is shown in Figure 3.14. An ideal operational does not exist in the real world. However, it is a good first approximation for use in developing op-amp application circuits.

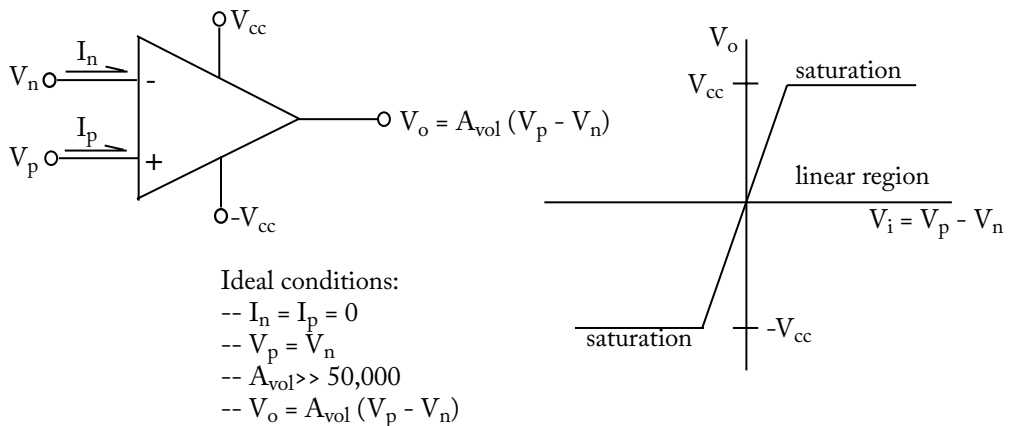
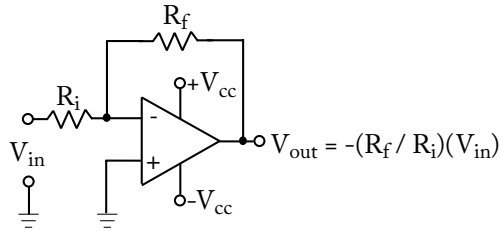


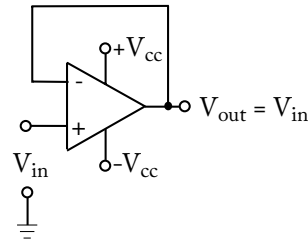
Figure 3.14: Ideal operational amplifier characteristics.

The op-amp is an active device (requires power supplies) equipped with two inputs, a single output, and several voltage source inputs. The two inputs are labeled V_p , or the non-inverting input, and V_n , the inverting input. The output of the op-amp is determined by taking the difference between V_p and V_n and multiplying the difference by the open loop gain (A_{vol}) of the op-amp, which is typically a large value much greater than 50,000.

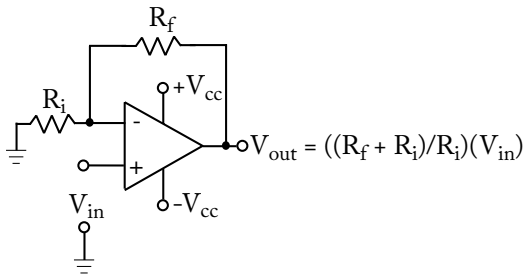
Due to the large value of A_{vol} , it does not take much of a difference between V_p and V_n before the op-amp will saturate. When an op-amp saturates, it does not damage the op-amp, but the output is limited to the range of the supply voltages ($\pm V_{cc}$). This will clip the output, and hence distort the signal, at levels less than $\pm V_{cc}$.



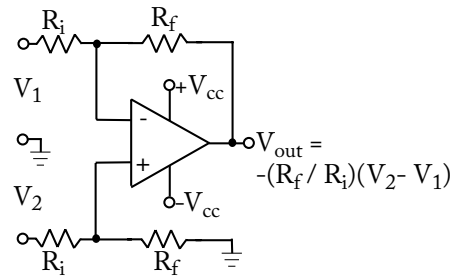
(a) Inverting amplifier.



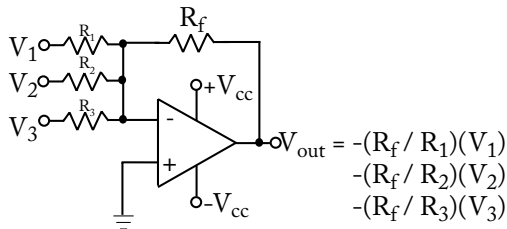
(b) Voltage follower.



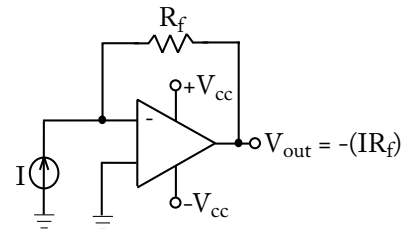
(c) Non-inverting amplifier.



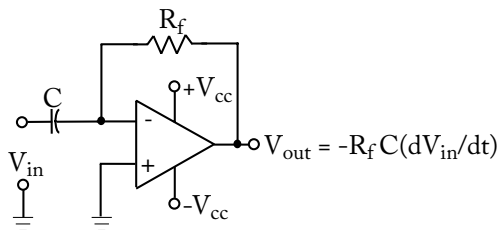
(d) Differential input amplifier.



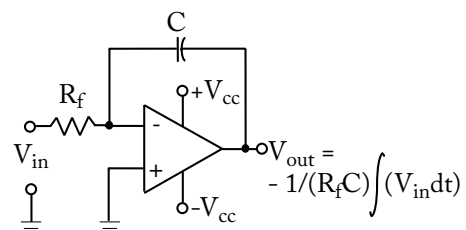
(e) Scaling adder amplifier.



(f) Transimpedance amplifier (current-to-voltage converter).



(g) Differentiator.



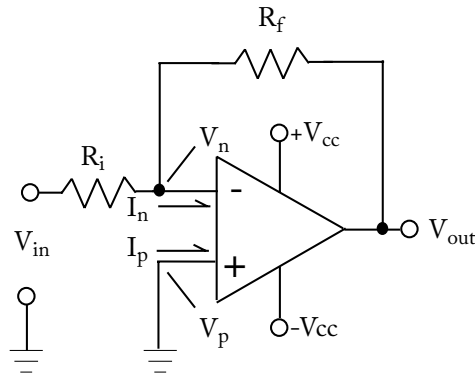
(h) Integrator.

Figure 3.15: Classic operational amplifier configurations. Adapted from [Faulkenberry, 1977].

Op-amps are typically used in a closed loop, negative feedback configuration. A sample of classic operational amplifier configurations with negative feedback are provided in Figure 3.15 [Faulkenberry, 1977]. It should be emphasized that the equations provided with each operational amplifier circuit are only valid if the circuit configurations are identical to those shown. Even a slight variation in the circuit configuration may have a dramatic effect on circuit operation. To analyze each operational amplifier circuit, use the following steps.

- Write the node equation at V_n for the circuit.
- Apply ideal op-amp characteristics to the node equation.
- Solve the node equation for V_o .

As an example, we provide the analysis of the non-inverting amplifier circuit in Figure 3.16. This same analysis technique may be applied to all of the circuits in Figure 3.15 to arrive at the equations for V_{out} provided.



Node equation at V_n

$$(V_n - V_{in}) / R_i + (V_n - V_{out}) / R_f + I_n = 0$$

Apply ideal conditions:

$$I_n = I_p = 0$$

$$V_n = V_p = 0 \text{ (since } V_p \text{ is grounded)}$$

Solve node equation for V_{out} :

$$V_{out} = - (R_f / R_i) (V_{in})$$

Figure 3.16: Operational amplifier analysis for the non-inverting amplifier. Adapted from [Faulkenberry, 1977].

Example 11: TID (continued). In the previous section, it was determined that the values of gain (K) and bias (B) were 1.3 and 3.3 VDC, respectively. The two-stage op-amp circuitry in Figure 3.17 implements these values of K and B . The first stage provides an amplification of -1.3 due to the use of the inverting amplifier configuration. In the second stage, a summing amplifier is used to add the output of the first stage with a bias of 3.3 VDC. Since this stage also introduces a minus sign to the result, the overall result of a gain of 1.3 and a bias of $+3.3$ VDC is achieved. Low-voltage operational amplifiers, operating in the 2.7–5 VDC range, are readily available from Texas Instruments.

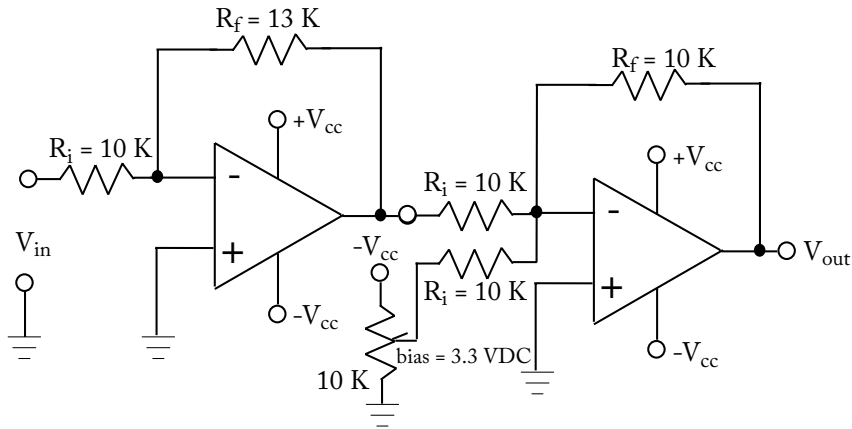


Figure 3.17: Operational amplifier implementation of the transducer interface design (TID) example circuit.

3.4 OUTPUT DEVICES

An external device should not be connected to a microcontroller without first performing careful interface analysis to ensure the voltage, current, and timing requirements of the microcontroller and the external device are met. In this section, we describe interface considerations for a wide variety of external devices. We begin with the interface for a single light-emitting diode.

3.4.1 LIGHT EMITTING DIODES (LEDS)

An LED is typically used as a logic indicator to inform the presence of a logic one or a logic zero at a microcontroller pin. An LED has two leads: the anode or positive lead and the cathode or negative lead. To properly bias an LED, the anode lead must be biased at a level approximately 1.7–2.2 volts higher than the cathode lead. This specification is known as the forward voltage (V_f) of the LED.

The LED current must also be limited to a safe current level known as the forward current (I_f). The diode voltage and current specifications are usually provided by the manufacturer.

Examples of various LED biasing circuits are provided in Figure 3.18. In Figure 3.18a, a logic one provided by the microcontroller provides the voltage to forward bias the LED. The microcontroller also acts as the source for the forward current through the LED. To properly bias the LED, the value of the limit resistor (R) is chosen. Also, we must insure the microcontroller is capable of supplying the voltage and current to the LED.

Example 12: LED Interface. A red (635 nm) LED is rated at 1.8 VDC with a forward operating current of 10 mA. Design a proper bias for the LED using the configuration of Figure 3.18a.

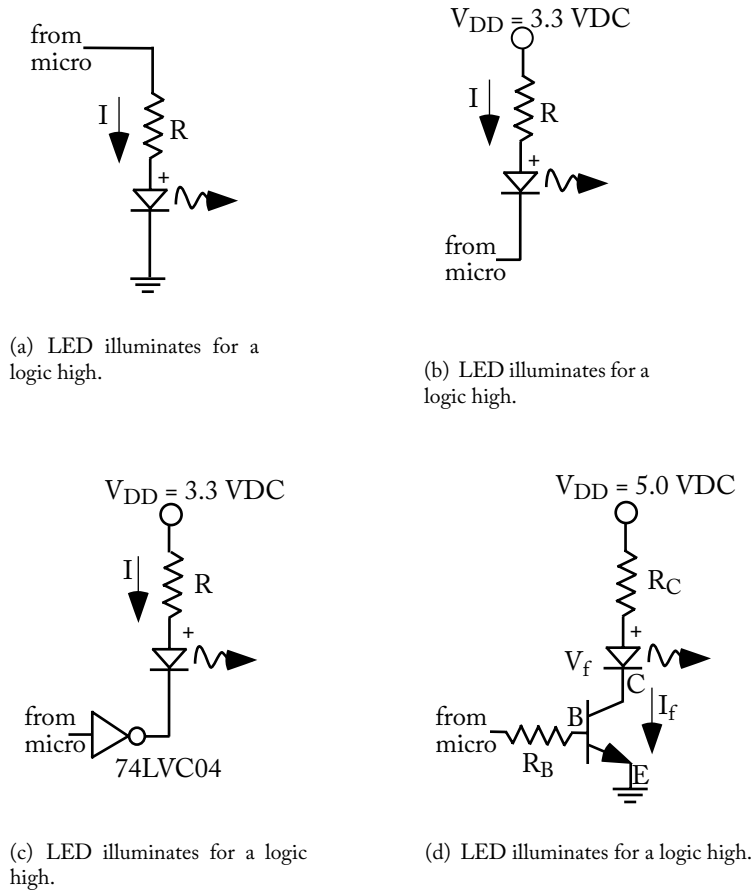


Figure 3.18: Interfacing an LED.

Answer: In the configuration of Figure 3.18a, the MSP432 microcontroller pin can be viewed as an unregulated power supply. That is, the pin's output voltage is determined by the current supplied by the pin. The current flows out of the microcontroller pin through the LED and resistor combination to ground (current source). In this example, we use the full drive strength characteristics described earlier in the chapter for the high level output voltage. When supplying 10 mA in the logic high case, let's assume the high level output voltage drops to approximately 2.75 VDC. The value of R may be calculated using Ohm's Law. The voltage drop across the resistor is the difference between the 2.75 VDC supplied by the microcontroller pin and the LED forward voltage of 1.8 VDC. The current flowing through the resistor is the LED's forward current (10 mA). This renders a resistance value of 95 ohms. A common resistor value close to 95 ohms is 100 ohms.

For the LED interface shown in Figure 3.18b, the LED is illuminated when the microcontroller provides a logic low. In this case, the current flows from the power supply back into the microcontroller pin (current sink). As before, the MSP microcontroller full drive strength parameters provided earlier in the chapter must be used.

If LEDs with higher forward voltages and currents are used, alternative interface circuits may be employed. Figures 3.18c and 3.18d provide two more LED interface circuits. In Figure 3.18c, a logic one is provided by the microcontroller to the input of the inverter. The inverter produces a logic zero at its output, which provides a virtual ground at the cathode of the LED. Therefore, the proper voltage biasing for the LED is provided. The resistor (R) limits the current through the LED. A proper resistor value can be calculated using $R = (V_{DD} - V_{DIODE})/I_{DIODE}$. It is important to note that the inverter used must have sufficient current sink capability (I_{OL}) to safely handle the forward current requirements of the LED.

An NPN transistor such as a 2N2222 (PN2222 or MPQ2222) may also be used in place of the inverter, as shown in Figure 3.18d. In this configuration, the transistor is used as a switch. When a logic low is provided by the microcontroller, the transistor is in the cutoff region. When a logic one is provided by the microcontroller, the transistor is driven into the saturation region. To properly interface the microcontroller to the LED, resistor values R_B and R_C must be chosen. The resistor R_B is chosen to limit the base current. The following example shows how an LED is interfaced with a microcontroller using an NPN transistor.

Example 13: LED Interface. Using the interface configuration of Figure 3.18d, design an interface for an LED with V_f of 2.2 VDC and I_f of 20 mA.

In this example, we can use the reduced drive strength of the MSP432 discussed earlier in the chapter. If we choose an I_{OH} value of 2 mA, the V_{OH} value will be approximately 3.0 VDC. A loop equation, which includes these parameters, may be written as:

$$V_{OH} = (I_B \times R_B) + V_{BE}.$$

The transistor V_{BE} is typically 0.7 VDC. Therefore, all equation parameters are known except R_B . Solving for R_B yields a value of 1.15 K ohm.

In this interface configuration, resistor R_C is chosen as in previous examples to safely limit the forward LED current to prescribed values. A loop equation may be written that includes R_C :

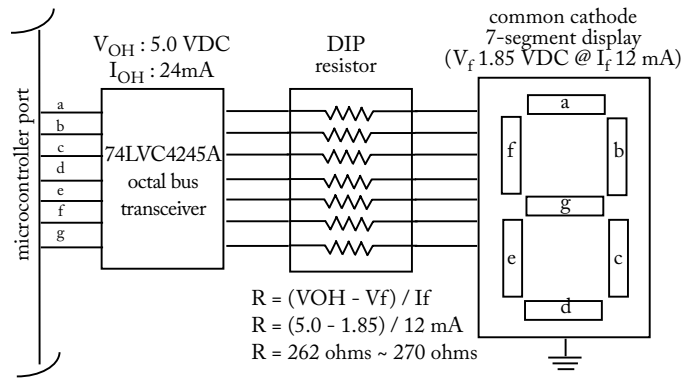
$$V_{CC} - (I_f \times R_C) - V_f - V_{CE(sat)} = 0.$$

A typical value for $V_{CE(sat)}$ is 0.2 VDC. All equation values are known except R_C . The equation may be solved rendering an R_C value of 130 ohms.

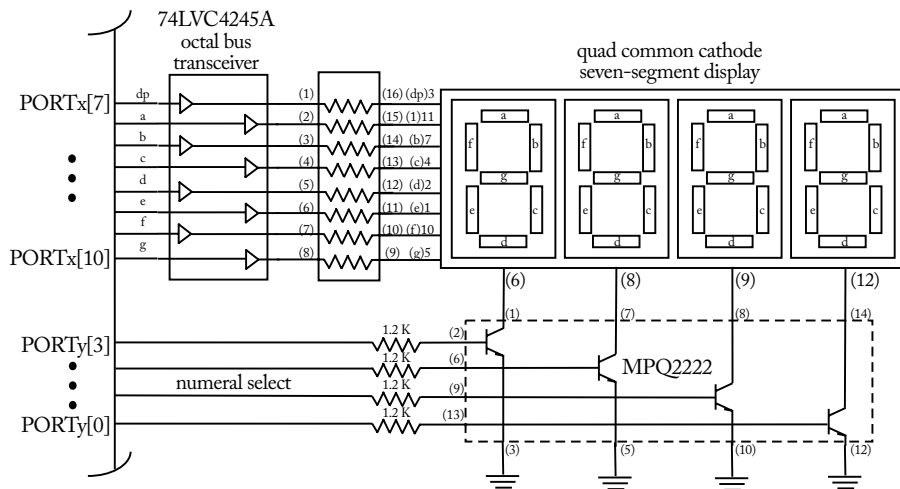
3.4.2 SEVEN SEGMENT LED DISPLAYS

To display numeric data, seven segment LED displays are available as shown in Figure 3.19b. Different numerals are displayed by asserting the proper LED segments. For example, to display

the number five, segments a, c, d, f, and g need to be illuminated. See Figure 3.19a. Seven segment displays are available in common cathode (CC) and common anode (CA) configurations. As the CC designation implies, all seven individual LED cathodes on the display are tied together.



(a) Seven-segment display interface.



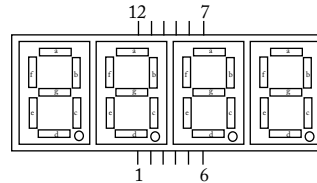
(b) Quad seven-segment display interface.

Figure 3.19: LED display devices. (*Continues.*)

As shown in Figure 3.19b, an interface circuit is required between the microcontroller and the seven segment LED. We use a 74LVC4245A octal bus transceiver circuit to translate the 3.3 VDC output from the microcontroller up to 5 VDC and also provide a maximum I_{OH} value of 24 mA. A limiting resistor is required for each segment to limit the current to a safe value for

Numeral	dp PORT x [7]	a PORT x [6]	b PORT x [5]	c PORT x [4]	d PORT x [3]	e PORT x [2]	f PORT x [1]	g PORT x [0]	hex rep
0	1	1	1	1	1	1	1	0	0 x 7E
1	0	1	1	1	0	0	0	0	0 x 30
2	1	1	0	1	1	0	0	1	0 x 6D
3	1	1	1	1	1	0	0	1	0 x 79
4	0	1	1	0	0	1	1	1	0 x 33
5	1	0	1	1	0	1	1	1	0 x 5D
6	0	0	1	1	1	1	1	1	0 x 1F
7	1	1	1	0	0	0	0	0	0 x 70
8	1	1	1	1	1	1	1	1	0 x 7F
9	1	1	1	1	0	0	1	1	0 x 73

(c) Numeral to segment conversion.



(d) Quad seven-segment display pinout UN(M)5624-11 EWRS.

Figure 3.19: (Continued.) LED display devices.

the LED. Conveniently, resistors are available in dual in-line (DIP) packages of eight for this type of application. Alternatively, a Texas Instrument LSF0101XEVM-001 discussed earlier in the chapter may be used for level shifting.

Seven-segment displays are available in multi-character panels. In this case, separate microcontroller ports are not used to provide data to each seven segment character. Instead, a single port is used to provide character data. A portion of another port is used to sequence through each of the characters as shown in Figure 3.19b. An NPN (for a CC display) transistor is connected to the common cathode connection of each individual character. As the base contact of each transistor is sequentially asserted, the specific character is illuminated. If the microcontroller sequences through the display characters at a rate greater than 30 Hz, the display will have steady illumination. Alternatively, specially equipped LED displays can be daisy changed together and controlled by common data, latch and clock lines as discussed earlier in the chapter. We investigate seven segment displays later in the chapter with the Grove Starter Kit for the Launch Pad.

3.4.3 TRI-STATE LED INDICATOR

A tri-state LED indicator is shown in Figure 3.20. It is used to provide the status of an entire microcontroller port. The indicator bank consists of eight green and eight red LEDs. When an individual port pin is logic high the green LED is illuminated. When logic low, the red LED is illuminated. If the port pin is at a tri-state, high impedance state, the LED is not illuminated. Tri-state logic is used to connect a number of devices to a common bus. When a digital circuit is placed in the Hi-z (high impedance) state it is electrically isolated from the bus.

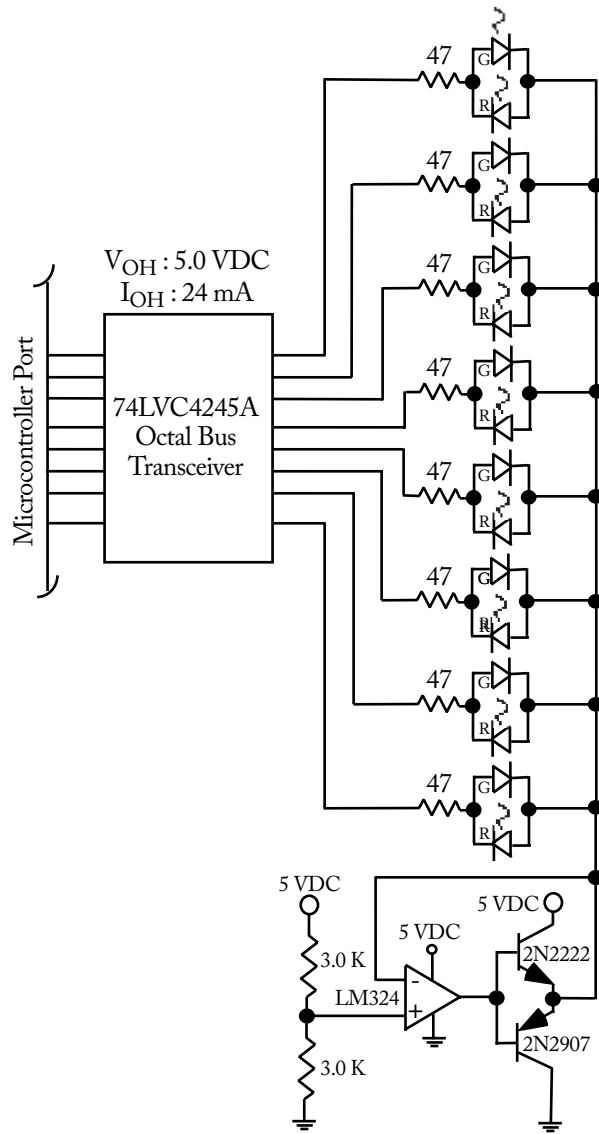


Figure 3.20: Tri-state LED display.

The NPN/PNP transistor pair at the bottom of the figure provides a 2.5 VDC voltage reference for the LEDs. When a specific port pin is logic high, the green LED will be forward biased, since its anode will be at a higher potential than its cathode. The 47 ohm resistor limits current to a safe value for the LED. Conversely, when a specific port pin is at a logic low (0 VDC), the red LED will be forward biased and illuminate. For clarity, the red and green LEDs are shown as being separate devices. LEDs are available that have both LEDs in the same device. The 74LVC4245A octal bus transceiver translates the output voltage of the microcontroller from 3.3 VDC to 5.0 VDC. Alternatively, a Texas Instrument LSF0101XEVM-001 discussed earlier in the chapter may be used for level shifting.

3.4.4 DOT MATRIX DISPLAY

The dot matrix display consists of a large number of LEDs configured in a single package. A typical 5×7 LED arrangement is a matrix of five columns of LEDs with seven LED rows as shown in Figure 3.21. Display data for a single matrix column [R6-R0] is provided by the microcontroller. That specific row is then asserted by the microcontroller using the column select lines [C2-C0]. The entire display is sequentially built up a column at a time. If the microcontroller sequences through each column fast enough (greater than 30 Hz), the matrix display appears to be stationary to a viewer.

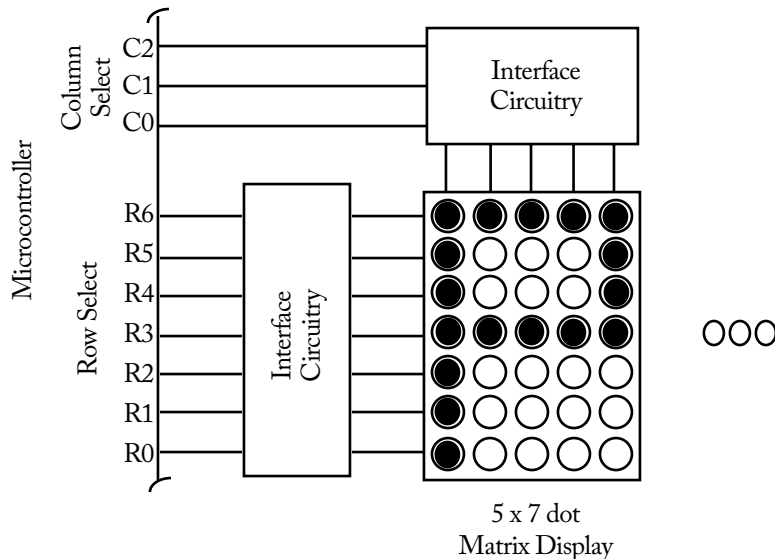


Figure 3.21: Dot matrix display.


```

//This example code is in the public domain.
//*****

#include "driverlib.h"

//function prototypes
void illuminate_LED(int row, int column);

void main(void)
{
// Stop watchdog timer
WDT_A_hold(WDT_A_BASE);

//Configure GPIO pins as output
//Column pins
GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN7); //column 1
GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN5); //column 2
GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN1); //column 3
GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN3); //column 4
GPIO_setAsOutputPin(GPIO_PORT_P6, GPIO_PIN7); //column 5

//Row pins
GPIO_setAsOutputPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7

//Configure GPIO pins for high drive strength
//Column pins
GPIO_setDriveStrengthHigh(GPIO_PORT_P3, GPIO_PIN7); //column 1
GPIO_setDriveStrengthHigh(GPIO_PORT_P3, GPIO_PIN5); //column 2
GPIO_setDriveStrengthHigh(GPIO_PORT_P5, GPIO_PIN1); //column 3
GPIO_setDriveStrengthHigh(GPIO_PORT_P2, GPIO_PIN3); //column 4
GPIO_setDriveStrengthHigh(GPIO_PORT_P6, GPIO_PIN7); //column 5

```

```

//Row pins
GPIO_setDriveStrengthHigh(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setDriveStrengthHigh(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setDriveStrengthHigh(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setDriveStrengthHigh(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setDriveStrengthHigh(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setDriveStrengthHigh(GPIO_PORT_P3, GPIO_PIN6); //Row 6
GPIO_setDriveStrengthHigh(GPIO_PORT_P5, GPIO_PIN2); //Row 7

while(1)
{
//specify row (1-7), column (1-5)
illuminate_LED(1, 2);
illuminate_LED(2, 2);
illuminate_LED(3, 2);
illuminate_LED(4, 2);
illuminate_LED(5, 2);
illuminate_LED(6, 2);
illuminate_LED(7, 2);
__delay_cycles(500000); //delay at 48 MHz, each delay count=0.2 us
}
}
//*****

void illuminate_LED(int row, int column)
{
switch(row) //select row R1 through R7
{
case 1 : //row 1: logic high row 2-7: logic low
GPIO_setOutputHighOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6 6
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
break;

case 2 : //row 2: logic high row 1, 3-7: logic low

```

```
GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
break;

case 3 : //row 3: logic high row 1,2, 4-7: logic low
GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
break;

case 4 : //row 4: logic high row 1-3, 5-7: logic low
GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
break;

case 5 : //row 5: logic high row 1-4, 6,7: logic low
GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
break;
```


132 3. MSP432 OPERATING PARAMETERS AND INTERFACING

```
case 6 : //row 6: logic high row 1-5, 7: logic low
        GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
        GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
        GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
        break;

case 7 : //row 7: logic high row 1-6: logic low
        GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN6); //Row 1
        GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN6); //Row 2
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN4); //Row 3
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //Row 4
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //Row 5
        GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //Row 6
        GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN2); //Row 7
        break;

default: break;
}

switch(column)
{
case 1 : //col 1: logic low col 2-5: logic high
        GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); //column 1
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); //column 2
        GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN1); //column 3
        GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN3); //column 4
        GPIO_setOutputHighOnPin(GPIO_PORT_P6, GPIO_PIN7); //column 5
        break;

case 2 : //col 2: logic low col 1, 3-5: logic high
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); //column 1
        GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN5); //column 2
        GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN1); //column 3
        GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN3); //column 4
        GPIO_setOutputHighOnPin(GPIO_PORT_P6, GPIO_PIN7); //column 5
```

```

        break;

    case 3 : //col 3: logic low  col 1-2, 4-5: logic high
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); //column 1
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); //column 2
        GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN1); //column 3
        GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN3); //column 4
        GPIO_setOutputHighOnPin(GPIO_PORT_P6, GPIO_PIN7); //column 5
        break;

    case 4 : //col 4: logic low  col 1-3, 5: logic high
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); //column 1
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); //column 2
        GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN1); //column 3
        GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN3); //column 4
        GPIO_setOutputHighOnPin(GPIO_PORT_P6, GPIO_PIN7); //column 5
        break;

    case 5 : //col 5: logic low  col 1-4: logic high
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); //column 1
        GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN5); //column 2
        GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN1); //column 3
        GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN3); //column 4
        GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN7); //column 5
        break;

    default: break;
}

__delay_cycles(500000); //delay at 48 MHz, each delay count=0.2 us
}

//*****

```

3.4.5 LIQUID CRYSTAL DISPLAY (LCD)

An LCD is an output device to display text information, as shown in Figure 3.23. LCDs come in a wide variety of configurations including multi-character, multi-line format. A 16×2 LCD format is common. That is, it has the capability of displaying 2 lines of 16 characters each. The characters are sent to the LCD via American Standard Code for Information Interchange (ASCII) format

a single character at a time. For a parallel configured LCD, an eight bit data path and two lines are required between the microcontroller and the LCD, as shown in Figure 3.23a. Many parallel configured LCDs may also be configured for a four bit data path thus saving several precious microcontroller pins. A small microcontroller mounted to the back panel of the LCD translates the ASCII data characters and control signals to properly display the characters. Several manufacturers provide 3.3 VDC compatible displays.

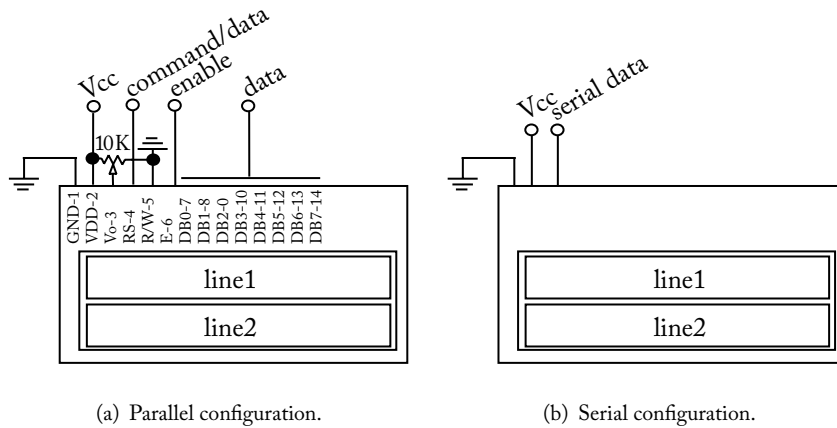


Figure 3.23: LCD display with (a) parallel interface and (b) serial interface.

To conserve precious, limited microcontroller input/output pins a serial configured LCD may be used. A serial LCD reduces the number of required microcontroller pins for interface, from ten down to one, as shown in Figure 3.23b. Display data and control information is sent to the LCD via an asynchronous UART serial communication link (8 bit, 1 stop bit, no parity, 9600 Baud). A serial configured LCD costs slightly more than a similarly configured parallel LCD. Additional information on using the serial LCD is provided with the 4 W robot example in Chapter 11.

Example 15: LCD. In this example a Sparkfun LCD-09067, 3.3 VDC, serial, 16 by 2 character, black on white LCD display is connected to the MSP432. Communication between the MSP432 and the LCD is accomplished by a single 9600 bits per second (BAUD) connection using the onboard Universal Asynchronous Receiver Transmitter (UART). The UART is configured for 8 bits, no parity, and one stop bit (8-N-1). The MSP-EXP432P401R LaunchPad is equipped with two UART channels. One is the back channel UART connection to the PC. The other is accessible by pin 3 (RX, P3.2) and pin 4 (TX, P3.3). Provided below is the sample Energia code to print a test message to the LCD. Note the UART is designated “Serial1” in the program. The back channel UART for the Energia serial monitor display is designated “Serial.”

```
//*****
```

```

//Serial_LCD_energia
//Serial 1 accessible at:
// - RX: P3.2, pin 3
// - TX: P3.3, pin 4
//
//This example code is in the public domain.
//*****

void setup()
{
  //Initialize serial channel 1 to 9600 BAUD and wait for port to open
  Serial1.begin(9600);
}

void loop()
{
  Serial1.print("Hello World");
  delay(500);
  Serial1.println("...Hello World");
  delay(500);
}

//*****

```

3.5 HIGH POWER DC INTERFACES

There are a wide variety of DC motor types that may be controlled by a microcontroller. To properly interface a motor to the microcontroller, we must be familiar with the different types of motor technologies. Motor types are illustrated in Figure 3.24.

General categories of DC motor types include the following:

- **DC motor:** A DC motor has a positive and a negative terminal. When a DC power supply of suitable current rating is applied to the motor, it will rotate. If the polarity of the supply is switched with reference to the motor terminals, the motor will rotate in the opposite direction. The speed of the motor is roughly proportional to the applied voltage up to the rated voltage of the motor.
- **Servo motor:** A servo motor provides a precision angular rotation for an applied pulse width modulation duty cycle. As the duty cycle of the applied signal is varied, the angular displacement of the motor also varies. This type of motor is used to change mechanical positions such as the steering angle of a wheel.

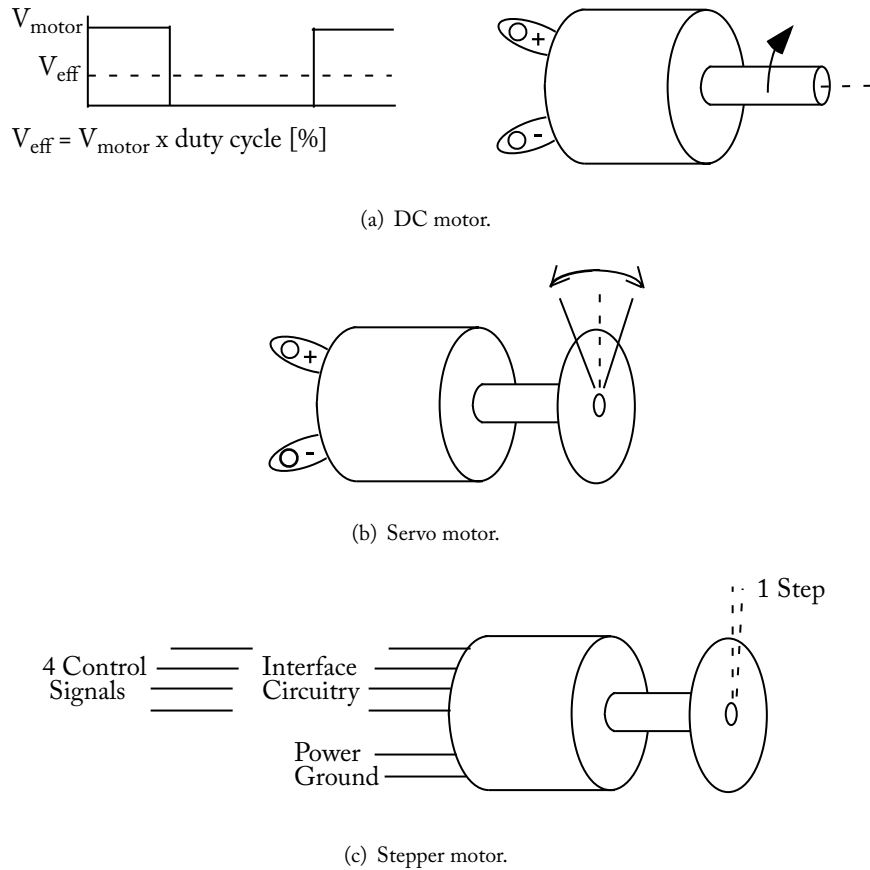


Figure 3.24: Motor types.

- Stepper motor:** A stepper motor, as its name implies, provides an incremental step change in rotation (typically 2.5° per step) for a step change in control signal sequence. The motor is typically controlled by a two- or four-wire interface. For the four-wire stepper motor, the microcontroller provides a four bit control sequence to rotate the motor clockwise. To turn the motor counterclockwise, the control sequence is reversed. The low power control signals are interfaced to the motor via MOSFETs or power transistors to provide for the proper voltage and current requirements of the pulse sequence.
- Linear actuator:** A linear actuator translates the rotation motion of a motor to linear forward and reverse movement. The actuators are used in a number of different applications where precisely controlled linear motion is required. The control software and interface for linear actuators are very similar to DC motors.

Example 16: DC Motor Interface. A general purpose DC motor interface is provided in Figure 3.25. This interface allows the low-voltage (3.3 VDC), low-current control signal to be interfaced to a higher voltage, higher current motor. This interface provides for unidirectional control. To control motor speed, pulse width modulation (PWM) techniques may be used. The control signal from the MSP432 is fed to the TIP 120 NPN Darlington transistor. The Darlington configuration allows high current gain to drive the motor. Diodes are placed in series with the motor to reduce the motor supply voltage to the required motor voltage. Each diode provides a drop of approximately 0.7 VDC. A reverse biased diode is placed across the motor and diode string to allow a safe path for reverse current. This configuration may be adjusted for many types of DC motors by appropriately adjusting supply voltage, number of series diodes, and the value of the base resistance.

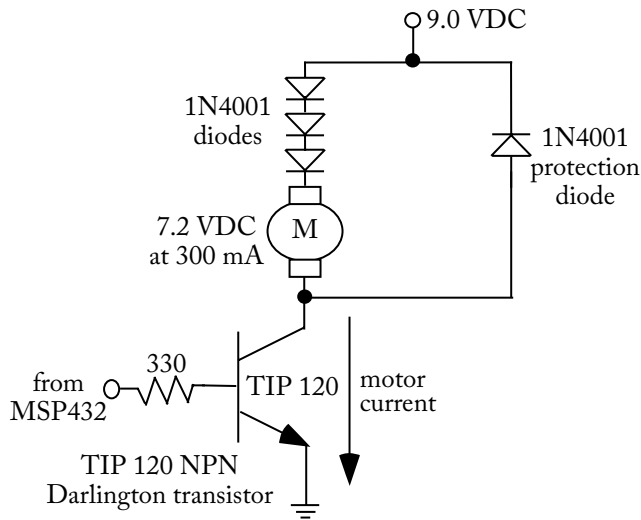


Figure 3.25: General purpose motor interface.

Example 17: Inexpensive Laser Light Show. An inexpensive laser light show can be constructed using two servos. This application originally appeared in the third edition of “Arduino Microcontroller Processing for Everyone!” The example has been adapted with permission for compatibility with the MSP432 [Barrett, 2013]. In this example we use two Futaba 180° range servos (Parallax 900-00005, available from Jameco #283021) mounted as shown in Figure 3.26. The servos operate from 4–6 VDC. The servos expect a pulse every 20 ms (50 Hz). The pulse length determines the degree of rotation from 1000 ms (5% duty cycle, -90° rotation) to 2000 ms (10% duty cycle, $+90^\circ$ rotation). The X and Y control signals are provided by the MSP432. The X and Y control signals are interfaced to the servos via LM324 operational amplifiers. The 3.3 VDC control signals from the MSP432 are up converted to 5.0 VDC by the op-amps. The op-amps

serve as voltage comparators with a 2.5 VDC threshold. The laser source is provided by an inexpensive laser pointer.

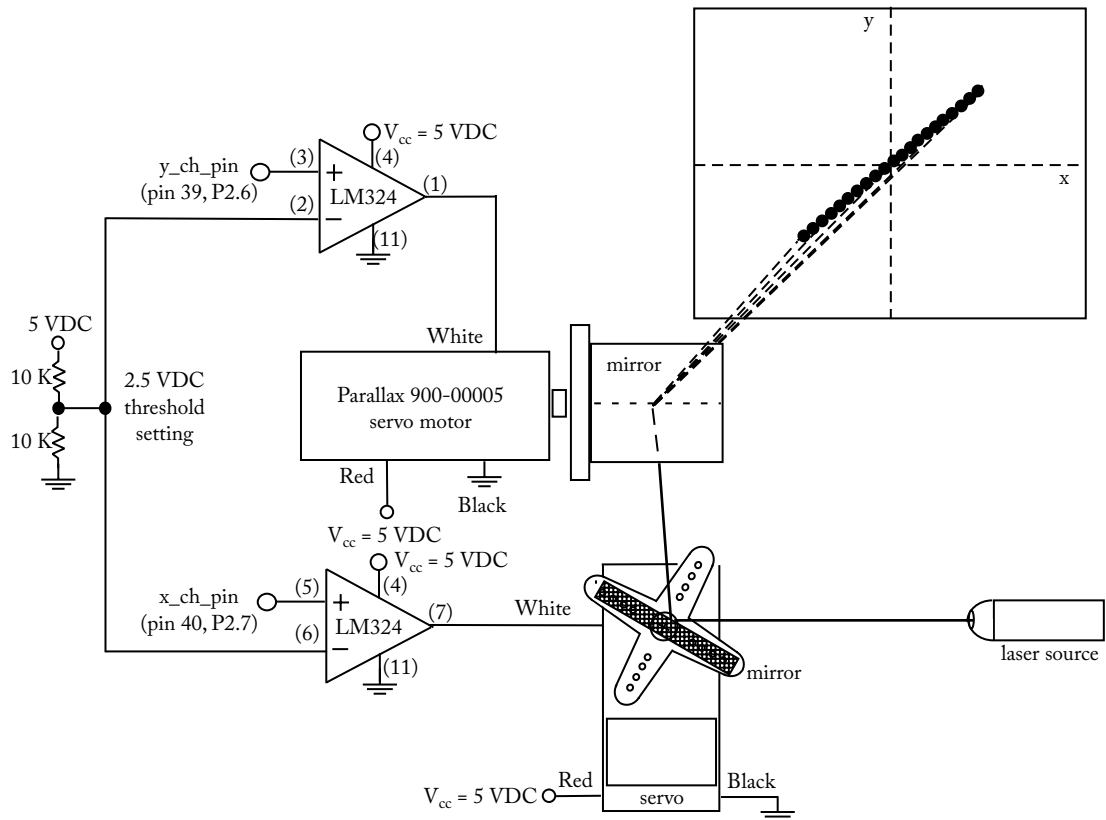


Figure 3.26: Inexpensive laser light show.

Energia contains useful servo configuration and control functions. The “attach” function initializes the servo at the specified pin. The MSP432 has pulse width modulated output features available on pins 19 (P2.5), 37 (P5.6), 38 (P2.4), 39 (P2.6), and 40 (P2.7). The “write” function rotates the servo the specified number of degrees. The program sends the same signal to both channel outputs (x_ch_pin, y_ch_pin) and traces a line with the laser. Any arbitrary shape may be traced by the laser using this technique.

```

//*****
//X-Y ramp
//
//This example code is in the public domain.
//*****

#include <Servo.h>          //Use Servo library, included with IDE

Servo myServo_x;          //Create Servo objects to control the
Servo myServo_y;          //X and Y servos

void setup()
{
  myServo_x.attach(40);    //Servo is connected to PWM pin 40
  myServo_y.attach(39);    //Servo is connected to PWM pin 39
}

void loop()
{
  int i = 0;
  for(i=0; i<=180; i++)    //Rotates servo 0 to 180 degrees
  {
    myServo_x.write(i);    //Rotate servo counter clockwise
    myServo_y.write(i);    //Rotate servo counter clockwise
    delay(20);             //Wait 20 milliseconds
    if(i==180)
      delay(5000);
  }
}

//*****

```

3.5.1 DC MOTOR INTERFACE, SPEED, AND DIRECTION CONTROL

Interface. A number of direct current devices are controlled with an electronic switching device such as a MOSFET (metal oxide semiconductor field effect transistor). Specifically, an N-channel enhancement MOSFET may be used to switch a high current load on and off (such as a motor) using a low current control signal from a microcontroller as shown in Figure 3.27. The low current control signal from the microcontroller is connected to the gate of the MOSFET via a MOSFET driver. As shown in Figure 3.27, an LTC 1157 MOSFET driver is used to boost the control signal

from the microcontroller to be compatible with an IRLR024 power MOSFET. The IRLR024 is rated at 60 VDC V_{DS} and a continuous drain current I_D of 14 amps. The IRLR024 MOSFET switches the high current load on and off consistent with the control signal. In a low-side connection, the high current load is connected between the MOSFET source and ground.

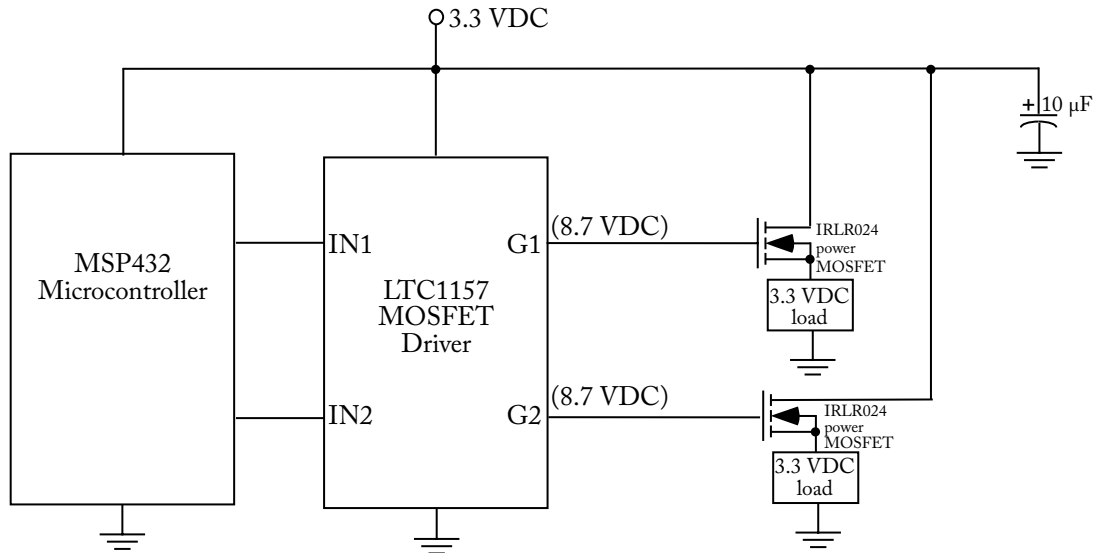


Figure 3.27: MOSFET drive circuit (adapted from Linear Technology).

Speed. As previously mentioned, DC motor speed may be varied by changing the applied voltage. This is difficult to do with a digital control signal. However, pulse width modulation (PWM) techniques combined with a MOSFET interface circuit may be used to precisely control motor speed. The duty cycle of the PWM signal governs the percentage of the motor supply voltage applied to the motor and hence the percentage of rated full speed at which the motor will rotate. The interface circuit to accomplish this type of control is shown in Figure 3.28. It is a slight variation of the control circuit provided in Figure 3.27. In this configuration, the motor supply voltage may be different than the microcontroller's 3.3 VDC supply. For an inductive load, a reverse biased protection diode should be connected across the load. The interface circuit allows the motor to rotate in a given direction.

Direction. For a DC motor to operate in both the clockwise and counterclockwise directions, the polarity of the DC motor supplied must be changed. To operate the motor in the forward direction, the positive battery terminal must be connected to the positive motor terminal while the negative battery terminal must be attached to the negative motor terminal. To reverse the motor direction, the motor supply polarity must be reversed. An H-bridge is a circuit employed to perform this polarity switch. An H-bridge may be constructed from discrete components as

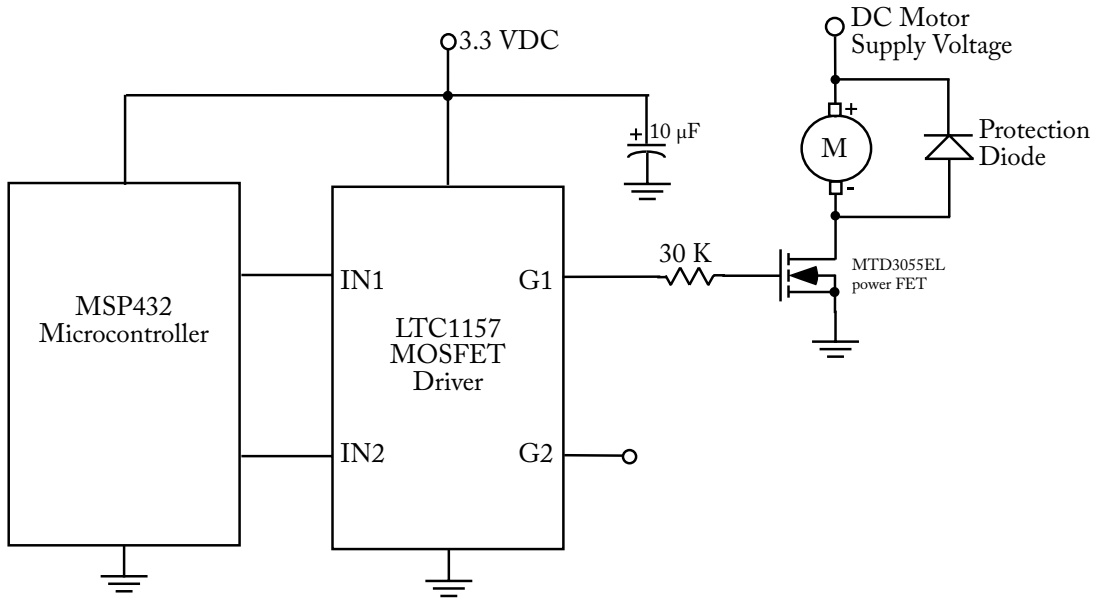


Figure 3.28: DC motor interface.

shown in Figure 3.29. If PWM signals are used to drive the base of the transistors, both motor speed and direction may be controlled by the circuit. The transistors used in the circuit must have a current rating sufficient to handle the current requirements of the motor during start and stall conditions.

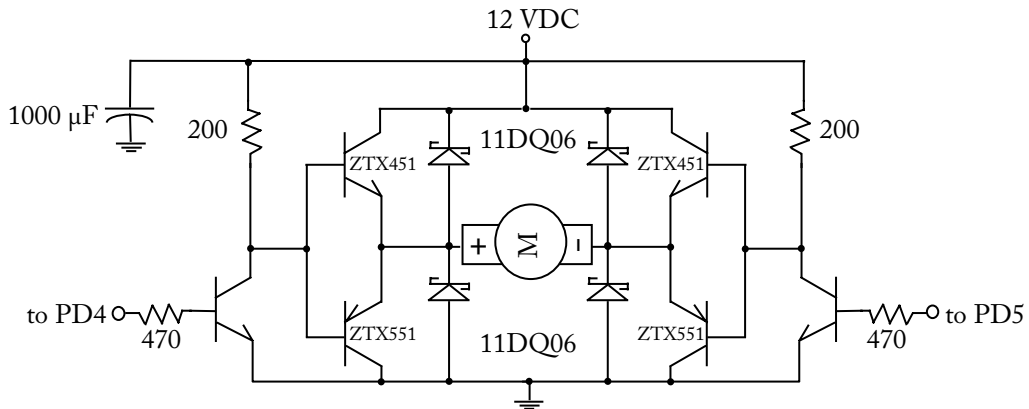


Figure 3.29: H-bridge control circuit.

Texas Instruments provides a self-contained H-bridge motor controller integrated circuit, the DRV8829. Within the DRV8829 package is a single H-bridge driver. The driver may control DC loads with supply voltages from 8–45 VDC with a peak current rating of 5 amps. The single H-bridge driver may be used to control a DC motor or one winding of a bipolar stepper motor [DRV8829, 2010].

Example 18: SN754410 H-bridge. Texas Instruments manufactures the SN754410 quad half-H driver. It provides bi-directional current up to 1A to a wide variety of loads including DC motors, relays, and solenoids. The loads can have operating voltages from 4.5–36 VDC [SLRS007C]. In this example, we use the SN754410 to give the Dagu Magician robot motors the capability to go forward and reverse. Recall from Chapter 2, the robot is controlled by two 7.2 VDC motors which independently drive left and right wheels. A third non-powered drag ball provides tripod stability for the robot.

In this example, we develop a circuit and test program to investigate the bi-directional features of the robot motors, as illustrated in Figure 3.30. A keypad is used to select the direction and speed for the Dagu robot motors. Motor control signals from the MSP432 are fed through digital logic to provide a direction and motor speed (PWM) signal to the two motors. The resulting control signals are then fed to the SN754410 to interface to the motors.

The Energia program was adapted from the keypad interface program provided earlier in the chapter.

```
//*****
//h_bridge
//
//This example code is in the public domain.
//*****

#define row1  23
#define row2  24
#define row3  25
#define row4  26

#define col1  27
#define col2  28
#define col3  29
#define col4  30

#define motorl_for_rev  37
#define motorl_pwm      38

#define motorr_for_rev  39
```

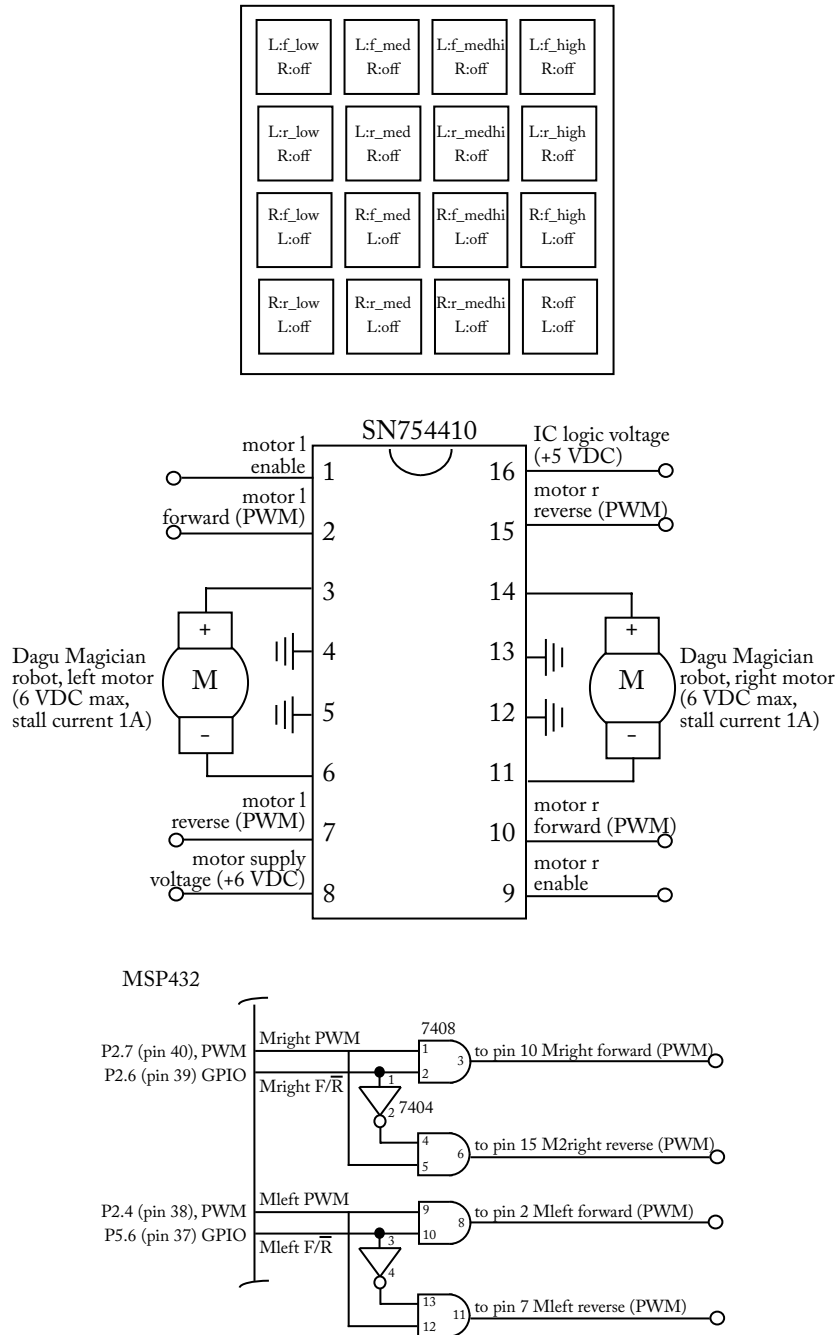


Figure 3.30: H-bridge control circuit for Daggu robot.

```
#define motorr_pwm      40

unsigned char  key_depressed = '*';

void setup()
{
  //start serial connection to monitor
  Serial.begin(9600);

  //configure row pins as ouput
  pinMode(row1, OUTPUT);
  pinMode(row2, OUTPUT);
  pinMode(row3, OUTPUT);
  pinMode(row4, OUTPUT);

  //configure column pins as input and assert pullup resistors
  pinMode(col1, INPUT_PULLUP);
  pinMode(col2, INPUT_PULLUP);
  pinMode(col3, INPUT_PULLUP);
  pinMode(col4, INPUT_PULLUP);

  //configure motor control pins as ouput
  pinMode(motorl_for_rev, OUTPUT);
  pinMode(motorl_pwm, OUTPUT);
  pinMode(motorr_for_rev, OUTPUT);
  pinMode(motorr_pwm, OUTPUT);
}

void loop()
{
  //Assert row1, deassert row 2,3,4
  digitalWrite(row1, LOW);  digitalWrite(row2, HIGH);
  digitalWrite(row3, HIGH); digitalWrite(row4, HIGH);

  //Read columns
  if (digitalRead(col1) == LOW)
    key_depressed = '0';
```

```
else if (digitalRead(col2) == LOW)
    key_depressed = '1';
else if (digitalRead(col3) == LOW)
    key_depressed = '2';
else if (digitalRead(col4) == LOW)
    key_depressed = '3';
else
    key_depressed = '*';

if (key_depressed == '*')
{
    //Assert row2, deassert row 1,3,4
    digitalWrite(row1, HIGH);    digitalWrite(row2, LOW);
    digitalWrite(row3, HIGH);    digitalWrite(row4, HIGH);

    //Read columns
    if (digitalRead(col1) == LOW)
        key_depressed = '4';
    else if (digitalRead(col2) == LOW)
        key_depressed = '5';
    else if (digitalRead(col3) == LOW)
        key_depressed = '6';
    else if (digitalRead(col4) == LOW)
        key_depressed = '7';
    else
        key_depressed = '*';
}

if (key_depressed == '*')
{
    //Assert row3, deassert row 1,2,4
    digitalWrite(row1, HIGH);    digitalWrite(row2, HIGH);
    digitalWrite(row3, LOW);     digitalWrite(row4, HIGH);

    //Read columns
    if (digitalRead(col1) == LOW)
        key_depressed = '8';
    else if (digitalRead(col2) == LOW)
```



```
digitalWrite(motorrr_for_rev, HIGH);
analogWrite(motorrr_pwm, 0);
break;

case '1' : Serial.println("Motor L - forward medium speed");
Serial.println("Motor R - off");
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 128);
digitalWrite(motorrr_for_rev, HIGH);
analogWrite(motorrr_pwm, 0);
break;

case '2' : Serial.println("Motor L - forward medium high speed");
Serial.println("Motor R - off");
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 192);
digitalWrite(motorrr_for_rev, HIGH);
analogWrite(motorrr_pwm, 0);
break;

case '3' : Serial.println("Motor L - forward high speed");
Serial.println("Motor R - off");
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 255);
digitalWrite(motorrr_for_rev, HIGH);
analogWrite(motorrr_pwm, 0);
break;

case '4' : Serial.println("Motor L - reverse low speed");
Serial.println("Motor R - off");
digitalWrite(motorl_for_rev, LOW);
analogWrite(motorl_pwm, 64);
digitalWrite(motorrr_for_rev, HIGH);
analogWrite(motorrr_pwm, 0);
break;

case '5' : Serial.println("Motor L - reverse medium speed");
Serial.println("Motor R - off");
digitalWrite(motorl_for_rev, LOW);
```



```
        analogWrite(motorl_pwm, 128);
        digitalWrite(motorrr_for_rev, HIGH);
        analogWrite(motorrr_pwm, 0);
        break;

    case '6' :   Serial.println("Motor L - reverse medium high speed");
                Serial.println("Motor R - off");
                digitalWrite(motorl_for_rev, LOW);
                analogWrite(motorl_pwm, 192);
                digitalWrite(motorrr_for_rev, HIGH);
                analogWrite(motorrr_pwm, 0);
                break;

    case '7' :   Serial.println("Motor L - reverse high speed");
                Serial.println("Motor R - off");
                digitalWrite(motorl_for_rev, LOW);
                analogWrite(motorl_pwm, 255);
                digitalWrite(motorrr_for_rev, HIGH);
                analogWrite(motorrr_pwm, 0);
                break;

    case '8' :   Serial.println("Motor R - forward low speed");
                Serial.println("Motor L - off");
                digitalWrite(motorrr_for_rev, HIGH);
                analogWrite(motorrr_pwm, 64);
                digitalWrite(motorl_for_rev, HIGH);
                analogWrite(motorl_pwm, 0);
                break;

    case '9' :   Serial.println("Motor R - forward medium speed");
                Serial.println("Motor L - off");
                digitalWrite(motorrr_for_rev, HIGH);
                analogWrite(motorrr_pwm, 128);
                digitalWrite(motorl_for_rev, HIGH);
                analogWrite(motorl_pwm, 0);
                break;

    case 'A' :   Serial.println("Motor R - forward medium high speed");
                Serial.println("Motor L - off");
```

```
digitalWrite(motorr_for_rev, HIGH);
analogWrite(motorr_pwm, 192);
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 0);
break;

case 'B' : Serial.println("Motor R - forward high speed");
Serial.println("Motor L - off");
digitalWrite(motorr_for_rev, HIGH);
analogWrite(motorr_pwm, 255);
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 0);
break;

case 'C' : Serial.println("Motor R - reverse low speed");
Serial.println("Motor L - off");
digitalWrite(motorr_for_rev, LOW);
analogWrite(motorr_pwm, 64);
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 0);
break;

case 'D' : Serial.println("Motor R - reverse medium speed");
Serial.println("Motor L - off");
digitalWrite(motorr_for_rev, LOW);
analogWrite(motorr_pwm, 128);
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 0);
break;

case 'E' : Serial.println("Motor R - reverse medium high speed");
Serial.println("Motor L - off");
digitalWrite(motorr_for_rev, LOW);
analogWrite(motorr_pwm, 192);
digitalWrite(motorl_for_rev, HIGH);
analogWrite(motorl_pwm, 0);
break;

case 'F' : Serial.println("Motor R - reverse high speed");
```

```

        Serial.println("Motor L - off");
        digitalWrite(motorr_for_rev, LOW);
        analogWrite(motorr_pwm, 0);
        digitalWrite(motorl_for_rev, HIGH);
        analogWrite(motorl_pwm, 0);
        break;
    }
}
//limit switch bounce
delay(200);
}

//*****

```

3.5.2 DC SOLENOID CONTROL

The interface circuit for a DC solenoid is shown in Figure 3.31. A solenoid is used to activate a mechanical insertion (or extraction). As in previous examples, we employ the LTC1157 MOSFET driver between the microcontroller and the power MOSFET used to activate the solenoid. A reverse biased diode is placed across the solenoid. Both the solenoid power supply and the MOSFET must have the appropriate voltage and current rating to support the solenoid requirements.

3.5.3 STEPPER MOTOR CONTROL

Stepper motors are used to provide a discrete angular displacement in response to a control signal step. There are a wide variety of stepper motors including bipolar and unipolar types with different configurations of motor coil wiring. Due to space limitations we only discuss the unipolar, 5 wire stepper motor. The internal coil configuration for this motor is shown in Figure 3.32b.

Often, a wiring diagram is not available for the stepper motor. Based on the wiring configuration (see Figure 3.32b), one can find out the common line for both coils. It has a resistance that is one-half of all of the other coils. Once the common connection is found, one can connect the stepper motor into the interface circuit. By changing the other connections, one can determine the correct connections for the step sequence. To rotate the motor either clockwise or counterclockwise, a specific step sequence must be sent to the motor control wires as shown in Figure 3.32b.

The microcontroller does not have sufficient capability to drive the motor directly. Therefore, an interface circuit is required as shown in Figure 3.33. The speed of motor rotation is determined by how fast the control sequence is completed.

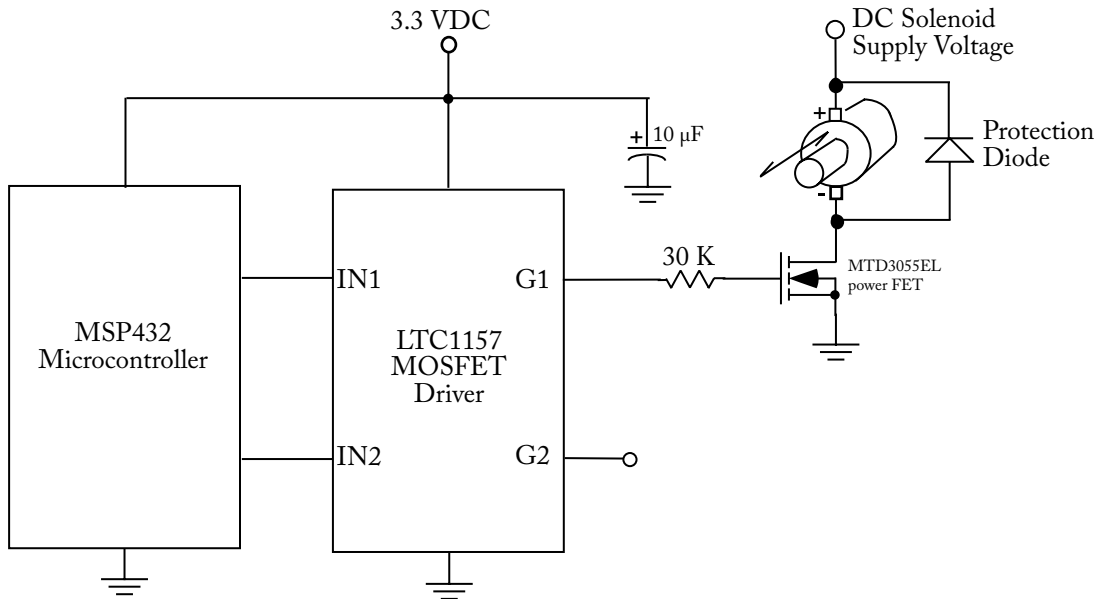


Figure 3.31: Solenoid interface circuit.

```

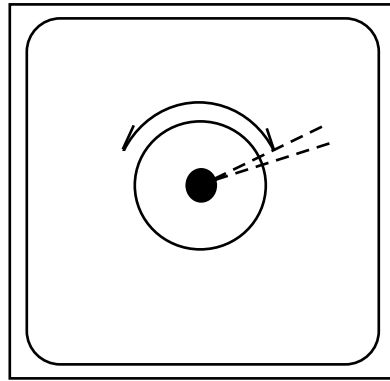
//*****
//stepper
//
//This example code is in the public domain.
//*****

//external switches
#define ext_sw1  23
#define ext_sw2  24

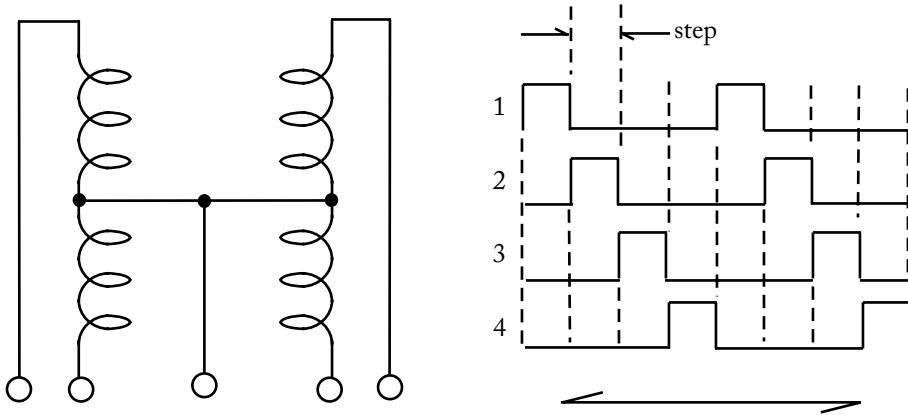
//stepper channels
#define stepper_ch1  25
#define stepper_ch2  26
#define stepper_ch3  27
#define stepper_ch4  28

int switch_value1, switch_value2;
int motor_speed = 1000;          //motor increment time in ms
int last_step = 1;

```



(a) A stepper motor rotates a fixed angle per step.



(b) Coil configuration and step sequence.

Figure 3.32: Unipolar stepper motor.

```

int next_step;

void setup()
{
  //Screen
  Serial.begin(9600);

  //external switches
  pinMode(ext_sw1, INPUT);

```

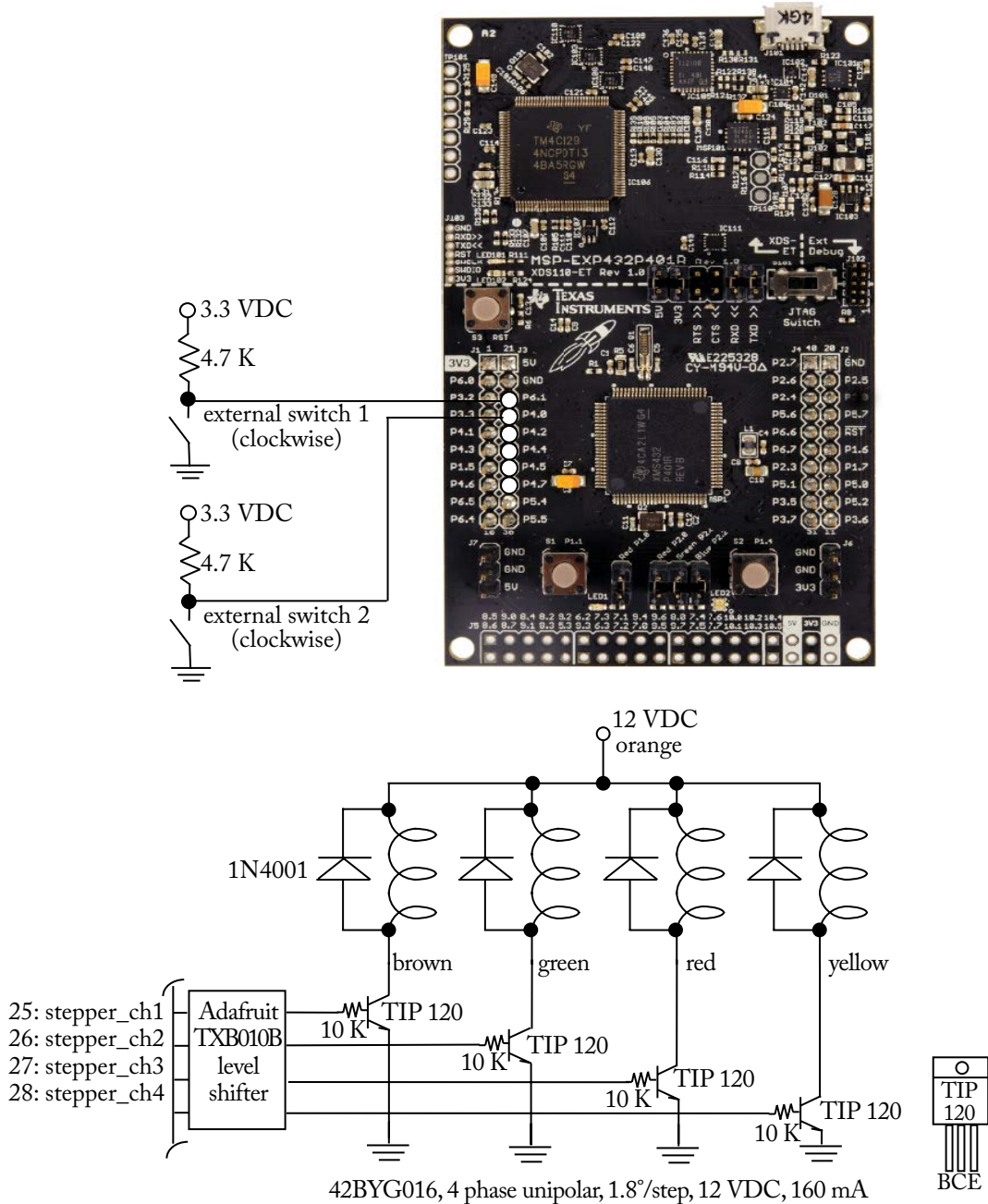


Figure 3.33: Unipolar stepper motor interface circuit.

```
pinMode(ext_sw2, INPUT);

//stepper channel
pinMode(stepper_ch1, OUTPUT);
pinMode(stepper_ch2, OUTPUT);
pinMode(stepper_ch3, OUTPUT);
pinMode(stepper_ch4, OUTPUT);

}

void loop()
{
switch_value1 = digitalRead(ext_sw1);
switch_value2 = digitalRead(ext_sw2);

if(switch_value1 == LOW)           //switch1 asserted
{
while(switch_value1 == LOW)       //clockwise
{
if(last_step == 1)
{
Serial.println("Switch 1: low, step 1");
digitalWrite(stepper_ch1, HIGH);
digitalWrite(stepper_ch2, LOW);
digitalWrite(stepper_ch3, LOW);
digitalWrite(stepper_ch4, LOW);
next_step = 2;
}
}
else if(last_step == 2)
{
Serial.println("Switch 1: low, step 2");
digitalWrite(stepper_ch1, LOW);
digitalWrite(stepper_ch2, HIGH);
digitalWrite(stepper_ch3, LOW);
digitalWrite(stepper_ch4, LOW);
next_step = 3;
}
}
else if(last_step == 3)
{
```

```
    Serial.println("Switch 1: low, step 3");
    digitalWrite(stepper_ch1, LOW);
    digitalWrite(stepper_ch2, LOW);
    digitalWrite(stepper_ch3, HIGH);
    digitalWrite(stepper_ch4, LOW);
    next_step = 4;
  }
else if(last_step == 4)
  {
    Serial.println("Switch 1: low, step 4");
    digitalWrite(stepper_ch1, LOW);
    digitalWrite(stepper_ch2, LOW);
    digitalWrite(stepper_ch3, LOW);
    digitalWrite(stepper_ch4, HIGH);
    next_step = 1;
  }
else
  {
    ;
  }
last_step = next_step;
delay(motor_speed);
switch_value1 = digitalRead(ext_sw1);
} //end while
} //end if

else if(switch_value2 == LOW) //switch2 asserted
  {
    while(switch_value2 == LOW) //counter clockwise
      {
        if(last_step == 1)
          {
            Serial.println("Switch 2: low, step 1");
            digitalWrite(stepper_ch1, HIGH);
            digitalWrite(stepper_ch2, LOW);
            digitalWrite(stepper_ch3, LOW);
            digitalWrite(stepper_ch4, LOW);
            next_step = 4;
          }
      }
  }
```



```
else if(last_step == 2)
{
  Serial.println("Switch 2: low, step 2");
  digitalWrite(stepper_ch1, LOW);
  digitalWrite(stepper_ch2, HIGH);
  digitalWrite(stepper_ch3, LOW);
  digitalWrite(stepper_ch4, LOW);
  next_step = 1;
}
else if(last_step == 3)
{
  Serial.println("Switch 2: low, step 3");
  digitalWrite(stepper_ch1, LOW);
  digitalWrite(stepper_ch2, LOW);
  digitalWrite(stepper_ch3, HIGH);
  digitalWrite(stepper_ch4, LOW);
  next_step = 2;
}
else if(last_step == 4)
{
  Serial.println("Switch 2: low, step 4");
  digitalWrite(stepper_ch1, LOW);
  digitalWrite(stepper_ch2, LOW);
  digitalWrite(stepper_ch3, LOW);
  digitalWrite(stepper_ch4, HIGH);
  next_step = 3;
}
else
{
  ;
}
last_step = next_step;
delay(motor_speed);
switch_value2 = digitalRead(ext_sw2);
} //end while
} //end if

else
{
```

```

digitalWrite(stepper_ch1, LOW);
digitalWrite(stepper_ch2, LOW);
digitalWrite(stepper_ch3, LOW);
digitalWrite(stepper_ch4, LOW);
}
}
//*****

```

3.5.4 OPTICAL ISOLATION

It is a good design practice to provide optical isolation between a motor control circuit and the motor. A typical optical isolator (e.g., 4N25) consists of an LED and an optical transistor in a common package as shown in Figure 3.34. The LED is driven by a low voltage control signal from the MSP432, whereas, the optical transistor provides the control signal to the motor interface circuit. The link between the MSP432 to the motor interface circuit is now enabled by light rather than an electrical link. This provides a high level of noise isolation between the processor and the motor interface circuit. Many optical isolators also provide a signal inversion.

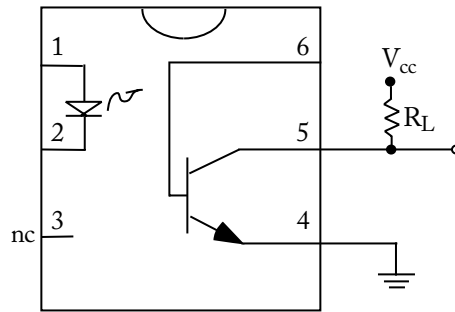


Figure 3.34: 4N25 optical isolator.

3.6 INTERFACING TO MISCELLANEOUS DC DEVICES

In this section, we present a potpourri of interface circuits to connect a microcontroller to a wide variety of DC peripheral devices.

3.6.1 SONALERTS, BEEPERS, BUZZERS

In Figure 3.35, we show several circuits used to interface a microcontroller to a buzzer, beeper, or other types of annunciator devices such as a sonalert. It is important that the interface transistor and the supply voltage are matched to the requirements of the sound producing device.

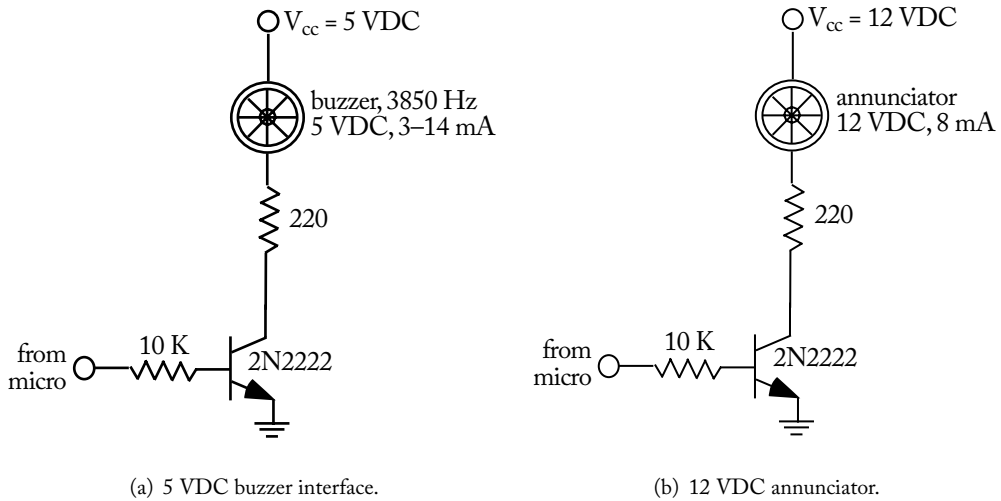


Figure 3.35: Sonalert, beepers, buzzers.

3.6.2 VIBRATING MOTOR

A vibrating motor is often used to gain one's attention as in a cell phone. These motors are typically rated at 3 VDC and a high current. The interface circuit shown in Figure 3.27 is used to drive the low voltage motor.

3.6.3 DC FAN

The interface circuit shown in Figure 3.25 may also be used to control a DC fan. As before, a reverse biased diode is placed across the DC fan motor.

3.6.4 BILGE PUMP

A bilge pump is a pump specifically designed to remove water from the inside of a boat. The pumps are powered from a 12 VDC source and have typical flow rates from 360 to over 3,500 gallons per minute. They range in price from US \$20–US \$80 (www.shorelinemarinedevelopment.com). An interface circuit to control a bilge pump from MSP432 is shown in Figure 3.36. The interface circuit consists of a 470 ohm resistor, a power NPN Darlington transistor (TIP 120) and a 1N4001 diode. The 12 VDC supply should have sufficient current capability to supply the needs of the bilge pump.

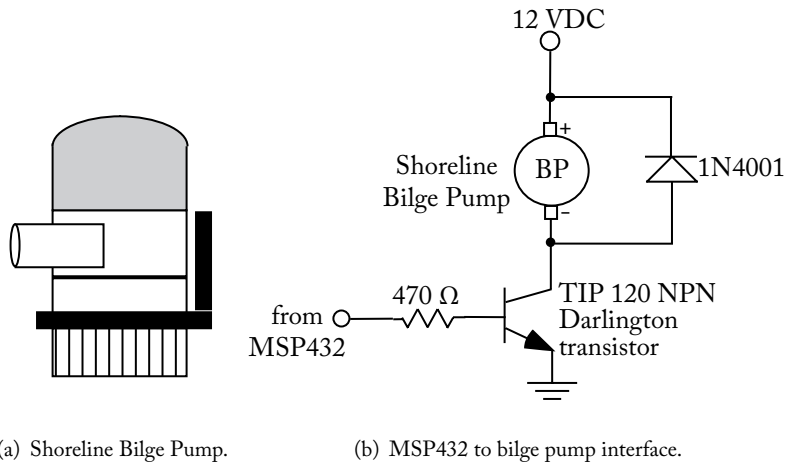


Figure 3.36: Bilge pump interface.

3.7 AC DEVICES

A high-power alternating current (AC) load may be switched on and off using a low-power control signal from the microcontroller. In this case, a Solid State Relay is used as the switching device. Solid state relays are available to switch a high power DC or AC load [Crydom]. For example, the Crydom 558-CX240D5R is a printed circuit board mounted, air-cooled, single pole single throw (SPST), normally open (NO) solid state relay. It requires a DC control voltage of 3–15 VDC at 15 mA. This microcontroller compatible DC control signal is used to switch 12–280 VAC loads rated from 0.06–5 amps [Crydom].

To vary the direction of an AC motor, you must use a bi-directional AC motor. A bi-directional motor is equipped with three terminals: common, clockwise, and counterclockwise. To turn the motor clockwise, an AC source is applied to the common and clockwise connections. In like manner, to turn the motor counterclockwise, an AC source is applied to the common and counterclockwise connections. This may be accomplished using two of the Crydom SSRs.

PowerSwitch manufactures an easy-to-use AC interface the PowerSwitch Tail II. The device consists of a control module with attached AC connections rated at 120 VAC, 15 A. The device to be controlled is simply plugged inline with the PowerSwitch Tail II. A digital control signal from MSP432 (3 VDC at 3 mA) serves as the on/off control signal for the controlled AC device. The controlled signal is connected to the PowerSwitch Tail II via a terminal block connection. The PowerSwitch II may be configured as either normally closed (NC) or normally open (NO) (www.powerswitchtail.com).

Example 19: PowerSwitch Tail II. In this example, we use an IR sensor to detect someone's presence. If the IR sensor's output reaches a predetermined threshold level, an AC desk lamp is illuminated as shown in Figure 3.37.

```
//*****
//switch_tail
//
//The circuit:
// - The IR sensor signal pin is connected to analog pin 0 (30).
//   The sensor power and ground pins are connected to 5 VDC and
//   ground respectively.
// - The analog output is designated as the onboard red LED.
// - The switch tail control signal is connected to P6.0 (pin 2)
//
//Adapted for code originally written by Tom Igoe
//Created: Dec 29, 2008
//Modified: Aug 30, 2011
//Author: Tom Igoe
//
//This example code is in the public domain.
//*****

const int analogInPin = 30;      //Energia analog input pin A0
const int analogOutPin = 75;     //Energia onboard red LED pin
const int switch_tail_control =2; //Switch Tail control signal

int sensorValue = 0;            //value read from the OR sensor
int outputValue = 0;           //value output to the PWM (red LED)

void setup()
{
  //initialize serial communications at 9600 bps:
  Serial.begin(9600);

  //configure Switch Tail control pin
  pinMode(switch_tail_control, OUTPUT);
}

void loop()
{
```

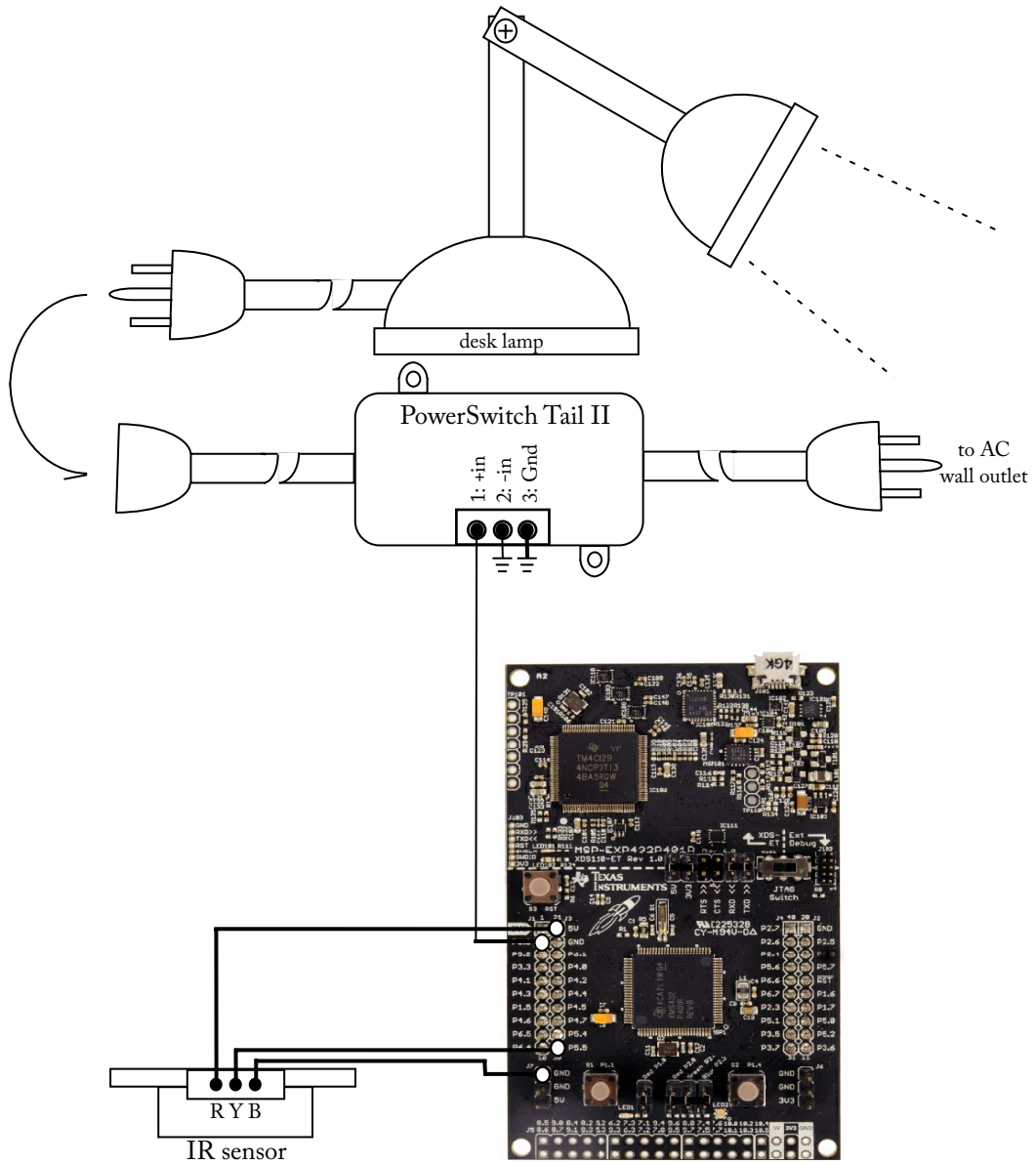


Figure 3.37: PowerSwitch Tail II.

```

//read the analog in value:
sensorValue = analogRead(analogInPin);

//map it to the range of the analog out:
outputValue = map(sensorValue, 0, 1023, 0, 255);

//change the analog out value:
analogWrite(analogOutPin, outputValue);

//Switch Tail control signal
if(outputValue >= 128)
  {
  digitalWrite(switch_tail_control, HIGH);
  Serial.print("Light on");
  }
else
  {
  digitalWrite(switch_tail_control, LOW);
  Serial.print("Light off");
  }

// print the results to the serial monitor:
Serial.print("sensor = " );
Serial.print(sensorValue);
Serial.print("\t output = ");
Serial.println(outputValue);

// wait 10 milliseconds before the next loop
// for the analog-to-digital converter to settle
// after the last reading:
delay(10);
}

//*****

```

3.8 EDUCATIONAL BOOSTER PACK MKII

The Educational Booster Pack MkII allows rapid prototyping of designs. Shown in Figure 3.38, it is equipped with a variety of transducers and output devices including [SLAU599, 2015].

- **Two-axis joystick.** The ITEAD Studio IM130330001 is a two-axis analog joystick equipped with a pushbutton. The two analog signals are generated by x- and y-oriented potentiometers. As the joystick is moved the analog signals relay the joystick position to the MSP432 via the J1.2 (X) and J3.26 (Y) header pins. The joystick select pushbutton is connected to pin J1.5.
- **Microphone.** The MkII is equipped with the CUI CMA-4544PW-W electret microphone. The microphone signal is amplified via an OPA344 operational amplifier. The microphone has a frequency response of 20 Hz to 20 kHz. The microphone is connected to MSP432 pin J1.6.
- **Light sensor.** The light sensor aboard the MkII is the OPT3001 digital ambient light sensor. The sensor measures ambient light intensity and it is tuned to the light response of the human eye. It also has filters to reject infrared (IR) light. It detects light intensity in the range from 0.01–83 lux. The I2C compatible output of the sensor is provided to MSP432 pins J1.9 (I2C SCL), J1.10 (I2C SDA), and J1.8 (sensor interrupt).
- **Temperature sensor.** The temperature sensor is also I2C compatible. The TMP006 is a noncontact sensor that passively absorbs IR wavelengths from 4–16 μm . The I2C compatible output is provided to MSP432 pins J1.9 (I2C SCL), J1.10 (I2C SDA), and J2.11 (sensor interrupt).
- **Servo motor controller.** The MkII is equipped with a convenient connector for a servo motor. The servo motor control signal is provided by MSP432 signal pin J2.19.
- **Three-axis accelerometer.** Aboard the MkII is a Kionix KXTC9-2050 three-axis accelerometer that measures acceleration in the X, Y, and Z directions. The three-channel analog output corresponds to acceleration from $\pm 1.5\text{ g}$ – $\pm 6\text{ g}$. The three channels of analog output are available at MSP432 pins J3.23 (X), J3.24 (Y), and J3.25 (Z).
- **Pushbuttons.** The MkII is equipped with two pushbuttons designated S1 and S2. They are connected to the MSP432 via pins J4.33 (S1) and J4.32 (S2).
- **Red-Green-Blue (RGB) LED.** The RGB LED aboard the MkII is the Cree CLV1A-FKB RGB multicolor LED. The three color components are accessible via MSP432 pins J4.39 (red), J4.38 (green), and J4.37 (blue). The intensity of each component may be adjusted using pulse width modulation (PWM) techniques.
- **Buzzer.** The piezo buzzer aboard the MkII is the CUI CEM-1230. The buzzer will operate at various frequencies using PWM techniques. The buzzer is accessible via MSP432 J4.40.
- **Color TFT LCD.** The color 2D LCD aboard the MkII is controlled via the serial peripheral interface (SPI) system. The Crystalfontz CFAF 128128B-0145T is a color 128 by 128 pixel display.

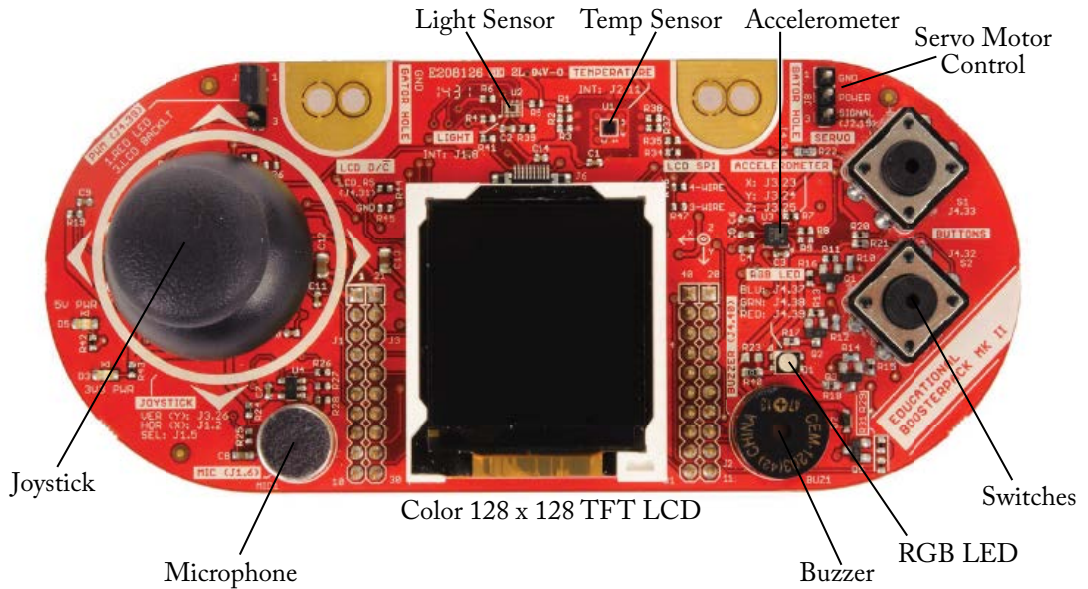


Figure 3.38: Educational Booster Pack MkII. Illustration used with permission of Texas Instruments www.ti.com.

Provided in Energia 17 (and beyond) is considerable software support for the MkII. This software will be explored in the Laboratory Exercise accompanying this chapter.

3.9 GROVE STARTER KIT FOR LAUNCHPAD

Seeed provides a Grove Starter Kit for the MSP432 LaunchPad shown in Figure 3.39. It consists of a BoosterPack configured breakout board for a number of sensors and output devices including (www.seeedstudio.com):

- buzzer;
- four-digit seven segment LED display;
- relay;
- proximity Infrared Sensor (PIR) sensor;
- ultrasonic ranger;
- light sensor;
- rotary angle sensor;

- sound sensor;
- moisture sensor; and
- temperature and humidity sensor.



Figure 3.39: Grove Starter Kit for LaunchPad (www.seeedstudio.com). Illustrations used with permission of Texas Instruments (www.TI.com).

The Grove Starter Kit is enhanced by considerable software support we explore in the Laboratory Exercise section of the chapter.

3.10 APPLICATION: SPECIAL EFFECTS LED CUBE

To illustrate some of the fundamentals of MSP432 interfacing, we construct a three-dimensional LED cube. This design was inspired by an LED cube kit available from Jameco (www.jameco.com). This application originally appeared in the third edition of “Arduino Microcontroller Processing for Everyone!” The LED cube example has been adapted with permission for compatibility with the MSP432 [Barrett, 2013].

The MSP432-EXP432P401R LaunchPad is a 3.3 VDC system. With this in mind, we take two different design approaches.

1. Interface the 3.3 VDC MSP432 to an LED cube designed for 5 VDC operation via a 3.3–5.0 VDC level shifter.
2. Modify the design of the LED cube to operate at 3.3 VDC.

We explore each design approach in turn.

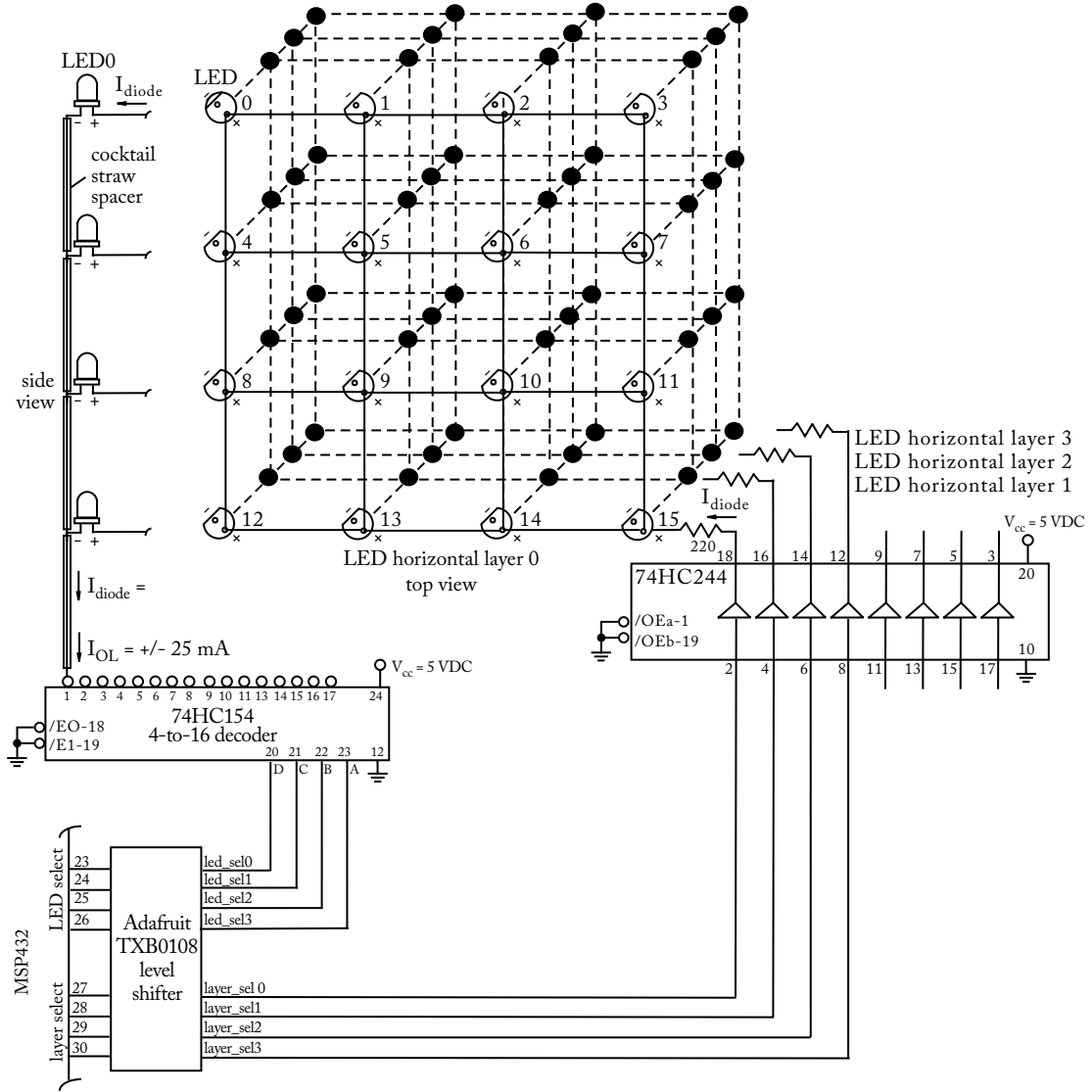
Approach 1: 5 VDC LED cube. The LED cube consists of four layers of LEDs with 16 LEDs per layer. Only a single LED is illuminated at a given time. However, different effects may be achieved by how long a specific LED is left illuminated and the pattern of LED sequence followed. A specific LED layer is asserted using the layer select pins on the microcontroller using a one-hot-code (a single line asserted while the others are de-asserted). The asserted line is fed through a 74HC244 (three state, octal buffer, line driver) which provides an I_{OH}/I_{OL} current of ± 35 mA as shown in Figure 3.40. A given output from the 74HC244 is fed to a common anode connection for all 16 LEDs in a layer. All four LEDs in a specific LED position, each in a different layer, share a common cathode connection. That is, an LED in a specific location within a layer shares a common cathode connection with three other LEDs that share the same position in the other three layers. The common cathode connection from each LED location is fed to a specific output of the 74HC154 4–16 decoder. The decoder has a one-cold-code output (one output at logic low while the others are at logic high). To illuminate a specific LED, the appropriate layer select and LED select line are asserted using the `layer_sel[3:0]` and `led_sel[3:0]` lines respectively. This basic design may be easily expanded to a larger LED cube.

To interface the 5 VDC LED cube to the 3.3 VDC MSP432, a 3.3 VDC–5 VDC level shifter is required for each of the control signals (`layer_sel` and `led_sel`). In this example, a TXB0108 (low voltage octal bidirectional transceiver) is employed to shift the 3.3 VDC signals of the MSP432 to 5 VDC levels. Adafruit provides a breakout board for the level shifter (#TXB0108)(www.adafruit.com). Alternatively, a Texas Instrument LSF0101XEVMM-001, discussed earlier in the chapter, may be used for level shifting.

Approach 2: 3.3 VDC LED cube. A 3.3 VDC LED cube design is shown in Figure 3.41. The 74HC154 1-of-16 decoder has been replaced by two 3.3 VDC 74LVX138 1-of-8 decoders. The two 74LVX138 decoders form a single 1-of-16 decoder. The `led_sel3` is used to select between the first decoder via enable pin /E2 or the second decoder via enable pin E3. Also, the 74HC244 has been replaced by a 3.3 VDC 74LVX244.

3.10.1 CONSTRUCTION HINTS

To limit project costs, low-cost red LEDs (Jameco #333973) are used. This LED has a forward voltage drop (V_f) of approximately 1.8 VDC and a nominal forward current (I_f) of 20 mA. The project requires a total of 64 LEDs (4 layers of 16 LEDs each). An LED template pattern was constructed from a 5" by 5" piece of pine wood. A 4-by-4 pattern of holes were drilled into the wood. Holes were spaced 3/4" apart. The hole diameter was slightly smaller than the diameter of the LEDs to allow for a snug LED fit.



- Notes:
1. LED cube consists of 4 layers of 16 LEDs each.
 2. Each LED is individually addressed by asserting the appropriate cathode signal (0-15) and asserting a specific LED layer.
 3. All LEDs in a given layer share a common anode connection.
 4. All LEDs in a given position (0-15) share a common cathode connection.

Figure 3.40: 5 VDC LED special effects cube.

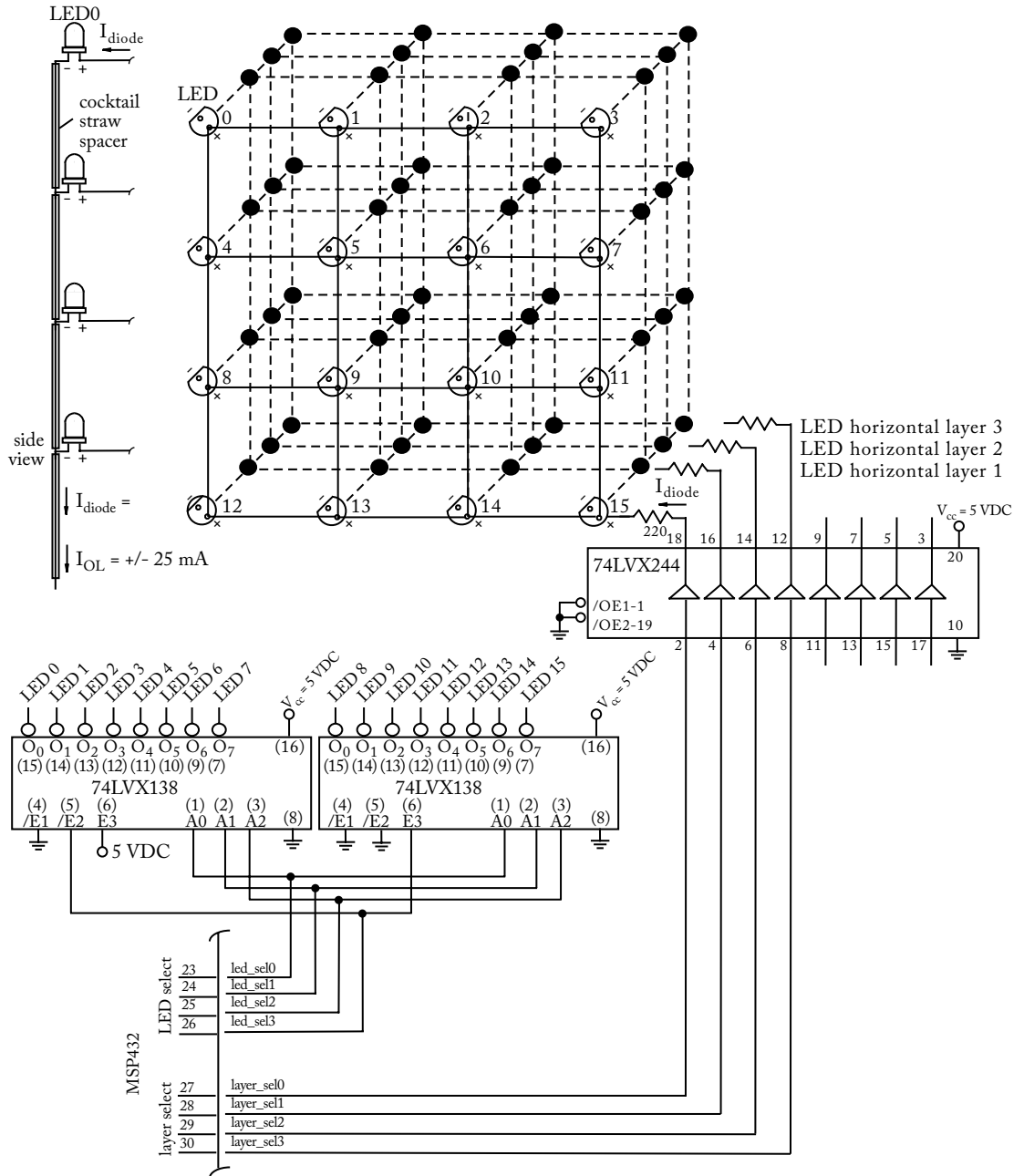


Figure 3.41: LED special effects cube.

The LED array was constructed a layer at a time using the wood template. Each LED was tested before inclusion in the array. A 5 VDC power supply with a series 220 ohm resistor was used to insure each LED was fully operational. The LED anodes in a given LED row were then soldered together. A fine tip soldering iron and a small bit of solder were used for each interconnect, as shown in Figure 3.42. Cross wires were then used to connect the cathodes of adjacent rows. A 22 gage bare wire was used. Again, a small bit of solder was used for the interconnect points. Four separate LED layers (4 by 4 array of LEDs) were completed.

To assemble the individual layers into a cube, cocktail straw segments were used as spacers between the layers. The straw segments provided spacing between the layers and also offered improved structural stability. The anodes for a given LED position were soldered together. For example, all LEDs in position 0 for all four layers shared a common anode connection.

The completed LED cube was mounted on a perforated printed circuit board (perfboard) to provide a stable base. LED sockets for the 74LS244 and the 74HC154 were also mounted to the perfboard. Connections were routed to a 16 pin ribbon cable connector. The other end of the ribbon cable was interfaced to the appropriate pins of the MSP432 via the level shifter. The entire LED cube was mounted within a 4" plexiglass cube. The cube is available from the Container Store (www.containerstore.com). A construction diagram is provided in Figure 3.42. A picture of the LED cube is shown in Figure 3.43.

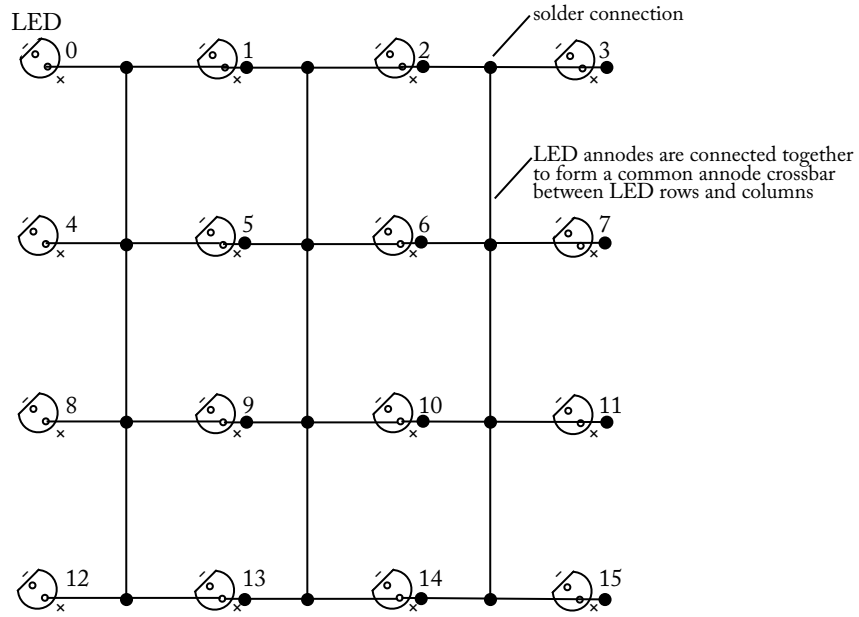
3.10.2 LED CUBE MSP432 ENERGIA CODE

Provided below is the basic code template to illuminate a single LED (LED 0, layer 0). This basic template may be used to generate a number of special effects (e.g., tornado, black hole, etc.). Pin numbers are provided for the MSP-EXP432P401R LaunchPad.

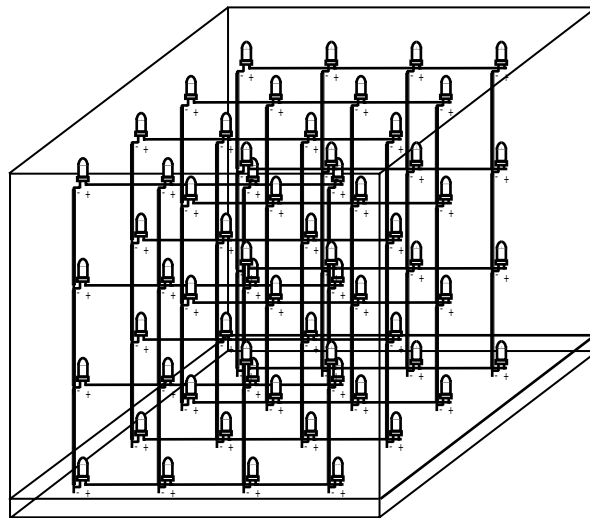
```
//*****
//led_cube
//
//This example code is in the public domain.
//*****

//led select pins
#define led_sel0 23
#define led_sel1 24
#define led_sel2 25
#define led_sel3 26

//layer select pins
#define layer_sel0 27
#define layer_sel1 28
#define layer_sel2 29
```



(a) LED soldering diagram.



(b) 3D LED array mounted within plexiglass cube.

Figure 3.42: LED cube construction.

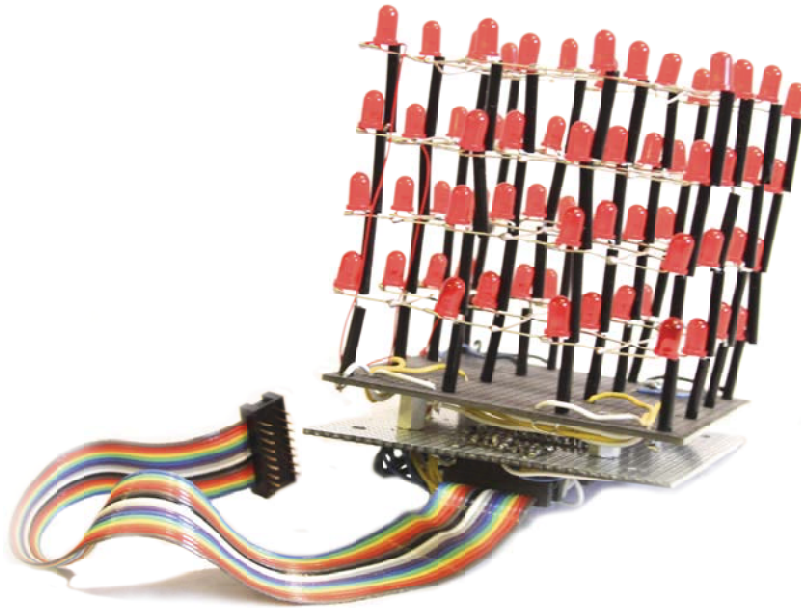


Figure 3.43: LED Cube (photo courtesy of J. Barrett, Closer to the Sun International, 2015).

```
#define layer_sel3 30

void setup()
{
  pinMode(led_sel0, OUTPUT);
  pinMode(led_sel1, OUTPUT);
  pinMode(led_sel2, OUTPUT);
  pinMode(led_sel3, OUTPUT);

  pinMode(layer_sel0, OUTPUT);
  pinMode(layer_sel1, OUTPUT);
  pinMode(layer_sel2, OUTPUT);
  pinMode(layer_sel3, OUTPUT);
}

void loop()
{
  //illuminate LED 0, layer 0
```



```

                //led select
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, LOW);
                //layer select
digitalWrite(layer_sel0, HIGH);
digitalWrite(layer_sel1, LOW);
digitalWrite(layer_sel2, LOW);
digitalWrite(layer_sel3, LOW);

delay(500);          //delay specified in ms
}

```

```
//*****
```

In the next example, a function “illuminate_LED” has been added. To illuminate a specific LED, the LED position (0–15), the LED layer (0–3), and the length of time to illuminate the LED in milliseconds are specified. In this short example, LED 0 is sequentially illuminated in each layer. An LED grid map is shown in Figure 3.44. It is useful for planning special effects.

```

//*****
//led_cube2
//
//This example code is in the public domain.
//*****

//led select pins
#define led_sel0  23
#define led_sel1  24
#define led_sel2  25
#define led_sel3  26

//layer select pins
#define layer_sel0  27
#define layer_sel1  28
#define layer_sel2  29
#define layer_sel3  30

void setup()
{

```

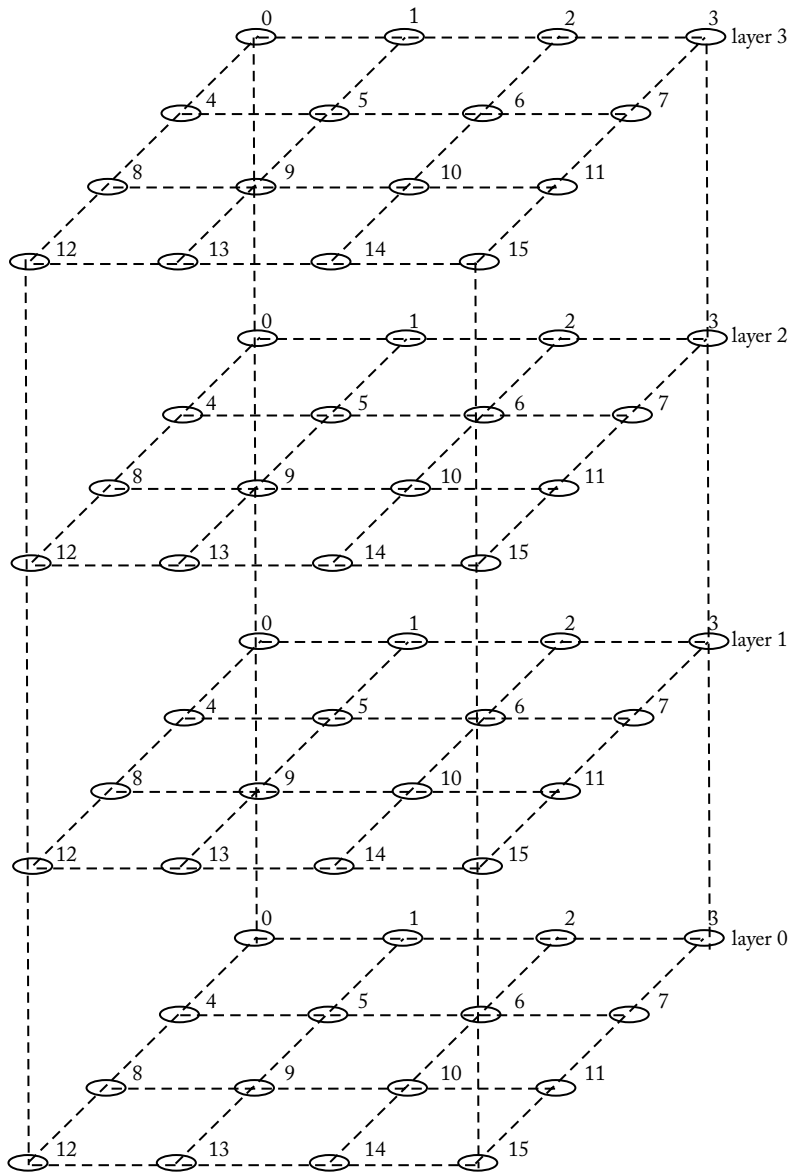


Figure 3.44: LED grid map.

```
pinMode(led_sel0, OUTPUT);
pinMode(led_sel1, OUTPUT);
pinMode(led_sel2, OUTPUT);
pinMode(led_sel3, OUTPUT);

pinMode(layer_sel0, OUTPUT);
pinMode(layer_sel1, OUTPUT);
pinMode(layer_sel2, OUTPUT);
pinMode(layer_sel3, OUTPUT);
}

void loop()
{
illuminate_LED(0, 0, 500);
illuminate_LED(0, 1, 500);
illuminate_LED(0, 2, 500);
illuminate_LED(0, 3, 500);
}

//*****

void illuminate_LED(int led, int layer, int delay_time)
{
if(led==0)
{
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, LOW);
}
else if(led==1)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, LOW);
}
else if(led==2)
{
```

```
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, LOW);
}
else if(led==3)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, LOW);
}
else if(led==4)
{
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, LOW);
}
else if(led==5)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, LOW);
}
else if(led==6)
{
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, LOW);
}
else if(led==7)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, LOW);
```

```
    }  
if(led==8)  
    {  
    digitalWrite(led_sel0, LOW);  
    digitalWrite(led_sel1, LOW);  
    digitalWrite(led_sel2, LOW);  
    digitalWrite(led_sel3, HIGH);  
    }  
else if(led==9)  
    {  
    digitalWrite(led_sel0, HIGH);  
    digitalWrite(led_sel1, LOW);  
    digitalWrite(led_sel2, LOW);  
    digitalWrite(led_sel3, HIGH);  
    }  
else if(led==10)  
    {  
    digitalWrite(led_sel0, LOW);  
    digitalWrite(led_sel1, HIGH);  
    digitalWrite(led_sel2, LOW);  
    digitalWrite(led_sel3, HIGH);  
    }  
else if(led==11)  
    {  
    digitalWrite(led_sel0, HIGH);  
    digitalWrite(led_sel1, HIGH);  
    digitalWrite(led_sel2, LOW);  
    digitalWrite(led_sel3, HIGH);  
    }  
else if(led==12)  
    {  
    digitalWrite(led_sel0, LOW);  
    digitalWrite(led_sel1, LOW);  
    digitalWrite(led_sel2, HIGH);  
    digitalWrite(led_sel3, HIGH);  
    }  
else if(led==13)  
    {  
    digitalWrite(led_sel0, HIGH);
```

```
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, HIGH);
}
else if(led==14)
{
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, HIGH);
}
else if(led==15)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, HIGH);
}

if(layer==0)
{
digitalWrite(layer_sel0, HIGH);
digitalWrite(layer_sel1, LOW);
digitalWrite(layer_sel2, LOW);
digitalWrite(layer_sel3, LOW);
}
else if(layer==1)
{
digitalWrite(layer_sel0, LOW);
digitalWrite(layer_sel1, HIGH);
digitalWrite(layer_sel2, LOW);
digitalWrite(layer_sel3, LOW);
}
else if(layer==2)
{
digitalWrite(layer_sel0, LOW);
digitalWrite(layer_sel1, LOW);
digitalWrite(layer_sel2, HIGH);
digitalWrite(layer_sel3, LOW);
}
```

```

    }
else if(layer==3)
    {
    digitalWrite(layer_sel0, LOW);
    digitalWrite(layer_sel1, LOW);
    digitalWrite(layer_sel2, LOW);
    digitalWrite(layer_sel3, HIGH);
    }

delay(delay_time);
}

```

```

//*****

```

In the next example, a “fireworks” special effect is produced. The firework goes up, splits into four pieces, and then falls back down as shown in Figure 3.45. It is useful for planning special effects.

```

//*****
//fireworks
//
//This example code is in the public domain.
//*****

```

```

//led select pins
#define led_sel0  23
#define led_sel1  24
#define led_sel2  25
#define led_sel3  26

//layer select pins
#define layer_sel0  27
#define layer_sel1  28
#define layer_sel2  29
#define layer_sel3  30
//*****

```

```

void setup()
{
pinMode(led_sel0, OUTPUT);
pinMode(led_sel1, OUTPUT);

```

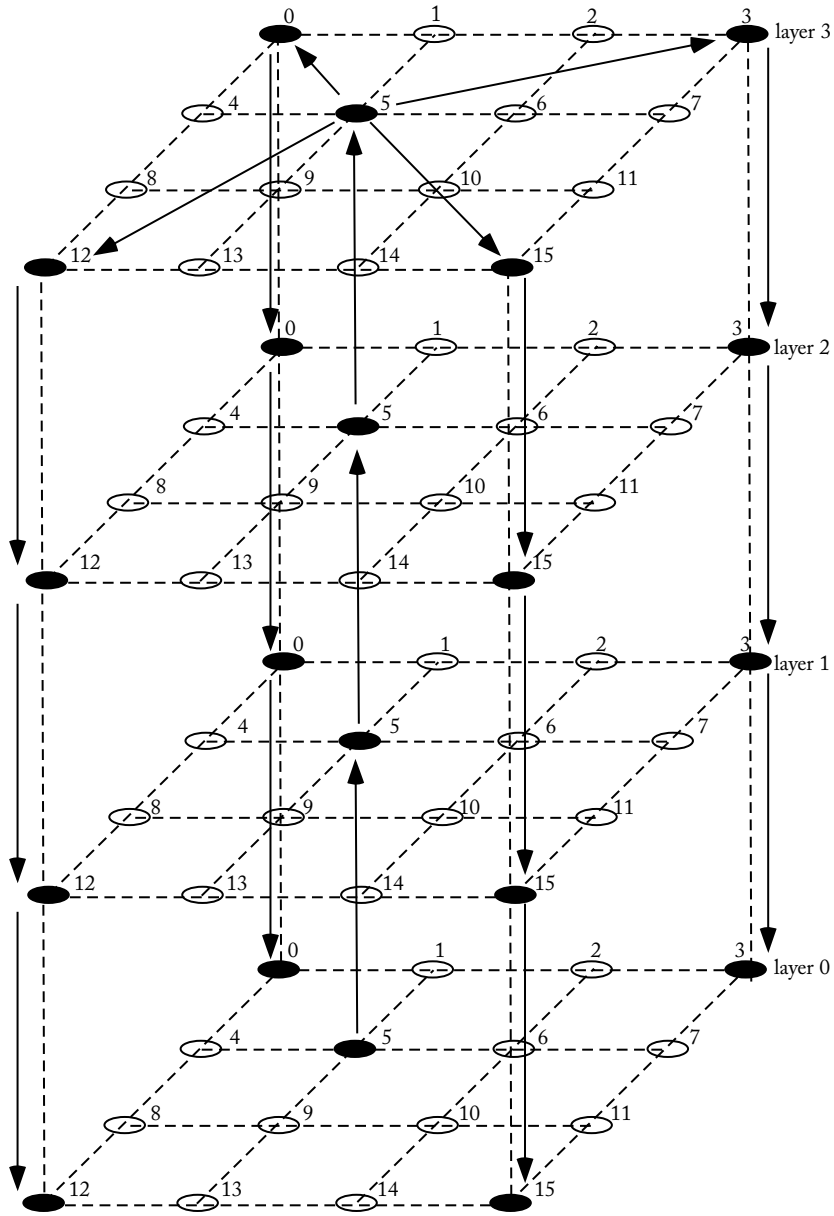


Figure 3.45: LED grid map for a fire work.


```
pinMode(led_sel2, OUTPUT);
pinMode(led_sel3, OUTPUT);

pinMode(layer_sel0, OUTPUT);
pinMode(layer_sel1, OUTPUT);
pinMode(layer_sel2, OUTPUT);
pinMode(layer_sel3, OUTPUT);
}

void loop()
{
int i;

//firework going up
illuminate_LED(5, 0, 100);
illuminate_LED(5, 1, 100);
illuminate_LED(5, 2, 100);
illuminate_LED(5, 3, 100);

//firework exploding into four pieces
//at each cube corner
for(i=0;i<=10;i++)
{
illuminate_LED(0, 3, 10);
illuminate_LED(3, 3, 10);
illuminate_LED(12, 3, 10);
illuminate_LED(15, 3, 10);
delay(10);
}

delay(200);

//firework pieces falling to layer 2
for(i=0;i<=10;i++)
{
illuminate_LED(0, 2, 10);
illuminate_LED(3, 2, 10);
illuminate_LED(12, 2, 10);
illuminate_LED(15, 2, 10);
```

```
    delay(10);
  }

delay(200);

//firework pieces falling to layer 1
for(i=0;i<=10;i++)
  {
  illuminate_LED(0,  1, 10);
  illuminate_LED(3,  1, 10);
  illuminate_LED(12, 1, 10);
  illuminate_LED(15, 1, 10);
  delay(10);
  }

delay(200);

//firework pieces falling to layer 0
for(i=0;i<=10;i++)
  {
  illuminate_LED(0,  0, 10);
  illuminate_LED(3,  0, 10);
  illuminate_LED(12, 0, 10);
  illuminate_LED(15, 0, 10);
  delay(10);
  }

delay(10);
}

//*****

void illuminate_LED(int led, int layer, int delay_time)
{
if(led==0)
  {
  digitalWrite(led_sel0, LOW);
  digitalWrite(led_sel1, LOW);
  digitalWrite(led_sel2, LOW);
```

```
    digitalWrite(led_sel3, LOW);
  }
else if(led==1)
  {
    digitalWrite(led_sel0, HIGH);
    digitalWrite(led_sel1, LOW);
    digitalWrite(led_sel2, LOW);
    digitalWrite(led_sel3, LOW);
  }
else if(led==2)
  {
    digitalWrite(led_sel0, LOW);
    digitalWrite(led_sel1, HIGH);
    digitalWrite(led_sel2, LOW);
    digitalWrite(led_sel3, LOW);
  }
else if(led==3)
  {
    digitalWrite(led_sel0, HIGH);
    digitalWrite(led_sel1, HIGH);
    digitalWrite(led_sel2, LOW);
    digitalWrite(led_sel3, LOW);
  }
else if(led==4)
  {
    digitalWrite(led_sel0, LOW);
    digitalWrite(led_sel1, LOW);
    digitalWrite(led_sel2, HIGH);
    digitalWrite(led_sel3, LOW);
  }
else if(led==5)
  {
    digitalWrite(led_sel0, HIGH);
    digitalWrite(led_sel1, LOW);
    digitalWrite(led_sel2, HIGH);
    digitalWrite(led_sel3, LOW);
  }
else if(led==6)
  {
```

```
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, LOW);
}
else if(led==7)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, HIGH);
digitalWrite(led_sel3, LOW);
}
if(led==8)
{
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, HIGH);
}
else if(led==9)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, LOW);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, HIGH);
}
else if(led==10)
{
digitalWrite(led_sel0, LOW);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, HIGH);
}
else if(led==11)
{
digitalWrite(led_sel0, HIGH);
digitalWrite(led_sel1, HIGH);
digitalWrite(led_sel2, LOW);
digitalWrite(led_sel3, HIGH);
```

```
    }
else if(led==12)
    {
    digitalWrite(led_sel0, LOW);
    digitalWrite(led_sel1, LOW);
    digitalWrite(led_sel2, HIGH);
    digitalWrite(led_sel3, HIGH);
    }
else if(led==13)
    {
    digitalWrite(led_sel0, HIGH);
    digitalWrite(led_sel1, LOW);
    digitalWrite(led_sel2, HIGH);
    digitalWrite(led_sel3, HIGH);
    }
else if(led==14)
    {
    digitalWrite(led_sel0, LOW);
    digitalWrite(led_sel1, HIGH);
    digitalWrite(led_sel2, HIGH);
    digitalWrite(led_sel3, HIGH);
    }
else if(led==15)
    {
    digitalWrite(led_sel0, HIGH);
    digitalWrite(led_sel1, HIGH);
    digitalWrite(led_sel2, HIGH);
    digitalWrite(led_sel3, HIGH);
    }

if(layer==0)
    {
    digitalWrite(layer_sel0, HIGH);
    digitalWrite(layer_sel1, LOW);
    digitalWrite(layer_sel2, LOW);
    digitalWrite(layer_sel3, LOW);
    }
else if(layer==1)
    {
```

3.11. LABORATORY EXERCISE: INTRODUCTION TO THE EDUCATIONAL BOOSTER PACK MKII AND THE

```
digitalWrite(layer_sel0, LOW);
digitalWrite(layer_sel1, HIGH);
digitalWrite(layer_sel2, LOW);
digitalWrite(layer_sel3, LOW);
}
else if(layer==2)
{
digitalWrite(layer_sel0, LOW);
digitalWrite(layer_sel1, LOW);
digitalWrite(layer_sel2, HIGH);
digitalWrite(layer_sel3, LOW);
}
else if(layer==3)
{
digitalWrite(layer_sel0, LOW);
digitalWrite(layer_sel1, LOW);
digitalWrite(layer_sel2, LOW);
digitalWrite(layer_sel3, HIGH);
}

delay(delay_time);
}

//*****
```

3.11 LABORATORY EXERCISE: INTRODUCTION TO THE EDUCATIONAL BOOSTER PACK MKII AND THE GROVE STARTER KIT

Introduction. In this laboratory exercise, we get acquainted with the features of the Educational Booster Pack MkII and the Grove Starter Kit.

Procedure 1: Access the MkII support software available with Energia. Execute the software and interact with the peripherals aboard the MkII. Develop a table of features for the MkII. The table should have column headings for:

- MkII feature,
- MSP432 pins used for interface, and
- notes on interesting aspects.

Procedure 2: Access the Grove Starter Kit support software. Execute the software and interact with the peripherals aboard the Grove. Develop a table of features for the Grove. The table should have column headings for:

- grove feature,
- MSP432 pins used for interface, and
- notes on interesting aspects.

3.12 SUMMARY

In this chapter, we presented the voltage and current operating parameters for the MSP432 microcontroller. We discussed how this information may be applied to properly design an interface for common input and output circuits. It must be emphasized a carefully and properly designed interface allows the microcontroller to operate properly within its parameter envelope. If due to a poor interface design, a microcontroller is used outside its prescribed operating parameter values, spurious and incorrect logic values will result. We provided interface information for a wide range of input and output devices. We also discussed the concept of interfacing a motor to a microcontroller using PWM techniques coupled with high power MOSFET or SSR switching devices.

3.13 REFERENCES AND FURTHER READING

- Barrett, S. (2013) *Arduino Microcontroller Processing for Everyone!* San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/s00283ed1v01y201005dcs029](https://doi.org/10.2200/s00283ed1v01y201005dcs029). 137, 165
- Barrett S. and Pack D. (2011) *Microcontroller Programming and Interfacing Texas Instruments MSP430*. San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/s00340ed1v01y201105dcs033](https://doi.org/10.2200/s00340ed1v01y201105dcs033).
- Barrett, S. and Pack, D. (2006) *Microcontrollers Fundamentals for Engineers and Scientists*. San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/s00025ed1v01y200605dcs001](https://doi.org/10.2200/s00025ed1v01y200605dcs001).
- Crydom Corporation, 2320 Paseo de las Americas, Suite 201, San Diego, CA (www.crydom.com). 159
- Faulkenberry, L. *An Introduction to Operational Amplifiers*, John Wiley & Sons, New York, 1977. 119, 120
- Faulkenberry, L. *Introduction to Operational Amplifiers with Linear Integrated Circuit Applications*, John Wiley & Sons, New York, 1982.
- Electrical Signals and Systems*. Primis Custom Publishing, McGraw-Hill Higher Education, Department of Electrical Engineering, United States Air Force Academy, CO.

Images Company, 39 Seneca Loop, Staten Island, NY 10314. 109

Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002 (www.jameco.com).

Linear Technology, LTC1157 3.3 Dual Micropower High-Side/Low-Side MOSFET Driver.

Milone Technologies eTape liquid level sensors, 17 Ravenswood Way, Sewell, NJ 08080 (www.milonetech.com).

Olimex Limited, 2 Pravda St., P.O.Box 237, Plovdiv 4000 Bulgaria (www.olimex.com).

Power Switching Tools for the Maker and DIY'ers, PowerSwitchTail.com, LLC (www.powerswitchtail.com).

Seeed Grow the Difference, Seeed Technology Limited, F5, Building 8, Shiling Industrial Park, Xinwei, Number32, Tongsha Road Xili Town, Nanshan District, Shenzhen, China. P.R.C 518055 (www.seeedstudio.com).

Shoreline Marine, 1950 Stanley Street Northbrook, IL 60062, (<http://www.shoreline\protect\discretionary{\char\hyphenchar\font}{-}{-1111-marinedevelopment.com/>).

Sick/Stegmann Incorporated, Dayton, OH (www.stegmann.com). 109

SparkFun Electronics, 6333 Dry Creek Parkway, Niwot, CO 80503 (www.sparkfun.com).

The Container Store, (www.containerstore.com).

Texas Instruments (www.ti.com). DOI: 10.1016/0141-9331(82)90083-7.

Texas Instruments 4N25, 4N26, 4N27, 4N28 Optocouplers SOOS035, 1978.

Texas Instruments BOOSTXL-EDUMKII Educational BoosterPack Mark II Plug-in Module SLAU599, 2015. 162

Texas Instruments DRV8829 5-A 45-V Single H-Bridge Motor Driver SLVSA74E, 2010. 142

Texas Instruments SN754410 Quadruple Half-H Driver SLR8007C, 2015.

Texas Instruments H-bridge Motor Controller IC, SLVSA74A, 2010.

Texas Instruments MSP432P401X Mixed-Signal Microcontrollers SLAS826A, 2015. 95, 96

Texas Instruments, TPIC6C596 Power Logic 8-Bit Shift Register (SLIS093D), Texas Instruments, 2015. 98

Texas Instruments TPIC6C596 Power Logic 8-Bit Shift Register TPIC6C596, 2015.

Texas Instruments, SN754410 Quadruple Half-H Driver (SLRS007C), Texas Instruments, 2015.
142

USAFA

3.14 CHAPTER PROBLEMS

Fundamental

1. What will happen if a microcontroller is used outside of its prescribed operating envelope?
2. Discuss the difference between the terms “sink” and “source” as related to current loading of a microcontroller.
3. Can an LED with a series limiting resistor be directly driven by the MSP432 microcontroller? Explain.
4. Describe the UART channels aboard the MSP432.
5. Discuss isolation methods to handle signal inversion.

Advanced

1. In your own words, provide a brief description of each of the microcontroller electrical parameters.
2. What is switch bounce? Describe in detail two techniques to minimize switch bounce.
3. Describe a method of debouncing a keypad.
4. What is the difference between an incremental encoder and an absolute encoder? Describe applications for each type.
5. What must be the current rating of the 2N2222 and 2N2907 transistors used in the tri-state LED circuit? Support your answer.
6. Why are internal pull-up or pull-down resistors help in switch debouncing.
7. Describe the difference between the MSP432 normal and high drive features. Describe how a specific drive feature is selected for a MSP432 pin.
8. A 60 pulse per revolution optical encoder is mounted to a motor shaft. If 125 pulses are counted in 80 ms, what is the speed of the motor in RPM?
9. Derive the output expressions for the classic operational amplifier configurations provided in Figure 3.15.

Challenging

1. Draw the circuit for a six-character seven-segment display. Fully specify all components. Write a program to display “MSP432” on the display.
2. Repeat the question above for a dot matrix display.
3. Repeat the question above for an LCD display.
4. An MSP432 has been connected to a JRP 42BYG016 unipolar, 1.8 degree per step, 12 VDC at 160 mA stepper motor. A 1 s delay is used between the steps to control motor speed. Pushbutton switches SW1 and SW2 are used to assert CW and CCW stepper motion. Write the code to support this application.
5. Describe the operation of the Texas Instrument LSF010XEVM-001 bi-directional multi-voltage level translator evaluation module (LSFEVM).
6. An analog joystick supplied at 3.3 VDC is used to control an underwater ROV. Develop an Energia program to map joystick response to the following nautical directions: fore, aft, port, and starboard.
7. Can the dot matrix display interface be accomplished without the level shifter? Explain.

MSP432 Memory System

Objectives: After reading this chapter, the reader should be able to:

- describe the importance of different memory components in a microcontroller-based system;
- employ the binary and hexadecimal numbering systems to describe the contents or address of a specific memory location;
- specify the length and width of a memory unit;
- describe the function of the address, data, and control buses of a memory unit;
- list the steps required for a memory unit to be read from or written to;
- provide the distinguishing characteristics of random access memory (RAM), read only memory (ROM), and electrically erasable ROM (EEPROM) type memory units;
- explain the concept of a memory map;
- sketch the memory map for the Texas Instruments MSP432 microcontroller;
- list the advantages of employing Direct Memory Access (DMA) techniques to transfer data;
- describe the key features and operation of the MSP432 flash memory controller, the MSP432 RAM controller, and the MSP432 DMA controller; and
- program the MSP432 DMA controller to transfer data between different portions of memory.

4.1 OVERVIEW

In general, there are two purposes for a memory system in a microcontroller. Memory resources serve as the storage and retrieval space for computer instructions and the storage and working space for data. As its name implies, the memory system allows a microcontroller to “remember” the program it is supposed to execute. Various memory components also allow data to be stored and modified during program execution.

The intent of this chapter is to acquaint the reader with basic memory concepts, types of memory, and its operation. The chapter begins with a review of key memory concepts and

terminology. This is followed by a detailed map of the Texas Instruments MSP432 microcontroller memory components. The different memory systems and controllers onboard the MSP432 are then discussed. These include the flash, RAM, flash main memory, flash information memory, and the DMA memory controllers. The chapter concludes with a laboratory exercise detailing the operation of the DMA memory controller.

4.2 BASIC MEMORY CONCEPTS

In this section, we review terminology and concepts associated with microcontroller memory. Figure 4.1a provides a general model for a memory system. A microcontroller's memory system consists of a variety of memory technologies, including random access memory (RAM), several types of read only memory (ROM), and electrically erasable programmable read only memory (EEPROM). Each technology has a specific function within the microcontroller.

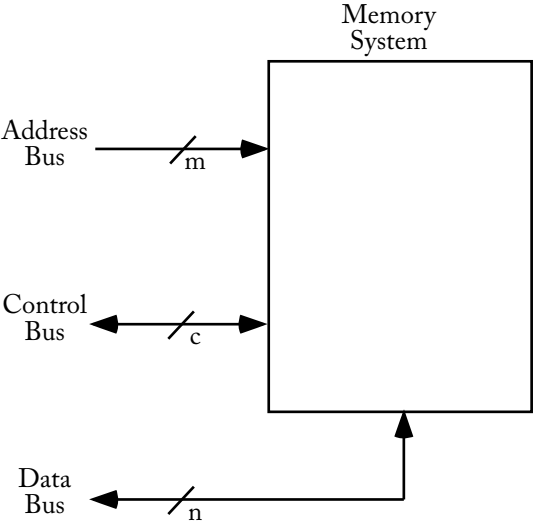
Memory may be viewed as a two-dimensional (2D) array of storage elements called bits. A memory bit can store a single piece of digital information: a logic high or a logic low. Rather than access a single bit for reading or writing, memory is typically configured such that a collection of bits are read or written in parallel. The width, or how many bits are simultaneously accessed, is a function of the specific microcontroller. Memory widths of a byte (8 bits) or a double byte (16 bits) are common. The term "word" is often used to describe the width of the memory system. The MSP432 has a word size of 32 bits. A memory system may be viewed as a series of different memory locations each with a separate and unique address, as shown in Figure 4.1b. At each address is the capacity to store a memory word of n data bits.

4.2.1 MEMORY BUSES

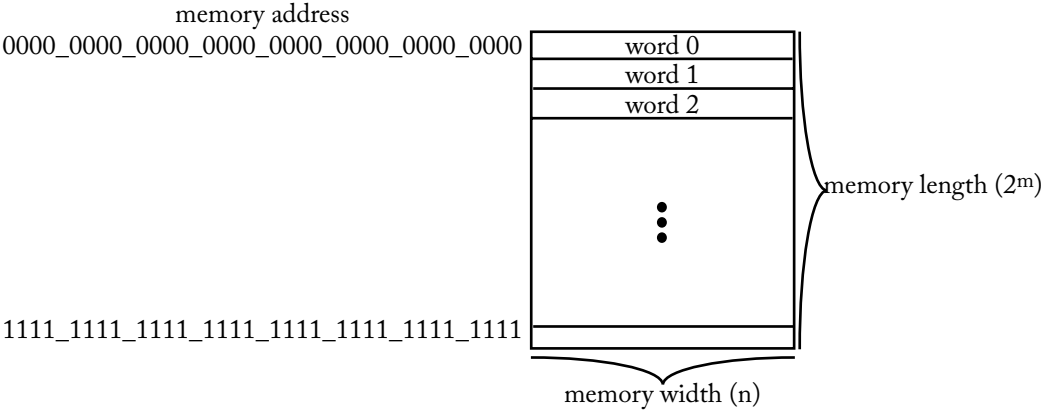
Memory system activities are controlled by several different buses including the address bus, the data bus, and the control bus. A bus is a collection of conductors with a common function. The address bus contains a number (m) of separate address lines. Using linear addressing techniques, the expression $2^{\text{address lines}}$ provides the number of uniquely addressable memory locations. For example, the MSP432 microcontroller is equipped with 32 address lines and, therefore, may separately address 2^{32} or 4,294,967,296 (approximately 4G) different memory locations.

The data bus width (n) usually matches the width of memory or the number of bits stored at each memory location. This allows the contents of a specific memory location to be read from or written to simultaneously. The MSP432 microcontroller has a 32-bit data path. The width of the data path also determines the maximum size of mathematical arguments that can be easily processed by the microcontroller. For example, with a 32-bit data width, the maximum unsigned integer that may be processed without overflow is 2^{32} or 4,294,967,296.

The control bus consists of the signal lines required to perform memory operations such as read and write. There are typically control signals to specify the memory operation (read or write), a clock input, and an enable output for the memory system.



(a) memory system model.



(b) memory length and width.

Figure 4.1: (a) General model for a memory system and (b) memory length and width.

4.2.2 MEMORY OPERATIONS

Operations that are typically performed on a memory system include read from and write to the memory system. The memory read operation consists of the following activities.

- The address of the memory location to be read is provided by the microcontroller on the address bus to the memory system.
- The control signal to read the specified memory location is asserted by the microcontroller.
- The data at the specified memory location is fetched from memory and placed on the memory system data lines.
- The control signal to enable the memory system output is asserted. This allows the fetched memory data access to the data bus.

The memory write operation consists of the following activities.


- The address of the memory location to be written to is provided by the microcontroller on the address bus to the memory system.
- The data to be written to the specified memory location is provided by the microcontroller on the data bus.
- The control signal to write to the specified memory location is asserted.
- The data is written to the specified memory location.

4.2.3 BINARY AND HEXADECIMAL NUMBERING SYSTEMS

The binary, or base 2, numbering system is used to specify addresses and data within microcontroller systems. A review of the binary numbers is provided in Figure 4.2a. The largest unsigned integer that can be specified with a 32 bit binary number is 1111_1111_1111_1111_1111_1111_1111_1111 or 4,294,967,296. Large binary numbers are difficult to read. To help in the readability of lengthy binary numbers, underscores are often inserted between every four bits as shown above.

As previously mentioned, the address bus and the data bus of the MSP432 is 32 bits wide. Rather than specifying large binary numbers, the contents of the address and data bus are often expressed in the hexadecimal numbering system to enhance readability. A review of this system is provided in Figure 4.2b.


To convert from the binary numbering system to the hexadecimal system, binary bits are grouped in fours from either side of the radix point. Each group of four binary bits is represented by its hexadecimal equivalent. Various methods are used to indicate that a specific number is represented in the hexadecimal numbering system. In assembly language, the number is followed by an “h”. In the C programming language the number is preceded by a 0x. Various documentation



32768
16834
8192
4096
2048
1024
512
256
128
64
32
16
8
4
2
1


2¹⁵ 2¹⁴ 2¹³ 2¹² 2¹¹ 2¹⁰ 2⁹ 2⁸ 2⁷ 2⁶ 2⁵ 2⁴ 2³ 2² 2¹ 2⁰

(a) binary number system.



binary	hex	binary	hex	binary	hex	binary	hex
0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

(b) hexadecimal number system.



32768 = 8000h = 1000_0000_0000_0000
16834 = 4000h = 0100_0000_0000_0000
8192 = 2000h = 0010_0000_0000_0000
4096 = 1000h = 0001_0000_0000_0000
2048 = 0800h = 0000_1000_0000_0000
1024 = 0400h = 0000_0100_0000_0000
512 = 0200h = 0000_0010_0000_0000
256 = 0100h = 0000_0001_0000_0000
128 = 0080h = 0000_0000_1000_0000
64 = 0040h = 0000_0000_0100_0000
32 = 0020h = 0000_0000_0010_0000
16 = 0010h = 0000_0000_0001_0000
8 = 0008h = 0000_0000_0000_1000
4 = 0004h = 0000_0000_0000_0100
2 = 0002h = 0000_0000_0000_0010
1 = 0001h = 0000_0000_0000_0001

△ 2¹⁵ 2¹⁴ 2¹³ 2¹² 2¹¹ 2¹⁰ 2⁹ 2⁸ 2⁷ 2⁶ 2⁵ 2⁴ 2³ 2² 2¹ 2⁰ △

(c) binary to hexadecimal conversion.

Figure 4.2: (a) The binary numbering system, (b) the hexadecimal numbering system, and (c) conversion from binary to hexadecimal.

sources will place a dollar sign (\$) before the numerical value or a subscripted 16 by the number indicating the hexadecimal content.

Examples:

1. Some microcontrollers have 16 address lines, giving it the capability to separately address 65,536 different memory locations. What is the address of the first and last memory location expressed in hexadecimal?

Answer: The first and last addressable memory locations expressed in hexadecimal notation are $(0000)_{16}$ and $(ffff)_{16}$.

2. Express the hexadecimal number $(cf)_{16}$ in binary.

Answer: Each hexadecimal value is converted into its four bit binary equivalent resulting in the value of $(1100_1111)_2$

3. The MSP432 has the value of $(0001_1010_1111_1100_0001_1010_1111_1100)_2$ present on the data base. Express the value in hexadecimal.

Answer: Each group of four binary bits is expressed with its corresponding hexadecimal equivalent resulting in $(1afc1afc)_{16}$.

4.2.4 MEMORY ARCHITECTURES

There are two basic types of computer architectures based on the memory organization: the von Neumann architecture and the Harvard architecture. The von Neumann architecture has computer instructions and data resident within the same memory system, whereas, the Harvard architecture uses separate units to store instructions and data. The MSP432 microcontroller employs the von Neumann type architecture since data and instructions are both retained within the same memory component.

4.2.5 MEMORY TYPES

Memory systems typically use several different types of memory technologies. Each technology type has its inherent advantages and disadvantages. We briefly describe each type.

RAM

Random access memory (RAM) is volatile. That is, it only retains its memory contents while power is present. Within a microcontroller system, RAM memory is used for storing global variables, local variables, stack implementation, and the dynamic allocation of user-defined data types during program execution. Variants of the MSP432 hosts onboard RAM memory of up to 64 Kbytes [SLAS826A, 2015].

ROM

Read only memory (ROM) is non-volatile. That is, a ROM memory retains its contents even when power is lost. A ROM variant, EEPROM (for electrically erasable programmable read

only memory), is often referred to as flash memory. There are two main variants of flash memory onboard the MSP432: flash main memory and flash information memory. Flash main memory is used to store programs and system constants that must be retained when system power is lost (non-volatile memory). Flash information memory contains data used for program execution. Variants of the MSP432 host flash main memory up to 256 Kbytes and flash information memory up to 16 Kbytes. The 256 kbytes of flash memory allow for substantial program development. Provided below is a list of MSP432 variants with associated flash main memory and RAM size [SLAS826A, 2015].

- MSP432P401RIPZ, flash main memory: 256 Kbytes, RAM: 64 Kbytes
- MSP432P401MIPZ, flash main memory: 128 Kbytes, RAM: 32 Kbytes
- MSP432P401RIZXH, flash main memory: 256 Kbytes, RAM: 64 Kbytes
- MSP432P401MIZXH, flash main memory: 128 Kbytes, RAM: 32 Kbytes
- MSP432P401RIRGC, flash main memory: 256 Kbytes, RAM: 64 Kbytes
- MSP432P401MIRGC, flash main memory: 128 Kbytes, RAM: 32 Kbytes

The MSP-EXP432P401R LaunchPad is equipped with the MSP432P401RIPZ processor [SLAU597A, 2015].

External Memory Components

A microcontroller's memory system may also be enhanced or extended using external memory components. For example, bulk storage capability may be added to a microcontroller-based system by interfacing a Multi Media Card/Secure Digital (MMC/SD) card. The MMC/SD card is equipped with a large complement of flash memory. The MMC/SD card is interfaced to the microcontroller via a serial communication link. The MMC/SD card is typically housed in a socket for easy removal from the host microcontroller-based system [SanDisk, 2000]. With an MMC/SD card, data may be logged over a long period of time. The MMC/SD card may then be removed from the microcontroller-based system and read by a personal computer (PC) to examine and analyze the data.

Examples: A microcontroller-based application is being developed to log wind data at various remote locations over long periods of time to determine the efficacy of a wind energy farm at a remote site. Answer the following questions based on this scenario.

1. The algorithm to store the data is fairly complex and will require much storage space. What memory component must you use to adequately hold the algorithm?

Answer: The coded algorithm to control the data collection system is stored in flash main memory. An MSP432 variant must be chosen that has sufficient flash memory capacity to hold the algorithm.

2. A good design technique is to compartmentalize specific algorithm operations into subroutines or functions. What memory component is required to support the call to subroutines or functions?

Answer: When a subroutine or function is called, local variables are placed on the stack. The stack is typically implemented as a portion of RAM memory.

3. The data logging system will be dispersed at a number of locations on existing farms and ranches. The plan is to collect the data over a six-month period and then have the property owner transfer the data to a central facility for processing. What is the appropriate memory technology to use in this situation?

Answer: A microcontroller-based data collection system equipped with a removable MMC/SD card would be a good choice in this situation. The data could be collected for a long period of time, and the MMC/SD card could then be removed and sent to the central facility.

4. How do you determine the required capacity for the MMC/SD card to log data over a six-month period?

Answer: To determine the required memory capacity the following parameters must be considered:

- How many data variables are collected (e.g., date, time, temperature, wind speed, altitude) at a time?
- In what format will the data be stored (e.g., integers, floating point numbers, custom abstract data type such as a record)?
- How often will data be collected (e.g., every 15 minutes, hourly, every 6 hours, daily)?
- Over what period of time will data be collected?

4.3 MEMORY OPERATIONS IN C USING POINTERS

Before delving into a detailed look at the MSP432 microcontroller's memory system, we need to discuss the concept of pointers in the C language. Pointer syntax allows one to easily refer to a memory location's address and the data contained at the address.¹

A pointer is another name for an address. To declare a pointer, an asterisk (*) is placed in front of the variable name. The compiler will designate the variable as an address to the variable type specified.

Shown below is the syntax to declare an integer and a pointer (address of) for an integer. It is helpful to choose a variable name that helps you remember that a pointer variable has been declared. A pointer may be declared as a global or local variable.

¹The information on pointers and examples provided were adapted from Dr. Jerry Cupal's, EE4390 Microprocessors class notes at the University of Wyoming.

```
int    x;
int    *ptr_x;
```

Once a pointer has been declared, there is other syntax that may be used in the body of a program to help manipulate memory addresses and data. The ampersand (&) is used to reference the address of a variable, whereas the asterisk (*) is used as a dereference operator to refer to the contents of a memory location that a pointer is referencing.

Example 1: Given the code snapshot below, what is the final value in variable n?

```
//*****

int    m,n;        //declare integers m and n
int    *ptr_m;     //declare pointer to integer type

m = 10;           //set integer m equal to 10
ptr_m = &m        //set integer pointer to address of integer m
n = *ptr_m;       //Note use of the dereference operator

//*****
```

Answer: The final value of n will be 10. The dereference operator in the last line of code refers to the contents of the memory location referenced.

Example 2: In this example, a technique is provided to point to a location in memory space.

```
//*****
char   *ptr_mem;   //configure a pointer to 8 bit locations

:
:

ptr_mem = (char*) 0x01000000; //cast the number 01000000h into a pointer
                               //that points to character locations

//*****
```

Example 3: This example shows how a pointer may be used to move about memory locations by using some basic mathematical operations.

```
//*****
// This function fills a buffer located between memory locations 4000h
// 5FFFh in memory space with incrementing 32 bit numbers.
```

```

The values
// loaded into memory start with 0000h and increments up to 0FFFh.
//*****

int      x;                //integer variable x
int      *ptr_buffer;     //pointer to buffer

void main( )
{
ptr_buffer = (int*) 0x4000; //cast a pointer equal to 4000
for(x=0x0000; x<=0x0fff; x++)
  {
  *ptr_buffer = x;        //move the variable x into buffer
  ptr_buffer++;         //increment the pointer
  }
}
//*****

```

4.4 MEMORY MAP

A memory map is a visualization tool used to map the memory system onboard a microcontroller. As previously mentioned, the MSP432 has a 32 bit memory address. This allows the microcontroller to span the address memory space from $(00000000)_{16}$ to $(ffffff)_{16}$. Although the microcontroller may span this space, it does not necessarily mean there are memory units installed at each and every location. A memory map shows which addresses in memory are occupied by a specific memory component and what locations are currently available for connection to other devices. The memory map for the MSP432 microcontroller is shown in Figure 4.3.

As can be seen in Figure 4.3, there are a variety of memory units using different technologies within the MSP432 memory map. For each memory component, the start and stop addresses are provided as well as the address span on the memory component. The span is provided as the number of locations in hexadecimal, decimal, and rounded off to the nearest byte. Additional details of each memory unit is provided moving in the figure from left to right.

4.5 FLASH MEMORY

In the MSP432, Flash memory is divided into Main Memory and Information Memory. As can be seen in the MSP432 memory map (Figure 4.3), the size of Flash Main Memory is 256 KB and it spans the memory addresses from $(00000000)_{16}$ to $(0003FFFF)_{16}$. This memory space is subdivided into 64 sectors of 4 KB each. Bulk operations on Flash Main Memory are performed in 4 KB blocks. The 256 KB Flash Main Memory is organized into two separate, independent

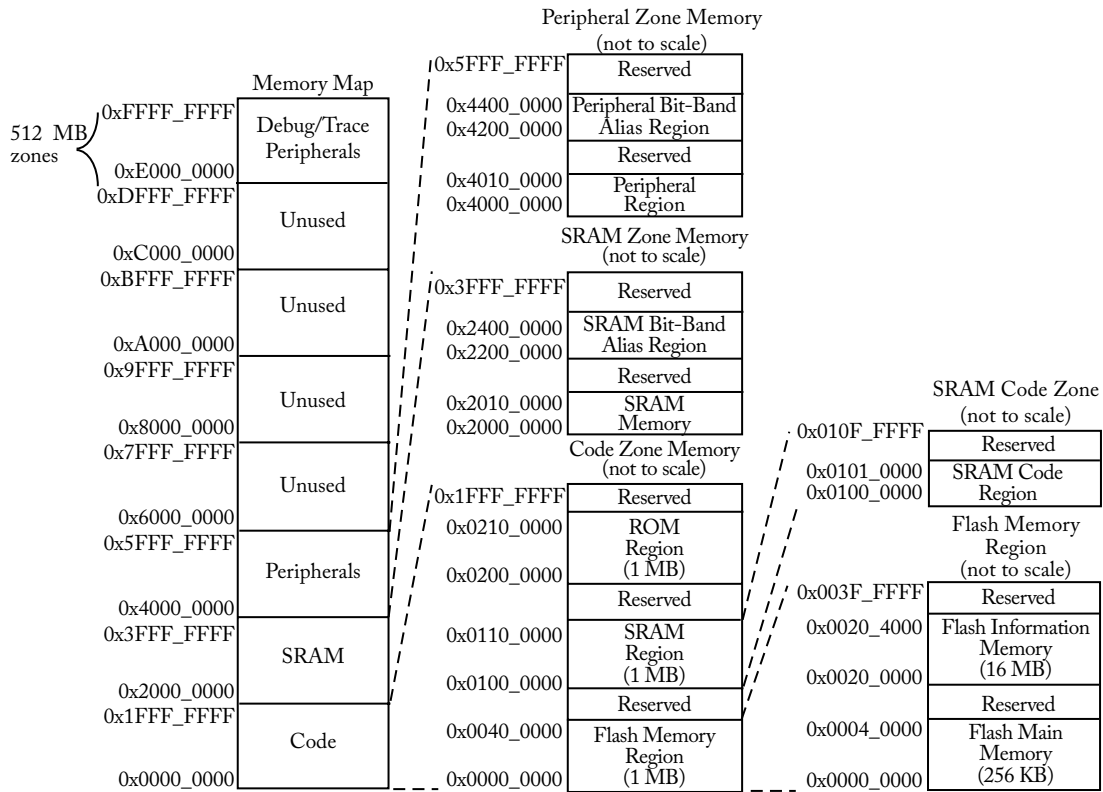


Figure 4.3: The MSP432 memory map [SLAS826A, 2015].

128 KB banks. This allows for simultaneous operations on both banks. Flash Main memory is primarily used for user application code and data [SLAU356A, 2015, SLAS826A, 2015].

Flash Information Memory spans the location from $(00200000)_{16}$ to $(0020FFFF)_{16}$. The 16 KB space is subdivided into 4 sectors of 4 KB each. The 16 KB Flash Information Memory is organized into two separate, independent 8 KB banks. This allows for simultaneous operations on both banks. The Flash Information Memory is primarily reserved, but can be used to store user data [SLAU356A, 2015, SLAS826A, 2015].

The Flash Controller (FLCTL) serves as the interface between the software and Flash memory assets. It controls operations such as memory read and write, memory write and erase protection, and auto-verify features. FLCTL operations are controlled by dedicated registers [SLAU356A, 2015]. Registers include:

- FLCTL_POWER_STAT Power Status Register
- FLCTL_BANK0_RDCTL Bank0 Read Control Register

- FLCTL_BANK1_RDCTL Bank1 Read Control Register
- FLCTL_RDBRST_CTLSTAT Read Burst/Compare Control and Status Register
- FLCTL_RDBRST_STARTADDR Read Burst/Compare Start Address Register
- FLCTL_RDBRST_LEN Read Burst/Compare Length Register
- FLCTL_RDBRST_FAILADDR Read Burst/Compare Fail Address Register
- FLCTL_RDBRST_FAILCNT Read Burst/Compare Fail Count Register
- FLCTL_PRG_CTLSTAT Program Control and Status Register
- FLCTL_PRGBRST_CTLSTAT Program Burst Control and Status Register
- FLCTL_PRGBRST_STARTADDR Program Burst Start Address Register
- FLCTL_PRGBRST_DATA0_0 Program Burst Data0 Register0
- FLCTL_PRGBRST_DATA0_1 Program Burst Data0 Register1
- FLCTL_PRGBRST_DATA0_2 Program Burst Data0 Register2
- FLCTL_PRGBRST_DATA0_3 Program Burst Data0 Register3
- FLCTL_PRGBRST_DATA1_0 Program Burst Data1 Register0
- FLCTL_PRGBRST_DATA1_1 Program Burst Data1 Register1
- FLCTL_PRGBRST_DATA1_2 Program Burst Data1 Register2
- FLCTL_PRGBRST_DATA1_3 Program Burst Data1 Register3
- FLCTL_PRGBRST_DATA2_0 Program Burst Data2 Register0
- FLCTL_PRGBRST_DATA2_1 Program Burst Data2 Register1
- FLCTL_PRGBRST_DATA2_2 Program Burst Data2 Register2
- FLCTL_PRGBRST_DATA2_3 Program Burst Data2 Register3
- FLCTL_PRGBRST_DATA3_0 Program Burst Data3 Register0
- FLCTL_PRGBRST_DATA3_1 Program Burst Data3 Register1
- FLCTL_PRGBRST_DATA3_2 Program Burst Data3 Register2
- FLCTL_PRGBRST_DATA3_3 Program Burst Data3 Register3

- FLCTL_ERASE_CTLSTAT Erase Control and Status Register
- FLCTL_ERASE_SECTADDR Erase Sector Address Register
- FLCTL_BANK0_INFO_WEPROT Information Memory Bank0 Write/Erase Protection Register
- FLCTL_BANK0_MAIN_WEPROT Main Memory Bank0 Write/Erase Protection Register
- FLCTL_BANK1_INFO_WEPROT Information memory Bank1 Write/Erase Protection Register
- FLCTL_BANK1_MAIN_WEPROT Main Memory Bank1 Write/Erase Protection Register
- FLCTL_BMRK_CTLSTAT Benchmark Control and Status Register
- FLCTL_BMRK_IFETCH Benchmark Instruction Fetch Count Register
- FLCTL_BMRK_DREAD Benchmark Data Read Count Register
- FLCTL_BMRK_CMP Benchmark Count Compare Register
- FLCTL_IFG Interrupt Flag Register
- FLCTL_IE Interrupt Enable Register
- FLCTL_CLRIFG Clear Interrupt Flag Register
- FLCTL_SETIFG Set Interrupt Flag Register
- FLCTL_READ_TIMCTL Read Timing Control Register
- FLCTL_READMARGIN_TIMCTL Read Margin Timing Control Register
- FLCTL_PRGVER_TIMCTL Program Verify Timing Control Register
- FLCTL_ERSVER_TIMCTL Erase Verify Timing Control Register
- FLCTL_LKGVVER_TIMCTL Leakage Verify Timing Control Register
- FLCTL_PROGRAM_TIMCTL Program Timing Control Register
- FLCTL_ERASE_TIMCTL Erase Timing Control Register
- FLCTL_MASSERASE_TIMCTL Mass Erase Timing Control Register
- FLCTL_BURSTPRG_TIMCTL Burst Program Timing Control Register

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [[SLAU356A](#), 2015] and will not be repeated here.

4.5.1 FLCTL DRIVELIB SUPPORT

Texas Instruments provides extensive MSP432 FLCTL support through a series of Application Program Interfaces (APIs). Provided below is a list of FLCTL APIs. Details on API settings are provided in MSP432 Peripheral Driver Library User's Guide [[DriverLib, 2015](#)] and will not be repeated here.

- void FlashCtl_clearInterruptFlag (uint32_t flags)
- void FlashCtl_clearProgramVerification (uint32_t verificationSetting)
- void FlashCtl_disableInterrupt (uint32_t flags)
- void FlashCtl_disableReadBuffering (uint_fast8_t memoryBank, uint_fast8_t accessMethod)
- void FlashCtl_disableWordProgramming (void)
- void FlashCtl_enableInterrupt (uint32_t flags)
- void FlashCtl_enableReadBuffering (uint_fast8_t memoryBank, uint_fast8_t accessMethod)
- void FlashCtl_enableWordProgramming (uint32_t mode) bool FlashCtl_eraseSector (uint32_t addr)
- uint32_t FlashCtl_getEnabledInterruptStatus (void)
- uint32_t FlashCtl_getInterruptStatus (void)
- uint32_t FlashCtl_getReadMode (uint32_t flashBank)
- uint32_t FlashCtl_getWaitState (uint32_t bank)
- bool FlashCtl_isSectorProtected (uint_fast8_t memorySpace, uint32_t sector)
- uint32_t FlashCtl_isWordProgrammingEnabled (void)
- bool FlashCtl_performMassErase (void)
- bool FlashCtl_programMemory (void *src, void *dest, uint32_t length)
- bool FlashCtl_protectSector (uint_fast8_t memorySpace, uint32_t sectorMask)
- void FlashCtl_registerInterrupt (void(*intHandler)(void))
- void FlashCtl_setProgramVerification (uint32_t verificationSetting)
- bool FlashCtl_setReadMode (uint32_t flashBank, uint32_t readMode)

- void FlashCtl_setWaitState (uint32_t bank, uint32_t waitState)
- bool FlashCtl_unprotectSector (uint_fast8_t memorySpace, uint32_t sectorMask)
- void FlashCtl_unregisterInterrupt (void)
- bool FlashCtl_verifyMemory (void *verifyAddr, uint32_t length, uint_fast8_t pattern)

Example 4: Texas Instruments provides a number of MSP432 examples in DriverLib available within Code Composer Studio. Several FLCTL examples are provided including the one shown below. This example demonstrates the mass erase API.

```

//*****
//MSP432 DriverLib - v01_04_00_18
//
//--COPYRIGHT--,BSD,BSD
//Copyright (c) 2015, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
// - Redistributions of source code must retain the above copyright
//   notice, this list of conditions and the following disclaimer.
// - Redistributions in binary form must reproduce the above copyright
//   notice, this list of conditions and the following disclaimer in the
//   documentation and/or other materials provided with the distribution.
// - Neither the name of Texas Instruments Incorporated nor the names of
//   its contributors may be used to endorse or promote products derived
//   from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```

```

//*****
//MSP432 Flash Controller - Mass Erase
//Description: This example shows the functionality of the mass erase
//API in DriverLib.
At the start of the example, flash bank 1 sectors
//30 and 31 are unprotected (0x3E000 - 0x3FFFF) and then programmed with
//filler data.
Sector 31 is then protected and a mass erase is performed.
//Since the mass erase will only erase unprotected sectors, after the
//mass erase finishes only the data in sector 30 will be erased (as can
//be observed through the memory browser in the debugger).
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>
#include <stdbool.h>
#include <string.h>

#define BANK1_S30 0x3E000

//Statics
uint8_t patternArray[8192];

int main(void)
{
//Since this program has a huge buffer that simulates the calibration
//data, halting the watch dog is done in the reset ISR to avoid a
//watchdog timeout during the zero

//Setting our MCLK to 48MHz for faster programming
MAP_PCM_setCoreVoltageLevel(PCM_VCORE1);
MAP_CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_48);

//Initializing buffer to a pattern of 0xA5

```

```
memset(patternArray, 0xA5, 8192);

//Unprotecting User Bank 1, Sectors 30 and 31
MAP_FlashCtl_unprotectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,
                             FLASH_SECTOR30 | FLASH_SECTOR31);

//Trying a mass erase.
Since we unprotected User Bank 1, sectors 31
//and 32, this should erase these sectors.
If it fails, we trap inside
//an infinite loop.
if(!MAP_FlashCtl_performMassErase())
    while(1);

//Trying to program the filler data.
If it fails, trap inside an
//infinite loop
if(!MAP_FlashCtl_programMemory (patternArray, (void*) BANK1_S30, 8192))
    while(1);

//Setting sector 31 back to protected
MAP_FlashCtl_protectSector(FLASH_MAIN_MEMORY_SPACE_BANK1,FLASH_SECTOR31);

//Performing the mass erase again.
Now, since we protected Sector31,
//only Sector 30 (0x3E000 - 0x3EFFF) should be erased.
Set a breakpoint
//after this call to observe the memory in the debugger.
if(!MAP_FlashCtl_performMassErase())
    while(1);

//Deep Sleeping when not in use
while (1)
{
    MAP_PCM_gotoLPM3();
}
}
//*****
```

4.6 DIRECT MEMORY ACCESS (DMA)

Direct memory access (DMA) provides the capability to move data from a source memory location to a destination memory location without involving the central processing unit (CPU). This is especially useful for low power operation when moving data from peripherals to specific memory locations [SLAU208G, 2010]. DMA may be likened to the highway bypass arteries around major cities. This allows the traveler to reach a destination in a timely manner without going through the heart of a city. Similarly, DMA provides for the rapid transmission of data from source to destination without involving the CPU.

In this section we briefly introduce the flexible and powerful MSP432 DMA system. We begin with an overview of system specifications, followed by a description different types of available transfers. We then review MSP432 registers and driverlib API support used to configure DMA transfers. We conclude with an illustrative example.

4.6.1 DMA SPECIFICATIONS

The MSP432 DMA is a flexible and powerful subsystem to enable fast data transfers without involving the CPU. It provides for eight channels of simultaneous transfers from specific sources and destinations within the MSP432, including MSP432 peripherals and memory locations. Figure 4.4 shows a summary of MSP432 sources. A specific source is specified by the Source Configuration Register (SRCCFG) [SLAS826A, 2015].

	SRCCFG=0	SRCCFG=1	SRCCFG=2	SRCCFG=3	SRCCFG=4	SRCCFG=5	SRCCFG=6	SRCCFG=7
Channel 0	Reserved	eUSCI_A0 TX	eUSCI_B0 TX	eUSCI_B3 TX1	eUSCI_B2 TX2	eUSCI_B1 TX3	TA0CCR0	AES256_Trigger0
Channel 1	Reserved	eUSCI_A0 RX	eUSCI_B0 RX	eUSCI_B3 RX1	eUSCI_B2 RX2	eUSCI_B1 RX3	TA0CCR2	AES256_Trigger0
Channel 2	Reserved	eUSCI_A1 TX	eUSCI_B1 TX	eUSCI_B0 TX1	eUSCI_B3 TX2	eUSCI_B2 TX3	TA1CCR0	AES256_Trigger0
Channel 3	Reserved	eUSCI_A1 RX	eUSCI_B1 RX	eUSCI_B0 RX1	eUSCI_B3 RX2	eUSCI_B2 RX3	TA1CCR2	Reserved
Channel 4	Reserved	eUSCI_A2 TX	eUSCI_B2 TX	eUSCI_B1 TX1	eUSCI_B0 TX2	eUSCI_B3 TX3	TA2CCR0	Reserved
Channel 5	Reserved	eUSCI_A2 RX	eUSCI_B2 RX	eUSCI_B1 RX1	eUSCI_B0 RX2	eUSCI_B3 RX3	TA2CCR2	Reserved
Channel 6	Reserved	eUSCI_A3 TX	eUSCI_B3 TX	eUSCI_B2 TX1	eUSCI_B1 TX2	eUSCI_B0 TX3	TA3CCR0	DMAEO
Channel 7	Reserved	eUSCI_A3 RX	eUSCI_B3 RX	eUSCI_B2 RX1	eUSCI_B1 RX2	eUSCI_B0 RX3	TA0CCR2	(external pin) ADC14

ADC: analog-to-digital conversion
eUSCI_A0 - A3: UART IrDA, SPI
eUSCI_B0 - B3: I2C, SPI
TA0 - TA3: timers
AES: security encryption/decryption
DMA: direct memory access

Figure 4.4: MSP432 DMA sources [SLAS826A, 2015].

The specific width of the DMA transfer may be set for 8, 16, or 32 bits, depending on the type of data being transferred. Whichever width is selected, both source and destination widths must be equal. The number of data items per transfer may be set from 1–1024. The details of a DMA transfer is specified using the DMA associated registers [SLAS826A, 2015].

4.6.2 DMA TRANSFER TYPES

The MSP432 has several different transfer types available. A brief definition of each follows. Additional technical detail of transfer types are provided in [SLAU356A \[2015\]](#), [SLAS826A \[2015\]](#).

- **Invalid:** The invalid DMA transfer type is used to signal the end of a DMA transfer. It prevents the DMA system from repeating a completed transfer [[SLAU356A, 2015](#)].
- **Auto-request:** The auto-request DMA transfer type is used for large DMA transfers. Once enabled the specified transfer will run to completion [[SLAU356A, 2015](#)].
- **Ping-Pong:** In the game of ping-pong, a ball is volleyed back and forth between two sides of the table. This also describes this type of DMA transfer. Data from a source is alternately deposited into two different destinations. The first batch of data is transferred to the first destination. While the second batch of data is being transferred to the second destination, while the first batch of data may be processed by the CPU. The process can repeat [[SLAU356A, 2015](#)].
- **Memory Scatter-Gather:** In the memory scatter-gather DMA transfer, *data from memory* is transferred in four packets from memory to two different data structures designated as primary and secondary. Continued transfers alternate between the primary and secondary structures [[SLAU356A, 2015](#)].
- **Peripheral Scatter-Gather:** In the peripheral scatter-gather DMA transfer, *data from a designated peripheral* is transferred in four packets from memory to two different data structures, designated as primary and secondary. Continued transfers alternate between the primary and secondary structures [[SLAU356A, 2015](#)].
- **Basic:** The basic DMA transfer is shown in Figure 4.5. When the DMA is enabled, it will poll for any active channel requests. If a request is active, the DMA system will then check to see if there is a higher priority request pending. If a higher priority request (lower number) is pending, the higher priority DMA transfer will start. If not, the DMA request with the lowest number will start.

4.6.3 DMA REGISTERS

The DMA system is controlled by register settings. A list of registers supporting the DMA system is provided below. Register setting details are provided in [[SLAU356A, 2015](#)]. We illustrate how to use the registers in an upcoming example.

- **DMA_DEVICE_CFG:** Device Configuration Status Register
- **DMA_SW_CHTRIG:** Software Channel Trigger Register

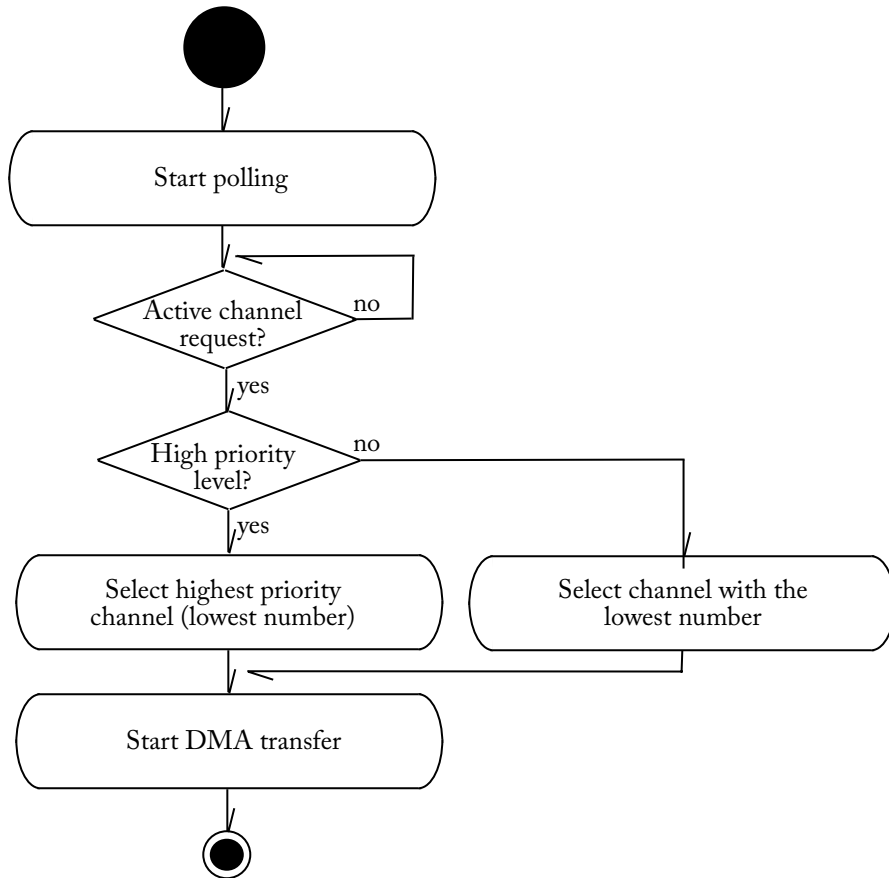


Figure 4.5: MSP432 basic DMA transfer [SLAU356A, 2015].

- **DMA_CHn_SRCCFG:** Channel n Source Configuration Register. Note: n = 0 to NUM_DMA_CHANNELS
- **DMA_INT1_SRCCFG:** Interrupt 1 Source Channel Configuration Register
- **DMA_INT2_SRCCFG:** Interrupt 2 Source Channel Configuration n Register
- **DMA_INT3_SRCCFG:** Interrupt 3 Source Channel Configuration Register
- **DMA_INT0_SRCFLG:** Interrupt 0 Source Channel Flag Register
- **DMA_INT0_CLRFLG:** Interrupt 0 Source Channel Clear Flag Register
- **DMA_STAT:** Status Register

- **DMA_CFG:** Configuration Register
- **DMA_CTLBASE:** Channel Control Data Base Pointer Register
- **DMA_ALTBASE:** Channel Alternate Control Data Base Pointer Register
- **DMA_WAITSTAT:** Channel Wait on Request Status Register
- **DMA_SWREQ:** Channel Software Request Register
- **DMA_USEBURSTSET:** Channel Useburst Set Register
- **DMA_USEBURSTCLR:** Channel Useburst Clear Register
- **DMA_REQMASKSET:** Channel Request Mask Set Register
- **DMA_REQMASKCLR:** Channel Request Mask Clear Register
- **DMA_ENASET:** Channel Enable Set Register
- **DMA_ENACLAR:** Channel Enable Clear Register
- **DMA_ALTSET:** Channel Primary-Alternate Set Register
- **DMA_ALTCLR:** Channel Primary-Alternate Clear Register
- **DMA_PRIOSET:** Channel Priority Set Register
- **DMA_PRIOCLR:** Channel Priority Clear Register
- **DMA_ERRCLR:** Bus Error Clear Register

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [[DriverLib, 2015](#)] and will not be repeated here.

4.6.4 DMA DRIVELIB SUPPORT

Texas Instruments provides extensive MSP432 DMA support through a series of Application Program Interfaces (APIs). Provided below is a list of DMA APIs. Details on API settings are provided in *MSP432 Peripheral Driver Library User's Guide* [[DriverLib, 2015](#)] and will not be repeated here.

- `#define DMA_TaskStructEntry(transferCount, itemSize, srcIncrement, srcAddr, dstIncrement, dstAddr, arbSize, mode)`
- `void DMA_assignChannel (uint32_t mapping)`
- `void DMA_assignInterrupt (uint32_t interruptNumber, uint32_t channel)`

- void DMA_clearErrorStatus (void)
- void DMA_clearInterruptFlag (uint32_t intChannel)
- void DMA_disableChannel (uint32_t channelNum)
- void DMA_disableChannelAttribute (uint32_t channelNum, uint32_t attr)
- void DMA_disableInterrupt (uint32_t interruptNumber)
- void DMA_disableModule (void)
- void DMA_enableChannel (uint32_t channelNum)
- void DMA_enableChannelAttribute (uint32_t channelNum, uint32_t attr)
- void DMA_enableInterrupt (uint32_t interruptNumber)
- void DMA_enableModule (void)
- uint32_t DMA_getChannelAttribute (uint32_t channelNum)
- uint32_t DMA_getChannelMode (uint32_t channelStructIndex)
- uint32_t DMA_getChannelSize (uint32_t channelStructIndex)
- void *DMA_getControlAlternateBase (void)
- void *DMA_getControlBase (void)
- uint32_t DMA_getErrorStatus (void)
- uint32_t DMA_getInterruptStatus (void)
- bool DMA_isChannelEnabled (uint32_t channelNum)
- void DMA_registerInterrupt (uint32_t intChannel, void(*intHandler)(void))
- void DMA_requestChannel (uint32_t channelNum)
- void DMA_requestSoftwareTransfer (uint32_t channel)
- void DMA_setChannelControl (uint32_t channelStructIndex, uint32_t control)
- void DMA_setChannelScatterGather (uint32_t channelNum, uint32_t taskCount, void *taskList, uint32_t isPeriphSG)
- void DMA_setChannelTransfer (uint32_t channelStructIndex, uint32_t mode, void *srcAddr, void *dstAddr, uint32_t transferSize) void DMA_setControlBase (void *controlTable) void DMA_unregisterInterrupt (uint32_t intChannel)

4.6.5 DMA EXAMPLE

Texas Instruments provides a number of MSP432 examples in DriverLib available within Code Composer Studio. Several DMA examples are provided including the one below. This example routes data quickly to the CRC32 module via the DMA controller. The CRC32 system is used to calculate a cyclic redundancy check (CRC) checksum. Additional discussion on this system will follow. Provided in Figure 4.6 is the UML activity diagram for the example.

Example 5: This example demonstrates the DMA on the MSP432.

```
//*****
//MSP432 DriverLib - v01_04_00_18
//
//--COPYRIGHT--,BSD,BSD
//Copyright (c) 2015, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
// - Redistributions of source code must retain the above copyright
//   notice, this list of conditions and the following disclaimer.
// - Redistributions in binary form must reproduce the above copyright
//   notice, this list of conditions and the following disclaimer in the
//   documentation and/or other materials provided with the distribution.
// - Neither the name of Texas Instruments Incorporated nor the names of
//   its contributors may be used to endorse or promote products derived
//   from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//*****
```

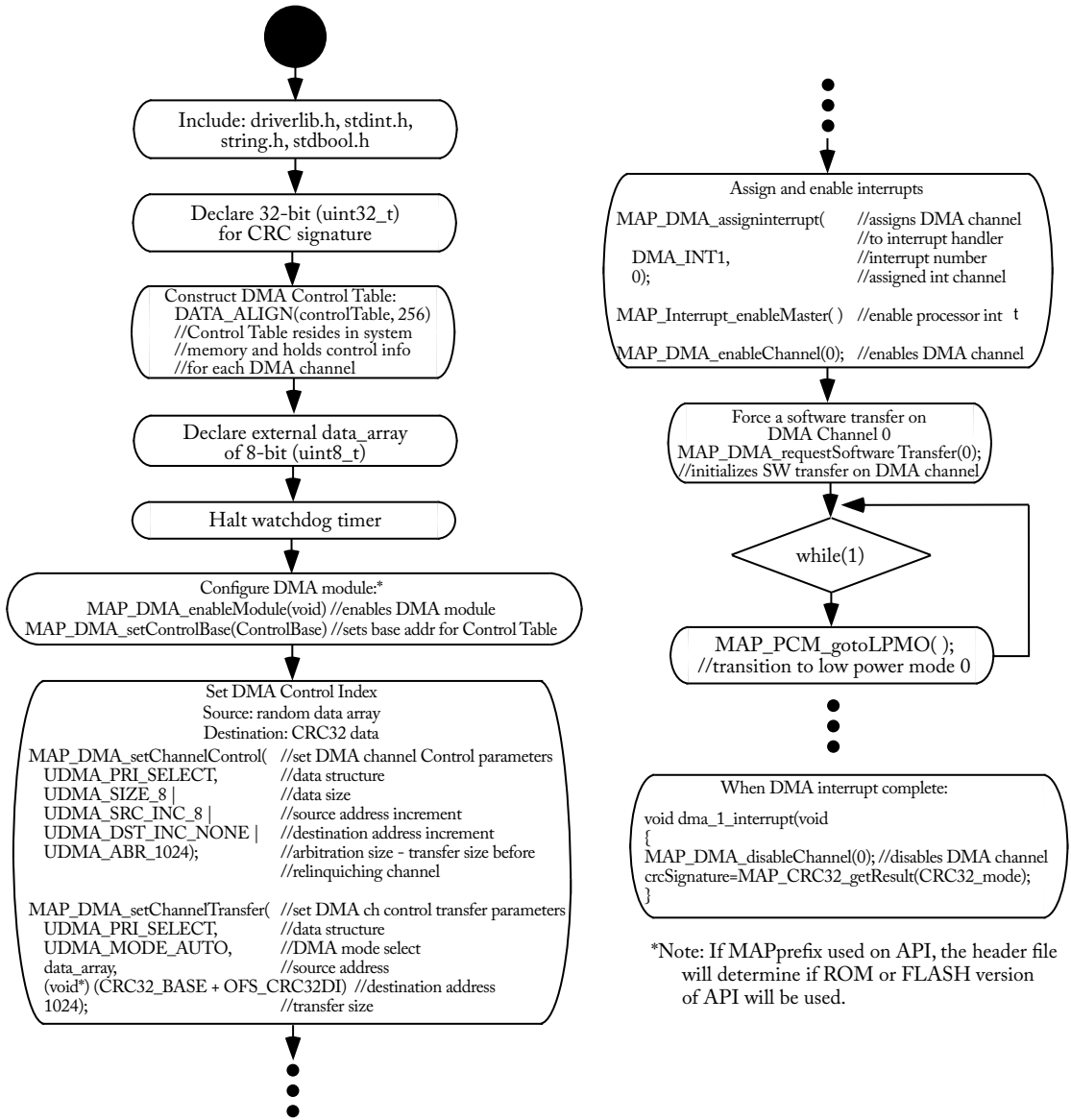


Figure 4.6: CRC Checksum via DMA [DriverLib, 2015].

```

//*****
//MSP432 DMA - CRC32 calculation using DMA
//
//Description: This code example shows how to use the DMA module on
//MSP432 to feed data into the CRC32 module for a CRC32 signature
//calculation.
This use case is particularly useful when the user wants
//to calculate the CRC32 signature of a large data array (such as a
//firmware image) but still wants to maximize power consumption.
After
//the DMA transfer is setup, a software initiation occurs and the MSP432
//device is put to sleep.
Once the transfer completes, the DMA interrupt
//occurs and the CRC32 result is placed into a local variable for the
//user to examine.
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>

#include <string.h>
#include <stdbool.h>

//Statics
static volatile uint32_t crcSignature;

//DMA Control Table
#ifdef ewarm
#pragma data_alignment=256
#else
#pragma DATA_ALIGN(controlTable, 256)
#endif
uint8_t controlTable[256];

```

```

//Extern
extern uint8_t data_array[];

int main(void)
{
//Halting Watchdog Timer
MAP_WDT_A_holdTimer();

//Configuring DMA module
MAP_DMA_enableModule();
MAP_DMA_setControlBase(controlTable);

//Setting Control Indexes.
In this case we will set the source of the
//DMA transfer to our random data array and the destination to the
//CRC32 data in register address
MAP_DMA_setChannelControl(UDMA_PRI_SELECT,
    UDMA_SIZE_8 | UDMA_SRC_INC_8 | UDMA_DST_INC_NONE | UDMA_ARB_1024);
MAP_DMA_setChannelTransfer(UDMA_PRI_SELECT, UDMA_MODE_AUTO, data_array,
    (void*) (CRC32_BASE + OFS_CRC32DI), 1024);

//Assigning/Enabling Interrupts
MAP_DMA_assignInterrupt(DMA_INT1, 0);
MAP_Interrupt_enableInterrupt(INT_DMA_INT1);
MAP_Interrupt_enableMaster();

//Enabling DMA Channel 0
MAP_DMA_enableChannel(0);

//Forcing a software transfer on DMA Channel 0
MAP_DMA_requestSoftwareTransfer(0);

while(1)
{
    MAP_PCM_gotoLPM0();
}
}

//*****

```

```
//Completion interrupt for DMA
//*****

void dma_1_interrupt(void)
{
MAP_DMA_disableChannel(0);
crcSignature = MAP_CRC32_getResult(CRC32_MODE);
}

//*****
```

4.7 EXTERNAL MEMORY: BULK STORAGE WITH AN MMC/SD CARD

A MultiMediaCard/SanDisk (MMC/SD) card provides a handy method of providing a low-power, non-volatile, and a small form factor (32 mm × 24 mm × 1.4 mm) bulk memory storage for a microcontroller. For example, the SD card is useful for data logging applications in remote locations. For example, if our goal was to measure wind resources at remote locations as potential windfarm sites, a MSP432-based data logging system equipped with an SD card could be used. Data could be logged over a long period of time and retrieved for later analysis on a PC.

The MMC/SD card is a smart peripheral device. It contains an onboard controller to manage MMC/SD operations. In the following laboratory exercise, we provide details of equipping the MSP432 with an MMC/SD card [[SanDisk, 2000](#)].

4.8 LABORATORY EXERCISE: MMC/SD CARD

Introduction. In this laboratory exercise, we equip the MSP-EXP432P401R LaunchPad with a 2 GB SD card, capture data from three analog sensors, and store the data to the SD card.

This laboratory consists of three parts:

- formatting the SD card with the File Allocation Table (FAT) 32 structure;
- constructing the interface between the SD card and the MSP-EXP432P401R LaunchPad; and
- modifying the Arduino based software for use with the MSP432 processor.

We complete each step in turn.

An SD card may be purchased in the camera or computer section at a local retailer. For this example, a 2 GB SD card was used. The SD card is inserted into the SD card slot in your laptop or PC. Open the file manager and right click on the SD card. Then click on “format” and select FAT32.

The MSP-EXP432P401R LaunchPad communicates with the SD card via the Serial Peripheral Interface (SPI) system. An easy method to construct an interface between the MSP432 and the SD card is via an SD card reader and breakout board. For this example, the card reader available from 43oh.com was used. A 6-pin header was soldered to the breakout board J5 connector. An interface diagram for the MSP432 and the 43oh.com breakout board is shown in Figure 4.7.

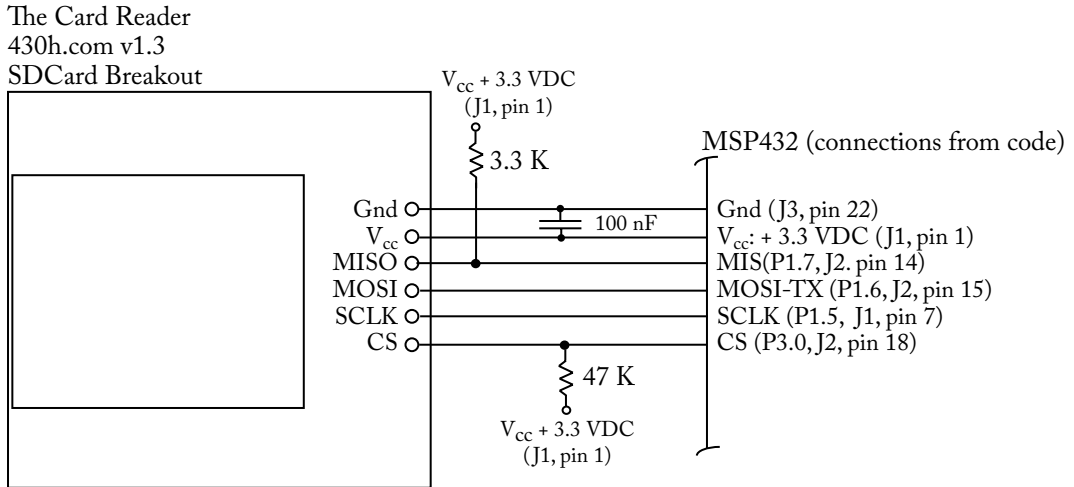


Figure 4.7: MSP432 to SD card breakout board interface.

Three analog sensors were connected to the MSP-EXP432P401R LaunchPad pins 12, 13, and 33.

The data logging software may be downloaded from the Arduino website www.arduino.cc. This public domain code was originally developed for the Arduino processor by Tom Igoe. It requires two modifications for use with the MSP432. The software modification was originally developed by M. Valencia (www.element14.com).

- File “Sd2PinMap.h” must be updated with the MSP-EXP432P401R LaunchPad pin assignments (MOSI-pin 15, MISO-pin 14, CLK-pin, and CS-pin 18). This may be accomplished using Code Composer Studio.
- As shown in the code below, the chip select assignment needs to be set to pin 18 (two places).

```
//*****
//SD card datalogger
//This example was originally developed by Tom Igoe for the Arduino
//processor on 24 November 2010.
```

```

//
//It was modified for use with the MSP432 LaunchPad by Martin Valencia
//in February 2016 Element 14 post "Interfacing SD Card with MSP432."
//
//Details on the interface between the SD card and the MSP432 were
//adapted from "Interfacing an SD-Card to the LaunchPad - A Walkthrough
//Tutorial by 43oh, December 21, 2013.
//
//All pin numbers and connections are provided for the
//MSP-EXP432P401R LaunchPad.
//
//This example shows how to log data from three analog sensors
//to an SD card using the SD library.
//
//The circuit:
// - Analog sensors are connected to MSP-EXP432P401R LaunchPad pins:
//   - 12, 13, 33
// - The SD card is interfaced to the MSP432 via the SPI. SPI
//   connections:
//   - MOSI - pin 15
//   - MISO - pin 14
//   - CLK - pin 7
//   - CS - pin 18 (also referred to as SS pin)
//
//Software modifications:
// - Update file: Sd2PinMap.h with pin assignments provided above.
// - Change CS (Chip Select) to pin 18 in code below.
//
//Created 24 Nov 2010
//Modified 9 Apr 2012
//by Tom Igoe
//This example code is in the public domain.
//www.arduino.cc
//*****

#include <SPI.h>
#include <SD.h>

#define left_sensor 12 //analog pin - left sensor

```


220 4. MSP432 MEMORY SYSTEM

```
#define center_sensor 33          //analog pin - center sensor
#define right_sensor 13          //analog pin - right sensor

int left_sensor_value;
int center_sensor_value;
int right_sensor_value;

const int chipSelect = 18;

void setup()
{
  delay(5000);                    //5s delay to open Serial Monitor Window
  Serial.begin(9600);             //Open serial communications
  Serial.print("Initializing SD card...");
  pinMode(18, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect))
  {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("card initialized.");
}

void loop()
{
  // make a string for assembling the data to log:
  String dataString = "";

  // read three sensors and append to the string:
  left_sensor_value  = analogRead(left_sensor);
  center_sensor_value = analogRead(center_sensor);
  right_sensor_value = analogRead(right_sensor);

  dataString += String(left_sensor_value);
  dataString += ",";
  dataString += String(center_sensor_value);
```

```

dataString += ",";
dataString += String(right_sensor_value);
dataString += ",";
delay(1000);

// open the file. note that only one file can be open at a time,
// so you have to close this one before opening another.
File dataFile = SD.open("datalog1.txt", FILE_WRITE);

// if the file is available, write to it:
if (dataFile)
{
  dataFile.println(dataString);
  dataFile.close();
  // print to the serial port too:
  Serial.println(dataString);
}
// if the file isn't open, pop up an error:
else
{
  Serial.println("error opening datalog1.txt");
}
}

//*****

```

Once the code is compiled and launched, data will be collected from the three analog sensors, displayed on the Serial Monitor, and logged to the SD card. Once the data is collected, the SD card may be removed from the card reader and inserted into the PC card reader. The data comma separated values (CSV) may then be read into Excel, analyzed, and plotted.

4.9 SUMMARY

In this chapter, with the help of the memory map, we presented the memory system of the MSP432 microcontroller. We demonstrated how contents of a memory location are accessed via read and write operations. We described the types of memories including RAM, ROM, and Flash. We then discussed the organization and operation of onboard flash memory and the direct memory access (DMA) system.

4.10 REFERENCES AND FURTHER READING

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015. 204, 211, 214

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. 201, 203, 209, 210

MultiMedia Card Product Manual, SanDisk Corporate Headquarters, 140 Caspian Court, Sunnyvale, CA, <http://www.sandisk.com>, 2000. 197, 217

Texas Instruments MSP432P401R LaunchPad Development Kit (MSP-EXP432P401R) (SLAU597A). Texas Instruments, 2015. 197

Texas Instruments MSP432P401x Mixed-Signal Microcontrollers (SLAS826A). Texas Instruments, 2015. 196, 197, 201, 208, 209

Texas Instruments MSP432x5xx/MSP432x6xx Family User's Guide (SLAU208G). Texas Instruments, 2010. 208

4.11 CHAPTER PROBLEMS

Fundamental

1. Convert CAFEh to binary.
2. Convert $(1101_1111_0000_1001)_2$ to decimal and hexadecimal numbers.
3. Convert $(11341)_{10}$ to binary and hexadecimal numbers.
4. A memory system is equipped with a 12 bit address bus. Using the linear addressing method, how many unique memory addresses are possible? What is the first and last memory addresses specified in binary? In hexadecimal?
5. A processor has a 16 bit data bus. What is the largest unsigned integer that may be carried by the bus? Signed integer?
6. Describe the different memory components available with the MSP432 microcontroller. Provide an application for each memory type.
7. Describe the purpose of the Direct Memory Access system.

Advanced

1. Research the interface between the MSP432 and a MultiMediaCard/Secure Digital (MMC/SD) card.

2. Sketch the memory map of the MSP432 microcontroller.

Challenging

1. Develop a MMC/SD card interface for the MSP432 microcontroller.
2. Write a function to clear a block of memory addresses in RAM memory. The start address and the number of memory locations to clear are passed into the function as arguments.

MSP432 Power Systems

Objectives: After reading this chapter, the reader should be able to:

- explain common practices to reduce power consumption in a microcontroller application;
- describe voltage regulation and methods of achieving regulation aboard the MSP432;
- illustrate the operation of the MSP432 Power Supply System;
- describe the operation of the MSP432 Power Control Module;
- define different operating modes of the MSP432 microcontroller;
- define battery capacity and its related parameters; and
- program the MSP432 to operate at different operating voltages and modes.

5.1 OVERVIEW

The MSP432 microcontroller is the lowest-power consuming microcontroller available on the market. It has been designed with a wide variety of ultra-low power (ULP) features. Although it can be used in a wide variety of applications, the controller is intended for battery operated applications where frequent battery replacement is undesirable or impractical. Application examples include pagers, battery-operated toys, portable measurement instruments, home automation products, medical instruments, metering applications, and portable smart card readers. The goal of this chapter is to present the MSP432's low power features so designers may take full advantage for microcontroller applications. The designer must also take into account battery capacity and understand how the required battery capacity relates to choosing an appropriate battery for a specific application.

In this chapter, we begin with a discussion on the balancing act microcontroller-based application designers must perform, during the design process, between the power requirements of a given project and available power sources. To that end, we overview the MSP432 low power features.

To optimize available power, we first present the active and low-power operating modes (LPM) of the MSP432 and how they help the controller to reduce its power consumption. We then investigate the MSP432 subsystems which contribute to low power operation, including the

Power Supply System (PSS) and the Power Control Module (PCM). The battery supply is considered next. We begin with a discussion of battery capacity and its key parameters. We also describe the important concept of voltage regulation and different methods of achieving a stable voltage source within a circuit. The chapter concludes with a laboratory exercise to investigate different MSP432 operating modes.

5.2 BACKGROUND THEORY

The MSP432 microcontroller is used in a number of applications where operation on a battery supply is required over a long period of time. To meet this operational requirement, the power demands of the MSP432 must be understood and balanced with the capacity of the battery source.

Generally speaking, the overall current demand of the MSP432, although the lowest in the industry, increases with [Day, 2009]:

- supply voltage level,
- central processing unit (CPU) clock speed,
- operating temperature,
- peripheral device selection,
- input/output use, and
- memory type and size.

The MSP432 microcontroller was designed as an ultra-low power processor. In general, to minimize power consumption in a given application and hence extend battery life, the following general procedures are followed [SLAA668, 2015]:

- reduce the microcontroller operating voltage;
- reduce the microcontroller operating frequency;
- minimize time spent in microcontroller active modes and maximize time spent in low power or sleep modes;
- minimize the transition time between power modes; and
- carefully examine the dependencies between modules.

Taking advantage of the low power features is, of course, more involved than simply minimizing the effects listed above. The general approach is to minimize instantaneous current draw while maximizing the time spent in low power modes. To do this, the designer must be well acquainted with the operating modes, the Power Supply System (PSS), and the Power Control Module (PCM). In addition, one must be well acquainted with the variety of MSP432 clock sources. MSP432 clock sources are described in Chapter 6.

5.3 OPERATING MODES AND SPEED OF OPERATION

As shown in Figure 5.1, the operating frequency of the MSP432 is related to the supply voltage and mode of operation. Shown around the outer arc is the desired processor operating frequency. Generally speaking, to achieve a higher operating frequency requires a higher operating voltage.

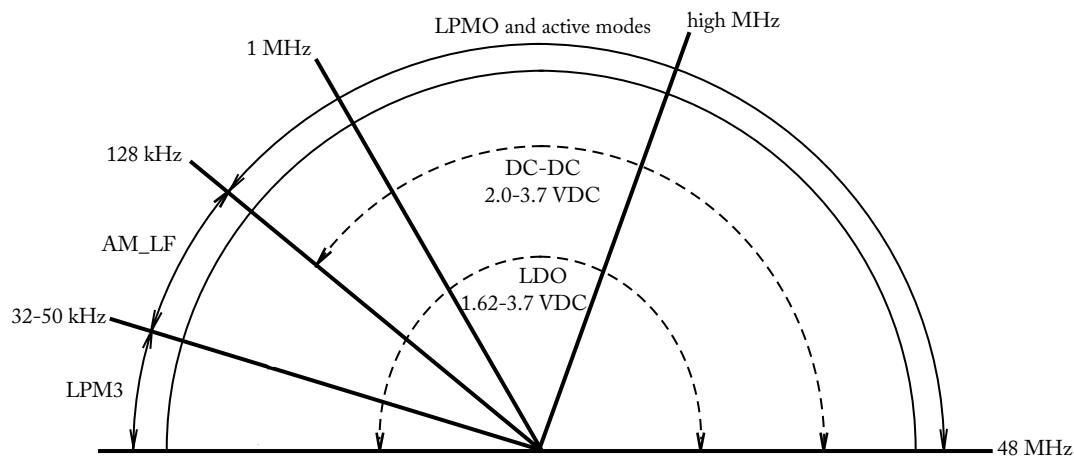


Figure 5.1: MSP432 operating modes vs. speed of operation [SLAA668, 2015].

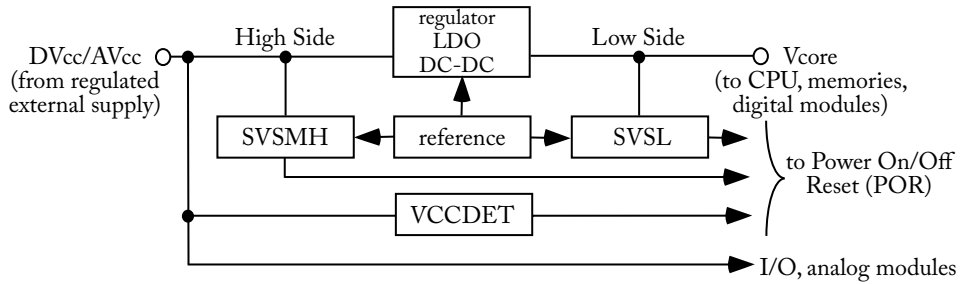
5.4 POWER SUPPLY SYSTEM

The MSP432 is equipped with a flexible and powerful Power Supply System (PSS). As shown in Figure 5.2, the PSS is an integral part of the Power Control Manager (PCM). We discuss the PSS first followed by the PCM [SLAU356A, 2015].

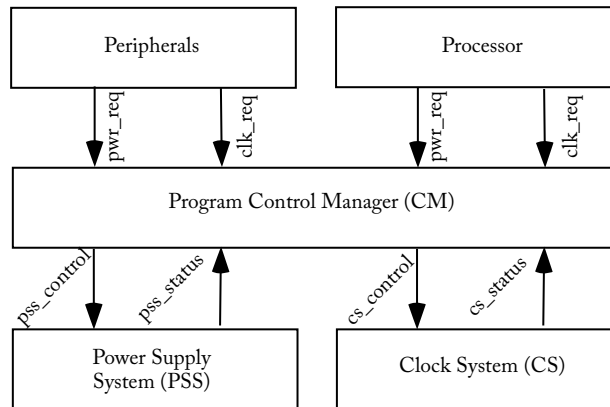
A voltage regulator maintains a stable output voltage under varying input voltage conditions. The main purpose of the PSS is to provide a regulated V_{CORE} voltage to the main processor should the input voltage provided to the processor vary. The V_{CORE} voltage ranges from 1.62–3.70 VDC. The exact value is determined by the operational needs of the specific application. As shown in Figure 5.2a, an external regulated voltage source is provided to the MSP432 via the DVcc pin. Typically, DVcc (digital source voltage) and AVcc (analog source voltage) are common. The DVcc voltage is regulated to the V_{CORE} voltage level using one of two regulators: the Low Drop Out (LDO) regulator or the DC-DC regulator [SLAU356A, 2015].

The LDO regulator is less efficient but more nimble in that fewer clock cycles are required between operating mode transitions. The DC-DC regulator operates with a better efficiency with a lower power consumption. However, it requires external components and has a longer wakeup time from various sleep modes [SLAU356A, 2015].

The PSS is equipped with a number of voltage monitors including VCCDET, SVSMH, and SVSL. All three monitors generate a Power On/Off Reset (POR) in the event of an out of tolerance voltage condition. VCCDET has a wide variation in threshold detection and is used primarily to detect a power on/off condition. The Supply Voltage Supervisor and Monitor (SVSMH) monitors the high side DV_{VCC} value. The Supply Voltage Supervisor monitors the low side V_{CORE} value [SLAU356A, 2015].



(a) Power Supply System (PSS).



(b) Power Control Manager (PCM).

Figure 5.2: MSP432 power systems. (a) Power Supply System, and (b) Power Control Manager [SLAU356A, 2015].

5.5 THE POWER CONTROL MODULE

The PSS is an integral portion of the Power Control Module (PCM). As shown in Figure 5.2, the PCM obtains system status information from the PSS and also the Clock System (CS). It also responds to requests from peripherals and the main processor core to optimize the operation of

the onboard power system. The PCM allows the MSP432 to be placed in a wide variety of active and low power modes. The selection of operational modes is determined by a variety of events including: PCM Control 0 register settings, interrupt and wakeup events, reset events, and debug events [SLAU356A, 2015].

5.6 OPERATING MODES

As shown in Figure 5.3, the MSP432 may be placed into a variety of operating modes. It is up to the designer to determine the best, most efficient operating mode(s) for the specific application. A detailed summary of each operating mode is provided in the next section. Out of hard reset the processor will transition to an active mode where CPU execution is possible. The Low Power Modes (LPM) may be entered from an active mode. When a specific low power mode is exited, the processor returns to an active mode. Processor execution is halted during an LPM [SLAU356A, 2015].

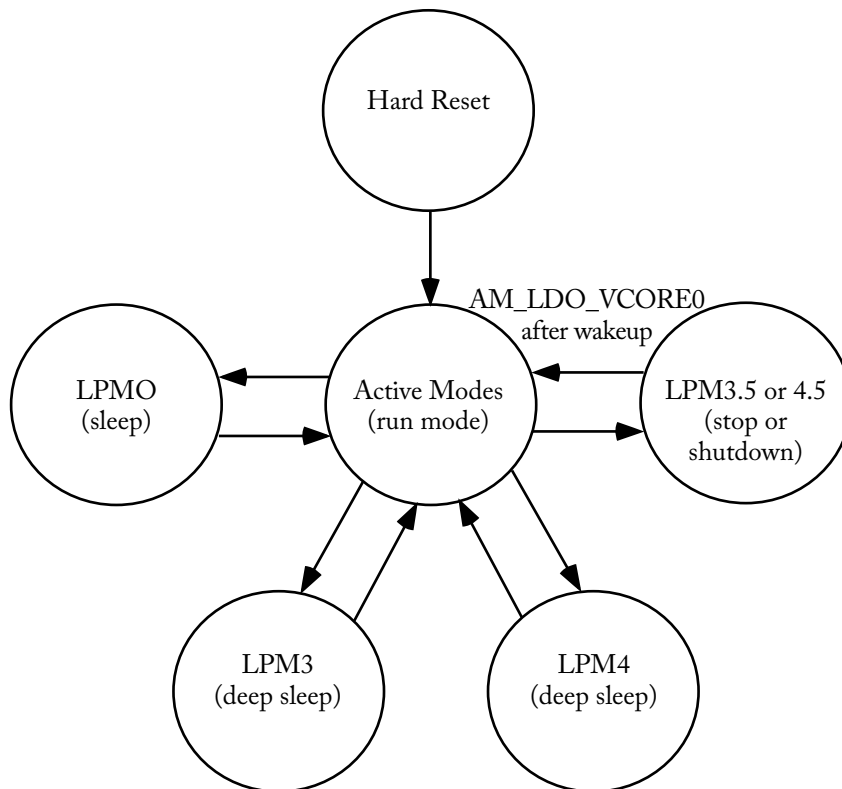


Figure 5.3: MSP432 power mode transitions [SLAU356A, 2015].

5.7 OPERATING MODE SUMMARY

Figures 5.4 and 5.5 provide a summary of the active and low power modes available for the MSP432. The designer chooses the appropriate mode(s) based on the specific application at hand. A determination is made based on the operating frequency required by the application and specific subsystems required. A specific LPM is determined based on the ongoing processor activity required by the application. An operating mode name indicates active or low power mode (AM or LPMx), followed by the regulator used (LDO or LF), and the core voltage selected (0 or 1) [SLAU356A, 2015].

5.8 OPERATING MODE TRANSITIONS

Switching operating modes in a microcontroller can be illustrated well with the analogy of driving a manual transmission vehicle. When first learning how to drive one, there are several basic rules to follow: (1) step on the clutch when changing gears, (2) when at a stop, start with first gear, and (3) as the vehicle gains speed, different gears are selected going from first, to second, to third, etc. When the vehicle slows, gears are selected going from a higher number to a lower number.

The operating mode transition on the MSP432 is very similar. One cannot simply change from one operational mode to another. Specific sequences of events must occur when changing operational modes. The sequences are followed to insure the V_{CORE} voltage is at a sufficient value to support the desired operational frequency of the processor. The specific sequence of mode changes are provided in Section 7.5 Power Mode Transitions of the *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015]. Examples are provided later in the chapter to illustrate transitions between processor operating modes.

5.9 PSS AND PCM REGISTERS

The PSS and PCM are supported by a variety registers including:

- PSSKEY Key Register
- PSSCTL0 Control 0 Register
- PSSIE Interrupt Enable Register
- PSSIFG Interrupt Flag Register
- PSSCLRIFG Clear Interrupt Flag Register
- PCMCTL0 Control 0 Register
- PCMCTL1 Control 1 Register
- PCMIE Interrupt Enable register

Power Mode	Operating State	Features/Application Constraints
Active Mode (Run Mode)	AM_LDO_VCORE0	LDO or DC-DC regulator-based active modes at core voltage level 0. CPU is active and full peripheral functionality is available. CPO and DMA maximum operating frequency is 24 MHz.
	AM_DCDC_VCORE0	Peripherals maximum input clock frequency is 12 MHz. All low- and high-frequency clock sources can be active. Flash memory and all enabled SRAM banks are active.
	AM_LDO_VCORE1	LDO or DC-DC regulator-based active modes at core voltage level 1. CPU is active and full peripheral functionality is available. CPU and DMA maximum operating frequency is 48 MHz.
	AM_DCDC_VCORE1	Peripherals maximum input clock frequency is 24 MHz. All low- and high-frequency clock sources can be active. Flash memory and all enabled SRAM banks are active.
	AM_LF_VCORE0	LDO-based low-frequency active modes at core voltage level 0 or 1. CPU is active and full peripheral functionality is available. CPU, DMA, and peripherals maximum operating frequency is 128 kHz. Only low-frequency clock sources (LFXT, REFO, and VLO) can be active.
	AM_LF_VCORE1	All high-frequency clock sources need to be disabled by application. Flash memory and all enabled SRAM banks are active. Flash erase/program operations and SRAM bank enable or retention enable configuration changes must not be performed by application. DC-DC regulator cannot be used.
LPM0 (Sleep)	LPM0_LDO_VCORE0	LDO or DC-DC regulator-based operating modes at core voltage level 0. CPU is inactive but full peripheral functionality is available.
	LPM0_DCDC_VCORE0	DMA maximum operating frequency is 24 MHz. Peripherals maximum input clock frequency is 12 MHz. All low- and high-frequency clock sources can be active. Flash memory and all enabled SRAM banks are active.
	LPM0_LDO_VCORE1	LDO or DC-DC regulator-based operating modes at core voltage level 1. CPU is inactive but full peripheral functionality is available.
	LPM0_DCDC_VCORE1	MA maximum operating frequency is 48 MHz. Peripherals maximum input clock frequency is 24 MHz. All low- and high-frequency clock sources can be active. Flash memory and all enabled SRAM banks are active.
	LPM0_LF_VCORE0	LDO-based low-frequency operating modes at core voltage level 0 or 1. CPU is inactive but full peripheral functionality is available. DMA and peripherals maximum operating frequency is 128 kHz. Only low-frequency clock sources (LXFT, REFO, and VLO) can be active.
	LPM0_LF_VCORE1	All high-frequency clock sources need to be disabled by application. Flash memory and all enabled SRAM banks are active. Flash erase/program operations and SRAM bank enable or retention enable configuration changes must not be performed by application. DC-DC regulator cannot be used.

Figure 5.4: MSP432 power mode summary [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

Power Mode	Operating State	Features/Application Constraints
LPM3 (Deep Sleep)	LDO_VCORE0	LDO-based operating modes at core voltage level 0 or 1. CPU is inactive and peripheral functionality is reduced. Only RTC and WDT modules can be functional with maximum input clock frequency of 32.768 kHz. All other peripherals and retention-enabled SRAM banks are kept under state retention power gating.
	LDO_VCORE1	Flash memory is disabled. SRAM banks not configured for retention are disabled. Only low-frequency clock sources (LFXT, REFO, and VLO) can be active. All high-frequency clock sources are disabled. Device I/O pin states are latched and retained. DC-DC regulator cannot be used.
LPM4 (Deep Sleep)	LDO_VCORE0	LDO-based operating modes at core voltage level 0 or 1. Achieved by entering LPM3 with RTC and WDT modules disabled. CPU is inactive with no peripheral functionality. All peripherals and retention-enabled SRAM banks are kept under state retention power gating.
	LDO_VCORE1	Flash memory is disabled. SRAM banks not configured for retention are disabled. All low- and high-frequency clock sources are disabled. Device I/O pin states are latched and retained. DC-DC regulator cannot be used.
LPM3.5 (Stop or Shut Down)	LDO_VCORE0	LDO-based operating mode at core voltage level 0. Only RTC and WDT modules can be functional with maximum input clock frequency of 32.768 kHz. CPU and all other peripherals are powered down. Only Bank-0 of SRAM is under data retention. All other SRAM banks and flash memory are powered down. Only low-frequency clock sources (LFXT, REFO, and VLO) can be active. All high-frequency clock sources are disabled. Device I/O pin states are latched and retained. DC-DC regulator cannot be used.
LPM4.5 (Stop or Shut Down)	VCORE_OFF	Core voltage is turned off. CPU, flash memory, all SRAM banks, and all peripherals are powered down. All low- and high-frequency clock sources are powered down. Device I/O pin states are latched and retained.

Figure 5.5: MSP432 power mode summary [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

- PCMIFG Interrupt Flag Register
- PCMCLRIFG Clear Interrupt Flag Register

Details of specific register and bit settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here. However, PCMCTL0 Control 0 Register contains control bits for the mode transitions. A summary of PCMCTL0 settings is shown in Figure 5.6.

5.10 BATTERY OPERATION

Many embedded applications involve remote, portable systems, operating from a battery supply. To properly design a battery source for an embedded system, the operating needs of the embedded system must be matched to the characteristics of the battery supply. To properly match the battery supply to the embedded system, the following operational details must be known.

- What are the voltage and current required by the embedded system?
- How long must the embedded system operate before battery replacement or recharge?
- Will the embedded system be powered from primary, non-rechargeable batteries or secondary, rechargeable batteries?
- Are there weight or size limitations to be considered in selecting a battery?

Once these questions have been answered, a battery may be chosen for a specific application. To choose an appropriate battery, the following items must be specified:

- battery voltage,
- battery capacity,
- battery size and weight, and
- primary or secondary battery.

Battery capacity is typically specified as a mA·H rating. The capacity is the product of the current drain and the battery operational life at that current level. It provides an approximate estimate of how long a battery will last under a given current drain. The capacity is reduced at higher discharge rates. It must also be kept in mind that a battery's voltage declines as the battery discharges.

Example 1: A typical 9 VDC non-rechargeable alkaline battery has a capacity of 550 mA·H. If the system has a maximum operating current of 50 mA, it will operate for approximately 11 hours before battery replacement is required. Also, the battery voltage will be somewhat less than 9 VDC at the 11 hour point.

Mode	Description	PCMCTL0		SLEEPDEEP	Entry Mechanism
		AMR[3:0]	LPMR[3:0]		
AM_LDO_VCORE0	Active mode Core voltage level 0 LDO operation	0h	x	0	Writing of AMR register
AM_LDO_VCORE1	Active mode Core voltage level 1 LDO operation	1h	x	0	
AM_DCDC_VCORE0	Active mode Core voltage level 0 DC-DC operation	4h	x	0	
AM_DCDC_VCORE1	Active mode Core voltage level 1 DC-DC operation	5h	x	0	
AM_LF_VCORE0	Low-Frequency Active mode Core voltage level 0 LDO operation	8h	x	0	
AM_LF_VCORE1	Low-Frequency Active mode Core voltage level 1 LDO operation	9h	x	0	
LPM0_LDO-VCORE0 LPM0_DCDC_VCORE0 LPM0_LF_VCORE0 LPM0_LDO-VCORE1 LPM0_DCDC_VCORE1 LPM0_LF_VCORE1	LPM0 modes Core voltage level same as respective active mode	Same as the corresponding active mode. Programming AMR is not a pre-requisite for the device to enter LPMO.	x	0	WFI, WFE, Sleep-on-exit
LPM3 LPM4	LPM3, LPM4 modes Core voltage level same as respective active mode	x	0h	1	WFI, WFE, Sleep-on-exit
LPM3.5	LPM3.5 mode Core voltage level 0	x	Ah	1	WFI, WFE, Sleep-on-exit
LPM4.5	LPM4.5 mode Core voltage turned off	x	Ch	1	WFI, WFE, Sleep-on-exit

Figure 5.6: PCMCTL0 Control 0 Register settings [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

A battery is typically used with a voltage regulator to maintain the voltage at a prescribed level. As mentioned earlier in the chapter, the MSP432 is typically powered from a 3.3 VDC source. Figure 5.7 shows a sample circuit to provide a +3.3 VDC source. The LM1117-3.3 is a 3.3 VDC, 800 mA low dropout regulator. The choice of a specific battery source depends on capacity requirements of the system.

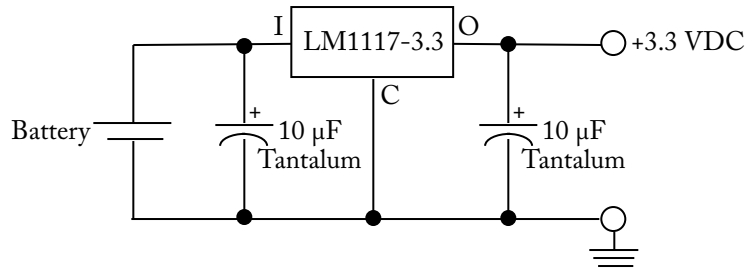


Figure 5.7: Battery supply circuits employing a 3.3 VDC regulators.

Primary and secondary batteries are manufactured using a wide variety of processes. In general, primary (non-rechargeable) batteries have a higher capacity than their secondary (rechargeable) counterparts. Also, batteries with higher capacity are more expensive than those using a lower capacity technology. A thorough review of the manufacturers' literature is recommended to select a battery for a specific application.

5.11 DRIVERLIB SUPPORT

The DriverLib library provides API support for the PSS and the PCM systems.

PSS support:

- void PSS_clearInterruptFlag(void)
- void PSS_disableHighSide(void)
- void PSS_disableHighSideMonitor(void)
- void PSS_disableHighSidePinToggle(void)
- void PSS_disableInterrupt(void)
- void PSS_disableLowSide(void)
- void PSS_enableHighSide(void)
- void PSS_enableHighSideMonitor(void)

- void PSS_enableHighSidePinToggle(bool activeLow)
- void PSS_enableInterrupt(void)
- void PSS_enableLowSide(void)
- uint_fast8_t PSS_getHighSidePerformanceMode(void)
- uint_fast8_t PSS_getHighSideVoltageTrigger(void)
- uint32_t PSS_getInterruptStatus(void)
- uint_fast8_t PSS_getLowSidePerformanceMode(void)
- void PSS_registerInterrupt(void(*intHandler)(void))
- void PSS_setHighSidePerformanceMode(uint_fast8_t powerMode)
- void PSS_setHighSideVoltageTrigger(uint_fast8_t triggerVoltage)
- void PSS_setLowSidePerformanceMode(uint_fast8_t ui8PowerMode)
- void PSS_unregisterInterrupt(void)

PCM support:

- void PCM_clearInterruptFlag(uint32_t flags)
- void PCM_disableInterrupt(uint32_t flags)
- void PCM_disableRudeMode(void)
- void PCM_enableInterrupt(uint32_t flags)
- void PCM_enableRudeMode(void)
- uint8_t PCM_getCoreVoltageLevel(void)
- uint32_t PCM_getEnabledInterruptStatus(void)
- uint32_t PCM_getInterruptStatus(void)
- uint8_t PCM_getPowerMode(void)
- uint8_t PCM_getPowerState(void)
- bool PCM_gotoLPM0(void)
- bool PCM_gotoLPM0InterruptSafe(void)

- `bool PCM_gotoLPM3(void)`
- `bool PCM_gotoLPM3InterruptSafe(void)`
- `void PCM_registerInterrupt(void(*intHandler)(void))`
- `bool PCM_setCoreVoltageLevel(uint_fast8_t voltageLevel)`
- `bool PCM_setCoreVoltageLevelWithTimeout(uint_fast8_t voltageLevel, uint32_t timeOut)`
- `bool PCM_setPowerMode(uint_fast8_t powerMode)`
- `bool PCM_setPowerModeWithTimeout(uint_fast8_t powerMode, uint32_t timeOut)`
- `bool PCM_setPowerState(uint_fast8_t powerState)`
- `bool PCM_setPowerStateWithTimeout(uint_fast8_t powerState, uint32_t timeout)`
- `bool PCM_shutdownDevice(uint32_t shutdownMode)`
- `void PCM_unregisterInterrupt(void)`

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015]. Examples are provided in the next section employing API support.

5.12 PROGRAMMING IN C

MSPWare provides a wide variety of examples related to power management and operating modes of the MSP432. In this section we highlight several representative examples from MSPWare [MSPWare].

Example 2: In this example the MSP432 is placed in LPM0 mode.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
```

```
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                                MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
```

```
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - Enter LPM0 with ACLK = REFO, SMCLK = 1.5MHz
//
//Description: Go to LPM0 mode
//
//          MSP432p401rpz
//          -----
//          /\| |
//          | | |
//          --|RST |
//          | | |
//          | | |
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****
```

```
#include "msp.h"
```

```
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;           // Stop WDT

    __sleep();
    __no_operation();
}
```

```
//*****
```

Example 3: In this example the MSP432 is changed from V_{CORE} level 0 to level 1.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
```

240 5. MSP432 POWER SYSTEMS

```
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                                MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
```

Also see:

```

// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - Change VCORE from LEVEL 0 to LEVEL 1
//
//
//Description: Change VCORE from LEVEL 0 to LEVEL 1
//
//          MSP432p401rpz
//          -----
//          /|\|          |
//          | |          |
//          --|RST       |
//          |           |
//          |           |
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include "stdint.h"

void error(void);

int main(void)
{
uint32_t currentPowerState;
volatile uint32_t i;

WDTCTL = WDTPW | WDTHOLD;    //Stop WDT
P1DIR |= BIT0;

```

```

//Get current power state
currentPowerState = PCMCTL0 & CPM_M;

//Transition to VCore Level 1 from current power state properly
switch (currentPowerState)
{
    case CPM_0: //AMO_LDO, need to switch to AM1_LDO
        while((PCMCTL1 & PMR_BUSY));
        PCMCTL0 = PCM_CTL_KEY_VAL | AMR_1;
        while((PCMCTL1 & PMR_BUSY));
        if(PCMIFG & AM_INVALID_TR_IFG)
            error(); //Error if transition was not successful
        break;

    case CPM_4: //AMO_DCDC, need to switch to AM1_DCDC
        while((PCMCTL1 & PMR_BUSY));
        PCMCTL0 = PCM_CTL_KEY_VAL | AMR_5;
        while((PCMCTL1 & PMR_BUSY));
        if (PCMIFG & AM_INVALID_TR_IFG)
            error(); //Error if transition was not successful
        break;

    default: //Device is in some other state, which is unexpected
        error();
}

P1OUT |= BIT0; //VCore switching sequence successful
__no_operation();
while(1);
}

//*****
// void error(void)
//*****

void error(void)
{
    volatile uint32_t i;

```

```

while(1)
{
  P1OUT ^= BIT0;
  for(i=0;i<20000;i++);      // Blink LED forever
}
}

```

```

//*****

```

Example 4: This example assumes the MSP432 is in an active mode employing the LDO regulator (mode AM0_LDO or AM1_LDO). A transition from DCDC mode to the low frequency (LF) mode requires an intermediate transition through LDO mode.

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,

```


244 5. MSP432 POWER SYSTEMS

```
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//           MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/-
//*****
//   MSP432P401 Demo - Enter Low-Frequency Active mode
//
//Note: This example assumes the device is currently in LDO mode
//      AM0_LDO or AM1_LDO (Active Mode using LDO, VCore=0/1)
//
//      A transition from DCDC mode to LF mode requires an intermediate
//      transition through LDO mode.
For more information on transition
//      patterns please refer to the PCM chapter in the device user's
//      guide.
//
//           AM1_DCDC <-----> AM1_LDO  <--@--> AM1_LF
//
//
//           |
//           |
```

```

//          |
//          v
//      AMO_DCDC <-----> AMO_LDO* <--@--> AMO_LF
//
//      *: power state condition after reset
//      @: transitions demonstrated in this code example
//
//      MSP432p401rpz
//      -----
//      /\| |          |
//      | |          |
//      --|RST       |
//      |          P1.0|----> LED
//      |          |
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include "stdint.h"

void error(void);

int main(void)
{
uint32_t currentPowerState;
volatile uint32_t i;

WDTCTL = WDTPW | WDTM0;           //Stop WDT
P1DIR |= BIT0;

//Switch MCLK over to REFO clock for low frequency operation first
CSKEY = CSKEY_VAL;                //Unlock CS module
CSCTL1 = (CSCTL1 & ~(SELM_M | DIVM_M)) | SELM_2;
CSKEY = 0;                         //Lock CS module

```

```
//Get current power state
currentPowerState = PCMCTL0 & CPM_M;

//Transition to Low-Frequency Mode from current LDO power
//state properly

switch (currentPowerState)
{
    case CPM_0: //AMO_LDO, need to switch to AM0_Low-Frequency Mode
        while((PCMCTL1 & PMR_BUSY));
        PCMCTL0 = PCM_CTL_KEY_VAL | AMR_8;
        while((PCMCTL1 & PMR_BUSY));
        if(PCMIFG & AM_INVALID_TR_IFG)
            error(); //Error if transition was not successful
        break;

    case CPM_1: //AM1_LDO, need to switch to AM1_Low-Frequency Mode
        while((PCMCTL1 & PMR_BUSY));
        PCMCTL0 = PCM_CTL_KEY_VAL | AMR_9;
        while((PCMCTL1 & PMR_BUSY));
        if(PCMIFG & AM_INVALID_TR_IFG)
            error(); //Error if transition was not successful
        break;

    case CPM_8: //Device is already in AM0_Low-Frequency Mode
        break;

    case CPM_9: //Device is already in AM1_Low-Frequency Mode
        break;

    default: //Device is in some other state, which is unexpected
        error();
}

P1OUT |= BIT0; //Transition LDO to Low-Frequency Mode successful
__no_operation();
while(1);
```

```

}

//*****
// void error(void)
//*****

void error(void)
{
volatile uint32_t i;

while(1)
{
    P1OUT ^= BIT0;
    for(i=0;i<20000;i++);    //Blink LED forever
}
}

//*****

```

Example 5: In this example the MSP432 PCM module is configured to use the DC-DC regulator instead of the default LDO regulator. The DC-DC regulator requires an external 4.7 uH inductor connected between VSW and V_{CORE} pins. The V_{CORE} pin requires a 100 nF and a 4.7 uF capacitor.

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//

```

248 5. MSP432 POWER SYSTEMS

```
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - Use DC-DC Regulator
//
//
//Description: Configure PCM module to use the DC-DC regulator instead of
```

```

//the default LDO. Note that DC-DC usage requires a 4.7uH inductor
//connected between VSW and VCORE pins.
VCORE pin still requires its
//regular 100nF and 4.7uF capacitors.
//
//Note: the code in this example assumes the device is currently in LDO
//mode AM_LDO_VCORE0 or AM_LDO_VCORE1 (Active Mode using LDO, VCore=0/1
//respectively)
//
//Transition from DCDC mode to Low-Frequency Mode requires intermediate
//transition through LDO mode.
For more information refer to the PCM
//chapter in the device user's guide.
//
//  AM_DCDC_VCORE1  <----->  AM_LDO_VCORE1  <----->  AM_LF_VCORE1
//                                     ^
//                                     |
//                                     |
//                                     |
//                                     v
//  AM_DCDC_VCORE0  <----->  AM_LDO_VCORE0* <----->  AM_LF_VCORE0
//
//  *: power state condition after reset
//  @: transitions demonstrated in this code example
//
//
//          MSP432P401R
//          -----
//          /|\|          |
//          | |          VCORE |-----
//          --|RST       |   |
//          |           |   4.7uH
//          |           |   |
//          |           VSW  |-----
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V5.5
//*****

```

```
#include "msp.h"
#include "stdint.h"

void error(void);

int main(void)
{
    uint32_t currentPowerState;

    WDTCTL = WDTPW | WDTHOLD;                //Stop WDT
    P1DIR |= BIT0;

    //Get current power state
    currentPowerState = PCMCTL0 & CPM_M;

    //Transition to DCDC from current LDO power state properly
    switch(currentPowerState)
    {
        case CPM_0: //AM_LDO_VCORE0, need to switch to AM_DCDC_VCORE0
            while((PCMCTL1 & PMR_BUSY));
            PCMCTL0 = PCM_CTL_KEY_VAL | AMR_4;
            while((PCMCTL1 & PMR_BUSY));
            if(PCMIFG & AM_INVALID_TR_IFG)
                error(); //Error if transition was not successful
            break;

        case CPM_1: //AM_LDO_VCORE1, need to switch to AM_DCDC_VCORE1
            while((PCMCTL1 & PMR_BUSY));
            PCMCTL0 = PCM_CTL_KEY_VAL | AMR_5;
            while((PCMCTL1 & PMR_BUSY));
            if(PCMIFG & AM_INVALID_TR_IFG)
                error(); //Error if transition was not successful
            break;

        case CPM_4: //Device is already in AM_LF_VCORE0
            break;

        case CPM_5: //Device is already in AM_LF_VCORE1
```

```

        break;

    default:    //Device is in some other state, which is unexpected
        error();
    }

P1OUT |= BIT0; //LDO --> DCDC switching sequence successful
__no_operation();
while(1);
}

//*****
// void error(void)
//*****

void error(void)
{
volatile uint32_t i;

while(1)
{
    P1OUT ^= BIT0;
    for(i=0;i<20000;i++);    //Blink LED forever
}
}

//*****

```

Example 6: In this example the MSP432 is configured to operate at 48 MHz using the DC-DC regulator instead of the default LDO regulator.

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions

```



```
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
```

```

//
// --/COPYRIGHT/--
//*****
//
//MSP432P401 Demo - device operation at 48MHz with DC-DC Regulator
//
//
//Description: Configure device to operate at 48MHz and use the DC-DC
//regulator instead of the default LDO. The following steps must be taken
//in sequence shown below to ensure proper Flash operation at 48MHz and
//with DC-DC regulator:
//  1. VCORE LEVEL = 1
//  2. Switch from LDO to DC-DC.
//  3. Configure flash Wait-state = 2 (Flash max frequency = 16MHz)
//  4. Configure DCO to 48MHz
//  5. Switch MCLK to use DCO as source
//
//Note: DC-DC usage requires a 4.7uH inductor connected between VSW and
//VCORE pins.
VCORE pin still requires its regular 100nF and 4.7uF
//capacitors.
//
//Note: the code in this example assumes the device is currently in
//AM0_LDO (Active Mode using LDO, VCore=0/1 respectively).
//
//Transition from DCDC mode to Low-Frequency Mode requires intermediate
//transition through LDO mode.
For more information refer to the PCM
//chapter in the device user's guide.
//
//  AM_DCDC_VCORE1 <-----> AM_LDO_VCORE1 <-----> AM_LF_VCORE1
//
//                                     ^
//                                     |
//                                     |
//                                     |
//                                     v
//  AM_DCDC_VCORE0 <-----> AM_LDO_VCORE0* <-----> AM_LF_VCORE0
//
//  *: power state condition after reset

```

254 5. MSP432 POWER SYSTEMS

```
// @: transitions demonstrated in this code example
//
//           MSP432P401R
//           -----
//           /|\|           |
//           | |           VCORE |-----
//           --|RST         |     |
//           |               |     4.7uH
//           |               |     |
//           |               VSW |-----
//           |               |
//           |               P1.0 |-----> LED
//           |               |
//           |               P4.3 |-----> MCLK
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include "stdint.h"

#define FLCTL_BANK0_RDCTL_WAIT__2    (2 << 12)
#define FLCTL_BANK1_RDCTL_WAIT__2    (2 << 12)

void error(void);

int main(void)
{
    uint32_t currentPowerState;

    WDTCTL = WDTPW | WDTHOLD;           //Stop WDT
    P1DIR |= BIT0;

    //NOTE: This example assumes the default power state is AM_LDO_VCORE0.
```

```

//Step 1: Transition to VCORE Level 1: AM_LDO_VCORE0 --> AM_LDO_VCORE1
//Get current power state, if it's not AM_LDO_VCORE0, error out
currentPowerState = PCMCTL0 & CPM_M;
if(currentPowerState != CPM_0)
    error();
while((PCMCTL1 & PMR_BUSY));
PCMCTL0 = PCM_CTL_KEY_VAL | AMR_1;
while((PCMCTL1 & PMR_BUSY));
if(PCMIFG & AM_INVALID_TR_IFG)
    error();          //Error if transition was not successful
if((PCMCTL0 & CPM_M) != CPM_1)
    error();          //Error if device is not in AM1_LDO mode

//Step 2: Transition from AM1_LDO to AM1_DCDC
while((PCMCTL1 & PMR_BUSY));
PCMCTL0 = PCM_CTL_KEY_VAL | AMR_5;
while((PCMCTL1 & PMR_BUSY));
if(PCMIFG & AM_INVALID_TR_IFG)
    error();          //Error if transition was not successful
if((PCMCTL0 & CPM_M) != CPM_5)
    error();          //Error if device is not in AM_DCDC_VCORE1 mode

//Step 3: Configure Flash wait-state to 2 for both banks 0 & 1
FLCTL_BANK0_RDCTL = FLCTL_BANK0_RDCTL & ~FLCTL_BANK0_RDCTL_WAIT_M |
                    FLCTL_BANK0_RDCTL_WAIT_2;
FLCTL_BANK1_RDCTL = FLCTL_BANK0_RDCTL & ~FLCTL_BANK1_RDCTL_WAIT_M |
                    FLCTL_BANK1_RDCTL_WAIT_2;

//Step 4&5: Configure DCO to 48MHz, ensure MCLK uses DCO as source
CSKEY = CSKEY_VAL;          //Unlock CS module for register access
CSCTL0 = 0;                 //Reset tuning parameters
CSCTL0 = DCORSEL_5;        //Set DCO to 48MHz

//Select MCLK = DCO, no divider
CSCTL1 = CSCTL1 & ~(SELM_M | DIVM_M) | SELM_3;
CSKEY = 0;                 //Lock CS module from unintended accesses

P1OUT |= BIT0;             //All operations successful

```

```

//Output MCLK to port pin to demonstrate 48MHz operation
P4DIR |= BIT3;
P4SEL0 |=BIT3;           //Output MCLK
P4SEL1 &= ~(BIT3);

__no_operation();
while(1);
}

//*****
// void error(void)
//*****

void error(void)
{
volatile uint32_t i;

while(1)
{
P1OUT ^= BIT0;
for(i=0;i<20000;i++);    //Blink LED forever
}
}

//*****

```

5.13 LABORATORY EXERCISE: OPERATING MODES

In this laboratory exercise the MSP432 is placed in its various operating modes. The program starts the MSP432 in PCM_AM0_LDO mode. Each time the pushbutton connected to P1.1 is depressed, the MSP432 cycles to the next operating mode. The LED connected to P1.0 blinks to indicate the state transition.

The EnergyTrace+ tool is used to measure the energy consumption of the state. EnergyTrace is a code analysis tool to measure and display an application's energy profile (www.ti.com).

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX

```

```
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
```

Also see:

```
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
/*****
//MSP432 Power Lab - Cycle through available power states & measure power
//
//Description: The goal of the lab is to explore and use the power API
//to exercise various power states available on the MSP432P401. The
//program will start up in default mode PCM_AMO_LDO. Push button P1.1
//can be used to cycle to the next power mode from the below.
P1.0 LED
//blinks to indicate the power state transition.
//
//In each power state, use EnergyTrace+ Tool to measure the energy
//consumption of that power state.
Document results across all 13
//different states.
//
// - \b PCM_AM_LDO_VCORE0,      [Active Mode, LDO, VCORE0]
// - \b PCM_AM_LDO_VCORE1,      [Active Mode, LDO, VCORE1]
// - \b PCM_AM_DCDC_VCORE0,     [Active Mode, DCDC, VCORE0]
// - \b PCM_AM_DCDC_VCORE1,     [Active Mode, DCDC, VCORE1]
// - \b PCM_AM_LPR_VCORE0,      [Active Mode, Low Frequency, VCORE0]
// - \b PCM_AM_LPR_VCORE1,      [Active Mode, Low Frequency, VCORE1]
// - \b PCM_LPMO_LDO_VCORE0,     [LPM0, LDO, VCORE0]
// - \b PCM_LPMO_LDO_VCORE1,     [LPM0, LDO, VCORE1]
// - \b PCM_LPMO_DCDC_VCORE0,    [LPM0, DCDC, VCORE0]
// - \b PCM_LPMO_DCDC_VCORE1,    [LPM0, DCDC, VCORE1]
// - \b PCM_LPMO_LPR_VCORE0,     [LPM0, Low Frequency, VCORE0]
// - \b PCM_LPMO_LPR_VCORE1,     [LPM0, Low Frequency, VCORE1]
// - \b PCM_LPM3,                [LPM3]
// - \b PCM_LPM35_VCORE0,        [LPM3.5 VCORE 0]
//
//Once Deep Sleep mode is entered, the next transition will start again
//with PCM_AMO_LDO.
```

```

//
//Power mode transition is accomplished using DriverLib API:
// PCM_setPowerState()
//Other related APIs: PCM_setPowerMode(), PCM_gotoSleep(), PCM_gotoLPM3()
//
//
//          MSP432P401
//          -----
//          /|\|          |
//          | |          |
//          --|RST      P1.1 |<--Toggle Switch
//          |          |
//          |          P1.0 |----> LED (red)
//          |          |
//          |          |
//          |          |
//          |          |
//
//Dung Dang
//Texas Instruments Inc.
//April 2014
//Built with Code Composer Studio V6.0
//*****

#include "driverlib.h"

//Application Data
volatile uint32_t curPowerState, ledState=0;
volatile bool stateChange;
volatile uint32_t ledBlinkCount, ledBlinkMax=0;

#define NUMBER_OF_POWER_STATES    13

void InitializeDevice(void);

int main(void)
{
//Halting the Watchdog
WDT_A_holdTimer();
InitializeDevice();

```



```
curPowerState=0;

while (1)
{
//If we have a state change request...
if(stateChange)
{
Interrupt_disableMaster();
stateChange = false;
Interrupt_enableMaster();

//Step 1: Find the correct Power API to change power state
//Step 2: Fill in different switch cases to change device to different
//      power states
//Step 3: Notice special clock handling for the Low-Power Run modes
//      where MCLK is restricted to <=128kHz
//Hint: Comment out the #error line after adding your solution code

switch(curPowerState)
{
case 0: #error "Invoke PCM API to change to Active Mode, VCORE = 0
          using LDO"
        break;

case 1: #error "Invoke PCM API to change to Active Mode, VCORE = 1
          using LDO"
        break;

case 2: #error "Invoke PCM API to change to Active Mode, VCORE = 0
          using DC-DC"
        break;

case 3: #error "Invoke PCM API to change to Active Mode, VCORE = 1
          using DC-DC"
        break;

case 4: //Switch all clocks to low-frequency operation prior to LF
        //operations
```

```
CS_initClockSignal(CS_MCLK, CS_REFCLK_SELECT,
                  CS_CLOCK_DIVIDER_1);
CS_initClockSignal(CS_SMCLK, CS_REFCLK_SELECT,
                  CS_CLOCK_DIVIDER_1);
CS_initClockSignal(CS_ACLK, CS_REFCLK_SELECT,
                  CS_CLOCK_DIVIDER_1);
#error "Invoke PCM API to change to Active Mode, VCORE = 0
using Low-Frequency Mode"
break;

case 5: #error "Invoke PCM API to change to Active Mode, VCORE = 1
using Low-Frequency Mode"
break;

case 6: //Switch back to using LDO regulator first before
//increasing the clocks
#error "Invoke PCM API to change to Active Mode, VCORE = 0
using LDO"

//Switch clocks back to high-frequency operation
CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT,
                  CS_CLOCK_DIVIDER_1);
CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT,
                  CS_CLOCK_DIVIDER_1);
#error "Invoke PCM API to change to LPM0 Mode, VCORE = 0
using LDO"
break;

case 7: #error "Invoke PCM API to change to LPM0 Mode, VCORE = 1
using LDO"
break;

case 8: #error "Invoke PCM API to change to LPM0 Mode, VCORE = 0
using DC-DC"
break;

case 9: #error "Invoke PCM API to change to LPM0 Mode, VCORE = 1
using DC-DC"
break;
```

```

    case 10: //Switch all clocks to low-frequency operation prior to LF
             //operations
             CS_initClockSignal(CS_MCLK, CS_REFOCLK_SELECT,
                                CS_CLOCK_DIVIDER_1);
             CS_initClockSignal(CS_SMCLK, CS_REFOCLK_SELECT,
                                CS_CLOCK_DIVIDER_1);
             CS_initClockSignal(CS_ACLK, CS_REFOCLK_SELECT,
                                CS_CLOCK_DIVIDER_1);
             #error "Invoke PCM API to change to LPM0 Mode, VCORE = 0
                    using Low-Frequency Mode"
             break;

    case 11: #error "Invoke PCM API to change to LPM0 Mode, VCORE = 1
                    using Low-Frequency Mode"
             break;

    case 12: //Switch back to using LDO regulator first before
             //increasing the clocks
             #error "Invoke PCM API to change to Active Mode, VCORE = 0
                    using LDO"

             //Switch clocks back to high-frequency operation
             CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT,
                                CS_CLOCK_DIVIDER_1);
             CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT,
                                CS_CLOCK_DIVIDER_1);
             #error "Invoke PCM API to change to LPM3 Mode"
             break;

    default: break;
}
}
}
}

//*****
//Port 1 interrupt handler. This handler is called whenever the switch
//attached to P1.1 is pressed.

```

```

A status flag is set to signal for the
//main application to change power states
//*****

void Port1IsrHandler(void)
{
uint32_t status = GPIO_getEnabledInterruptStatus(GPIO_PORT_P1);
GPIO_clearInterruptFlag(GPIO_PORT_P1, status);

if(status & GPIO_PIN1)
{
Interrupt_disableInterrupt(INT_PORT1);
if(curPowerState == 12) //Power State is PCM_DSL[Deep Sleep Mode]
{
curPowerState = 0;
}
else
{
curPowerState++;
}

ledBlinkMax = (curPowerState) * 2;
ledBlinkCount = 0;
if(++ledState==8)
ledState = 1;
P2OUT &= ~(0x07);
Interrupt_enableInterrupt(INT_TAO_0);
Timer_A_startCounter(TIMER_A0_MODULE, TIMER_A_UP_MODE);

}
}

//*****
//Flashes LED
//*****

void Timer_AIsrHandler(void)
{
Timer_A_clearCaptureCompareInterrupt(TIMER_A0_MODULE,

```

```
TIMER_A_CAPTURECOMPARE_REGISTER_0);

if(ledBlinkMax>0)
    P2OUT ^= ledState;
ledBlinkCount++;

if((ledBlinkCount == ledBlinkMax) || (ledBlinkMax==0))
{
    stateChange = true;
    Timer_A_stopTimer(TIMER_A0_MODULE);
    Interrupt_disableInterrupt(INT_TAO_0);
    //Re-enabling port pin interrupt
    GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1);
    Interrupt_enableInterrupt(INT_PORT1);
}
}

//*****
//Terminate GPIO
//*****

void TerminateGPIO(void)
{
    P1DIR = 0x00;
    P2DIR = 0x00;
    P3DIR = 0x00;
    P4DIR = 0x00;
    P5DIR = 0x00;
    P6DIR = 0x00;
    P7DIR = 0x00;
    P8DIR = 0x00;
    P9DIR = 0x00;
    P10DIR = 0x00;
    P1REN = 0xff;
    P2REN = 0xff;
    P3REN = 0xff;
    P4REN = 0xff;
    P5REN = 0xff;
    P6REN = 0xff;
    P7REN = 0xff;
```

```

P8REN = 0xff;
P9REN = 0xff;
P10REN = 0xff;
P1OUT = 0x00;
P2OUT = 0x00;
P3OUT = 0x00;
P4OUT = 0x00;
P5OUT = 0x00;
P6OUT = 0x00;
P7OUT = 0x00;
P8OUT = 0x00;
P9OUT = 0x00;
P10OUT = 0x00;

PSS_setHighSidePerformanceMode(PSS_NORMAL_PERFORMANCE_MODE);
//PSS_setLowSidePerformanceMode(PSS_NORMAL_PERFORMANCE_MODE);
PCM_enableRudeMode();
}

//*****
//Initialize Device
//*****

void InitializeDevice(void)
{
//TimerA UpMode Configuration Parameter
Timer_A_UpModeConfig upConfig =
{
    TIMER_A_CLOCKSOURCE_ACLK,           //SMCLK Clock Source
    TIMER_A_CLOCKSOURCE_DIVIDER_1,     //SCLK/1 = 3MHz
    16000,                               //50000 tick period
    TIMER_A_TAIE_INTERRUPT_DISABLE,    //Disable Timer interrupt
    TIMER_A_CCIE_CCRO_INTERRUPT_ENABLE, //Enable CCRO interrupt
    TIMER_A_SKIP_CLEAR                 //Clear value
};

Interrupt_disableMaster();
TerminateGPIO();

```

```

//Initializing Variables
curPowerState = 0;
stateChange = false;
ledBlinkCount = 0;

//Setting the Reference Oscillator to 128KHz.
For Low Power Run modes,
//the MCLK frequency is required to be scaled back to 128KHz.
CS_setReferenceOscillatorFrequency(CS_REF0_128KHZ);

//Setting up TimerA to be sourced from ACLK and for ACLK to be sourced
//from the 128Khz REF0. Since the frequency of MCLK will be changed when
//we go into LPR mode, we want to make our LED blink look consistent.
CS_initClockSignal(CS_ACLK, CS_REF0CLK_SELECT, CS_CLOCK_DIVIDER_1);
Timer_A_configureUpMode(TIMER_A0_MODULE, &upConfig);
Timer_A_enableCaptureCompareInterrupt(TIMER_A0_MODULE,
                                     TIMER_A_CAPTURECOMPARE_REGISTER_0);

//Configuring P2 as output and P1.1 (switch) as input
GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 );

//Configuring P1.1 as an input and enabling interrupts
GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1);
GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1);
GPIO_interruptEdgeSelect(GPIO_PORT_P1, GPIO_PIN1,
                        GPIO_HIGH_TO_LOW_TRANSITION);
Interrupt_enableInterrupt(INT_PORT1);
Interrupt_disableSleepOnIsrExit();
SysCtl_enableSRAMBankRetention(SYSCTL_SRAM_BANK7);
Interrupt_enableMaster();
}

//*****

```

5.14 SUMMARY

In this chapter, we began with a discussion on the balancing act microcontroller-based designers must perform between the power requirements of a given application and available power sources. To assist the designers, we overviewed the low power features of the MSP432 microcontroller.

To optimize the available power use, we presented the active and low-power operating modes (LPM) of the MSP432 and how they help to reduce power consumption. We then investigated the MSP432 subsystems which contribute to low-power operation, including the Power Supply System (PSS) and the Power Control Module (PCM). By evaluating the required battery capacity and the operating modes of the MSP432 controllers, one can choose appropriate batteries to satisfy system requirements. We then looked at the other side of the coin, the battery supply. We began with a discussion of battery capacity and its key parameters. We also described the important concept of voltage regulation and different methods of achieving a stable source of voltage within a circuit. The chapter concluded with a laboratory exercise to investigate different MSP432 operating modes.

5.15 REFERENCES AND FURTHER READING

Day, M. Using power solutions to extend battery life in MSP430 applications. *Analog Applic. J.*, Fourth Quarter 2009, Texas Instruments Incorporated, 10–12. 226

Designing an Ultra-Low-Power (ULP) Application With MSP432 Microcontrollers (SLAA668). Texas Instruments, 2015. 226, 227

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015. 237

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. 227, 228, 229, 230, 231, 232, 233, 234

MSPWare for MSP Microcontrollers., <http://www.ti.com/tool/mspware>, Texas Instruments, 2016. 237

5.16 CHAPTER PROBLEMS

Fundamental

1. Draw a block diagram and describe the operation of the Power Management System (PMM).
2. Draw a block diagram and describe the operation of the Power Control Module (PCM).
3. What is the difference between supply voltage supervision and monitoring?
4. What is the difference between a primary voltage source and a secondary voltage source?

Advanced

1. Design a 3.3 VDC source for an MSP432 using a AA battery pack. Fully specify all components. What is the capacity of the battery pack?

2. Write a one-page point paper summarizing best practices for low-power operation.
3. Write a one-page point paper on the concept of battery capacity.
4. A common battery used for microcontroller applications is the CR2032 “coin” battery. This is a lithium battery at 3 VDC with a capacity of 250 mAh. Is the battery suitable to power the MSP432? Explain.
5. Construct a table summarizing low power modes (LPM). The table should include bit settings to enter the specific LPM and features available in the mode.
6. Construct an experiment to monitor battery voltage degradation during use. Plot results for several different battery technologies.
7. Construct a table summarizing available primary and secondary battery sources. At a minimum, the table should include common battery sizes (AA, AAA, C, D, and 9 VDC) and their capacity.

Challenging

1. Write a function in C to place the MSP432 in a specified low power mode (LPM). The desired LPM is passed into the function as an unsigned integer variable.
2. Compile a list of best practices to operate the MSP432 microcontroller in the most efficient manner.
3. Write a power management program to be installed on a remote weather station using the MSP432. Make reasonable assumptions for sensors on the weather station and its power usage.

Time-Related Systems

Objectives: After reading this chapter, the reader should be able to:

- explain clock signal generators available on the MSP432;
- describe the process used to select each clock signal generator as the MSP432 clock source;
- illustrate the use of the Watchdog Timer;
- describe the features of the Timer32 system;
- configure the Timer32 system for different applications;
- describe the features of the Timer_A system;
- configure the Timer_A system for different applications;
- explain the operation of the Real-Time Clock timer (RTC_C);
- describe the Real-Time Clock timer system;
- program Timer_A capture and compare subsystems to interface with external devices;
- describe time related features supported within Energia; and
- write programs using the timer subsystems (Watchdog, real-time clock, and capture/compare subsystems) and their interrupt modules.

6.1 OVERVIEW

One of the main reasons for the proliferation of microcontrollers as the “brain” of embedded systems is their ability to interface with multiple external devices such as sensors, actuators, and display units. In order to communicate successfully with such devices, microcontrollers must have capabilities to meet signal time constraints required by the external devices. As the number and complexities of external devices intended to be interfaced with the microcontroller grow, the associated timer systems must also possess sophisticated timer features to accommodate the needs of the devices.

For example, consider an actuator controlled by a servo motor requiring a pulse width modulated signal with precise timing requirements as its input. A communication device connected to

a microcontroller may need a unique pulse with a specified width to initiate its process. In other applications, microcontrollers need to capture the time of an external event or distinguish periodic input signals by computing their frequencies and periods. To meet these time constraints, embedded systems must have a fairly sophisticated timer system to generate a variety of clock signals, capture external event capabilities, and produce desired output time related signals.

The goal of this chapter is to introduce the timing related features of the MSP432. We begin with background information and related terminology followed by the clock systems of MSP432. We then discuss the Watchdog Timer (WDT), Timer32, Timer_A, and the Real-Time Clock (RTC_3) system.

6.2 BACKGROUND

Throughout the history of microcontrollers, one of the main challenges was the need to operate with minimal power. The motivation comes from microcontrollers working as the “brain” of embedded systems that are often operating remotely without a continuous external power source. Since the power used by a microcontroller is directly proportional to the speed of transistors switching logic states, computer designers implemented multiple methods to reduce the clock speed. One method was to design a controller such that the central processing unit operates at a high clock speed while other subsystems run at a lower clock speed.

Such architectures with multiple clock sources can also allow programmers/engineers to turn off subsystems while they are not in use, saving more power for the overall embedded system. The MSP432 designers adopted this philosophy of providing users with multiple clock sources such that, depending on applications, one can have the flexibility to configure his or her controller appropriately. In the next section, we review time-related terminology followed by descriptions of the MSP432 clock system and the timer systems that take advantages of the multiple clock sources.

6.3 TIME-RELATED SIGNAL PARAMETERS

In this section we review time related terminology used throughout the chapter.

6.3.1 FREQUENCY

Consider a signal $x(t)$ that repeats itself. We call this signal periodic with period T , if it satisfies the following equation:

$$x(t) = x(t + T).$$

To measure the frequency of a periodic signal, we count the number of times a particular event repeats within a one second period. The unit of frequency is Hertz or cycles per second. For example, a sinusoidal signal with a 60 Hz frequency means that a full cycle of the sinusoid signal repeats itself 60 times each second or once every 16.67 ms.

6.3.2 PERIOD

The reciprocal of frequency is a period. If an event occurs with a rate of 1 Hz, the period of that event is 1 s. To find the period, given a frequency, or vice versa, we simply need to remember their inverse relationship $f = \frac{1}{T}$ where f and T represent a frequency and the corresponding period, respectively. Both periods and frequencies of signals are often used to specify timing constraints of embedded systems.

For example, when your car is on a wintery road and slipping, the engineers who designed your car configured the anti-slippage unit to react within some millisecond period, say 20 ms. The constraint then forces the design team that monitors the slippage to program their monitoring system to check a slippage at a minimum rate of 50 Hz.

6.3.3 DUTY CYCLE

In many applications, periodic pulses are used as control signals. A good example is the use of a periodic pulse to control a servo motor. To control the direction and sometimes the speed of a motor, a periodic pulse signal with a changing duty cycle over time is used. The periodic pulse signal, shown in Figure 6.1 frame (a), is on for 50% of the signal period and off for the rest of the period. The pulse shown in frame (b) is on for only 25% of the same period as the signal in frame (a) and off for 75% of the period. The duty cycle is defined as the percentage of one period a signal is on. Therefore, we call the signal in frame (a) in Figure 6.1 as a periodic pulse signal with a 50% duty cycle and the corresponding signal in frame (b), a periodic pulse signal with a 25% duty cycle.

6.3.4 PULSE WIDTH MODULATION

The speed of a DC motor can be controlled by a pulse width modulated (PWM) signal. Suppose you have the circuit setup as shown in Figure 6.2. The figure shows how the batteries are connected to power the motor through a switch. It is obvious that when we close the switch the DC motor will rotate and continue to rotate with a set speed proportional to the DC voltage provided by the batteries.

Now suppose we open and close the switch rapidly. It will cause the motor to rotate and stop rotating per the switch position. As the time between the closing and opening of the switch decreases, the motor will not have time to make a complete stop and will continue to rotate with a speed proportional to the time the switch is closed. This is the underlying principle of controlling DC motor speed using the PWM signal. When the logic of the PWM signal is high, the motor is turned on, and when the logic of the waveform is low, the motor is turned off. By controlling the time the motor is on, we can control the speed of the DC motor.

The duty cycle is defined as the fractional time the logic is high with respect to one cycle time of the PWM signal. Thus, 0% duty cycle means the motor is completely turned off while 100% duty cycle means the motor is on all the time. Aside from motor speed control applications,

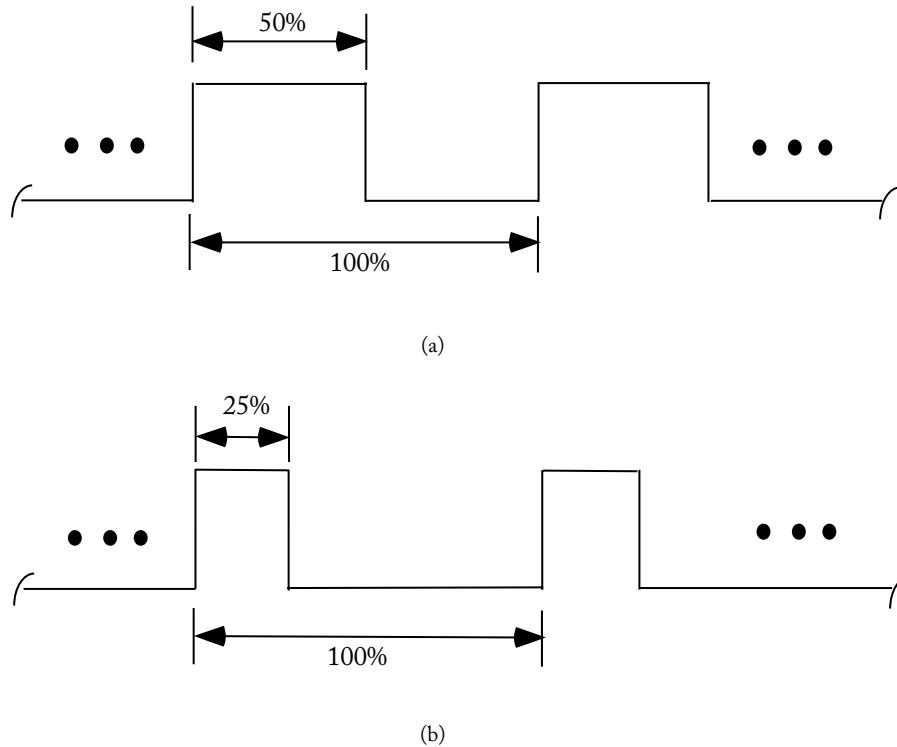


Figure 6.1: Two signals with the same period but different duty cycles. Frame (a) shows a periodic signal with a 50% duty cycle, and frame (b) displays a periodic signal with a 25% duty cycle.

PWM techniques are used in a wide variety of applications such as in communications, as control signals, in power delivery systems, and regulating voltage.

6.3.5 INPUT CAPTURE AND OUTPUT COMPARE

The heart of a timing system is the time base. The time base frequency of an oscillating signal is used to generate a baseline clock signal. For a timer system, the system clock is used to update the contents of a special register called a free running counter. The job of a free running counter is to count up (increment) each time it sees a rising edge (or a falling edge) of a clock signal. Thus, if a clock is running at the rate of 2 MHz, the free running counter will count up at every $0.5 \mu\text{s}$. All other timer-related units reference the contents of the free running counter to perform input and output time-related activities: measurement of time periods, capture of timing events, and generation of time-related signals.

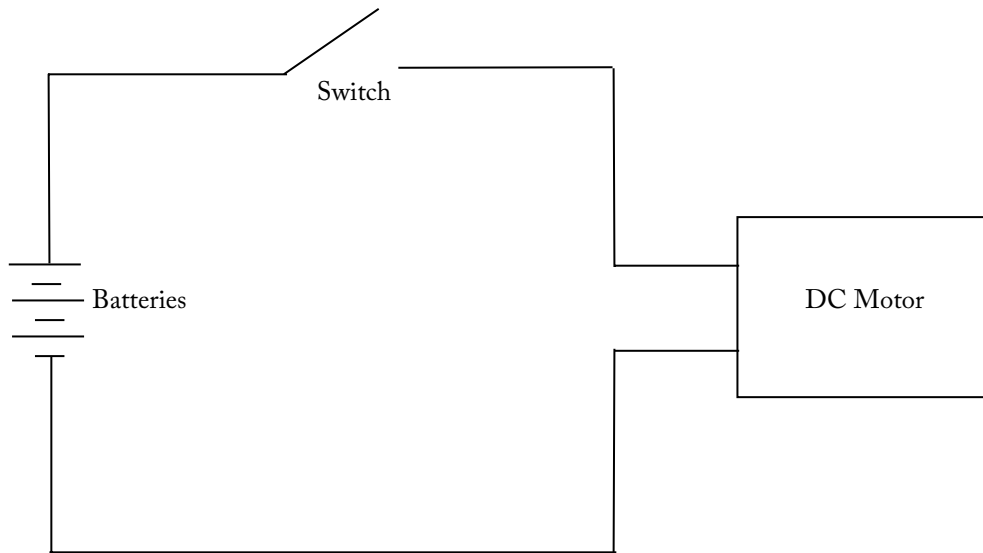


Figure 6.2: An example setup for controlling a DC motor.

For input time-related activities, all microcontrollers typically have timer hardware components that detect signal logic changes on one or more input pins. Such components rely on a free running counter to capture external event times. We can use this feature to measure the period of an incoming signal, the width of a pulse, and the time of a signal logic change.

You can also use the timer input system to measure the pulse width of an aperiodic signal. For example, suppose that the times for the rising edge and the falling edge of an incoming signal are 1.5 s and 1.6 s, respectively. We can use these values to easily compute the pulse width of 0.1 s.

For output timer functions, a microcontroller uses a comparator, a free running counter, logic switches, and special purpose registers to generate time related signals on one or more output pins. A comparator checks the value of the free running counter for a match with the contents of another special purpose register where a programmer stores a specified time in terms of the free running counter value. The checking process is executed at each clock cycle and when a match occurs, the corresponding hardware system induces a programmed logic change on a programmed output port pin. Using this feature, one can generate a simple logic change at a designated time incident: a pulse with a desired time width or a pulse width modulated signal to control servo or Direct Current (DC) motors.

From the examples above, you may have wondered how a microcontroller can compute absolute times from the relative free running counter values, say 1.5 s and 1.6 s. The simple answer is that we cannot do so directly. A programmer must use the relative system clock values and derive the absolute time values.

Suppose your microcontroller is clocked by a 2 MHz signal and the system clock uses a 16-bit free running counter. For such a system, each clock period represents $0.5 \mu\text{s}$, and it takes approximately 32.78 ms to count from $0-2^{16}$ (65,536). The timer input system then uses the clock values to compute frequencies, periods, and pulse widths. Again, suppose you want to measure a pulse width of an incoming aperiodic signal. If the rising edge and the falling edge occurred at count values \$0010 and \$0114,¹ can you find the pulse width when the free running counter is counting at 2 MHz? Let's first convert the two values into their corresponding decimal values, 16 and 276. The pulse width of the signal in the number of counter value is 260. Since we already know how long it takes for the system to count one, we can readily compute the pulse width as $260 \times 0.5 \mu\text{s} = 130 \mu\text{s}$.

Our calculations do not take into account time increments lasting longer than the rollover time of the counter. When a counter rolls over from its maximum value back to zero, a flag is set to notify the processor of this event. The rollover events may be counted to correctly determine the overall elapsed time of an event. Elapsed time may be calculated using the following:

$$\text{elapsed clock ticks} = (n \times 2^b) + (\text{stop count} - \text{start count})[\text{clock ticks}]$$

$$\text{elapsed time} = (\text{elapsed clock ticks}) \times (\text{FRC clock period})[\text{seconds}].$$

In this first equation, “n” is the number of timer overflow events that occur between the start and stop events, and “b” is the number of bits in the timer counter. The equation yields the elapsed time in clock ticks. To convert to seconds, the number of clock ticks are multiplied by the period of the clock source of the free running counter.

6.4 MSP432 CLOCK SYSTEM

The architects of the MSP432 designed the controller's clock system to provide flexibility and minimum power consumption based on intended applications. Note microcontroller power consumption is directly related to the oscillation speed of the clock. The higher the frequency of a clock signal, the more power is used. For example, typical microcontroller applications require a high frequency clock for the central processing unit but not for the input/output interface systems. Running input/output subsystems based on a slower clock saves power.

The MSP432 clock system is quite versatile. It may use an internal time base, external crystals for precise time resolution, or external resonators for the time base. The MSP432 clock system is shown in Figure 6.3 [SLAU356A, 2015].

The MSP432 clock system's time base may be provided for an external source via the LFXIN (low frequency) or the HFXIN (high frequency) external input pins. These input signals become the LFXTCLK or the HFXTCLK signals. Alternatively, the time base may be provided internally via the Digital Controlled Oscillator (DCO), REFOCLK, MODOSC, or SYSOSC. Provided below is additional information on each clock source [SLAU356A, 2015]:

¹The \$ symbol represents that the following value is in a hexadecimal form.

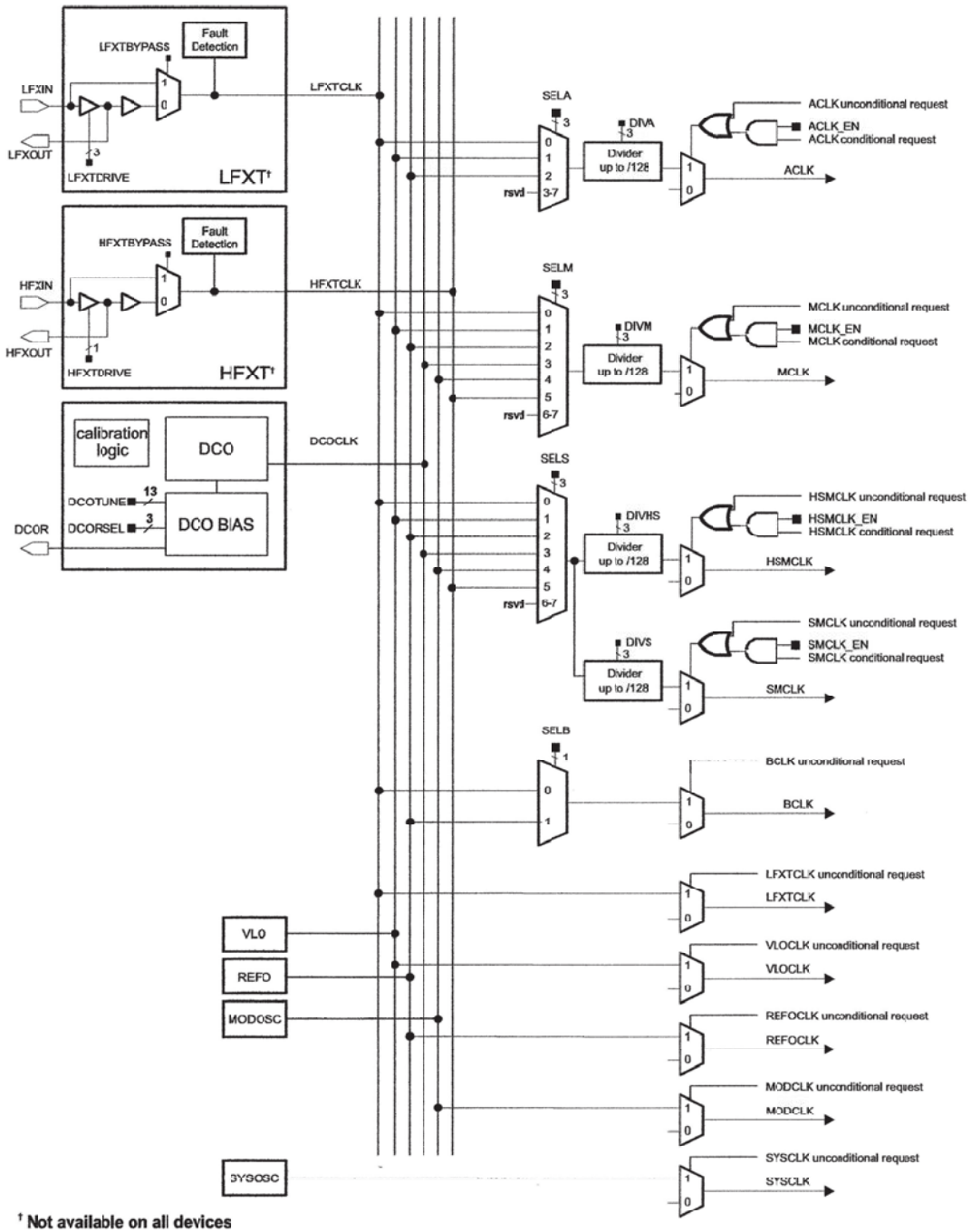


Figure 6.3: MSP432 clock system. Illustration used with permission of Texas Instruments www.ti.com.

- **LFXTCLK:** This low-frequency external time source is typically provided by a 32,768 Hz time source such as an external crystal or resonator. The frequency is 2^{15} and may be divided to provide a 1 Hz time base for use in keeping real time. For example, a 15-bit counter will rollover once per second when clocked from a 32,768 Hz source.
- **HFXTCLK:** This high-frequency time source is provided externally in a range from 1–48 MHz. The time base may be an external crystal or resonator. Typically, a crystal is more accurate than a ceramic resonator.
- **DCOCLK:** This signal is provided internally by the Digitally Controlled Oscillator. The DCOCLK operating frequency may be set for a variety of different frequencies using software configurable register settings.
- **VLOCLK:** This internal clock source features very low-power, very low-frequency operation. It is typically set for 10 kHz operation.
- **REFOCLK:** This internal clock source features low-power, low-frequency operation. It is typically set for a 32,768 Hz or 128 kHz frequency of operation.
- **MODCLK:** This internal low power oscillator operates at 24 MHz.
- **SYSCLK:** This internal clock source typically operates at 5 MHz.

As can be seen in Figure 6.3, the clock sources are routed to different portions of the MSP432 to become the ACLK, MCLK, HSMCLK, SMCLK, BCLK, LFXTCLK, VLOCLK, REFOCLK, MODCLK, and SYSCLK. Source selection for each of these clocks are controlled by various multiplexers input select bits (e.g., SELA, SELM, SELS, etc.). The sources may also be divided down to slower frequencies as shown in Figure 6.3 [SLAU356A, 2015].

6.4.1 CLOCK SOURCE REGISTERS

The clock source is configured using a series of registers including [SLAU356A, 2015]:

- **CSKEY:** Clock System Key Register
- **CSCTL0:** Clock System Control 0 Register
- **CSCTL1:** Clock System Control 1 Register
- **CSCTL2:** Clock System Control 2 Register
- **CSCTL3:** Clock System Control 3 Register
- **CSCLKEN:** Clock System Clock Enable Register
- **CSSTAT:** Clock System Status Register

- **CSIE:** Clock System Interrupt Enable Register
- **CSIFG:** Clock System Interrupt Flag Register
- **CSCLRIFG:** Clock System Clear Interrupt Flag Register
- **CSSETIFG:** Clock System Set Interrupt Flag Register
- **CSDCOERCAL:** Clock System DCO External Resistor Calibration Register

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

6.4.2 DRIVERLIB APIS

The Clock System is well supported by several APIs including [DriverLib, 2015]:

- void CS_clearInterruptFlag(uint32_t flags)
- void CS_disableClockRequest(uint32_t selectClock)
- void CS_disableDCOExternalResistor(void)
- void CS_disableFaultCounter(uint_fast8_t counterSelect)
- void CS_disableInterrupt(uint32_t flags)
- void CS_enableClockRequest(uint32_t selectClock)
- void CS_enableDCOExternalResistor(void)
- void CS_enableFaultCounter(uint_fast8_t counterSelect)
- void CS_enableInterrupt(uint32_t flags)
- uint32_t CS_getACLK(void)
- uint32_t CS_getBCLK(void)
- uint32_t CS_getDCOFrequency(void)
- uint32_t CS_getEnabledInterruptStatus(void)
- uint32_t CS_getHSMCLK(void)
- uint32_t CS_getInterruptStatus(void)
- uint32_t CS_getMCLK(void)
- uint32_t CS_getSMCLK(void)

- void CS_initClockSignal(uint32_t selectedClockSignal, uint32_t clockSource, uint32_t clockSourceDivider)
- void CS_registerInterrupt(void(*intHandler)(void))
- void CS_resetFaultCounter(uint_fast8_t counterSelect)
- void CS_setDCOCenteredFrequency(uint32_t dcoFreq)
- void CS_setDCOExternalResistorCalibration(uint_fast8_t uiCalData)
- void CS_setDCOFrequency(uint32_t dcoFrequency)
- void CS_setExternalClockSourceFrequency(uint32_t lfxxt_XT_CLK_frequency, uint32_t hfxt_XT_CLK_frequency)
- void CS_setReferenceOscillatorFrequency(uint8_t referenceFrequency)
- void CS_startFaultCounter(uint_fast8_t counterSelect, uint_fast8_t countValue)
- void CS_startHFXT(bool bypassMode)
- void CS_startHFXTWithTimeout(bool bypassMode, uint32_t timeout)
- void CS_startLFXT(uint32_t xtDrive)
- void CS_startLFXTWithTimeout(uint32_t xtDrive, uint32_t timeout)
- void CS_tuneDCOFrequency(int16_t tuneParameter)
- void CS_unregisterInterrupt(void)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 1: In this example an external high frequency crystal serves as the clock source and is also routed to serve as the source for the MCLK. Also, a general purpose input/output pin is configured as an output to support an LED [DriverLib, 2015].

```

/*****
//This example is used with permission from:
//MSP432 Peripheral Driver Library User's Guide [DriverLib]
/*****

//Configuring pins for peripheral/crystal usage and LED for output
MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_PJ,
          GPIO_PIN3 | GPIO_PIN4, GPIO_PRIMARY_MODULE_FUNCTION);

```

```

MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

//Setting the external clock frequency.
This API is optional,
//but will come in handy if the user ever wants to use the
//getMCLK, getACLK, etc. functions
CS_setExternalClockSourceFrequency(32000, 48000000);

//Starting HFXT in non-bypass mode without a timeout.
//Before we start we have to change VCORE to 1 to support
//the 48MHz frequency
MAP_PCM_setCoreVoltageLevel(PCM_VCORE1);
MAP_FlashCtl_setWaitState(FLASH_BANK0, 2);
MAP_FlashCtl_setWaitState(FLASH_BANK1, 2);
CS_startHFXT(false);

//Initializing MCLK to HFXT (effectively 48MHz)
MAP_CS_initClockSignal(CS_MCLK, CS_HFXTCLK_SELECT, CS_CLOCK_DIVIDER_1);
//*****

```

6.4.3 TIMER APPLICATIONS IN C

The MSP432 timer system may also be programmed using C with register configuration techniques. In this section we provide several representative examples.

Example 2: In this example the MSP432 is configured for an MCLK frequency of 12 MHz. The SMCLK is also set for this frequency while the ACLK is sourced from the REFOCLK operating at approximately 32 kHz.

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.

```

280 6. TIME-RELATED SYSTEMS

```
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
```



```

//SMCLK = MCLK = DCO
CSCTL1 = SELA_2 | SELS_3 | SELM_3;
CSKEY = 0;
//Lock CS module from
//unintended accesses

while(1) //continuous loop
{
    P1OUT ^= BIT0; //XOR P1.0
    for (i = 20000; i > 0; i--); //Delay
}
}
//*****

```

Example 3: In this example the MSP432 is configured for an MCLK operating frequency of 48 MHz. This will include actions to configure the VCORE level to one, the flash wait state to two, the DCO frequency to 48 MHz, and the DCO as the MCLK time source. After these configuration actions, the MCLK is output to port pin P4.3 for observation.

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

```

```

// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
// COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
// BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
// OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
// ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
// TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
// USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
// DAMAGE.
// --COPYRIGHT--
// *****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
// MSP432 code examples are self-contained low-level programs that
// typically demonstrate a single peripheral function or device feature
// in a highly concise manner. For this the code may rely on the device's
// power-on default register values and settings such as the clock
// configuration and care must be taken when combining code from several
// examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
// for an API functional library and:
// https://dev.ti.com/pinmux/
// for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
// *****
// MSP432P401 Demo - Device configuration for operation at
// MCLK = DCO = 48MHz
//
// Description: Proper device configuration to enable operation at
// MCLK=48MHz including:
// 1. Configure VCORE level to 1
// 2. Configure flash wait-state to 2
// 3. Configure DCO frequency to 48MHz
// 4. Ensure MCLK is sourced by DCO

```



```

//
//After configuration is complete, MCLK is output to port pin P4.3.
//
//          MSP432P401RPZ
//          -----
//          /|\|          |
//          | |          |
//          --|RST       |
//          |          P4.3|----> MCLK
//          |          |
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include "stdint.h"

#define FLCTL_BANK0_RDCTL_WAIT_2    (2 << 12)
#define FLCTL_BANK1_RDCTL_WAIT_2    (2 << 12)

void error(void);

int main(void)
{
uint32_t currentPowerState;
WDTCTL = WDTPW | WDTHOLD;           //Stop the WDT

//NOTE: This example assumes the default power state is AM0_LDO.
//Refer to MSP4322001_pcm_0x code examples for more complete PCM
//operations to exercise various power state transitions between
//active modes.
//
//Step 1: Transition to VCORE Level 1: AM0_LDO --> AM1_LDO
//Get current power state, if it's not AM0_LDO, error out

currentPowerState = PCMCTL0 & CPM_M;

```

```

if(currentPowerState != CPM_0)
    error();

while((PCMCTL1 & PMR_BUSY));
PCMCTL0 = PCM_CTL_KEY_VAL | AMR_1;
while((PCMCTL1 & PMR_BUSY));
if(PCMIFG & AM_INVALID_TR_IFG)
    error();                                //Error if transition was not
                                            //successful

if((PCMCTL0 & CPM_M) != CPM_1)
    error();                                //Error if device is not in
                                            //AM1_LDO mode

//Step 2: Configure Flash wait-state to 2 for both banks 0 & 1

FLCTL_BANK0_RDCTL = FLCTL_BANK0_RDCTL & ~FLCTL_BANK0_RDCTL_WAIT_M |
                    FLCTL_BANK0_RDCTL_WAIT_2;
FLCTL_BANK1_RDCTL = FLCTL_BANK0_RDCTL & ~FLCTL_BANK1_RDCTL_WAIT_M |
                    FLCTL_BANK1_RDCTL_WAIT_2;

//Step 3: Configure DCO to 48MHz, ensure MCLK uses DCO as source

CSKEY = CSKEY_VAL;                        //Unlock CS module for register
                                            //access
CSCTL0 = 0;                                //Reset tuning parameters
CSCTL0 = DCORSEL_5;                       //Set DCO to 48MHz
                                            //Select MCLK = DCO,
                                            //no divider
CSCTL1 = CSCTL1 & ~(SELM_M | DIVM_M) | SELM_3;
CSKEY = 0;                                //Lock CS module from unintended
                                            //accesses

//Step 4: Output MCLK to port pin to demonstrate 48MHz operation
P4DIR |= BIT3;
P4SEL0 |= BIT3;                            //Output MCLK
P4SEL1 &= ~(BIT3);

//Go to sleep
__sleep();

```

```

__no_operation();                               //For debugger
}

//*****

void error(void)
{
volatile uint32_t i;
P1DIR |= BIT0;

while (1)
{
P1OUT ^= BIT0;
for(i=0;i<20000;i++);                          //Blink LED forever
}
}

//*****

```

6.5 ENERGIA-RELATED TIME FUNCTIONS

For the remainder of the chapter we investigate time-related peripherals onboard the MSP432 including the Watchdog Timer, Timer32, Timer_A, and the Real-Time Clock (RTC_C). Before doing so, we review time related functions available within Energia.

The Energia Development Environment has several built-in functions related to timing events, providing delays, or generating pulse width modulated (PWM) signals. The functions include (www.energia.nu) the following.

- **millis():** This function provides the number of milliseconds that has occurred since the processor began running the current program.
- **micros():** This function provides the number of microseconds that has occurred since the processor began running the current program.
- **delay():** Provides a program pause for the specified number of milliseconds.
- **delayMicroseconds():** Provides a program pause for the specified number of microseconds. Note: This function is accurate for values 16,383 μ s or less.
- **analogWrite():** The analogWrite function provides a 490 Hz pulse width modulated signal on the specified PWM capable pin. The duty cycle is provided as an argument to the function from 0–255. For example, to specify a 90% duty cycle, the value would be 230.


```
{
pinMode(buttonPin, INPUT_PULLUP);
pinMode(ledPin, OUTPUT);
}

void loop()
{
//read the state of the switch into a local variable:
int reading = digitalRead(buttonPin);

//check to see if you just pressed the button
//(i.e., the input went from LOW to HIGH), and you've waited
//long enough since the last press to ignore any noise:
//If the switch changed, due to noise or pressing:
if (reading != lastButtonState)
    {
    lastDebounceTime = millis();
    }

if ((millis() - lastDebounceTime) > debounceDelay)
    {
    //whatever the reading is at, it's been there for longer
    //than the debounce delay, so take it as the actual current state:
    buttonState = reading;
    }

//set the LED using the state of the button:
digitalWrite(ledPin, buttonState);

//save the reading.
Next time through the loop,
//it'll be the lastButtonState:
lastButtonState = reading;
}

//*****
```

6.6 WATCHDOG TIMER

The MSP432 is equipped with a Watchdog Timer designated WDT_A. As the name implies, the primary purpose of the Watchdog timer is to monitor and prevent software failure by forcing the user code to refresh a designated control register periodically throughout the execution of a program. The secondary purpose of the watchdog timer is to generate periodic time intervals.

By software failure, we mean the execution of unintended instructions by the MSP432, whether it is an unintended infinite loop or a wrong segment of program being executed due to hardware errors, programmer errors, or noise related malfunctions. We now present how we configure the watchdog system to prevent software failure or as a periodic interval generator.

6.6.1 WDT MODES OF OPERATION

The Watchdog timer prevents software failure by enforcing the following rule: a 32-bit register, called the Watchdog Count (WDTCNT) register, counts up at each clock cycle. If the register reaches its maximum count, the processor generates a reset. When developing an application, the designer should strategically place commands to reset the WDTCNT from reaching its maximum count and thereby preventing a reset. Under normal operation a reset will not occur. However, if the processor experiences a fault, the WDTCNT may not receive its required reset signal and the processor will reset in attempt to clear the fault. The WDT feature is available in the active and LPM0 power modes [SLAU356A, 2015].

The Watchdog timer can also be configured to generate a periodic interval. In this mode the WDT generates an interrupt rather than a processor reset. The interval timer mode is available in the active mode, LPM0, LPM3 with the BCLK or VLOCLK, or LPM 3.5 with the BCLK or VLOCLK [SLAU356A, 2015].

It is important to note that from reset, the MSP432 is configured with the Watchdog Timer active with an SMCLK source. Therefore, the WDT must be halted, reset, or reconfigured before the WDTCNT reaches its maximum count at approximately 10.92 ms. Recall that in examples provided thus far in the book, the WDT is typically halted in the first step of each program [SLAU356A, 2015].

6.6.2 WDT SYSTEM

A block diagram of the Watchdog Timer (WDT) is provided in Figure 6.4. The WDT contains two key registers: the WDT Timer Control Register (WDTCTL) and the 32-bit WDT Count (WDTCNT) register. The WDTCTL is a 16-bit register containing a password compare upper byte and control bits in the lower byte. When writing to this register, it must be written using half word (16-bit) based instructions. During the write operation the upper byte must contain the password (5A)_h. If the password is incorrect, the processor will reset. As shown in Figure 6.4, the time base for the WDT system may be the SMCLK, ACLK, VLOCLK, or the BCLK. The clock

source is chosen using the WDTSSSEL[1:0] select bits in the WDTCTL register [SLAU356A, 2015].

6.6.3 WATCHDOG DRIVERLIB APIS

The following APIs are available in DriverLib to support the Watchdog System:

- void WDT_A_clearTimer(void)
- void WDT_A_holdTimer(void)
- void WDT_A_initIntervalTimer(uint_fast8_t clockSelect, uint_fast8_t clockDivider)
- void WDT_A_initWatchdogTimer(uint_fast8_t clockSelect, uint_fast8_t clockDivider)
- void WDT_A_registerInterrupt(void(*intHandler)(void))
- void WDT_A_setPasswordViolationReset(uint_fast8_t resetType)
- void WDT_A_setTimeoutReset(uint_fast8_t resetType)
- void WDT_A_startTimer(void)
- void WDT_A_unregisterInterrupt(void)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 5: In this short code snapshot, the WDT is configured to generate a periodic interrupt [DriverLib, 2015].

```
//*****
//This example is used with permission from:
//MSP432 Peripheral Driver Library User's Guide [DriverLib]
//*****
//Configure the WDT in the interval mode to trigger every
//32K clock cycles - approximately half second intervals
//*****

MAP_WDT_A_initIntervalTimer(WDT_A_CLOCKSOURCE_SMCLK, WDT_A_CLOCKITERATIONS_32K);

//*****
```

Example 6: In the following example, the WDT is configured in the interval mode to blink an LED at a regular interval.

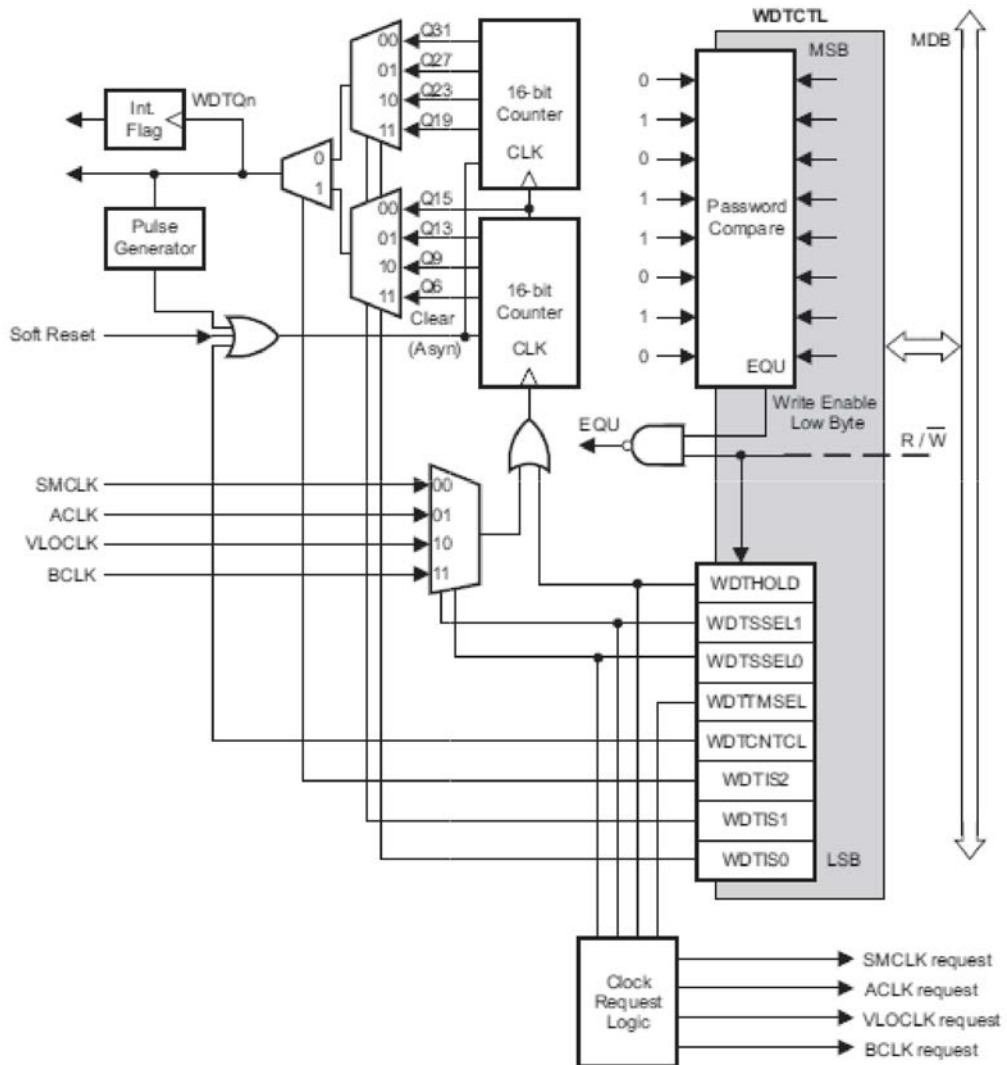


Figure 6.4: Watchdog Timer [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

292 6. TIME-RELATED SYSTEMS

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
//  notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
//  notice, this list of conditions and the following disclaimer in the
//  documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//                                MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
```

```

//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 WDT - Interval Mode
//
//Description: In this example, the WDT module is setup in interval mode.
//This turns the watchdog timer into a normal 16-bit up counter that can
//be operated in LPM3 mode.
Given that the WDT can be operated in LPM3
//mode (DSL), this is useful to wake up processor once put to LPM3. In
//this example, a simple LED is blinked at a constant interval using the
//WDT.
//
//
//          MSP432P401
//          -----
//          /|\|          |
//          | |          |
//          --|RST      P1.0 |---> P1.0 LED
//          |          |
//          |          |
//          |          |
//          |          |
//          |          |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

```

```
//Standard Includes
#include <stdint.h>
#include <stdbool.h>

#define WDT_A_TIMEOUT RESET_SRC_1

int main(void)
{
//Halting the Watchdog (while we set it up)
MAP_WDT_A_holdTimer();

//Setting MCLK to REFO at 128kHz for LF mode
//Setting SMCLK to 64kHz
MAP_CS_setReferenceOscillatorFrequency(CS_REFO_128KHZ);
MAP_CS_initClockSignal(CS_MCLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1);
MAP_CS_initClockSignal(CS_SMCLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_2);
MAP_PCM_setPowerState(PCM_AM_LF_VCORE0);

//Enabling SRAM Bank Retention
MAP_SysCtl_enableSRAMBankRetention(SYSCTL_SRAM_BANK1);

//Configuring GPIO1.0 as an output
MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

//Configuring WDT in interval mode to trigger every 32K clock iterations.
//This comes out to roughly every half a second
MAP_WDT_A_initIntervalTimer(WDT_A_CLOCKSOURCE_SMCLK,
                            WDT_A_CLOCKITERATIONS_32K);

//Enabling interrupts and starting the watchdog timer
MAP_Interrupt_enableInterrupt(INT_WDT_A);
MAP_Interrupt_enableMaster();
MAP_Interrupt_enableSleepOnIsrExit();
MAP_WDT_A_startTimer();

//LPM3ing when not in use
while(1)
```

```

    {
    MAP_PCM_gotoLPM3();
    }
}

//*****
//WDT ISR - This ISR toggles the LED on P1.0
//*****

void WDT_A_isr(void)
{
MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
}

//*****

```

Example 7: In this example, the WDT is configured as a watchdog timer to detect a system fault and generate a processor reset.

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

```

296 6. TIME-RELATED SYSTEMS

```
/"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 WDT - Servicing the Dog
//
//Description: In this example, the WDT module is used in a typical use
//case that illustrates how the watchdog can initiate a reset if the
//system becomes unresponsive.
The watchdog timer is setup to initiate
//a soft reset if it hasn't been serviced in 4 seconds.
A simple SysTick
//is also setup to make it so that the watchdog is serviced every second.
```

```

//When the GPIO button connected to P1.1 is pressed, the SysTick
//interrupt will be disabled causing the watchdog to timeout.
Upon reset,
//the program will detect that the watchdog timeout triggered a soft
//reset and blink the LED on P1.0 to signify the watchdog timeout.
//
//
//          MSP432P401
//          -----
//          /\| |
//          | | |
//          --|RST      P1.1 |<--- Switch
//          | | |
//          | | |
//          | | |
//          | | |
//          | | |
//          | | |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>
#include <stdbool.h>

#define WDT_A_TIMEOUT RESET_SRC_1

int main(void)
{
volatile uint32_t ii;

//Halting the Watchdog (while we set it up)
MAP_WDT_A_holdTimer();

//If the watchdog just reset, we want to toggle a GPIO to illustrate
//that the watchdog timed out.

```

```
Period of LED is 1s
if(MAP_ResetCtl_getSoftResetSource() & WDT_A_TIMEOUT)
{
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

    while(1)
    {
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
        for(ii=0;ii<4000;ii++)
        {

        }
    }
} //end if

//Setting MCLK to REFO at 128Khz for LF mode and SMCLK to REFO
MAP_CS_setReferenceOscillatorFrequency(CS_REFO_128KHZ);
MAP_CS_initClockSignal(CS_MCLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1);
MAP_CS_initClockSignal(CS_HSMCLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1);
MAP_CS_initClockSignal(CS_SMCLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1);
MAP_PCM_setPowerState(PCM_AM_LF_VCORE0);

//Configuring GPIO6.7 as an input for button press
MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1);

//Configuring WDT to timeout after 512k iterations of SMCLK, at 128k,
//this will roughly equal 4 seconds
MAP_SysCtl_setWDTTimeoutResetType(SYSCTL_SOFT_RESET);
MAP_WDT_A_initWatchdogTimer(WDT_A_CLOCKSOURCE_SMCLK,
                            WDT_A_CLOCKITERATIONS_512K);

//Setting our SysTick to wake up every 128000 clock iterations to service
//the dog.
MAP_SysTick_enableModule();
MAP_SysTick_setPeriod(128000);
MAP_SysTick_enableInterrupt();

//Enabling interrupts and starting the watchdog timer
```

```
MAP_GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1);
MAP_Interrupt_enableInterrupt(INT_PORT1);
MAP_Interrupt_enableSleepOnIsrExit();
MAP_Interrupt_enableMaster();
MAP_WDT_A_startTimer();

//Sleeping when not active
while(1)
{
    MAP_PCM_gotoLPM0();
}

//*****
//SysTick ISR - This ISR will fire every 1s and "service the dog"
//(reset) to prevent a watchdog timeout
//*****

void systick_isr(void)
{
    MAP_WDT_A_clearTimer();
}

//*****
//GPIO ISR for button press - When a button is pressed
//*****

void gpio_isr(void)
{
    uint32_t status;

    status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P1);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, status);

    if(status & GPIO_PIN1)
    {
        MAP_SysTick_disableInterrupt();
    }
}
```



```
//*****
```

6.7 TIMER32

The Timer32 system consists of two identical channels of 32-bit (or 16-bit) timers. The timers may be configured for three different modes [SLAU356A, 2015].

- **Free running mode:** In this mode the counter counts down from a preloaded value to zero. When zero is reached the counter returns to the preset value and continues counting down on each clock pulse.
- **Periodic timer mode:** In this mode the counter counts down from a preloaded value to zero. When zero is reached the counter generates an interrupt and returns to the preset value and continues counting down on each clock pulse. This results in a periodic interrupt at an interval set by the preset counter value.
- **One shot timer mode:** In this mode the counter counts down from a preloaded value to zero. When zero is reached the counter generates an interrupt and stops.

In all modes, the counter decrements by one on each incoming clock pulse. The clock source may be prescaled by a factor of 1, 16, or 256.

6.7.1 REGISTERS

The Timer32 channels are supported and configured by the following registers:

- **T32LOAD1:** Timer 1 Load Register
- **T32VALUE1:** Timer 1 Current Value Register
- **T32CONTROL1:** Timer 1 Timer Control Register
- **T32INTCLR1:** Timer 1 Interrupt Clear Register
- **T32RIS1:** Timer 1 Raw Interrupt Status Register
- **T32MIS1:** Timer 1 Interrupt Status Register
- **T32BGLOAD1:** Timer 1 Background Load Register
- **T32LOAD2:** Timer 2 Load Register
- **T32VALUE2:** Timer 2 Current Value Register
- **T32CONTROL2:** Timer 2 Timer Control Register

- **T32INTCLR2:** Timer 2 Interrupt Clear Register
- **T32RIS2:** Timer 2 Raw Interrupt Status Register
- **T32MIS2:** Timer 2 Interrupt Status Register
- **T32BGLOAD2:** Timer 2 Background Load Register

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

6.7.2 DRIVERLIB APIS

DriverLib provides a library of helpful APIs to support the configuration and operation of Timer32 [DriverLib, 2015]:

- void Timer32_clearInterruptFlag (uint32_t timer)
- void Timer32_disableInterrupt (uint32_t timer)
- void Timer32_enableInterrupt (uint32_t timer)
- uint32_t Timer32_getInterruptStatus (uint32_t timer)
- uint32_t Timer32_getValue (uint32_t timer)
- void Timer32_haltTimer (uint32_t timer)
- void Timer32_initModule (uint32_t timer, uint32_t preScaler, uint32_t resolution, uint32_t mode)
- void Timer32_registerInterrupt (uint32_t timerInterrupt, void(*intHandler)(void))
- void Timer32_setCount (uint32_t timer, uint32_t count)
- void Timer32_setCountInBackground (uint32_t timer, uint32_t count)
- void Timer32_startTimer (uint32_t timer, bool oneShot)
- void Timer32_unregisterInterrupt (uint32_t timerInterrupt)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 8: In the following example, Timer32 is configured as a down counter to generate interrupts [DriverLib, 2015].

```

//*****
//This example is used with permission from:
//MSP432 Peripheral Driver Library User's Guide [DriverLib]
//www.TI.com
//*****

int main(void)
{
volatile uint32_t curValue;

//Holding the Watchdog
MAP_WDT_A_holdTimer();

//Initializing Timer32 in 32-bit free-run mode.
//The timer has a maximum value of 0xFFFFFFFF.
MAP_Timer32_initModule(TIMER32_0_MODULE, TIMER32_PRESCALER_256,
                      TIMER32_32BIT, TIMER32_FREE_RUN_MODE);

//Starting the timer
MAP_Timer32_startTimer(TIMER32_0_MODULE, true);
while(1)
{
//Getting the current value of the Timer32
curValue = MAP_Timer32_getValue(TIMER32_0_MODULE);
}
}

//*****

```

Example 9: In this example Timer32 is configured for one shot mode with the MCLK, scaled by 256 serving as the time source.

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without

```

```
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
```

304 6. TIME-RELATED SYSTEMS

```
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 Timer32 - Free Run One Shot
//
//Description: In this very simple code example, one of the Timer32
//modules is setup in 32-bit free-run mode and started in one shot mode.
//This means that the timer starts at UINT32_MAX (0xFFFFFFFF) and runs to
//0. Once the timer reaches zero, it halts (one shot). The Timer32 is
//sourced by MCLK and in this example is configured to have a prescaler
//of 256 every one tick of the Timer32 module is 256 ticks of MCLK).
//
//
//              MSP432P401
//              -----
//              /|\|
//              ||
//              --|RST
//              |
//              |
//              |
//              |
//              |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>
#include <stdbool.h>

int main(void)
{
volatile uint32_t curValue;

//Holding the Watchdog
```

```

MAP_WDT_A_holdTimer();

//Initializing Timer32 in module in 32-bit free-run mode
//(with max value of 0xFFFFFFFF
MAP_Timer32_initModule(TIMER32_0_MODULE, TIMER32_PRESCALER_256,
                      TIMER32_32BIT, TIMER32_FREE_RUN_MODE);

//Starting the timer
MAP_Timer32_startTimer(TIMER32_0_MODULE, true);

while(1)
{
  //Getting the current value of the Timer32
  curValue = MAP_Timer32_getValue(TIMER32_0_MODULE);
}
}

//*****

```

Example 10: In this example Timer32 is started with a button press, an LED is then toggled for a timer period. The LED is turned off at the end of the period.

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived

```

306 6. TIME-RELATED SYSTEMS

```
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//
//MSP432 Timer32 - Periodic LED Blink
//
//Description: Starts timer on a button press, toggles GPIO/LED for
//a timer period, and then turns off LED at end of period.
//
```

```

//          MSP432P401
//          -----
//          /\|
//          | |
//          --|RST          P1.1 |<--- Switch
//          |
//          |
//          |          P1.0 |---> LED
//          |
//          |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>
#include <stdbool.h>

int main(void)
{
//Halting the Watchdog
MAP_WDT_A_holdTimer();

//Setting MCLK to REF0 at 128Khz for LF mode
MAP_CS_setReferenceOscillatorFrequency(CS_REF0_128KHZ);
MAP_CS_initClockSignal(CS_MCLK, CS_REF0CLK_SELECT, CS_CLOCK_DIVIDER_1);
MAP_PCM_setPowerState(PCM_AM_LF_VCORE0);

//Configuring GPIO
MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P1, GPIO_PIN1);
MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, GPIO_PIN1);
MAP_GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1);

//Configuring Timer32 to 128000 (1s) of MCLK in periodic mode

```


308 6. TIME-RELATED SYSTEMS

```
MAP_Timer32_initModule(TIMER32_0_MODULE, TIMER32_PRESCALER_1,
                      TIMER32_32BIT, TIMER32_PERIODIC_MODE);

//Enabling interrupts
MAP_Interrupt_enableInterrupt(INT_PORT1);
MAP_Interrupt_enableInterrupt(INT_T32_INT1);
MAP_Interrupt_enableMaster();

//Sleeping when not in use
while(1)
{
    MAP_PCM_gotoLPMO();
}

//*****
//GPIO ISR
//*****

void gpio_isr(void)
{
    uint32_t status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P1);
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P1, status);

    if(GPIO_PIN1 & status)
    {
        MAP_GPIO_disableInterrupt(GPIO_PORT_P1, GPIO_PIN1);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
        MAP_Timer32_setCount(TIMER32_0_MODULE, 128000);
        MAP_Timer32_enableInterrupt(TIMER32_0_MODULE);
        MAP_Timer32_startTimer(TIMER32_0_MODULE, true);
    }
}

//*****
//Timer32 ISR
//*****

void timer32_isr(void)
```

```

{
MAP_Timer32_clearInterruptFlag(TIMER32_0_MODULE);
MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
MAP_GPIO_enableInterrupt(GPIO_PORT_P1, GPIO_PIN1);
}

//*****

```

6.8 TIMER_A

The Timer_A subsystem consists of a complement of identical 16-bit timers. The individual timers are designated Timer_A0 through Timer_A3. The heart of each timer is a 16-bit timer register TAxR. Where “x” corresponds to the Timer_A timer number. The block diagram of a Timer_A channel is provided in Figure 6.5 [SLAU356A, 2015].

The TAxR register can be clocked from several sources including the TAxCLK, ACLK, SMCLK, and INCLK. The specific clock source is determined by the Timer_A Clock Source Select (TASSEL) bits. The timer source is routed to the TAxR timer via two stages of dividers. The first divider may be set to 1, 2, 4, or 8 using the ID bits. The second divider may be set from 1–7 using the IDEX bits [SLAU356A, 2015].

The TAxR timer may be configured for various count modes using the Mode Control (MC) bits. These modes include stop (00), up (01), continuous (10), and up/down (11). As can be seen in Figure 6.5, each TAxR timer is equipped with seven identical capture/compare blocks designated CCR0 through CCR7 [SLAU356A, 2015].

As seen in Figure 6.5 each capture/compare block is equipped with hardware for supporting the capture mode and the compare mode. The capture portion of the hardware is selected by setting the Capture Mode (CAP) bit to logic one. The purpose of the capture mode is to capture key events of an input signal such as a rising edge, a falling edge, or any edge. The specific event of interest is selected using the Capture Mode (CM) bits. The input signal for the capture system is selected by the CCIS bits. The input may be from an external pin (CCInA or CCInB), ground, or Vcc. A specific input is selected using the Capture/Compare Input Select (CCIS) bits. The “n” corresponds to the specific capture/compare block in use (0–6). When the specified edge occurs on the input signal, the current count in the 16-bit timer (TAxR) is captured to the TAxCCRn register. Also, the Capture/Compare Interrupt Flag (CCIFG) interrupt occurs.

The compare hardware is selected by setting the CAP bit to logic zero. The compare hardware is used to generate output signals such as a periodic signal, a pulse, or a pulse width modulated signal. The signals are constructed as a series of key events (toggling, setting, or resetting) at specific points in time. The desired event time is loaded in advance to the TAxCCRn register. The value of the current value of the 16-bit Timer (TAxR) is constantly compared to the preset value of the TAxCCRn register. When the two register values match, the prescribed key event occurs

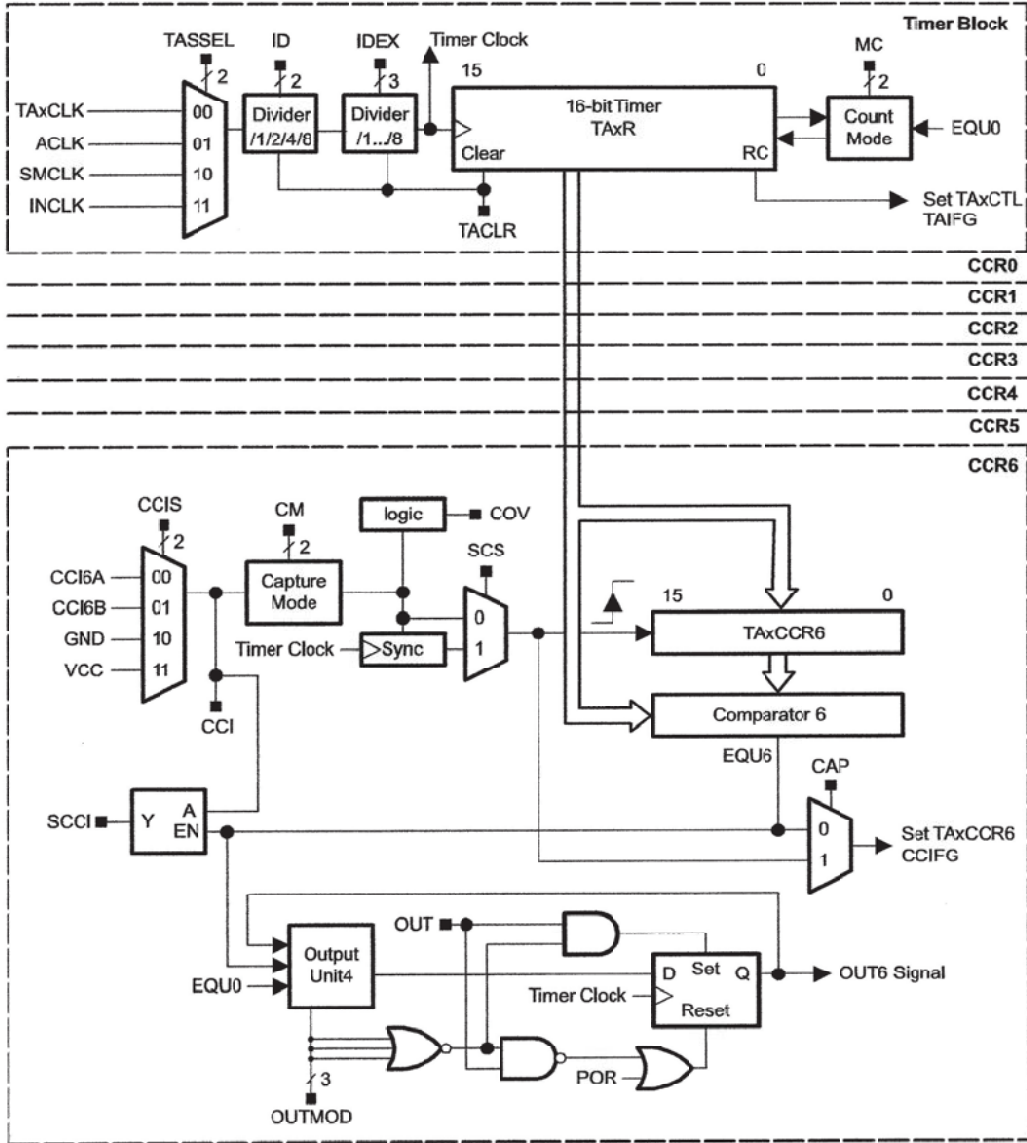


Figure 6.5: Timer_A block diagram [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

and the CCIFG interrupt is triggered. A signal is built up as a series of key events. The available output modes, selected by the OUTMOD bits include the following [SLAU356A, 2015].

- **Output, OUTMODx [000]:** The output signal (OUTn) is updated when OUT is updated.
- **Set, OUTMODx [001]:** The output signal (OUTn) is set when the TAxR 16-bit timer value equals TAxCCRn.
- **Toggle/Reset, OUTMODx [010]:** The output signal (OUTn) is toggled when the TAxR 16-bit timer value equals TAxCCRn and subsequently resets when the timer reaches the TAxCCR0 value.
- **Set/Reset, OUTMODx [011]:** The output signal (OUTn) is set when the TAxR 16-bit timer value equals TAxCCRn and subsequently resets when the timer reaches the TAxCCR0 value.
- **Toggle, OUTMODx [100]:** The output signal (OUTn) toggles when the TAxR 16-bit timer value equals TAxCCRn.
- **Reset, OUTMODx [101]:** The output signal (OUTn) resets when the TAxR 16-bit timer value equals TAxCCRn.
- **Toggle/Set, OUTMODx [110]:** The output signal (OUTn) is toggled when the TAxR 16-bit timer value equals TAxCCRn and subsequently sets when the timer reaches the TAxCCR0 value.
- **Reset/Set, OUTMODx [111]:** The output signal (OUTn) is reset when the TAxR 16-bit timer value equals TAxCCRn and subsequently sets when the timer reaches the TAxCCR0 value.

6.8.1 REGISTERS

The Timer_A system channels are supported by several registers including [SLAU356A, 2015]:

- TAxCTL Timer_Ax Control
- TAxCCTL0 Timer_Ax Capture/Compare Control 0 to TAxCCTL6 Timer_Ax Capture/Compare Control 6
- TAxR Timer_Ax Counter
- TAxCCR0 Timer_Ax Capture/Compare 0 to TAxCCR6 Timer_Ax Capture/Compare 6
- TAxIV Timer_Ax Interrupt Vector
- TAxEX0 Timer_Ax Expansion 0

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

6.8.2 DRIVERLIB APIS

The Timer_A system is supported by a number of DriverLib data structures, type definitions, and APIs [[DriverLib, 2015](#)]:

The data structures include:

- struct _Timer_A_CaptureModeConfig
- struct _Timer_A_CompareModeConfig
- struct _Timer_A_ContinuousModeConfig
- struct _Timer_A_PWMConfig
- struct _Timer_A_UpDownModeConfig
- struct _Timer_A_UpModeConfig

The type definitions include:

- typedef struct _Timer_A_CaptureModeConfig Timer_A_CaptureModeConfig
- typedef struct _Timer_A_CompareModeConfig Timer_A_CompareModeConfig
- typedef struct _Timer_A_ContinuousModeConfig Timer_A_ContinuousModeConfig
- typedef struct _Timer_A_PWMConfig Timer_A_PWMConfig
- typedef struct _Timer_A_UpDownModeConfig Timer_A_UpDownModeConfig
- typedef struct _Timer_A_UpModeConfig Timer_A_UpModeConfig

The functions include:

- void Timer_A_clearCaptureCompareInterrupt(uint32_t timer, uint_fast16_t captureCompareRegister)
- void Timer_A_clearInterruptFlag(uint32_t timer)
- void Timer_A_clearTimer(uint32_t timer)
- void Timer_A_configureContinuousMode(uint32_t timer, const Timer_A_ContinuousModeConfig *config)
- void Timer_A_configureUpDownMode(uint32_t timer, const Timer_A_UpDownModeConfig *config)
- void Timer_A_configureUpMode(uint32_t timer, const Timer_A_UpModeConfig *config)

- void Timer_A_disableCaptureCompareInterrupt(uint32_t timer, uint_fast16_t captureCompareRegister)
- void Timer_A_disableInterrupt(uint32_t timer)
- void Timer_A_enableCaptureCompareInterrupt(uint32_t timer, uint_fast16_t captureCompareRegister)
- void Timer_A_enableInterrupt(uint32_t timer) void Timer_A_generatePWM(uint32_t timer, const Timer_A_PWMConfig *config)
- uint_fast16_t Timer_A_getCaptureCompareCount(uint32_t timer, uint_fast16_t captureCompareRegister)
- uint32_t Timer_A_getCaptureCompareEnabledInterruptStatus(uint32_t timer, uint_fast16_t captureCompareRegister)
- uint32_t Timer_A_getCaptureCompareInterruptStatus(uint32_t timer, uint_fast16_t captureCompareRegister, uint_fast16_t mask)
- uint16_t Timer_A_getCounterValue(uint32_t timer)
- uint32_t Timer_A_getEnabledInterruptStatus(uint32_t timer)
- uint32_t Timer_A_getInterruptStatus(uint32_t timer)
- uint_fast8_t Timer_A_getOutputForOutputModeOutBitValue (uint32_t timer, uint_fast16_t captureCompareRegister)
- uint_fast8_t Timer_A_getSynchronizedCaptureCompareInput (uint32_t timer, uint_fast16_t captureCompareRegister, uint_fast16_t synchronizedSetting)
- void Timer_A_initCapture(uint32_t timer, const Timer_A_CaptureModeConfig *config)
- void Timer_A_initCompare(uint32_t timer, const Timer_A_CompareModeConfig *config)
- void Timer_A_registerInterrupt (uint32_t timer, uint_fast8_t interruptSelect, void(*intHandler)(void))
- void Timer_A_setCompareValue(uint32_t timer, uint_fast16_t compareRegister, uint_fast16_t compareValue)
- void Timer_A_setOutputForOutputModeOutBitValue(uint32_t timer, uint_fast16_t captureCompareRegister, uint_fast8_t outputModeOutBitValue)
- void Timer_A_startCounter(uint32_t timer, uint_fast16_t timerMode)

- void Timer_A_stopTimer(uint32_t timer)
- void Timer_A_unregisterInterrupt(uint32_t timer, uint_fast8_t interruptSelect)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 11: In this example, Timer_A is used in the up mode to toggle an LED every 0.5 s.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
```

```

// --COPYRIGHT--
//*****
//
//          MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 PWM TA1.1-2, Up/Down Mode, DCO SMCLK
//
//Description: Toggle P1.0 using software and TA_0 ISR. Timer0_A is
//configured for up mode, thus the timer overflows when TAR counts
//to CCR0. In this example, CCR0 is loaded with 0x2DC6 which makes the
//LED toggle every half a second.
//
//ACLK = n/a, MCLK = SMCLK = default DCO ~1MHz
//TACLK = SMCLK/64
//
//          MSP432P401
//          -----
//  /|\|          |
//  | |          |
//  --|RST       |
//  |           |
//  |           P1.0|-->LED
//  |           |
//
//Author: Timothy Logan

```



```

//*****

//DriverLib Includes
#include "driverlib.h"

//Application Defines
#define TIMER_PERIOD    0x2DC6

//Timer_A UpMode Configuration Parameter
const Timer_A_UpModeConfig upConfig =
{
    TIMER_A_CLOCKSOURCE_SMCLK,           //SMCLK Clock Source
    TIMER_A_CLOCKSOURCE_DIVIDER_64,     //SMCLK/1 = 3MHz
    TIMER_PERIOD,                        //5000 tick period
    TIMER_A_TAIE_INTERRUPT_DISABLE,     //Disable Timer interrupt
    TIMER_A_CCIE_CCRO_INTERRUPT_ENABLE , //Enable CCRO interrupt
    TIMER_A_DO_CLEAR                     //Clear value
};

int main(void)
{
    //Stop WDT
    MAP_WDT_A_holdTimer();

    //Configuring P1.0 as output
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

    //Configuring Timer_A1 for Up Mode
    MAP_Timer_A_configureUpMode(TIMER_A1_MODULE, &upConfig);

    //Enabling interrupts and starting the timer
    MAP_Interrupt_enableSleepOnIsrExit();
    MAP_Interrupt_enableInterrupt(INT_TA1_0);
    MAP_Timer_A_startCounter(TIMER_A1_MODULE, TIMER_A_UP_MODE);

    //Enabling MASTER interrupts
    MAP_Interrupt_enableMaster();

```

```

//Sleeping when not in use
while(1)
{
    MAP_PCM_gotoLPM0();
}
}

//*****
void timer_a_0_isr(void)
{
MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
MAP_Timer_A_clearCaptureCompareInterrupt(TIMER_A1_MODULE,
                                           TIMER_A_CAPTURECOMPARE_REGISTER_0);
}
}

```

```

//*****

```

Example 12: In this example an Interrupt Service Routine (ISR) is triggered every time the timer overflows. An LED is toggled within the ISR.

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
//  notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
//  notice, this list of conditions and the following disclaimer in the
//  documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//

```

```

//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 Timer_A - Continuous Overflow Interrupt
//
//Description: Toggle P1.0 using software and the Timer0_A overflow ISR.
//In this example an ISR triggers when TA overflows.
Inside the ISR P1.0
//is toggled.
Toggle rate is exactly 0.5Hz.
//

```

```

//ACLK = TACLK = 32768Hz, MCLK = SMCLK = DCO = 3MHz
//
//          MSP432P401
//          -----
//          /\| |
//          | | |
//          --|RST      P1.0 |----> P1.0 LED
//          | | |
//          | | |
//          | | |
//          | | |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>

//Statics
const Timer_A_ContinuousModeConfig continuousModeConfig =
{
    TIMER_A_CLOCKSOURCE_ACLK,          //ACLK Clock Source
    TIMER_A_CLOCKSOURCE_DIVIDER_1,    //ACLK/1 = 32.768khz
    TIMER_A_TAIE_INTERRUPT_ENABLE,    //Enable Overflow ISR
    TIMER_A_DO_CLEAR                  //Clear Counter
};

int main(void)
{
    //Stop watchdog timer
    MAP_WDT_A_holdTimer();

    //Configuring P1.0 as output
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

```

320 6. TIME-RELATED SYSTEMS

```
//Starting and enabling ACLK (32kHz)
MAP_CS_setReferenceOscillatorFrequency(CS_REFO_128KHZ);
MAP_CS_initClockSignal(CS_ACLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_4);

//Configuring Continuous Mode
MAP_Timer_A_configureContinuousMode(TIMER_A0_MODULE,
                                     &continuousModeConfig);

//Enabling interrupts and going to sleep
MAP_Interrupt_enableSleepOnIsrExit();
MAP_Interrupt_enableInterrupt(INT_TAO_N);

//Enabling MASTER interrupts
MAP_Interrupt_enableMaster();

//Starting the Timer_A0 in continuous mode
MAP_Timer_A_startCounter(TIMER_A0_MODULE, TIMER_A_CONTINUOUS_MODE);

while(1)
{
    MAP_PCM_gotoLPM0();
}

//*****
//TIMER_A interrupt vector service routine
//*****

void timer_a_0_isr(void)
{
    MAP_Timer_A_clearInterruptFlag(TIMER_A0_MODULE);
    MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
}

//*****

Example 13: In this example the input capture feature of Timer_A is used to measure the period
of the clock.

//*****
//    MSP432 DriverLib - v2_20_00_08
```

```
//*****  
//  
//--COPYRIGHT--,BSD_EX  
//Copyright (c) 2013, Texas Instruments Incorporated  
//All rights reserved.  
//  
//Redistribution and use in source and binary forms, with or without  
//modification, are permitted provided that the following conditions  
//are met:  
//- Redistributions of source code must retain the above copyright  
// notice, this list of conditions and the following disclaimer.  
//- Redistributions in binary form must reproduce the above copyright  
// notice, this list of conditions and the following disclaimer in the  
// documentation and/or other materials provided with the distribution.  
//  
//Neither the name of Texas Instruments Incorporated nor the names of  
//its contributors may be used to endorse or promote products derived  
//from this software without specific prior written permission.  
//  
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE  
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,  
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS  
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND  
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR  
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
//DAMAGE.  
// --COPYRIGHT--  
//*****  
//  
//  
//  
//MSP432 CODE EXAMPLE DISCLAIMER  
//  
//MSP432 code examples are self-contained low-level programs that  
//typically demonstrate a single peripheral function or device feature  
//in a highly concise manner. For this the code may rely on the device's
```

322 6. TIME-RELATED SYSTEMS

```
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 Timer_A - VLO Period Capture
//
//Description: Capture a number of periods of the VLO clock and store
//them in an array.
When the set number of periods is captured the
//program is trapped and the LED on P1.0 is toggled.
At this point halt
//the program execution read out the values using the debugger.
//
//ACLK = VLOCLK = 14kHz (typ.), MCLK = SMCLK = default DCO = 3MHz
//
//
//              MSP432P401
//              -----
//              /|\|
//              ||
//              --|RST          P1.0 |---> P1.0 LED
//              |
//              |
//              |
//              |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
```

```
#include <stdint.h>
#define NUMBER_TIMER_CAPTURES      20

//Timer_A Continuous Mode Configuration Parameter
const Timer_A_ContinuousModeConfig continuousModeConfig =
{
    TIMER_A_CLOCKSOURCE_SMCLK,      //SMCLK Clock Source
    TIMER_A_CLOCKSOURCE_DIVIDER_1,  //SMCLK/1 = 3MHz
    TIMER_A_TAIE_INTERRUPT_DISABLE, //Disable Timer ISR
    TIMER_A_SKIP_CLEAR              //Skip Clear Counter
};

//Timer_A Capture Mode Configuration Parameter
const Timer_A_CaptureModeConfig captureModeConfig =
{
    TIMER_A_CAPTURECOMPARE_REGISTER_1, //CC Register 2
    TIMER_A_CAPTUREMODE_RISING_EDGE,   //Rising Edge
    TIMER_A_CAPTURE_INPUTSELECT_CCIxB, //CCIxB Input Select
    TIMER_A_CAPTURE_SYNCHRONOUS,       //Synchronized Capture
    TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE, //Enable interrupt
    TIMER_A_OUTPUTMODE_OUTBITVALUE     //Output bit value
};

//Statics
static volatile uint_fast16_t timerAcaptureValues[NUMBER_TIMER_CAPTURES];
static volatile uint32_t timerAcapturePointer = 0;

int main(void)
{
    //Stop watchdog timer
    MAP_WDT_A_holdTimer();

    //Configuring P1.0 as output
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

    //Setting ACLK = VLO = 14kHz
    MAP_CS_initClockSignal(CS_ACLK, CS_VLOCLK_SELECT, CS_CLOCK_DIVIDER_1);
}
```


324 6. TIME-RELATED SYSTEMS

```
//Configuring Capture Mode
MAP_Timer_A_initCapture(TIMER_A0_MODULE, &captureModeConfig);

//Configuring Continuous Mode
MAP_Timer_A_configureContinuousMode(TIMER_A0_MODULE,
                                     &continuousModeConfig);

//Enabling interrupts and going to sleep
MAP_Interrupt_enableSleepOnIsrExit();
MAP_Interrupt_enableInterrupt(INT_TAO_N);
MAP_Interrupt_enableMaster();

//Starting the Timer_A0 in continuous mode
MAP_Timer_A_startCounter(TIMER_A0_MODULE, TIMER_A_CONTINUOUS_MODE);

MAP_PCM_gotoLPM0();

}

//*****
//TIMERA interrupt vector service routine
//*****

void timer_a_ccr_isr(void)
{
    uint32_t jj;

    timerAcaptureValues[timerAcapturePointer++] =
        MAP_Timer_A_getCaptureCompareCount(TIMER_A0_MODULE,
                                           TIMER_A_CAPTURECOMPARE_REGISTER_1);

    if(timerAcapturePointer >= NUMBER_TIMER_CAPTURES)
    {
        while(1)
        {
            MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
            for(jj=0;jj<10000;jj++);
        }
    }
}
```

}

//*****

6.9 REAL-TIME CLOCK, RTC_C

RTC_C, the MSP432 real-time clock (RTC), provides a clock based on seconds, minutes, hours, etc. The RTC time base is provided by a 32,768 Hz external crystal. This time base is shown as the BCLK in Figure 6.6. The time base is routed to the RTOPS and RT1PS dividers to provide a 1 Hz time base to the time keeping registers. The register contains place holders for seconds (RTCSEC), minutes (RTCMIN), hours (RTCHOUR), day of the week (RTCADOW), day (RTCDAY), month (RTCMON), and year (RTCYEARH, RTCYEARL). Data may be stored in binary coded decimal (BCD) or hexadecimal binary format. BCD represents each digit in a number individually from 0–9 [SLAU356A, 2015].

The RTC_C is also equipped with an alarm function. The alarm is configured for a specific minute (RTCAMIN), hour (RTCAHOUR), day (RTCADAY), and day of the week (RTCADOW). The write operation for RTC control, clock, calendar, prescale, and offset error are key protected [SLAU356A, 2015].

The RTC_C is supported by six prioritized interrupts designated RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, RTCAIFG, and RTCOFIFG. The six interrupt signal flags are combined to provide a single interrupt signal. When an interrupt occurs the interrupt vector register (RTCIV) provides the specific interrupt source [SLAU356A, 2015].

6.9.1 RTC REGISTERS

RTC_C is supported by a complement of registers including [SLAU356A, 2015]:

- RTCCTL0 Real-Time Clock Control 0
- RTCCTL0_L Real-Time Clock Control 0
- RTCCTL0_H Real-Time Clock Control 0
- RTCCTL13 Real-Time Clock Control 1
- RTCCTL1 Real-Time Clock Control 1
- RTCCTL3 Real-Time Clock Control 3
- RTCOCAL Real-Time Clock Offset
- RTCTCMP Real-Time Clock Temperature
- RTCPS0CTL Real-Time Prescale Timer 0

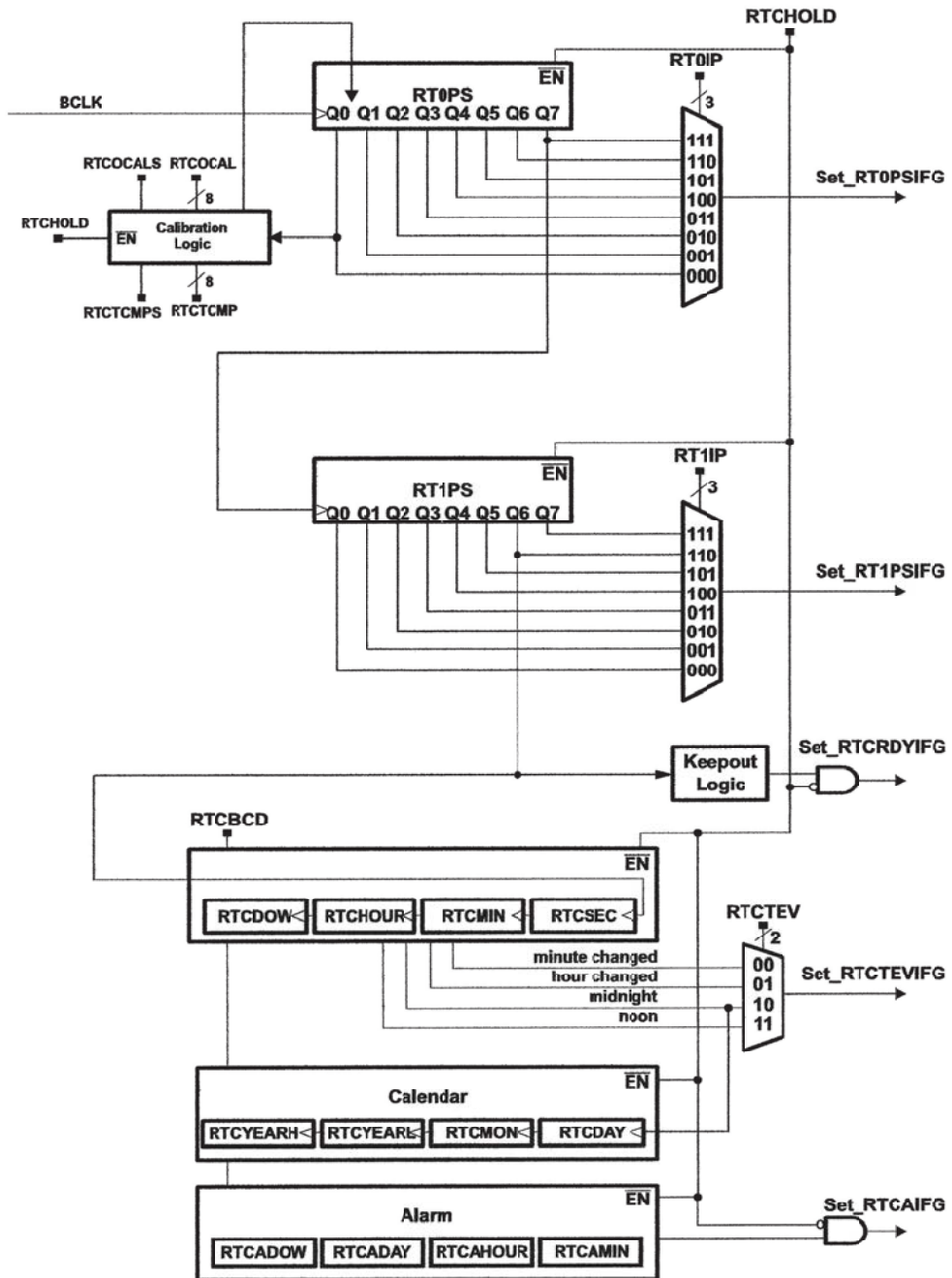


Figure 6.6: Real-Time Clock [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

- RTCPS1CTL Real-Time Prescale Timer 1
- RTCPS Real-Time Prescale Timer 0
- RTCPS0 Real-Time Prescale Timer 0
- RTCPS1 Real-Time Prescale Timer 1
- RTCIV Real-Time Clock Interrupt Vector
- RTCSEC Real-Time Clock Seconds
- RTCMIN Real-Time Clock Minutes
- RTCTIM1 Real-Time Clock Hour, Day of Week
- RTCHOUR Real-Time Clock Hour
- RTCDOW Real-Time Clock Day of Week
- RTCDATE Real-Time Clock Date
- RTCDAY Real-Time Clock Day of Month
- RTCMON Real-Time Clock Month
- RTCYEAR Real-Time Clock Year
- RTCAMINHR Real-Time Clock Minutes, Hour Alarm
- RTCAMIN Real-Time Clock Minutes Alarm
- RTCAHOUR Real-Time Clock Hours Alarm
- RTCADOWDAY Real-Time Clock Day of Week, Day of Month Alarm
- RTCADOW Real-Time Clock Day of Week Alarm
- RTCADAY Real-Time Clock Day of Month Alarm
- RTCBIN2BCD Binary-to-BCD conversion register
- RTCBCD2BIN BCD-to-binary conversion register

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [[SLAU356A](#), 2015] and will not be repeated here.

6.9.2 RTC DRIVERLIB API SUPPORT

RTC_C is supported by a complement of DriverLib APIs [[DriverLib, 2015](#)]:

- void RTC_C_clearInterruptFlag(uint_fast8_t interruptFlagMask)
- uint16_t RTC_C_convertBCDToBinary(uint16_t valueToConvert)
- uint16_t RTC_C_convertBinaryToBCD(uint16_t valueToConvert)
- void RTC_C_definePrescaleEvent(uint_fast8_t prescaleSelect, uint_fast8_t prescaleEventDivider)
- void RTC_C_disableInterrupt(uint8_t interruptMask)
- void RTC_C_enableInterrupt(uint8_t interruptMask)
- RTC_C_Calendar RTC_C_getCalendarTime(void)
- uint_fast8_t RTC_C_getEnabledInterruptStatus(void)
- uint_fast8_t RTC_C_getInterruptStatus(void)
- uint_fast8_t RTC_C_getPrescaleValue(uint_fast8_t prescaleSelect)
- void RTC_C_holdClock(void)
- void RTC_C_initCalendar(const RTC_C_Calendar *calendarTime, uint_fast16_t formatSelect)
- void RTC_C_registerInterrupt(void(*intHandler)(void))
- void RTC_C_setCalendarAlarm(uint_fast8_t minutesAlarm, uint_fast8_t hoursAlarm, uint_fast8_t dayOfWeekAlarm, uint_fast8_t dayOfMonthAlarm)
- void RTC_C_setCalendarEvent(uint_fast16_t eventSelect)
- void RTC_C_setCalibrationData(uint_fast8_t offsetDirection, uint_fast8_t offsetValue)
- void RTC_C_setCalibrationFrequency(uint_fast16_t frequencySelect)
- void RTC_C_setPrescaleValue(uint_fast8_t prescaleSelect, uint_fast8_t prescaleCounterValue)
- bool RTC_C_setTemperatureCompensation(uint_fast16_t offsetDirection, uint_fast8_t offsetValue)
- void RTC_C_startClock(void)

- void RTC_C_unregisterInterrupt(void)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 14: This code example shows how to configure the RTC_C module and create a calendar event.

```

//*****
//This example is used with permission from:
//MSP432 Peripheral Driver Library User's Guide [DriverLib]
//This code example shows how to configure the RTC_C module and create a
//calendar event.
//[www.ti.com]
//*****

//Configuration structure to set the date:
//Time is November 12th 1955 10:03:00 PM

const RTC_C_Calendar currentTime =
{
    0x00,
    0x03,
    0x22,
    0x12,
    0x11,
    0x1955
};

//Initializing RTC with current time as described in time in
//definitions section
MAP_RTC_C_initCalendar(&currentTime, RTC_C_FORMAT_BCD);

//Setup Calendar Alarm for 10:04pm (for the flux capacitor)
MAP_RTC_C_setCalendarAlarm(0x04, 0x22, RTC_C_ALARMCONDITION_OFF,
                           RTC_C_ALARMCONDITION_OFF);

//Specify an interrupt to assert every minute
MAP_RTC_C_setCalendarEvent(RTC_C_CALENDAREVENT_MINUTECHANGE);

//Enable interrupt for RTC Ready Status, which asserts when the RTC
//Calendar registers are ready to read.

```

```

//Also, enable interrupts for the Calendar alarm and Calendar event.

MAP_RTC_C_clearInterruptFlag(RTC_C_CLOCK_READ_READY_INTERRUPT |
                             RTC_C_TIME_EVENT_INTERRUPT |
                             RTC_C_CLOCK_ALARM_INTERRUPT);
MAP_RTC_C_enableInterrupt(RTC_C_CLOCK_READ_READY_INTERRUPT |
                           RTC_C_TIME_EVENT_INTERRUPT |
                           RTC_C_CLOCK_ALARM_INTERRUPT);

//Start RTC Clock
MAP_RTC_C_startClock();

```

```

//*****

```

Example 15: This example demonstrates the use of the RTC_C and generates interrupts every second and every minute.

```

//*****
//   MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE

```

```

//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432 RTC - Calendar Mode
//
//Interruptions every 1s, 1m, and 5th day of week at 5:00pm
//
//Description: This program demonstrates the RTC mode by triggering an
//interrupt every second and minute.
This code toggles P1.0 every second.
//This code recommends an external LFXT1 crystal for RTC accuracy.
//Note that if XT1 is not present the code loops in an infinite loop.
//
//ACLK = LFXT1 = 32768Hz, MCLK = default DCO of 3MHz
//

```


332 6. TIME-RELATED SYSTEMS

```
//          MSP432P401
//          -----
//          /|\|
//          | |
//          --|RST          P1.0 |---> P1.0 LED
//          |          PJ.0 LFXIN |-----
//          |          |          |
//          |          |          < 32khz xTal >
//          |          |          |
//          |          PJ.1 LFXOUT |-----
//
//Author: Timothy Logan
// //*****

//DriverLib Includes
#include "driverlib.h"

//Statics
static volatile RTC_C_Calendar newTime;

//Time is November 12th 1955 10:03:00 PM
const RTC_C_Calendar currentTime =
{
    0x00,
    0x03,
    0x22,
    0x12,
    0x11,
    0x1955
};

int main(void)
{
    //Halting WDT
    MAP_WDT_A_holdTimer();

    //Configuring pins for peripheral/crystal usage and LED for output
    MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_PJ,
```

```
    GPIO_PIN0 | GPIO_PIN1, GPIO_PRIMARY_MODULE_FUNCTION);
MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0);

//Setting the external clock frequency.
This API is optional, but will
//come in handy if the user ever wants to use the getMCLK/getACLK/etc
//functions
CS_setExternalClockSourceFrequency(32000,48000000);

//Starting LFXT in non-bypass mode without a timeout.
CS_startLFXT(false);

//Initializing RTC with current time as described in time in
//definitions section
MAP_RTC_C_initCalendar(&currentTime, RTC_C_FORMAT_BCD);

//Setup Calendar Alarm for 10:04pm (for the flux capacitor)
MAP_RTC_C_setCalendarAlarm(0x04, 0x22, RTC_C_ALARMCONDITION_OFF,
                          RTC_C_ALARMCONDITION_OFF);

//Specify an interrupt to assert every minute
MAP_RTC_C_setCalendarEvent(RTC_C_CALENDAREVENT_MINUTECHANGE);

//Enable interrupt for RTC Ready Status, which asserts when the RTC
//Calendar registers are ready to read.
Also, enable interrupts for the
//Calendar alarm and Calendar event.
MAP_RTC_C_clearInterruptFlag(RTC_C_CLOCK_READ_READY_INTERRUPT |
                             RTC_C_TIME_EVENT_INTERRUPT |
                             RTC_C_CLOCK_ALARM_INTERRUPT);
MAP_RTC_C_enableInterrupt(RTC_C_CLOCK_READ_READY_INTERRUPT |
                          RTC_C_TIME_EVENT_INTERRUPT |
                          RTC_C_CLOCK_ALARM_INTERRUPT);

//Start RTC Clock
MAP_RTC_C_startClock();

//Enable interrupts and go to sleep.
MAP_Interrupt_enableInterrupt(INT_RTC_C);
MAP_Interrupt_enableSleepOnIsrExit();
```

334 6. TIME-RELATED SYSTEMS

```
MAP_Interrupt_enableMaster();

while(1)
{
    MAP_PCM_gotoLPM3();
}

//*****
//RTC ISR
//*****

void rtc_isr(void)
{
    uint32_t status;

    status = MAP_RTC_C_getEnabledInterruptStatus();
    MAP_RTC_C_clearInterruptFlag(status);

    if(status & RTC_C_CLOCK_READ_READY_INTERRUPT)
    {
        MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }

    if(status & RTC_C_TIME_EVENT_INTERRUPT)
    {
        //Interrupts every minute - Set breakpoint here
        __no_operation();
        newTime = MAP_RTC_C_getCalendarTime();
    }

    if(status & RTC_C_CLOCK_ALARM_INTERRUPT)
    {
        //Interrupts at 10:04pm
        __no_operation();
    }
}

//*****
```

6.10 LABORATORY EXERCISE: GENERATION OF VARYING PULSE WIDTH MODULATED SIGNALS TO CONTROL DC MOTORS

In this laboratory the parameters of a pulse width modulated signal is set by two potentiometers. The first potentiometer is used to set the frequency from 1 kHz (0 VDC) to 10 kHz (3.3 VDC). The second potentiometer sets the duty cycle of the signal from 5% (0 VDC) to 95% (3.3 VDC).

Your solution should include a UML activity diagram and also well-documented code.

6.11 SUMMARY

This chapter provided an introduction to the time related features onboard the MSP432. We began with an introduction to MSP432 clock system followed by a discussion of the Watchdog Timer system. We then examined the MSP432 timer systems including Timer32 and the complement of Timer_A channels. We concluded by investigating the Real-Time clock, RTC_C. Throughout the chapter, examples were used to illustrate related concepts.

6.12 REFERENCES AND FURTHER READING

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015. 277, 278, 290, 301, 312, 314, 328, 329

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. 274, 276, 277, 289, 290, 291, 300, 301, 309, 310, 311, 325, 326, 327

6.13 CHAPTER PROBLEMS

Fundamental

1. Describe the different time base sources available on the MSP432. Identify whether the sources are external or internal to the MSP432.
2. What is the significance of a 32,768 Hz time base?
3. What is the purpose of Watchdog Timer?
4. Describe the two different modes of the Watchdog Timer. In what power modes are these different WDT modes available?
5. What is the password value and where should you write it to access the Watchdog Timer system control register?
6. Describe the three modes of operation for Timer32.

7. Describe the modes of operation for `Timer_A`.
8. What are the timing sources available for `Timer_A`? How is a specific timing source selected?
9. Describe the different operating modes of `Timer_A`. Provide an example application where each mode might be used.
10. What is the time base for the Real-Time Clock? Is it preferable to use a crystal or resonator for the time base? Explain.

Advanced

1. Program your MSP432 to generate clock signal frequency of 1.2 MHz.
2. Program your MSP432 controller to accept a pulse on the P1.0 pin and compute the pulse width.
3. Given a periodic pulse width input signal, write a segment of code to compute the duty cycle using the input capture interrupt system of the MSP432 controller.
4. Program the MSP432 controller to generate a pulse (0–3.3 V and back down to 0 V) with 2 ms width using the `Timer_A` system.
5. Program your MSP432 using `Timer_A` system to generate a pulse width modulated signal with frequency of 50 Hz and duty cycle of 40%.

Challenging

1. Program your MSP432 to accept any periodic signal input with varying frequency ranging from 10–1000 Hz and compute the input signal frequency.
2. Write a program that only activates itself if your MSP432 controller receives a 200 usec pulse (10% tolerance on the pulse width) from an external device on a specific pin, updates the number of times the designated pulse was received, displays the number on an LCD display unit for 5 s, and “sleeps” until the next pulse arrives.

Resets and Interrupts

Objectives: After reading this chapter, the reader should be able to:

- describe MSP432 resets and their functions;
- explain the general concept of and the need for interrupts;
- describe in general terms the steps required to implement an interrupt service routine;
- identify MSP432 microcontroller maskable and non-maskable interrupts;
- illustrate how the priority among resets and interrupts is determined in the MSP432 microcontroller;
- explain the process to identify the source of resets and interrupts;
- describe the operation of the MSP432 Nested Vector Interrupt Controller (NVIC);
- describe the process to service interrupts; and
- properly configure the MSP432 microcontroller and write interrupt service routines to respond to interrupts.

7.1 OVERVIEW

In computer operation, there is often a need to bring the computer back to a known state due to program or system errors or simply because it serves the purpose of an application. Bringing the computer back to a known state involves re-initializing registers, executing start up instructions, and setting up peripheral devices, including input and output systems, to default states. This process and the source that caused the process is called a reset.

In other applications, there is a need to stop executing the current task and take care of an urgent, higher priority request made by an internal device, external signal, or the result of a current software operation. These requests are called interrupts.

Resets and interrupts are closely related. The process of bringing the computer back to a known state and performing an unplanned service as a response to an urgent request is almost identical as we will see in this chapter.

7.2 BACKGROUND

Typical embedded systems operate in environments where electrical and mechanical noise sources abound. These noise sources can interfere with the proper function of the microcontroller in an embedded system, which can appear during the operation of the system as skipping intended instructions and unintentionally changing the contents of registers.

One of the primary means to remedy such malfunctions in the MSP432 microcontroller is the Watchdog Timer System we introduced in Chapter 6. By forcing the program to update a designated register periodically, one can make sure that intended instructions are executed in the proper order, and if not, make the controller reset and resume normal operation.

Interrupts are requests whose time of occurrence is not known in advance, but the programmer can plan to service them when they occur. For example, suppose that you know a user will push an external button to indicate that a task should be finished during the course of an operation of your MSP432 microcontroller but do not know when it will occur. You can write a separate “program” that will respond to the event appropriately.

There are two ways for a microcontroller to detect the time of an event. The first method, called polling, relies on the resources of the controller to continuously monitor whether or not an event has occurred. This can be in the form of checking a flag continuously to see the flag status change (bit changes from 1–0, or vice versa) or the change of the logic level on an input pin. The second method, which is the focus of this chapter, is using the interrupt method. In this approach, the processor performs other critical tasks (use resources optimally) or even placed in a low power mode to save power and only react to an event when it occurs. Naturally, the polling method is simple to implement compared to the configuration required to implement the interrupt method. The benefit, however, of the interrupt method is the conservation of limited, precious resources.

You can imagine how inefficient your time spent in a course would be, if your instructor would occasionally suspend their lecture while each student was sequentially asked if she or he had any question on the ongoing lecture. This is similar to the technique of microcontroller polling. Instead, students “interrupt” the normal flow of a lecture when they have an important question to ask about the material. This is similar to the microcontroller interrupt technique.

7.3 MSP432 RESETS

The MSP432, hosting the Cortex M4F processor, is equipped with different classes of resets designated 0–4. The Class 0 reset is the highest priority. Each class of reset generates associated actions and also incorporates lower priority (higher class number) reset actions. Here is a brief summary of resets and associated actions [SLAU356A, 2015].

- **Class 0:** The Class 0 reset is also known as the Power On/Off Reset (POR) Class. It results due to a power-on event or loss of power to the device, an exception from the Power Supply System (PSS), exit from LPM 3.5 or 4.5 modes of operation, or user initiated reset via the Reset pushbutton (S3 RST on the MSP-EXP432P401R) [SLAU356A, 2015].

- **Class 1:** The Class 1 reset is designated as the Reboot Reset. This reset is initiated under software control and is similar in result to the POR reset [SLAU356A, 2015].
- **Class 2:** The Class 2 reset is designated as the Hard Reset. It is initiated as directed by a user-written application. It is useful to return the system to a known state in response to an application detected fault. It is important to note this type of reset does not reboot the processor [SLAU356A, 2015].
- **Class 3:** The Class 3 reset is designated as the Soft Reset. It is initiated as directed by the action of a user-written application. It is useful to return the system to a known state in response to an application detected fault. It is important to note this type of reset does not reboot the processor [SLAU356A, 2015].

7.4 INTERRUPTS

A microcontroller normally executes instructions in an orderly fetch-decode-execute sequence as dictated by a user-written program, as shown in Figure 7.1. However, the microcontroller must be equipped to handle unscheduled, yet planned, higher priority events that might occur inside or outside the microcontroller. To process such events, a microcontroller requires an interrupt system.

The interrupt system onboard a microcontroller allows it to respond to higher priority events. These events are planned, but we do not know when they will occur. When an interrupt event occurs, the microcontroller will normally complete the instruction it is currently executing, store key register values to capture the context of current events, and then transition program control to interrupt event specific tasks. These tasks, which resolve the interrupt event, are organized into a function called an interrupt service routine (ISR). Each interrupt will normally have its own interrupt specific ISR. Once the ISR is executed, the microcontroller restores the register values before the interrupt occurred, and resumes processing where it left off.

Applying the general concept of an interrupt, one can consider resets as interrupts with one exception. Resets do not return to the original task and instead resets the controller, whereas, after an interrupt service routine, a controller resumes execution of the task just before the interrupt was detected.

In most microcontrollers, including the MSP432, the starting address for each interrupt service routine, the special function to perform the service, is stored in a pre-designated location which the CPU recognizes. These locations are located in consecutive memory locations and are collectively designated interrupt vectors.

7.4.1 INTERRUPT HANDLING PROCESS

In this subsection, we describe the process of handling an interrupt event. Once a maskable interrupt is configured to be active, and an interrupt event occurs, a flag that corresponds to the particular interrupt event is asserted to indicate to the processor that there is an interrupt waiting

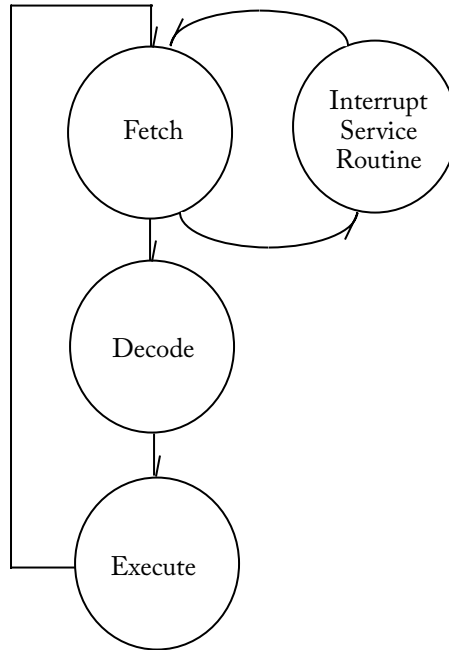


Figure 7.1: Microcontroller interrupt response.

to be serviced. The processor then takes the following actions in the order shown to provide an orderly transition from normal program operation to the interrupt service routine and back again.

1. Complete the current instruction.
2. Store the contents of the Program Counter onto the stack.
3. Store the contents of the Status Register onto the stack.
4. Choose the highest priority interrupt if multiple interrupts are pending.
5. Reset the interrupt request flag.
6. Clear the Status Register to prevent additional interrupts from occurring.
7. Load the contents of the interrupt vector onto the Program Counter.
8. Execute the specified interrupt service routine.
9. Once the service routine is finished, restore the Status Register and the Program Counter values from the stack.
10. Resume normal operation.

7.5 MSP432 INTERRUPT SYSTEM

The MSP432 hosting the Cortex M4F processor is equipped with a powerful and flexible exception system controlled by the Nested Vector Interrupt Controller (NVIC). The NVIC works closely with the processor to prioritize and respond to exception events including resets, non-maskable (NMI) interrupts, and user-programmable interrupts. A summary of exception types is provided in Figure 7.2. The resets have the highest priority, followed by NMI interrupts, and the user-programmable interrupts. Priorities among interrupts must be defined in advance to handle situations when more than one interrupt occurs simultaneously [SLAU356A, 2015].

Exception Type	Vector Number	Priority	Vector Address or Offset	Activation
-	0	-	0x0000_0000	Stack top is loaded from the first entry of the vector table on reset
Reset	1	-3 (highest)	0x0000_0004	Asynchronous
Non-Maskable interrupt (NMI)	2	-2	0x0000_0008	Asynchronous
Hard Fault	3	-1	0x0000_000C	-
Memory Management	4	Programmable	0x0000_0010	Synchronous
Bus Fault	5	Programmable	0x0000_0014	Synchronous when precise and asynchronous when imprecise
Usage Fault	6	Programmable	0x0000_0018	Synchronous
-	7-10	-	-	Reserved
SVCall	11	Programmable	0x0000_002C	Synchronous
Debug Monitor	12	Programmable	0x0000_0030	Synchronous
-	13	-	-	Reserved
PendSV	14	Programmable	0x0000_0038	Asynchronous
SysTick	15	Programmable	0x0000_003C	Asynchronous
Interrupts	16 and above	Programmable	0x0000_0040 and above	Asynchronous

Figure 7.2: MSP432 Exceptions [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

When an exception occurs, the pre-defined actions associated with the exception takes place. For an interrupt, the ISR associated with the interrupt is executed. While the key register values are being stored on the stack, the starting address for the ISR (the vector address) is also fetched. Typically, it takes 12 clock cycles for the MSP432 controller before the interrupt processing starts once an interrupt is detected, and it takes 12 clock cycles to restore the Status Register and Program Counter values and resume executing normally after the interrupt service routine is completed. Figure 7.3 shows the exception stack frame after the interrupt event has occurred. The Pre-IRQ top of stack is shown at the top of the image. The frame illustrates the key register values stored on the stack for safe keeping during execution of the interrupt service routine [SLAU356A, 2015].

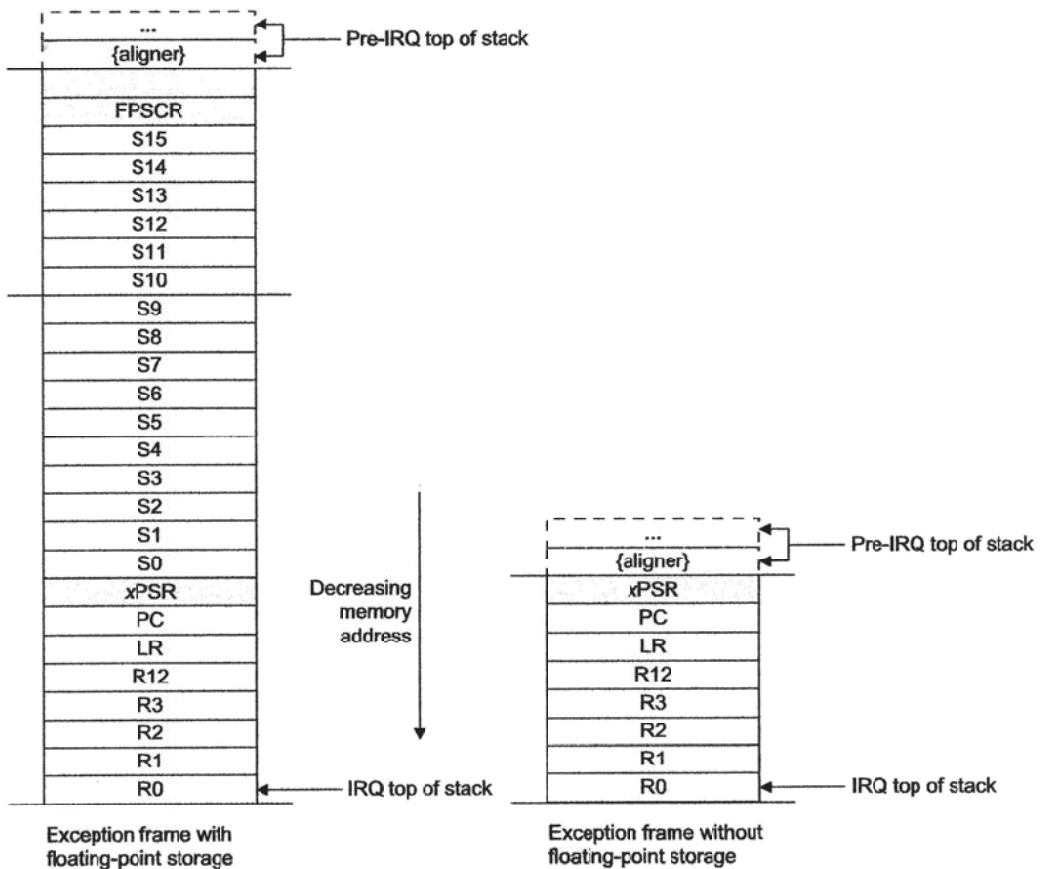


Figure 7.3: MSP432 stack frame [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

7.5.1 INTERRUPT SERVICE ROUTINE (ISR)

Most of the interrupt handling process described in this chapter takes place automatically (you, as a programmer, do not need to program them). In fact, for the resets, all processing is completed automatically. For maskable interrupts; however, your responsibility as a programmer is to:

1. turn on the global interrupt,
2. initialize the stack pointer,
3. configure the interrupt vector table (initialize the start address of your ISR),
4. enable the appropriate interrupt local enable bit, and
5. write the corresponding interrupt service routine.

The next several sections provide examples on configuring interrupts using Energia, the DriverLib library, and the C programming language.

7.6 ENERGIA INTERRUPT SUPPORT

Energia provides four functions to support interrupt operations including (www.energia.nu).

- **noInterrupts:** This function allows maskable interrupts to be suspended. This function is usually used to temporarily suspend interrupts during a portion of time-sensitive code. The interrupts are then re-enabled using the interrupts function.
- **interrupts:** This Energia function enables interrupts after they have been suspended by the noInterrupts function.
- **attachInterrupt:** The attachInterrupt function allows a pin driver interrupt event to be associated with a specific interrupt routine. The function requires three arguments: the interrupt pin, the name of the associated interrupt service routine, and the edge type to trigger the interrupt (RISING or FALLING).
- **detachInterrupt:** This function allows a specific interrupt to be suspended. The required argument is the corresponding interrupt pin used in the attachInterrupt function.

The following examples illustrate how to configure pin change interrupts using Energia functions.

Example 1: In this example the program normally executes instructions in function void loop(). When switch 2 (SW2) is pushed on the MSP432, an interrupt is asserted, the green LED changes state, and an interrupt counter is incremented. Printouts are provided on the serial monitor to indicate when the program is in the main program or in the interrupt.

344 7. RESETS AND INTERRUPTS

```

//*****
//Interrupt1
//Adapted from example provided at www.arduino.cc
//This example is in the public domain.
//*****

volatile int state = HIGH;
volatile int flag = HIGH;
int count = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(GREEN_LED, OUTPUT);
  digitalWrite(GREEN_LED, state);

  //Enable internal pullup.
  Without the pin will float
  //and the example will not work.
  pinMode(PUSH2, INPUT_PULLUP);

  // Interrupt is asserted when switch 2 is depressed
  attachInterrupt(PUSH2, blink, FALLING);
}

void loop()
{
  digitalWrite(GREEN_LED, state); //LED starts ON
  delay(1000);
  Serial.println("In the main program\n");
  if(flag)
  {
    count++;
    Serial.println(count);
    flag = LOW;
  }
}

//*****
```

```
//blink interrupt service routine
//*****

void blink()
{
state = !state;
flag = HIGH;
Serial.println("Inside interrupt 1\n\n");
}

//*****
```

Example 2: In this example the program normally executes instructions in function `void loop()`. When switch 1 (SW1) is pushed on the MSP432, an interrupt is asserted, the red LED changes state, and an interrupt counter is incremented. Similarly, when switch 2 (SW2) is pushed on the MSP432, an interrupt is asserted, the green LED changes state, and an interrupt counter is incremented. Printouts are provided on the serial monitor to indicate when the program is in the main program or in the interrupts.

```
//*****
//Interrupt2
//Adapted from example provided at www.arduino.cc
//This example is in the public domain.
//*****

volatile int state1 = HIGH;
volatile int flag1 = HIGH;
int count1 = 0;

volatile int state2 = HIGH;
volatile int flag2 = HIGH;
int count2 = 0;

void setup()
{
Serial.begin(9600);
pinMode(RED_LED, OUTPUT);
digitalWrite(RED_LED, state1);

pinMode(GREEN_LED, OUTPUT);
digitalWrite(GREEN_LED, state2);
```

```
//Enable internal pullup.  
Without the pin will float  
//and the example will not work.  
pinMode(PUSH1, INPUT_PULLUP);  
  
// Interrupt is asserted when switch 1 is depressed  
attachInterrupt(PUSH1, blink1, FALLING);  
  
//Enable internal pullup.  
Without the pin will float  
//and the example will not work.  
pinMode(PUSH2, INPUT_PULLUP);  
  
// Interrupt is asserted when switch 1 is depressed  
attachInterrupt(PUSH2, blink2, FALLING);  
}  
  
void loop()  
{  
digitalWrite(RED_LED, state1);           //LED starts ON  
digitalWrite(GREEN_LED, state2);        //LED starts ON  
delay(500);  
Serial.println("In the main program\n");  
  
if(flag1)  
{  
count1++;  
Serial.print("Count1:");  
Serial.println(count1);  
Serial.println("");  
flag1 = LOW;  
}  
  
if(flag2)  
{  
count2++;  
Serial.print("Count2:");  
Serial.println(count2);
```

```

    Serial.println("");
    flag2 = LOW;
  }
}

//*****
//blink1 interrupt service routine
//*****
void blink1()
{
    state1 = !state1;
    flag1 = HIGH;
    Serial.println("Inside interrupt 1\n\n\n");
}

//*****
//blink2 interrupt service routine
//*****

void blink2()
{
    state2 = !state2;
    flag2 = HIGH;
    Serial.println("Inside interrupt 2\n\n\n");
}

//*****

```

7.7 DRIVERLIB

The MSP432 interrupt system is well supported by several APIs including [[DriverLib, 2015](#)]:

- void Interrupt_disableInterrupt(uint32_t interruptNumber)
- bool Interrupt_disableMaster(void)
- void Interrupt_disableSleepOnIsrExit(void)
- void Interrupt_enableInterrupt(uint32_t interruptNumber)
- bool Interrupt_enableMaster(void)
- void Interrupt_enableSleepOnIsrExit(void)

- uint8_t Interrupt_getPriority(uint32_t interruptNumber)
- uint32_t Interrupt_getPriorityGrouping(void)
- uint8_t Interrupt_getPriorityMask(void)
- uint32_t Interrupt_getVectorTableAddress(void)
- bool Interrupt_isEnabled(uint32_t interruptNumber)
- void Interrupt_pendInterrupt(uint32_t interruptNumber)
- void Interrupt_registerInterrupt(uint32_t interruptNumber, void(*intHandler)(void))
- void Interrupt_setPriority(uint32_t interruptNumber, uint8_t priority)
- void Interrupt_setPriorityGrouping(uint32_t bits)
- void Interrupt_setPriorityMask(uint8_t priorityMask)
- void Interrupt_setVectorTableAddress(uint32_t addr)
- void Interrupt_unpendInterrupt(uint32_t interruptNumber)
- void Interrupt_unregisterInterrupt(uint32_t interruptNumber)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

7.8 PROGRAMMING INTERRUPTS IN C

Example 3: In this example the technique of polling is illustrated. Pin P6.7 is polled within a loop. If the pin is found to be high, pin P1.0 is set high, else it is set low.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
```

```
// notice, this list of conditions and the following disclaimer.
// - Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
// Neither the name of Texas Instruments Incorporated nor the names of
// its contributors may be used to endorse or promote products derived
// from this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
// COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
// BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
// OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
// ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
// TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
// USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
// DAMAGE.
// --COPYRIGHT--
// *****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
// MSP432 code examples are self-contained low-level programs that
// typically demonstrate a single peripheral function or device feature
// in a highly concise manner. For this the code may rely on the device's
// power-on default register values and settings such as the clock
// configuration and care must be taken when combining code from several
// examples to avoid potential side effects.
// Also see:
// http://www.ti.com/tool/mspdriverlib
// for an API functional library and:
// https://dev.ti.com/pinmux/
// for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
```

350 7. RESETS AND INTERRUPTS

```

//*****
//MSP432P401 Demo - Software Poll P6.7, Set P1.0 if P6.7 = 1
//
//Description: Poll P6.7 in a loop, if high P1.0 is set,
//             if low, P1.0 reset.
//
//ACLK = n/a, MCLK = SMCLK = default DCO
//
//             MSP432p401rpz
//             -----
//             /\|          XIN|-
//             | |          |
//             --|RST      XOUT|-
//             /\  |          |
//             --o--|P6.7   P1.0|-->LED
//             \|/
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;           //Stop watchdog timer

                                       //Configure GPIO
    P1DIR |= BIT0;                     //Set P1.0 to output direction
    P6DIR &= ~BIT7;                    //Set P6.7 to input direction

    while(1)                           //Test P6.7
    {
        if (P6IN & BIT7)
            P1OUT |= BIT0;             //if P6.7 set, set P1.0
        else
            P1OUT &= ~BIT0;            //else reset
    }
}

```

```

}
}

```

```

//*****

```

Example 4: In this example the processor is configured to enter the LPM3 mode. Pin P1.1 is configured as an input. When the button connected to P1.1 is depressed, the processor wakes up and executes the P1.1 associated interrupt. The ISR toggles the LED connected to P1.0.

As illustrated in this example, unused pins should be set to output for proper termination. An unused pin configured as an input may provide a path for noise into the processor.

A UML activity diagram for the example is provided in Figure 7.4.

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
//  notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
//  notice, this list of conditions and the following disclaimer in the
//  documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND

```

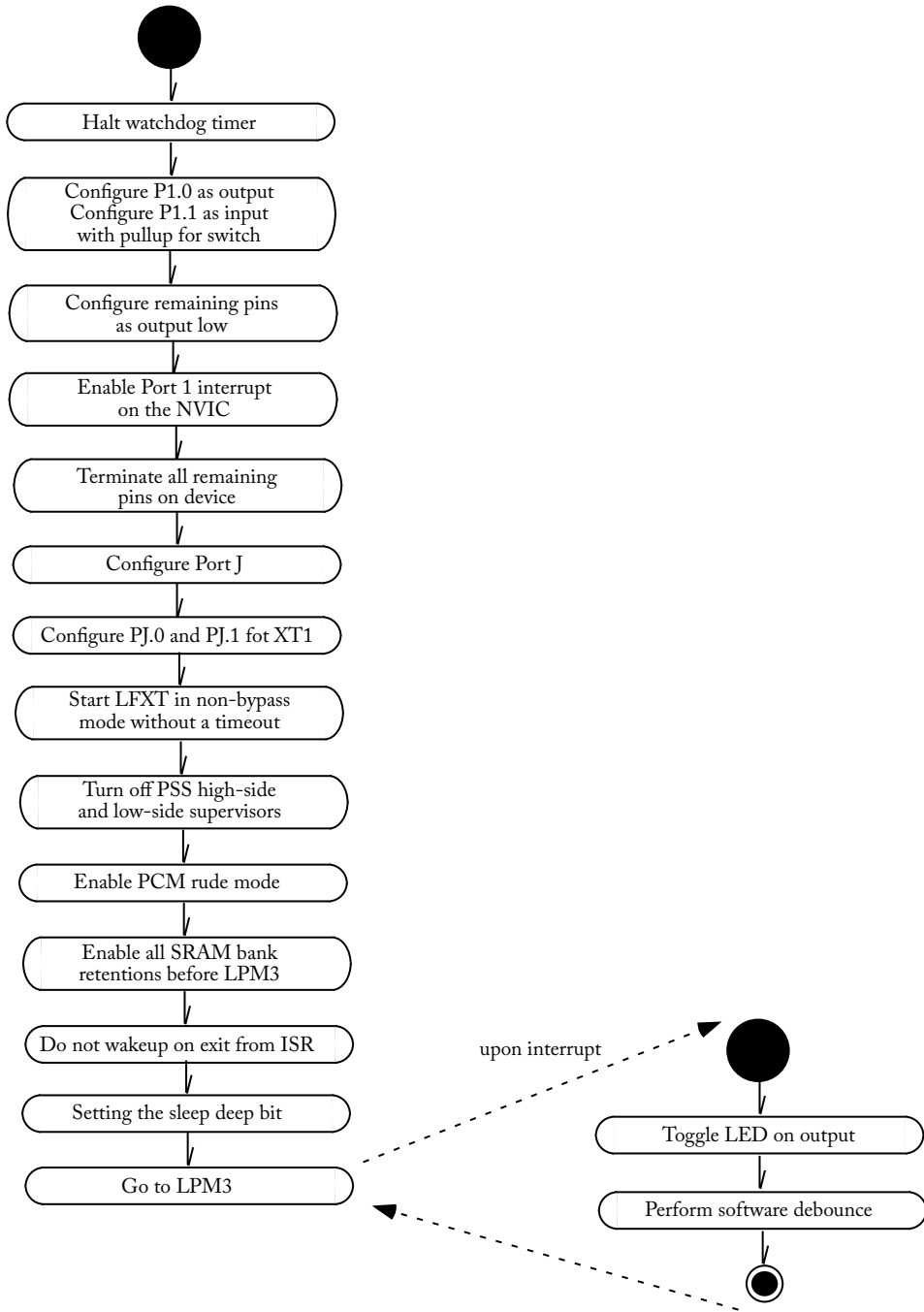


Figure 7.4: Interrupt example.

```

//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//          MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//*****
//MSP432P401 Demo - Software Port Interrupt Service on P1.1 from LPM3
//
//Description: MSP432 device is configured to enter LPM3 mode
//with GPIOs properly terminated.
P1.1 is configured as an input.
//Pressing the button connected to P1.1 results in device waking up and
//servicing the Port1 ISR. LPM3 current can be measured when P1.0 is
//output low (e.g., LED off).
//
// ACLK = 32kHz, MCLK = SMCLK = default DC0
//
//          MSP432p401rpz
//          -----
//          /|\|          |
//          | |          |
//          --|RST       |
//          /\ |          |
//          --o--|P1.1    P1.0|-->LED

```

354 7. RESETS AND INTERRUPTS

```
//    \|\n//\n//Dung Dang\n//Texas Instruments Inc.\n//Nov 2013\n//Built with Code Composer Studio V6.0\n//*****\n\n#include "msp.h"\nint main(void)\n{\n//Hold the watchdog\nWDTCTL = WDTPW | WDTHOLD;\n\n//Configuring P1.0 as output and P1.1 (switch) as input with pull-up\n//resistor. Rest of pins are configured as output low.\n//Notice intentional '=' assignment since all P1 pins are being\n//deliberately configured.\n\nP1DIR = ~(BIT1);\nP1OUT = BIT1;\nP1REN = BIT1;           //Enable pull-up resistor (P1.1 output high)\n\nP1SELO = 0;\nP1SEL1 = 0;\nP1IFG = 0;             //Clear all P1 interrupt flags\nP1IE = BIT1;          //Enable interrupt for P1.1\nP1IES = BIT1;         //Interrupt on high-to-low transition\n\n//Enable Port 1 interrupt on the NVIC\nNVIC_ISER1 = 1 << ((INT_PORT1 - 16) & 31);\n\n//Terminate all remaining pins on the device\nP2DIR |= 0xFF; P2OUT = 0;\nP3DIR |= 0xFF; P3OUT = 0;\nP4DIR |= 0xFF; P4OUT = 0;\nP5DIR |= 0xFF; P5OUT = 0;\nP6DIR |= 0xFF; P6OUT = 0;\nP7DIR |= 0xFF; P7OUT = 0;
```

```
P8DIR |= 0xFF; P8OUT = 0;
P9DIR |= 0xFF; P9OUT = 0;
P10DIR |= 0xFF; P10OUT = 0;

//Configure Port J
PJDIR |= (BIT2 | BIT3); PJOUT &= ~(BIT2 | BIT3);

//PJ.0 & PJ.1 configured for XT1
PJSELO |= BIT0 | BIT1;
PJSEL1 &= ~(BIT0 | BIT1);

//Starting LFXT in non-bypass mode without a timeout.
CSKEY = CSKEY_VAL;
CSCTL1 &= ~(SELA_M | SELB);
CSCTL1 |= SELA_LFXTCLK; //Source LFXTCLK to ACLK & BCLK
CSCTL2 &= ~(LFXTDRIVE_M); //Configure to lowest drive-strength
CSCTL2 |= LFXT_EN;
while (CSIFG & LFXTIFG)
    CSCLRIFG |= LFXTIFG;
CSKEY = 0;

//Turn off PSS high-side & low-side supervisors
PSSKEY = PSS_KEY_KEY_VAL;
PSSCTL0 |= SVSMHOFF | SVSLOFF;
PSSKEY = 0;

//Enable PCM rude mode, which allows device to enter LPM3 without
//waiting for peripherals
PCMCTL1 = PCM_CTL_KEY_VAL | FORCE_LPM_ENTRY;

//Enable all SRAM bank retentions prior to going to LPM3
SYSCTL_SRAM_BANKRET |= SYSCTL_SRAM_BANKRET_BNK7_RET;
__enable_interrupt();

//Do not wake up on exit from ISR
SCB_SCR |= SCB_SCR_SLEEPONEXIT;

//Setting the sleep deep bit
SCB_SCR |= (SCB_SCR_SLEEPDEEP);
```



```

//Go to LPM3
__sleep();
}

//*****
//Port1 ISR
//*****

void Port1Handler(void)
{
volatile uint32_t i, status;

//Toggling the output on the LED
if(P1IFG & BIT1)
    P1OUT ^= BIT0;

//Delay for switch debounce
for(i = 0; i < 10000; i++)

P1IFG &= ~BIT1;

}

//*****

```

7.9 LABORATORY EXERCISE: AUTONOMOUS ROBOT

In Chapter 2 we configured a Dagu Magician robot as an autonomous maze navigating robot. For this laboratory exercise add an additional IR sensor to the robot to detect “landmines.” A landmine in the maze is a paper strip placed across the robot’s path in the maze. When the robot detects the landmine, an interrupt service routine is executed to deactivate the landmine. The deactivation sequence should be some distinctive operation by the robot. For example, the robot could rotate about its axis several times, it could flash a special sequence of LEDs, it could make a distinctive sound, or better yet a combination of all three actions. If the robot rolls over the paper strip without executing the landmine deactivation sequence, the robot is considered destroyed and out of action. Have fun!

7.10 SUMMARY

For efficient use of a microcontroller, resets and interrupts offer the flexibility needed by a programmer. In this chapter we have described the MSP432 resets and their functions and explored the general concepts of interrupts. We also investigated the steps required to implement an interrupt service routine. We discussed the assigned priorities for resets and interrupts. A brief introduction was provided to the MSP432 Nested Vector Interrupt Controller (NVIC). Several interrupt examples were shown.

7.11 REFERENCES AND FURTHER READING

Arduino homepage, www.arduino.cc.

Energia homepage, www.energia.nu.

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015. 347, 348

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. 338, 339, 341, 342

Texas Instruments Code Composer Studio 6.1 for MSP432 (SLAU575B). Texas Instruments, 2015.

7.12 CHAPTER PROBLEMS

Fundamental

1. List three different types of resets in the MSP432 microcontroller.
2. State the purpose of resets and interrupts.
3. What is the main difference between a reset and an interrupt?
4. What are the differences between maskable and nonmaskable interrupts?
5. In addition to setting up a local interrupt enable bit, you must also set the global enable bit. Where is the global enable bit for all MSP432 maskable interrupts? Write an instruction to enable this global maskable interrupt enable bit.
6. What are the steps one must take to properly configure a maskable interrupt?
7. When more than one maskable interrupt occurs simultaneously, how does the MSP432 controller decide the order in which the controller service the interrupts?

Advanced

1. Why did the designers of MSP432 come up with three different types of resets?

2. Refer to the Interrupt Handling Process section and explain the purpose for each of the ten steps.
3. Write a segment of code to initialize the MSP432 microcontroller to operate in the power save LPM3.5 mode and only operate in the normal mode during an interrupt.

Challenging

1. It is challenging to handle nested interrupts. What might be some applications where nested interrupts are necessary?

Analog Peripherals

Objectives: After reading this chapter, the reader should be able to:

- describe the function of an analog-to-digital converter (ADC) and a digital-to-analog converter (DAC);
- explain the methods used to perform ADC conversions on the MSP432 microcontroller;
- configure the MSP432 microcontroller to accept analog signals and convert them into digital forms;
- use interrupts associated with the MSP432 microcontroller's ADC system;
- describe the operation and function of the MSP432 REF_A system;
- describe the operation and function of the MSP432 COMP_E system; and
- interface the MSP432 microcontroller with compatible Texas Instruments analog devices.

8.1 OVERVIEW

In this chapter, we discuss the subsystems that allow the microcontroller to accept and analyze external analog signals and to build and generate analog signals for other devices. Physical parameters of interest such as temperature, light intensity, sound, etc. are analog in nature. A physical parameter of interest must first be converted to an analog signal such as voltage by a transducer. The voltage is then conditioned such that it is compatible with the input requirements of the microcontroller's ADC system.

Consider a simple example of your voice signal. To process, store, and transmit a voice signal using a digital system, such as the MSP432 controller, a system must have a means to convert analog signals to their equivalent digital forms and also to convert back to a useful analog forms. The analog-to-digital converter and the digital-to-analog converter perform these required tasks. In Chapter 4, we presented the input and output interfaces of MSP432 microcontroller. In this chapter, we introduce the MSP432's analog-to-digital converter (ADC) system, the analog reference system (REF_A), and the analog comparator (COMP_E).

8.2 BACKGROUND

To convert an analog signal to a digital form, one must first capture the signal at a particular time (sample), find the analog signal value (quantize), and represent it (encode) before it is sent to a digital system. An analog-to-digital converter's job is to perform these three tasks. Depending on how fast we can sample and how many bits are used to represent an analog value, the quality and accuracy of representation of analog signals vary, which we study in this chapter.

Once a signal is processed, the processed signal may need to be sent back to the analog world. The digital-to-analog converter performs this task. A simple example is to amplify your voice using a microphone using a digital system. Your voice is first converted into its corresponding digital representation, the resulting signals are converted back to analog forms using a digital-to-analog converter, and its volume is amplified. We present the processes involved in both conversions and how those conversions are performed in the MSP432 microcontroller next.

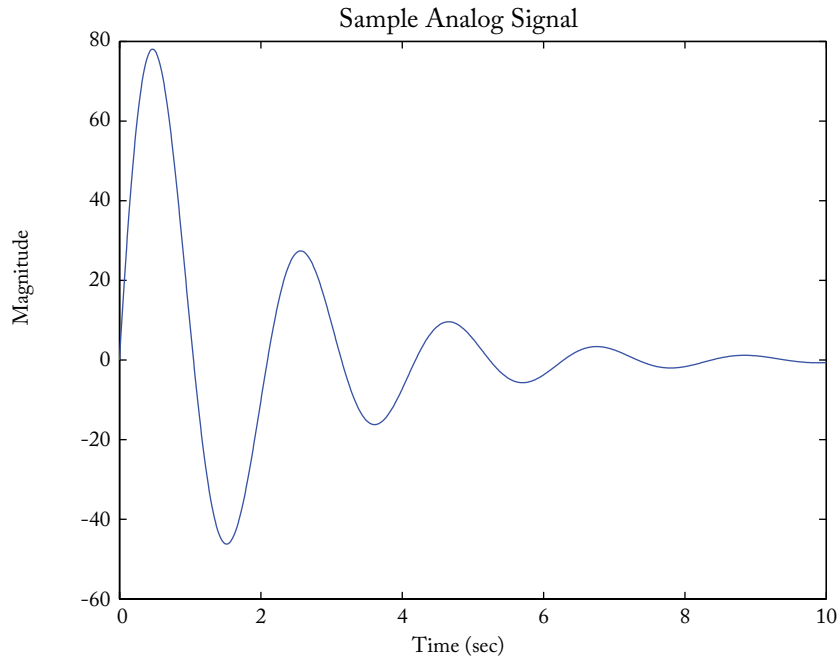


Figure 8.1: Sample analog signal.

8.3 ANALOG-TO-DIGITAL CONVERSION

Before a computer can process a physical signal, the signal must first be converted to a corresponding digital form. Figure 8.1 shows an example of an analog signal. Notice that for a given time,

t_x , the signal can hold any value along the y axis. So, how do analog signals such as the one shown in the figure get converted into digital signals? The analog-to-digital conversion process consists of the three separate sub-processes previously discussed: sampling, quantization, and encoding.

8.3.1 SAMPLING

The sampling process is the way for a digital system to capture an analog signal at a particular point in time. One can consider the sampling process as similar to taking snap shots of changing scenery using a camera.

Suppose we want to capture the movement of a baseball pitcher as he throws a ball toward home plate. Also let us assume the only means for you to capture the motion of the pitcher is with a camera. Suppose it takes two seconds for the pitcher to throw a baseball. If you take a picture at the start of the pitching and another one two seconds later, you have missed most of the action and will not be able to accurately represent the motion of the pitcher.

The inverse of the period between taking pictures in this example is the sampling frequency with the unit of Hertz (Hz). Since there is a 2 s interval between samples, the sampling rate is $1/2 = 0.5$ Hz. As you can imagine, the faster you take the pictures the more accurately you can re-create the pitcher's motion by sequencing photos.

The above example illustrates the primary issue of the sampling process, that of the sampling frequency. A correct sampling frequency depends on the characteristics of the analog signal of interest. If the analog signal changes quickly, the sampling frequency must be high. If the signal does not change rapidly, the sampling frequency can be slow and still capture the essence of the incoming signal.

You may wonder what harm does it cause to sample at the highest frequency possible regardless of the frequency content of the analog signal? The answer lies in optimizing resources. Just as it would be a waste to take multiple pictures of the same static object, it would not be a good use of resources to sample with a high frequency rate regardless of the nature of an analog signal. In the 1940's, Henry Nyquist, who worked at IBM Bell Laboratory, developed the concept that the minimum required sampling rate is a function of the highest input analog signal frequency: $f_s \geq 2 \times f_h$. The frequency f_s and f_h are the sampling frequency and the highest frequency of the signal we want to capture, respectively. That is, the sampling frequency must be greater than or equal to two times the highest frequency component of the input signal. Using the illustration of taking pictures again, the rate that you take a sequence of pictures must be at least two times as fast as the highest frequency of changes in the environment of which you are taking pictures to reconstruct the "signals" in the environment. We illustrate the Nyquist sampling rate using Figure 8.2.

Figure 8.2 shows the analog signal of interest, a sinusoidal signal. Frame (b) of the figure shows sampling points with a rate slower than the Nyquist rate and frame (c) shows the reconstruction of the original signals using only the sampled points shown in frame (b). Frame (d) shows the sampled points at the Nyquist rate and the corresponding reconstructed signal is shown

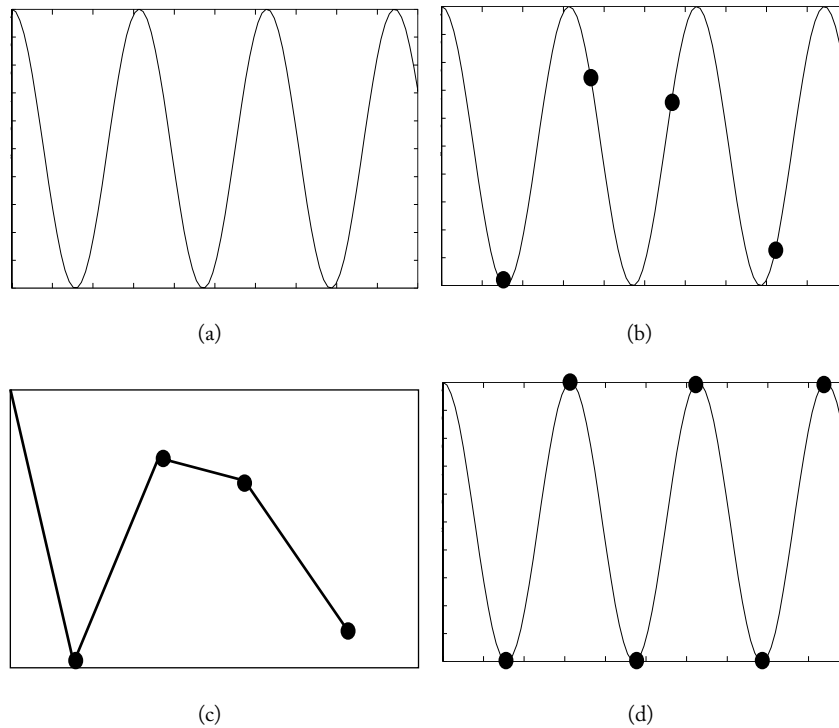


Figure 8.2: Sampling rate illustration. (*Continues.*)

in frame (e). Finally, frame (f) shows sampled points at a rate higher than the Nyquist sampling rate and frame (g) shows the reconstructed signal.

As can be seen from this simple example, when we sample an analog signal at the Nyquist sampling rate, we can barely generate the characteristics of the original analog signal. While at a higher sampling rate, we can retain the nature of an input analog signal more accurately at a higher cost, requiring a faster clock, additional processing power, and more storage facilities. Let us examine one final example before we move on to the quantization process.

Example 1: An average human voice contains frequencies ranging from about 200 Hz to 3.5 kHz. What should be the minimum sampling rate for an ATD converter?

Answer: According to the Nyquist sampling rate rule, we should sample at $3.5 \text{ kHz} \times 2 = 7 \text{ kHz}$, which translate to taking a sample every 142.9 μs . Your telephone company uses sampling rate of 8 kHz to sample your voice so that it can be delivered to a receiver.

It is important to note that once the upper frequency of interest is established, the sampled signal does not exceed this frequency. A low pass filter (LPF) is typically used to limit the fre-

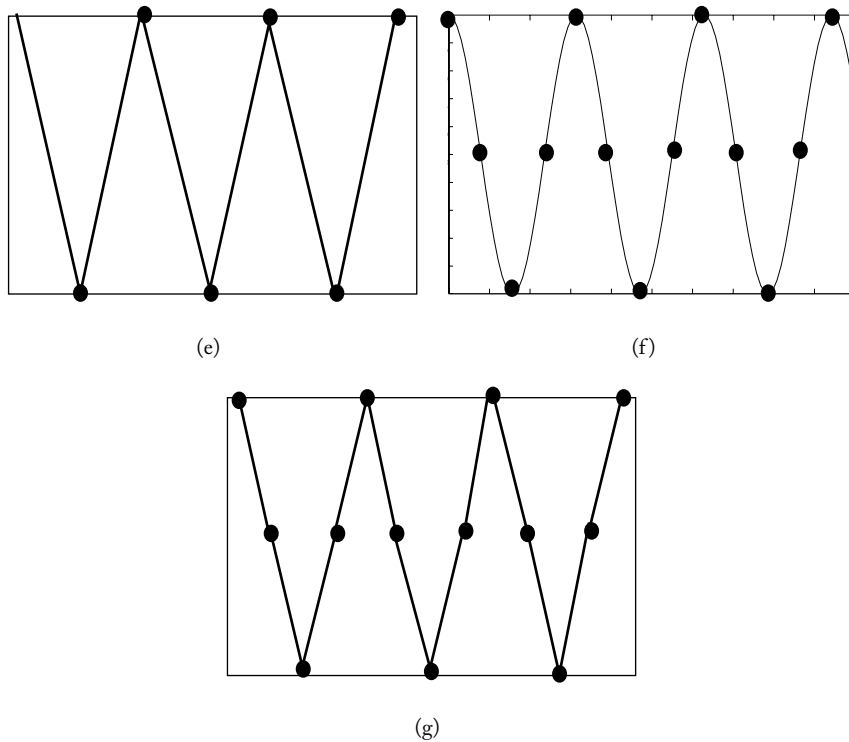


Figure 8.2: (Continued.) Sampling rate illustration.

quency of the sampled signal. The LPF prevents aliasing errors and is typically referred to as an anti-aliasing filter.

8.3.2 QUANTIZATION

Once a sample is captured, then the second step, quantization, can commence. Before we explain the process, we first need to define the term *quantization level*. Suppose we are working with an analog voltage signal whose values can change from 0–5 V. Now suppose we pick a point in time. The analog signal, at that point in time, can have any value between 0 V and 5 V, an infinite number of possibilities (think of real numbers). Since we do not have a means to represent infinitely different values in a digital system, we limit the possible values to a finite number. So, what should this number be? Previously, we saw that there are 2^b number of values we can represent with b bits. If we have a 2-bit analog-to-digital converter, 2-bits are used to represent the analog signal; there are four different representations we can choose from. In a 4-bit converter, we have 16 different ways to do so. If we have an 8-bit converter, we have 256 different representations, and so on.

As you may suspect, there is a tradeoff between using a large number of bits for an accurate representation vs. the hardware cost of designing and manufacturing a converter. A converter which employs more bits will yield more accurate representations of the sampled signal values. A decision made by an ADC converter designer determines the accuracy of a sampled data and the cost to manufacture the converter. The number of bits used to quantize a sampled value determines the available quantization levels. Therefore, a converter which uses 8-bits has 256 quantization levels while a converter that uses 10-bits has 1024 quantization levels.

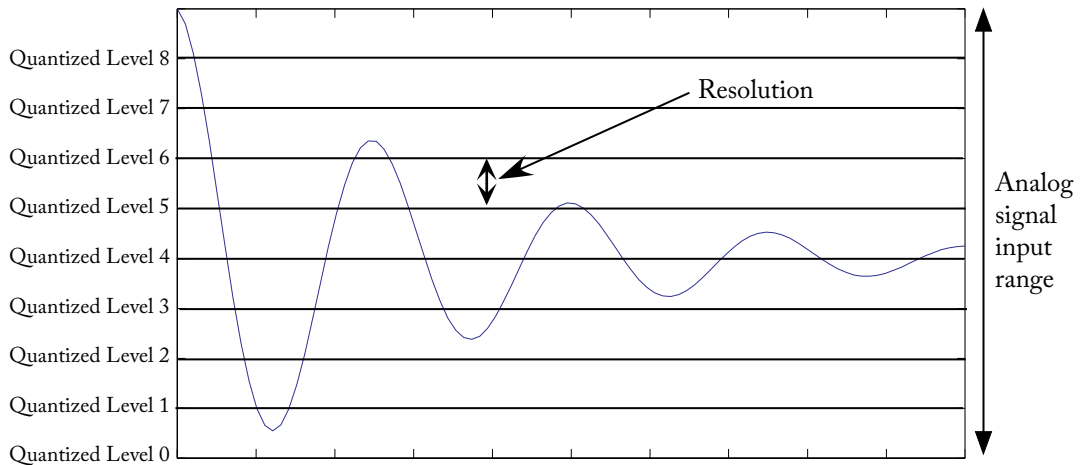


Figure 8.3: Quantization levels and resolution of an analog-to-digital converter.

We need to also define what is known as the resolution of an ADC converter. Simply put, the resolution is the smallest quantity a converter can represent or the “distance” between two adjacent quantization levels. The resolution will naturally vary depending on the range of input analog signal values. Suppose the input range is from 0–5 V, a typical input range of an ADC converter, and we have an 8-bit converter. The resolution δ is then

$$\delta = \frac{\text{analog input highest value} - \text{analog input lowest value}}{2^b} = \frac{5 - 0}{256} = 19.5312\text{mV}.$$

Figure 8.3 shows both quantization levels and the corresponding resolution for a sample input signal for an ADC converter.

Example 2: Given an application that requires an input signal range of 10 V and the resolution less than 5 mV, what is the minimum number of bits required for the ADC converter?

Answer: $\frac{10-0}{2^{10}} = 9.77 \text{ mV}$ and $\frac{10-0}{2^{11}} = 4.88 \text{ mV}$. Thus, the required number of bits for the ADC converter is 11-bits.

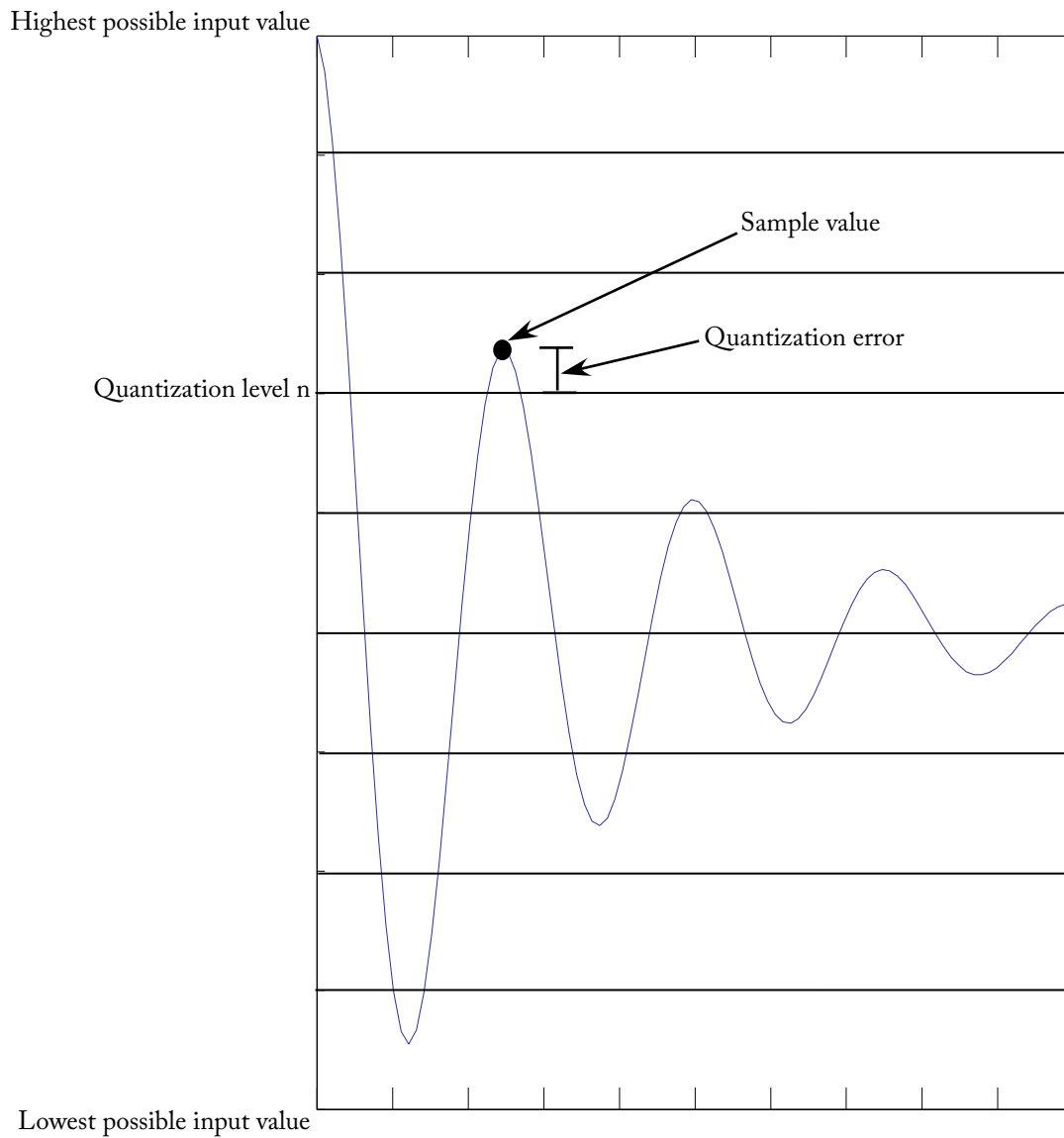


Figure 8.4: Quantized error of a sampled input signal.

We can now combine both sampling and quantization processes together to identify a quantized level of a sampled value. Suppose a sampled analog signal value is 3.4 V and the input signal range is 0–5 V. Using a converter with 8-bits, we can find the quantized level of the sampled value using the following equation:

$$\text{Quantized level} = \frac{\text{sampled input value} - \text{lowest possible input value}}{\delta}$$

Thus, given the input sample value of 3.4 V, the quantized level becomes $\frac{3.4 \text{ V} - 0 \text{ V}}{19.53 \text{ mV}} \cong 174.09$. Since we can only have integer levels, the quantized level becomes 174. So the sampling error is the difference between the true analog value and the sampled value. It is the amount of approximation the converter had to make. See Figure 8.4 for a pictorial view of this concept. For the example, the input sampled value 3.4 V is represented as the quantized level 174 and the quantized error is $0.09 \times \delta = 1.76 \text{ mV}$. Note that the maximum quantization error is the resolution of the converter.

Example 3: Given a sampled signal value of 7.21 V, using a 10 bit ADC converter with input range of 0 V and 10 V, find the corresponding quantization level and the associated quantization error.

Answer: First, we find the quantized level:

$$\text{Quantized level} = \frac{7.21 - 0}{\delta},$$

where $\delta = \frac{10}{2^{10}} = 9.77 \text{ mV}$. Thus, the quantized level is 738.3059. If the converter rounds down, the quantized level is 738 and the associated quantization error is $0.3059 \times 9.77 \text{ mV} \cong 2.987 \text{ mV}$.

8.3.3 ENCODING

The last step of the ADC conversion process is the encoding. The encoding process converts the quantized level of a sampled analog signal value into a binary number representation. Consider the following simple case, first. Suppose we have a converter with four bits. The available quantization levels for this converter are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. Using four bits, we can represent the quantization levels as 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111. Once we identify a quantization level, we can uniquely represent the quantization level as a binary number. This process is called encoding. Similar to a decimal number, the position of each bit in a binary number represents a different value. For example, binary number 1100 is decimal number $(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 12$. Knowing the weight of each bit, it is a straightforward process to represent a decimal number as a binary number.

Example 4: Find the encoded value of the quantization found in the previous example: 738. Recall we are using 10 bits.

Answer: Since we are using 10 bits to represent this number, the encoded value is $(1 \times 2^9) + (0 \times 2^8) + (1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 738$.

Thus, the encoded value is 1011100010.

8.4 DIGITAL-TO-ANALOG CONVERTER

The opposite function of an ADC converter is performed by a Digital-to-Analog (DAC) converter. The input to a DAC converter is an encoded value which specifies the desired output analog value. Similar to the ADC converter just discussed, a DAC converter must have both minimum and maximum reference analog voltages. The job of a DAC converter is then to map a minimum digital representation to its corresponding minimum analog value, a maximum digital representation to the maximum reference analog value, and representations in between minimum and maximum digital values to their appropriate analog counter parts. The most common method used to perform a DAC conversion is to pre-designate the analog weight of each bit in the digital input representation and then sum up the contributions to form an analog output. For example, suppose the range of output values for a DAC converter is from 0–5 V. If we have a four bit DAC converter, from the most to the least significant bits, each specific bit would be weighted 2.5 V, 1.25 V, 0.625 V, and 0.3125 V. Thus, a digital input of 1010 to this converter will result in $2.5 + 0.625 = 3.125$ V, and a digital input of 1111 to the DAC converter would result in $2.5 + 1.25 + 0.625 + 0.3125 = 4.6875$ V. Given an N-bit converter, it is straightforward to develop the following equation to describe the relationships among the input, number of bits used, and the output:

$$\text{Analog output} = \frac{\text{digital input}}{2^N} V_{refmax},$$

where N stands for the number of bits used in the converter and V_{refmax} represents the maximum analog reference voltage of the converter. Figure 8.5 illustrates a four bit DAC converter with the maximum and minimum output values of 2.5 V and 0 V, respectively.

8.5 MSP432 ANALOG-TO-DIGITAL CONVERTER

In this section we introduce the MSP432 ADC referred to as ADC14. We describe its features, operation, registers, and conclude with several programming examples.

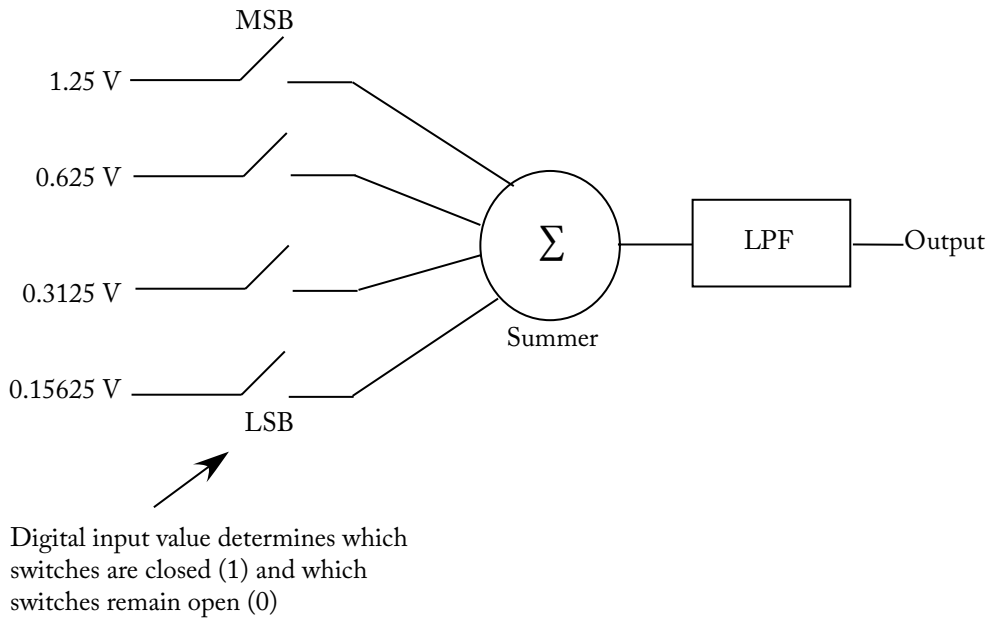


Figure 8.5: A sample 4-bit digital to analog converter. A digital input governs the positions of the switches, which determine whether or not their corresponding voltage values with respect to the reference voltage output values should contribute to the converter output value. The output of the converter is typically connected to a low pass filter before being used to remove any sharp edges resulting from switching from discrete voltage outputs of the converter.

8.5.1 FEATURES

The ADC14 system is a flexible and powerful analog-to-digital conversion system with an extensive list of features [SLAU356A].

- ADC14 provides 14-bits of ADC resolution. Recall, the resolution of a converter provides 2^b number of incremental steps between the high and low reference voltage.
- ADC14 is a successive approximation register (SAR) type converter. A SAR converter takes the same amount of time for converting an unknown voltage regardless of its magnitude. We discuss the operation of a SAR type converter in the next section.
- The maximum conversion rate of ADC14 is 1 Mega sample per second (Msps).
- ADC14 is equipped with 32 individual input channels. The inputs may be configured for single-ended conversion where the input signal is referenced to ground. The inputs may also be configured for differential input. In this type of conversion, two signals are subtracted

from one another and their difference is converted. This is especially useful in noisy environments. Signals that are common to both inputs (noise) are cancelled and the actual signal is amplified.

- Selected, specific internal signals within the MSP432 processor may be selected for ADC conversion.
- The ADC14 may be set to provide conversion on a single channel, multiple conversions of a single channel, a single conversion of a sequence of channels, or multiple conversions of a sequence of channels.
- ADC14 is supported by a variety of interrupts.

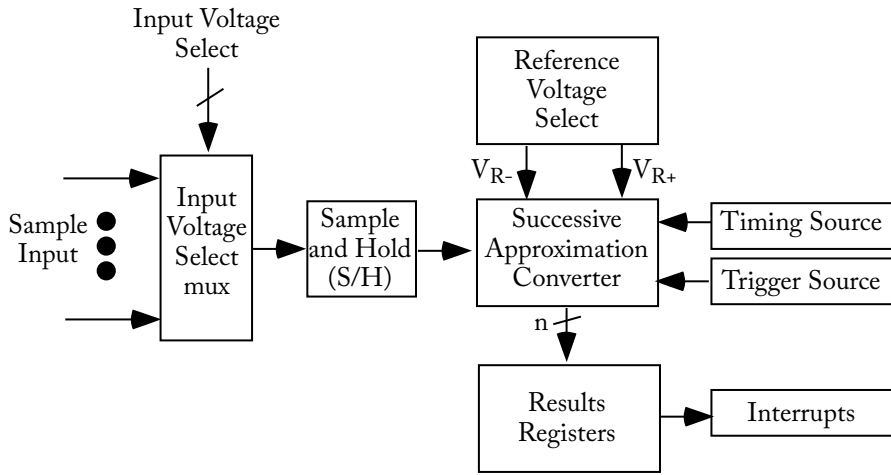
8.5.2 OPERATION

A basic block diagram of ADC14 is shown in Figure 8.6a. An input analog channel is selected for conversion by the input voltage select multiplexer (mux). The selected signal is held constant by the sample and hold (S/H) circuitry during the conversion process. The stable signal is then fed to the SAR converter. The SAR converter receives input from the reference voltage select, the timing source, and trigger source for conversion. The digital result of the conversion, provided as n bits, is stored in result registers. Specific interrupts may be selected to signal different significant events in the ADC process.

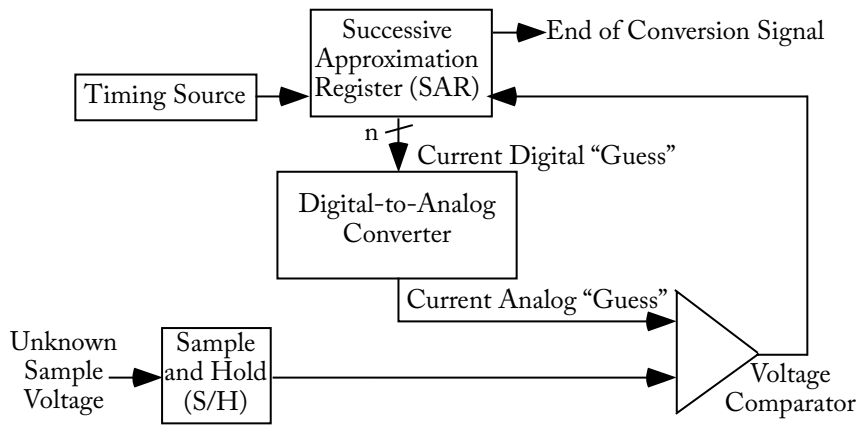
A block diagram of SAR converter operation is provided in Figure 8.6b. As its name implies, the SAR converter will make successive guesses at the unknown sample voltage value. It begins with a guess of one-half of the reference voltage. This digital guess is converted to a corresponding analog value by the digital-to-analog converter (DAC). The analog guess is compared to the unknown sample voltage by the voltage comparator. The output from the comparator prompts the SAR to guess higher or lower. This process continues n times (one for each bit in the SAR register). The guess progresses from one-half of the reference voltage to one-fourth to one-eighth, etc. When the conversion is complete, the end of conversion signal goes logic high.

The detailed block diagram of ADC14 is provided in Figure 8.7. It may appear a bit overwhelming at first; however, it is simply a more detailed version of the basic block diagram already provided. The operation of the ADC14 is configured and controlled by registers ADC14CTL0 and ADC14CTL1. The bit designators from these registers are shown at various points on the diagram.

As seen in the figure, an input analog channel is selected for conversion by the input voltage select multiplexer by the ADC14INCH x bits. The selected signal is held constant by the sample and hold (S/H) circuitry during the conversion process. The stable signal is then fed to the SAR converter. The SAR converter receives input from the reference voltage select, the timing source, and trigger source for conversion. The specific reference voltage is selected by the ADC14VRSEL bits. The specific timing source (MODCLK, SYSCLK, ACLK, MCLK, SMCLK, or HSM-



(a) successive approximation ADC block diagram.



(b) successive approximation register ADC converter.

Figure 8.6: Basic ADC14 block diagram.

CLK) is selected by the ADC14SSELx bits. The selected clock source may be further divided by the ADC14PDIV and the ADC14DIVx bit settings. The overall result is the ADC14CLK signal. The trigger source to initiate the ADC conversion is the SAMPCON signal. The specific trigger source is selected by the ADC14SHSx bits. The digital result of the conversion provided as n bits is stored in the ADC14MEM0 result registers. Specific interrupts may be selected to signal different significant events in the ADC process [SLAU356A, 2015].

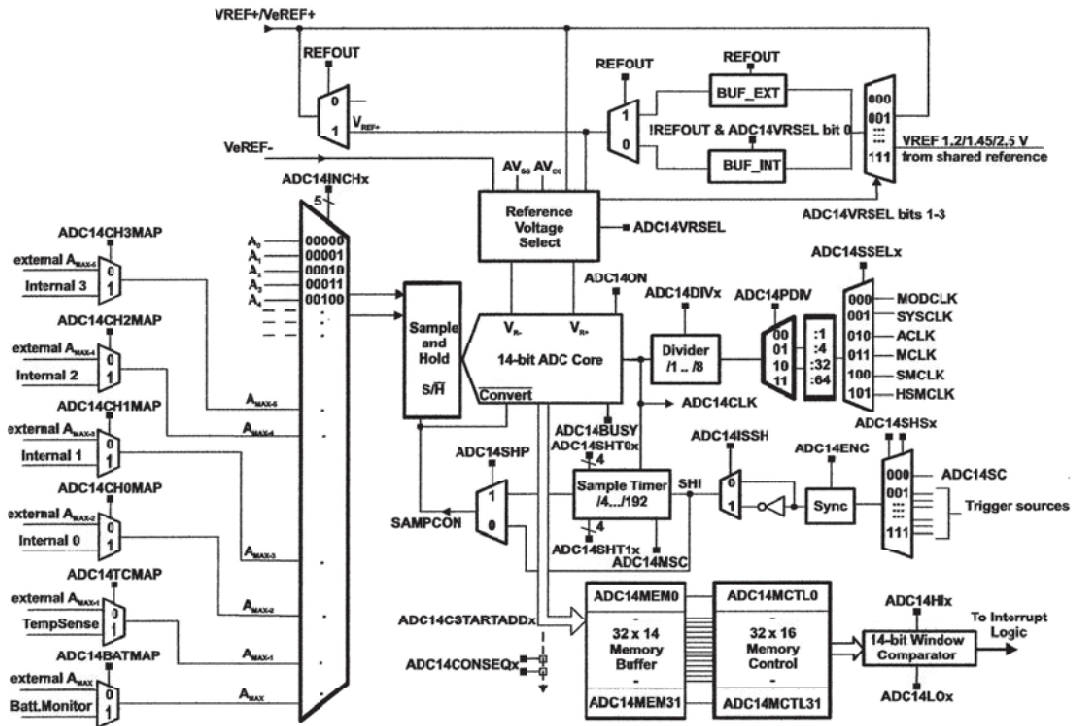


Figure 8.7: ADC14 block diagram. Illustration used with permission of Texas Instruments www.ti.com.

8.5.3 ADC REGISTERS

The operation of the ADC14 is configured and controlled by registers ADC14CTL0 and ADC14CTL1, as shown in Figure 8.8. Details concerning register settings are provided in SLAU356A the *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015].

8.5.4 ANALYSIS OF RESULTS

ADC14 provides a digital representation of the analog sample in a binary unsigned format. The values range from 0000_h to $3fff_h$. If the sampled signal is below the low reference voltage, ADC14 reports 0000_h ; whereas, if the sampled signal exceeds the high reference, ADC14 reports $3fff_h$. For analog sensed values between the low and high reference voltage, ADC14 reports a value of N_{ADC} when configured for single-ended operation [SLAU356A, 2015]:

$$N_{ADC} = 2^{14} \times ((V_{IN+} - V_{R+}) / (V_{R+} - V_{R-})).$$

31	30	29	28	27	26	25	24
ADC14PDIV		ADC14SHSx			ADC14SHP	ADC14SSH	ADC14DIVx
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
23	22	21	20	19	18	17	16
ADC14DIVx		ADC14SSELx			ADC14CONSEQx		ADC14BUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0
15	14	13	12	11	10	9	8
ADC14SHT1x				ADC14SHT0x			
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
ADC14MSC	Reserved		ADC14ON	Reserved		ADC14ENC	ADC14SC
rw-0	r-0	r-0	rw-0	r-0	r-0	rw-0	rw-0
Can be modified only when ADC14ENC = 0							
31	30	29	28	27	26	25	24
Reserved				ADC14CH3MAP	ADC14CH2MAP	ADC14CH1MAP	ADC14CH0MAP
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0
23	22	21	20	19	18	17	16
ADC14TCMAP	ADC14BATMAP	Reserved	ADC14STARTADDx				
rw-0	rw-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0
15	14	13	12	11	10	9	8
Reserved							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
Reserved		ADC14RES		ADC14DF	ADC14REFBURST	ADC14PWRMD	
r-0	r-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0
Can be modified only when ADC14ENC = 0							

Figure 8.8: ADC14CTL0 and ADC14CTL1 ADC14 registers [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

If configured for differential mode, ADC14 reports a value of:

$$N_{ADC} = 2^{14} \times ((V_{IN+} - V_{IN-}) / (V_{R+} - V_{R-})) + 8192.$$

8.6 PROGRAMMING THE MSP432 ADC14 SYSTEM

The ADC14 system may be programmed using Energia, APIs contained in DriverLib, and via register settings in C. We discuss each in turn.

8.6.1 ENERGIA PROGRAMMING

The Energia library contains several functions to support analog conversions including ADC- and DAC-related functions (www.energia.nu).

- **analogRead():** The analogRead function performs an ADC conversion on the indicated analog pin. The MSP-EXP432P401R LaunchPad analog pins are provided in the MSP432 BoosterPack standard interface diagram in Figure 1.6. The measured voltage is converted to an integer value between 0 and 1023 where 0 corresponds to 0 VDC and 1023 corresponds to 3.3 VDC.
- **analogReference():** The analogReference function provides for changing the high level reference voltage for ADC conversion. The different settings include:
 - **DEFAULT:** sets ADC high reference level to VCC 3.3 V.
 - **INTERNAL1V5:** sets ADC high reference level to internal 1.5 VDC reference.
 - **INTERNAL2V5:** sets ADC high reference level to internal 2.5 VDC reference.
 - **EXTERNAL:** sets ADC high reference level to the VREF pin value.
- **map:** As its name implies the map function maps a range of integers (fromLow, fromHigh) to a new range of integers (toLow, toHigh).
- **analogWrite:** The analogWrite function generates a pseudo analog output signal using a pulse width modulated signal. The analogWrite function generates a 490 Hz signal on the specified pin with a duty cycle specified from 0–255.

Example 5: The Educational BoosterPack MKII is equipped with a number of analog sensors. The joystick is an analog input sensor that provides analog x and y voltages indicating the position of the joystick. In this example, the x position of the joystick is used to determine the intensity of the onboard green LED. If the joystick is pushed straight down, the red LED is illuminated. Note how different analog related functions are employed in the example.

```
//*****
//Joystick example for Educational BoosterPack MK II
//http://boosterpackdepot.info/wiki/index.php?title=
//    Educational_BoosterPa//ck_MK_II
//
//Move the joystick around in x & y axes (pin 2 & 26) to
//adjust/mix the Green LED.
//Press straight down on the joystick center button to turn
//on the Red LED.
//
//The circuit:
// * Joystick X attached to pin 2
// * Joystick Y attached to pin 26
// * Joystick Sel attached to pin 5
```

374 8. ANALOG PERIPHERALS

```
// * Blue LED (analog) attached to pin 37
// * Green LED (analog) attached to pin 38
// * Red LED (digital) attached to pin 39
//
//Dec 03 2013 for Educational BoosterPack MK II
//by Dung Dang
//
//This example code is in the public domain.
//
//*****

const int joystickSel = 5;    //joystick select pin
const int joystickX = 2;     //joystick X-axis analog
const int joystickY = 26;    //joystick Y-axis analog

const int ledBlue = 37;     //LED pin number
const int ledGreen = 38;    //LED pin number
const int ledRed = 39;      //LED pin number
int joystickSelState = 0;    //joystick sel status
int joystickXState, joystickYState; //joystick position

void setup()
{
  //By default MSP432 has analogRead() set to 10 bits.
  //This Sketch assumes 12 bits.
  Uncomment to line below
  //to set analogRead() to 12 bit resolution for MSP432.
  //analogReadResolution(12);

  //initialize the LED pins as output:
  pinMode(ledRed, OUTPUT);

  //initialize the pushbutton pin as an input:
  pinMode(joystickSel, INPUT_PULLUP);
}

void loop()
{
  //read the analog value of joystick x axis
```

```
joystickXState = analogRead(joystickX);

//scale the analog input range [0,4096]
//into the analog write range [0,255]
joystickXState = map(joystickXState, 0, 4096, 0, 255);

//output to the led
analogWrite(ledGreen, joystickXState);

//read the analog value of joystick y axis
joystickYState = analogRead(joystickY);

//scale the analog input range [0,4096]
//into the analog write range [0,255]
joystickYState = map(joystickYState, 0, 4096, 0, 255);

//output to the led
analogWrite(ledBlue, joystickYState);

// read the state of the joystick select button value:
joystickSelState = digitalRead(joystickSel);

//check if the pushbutton is pressed.
//if it is, the buttonState is HIGH:
if(joystickSelState == LOW)
  {
  //turn LED on:
  digitalWrite(ledRed, HIGH);
  }
else
  {
  //turn LED off:
  digitalWrite(ledRed, LOW);
  }
}
//*****
```

8.6.2 MSP432 DRIVER LIBRARY

The MSP432 Peripheral Driver Library has many application programming interface (API) functions supporting the ADC14 system. API details are provided in the MSP432 Peripheral Driver Library User's Guide. An API function list is provided below.

- void ADC14_clearInterruptFlag(uint_fast64_t mask)
- bool ADC14_configureConversionMemory(uint32_t memorySelect, uint32_t refSelect, uint32_t channelSelect, bool differentialMode)
- bool ADC14_configureMultiSequenceMode(uint32_t memoryStart, uint32_t memoryEnd, bool repeatMode)
- bool ADC14_configureSingleSampleMode(uint32_t memoryDestination, bool repeatMode)
- bool ADC14_disableComparatorWindow(uint32_t memorySelect)
- void ADC14_disableConversion(void)
- void ADC14_disableInterrupt(uint_fast64_t mask)
- bool ADC14_disableModule(void)
- bool ADC14_disableReferenceBurst(void)
- bool ADC14_disableSampleTimer(void)
- bool ADC14_enableComparatorWindow(uint32_t memorySelect, uint32_t windowSelect)
- bool ADC14_enableConversion(void)
- void ADC14_enableInterrupt(uint_fast64_t mask)
- void ADC14_enableModule(void)
- bool ADC14_enableReferenceBurst(void)
- bool ADC14_enableSampleTimer(uint32_t multiSampleConvert)
- uint_fast64_t ADC14_getEnabledInterruptStatus(void)
- uint_fast64_t ADC14_getInterruptStatus(void)
- void ADC14_getMultiSequenceResult(uint16_t *res)
- uint_fast32_t ADC14_getResolution(void)

- uint_fast16_t ADC14_getResult(uint32_t memorySelect)
- void ADC14_getResultArray(uint32_t memoryStart, uint32_t memoryEnd, uint16_t *res)
- bool ADC14_initModule(uint32_t clockSource, uint32_t clockPredivider, uint32_t clock-Divider, uint32_t internalChannelMask)
- bool ADC14_isBusy(void)
- void ADC14_registerInterrupt(void(*intHandler)(void))
- bool ADC14_setComparatorWindowValue(uint32_t window, int16_t low, int16_t high)
- bool ADC14_setPowerMode(uint32_t powerMode)
- void ADC14_setResolution(uint32_t resolution)
- bool ADC14_setResultFormat(uint32_t resultFormat)
- bool ADC14_setSampleHoldTime(uint32_t firstPulseWidth, uint32_t secondPulseWidth)
- bool ADC14_setSampleHoldTrigger(uint32_t source, bool invertSignal)
- bool ADC14_toggleConversionTrigger(void)
- void ADC14_unregisterInterrupt(void)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 6: The example is included within MSPWare and demonstrates how DriverLib APIs may be used to sense the internal temperature sensor in both Celsius and Fahrenheit. A UML activity diagram for this example is provided in Figure 8.9.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD,BSD
//Copyright (c) 2014, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
```

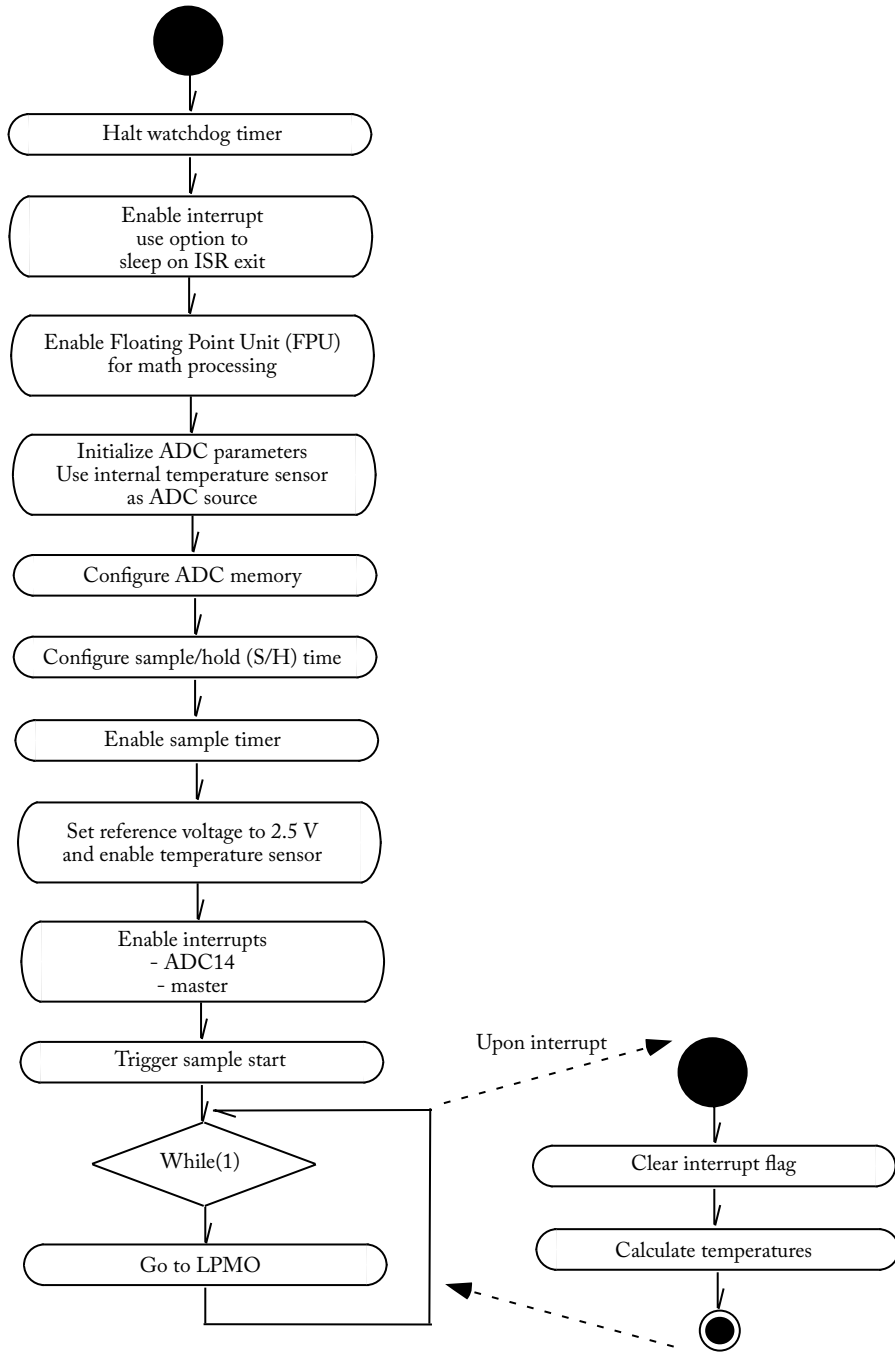


Figure 8.9: ADC14 example using DriverLib APIs.

```

// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//MSP432 ADC14 - Single Channel Repeat Temperature Sensor
//
//Description: This example shows the use of the internal temperature
//sensor. A simple continuous ADC sample/conversion is set up with a
//software trigger. The sample time is set as speced by the User's
//Guide.
All calculations take place in the ISR which take advantage of
//the Stacking Mode of the FPU. The temperature is calculated in both
//Celsius and Fahrenheit.
//
//          MSP432P401
//          -----
//          /|\|          |
//          | |          |
//          --|RST          P5.5 |
//          |          |

```



```

//          |          |
//          |          |
//
//Author: Timothy Logan
//*****

//DriverLib Includes
#include "driverlib.h"

//Standard Includes
#include <stdint.h>
#include <string.h>

volatile float tempC;
volatile float tempF;

int main(void)
{
//Halting WDT
WDT_A_holdTimer();
Interrupt_enableSleepOnIsrExit();

//Enabling the FPU with stacking enabled (for use within ISR)
FPU_enableModule();
FPU_enableLazyStacking();

//Initializing ADC (MCLK/1/1) with temperature sensor routed
ADC14_enableModule();
ADC14_initModule(ADC_CLOCKSOURCE_MCLK, ADC_PREDIVIDER_1,
                 ADC_DIVIDER_1, ADC_TEMPSENSEMAP);

//Configuring ADC Memory (ADC_MEM0 A22 (Temperature Sensor)
//in repeat mode).
ADC14_configureSingleSampleMode(ADC_MEM0, true);
ADC14_configureConversionMemory(ADC_MEM0, ADC_VREFPOS_AVCC_VREFNEG_VSS,
                                ADC_INPUT_A22, false);

//Configuring the sample/hold time
ADC14_setSampleHoldTime(ADC_PULSE_WIDTH_192, ADC_PULSE_WIDTH_192);

```

```

//Enabling sample timer in auto iteration mode and interrupts
ADC14_enableSampleTimer(ADC_AUTOMATIC_ITERATION);
ADC14_enableInterrupt(ADC_INT0);

//Setting reference voltage to 2.5 and enabling temperature sensor
REF_A_setReferenceVoltage(REF_A_VREF2_5V);
REF_A_enableReferenceVoltage();
REF_A_enableTempSensor();

//Enabling Interrupts
Interrupt_enableInterrupt(INT_ADC14);
Interrupt_enableMaster();

//Triggering the start of the sample
ADC14_enableConversion();
ADC14_toggleConversionTrigger();

//Going to sleep
while(1)
{
    PCM_gotoLPM0();
}

//*****
//This interrupt happens every time a conversion has completed.
Since
//the FPU is enabled in stacking mode, we are able to use the FPU safely
//to perform efficient floating point arithmetic.
//*****

void adc_isr(void)
{
    uint64_t status;
    uint32_t cal30, cal85;

    status = ADC14_getEnabledInterruptStatus();
    ADC14_clearInterruptFlag(status);

```

```

if(status & ADC_INT0)
{
    cal30 = SysCtl_getTempCalibrationConstant(SYSCTL_2_5V_REF,
        SYSCTL_30_DEGREES_C);
    cal85 = SysCtl_getTempCalibrationConstant(SYSCTL_2_5V_REF,
        SYSCTL_85_DEGREES_C);

    tempC =(float)((((uint32_t) ADC14_getResult(ADC_MEM0)-cal30)*(85 - 30))
        /(cal85 - cal30) + 30.0f;
    tempF = tempC * 9.0f / 5.0f + 32.0f;
}
}
//*****

```

8.6.3 PROGRAMMING ADC14 IN C

In this section, a representative example is provided of programming the ADC14 system using C. In this example the repeat sequence-of-channels mode is used on channels A0, A1, A2, and A3 with results stored to ADC14MEM0, ADC14MEM1, ADC14MEM2, and ADC14MEM3. After the completion of each sequence, the four conversion results are moved to A0results[], A1results[], A2results[], and A3results[]. A UML activity diagram for this example is provided in Figure 8.10.

Example 7: This example performs a repeated sequence of conversions using repeat sequence-of-channels mode.

```

//*****
//--COPYRIGHT-- ,BSD, EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of

```

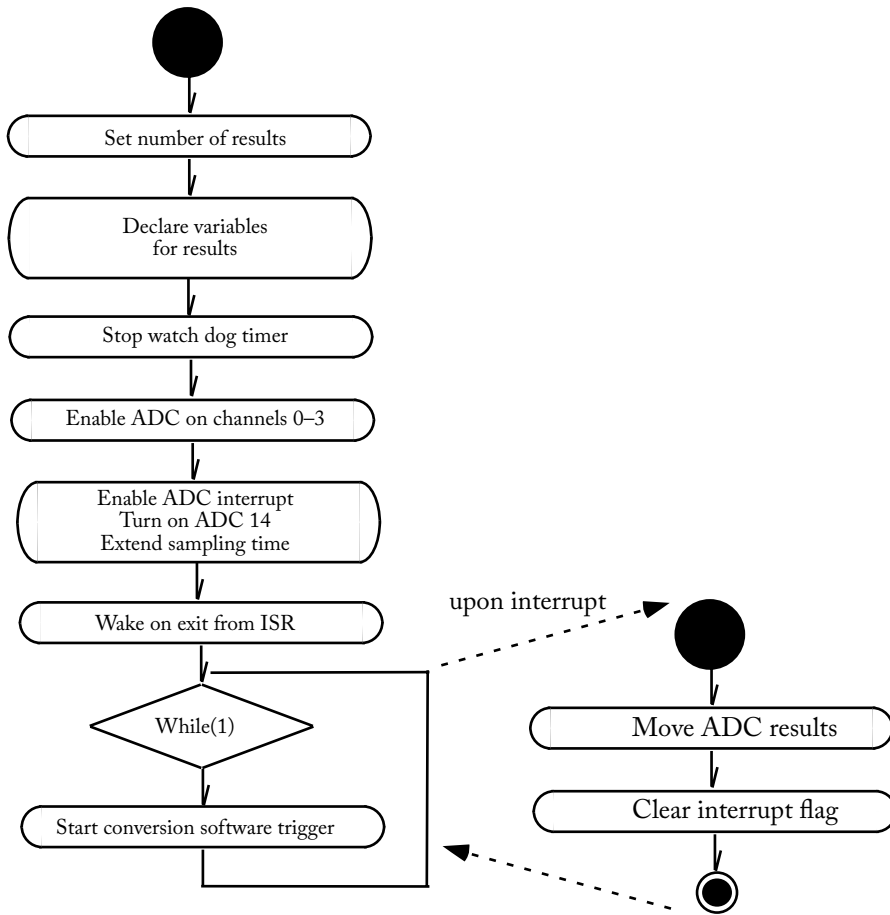


Figure 8.10: ADC14 example using C.

```

//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND

```

384 8. ANALOG PERIPHERALS

```
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
//
//*****
//
//
//          MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
//  http://www.ti.com/tool/mspdriverlib for an API functional library
//  https://dev.ti.com/pinmux/ for a GUI approach to peripheral
//          configuration.
//
//--COPYRIGHT--
//*****
//MSP432P401 Demo - ADC14, Repeated Sequence of Conversions
//
//Description: This example shows how to perform a repeated sequence of
//conversions using "repeat sequence-of-channels" mode.
AVcc is used
//for the reference and repeated sequence of conversions is performed on
//Channels A0, A1, A2, and A3. Each conversion result is stored in
//ADC14MEM0, ADC14MEM1, ADC14MEM2, and ADC14MEM3 respectively.
After each
//sequence, the 4 conversion results are moved to A0results[],
//A1results[], A2results[], and A3results[].
//
//Test by applying voltages to channels A0 - A3. Open a watch window in
//debugger and view the results.
Set Breakpoint1 in the index increment
//line to see the array values change sequentially and Breakpoint2 to see
//the entire array of conversion results in A0results[], A1results[],
```

```

//A2results[], and A3results[] for the specified Num_of_Results.
//
//Note that a sequence has no restrictions on which channels are
//converted.
For example, a valid sequence could be A0, A3, A2, A4, A2,
//A1, A0, and A7.
//
//          MSP432P401
//          -----
//          /\| |
//          | | |
//          --|RST
//          | |
//  Vin0 -->|P5.5/A0
//  Vin1 -->|P5.4/A1
//  Vin2 -->|P5.3/A2
//  Vin3 -->|P5.2/A3
//          | |
//
//Wei Zhao
//Texas Instruments Inc.
//June 2014
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include <stdint.h>

#define    Num_of_Results    8

volatile uint16_t A0results[Num_of_Results];
volatile uint16_t A1results[Num_of_Results];
volatile uint16_t A2results[Num_of_Results];
volatile uint16_t A3results[Num_of_Results];
static uint8_t index;

int main(void)
{
WDTCTL = WDTPW+WDTHOLD;                //Stop watchdog timer

```

```

//Configure GPIO
P5SEL1 |= BIT5 | BIT4 | BIT3 |BIT2;           //Enable A/D channel A0-A3
P5SEL0 |= BIT5 | BIT4 | BIT3 |BIT2;

__enable_interrupt();
NVIC_ISER0 = 1 << ((INT_ADC14 - 16) & 31); //Enable ADC interrupt
                                           //in NVIC module
                                           //Turn on ADC14,
                                           //extend sampling time
ADC14CTL0 = ADC14ON | ADC14MSC | ADC14SHTO__192 | ADC14SHP |
           ADC14CONSEQ_3;

//to avoid overflow of results
ADC14MCTL0 = ADC14INCH_0;                     //ref+=AVcc, channel = A0
ADC14MCTL1 = ADC14INCH_1;                     //ref+=AVcc, channel = A1
ADC14MCTL2 = ADC14INCH_2;                     //ref+=AVcc, channel = A2
ADC14MCTL3 = ADC14INCH_3+ADC14EOS;            //ref+=AVcc, channel = A3,
                                           //end seq.
ADC14IER0 = ADC14IE3;                         //Enable ADC14IFG.3

SCB_SCR &= ~SCB_SCR_SLEEPONEXIT;             //Wake up on exit from ISR

while(1)
{
    ADC14CTL0 |= ADC14ENC | ADC14SC;           //Start conv-software trigger
    __sleep();
    __no_operation();                         //For debugger
}

}

//*****
// ADC14 interrupt service routine
//*****

void ADC14IsrHandler(void)
{
    if(ADC14IFGR0 & ADC14IFG3)

```

```

{
A0results[index] = ADC14MEM0;          //Move A0 results, IFG is cleared
A1results[index] = ADC14MEM1;          //Move A1 results, IFG is cleared
A2results[index] = ADC14MEM2;          //Move A2 results, IFG is cleared
A3results[index] = ADC14MEM3;          //Move A3 results, IFG is cleared
index = (index + 1) & 0x7;             //Increment results index, modulo
__no_operation();                       //Set Breakpoint1 here
}
}

//*****

```

8.7 VOLTAGE REFERENCE

The MSP432 is equipped with an internal voltage reference system called REF_A. It provides reference voltages from peripheral systems throughout the MSP432 but also may be routed to an external pin. The three available reference voltages are 1.2 V, 1.45 V, or 2.5 V [DriverLib, 2015, SLAU356A, 2015].

The REF_A operation is controlled by the settings of REF Control Register 0 (REFCTL0) shown in Figure 8.11.

15	14	13	12	11	10	9	8
Reserved		REFBGRDY	REFGENRDY	BGMODE	REFGENBUSY	REFBGACT	REFGENACT
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
REFBGOT	REFGENOT	REFVSEL		REFTCOFF	Reserved	REFOUT	REFON
rw-0	rw-0	rw-0	rw-0	rw-1	rw-0	rw-0	rw-0

Can be modified only when REFGENBUSY = 0

Figure 8.11: REF_A Control Register 0 [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

The DriverLib contains the following APIs to support use of REF_A [DriverLib, 2015]:

- void REF_A_disableReferenceVoltage (void)
- void REF_A_disableReferenceVoltageOutput (void)
- void REF_A_disableTempSensor (void)
- void REF_A_enableReferenceVoltage (void)
- void REF_A_enableReferenceVoltageOutput (void)

- void REF_A_enableTempSensor (void)
- uint_fast8_t REF_A_getBandgapMode (void)
- bool REF_A_getBufferedBandgapVoltageStatus (void)
- bool REF_A_getVariableReferenceVoltageStatus (void)
- bool REF_A_isBandgapActive (void)
- bool REF_A_isRefGenActive (void)
- bool REF_A_isRefGenBusy (void)
- void REF_A_setBufferedBandgapVoltageOneTimeTrigger (void)
- void REF_A_setReferenceVoltage (uint_fast8_t referenceVoltageSelect)
- void REF_A_setReferenceVoltageOneTimeTrigger (void)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 8: Provided below is an example using the DriverLib APIs to enable REF_A module with a 2.5 VDC reference [DriverLib, 2015].

```
//*****
//Setting reference voltage to 2.5 and enabling reference
MAP_REF_A_setReferenceVoltage(REF_A_VREF2_5V);
MAP_REF_A_enableReferenceVoltage();
```

```
//*****
```

Example 9: In this example, the REF_A voltage level is routed to pin P5.6. The program cycles through the available voltage levels.

```
//*****
//--COPYRIGHT--,BSD, EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
```

```

// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// //*****
//
//                                     MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib for an API functional library
// https://dev.ti.com/pinmux/ for a GUI approach to peripheral
// configuration.
//
//--COPYRIGHT--
//*****
//MSP432P401 Demo - Output reference module voltage to a port pin

```

```

//
//Description: Configure and enable the reference module.
Output the
//reference voltage to a port pin.
Cycle through the available voltage
//levels, which can be observed on the oscilloscope/meter at port
//pin P5.6.
//
//
//          MSP432p401rpz
//          -----
//          /|\|          |
//          | |          |
//          --|RST       |
//          |          P5.6|-->VREF
//          |          P1.0|-->LED
//
//  Dung Dang
//  Texas Instruments Inc.
//  Nov 2013
//  Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include "stdint.h"

int main(void)
{
volatile uint32_t i;
WDTCTL = WDTPW | WDTHOLD;           //Stop WDT
P1DIR |= BIT0;                      //P1.0 set as output

//Configure P5.6 to its analog function to output VREF
P5SELO |= BIT6;
P5SEL1 |= BIT6;

REFCTL0 |= REFON;                   //Turn on reference module
REFCTL0 |= REFOUT;                  //Output ref voltage to a pin

```

```

while (1)
{
//Output VREF = 1.2V
REFCTL0 &= ~(REFVSEL_3);           //Clear existing VREF voltage
                                   //level setting
REFCTL0 |= REFVSEL_0;             //Set VREF = 1.2V
while (REFCTL0 & REFGENBUSY);     //Wait until the reference
                                   //generation is settled

for (i = 50000; i > 0; i--);
P1OUT ^= BIT0;                   //Toggle P1.0 LED indicator

//Output VREF = 1.45V
REFCTL0 &= ~(REFVSEL_3);           //Clear existing VREF voltage
                                   //level setting
REFCTL0 |= REFVSEL_1;             //Set VREF = 1.45V
while (REFCTL0 & REFGENBUSY);     //Wait until the reference
                                   //generation is settled

for (i = 50000; i > 0; i--);
P1OUT ^= BIT0;                   //Toggle P1.0 LED indicator

//Output VREF = 2.0V
REFCTL0 &= ~(REFVSEL_3);           //Clear existing VREF voltage
                                   //level setting
REFCTL0 |= REFVSEL_1;             //Set VREF = 1,45V
while (REFCTL0 & REFGENBUSY);     //Wait until the reference
                                   //generation is settled

for (i = 50000; i > 0; i--);
P1OUT ^= BIT0;                   //Toggle P1.0 LED indicator

//Output VREF = 2.5V
REFCTL0 |= REFVSEL_3;             //Set VREF = 2.5V
while (REFCTL0 & REFGENBUSY);     //Wait until the reference
                                   //generation is settled

for (i = 50000; i > 0; i--);
P1OUT ^= BIT0;                   //Toggle P1.0 LED indicator
}
}
//*****

```

8.8 COMPARATOR

The MSP432 is equipped with a comparator system called COMP_E. COMP_E provides two channels of analog comparators. As its name implies, a comparator compares an analog signal with a known reference. If the analog signal is greater than the reference, the output of the comparator is logic high. On the other hand, if the analog signal is less than the reference, the analog output is low.

The COMP_E system is controlled by a complement of registers. Register settings are provided in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015].

- CExCTL0 Comparator control register 0
- Comparator control register 1
- Comparator control register 2
- Comparator control register 3
- Comparator interrupt register
- Comparator interrupt vector

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

COMP_E is supported by a series of DriverLib APIs provided below.

- typedef struct _COMP_E_Config COMP_E_Config
- void COMP_E_clearInterruptFlag (uint32_t comparator, uint_fast16_t mask)
- void COMP_E_disableInputBuffer (uint32_t comparator, uint_fast16_t inputPort)
- void COMP_E_disableInterrupt (uint32_t comparator, uint_fast16_t mask)
- void COMP_E_disableModule (uint32_t comparator)
- void COMP_E_enableInputBuffer (uint32_t comparator, uint_fast16_t inputPort)
- void COMP_E_enableInterrupt (uint32_t comparator, uint_fast16_t mask)
- void COMP_E_enableModule (uint32_t comparator)
- uint_fast16_t COMP_E_getEnabledInterruptStatus (uint32_t comparator)
- uint_fast16_t COMP_E_getInterruptStatus (uint32_t comparator)
- bool COMP_E_initModule (uint32_t comparator, const COMP_E_Config *config)

- uint8_t COMP_E_outputValue (uint32_t comparator)
- void COMP_E_registerInterrupt (uint32_t comparator, void(*intHandler)(void))
- void COMP_E_setInterruptEdgeDirection (uint32_t comparator, uint_fast8_t edgeDirection)
- void COMP_E_setPowerMode (uint32_t comparator, uint_fast16_t powerMode)
- void COMP_E_setReferenceAccuracy (uint32_t comparator, uint_fast16_t referenceAccuracy)
- void COMP_E_setReferenceVoltage (uint32_t comparator, uint_fast16_t supplyVoltageReferenceBase, uint_fast16_t lowerLimitSupplyVoltageFractionOf32, uint_fast16_t upperLimitSupplyVoltageFractionOf32)
- void COMP_E_shortInputs (uint32_t comparator)
- void COMP_E_swapIO (uint32_t comparator)
- void COMP_E_toggleInterruptEdgeDirection (uint32_t comparator)
- void COMP_E_unregisterInterrupt (uint32_t comparator)
- void COMP_E_unshortInputs (uint32_t comparator)

Details of specific APIs are contained in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Example 11: Provided below is an example of programming the COMP_E system in C. In the example an input voltage “Vcompare” is compared to a 2.0 VDC reference. If the input is greater than 2.0 VDC, C0OUT is logic high, otherwise, it is logic low.

```
//*****
//--COPYRIGHT--,BSD, EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
```

```

//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
//
//*****
//
//
//
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib for an API functional library
// https://dev.ti.com/pinmux/ for a GUI approach to peripheral
// configuration.
//
//--COPYRIGHT--
//*****
//MSP432P401 Demo - COMP output Toggle in LPM0 ; input channel C01;
//
//          Vcompare is compared against internal 2.0V reference
//
//Description: Use Comp and internal reference to determine if

```

```

//input'Vcompare' is high or low.
When Vcompare exceeds 2.0V COOUT goes
//high and when Vcompare is less than 2.0V then COOUT goes low.
//
//          MSP432P401RPZ
//          -----
//          /\|
//          | |
//          --|RST      P8.0/C01|<--Vcompare
//          |
//          |          P7.1/COOUT|----> 'high'(Vcompare>2.0V);
//          |          |          'low'(Vcompare<2.0V)
//          |
//
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include "stdint.h"

int main(void)
{
volatile uint32_t i;

WDTCTL = WDTPW | WDTHOLD;           //Stop WDT

//Configure COOUT port ping
P7DIR |= BIT1;                      //P3.5 output direction
P7SELO |= BIT1;                     //Select COOUT function on P7.1

//Setup Comparator_E
CEOCTL0 = CEIPEN | CEIPSEL_M;       //Enable V+, input channel CE1
CEOCTL1 = CEPWRMD_1;               //normal power mode
CEOCTL2 = CEREF2_2 | CERS_3 | CERSEL; //VREF is applied to -terminal
//R-ladder off; bandgap ref

```



```

CEOCTL3 = BIT1;
CEOCTL1 |= CEON;
for (i=0;i<75;i++);

__sleep();
__no_operation();
}

//*****

```

8.9 LABORATORY EXERCISE: EDUCATIONAL BOOSTERPACK MK II

In this laboratory exercise, we equip the MSP-EXP432P401R LaunchPad with the Educational BoosterPack Mk II and exercise several onboard analog sensors.

The Educational BoosterPack Mk II is equipped with a number of analog sensors. Complete the following tasks.

- Develop a table of analog sensors onboard the MkII including their function and connection pin to the MSP-EXP432P401R LaunchPad.
- Develop an application using as many of the sensors as possible.
- Provide a UML activity diagram, structure chart, and code for your application.
- Develop and execute a test plan to demonstrate the function of your application.

8.10 SUMMARY

In this chapter, we discussed the subsystems that allow the microcontroller to input and analyze analog signals. We provided an introduction to the MSP432's analog-to-digital converter (ADC) system, the analog reference system (REF_A), and the analog comparator (COMP_E).

8.11 REFERENCES AND FURTHER READING

Arduino homepage, www.arduino.cc.

Barrett, S. and Pack, D. 2012. *Atmel AVR Microcontroller Primer: Programming and Interfacing*, 2nd ed., San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/s00427ed1v01y201206dcs039](https://doi.org/10.2200/s00427ed1v01y201206dcs039).

Barrett, S. and Pack, D. 2006. *Microcontrollers Fundamentals for Engineers and Scientists*, Morgan & Claypool Publishers. DOI: [10.2200/s00025ed1v01y200605dcs001](https://doi.org/10.2200/s00025ed1v01y200605dcs001).

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015. [377](#), [387](#), [388](#), [393](#)

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. [370](#), [371](#), [372](#), [387](#), [392](#)

Texas Instruments BOOSTXL-EDUMKII Educational BoosterPack Mark II Plug-in Module SLAU599, 2015.

Texas Instruments Code Composer Studio6.1 for MSP432 (SLAU575B). Texas Instruments, 2015.

Texas Instruments Meet the MSP432P401R LaunchPad Development Kit (SLAU596). Texas Instruments, 2015.

Texas Instruments MSP432P401R LaunchPad Development Kit (MSP-EXP432P401R) (SLAU597A). Texas Instruments, 2015.

Texas Instruments MSP432P401x Mixed-Signal Microcontrollers (SLAS826A). Texas Instruments, 2015.

Texas Instruments MSP432x5xx/MSP432x6xx Family User's Guide (SLAU208G). Texas Instruments, 2010.

8.12 CHAPTER PROBLEMS

Fundamental

1. Using the Nyquist sampling rate, find the minimum sampling frequency of an ATD converter, if the highest frequency of an input analog signal is 2 kHz.
2. Given a sinusoidal input analog signal, $5 \cos(2\pi 10kt)$, and sampling frequency of 1 KHz, find the first three sampled values with starting time 0.
3. Given an 8 bit ATD converter and input range of 0 V and 3.3 V, what is the quantization level for sampled value of 2.9 V?
4. What is the quantization error for the sampled signal in Problem 3?
5. What is the encoded value of quantization level from Problem 3?

Advanced

1. Write a program segment using the ADC14 converter to (1) operate with 14 bit resolution, (2) use internal reference voltages of 2.5 V and 0 V, (3) continuously sample analog signals from pins A0 and A1, (4) use the unsigned binary format, (5) compare the input analog values, (6) turn the logic state on Pz.x pin high if the signal on A0 is higher than the one on A1; otherwise, turn the logic state low, and (7) turn the logic state on Pz.y pin high if the signal on A1 is higher than the one on A1; otherwise, turn the logic state low.

Challenging

1. Present your design and write a program to construct a smart home program that locates your position in room whose size is 10 ft wide, 10 ft long, and 9 ft high. Assume that you need to use infrared sensors to do the job. You can use as many sensors as you need but want to minimize the number used. Suppose the infrared sensor output is fed to an ADC14 converter of a MSP432 and you have means to communicate among MSP432s. Design the sensor positions and write a program to locate a person in the room.

Communication Systems

Objectives: After reading this chapter, the reader should be able to:

- describe the differences between serial and parallel communication methods;
- present the features of the MSP432 microcontroller's enhanced Universal Serial Communication Interface (eUSCI) systems A and B;
- illustrate the operation of the Universal Asynchronous Serial Receiver and Transmitter (UART) mode of the eUSCI;
- program the UART for basic transmission and reception;
- describe the operation of the Serial Peripheral Interface (SPI) mode of the eUSCI;
- configure a SPI-based system to extend the features of the MSP432 microcontroller;
- describe the purpose and function of the Inter-Integrated Communication (I²C) mode of the eUSCI; and
- program the I²C communication system.

9.1 OVERVIEW

Microcontrollers must often exchange data with other microcontrollers or peripheral devices. For such applications, data may be exchanged by using parallel or serial techniques. With parallel techniques, an entire byte (or a set of n bits) of data is typically sent simultaneously from a transmitting device to a receiving device or received at the same time from an external device. While this is efficient from a time point of view, it requires eight separate lines (or n separate lines) for the data transfer.

In serial transmission, however, data is sent or received single bit at a time. For a byte size data transmission, once eight bits have been received at the receiver, the data byte is reconstructed. While this is inefficient from a time point of view, it only requires a line (or two) to transmit and receive the data. Serial transmission techniques also help minimize the use of precious microcontroller input/output pins.

9.2 BACKGROUND

The MSP432 microcontroller is equipped with the enhanced Universal Serial Communication Interface (eUSCI). The system is equipped with a host of different serial communication subsystems, as shown in Figure 9.1. The eUSCI consists of two different communication subsystems: eUSCI A type modules and eUSCI B modules. Each microcontroller in the MSP432 line has a complement of A and B type eUSCI modules. Should a specific MSP432 microcontroller type have more than one of the A and/or B type modules, they are numbered sequentially starting with zero (e.g., e_USCI A0, A1, etc.) [SLAU356A, 2015].

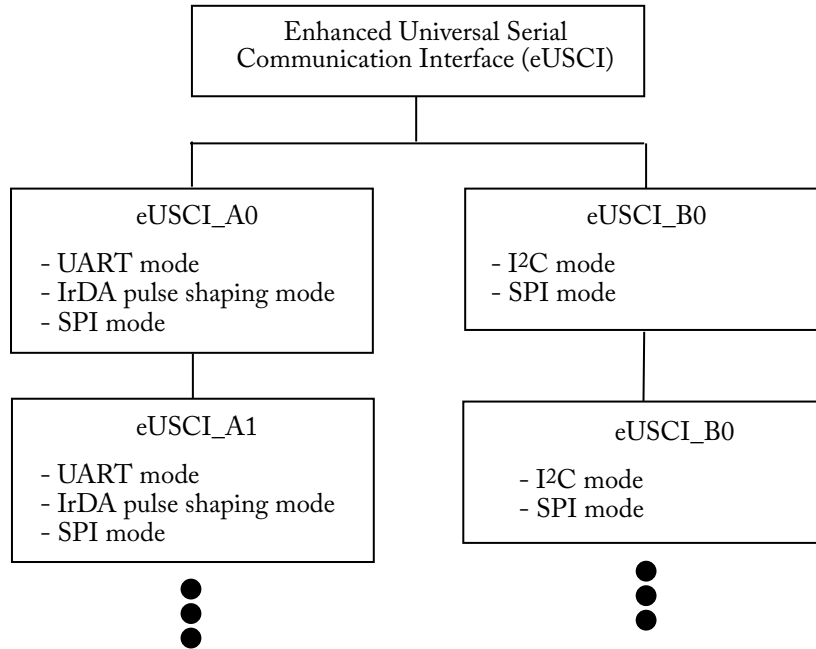


Figure 9.1: MSP432 enhanced Universal Serial Communication Interface (eUSCI).

As can be seen in the figure, eUSCI A modules provide support for [SLAU356A, 2015].

- **Universal Asynchronous Serial Receiver and Transmitter (UART).** The UART provides a serial data link between a transmitter and a receiver. The transmitter and receiver pair maintains synchronization using start and stop bits that are embedded in the data stream.
- **Infrared Data Association (IrDA).** The IrDA protocol provides for a short-range data link using an infrared (IR) link. It is a standardized protocol for IR optically linked devices. It is used in various communication devices, personal area networks, and instrumentation.

- **The Serial Peripheral Interface (SPI).** The SPI provides synchronous communications between a receiver and a transmitter. The SPI system maintains synchronization between the transmitter and receiver pair using a common clock provided by the master designated microcontroller. An SPI serial link has a much faster data rate than UART.

The eUSCI B modules also provide support for SPI communications and Inter-Integrated Communication (I²C) communications. The I²C is one of prominent communication modes used when multiple serial devices are interconnected using a serial bus. The I²C bus is a two-wire bus with the serial data line (SDL) and the serial clock line (SCL). By configuring devices connected to the common I²C line as either a master device or a slave device, multiple devices can share information using a common bus. The I²C system is used to link multiple peripheral devices to a microcontroller or several microcontrollers together in a system that are in close proximity to one another [SLAU356A, 2015].

Space does not permit an in-depth discussion of all communication features of the eUSCI system. We concentrate on the basic operation of the UART, SPI, and I²C systems. For each system, we provide a technical overview, a review of system registers, and code examples. We begin with a review of serial communication concepts.

9.3 SERIAL COMMUNICATION CONCEPTS

Before we delve into the serial communication technologies, we first review common serial communication terminology.

Asynchronous vs. synchronous serial transmission: In serial communications, the transmitting and receiving devices must agree on the “rules of engagement” by using a common data rate and protocol. This allows both the transmitter and receiver to properly coordinate data transmission/reception. There are two basic methods of maintaining coordination or “sync” between the transmitter and receiver: asynchronous and synchronous.

In an asynchronous serial communication system, such as the UART aboard the MSP432 microcontroller, framing bits are used at the beginning and end of a data byte. These framing bits alert the receiver that an incoming data byte has arrived and also signals the completion of the data byte reception. The data rate for an asynchronous serial system is typically much slower than the synchronous system, but it only requires a single wire between the transmitter and receiver for simplex (one way) communication.

A synchronous serial communication system maintains “sync” between the transmitter and receiver by employing a common clock between the two devices. Data bits are sent and received on the edge of the clock. This allows data transfer rates higher than with asynchronous techniques but requires two lines, data and clock, to connect a receiver and a transmitter for simplex communications.

Baud rate: Data transmission rates are typically specified as a Baud or bits per second rate. For example, 9600 Baud indicates the data is being transferred at 9600 bits per second.

Full Duplex: Often serial communication systems must both transmit and receive data simultaneously. To do so requires separate hardware for transmission and reception at each end of the communication link. A single duplex system has a single complement of hardware that must be switched from transmission to reception configuration. A full duplex serial communication system has separate hardware for transmission and reception.

Non-return to Zero (NRZ) Coding Format: There are many different coding standards used within serial communications. The important point is a transmitter and a receiver must use a common coding standard so data may be interpreted correctly at the receiving end. The MSP432 microcontroller uses a non-return to zero (NRZ) coding standard. In NRZ coding a logic one is signaled by a logic high during the entire time slot allocated for a single bit, whereas, a logic zero is signaled by a logic low during the entire time slot allocated for a single bit.

The RS-232 Communication Protocol: When serial transmission occurs over a long distance, additional techniques may be used to insure data integrity. Over long distances, logic levels degrade and may be corrupted by noise. When this happens at the receiving end, it is difficult to discern a logic high from a logic low. The RS-232 standard has been around for some time. With the RS-232 standard (EIA-232), a logic one is represented with a -12 VDC level while a logic zero is represented by a $+12$ VDC level. Chips are commonly available (e.g., MAX232) that convert the output levels from a microcontroller to RS-232 compatible levels and convert back to microcontroller compatible levels at the receiver. The RS-232 standard also specifies other features for this communication protocol such as connector type and pinout.

Parity: To further enhance data integrity during transmission, parity techniques may be used. A parity bit is an additional bit (or bits) that is transmitted with the data byte. With a single parity bit, a single bit error may be detected. Parity may use an even or odd parity bit. In even parity, the parity bit is set to one or zero such that the number of ones in the data byte including the parity bit is an even number. In odd parity, the parity bit is set to one or zero such that the number of ones in the data byte including the parity bit is odd. At the receiver, the number of bits within a data byte, including the parity bit, are counted to insure that parity has not changed, indicating an error did not occur during transmission. For single bit error correction or multiple bit error detection, additional parity bits are required.

ASCII: The American Standard Code for Information Interchange or ASCII is a standardized, seven bit method of encoding alphanumeric data. It has been in use for many decades, so some of the characters and actions listed in the ASCII table are not in common use today. However, ASCII is still the most common method of encoding alphanumeric data. The ASCII code is shown in Figure 9.2. For example, the capital letter “G” is encoded in ASCII as 0x47. The “0x” symbol indicates the hexadecimal number representation. Unicode is the international counterpart of ASCII. It provides standardized 16-bit encoding format for the written languages of the world. ASCII is a subset of Unicode. The interested reader is referred to the Unicode home page website at: www.unicode.org for additional information on this standardized encoding format.

		Most Significant Digit							
		0x0_	0x1_	0x2_	0x3_	0x4_	0x5_	0x6_	0x7_
Least significant digit	0x_0	NUL	DLE	SP	0	@	P	`	p
	0x_1	SOH	DC1	!	1	A	Q	a	q
	0x_2	STX	DC2	“	2	B	R	b	r
	0x_3	ETX	DC3	#	3	C	S	c	s
	0x_4	EOT	DC4	\$	4	D	T	d	t
	0x_5	ENQ	NAK	%	5	E	U	e	u
	0x_6	ACK	SYN	&	6	F	V	f	v
	0x_7	BEL	ETB	‘	7	G	W	g	w
	0x_8	BS	CAN	(8	H	X	h	x
	0x_9	HT	EM)	9	I	Y	i	y
	0x_A	LF	SUB	*	:	J	Z	j	z
	0x_B	VT	ESC	+	;	K	[k	{
	0x_C	FF	FS	‘	<	L	\	l	
	0x_D	CR	GS	-	=	M]	m	}
	0x_E	SO	RS	.	>	N	^	n	~
	0x_F	SI	US	/	?	O	_	o	DEL

Figure 9.2: ASCII Code. The ASCII code is used to encode alphanumeric characters. The “0x” indicates hexadecimal notation in the C programming language.

9.4 MSP432 UART

The UART system is located within the eUSCI module A. In this section, we discuss UART features, provide an overview of the UART hardware operation and character format, discuss how to set the UART Baud rate, provide an overview of UART related registers, and conclude with several examples.

9.4.1 UART FEATURES

The MSP432 microcontroller is equipped with a powerful and flexible UART system. To select the UART mode the Synchronous Mode Enable bit (UCSYNC bit) located in the eUSCI_Ax Control Register 0 (part of eUSCI_Ax Control Word 0) must be cleared to 0. This places the system in the asynchronous mode when, in this mode, serial data is transmitted outside of the

microcontroller via the UCAxTXD pin and received via the UCAxRXD pin. (Note: The “x” designates which USCI A module is employed (e.g., 0, 1, 2)) [SLAU356A, 2015].

The UART system provides a number of features that allow the MSP432 to communicate with a wide variety of peripheral devices or another microcontroller. These features include [SLAU356A, 2015]:

- support for serial transmission protocols including the capability to transmit 7- or 8-bit data with odd, even, or no parity;
- independent transmit and receive shift registers equipped with separate transmit and receive buffer registers;
- the capability to send or receive data the least significant bit (LSB) first or the most significant bit (MSB) on both the transmit and receive channels. This feature allows the MSP432 microcontroller to match the protocol of an existing peripheral device;
- the capability to operate within a multiprocessor system using the built-in, idle-line, and address-bit communication protocols;
- auto wake-up feature from a low power mode (LPMx) when a start edge is received;
- extensive flexibility in setting programmable baud rates;
- a number of system status flags for error detection, error suppression, and address detection; and
- interrupts for the data receive and transmit.

In the next section, we examine how these features are incorporated into the UART hardware.

9.4.2 UART OVERVIEW

Provided in Figure 9.3 is a block diagram of the eUSCI_Ax module configured for UART mode (UCSYNC bit = 0). The UART module can be subdivided into the Baudrate Generator (center of Figure 9.3), the receiver related hardware (top of figure), and the transmit hardware (lower portion of figure). We discuss each in turn.

The eUSCI_Ax module communicates asynchronously with another device (e.g., peripheral) when the UCSYNC mode is set to zero. As previously mentioned, in an asynchronous mode, the transmitter and receiver maintain synchronization with one another, using start and stop bits to frame each data byte sent. It is essential that both transmitter and receiver are configured with the same Baud rate, number of start and stop bits, and the type of parity employed (odd, even or none.)

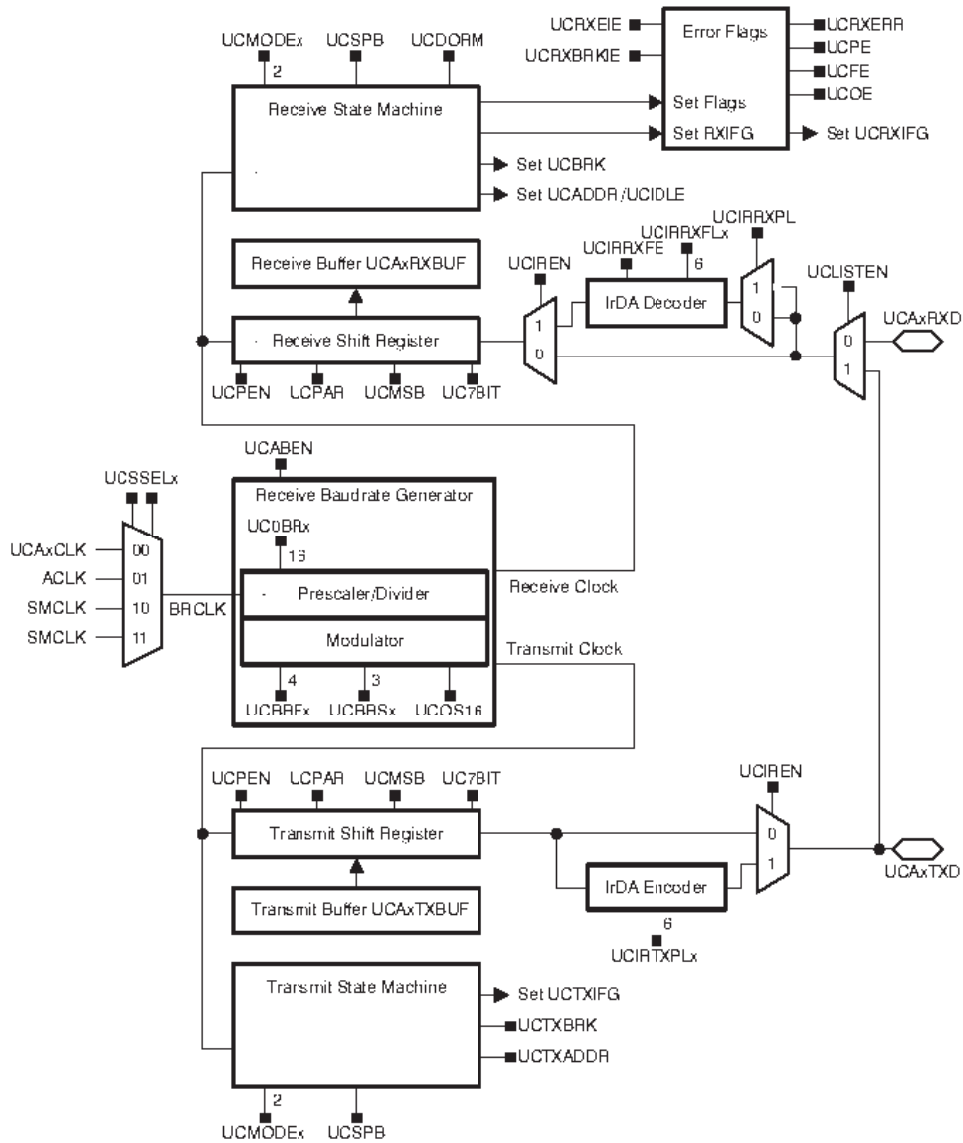


Figure 9.3: Block diagram of the eUCSI_{Ax} module configured for UART mode (UCSYNC bit = 0) [slau208g]. Illustration used with permission of Texas Instruments www.ti.com.

The Baud rate is set using the Baudrate generator shown in the center of Figure 9.3. The clock source for the Baudrate generator may either be the UCAXCLK, the ACLK, or the SMCLK. The clock source is selected using the eUSCI clock source select bits (UCSSEL_x) located in the eUSCI_Ax Control Register 1 (UCAXCTL1). The source selected becomes the Baudrate clock (BRCLK). The Baudrate clock may then be prescaled and divided to set the Baud rate for the transmit and receive clock.

The receive portion of the UART system is in the upper portion of Figure 9.3. Serial data is received via the UCAXRXD pin. The serial data is routed into the Receive Shift Register when the UCLISTEN bit located within the eUSCI_Ax Status Register (UCAXSTAT) is set to zero. If required by the specific application, the data may first be routed through the IrDA Decoder.

The configuration of the Receive Shift Register is set by several bits located within the eUSCI_Ax Control Register 0 (UCAXCTL0). These include the:

- parity enable bit, UCPEN (0: parity disabled, 1: parity enabled);
- parity select bit, UCPAR (0: odd parity, 1: even parity);
- MSB first select, UCMSB (0: LSB first, 1: MSB first); and
- character length bit, UC7BIT (0: 8-bit data, 1: 7-bit data).

The Receive State Machine controls the operation of the receive associated hardware. It has control bits to:

- select the number of stop bits, UCSPB (0: one stop bit, 1: two stop bits);
- select the eUSCI mode, UCMODE_x (00: UART mode); and
- select the synchronous mode, UCSYNC (0: asynchronous mode, 1: synchronous mode).

The hardware associated with serial data transmission is very similar to the receive hardware with the direction of data routed for transmission out of the UCAXTXD pin.

9.4.3 CHARACTER FORMAT

As previously mentioned, the UART system has great flexibility in setting the protocol of the serial data, including the number of bits (7 or 8), parity (even, odd, or none), MSB or LSB first, and selection of transmit/receive operation. A typical serial data word is illustrated in Figure 9.4. To verify the valid functionality of the communication using the MSP432 microcontroller, it is very helpful to write a short program to transmit the same piece of data continuously from the UART and observe the transmission on the UCAXTXD pin with an oscilloscope or a logic analyzer.

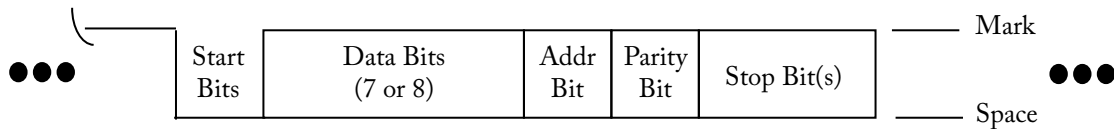


Figure 9.4: UART serial data format [slau208g].

9.4.4 BAUD RATE SELECTION

The MSP microcontroller also has considerable flexibility in setting the Baud rate for UART transmission and reception. It has two different modes for Baud rate generation.

- **Low-frequency Baud rate generation (UCOS16, Oversampling Mode Enable bit = 0).** The mode allows Baud rates to be set when the microcontroller is being clocked by a low frequency clock. It is advantageous to do this to reduce power consumption by using a lower frequency time base. In this mode, the Baudrate Generator uses a prescaler and a modulator to generate the desired Baud rate. The maximum selectable Baud rate in this mode is limited to one-third of the Baud rate clock (BRCLK).
- **Oversampling Baud rate generation (UCOS16 = 1).** This mode employs a prescaler and a modulator to generate higher sampling frequencies.

To set a specific Baud rate, the following parameters must be determined.

- The clock prescaler setting (UCBRx) in the Baud Rate Control Register 0 and 1 (USAxBR0 and UCAxBR0) must be determined. The 16-bit value of the UCBRx prescaler value is determined by $UCAxBR0 + UCAxBR1 \times 256$.
- First, modulation stage setting, UCBRFx bits in the eUSCI_Ax Modulation Control Register (UCAxMCTL).
- Second, modulation stage setting, UCBRSx bits in the eUSCI_Ax Modulation Control Register (UCAxMCTL).

The documentation for the MSP432 microcontroller contains extensive tables for determining the UCBRx, UCBRFx, and UCBRSx bit settings for various combinations of the Baud rate clock (BRCLK) and desired Baud rate [SLAU356A, 2015].

9.4.5 UART ASSOCIATED INTERRUPTS

The UART system has two associated interrupts. The Transmit Interrupt Flag (UCTXIFG) is set when the UCAxTXBUF is empty, indicating another data byte may be sent. The Receive Interrupt Flag (UCRXIFG) is set when the receive buffer (UCAxRXBUF) has received a complete character. Both of these interrupt flags are contained within the eUSCI_Ax Interrupt Flag Register (UCAxIFG).

9.4.6 UART REGISTERS

As discussed throughout this section, the basic features of the UART system is configured and controlled by the following UART related registers [SLAU356A, 2015]:

- UCAxCTLW0 eUSCI_Ax Control Word 0
- UCAxCTL0(1) eUSCI_Ax Control 0
- UCAxCTL1 eUSCI_Ax Control 1
- UCAxCTLW1 eUSCI_Ax Control Word 1
- UCAxBRW eUSCI_Ax Baud Rate Control Word
- UCAxBR0(1) eUSCI_Ax Baud Rate Control 0
- UCAxBR1 eUSCI_Ax Baud Rate Control 1
- UCAxMCTLW eUSCI_Ax Modulation Control Word
- UCAxSTATW eUSCI_Ax Status
- UCAxRXBUF eUSCI_Ax Receive Buffer
- UCAxTXBUF eUSCI_Ax Transmit Buffer
- UCAxABCTL eUSCI_Ax Auto Baud Rate Control
- UCAxIRCTL eUSCI_Ax IrDA Control
- UCAxIRTCTL eUSCI_Ax IrDA Transmit Control
- UCAxIRRCTL eUSCI_Ax IrDA Receive Control
- UCAxIE eUSCI_Ax Interrupt Enable
- UCAxIFG eUSCI_Ax Interrupt Flag
- UCAxIV eUSCI_Ax Interrupt Vector

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

9.4.7 API SUPPORT

Texas Instruments provides extensive MSP432 UART support through a series of Application Program Interfaces (APIs). Provided below is a list of UART data structures and APIs. Details on API settings are provided in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Data Structures

- struct `_eUSCI_eUSCI_UART_Config`

Functions

- void `UART_clearInterruptFlag(uint32_t moduleInstance, uint_fast8_t mask)`
- void `UART_disableInterrupt(uint32_t moduleInstance, uint_fast8_t mask)`
- void `UART_disableModule(uint32_t moduleInstance)`
- void `UART_enableInterrupt(uint32_t moduleInstance, uint_fast8_t mask)`
- void `UART_enableModule(uint32_t moduleInstance)`
- uint_fast8_t `UART_getEnabledInterruptStatus(uint32_t moduleInstance)`
- uint_fast8_t `UART_getInterruptStatus(uint32_t moduleInstance, uint8_t mask)`
- uint32_t `UART_getReceiveBufferAddressForDMA(uint32_t moduleInstance)`
- uint32_t `UART_getTransmitBufferAddressForDMA(uint32_t moduleInstance)`
- bool `UART_initModule(uint32_t moduleInstance, const eUSCI_UART_Config *config)`
- uint_fast8_t `UART_queryStatusFlags(uint32_t moduleInstance, uint_fast8_t mask)`
- uint8_t `UART_receiveData(uint32_t moduleInstance)`
- void `UART_registerInterrupt(uint32_t moduleInstance, void(*intHandler)(void))`
- void `UART_resetDormant(uint32_t moduleInstance)`
- void `UART_selectDeglitchTime(uint32_t moduleInstance, uint32_t deglitchTime)`
- void `UART_setDormant(uint32_t moduleInstance)`
- void `UART_transmitAddress(uint32_t moduleInstance, uint_fast8_t transmitAddress)`
- void `UART_transmitBreak(uint32_t moduleInstance)`
- void `UART_transmitData(uint32_t moduleInstance, uint_fast8_t transmitData)`
- void `UART_unregisterInterrupt(uint32_t moduleInstance)`

Texas Instruments provides extensive MSP432 UART support through a series of Application Program Interfaces (APIs). Provided below is a list of UART data structures and APIs. Details on API settings are provided in *MSP432 Peripheral Driver Library User's Guide* [[Driver-Lib, 2015](#)] and will not be repeated here.

9.5 CODE EXAMPLES

The MSP432 UART features may be programmed using Energia, DriverLib APIs, or in C.

9.5.1 ENERGIA

Example: LCD. In this example a Sparkfun LCD-09067, 3.3 VDC, serial, 16 by 2 character, black on white LCD display is connected to the MSP432. Communication between the MSP432 and the LCD is accomplished by a single 9600 bits per second (BAUD) connection using the onboard Universal Asynchronous Receiver Transmitter (UART). The UART is configured for 8 bits, no parity, and one stop bit (8-N-1). The MSP-EXP432P401R LaunchPad is equipped with two UART channels. One is the back channel UART connection to the PC. The other is accessible by pin 3 (RX, P3.2) and pin 4 (TX, P3.3). Provided below is the sample Energia code to print a test message to the LCD. Note the UART is designated “Serial1” in the program. The back channel UART for the Energia serial monitor display is designated “Serial.”

```

//*****
//Serial_LCD_energia
//Serial 1 accessible at:
// - RX: P3.2, pin 3
// - TX: P3.3, pin 4
//*****

void setup()
{
  //Initialize serial channel 1 to 9600 BAUD and wait for port to open
  Serial1.begin(9600);
}

void loop()
{
  Serial1.print("Hello World");
  delay(500);
  Serial1.println("...Hello World");
  delay(500);
}

//*****

```

Example: Voice chip. For speech synthesis, we use the SP0-512 text to speech chip (www.speechechips.com). The SP0-512 accepts UART compatible serial text stream. The text stream should be terminated with the carriage return control sequence (back slash r). The text stream is

converted to phoneme codes used to generate an audio output. The chip requires a 9600 Baud bit stream with no parity, 8 data bits and a stop bit. The associated circuits is provided in Figure 9.5. Additional information on the chip and its features are available at www.speechchips.com.

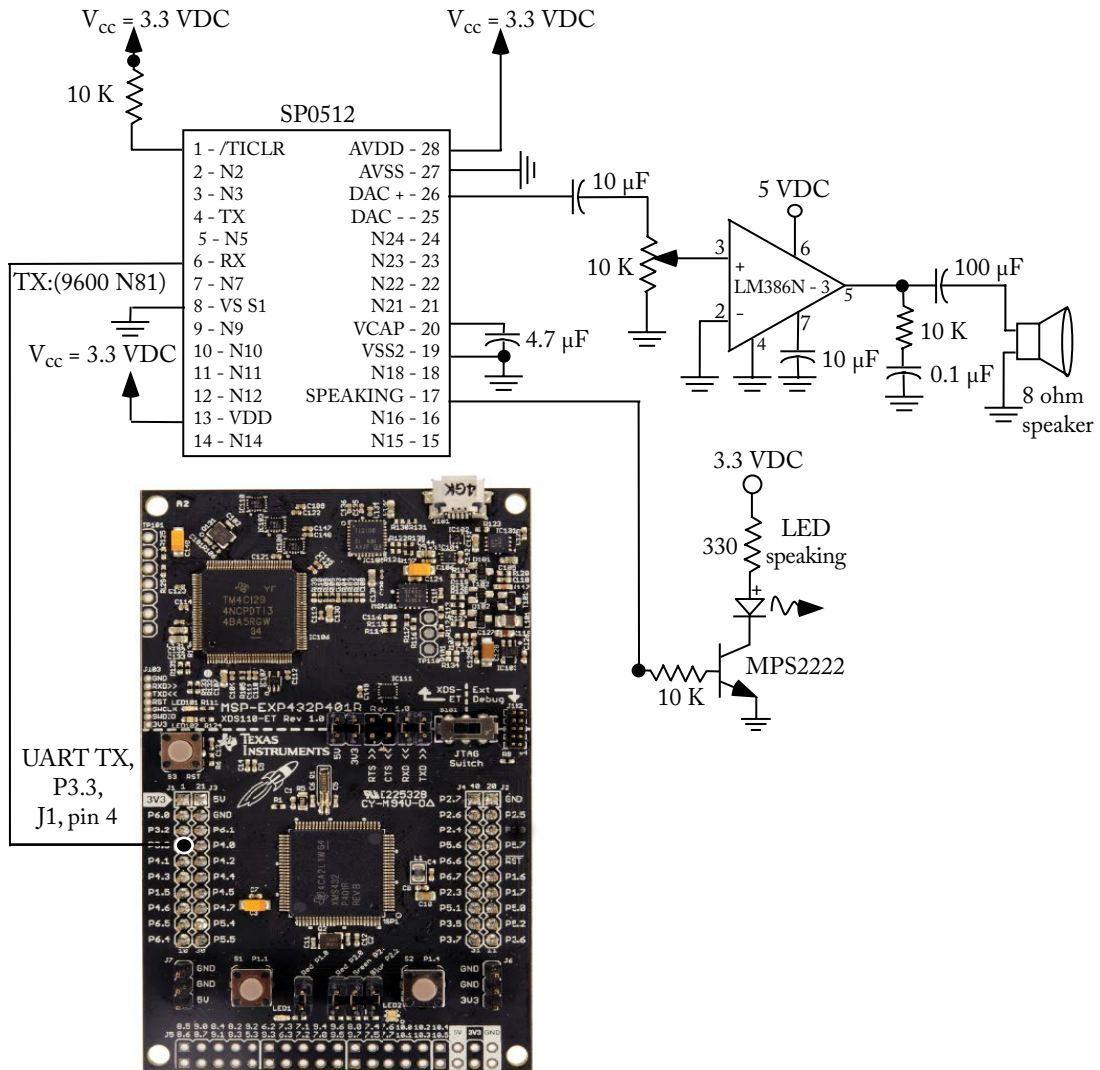


Figure 9.5: Speech synthesis support circuit (www.speechchips.com).

```
//*****
//SP0512
```



```

//Serial 1 accessible at:
// - RX: P3.2, pin 3
// - TX: P3.3, pin 4
//*****

void setup()
{
  //Initialize serial channel 1 to 9600 BAUD and wait for port to open
  Serial1.begin(9600);
}

void loop()
{
  Serial1.print("[BD]This [BD]is [BD]a [BD]test \r");
  delay(3000);
}

//*****

```

9.5.2 UART DRIVERLIB API EXAMPLE

Example: Provided below is a very brief code example showing how to configure and enable the UART module using DriverLib APIs [DriverLib, 2015].

```

//*****
//MCLK operating off of DCO and DCO tuned to 12MHz.
// - eUSCI A UART module to operate with a 9600 baud rate
// - Configuration parameter values determined using online calculator
// at:
//http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/
//      MSP430BaudRateConverter/index.html
//
//Used courtesy of Texas Instruments, Inc.
//
//*****

//UART data structure
const eUSCI_UART_Config uartConfig =
{
  EUSCI_A_UART_CLOCKSOURCE_SMCLK, //SMCLK Clock Source
  78, //BRDIV = 78

```

```

2,                //UCxBRF = 2
0,                //UCxBRS = 0
EUSCI_A_UART_NO_PARITY,    //No Parity
EUSCI_A_UART_MSB_FIRST,    //MSB First
EUSCI_A_UART_ONE_STOP_BIT, //One stop bit
EUSCI_A_UART_MODE,        //UART mode
EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION //Oversampling
};

//This code snippet is the actual configuration of the UART
//module using the DriverLib APIs:

//Configuring UART Module
MAP_UART_initModule(EUSCI_A0_MODULE, &uartConfig);

//Enable UART module
MAP_UART_enableModule(EUSCI_A0_MODULE);

//Enabling interrupts
MAP_UART_enableInterrupt(EUSCI_A0_MODULE,
                        EUSCI_A_UART_RECEIVE_INTERRUPT);
MAP_Interrupt_enableInterrupt(INT_EUSCIA0);
MAP_Interrupt_enableSleepOnIsrExit();
MAP_Interrupt_enableMaster();

//*****

```

9.5.3 UART C EXAMPLE

Example: In this example the MSP432 UART echoes back characters received via a PC serial port. The SMCLK/DCO is used as a clock source and the device is placed to operate in the LPM3 mode. **Note:** Level shifter hardware (MAXIM232) is needed to shift between the RS232 and the MSP 3.3 VDC voltage levels (www.maximintegrated.com).

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.

```

```
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                                MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
```

```

//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - eUSCI_A0 UART echo at 9600 baud using BRCLK = 12MHz
//
//Description: This demo echoes back characters received via a PC serial
//port.
SMCLK/ DCO is used as a clock source and the device is put in
//LPM3. The auto-clock enable feature is used by the eUSCI and SMCLK is
//turned off when the UART is idle and turned on when a receive edge is
//detected.
Note that level shifter hardware is needed to shift between
//RS232 and MSP voltage levels.
//
//The example code shows proper initialization of registers and
//interrupts to receive and transmit data.
To test code in LPM3,
//disconnect the debugger.
//
//
//
//           MSP432P401
//           -----
//           /|\|
//           | |
//           --|RST
//           |
//           |
//           | P1.3/UCA0TXD|----> PC (echo)
//           | P1.2/UCA0RXD|<---- PC
//           |
//
// Wei Zhao
// Texas Instruments Inc.
// June 2014
// Built with Code Composer Studio V6.0

```

```

//*****

#include "msp.h"

int main(void)
{
    WDCTL = WDTW | WDTW;           //Stop watchdog timer

    CSKEY = 0x695A;                //Unlock CS module for reg access
    CSCTL0 = 0;                    //Reset tuning parameters
    CSCTL0 = DCORSEL_3;            //Set DCO to 12MHz (nominal,
                                   //center of 8-16MHz range)
                                   //Select ACLK=REF0, SMCLK=MCLK=DCO

    CSCTL1 = SELA_2 | SELS_3 | SELM_3;
    CSKEY = 0;                     //Lock CS module from unintended
                                   //accesses

    //Configure UART pins
    P1SEL0 |= BIT2 | BIT3;         //set 2-UART pin as second function
    __enable_interrupt();
    NVIC_ISER0 = 1 << ((INT_EUSCIA0 - 16) & 31); //Enable eUSCIA0 interrupt
                                           //in NVIC module

    //Configure UART
    UCACTLW0 |= UCSWRST;
    UCACTLW0 |= UCSSEL__SMCLK;     //Put eUSCI in reset

    //Baud Rate calculation:
    // 12000000/(16*9600) = 78.125
    // Fractional portion = 0.125
    // User's Guide Table 21-4: UCBR5x = 0x10
    // UCBR5x = int ( (78.125-78)*16) = 2
    UCAOBR0 = 78;                  //12000000/16/9600
    UCAOBR1 = 0x00;
    UCAOMCTLW = 0x1000 | UCOS16 | 0x0020;

    UCACTLW0 &= ~UCSWRST;         //Initialize eUSCI
    UCAOIE |= UCRXIE;             //Enable USCI_A0 RX interrupt

```

```

__sleep();
__no_operation();           //For debugger
}

//*****
// UART interrupt service routine
//*****

void eUSCIA0IsrHandler(void)
{
if(UCA0IFG & UCRXIFG)
    {
    while(!(UCA0IFG&UCTXIFG));
    UCA0TXBUF = UCA0RXBUF;
    __no_operation();
    }
}

//*****

```

9.6 SERIAL PERIPHERAL INTERFACE-SPI

The Serial Peripheral Interface or SPI is also used for two-way serial communication between a transmitter and a receiver. In the SPI system, the transmitter and receiver pair shares a common clock source (UCxCLK). This requires an additional clock line between the transmitter and the receiver but allows for higher data transmission rates as compared to the UART. The SPI system allows for fast and efficient data exchange between microcontrollers or peripheral devices. There are many SPI compatible external systems available to extend the features of the microcontroller. For example, a liquid crystal display or a multi-channel digital-to-analog converter could be added to the microcontroller using the SPI system.

9.6.1 SPI OPERATION

The SPI may be viewed as a synchronous 16-bit shift register with an 8-bit, half residing in the transmitter and the other 8-bit half residing in the receiver, as shown in Figure 9.6. The transmitter is designated as the master since it is providing the synchronizing clock source between the transmitter and the receiver. The receiver is designated as the slave. A slave is chosen for reception by taking its Slave Select (\overline{SS}) line low. When the \overline{SS} line is taken low, the slave's shifting capability is enabled.

SPI transmission is initiated by loading a data byte into the master-configured Transmit Buffer (UC_xTXBUF). At that time, the UCSI SPI mode Bit Clock Generator provides clock pulses to the master and also to the slave via the UC_xCLK pin. A single bit is shifted out of the master designated shift register on the Slave In Master Out (UC_xSIMO) microcontroller pin on every SCK pulse. The data is received at the UC_xSIMO pin of the slave designated device. In some peripheral devices, this is referred to as Master Out Slave In (MOSI). At the same time, a single bit is shifted out of the Slave Out Master In (UC_xSOMI) pin of the slave device and into the UC_xSOMI pin of the master device. After eight master UC_xCLK clock pulses, a byte of data has been exchanged between the master and slave designated SPI devices. Completion of data transmission in the master and data reception in the slave is signaled by SPI related interrupts in both devices. At that time, another data byte may be transmitted.

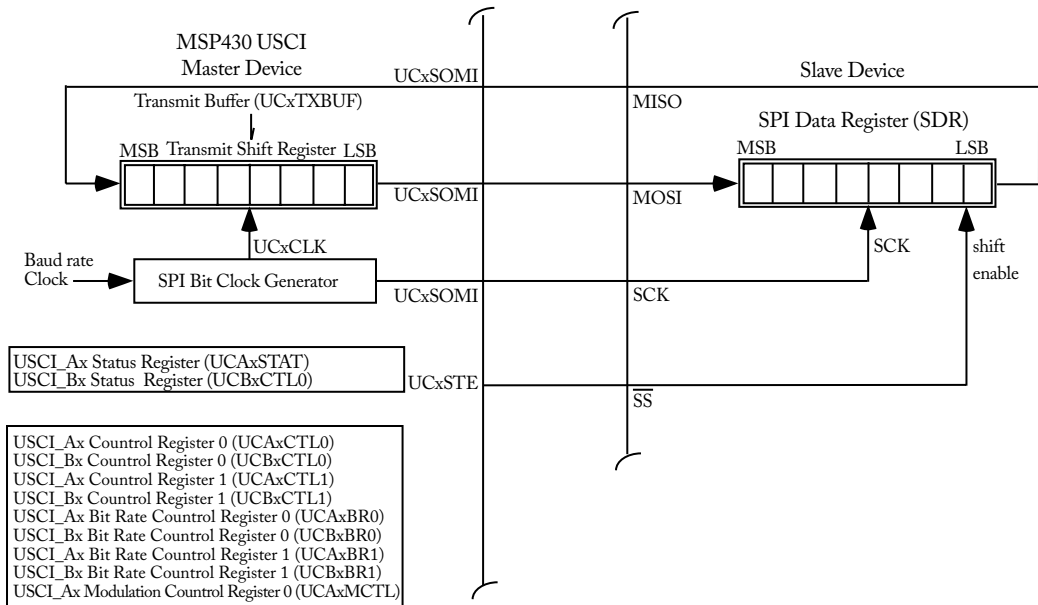


Figure 9.6: SPI overview.

9.6.2 MSP432 SPI FEATURES

As previously mentioned, the MSP432 SPI system has many features that allow the system to be interfaced to a wide variety of SPI configured devices. These features include [SLAU356A, 2015]:

- 7-bit or 8-bit data length;
- LSB-first or MSB-first data transmit and receive capability;

- 3 or 4 wire SPI operation;
- master or slave modes;
- independent transmit and receive shift registers which provide continuous transmit and receive operation;
- selectable clock polarity and phase control;
- programmable clock frequency in master mode; and
- independent interrupt capability for receive and transmit.

9.6.3 MSP432 SPI HARDWARE CONFIGURATION

The MSP432 provides support for SPI communication in both of the eUSCI_A and eUSCI_B modules. A block diagram of an UCSI module configured for SPI operation is shown in Figure 9.7. SPI operation is selected by setting the UCSYNC (Synchronous mode enable) bit to logic one in the module's eUSCI_Ax or USCI_Bx Control Register 0 (UCAxCTL0 or UCBxCTL0).

Located in the center of the diagram, the clock source for the SPI Baud rate clock (BRCLK) is either provided by the ACLK or the SMCLK. The clock source is chosen using the eUSCI clock source select (UCSSELx) bits in eUSCI_Ax (or B) Control Register 1 (UCAxCTL1 or UCBxCTL1).

The Baud rate clock is fed to the Bit Clock Generator. The 16-bit clock prescaler is formed using $(UC_{xx}BR0 + UC_{xx}BR1 \times 256)$. The values for UC_{xx}BR0 and UC_{xx}BR1 are contained in the eUSCI_{xx} Bit Rate Control Registers 0 and 1 (UC_{xx}BR0 and UC_{xx}BR1).

The MSP432 eUSCI provides the flexibility to configure the SPI data transmission format to match that of many different peripheral devices. Either a seven or eight bit data format may be selected using the UC7BIT. Also, the phase and polarity of the data stream may be adjusted to match peripheral devices. The polarity setting determines active high or low transmission while the polarity bit determines if the signal is asserted in the first half of the bit frame or in the second half. Furthermore, the data may be transmitted with the least significant bit (LSB) first or the most significant bit (MSB) first. In summary, the serial data stream format is configured using the following bits in the eUSCI_Ax (or Bx) Control Register 0 (UCAxCTL0) [SLAU356A, 2015].

- UCCCPH: clock phase select bit - 0: data changed on the first UCLK edge and captured on the following edge; 1: data captured on the first edge and changed on the second edge.
- UCCKPL: clock polarity select bit - 0: inactive state low; 1: inactive state high.
- UCMSB: MSB first select bit - 0: LSB transmitted first; 1: MSB transmitted first.
- UC7BIT: character length select bit - 0: 8-bit data; 1: 7-bit data.

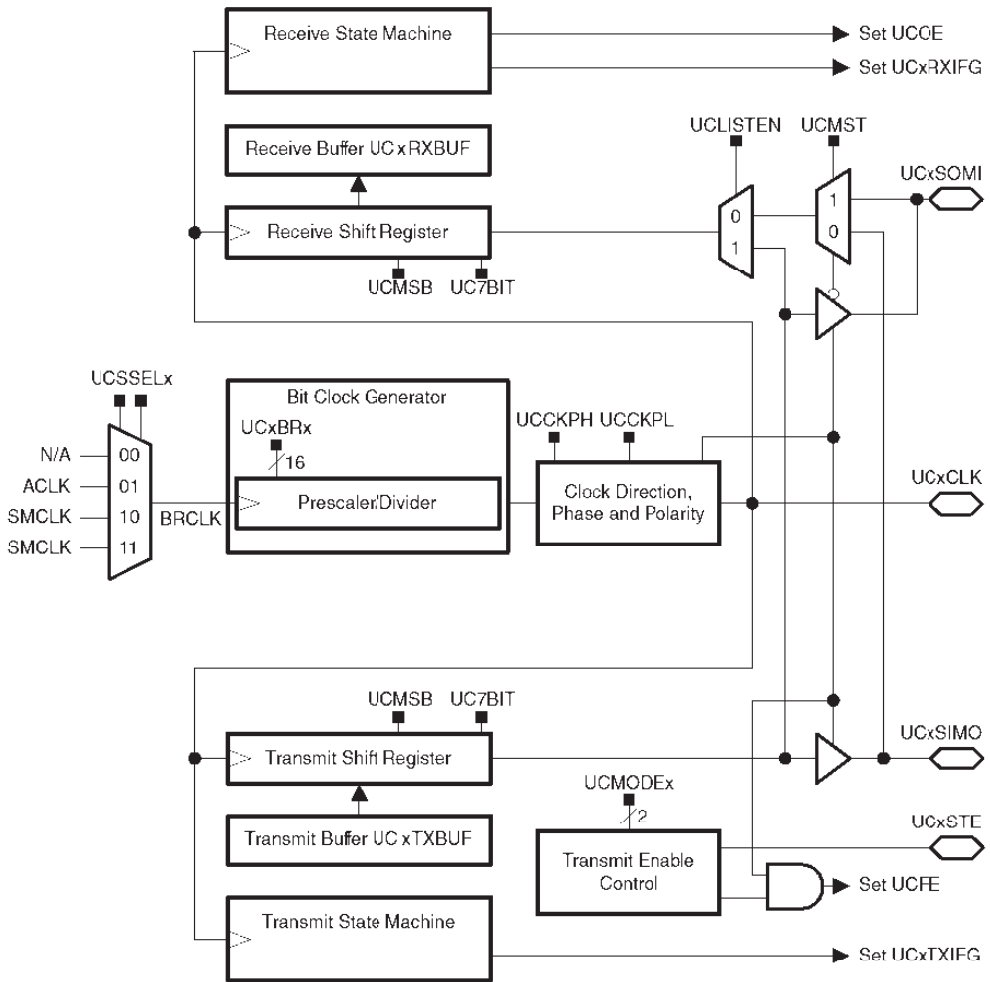


Figure 9.7: SPI hardware overview [SLAU356A, 2015]. Illustration used with permission of Texas Instruments www.ti.com.

The clock signal is routed from the Bit Clock Generator to both the receive state machine and the transmit state machine. To transmit data, the data is loaded to the Transmit Buffer (UCxTXBUF). Writing to the UCxTXBUF activates the bit clock generator. The data begins to transmit. Also, the SPI system receives data when the transmission is active. The transmit and receive operations occur simultaneously [SLAU356A, 2015].

The SPI system is also equipped with interrupts. The UXTXIFG interrupt flag in the eUSCI_Ax (or Bx) Interrupt Flag Register (UCAxIFG, UCBxIFG) is set when the UCxxTXBUF is empty indicating another character may be transmitted. The UCRXIFG interrupt flag is set when a complete character has been received.

9.6.4 SPI REGISTERS

As discussed throughout this section, the basic features of the SPI system is configured and controlled by the following SPI-related registers [SLAU356A, 2015]:

eUSCI_A SPI Registers

- UCAxCTLW0 eUSCI_Ax Control Word 0
- UCAxCTL1 eUSCI_Ax Control 1
- UCAxCTL0 eUSCI_Ax Control 0
- UCAxBRW eUSCI_Ax Bit Rate Control Word
- UCAxBR0 eUSCI_Ax Bit Rate Control 0
- UCAxBR1 eUSCI_Ax Bit Rate Control 1
- UCAxSTATW eUSCI_Ax Status
- UCAxRXBUF eUSCI_Ax Receive Buffer
- UCAxTXBUF eUSCI_Ax Transmit Buffer
- UCAxIE eUSCI_Ax Interrupt Enable
- UCAxIFG eUSCI_Ax Interrupt Flag
- UCAxIV eUSCI_Ax Interrupt Vector

eUSCI_B SPI Registers

- UCBxCTLW0 eUSCI_Bx Control Word 0
- UCBxCTL1 eUSCI_Bx Control 1
- UCBxCTL0 eUSCI_Bx Control 0

- UCBxBRW eUSCI_Bx Bit Rate Control Word
- UCBxBR0 eUSCI_Bx Bit Rate Control 0
- UCBxBR1 eUSCI_Bx Bit Rate Control 1
- UCBxSTATW eUSCI_Bx Status
- UCBxRXBUF eUSCI_Bx Receive Buffer
- UCBxTXBUF eUSCI_Bx Transmit Buffer
- UCBxIE eUSCI_Bx Interrupt Enable
- UCBxIFG eUSCI_Bx Interrupt Flag
- UCBxIV eUSCI_Bx Interrupt Vector

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

9.6.5 SPI DATA STRUCTURES API SUPPORT

Texas Instruments provides extensive MSP432 SPI support through a series of Application Program Interfaces (APIs). Provided below is a list of SPI data structures and APIs. Details on API settings are provided in *MSP432 Peripheral Driver Library User's Guide* [DriverLib, 2015] and will not be repeated here.

Data Structures

```
struct _eUSCI_SPI_MasterConfig
struct _eUSCI_SPI_SlaveConfig
```

Typedefs

```
typedef struct
_eUSCI_SPI_MasterConfig eUSCI_SPI_MasterConfig
typedef struct
_eUSCI_SPI_SlaveConfig eUSCI_SPI_SlaveConfig
```

Functions

- void EUSCI_A_SPI_changeClockPhasePolarity(uint32_t baseAddress, uint16_t clockPhase, uint16_t clockPolarity)
- void EUSCI_A_SPI_clearInterruptFlag(uint32_t baseAddress, uint8_t mask)
- void EUSCI_A_SPI_disable(uint32_t baseAddress)

- void EUSCI_A_SPI_disableInterrupt(uint32_t baseAddress, uint8_t mask)
- void EUSCI_A_SPI_enable(uint32_t baseAddress)
- void EUSCI_A_SPI_enableInterrupt(uint32_t baseAddress, uint8_t mask)
- uint8_t EUSCI_A_SPI_getInterruptStatus(uint32_t baseAddress, uint8_t mask)
- uint32_t EUSCI_A_SPI_getReceiveBufferAddressForDMA(uint32_t baseAddress)
- uint32_t EUSCI_A_SPI_getTransmitBufferAddressForDMA(uint32_t baseAddress)
- bool EUSCI_A_SPI_isBusy(uint32_t baseAddress)
- void EUSCI_A_SPI_masterChangeClock(uint32_t baseAddress, uint32_t clockSource-Frequency, uint32_t desiredSpiClock)
- uint8_t EUSCI_A_SPI_receiveData(uint32_t baseAddress)
- void EUSCI_A_SPI_select4PinFunctionality(uint32_t baseAddress, uint8_t select4PinFunctionality)
- bool EUSCI_A_SPI_slaveInit(uint32_t baseAddress, uint16_t msbFirst, uint16_t clock-Phase, uint16_t clockPolarity, uint16_t spiMode)
- void EUSCI_A_SPI_transmitData(uint32_t baseAddress, uint8_t transmitData)
- void EUSCI_B_SPI_changeClockPhasePolarity(uint32_t baseAddress, uint16_t clock-Phase, uint16_t clockPolarity)
- void EUSCI_B_SPI_clearInterruptFlag(uint32_t baseAddress, uint8_t mask)
- void EUSCI_B_SPI_disable(uint32_t baseAddress)
- void EUSCI_B_SPI_disableInterrupt(uint32_t baseAddress, uint8_t mask)
- void EUSCI_B_SPI_enable(uint32_t baseAddress)
- void EUSCI_B_SPI_enableInterrupt(uint32_t baseAddress, uint8_t mask)
- uint8_t EUSCI_B_SPI_getInterruptStatus(uint32_t baseAddress, uint8_t mask)
- uint32_t EUSCI_B_SPI_getReceiveBufferAddressForDMA(uint32_t baseAddress)
- uint32_t EUSCI_B_SPI_getTransmitBufferAddressForDMA(uint32_t baseAddress)
- bool EUSCI_B_SPI_isBusy(uint32_t baseAddress)

- void EUSCI_B_SPI_masterChangeClock(uint32_t baseAddress, uint32_t clockSourceFrequency, uint32_t desiredSpiClock)
- uint8_t EUSCI_B_SPI_receiveData(uint32_t baseAddress)
- void EUSCI_B_SPI_select4PinFunctionality(uint32_t baseAddress, uint8_t select4PinFunctionality)
- bool EUSCI_B_SPI_slaveInit(uint32_t baseAddress, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- void EUSCI_B_SPI_transmitData(uint32_t baseAddress, uint8_t transmitData)
- void SPI_changeClockPhasePolarity(uint32_t moduleInstance, uint_fast16_t clockPhase, uint_fast16_t clockPolarity)
- void SPI_changeMasterClock(uint32_t moduleInstance, uint32_t clockSourceFrequency, uint32_t desiredSpiClock)
- void SPI_clearInterruptFlag(uint32_t moduleInstance, uint_fast8_t mask)
- void SPI_disableInterrupt(uint32_t moduleInstance, uint_fast8_t mask)
- void SPI_disableModule(uint32_t moduleInstance)
- void SPI_enableInterrupt(uint32_t moduleInstance, uint_fast8_t mask)
- void SPI_enableModule(uint32_t moduleInstance)
- uint_fast8_t SPI_getEnabledInterruptStatus(uint32_t moduleInstance)
- uint_fast8_t SPI_getInterruptStatus(uint32_t moduleInstance, uint16_t mask)
- uint32_t SPI_getReceiveBufferAddressForDMA(uint32_t moduleInstance)
- uint32_t SPI_getTransmitBufferAddressForDMA(uint32_t moduleInstance)
- bool SPI_initMaster(uint32_t moduleInstance, const eUSCI_SPI_MasterConfig *config)
- bool SPI_initSlave(uint32_t moduleInstance, const eUSCI_SPI_SlaveConfig *config)
- uint_fast8_t SPI_isBusy(uint32_t moduleInstance)
- uint8_t SPI_receiveData(uint32_t moduleInstance) void SPI_registerInterrupt(uint32_t moduleInstance, void(*intHandler)(void))
- void SPI_selectFourPinFunctionality(uint32_t moduleInstance, uint_fast8_t select4PinFunctionality)
- void SPI_transmitData(uint32_t moduleInstance, uint_fast8_t transmitData)
- void SPI_unregisterInterrupt(uint32_t moduleInstance)

9.6.6 SPI CODE EXAMPLES

Energia

In Chapter 2 we used the Energia SPI features to control a one meter, 32 RGB LED strip available from Adafruit (#306) (www.adafruit.com). Recall the red, blue, and green component of each RGB LED was independently set using an eight-bit code. The most significant bit (MSB) was logic one followed by seven bits to set the LED intensity (0–127). The component values were sequentially shifted out of the MSP432-EXP432P401R LaunchPad using the Serial Peripheral Interface (SPI) features.

SPI API Example

Example: In this code example the SPI module is configured in three wire master mode.

```
//*****
//SPI Master Configuration Parameter
const eUSCI_SPI_MasterConfig spiMasterConfig =
{
EUSCI_A_SPI_CLOCKSOURCE_ACLK,           //ACLK Clock Source
32768,                                   //ACLK = LFXT = 32.768khz
500000,                                  //SPICLK = 500khz
EUSCI_A_SPI_MSB_FIRST,                  //MSB First
EUSCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT, //Phase
EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_HIGH, //High polarity
EUSCI_A_SPI_3PIN                          //3Wire SPI Mode
};

//Selecting P1.1 P1.2 and P1.3 in SPI mode
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P1,
GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

//Configuring SPI in 3wire master mode
SPI_initMaster(EUSCI_A0_MODULE, &spiMasterConfig);

//Enable SPI module
SPI_enableModule(EUSCI_A0_MODULE);

//Enabling interrupts
SPI_enableInterrupt(EUSCI_A0_MODULE, EUSCI_A_SPI_RECEIVE_INTERRUPT);
Interrupt_enableInterrupt(INT_EUSCIA0);
Interrupt_enableSleepOnIsrExit();
```

```
//*****
```

SPI C Example

In this example, code is provided for both the SPI master and the SPI slave configured processor using the SPI 3-wire mode. Incrementing data is sent by the master configured processor starting at 0x01. The slave configured processor received data is expected to be same as the previous transmission $TXData = RXData - 1$. The eUSCI RX interrupt service routine is used to handle communication with the processor (www.ti.com).

Master configured SPI processor code:

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
```

```

//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - eUSCI_A3, SPI 3-Wire Master Incremented Data
//
//Description: SPI master talks to SPI slave using 3-wire mode.
//Incrementing data is sent by the master starting at 0x01.
//Received data is expected to be same as the previous transmission
//TXData = RXData-1. USCI RX ISR is used to handle communication with
//the CPU, normally in LPM0.
//
//
//                               MSP432P401R
//                               -----
//          /|\|                    |
//          | |                    |
//          --|RST                  |
//          |                      |
//          |                      P9.7|-> Data In (UCA3SIMO)

```



```

//          |          |
//          |          P9.6|<- Data OUT (UCA3SOMI)
//          |          |
//          |          P9.5|-> Serial Clock Out (UCA3CLK)
//
//
//Wei Zhao
//Texas Instruments Inc.
//June 2014
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"

static uint8_t RXData = 0;
static uint8_t TXData;

int main(void)
{
volatile uint32_t i;

WDTCTL = WDTPW | WDTHOLD;           //Stop watchdog timer

P9SEL0 |= BIT5 | BIT6 | BIT7;      //set 3-SPI pin as second
                                   //function

__enable_interrupt();
NVIC_ISER0 = 1 << ((INT_EUSCIA3 - 16) & 31); //Enable eUSCIA3
                                             //interrupt in NVIC module

UCA3CTLW0 |= UCSWRST;               /**Put state machine in
                                   //reset**
UCA3CTLW0 |= UCMST|UCSYNC|UCCKPL|UCMSB; //3-pin, 8-bit SPI master
                                   //Clock polarity high, MSB
UCA3CTLW0 |= UCSSEL__ACLK;          //ACLK
UCA3BRO = 0x01;                     // /2,fBitClock =
                                   //fBRCLK/(UCBRx+1).

UCA3BR1 = 0;
UCA3MCTLW = 0;                       //No modulation

```

```

// **Initialize USCI state machine**
UCA3CTLW0 &= ~UCSWRST;                //UCA3IE |= UCRXIE;

//Enable USCI_A3 RX interrupt
TXData = 0x01;                        //Holds TX data

SCB_SCR &= ~SCB_SCR_SLEEPONEXIT;      //Wake up on exit from ISR

while(1)
{
    UCA3IE |= UCTXIE;                  //Enable TX interrupt
    __sleep();
    __no_operation();                 //For debug,Remain in LPM0

for(i = 2000; i > 0; i--);            //Delay before next tx
    TXData++;                          //Increment transmit data

}
}

//*****
//SPI interrupt service routine
//*****

void eUSCIA3IsrHandler(void)
{
    if(UCA3IFG & UCTXIFG)
    {
        UCA3TXBUF = TXData;            //Transmit characters
        UCA3IE &= ~UCTXIE;
        while (!(UCA3IFG&UCRXIFG));
        RXData = UCA3RXBUF;
        UCA3IFG &= ~UCRXIFG;
    }
}

//*****

```


9.7 INTER-INTEGRATED COMMUNICATION - I²C MODULE

9.7.1 OVERVIEW

The I²C is one of prominent communication modes used when multiple serial devices are inter-connected using a serial bus. The I²C bus is a two-wire bus with the serial data line (SDL) and the serial clock line (SCL). By configuring devices connected to the common I²C line as either a master device or a slave device, multiple devices can share information using the common bus.

The difference between a master device and a slave device is that a master device initiates a communication by means of either requesting data from another device or sending data to a designated device. A master device must also provide a clock signal (SCL).

The MSP432's universal serial communication interface (USCI) can be programmed to operate in the I²C communication mode. As seen earlier in this chapter, the eUSCI_Ax ports are programmed to operate in the UART, IrDA, and SPI communication mode while the eUSCI_Bx ports are used for the I²C and SPI serial communication modes.

I²C communication is initiated by sending an address of a desired destination device connected to a common I²C bus. The device address can be either a 7-bit number or a 10-bit number, depending on the number of devices connected to the bus. Of course, all devices on the same bus must use the same addressing mode and program accordingly.

One of the reasons the I²C serial communication became popular is its flexibility to allow multiple master devices to co-exist on a same bus. The MSP432 eUSCI device allows its I²C communication unit to operate either in the standard mode (100 kbps) or in the fast mode (400 kbps).

9.7.2 PROGRAMMING

To initialize a eUSCI_Bx port as an I²C communication port, you must: (1) set the UCSWRST bit in the UCxCTL1 register to one, (2) configure the I²C mode of operation by setting UC-MODEx bits to 11 and initialize the eUSCI registers, and (3) set up an actual port with a pull up resistor. As soon as the UCSWRST bit is cleared, the I²C communication of MSP432 can commence.

The communication performed on the I²C bus must follow a set of agreed rules, including the data format used on the bus. Data is transferred between devices connected on the bus in 8 bits per segment, followed by control bits. For each communication “session,” it must be started by a master device with a start condition, which is defined as the signal changing from logic high to low on the SDA line while the logic state on the SCL line is high. Following the start condition, the master device must send either the 7- or 10-bit address of a destination device on the SDA line.

7-bit addressing mode

# clocks	1	7	1	1	8	1	8	1	1
Data on SDA line	S	A	R/W	Ack	D	Ack	D	Ack	P

10-bit addressing mode

# clocks	1	7	1	1	8	1	8	1	8	1	1
Data on SDA line	S	A1	R/W	Ack	A2	Ack	D	Ack	D	Ack	P

S - Start condition

A - Slave address (7-bit addressing mode)

A1 - MSB slave address - 11110 xx (10-bit addressing mode)

A2 - LSB slave address

R/W - Read or Write

D - Data

Ack - Acknowledgment

P - Stop condition

Figure 9.8: Data format for both 7-bit and 10-bit addressing modes.

Following the address, the master device sends a Read/Write bit describing its intent and listens on the bus to hear an acknowledge bit from the receiver on the 9th SCL clock for the 7-bit addressing mode or on the both 9th and 18th clocks for the 10-bit addressing mode.

For the 10-bit addressing mode, the 10-bit address is split into two segments: two most significant bits (MSBs) and eight least significant bits (LSBs). The MSBs are sent along with pre-designated bits (11110), and the LSBs are sent separately. After the first part of the address is sent, a Read/Write bit, followed by an acknowledgment bit, must appear on the bus before the second part of the address is sent. After the second part of the address, an acknowledgement bit must appear before data is sent over the bus. Figure 9.8 shows the format of data transfer between two devices, using both the 7-bit and the 10-bit addressing modes. For each communication session, it must end with a stop condition (P in the figure), which is defined as the signal state on the SDA line changing from logic low to logic high while the clock signal on the SCL line is high.

9.7.3 MSP432 AS A SLAVE DEVICE

The MSP432 microcontroller can also be configured to be either as a slave device or as a master device. To configure the controller as a slave device, the eUSCI_Bx ports must first be programmed to operate in the I²C slave mode (UCMODEx = 11, UCSYNC = 1, UCMST = 0). The slave address of MSP432 is defined using UCBxI2COA register. The UCA10 bit in the UCBx Control

Register 0 (UCBxCTL0) determines whether the controller is using a 7-bit address or a 10-bit address.

You can program the MSP432 microcontroller to respond to a general call by setting the general call response enable bit (UCGCEN) in the UCBxI2COA register. To receive device addresses sent by masters, the eUSCI_Bx ports must also be configured in the receiver mode (UCTR = 0). When the start condition is detected on the bus, the address bits are compared, and if there is a match, the UCSTTIFG flag is set.

After testing that the Read/Write bit is high, MSP432 uses the clock signal on the SLK line to send data on the SDA line. To do so, the UCTR and UCTXIFG bits are set while holding the SCL line logic low. While the logic state on the SCL line is low, the transmit buffer register (UCBxTXBUF) is loaded with data. Once the buffer is loaded, the UCSTTIFG flag is cleared, which sends the data out to the SDA line, and the UCTXIFG flag is automatically set again for the next data to be transmitted, which occurs after an acknowledge bit is detected on the bus. If the not-acknowledge (NACK) bit is detected, followed by a stop condition, instead, the UCSTPIFG flag is set. If the NACK bit is detected followed by a start condition, MSP432 starts to monitor this device address, again, on the SDA line.

If the MSP432 controller should receive data from a slave device (the Read/Write bit is low), the UCTR bit is cleared, the receive buffer (UCBxRXBUF) is loaded with the data from the bus, and the UCRXIFG flag is set, acknowledging the receipt of the data. Once the data in the bus is read, the flag is cleared, and the controller is ready to receive the next 8-bit data. The controller has an option to send the UCTXNACK bit to a master to release the bus. When a stop condition is detected on the bus, the UCSTPIFG flag is set. If two repeated start conditions are detected or the UCSTPIFG flag is set, the MSP432 terminates its current session and starts monitoring its address on the bus.

9.7.4 MSP432 AS A MASTER DEVICE

To configure the MSP432 controller to function as a master device, the eUSCI_Bx ports must be programmed to operate in the I²C mode (UCMODEx = 11, UCSYNC = 1), and one must configure the MSP432 to operate in the master mode by setting the UCMST bit. Since the I²C bus can handle more than one master device and if there are multiple master devices, the MSP432 needs to be programmed as one of many master devices on the bus by setting the UCMM bit and storing the address (either 7- or 10-bits) of MSP432 in the UCBxI2COA register. As in the case of the slave mode, the address size is determined by the UCA10 bit, and the general call response is programmed using the UCGCEN bit.

To initiate a session to transmit data, the UCTR bit and the UCTxSTT bit are set, the UCSLA10 bit is configured to match the slave address size, and the address of a slave device is loaded to the UCBxI2CSA. When the start condition is generated by setting the UCTxSTT bit, the data can be loaded to the UCBxTXBUF, and the UCTxIFG bit is set. Once a slave address acknowledges its address, the UCTxSTT and UCTxIFG bits are cleared. Once the data is sent,

the UCTxIFG flag bit is set, again, for the next set of data transfer. To generate a stop condition, set UCTxSTP bit while UCTxIFG and UCTxSTP bits are set. If a repeated start conditions are necessary, set UCTxSTT bit. During a data transfer session, if a slave does not respond (i.e., send acknowledge bits), the MSP432 must either send a stop condition or a repeated start conditions.

When the MSP432 controller needs to receive data from a slave, the UCTR bit must be cleared, and the UCTxSTT bit must be set to generate a start condition. When a slave device sends an acknowledgement, the UCTxSTT bit is cleared, and the data is received. Upon receiving an 8-bit data set, the UCRxIFG flag is set. Once the data is read from the buffer, the UCRxIFG flag is cleared, and the next data can be received. If only a single 8-bit byte should be received, the controller must set the UCTxSTP bit while the byte is received.

9.7.5 I²C REGISTERS

I²C associated registers include:

- UCBxCTLW0 eUSCI_Bx Control Word 0
- UCBxCTL1 eUSCI_Bx Control 1
- UCBxCTL0 eUSCI_Bx Control 0
- UCBxCTLW1 eUSCI_Bx Control Word 1
- UCBxBRW eUSCI_Bx Bit Rate Control Word
- UCBxBR0 eUSCI_Bx Bit Rate Control 0
- UCBxBR1 eUSCI_Bx Bit Rate Control 1
- UCBxSTATW eUSCI_Bx Status Word
- UCBxSTAT eUSCI_Bx Status
- UCBxBCNT eUSCI_Bx Byte Counter Register
- UCBxTBCNT eUSCI_Bx Byte Counter Threshold Register
- UCBxRXBUF eUSCI_Bx Receive Buffer
- UCBxTXBUF eUSCI_Bx Transmit Buffer
- UCBxI2COA0 eUSCI_Bx I2C Own Address 0
- UCBxI2COA1 eUSCI_Bx I2C Own Address 1
- UCBxI2COA2 eUSCI_Bx I2C Own Address 2
- UCBxI2COA3 eUSCI_Bx I2C Own Address 3

- UCBxADDRX eUSCI_Bx Received Address Register
- UCBxADDMASK eUSCI_Bx Address Mask Register
- UCBxI2CSA eUSCI_Bx I2C Slave Address
- UCBxIE eUSCI_Bx Interrupt Enable
- UCBxIFG eUSCI_Bx Interrupt Flag
- UCBxIV eUSCI_Bx Interrupt Vector

9.7.6 I²C API SUPPORT

In this section, we list the data structures and API functions associated with the I²C subsystem.

Data Structures

```
struct _eUSCI_I2C_MasterConfig
```

Functions

- void I2C_clearInterruptFlag(uint32_t moduleInstance, uint_fast16_t mask)
- void I2C_disableInterrupt(uint32_t moduleInstance, uint_fast16_t mask)
- void I2C_disableModule(uint32_t moduleInstance)
- void I2C_disableMultiMasterMode(uint32_t moduleInstance)
- void I2C_enableInterrupt(uint32_t moduleInstance, uint_fast16_t mask)
- void I2C_enableModule(uint32_t moduleInstance)
- void I2C_enableMultiMasterMode(uint32_t moduleInstance)
- uint_fast16_t I2C_getEnabledInterruptStatus(uint32_t moduleInstance)
- uint_fast16_t I2C_getInterruptStatus(uint32_t moduleInstance, uint16_t mask)
- uint_fast8_t I2C_getMode(uint32_t moduleInstance)
- uint32_t I2C_getReceiveBufferAddressForDMA(uint32_t moduleInstance)
- uint32_t I2C_getTransmitBufferAddressForDMA(uint32_t moduleInstance)
- void I2C_initMaster(uint32_t moduleInstance, const eUSCI_I2C_MasterConfig *config)
- void I2C_initSlave(uint32_t moduleInstance, uint_fast16_t slaveAddress, uint_fast8_t slaveAddressOffset, uint32_t slaveOwnAddressEnable)

- uint8_t I2C_isBusBusy(uint32_t moduleInstance)
- bool I2C_masterIsStartSent(uint32_t moduleInstance)
- uint8_t I2C_masterIsStopSent(uint32_t moduleInstance)
- uint8_t I2C_masterReceiveMultiByteFinish(uint32_t moduleInstance)
- bool I2C_masterReceiveMultiByteFinishWithTimeout(uint32_t moduleInstance, uint8_t *txData, uint32_t timeout)
- uint8_t I2C_masterReceiveMultiByteNext(uint32_t moduleInstance)
- void I2C_masterReceiveMultiByteStop(uint32_t moduleInstance)
- uint8_t I2C_masterReceiveSingle(uint32_t moduleInstance)
- uint8_t I2C_masterReceiveSingleByte(uint32_t moduleInstance)
- void I2C_masterReceiveStart(uint32_t moduleInstance)
- void I2C_masterSendMultiByteFinish(uint32_t moduleInstance, uint8_t txData)
- bool I2C_masterSendMultiByteFinishWithTimeout(uint32_t moduleInstance, uint8_t txData, uint32_t timeout)
- void I2C_masterSendMultiByteNext(uint32_t moduleInstance, uint8_t txData)
- bool I2C_masterSendMultiByteNextWithTimeout(uint32_t moduleInstance, uint8_t txData, uint32_t timeout)
- void I2C_masterSendMultiByteStart(uint32_t moduleInstance, uint8_t txData)
- bool I2C_masterSendMultiByteStartWithTimeout(uint32_t moduleInstance, uint8_t txData, uint32_t timeout)
- void I2C_masterSendMultiByteStop(uint32_t moduleInstance)
- bool I2C_masterSendMultiByteStopWithTimeout(uint32_t moduleInstance, uint32_t timeout)
- void I2C_masterSendSingleByte(uint32_t moduleInstance, uint8_t txData)
- bool I2C_masterSendSingleByteWithTimeout(uint32_t moduleInstance, uint8_t txData, uint32_t timeout)
- void I2C_masterSendStart(uint32_t moduleInstance)
- void I2C_registerInterrupt(uint32_t moduleInstance, void(*intHandler)(void))

- void I2C_setMode(uint32_t moduleInstance, uint_fast8_t mode) void I2C_setSlaveAddress(uint32_t moduleInstance, uint_fast16_t slaveAddress)
- uint8_t I2C_slaveGetData(uint32_t moduleInstance)
- void I2C_slavePutData(uint32_t moduleInstance, uint8_t transmitData)
- void I2C_unregisterInterrupt(uint32_t moduleInstance)

9.7.7 I²C CODE EXAMPLES

API

Provided in Figure 9.9 are UML diagrams to configure the master and slave configured I²C processors. The code example configures a processor for master operation using DriverLib APIs [DriverLib, 2015].

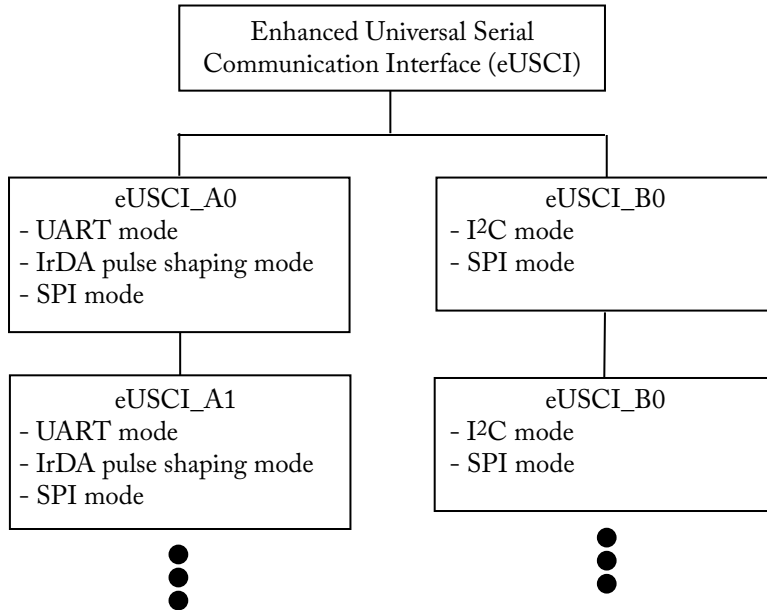


Figure 9.9: I2C UML for master and slave configured processors [DriverLib, 2015].

```

//*****
//I2C Master Configuration Parameter
//*****

const eUSCI_I2C_MasterConfig i2cConfig =

```

```

{
EUSCI_B_I2C_CLOCKSOURCE_SMCLK,    //SMCLK Clock Source
3000000,                          //SMCLK = 3MHz
EUSCI_B_I2C_SET_DATA_RATE_400KBPS, //Desired I2C Clock of 400khz
0,                                //No byte counter threshold
EUSCI_B_I2C_NO_AUTO_STOP          //No Autostop
};

//DriverLib calls to configure/setup the I2C module

//Initializing I2C Master to SMCLK at 400kbs with no autostop
MAP_I2C_initMaster(EUSCI_BO_MODULE, &i2cConfig);

//Specify slave address
MAP_I2C_setSlaveAddress(EUSCI_BO_MODULE, SLAVE_ADDRESS);

//Set Master in receive mode
MAP_I2C_setMode(EUSCI_BO_MODULE, EUSCI_B_I2C_TRANSMIT_MODE);

//Enable I2C Module to start operations
MAP_I2C_enableModule(EUSCI_BO_MODULE);

//Enable and clear the interrupt flag
MAP_I2C_clearInterruptFlag(EUSCI_BO_MODULE,
EUSCI_B_I2C_TRANSMIT_INTERRUPT0 + EUSCI_B_I2C_NAK_INTERRUPT);

//Enable master Receive interrupt
MAP_I2C_enableInterrupt(EUSCI_BO_MODULE,
EUSCI_B_I2C_TRANSMIT_INTERRUPT0 + EUSCI_B_I2C_NAK_INTERRUPT);
MAP_Interrupt_enableInterrupt(INT_EUSCIB0);

//*****

```

C

Master configured processor:

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//

```

440 9. COMMUNICATION SYSTEMS

```
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
```

```
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - eUSCI_B0 I2C Master RX multiple bytes from MSP432
//Slave
//
//Description: This demo connects two MSP432's via the I2C bus.
//The master reads 5 bytes from the slave.
This is the MASTER CODE. The
//data from the slave transmitter begins at 0 and increments with each
//transfer. The USCI_B0 RX interrupt is used to know when new data has
//been received.
//
// ****used with "MSP432P401_euscib0_i2c_11.c"****
//
//
//          /\  /\
//      MSP432P401    10k  10k    MSP432P401
//          slave      |      |      master
//      -----|      |      -----
//      |      P1.6/UCBOSDA|<-|----|->|P1.6/UCBOSDA      |
//      |              |  |      |              |
//      |              |  |      |              |
//      |      P1.7/UCBOSCL|<-|----->|P1.7/UCBOSCL      |
//      |              |              |              P1.0|--> LED
//
//Wei Zhao
//Texas Instruments Inc.
//June 2014
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"
#include <stdint.h>
```

```

uint8_t RXData = 0;

int main(void)
{
volatile uint32_t i;
WDTCTL = WDTPW | WDTHOLD;

//Configure GPIO
P1OUT &= ~BIT0; //Clear P1.0 output latch
P1DIR |= BIT0; //For LED
P1SELO |= BIT6 | BIT7; //I2C pins

__enable_interrupt();
NVIC_ISER0 = 1 << ((INT_EUSCIB0 - 16) & 31); //Enable eUSCIB0 interrupt
//in NVIC module

//Configure USCI_B0 for I2C mode
UCBOCTLW0 |= UCSWRST; //Software reset enabled
UCBOCTLW0 |= UCMODE_3 | UCMST | UCSYNC; //I2C mode, Master mode, sync
UCBOCTLW1 |= UCASTP_2; //Automatic stop generated
//after UCBOCBCNT is reached
UCBOBRW = 0x0018; //baudrate = SMCLK / 8
UCBOCBCNT = 0x0005; //number of bytes to be received
UCBOI2CSA = 0x0048; //Slave address
UCBOCTLW0 &= ~UCSWRST;
UCBOIE |= UCRXIE | UCNACKIE | UCBCNTIE;

while(1)
{
for (i = 2000; i > 0; i--);
while (UCBOCTLW0 & UCTXSTP); //Ensure stop condition got sent
UCBOCTLW0 |= UCTXSTT; //I2C start condition
__sleep(); //Go to LPM0
}
}

//*****
// I2C interrupt service routine
//*****

```

```

void eUSCIB0IsrHandler(void)
{
if(UCB0IFG & UCNACKIFG)
{
UCB0IFG &= ~ UCNACKIFG;
SCB_SCR |= SCB_SCR_SLEEPONEXIT;           //Don't wake up on exit from ISR
UCB0CTLW0 |= UCTXSTT;                       //I2C start condition
}

if(UCB0IFG & UCRXIFG0)
{
UCB0IFG &= ~ UCRXIFG0;
SCB_SCR &= ~SCB_SCR_SLEEPONEXIT;           //Wake up on exit from ISR
RXData = UCBORXBUF;                         //Get RX data
}

if(UCB0IFG & UCBCNTIFG)
{
UCB0IFG &= ~ UCBCNTIFG;
SCB_SCR |= SCB_SCR_SLEEPONEXIT;           //Don't wake up on exit from ISR
P1OUT ^= BIT0;                             //Toggle LED on P1.0
}
}

//*****
Slave configured processor:
//*****
//MP432P401 Demo - eUSCI_B0 I2C Slave TX multiple bytes to MSP432 Master
//
//Description: This demo connects two MSP432's via the I2C bus.
The
//master reads from the slave.
This is the SLAVE code.
The TX data begins
//at 0 and is incremented each time it is sent.
A stop condition is used
//as a trigger to initialize the outgoing data.
The USCIB0 TX interrupt

```


444 9. COMMUNICATION SYSTEMS

```

//is used to know when to TX.
//
// ****used with "MSP432P401_euscib0_i2c_10.c"****
//
//
//          MSP432P401          /\  /\
//          slave          |  |          master
//          -----          |  |          -----
//          |      P1.6/UCBOSDA|<-|----|->|P1.6/UCBOSDA      |
//          |                  |  |          |                  |
//          |                  |  |          |                  |
//          |      P1.7/UCBOSCL|<-|----->|P1.7/UCBOSCL      |
//          |                  |  |          |                  |
//
//Wei Zhao
//Texas Instruments Inc.
//June 2014
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"

uint8_t TXData;

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;

    //Configure GPIO
    P1SELO |= BIT6 | BIT7;                //I2C pins

    __enable_interrupt();
    NVIC_ISER0 = 1 << ((INT_EUSCIB0 - 16) & 31); //Enable eUSCIB0 interrupt
                                                //in NVIC module

    //Configure USCI_B0 for I2C mode
    UCBOCTLW0 = UCSWRST;                  //Software reset enabled
    UCBOCTLW0 |= UCMODE_3 | UCSYNC;       //I2C mode, sync mode
    UCBOI2COA0 = 0x48 | UCOAEN;          //own address is 0x48 + enable

```

```

UCBOCTLW0 &= ~UCSWRST;           //clear reset register
UCBOIE |= UCTXIE0 | UCSTPIE;     //transmit,stop int enable

SCB_SCR |= SCB_SCR_SLEEPONEXIT;  //Don't wake up on ISR exit

__sleep();
__no_operation();
}

//*****
// I2C interrupt service routine
//*****

void eUSCIB0IsrHandler(void)
{
if(UCBOIFG & UCSTPIFG)
    {
    UCBOIFG &= ~ UCSTPIFG;
    TXData = 0;
    UCBOIFG &= ~UCSTPIFG;           //Clear stop condition int flag
    }

if(UCBOIFG & UCTXIFG0)
    {
    UCBOIFG &= ~ UCTXIFG0;
    UCBO_TXBUF = TXData++;
    }
}

//*****

```

9.8 LABORATORY EXERCISE: UART AND SPI COMMUNICATIONS

Configure two MSP432 LaunchPads to communicate using the UART and SPI.

9.9 SUMMARY

In this chapter we have discussed the complement of serial communication features aboard the MSP432 microcontroller. The system is equipped with a host of different serial communication

subsystems including eUSCI A type modules and eUSCI B modules. Each microcontroller in the MSP432 line has a complement of A and B type eUSCI modules.

9.10 REFERENCES AND FURTHER READING

Maxim Integrated, www.maximintegrated.com.

MSP430x5xx and MSP430x6xx Family User's Guide (slaug208g). Texas Instruments, 2015. 405, 407

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015. 408, 409, 412, 422, 438

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. 400, 401, 404, 407, 408, 418, 419, 420, 421, 422

Unicode Consortium, www.unicode.org.

9.11 CHAPTER PROBLEMS

Fundamental

1. Describe the difference between parallel and serial communications.
2. If the communication cost is the primary issue, which communication methods (parallel, series) should be used? Why?
3. What is the difference between synchronous and asynchronous communications?
4. The eUSCI in the UART mode supports LIN and IrDA. For each identify the protocol used: serial/parallel and synchronous/asynchronous.
5. In the I²C communication protocol, how does one configure the MSP432 to become a master device? What must be done to configure it as a slave device?
6. Give a brief description of a communication protocol.

Advanced

1. Write a subroutine that properly initialize the SPI unit. Specify the configuration parameter values used for the external device.
2. Describe interrupts associated with the I²C unit.
3. There are multiple I²C interrupts but a single interrupt vector. After detecting an interrupt, the I²C interrupt system must identify the source of the interrupt. How is this accomplished?

Challenging

1. Design and program three MSP432 controller systems to measure temperatures surrounding the three controllers. Create a wireless communication network using the three controllers along with the CC2530-ZNP radio transceivers. The controllers should constantly share the temperature sensor data among the members. Select a central controller and display the three temperature values on a LCD display once every 5 s.

MSP432 System Integrity

Objectives: After reading this chapter, the reader should be able to:

- describe the sources of noise in a microcontroller system;
- describe the concept of electromagnetic interference (EMI);
- differentiate between conducted and radiated EMI;
- list different sources of EMI;
- list design techniques to minimize EMI;
- describe how a cyclic redundancy check (CRC) may be used to insure the integrity of data;
- describe the features of the CRC32 system onboard the MSP432;
- sketch a linear feedback shift register for a given generator polynomial;
- list common generator polynomials used within CRC systems;
- program the MSP432 CRC32 system to generate a data checksum;
- describe how the MSP432 advanced encryption standard module, the AES256, may be used to provide for data transmission integrity;
- describe the steps used to encrypt/decrypt data using the AES256 standard;
- sketch a block diagram of the MSP432 AES256 module; and
- program the MSP432 AES256 module to encrypt and decrypt data.

10.1 OVERVIEW

This chapter may be the most important chapter in the book. It contains essential information about how to maintain the integrity of a microcontroller based system. The chapter begins with a discussion on electromagnetic interference (EMI), also known as noise. Design practices to minimize EMI are then discussed. The second section of the chapter discusses the concept of the Cyclic Redundancy Check or CRC. This is a hardware-based subsystem used to generate a checksum of a block of data. The checksum may be used to check the integrity of data once it

has been transmitted or loaded to a new location. The final section covers the MSP432 advanced encryption standard module, the AES256. This module is used to insure the integrity of data transmission using a key-based encryption and decryption technique.

10.2 ELECTROMAGNETIC INTERFERENCE

Electromagnetic interference (EMI), commonly referred to as noise, may come from a number of sources as shown in Figure 10.1. Noise causes program malfunction and data corruption, making it impossible to complete the intended task of the controller. It is important to understand the sources of noise and coupling mechanisms to a microcontroller-based project, so proper preventive techniques may be employed during the design process. As shown in Figure 10.1, noise may be coupled to a victim receptor system via radiated or conducted mechanisms. Radiated sources include radio frequency sources such as radio stations and cell phones. Naturally occurring lightning is also a source of noise. A nearby lightning strike generates a tremendous amount of noise at a variety of frequencies. Noise may also be generated by motors and motor based appliances such as drills, mixers, and blenders. Often microcontrollers are used to control a motor. The motor, although part of the designed system, may be a source of noise for the microcontroller controlling its operation. Electrostatic discharge (ESD), e.g., static electricity, may inject noise or damage a microcontroller based system. Conducted sources of noise in a microcontroller based system include other system components or the power supply serving the system. It is interesting to note the microcontroller itself may also serve as a noise source for other system components or nearby systems [AN1705, 2004, COP888, 1996].

10.2.1 EMI REDUCTION STRATEGIES

There are several strategies to follow to minimize EMI interference. These include [AN1705, 2004, COP888, 1996]:

- implementing EMI suppression techniques early in the design process. It is very challenging to provide EMI suppression after a system has been implemented; and
- implementing noise suppression techniques at the source of the noise, disrupting the source to receptor transmission path, protecting the receptor system from noise, and a combination of all three techniques.

Provided below are specific techniques to suppress EMI noise in a microcontroller-based system [Barrett and Pack, 2004, AN1705, 2004, COP888, 1996].

- If possible, incoming signal lines to a microcontroller based system should be twisted. This will minimize the chance of parallel conductors inducing noise in an adjacent conductor via crosstalk. If signals are being transmitted by a multiple conductor ribbon cable, consider gently twisting the cable and also providing a ground conductor alternating with signal carrying conductors.

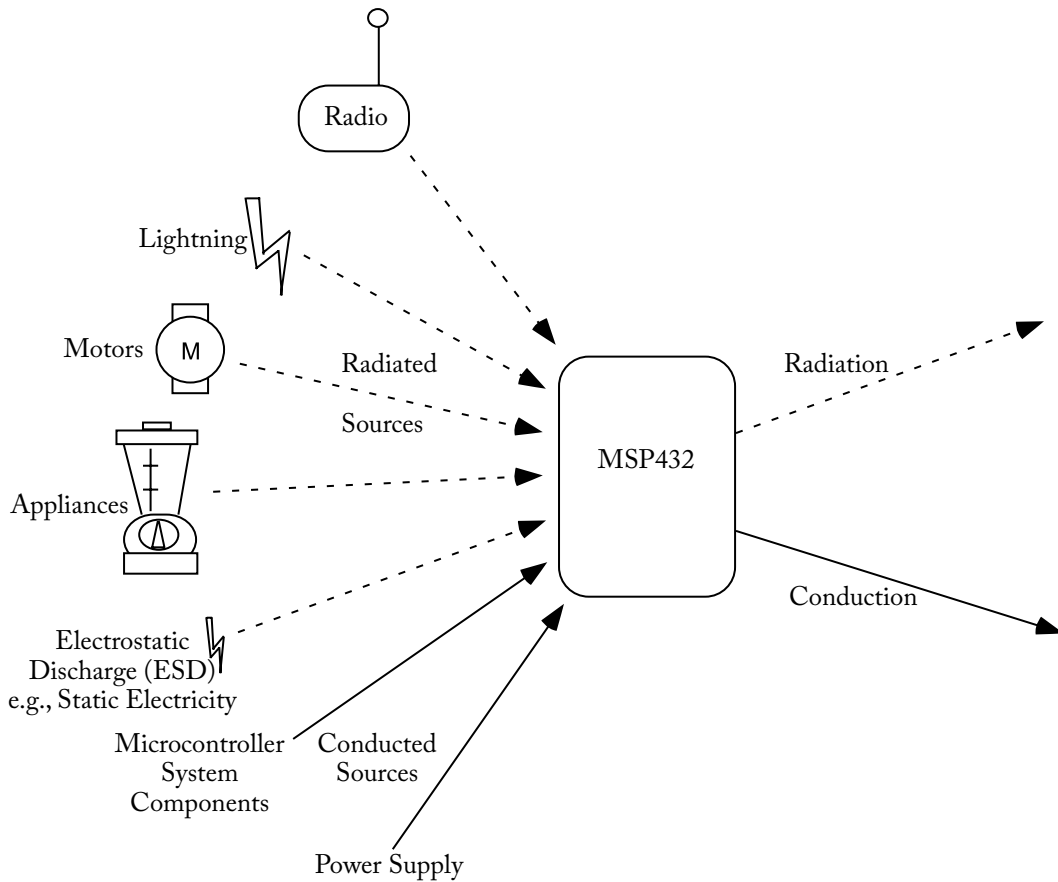


Figure 10.1: Noise sources in a microcontroller-based system (adapted from [AN1705, 2004, COP888, 1996]).

- Use shielded cable for signal conductors coming into the microcontroller based system.
- If the microcontroller is being used to control a motor, use an opto-isolator between the microcontroller and the motor interface circuit. Also, the motor and the microcontroller should not share a common power supply.
- Use filters for signals coming into a microcontroller-based system. Filters are commonly available in the form of ferrite beads.
- Filter the power supply lines to the circuit. Typically, a 10–470 μF capacitor is employed for this purpose. Also, a 0.1 μF capacitor should be used between the power and ground pins on each integrated circuit.

- Ground the metal crystal time base case to insure it does not radiate a noise signal.
- Mount the microcontroller based project in a metal chassis.
- There are several defensive programming techniques to help combat noise. One easy to implement technique is to declare unused microcontroller pins as output.

10.3 CYCLIC REDUNDANCY CHECK

In a previous professional life, one of the authors (sfb) served as a missileer in the United States Air Force. On a routine basis, the guidance set aboard an assigned missile was updated with critical data to insure the missile would serve its intended mission. Maintenance crews from a nearby support base would transport information tapes out to the missile site where the onboard missile guidance set was updated. A CRC checksum was generated on the information tapes before they left the support base. After the information was loaded from the tapes to the missile guidance set, a CRC checksum was performed. If the checksum generated by the missile guidance set matched the checksum generated at the support base, the missile was designated as properly updated.

This scenario illustrates the application and importance of using a Cyclic Redundancy Check (CRC) to maintain data integrity. This technique is often performed to ensure the integrity of transmitted or stored data.

The basic concept behind generating a CRC checksum is binary division. The basic operation of division can be defined as [AN370]:

$$\text{Dividend} / \text{divisor} = \text{quotient} + \text{remainder}$$

The block of data to be protected via the checksum is considered the dividend. The dividend is divided by a pre-selected CRC polynomial which serves as the divisor. At the completion of the division operation, a quotient and a remainder result. The remainder of the operation serves as the CRC checksum.

Generation of a checksum is based on the concept that when a given block of data is divided by a specific polynomial with the division hardware initialized with the same value (seed), the same checksum will result every time the operation is performed. Similarly, if the input data is different, the polynomial is changed, or the division hardware is seeded with a different initial value, a different checksum will result. A number of common polynomials have been developed to support CRC checksum generation. Two common ones include [SLAU356A, 2015]:

- CRC16-CCITT defined as $f(x) = X^{15} + X^{12} + X^5 + 1$
- CRC32-IS3309 defined as

$$f(x) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1 \quad (10.1)$$

A linear feedback shift register (LFSR) is used to generate the checksum. The polynomial divisor chosen to generate the checksum specifies the hardware connection for the LFSR. For example, the LFSR configuration for the CRC16-CCITT polynomial is shown in Figure 10.2. Note how the polynomial terms specify the output connections of certain flip-flops within the LFSR. To generate the checksum, the LFSR is initially configured to the seed value. The data block is fed in as a serial data stream. The resulting remainder is used as the checksum and appended to the original data block for transmission.

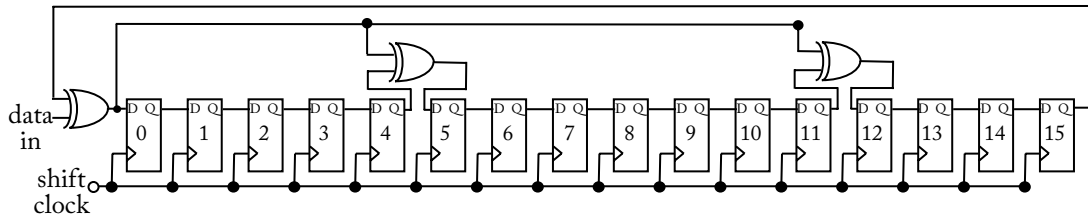


Figure 10.2: CRC16-CCITT polynomial and LFSR configuration [SLAU356A, 2015].

10.3.1 MSP432 CRC32 MODULE

A block diagram for the MSP432 CRC32 module is provided in Figure 10.3a. The MSP432 CRC32 module is quite flexible. It allows for 16- or 32-bit CRC generation. It also provides for data bit 0 being the MSB or LSB. This allows for compatibility with both modern and legacy hardware. Also, to speed up the calculation of the CRC checksum, the linear feedback shift register operation is implemented with an equivalent XOR gate combinational logic tree [SLAU356A, 2015].

The UML activity diagram for the CRC operation is provided in Figure 10.3b. The operation is quite straightforward. The CRC polynomial is provided to the CRC32 system along with the seed via the CRC16INIRES (CRC32INIRES) register. The data block to perform the checksum operation is fed into the CRC16DI (CRC32DI) register. The CRC checksum operation is performed and the checksum is available at the CRC16INIRES (CRC32INIRES) register [SLAU356A, 2015].

10.3.2 CRC32 REGISTERS

The CRC32 system is supported by a set of the following registers [SLAU356A, 2015].

- CRC32DI CRC32 Data Input Low
- CRC32DIRB CRC32 Data In Reverse Low
- CRC32INIRES_LO CRC32 Initialization and Result Low

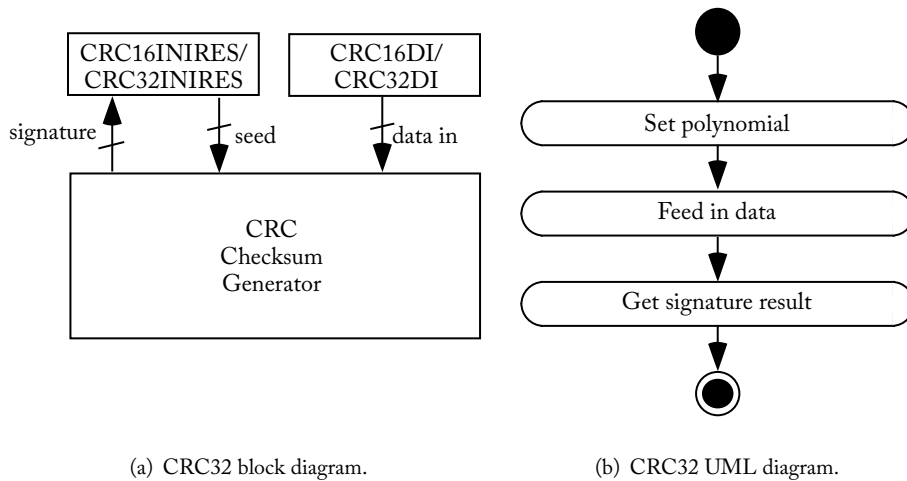


Figure 10.3: MSP432 CRC32 module.

- CRC32INIRES_HI CRC32 Initialization and Result High
- CRC32RESR_LO CRC32 Result Reverse Low
- CRC32RESR_HI CRC32 Result Reverse High
- CRC16DI CRC16 Data Input Low
- CRC16DIRB CRC16 Data In Reverse Low
- CRC16INIRES CRC16 Initial and Result
- CRC16RESR CRC16 Result Reverse

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

10.3.3 API SUPPORT

Texas Instruments provides extensive MSP432 CRC32 support through a series of Application Program Interfaces (APIs). Provided below is a list of CRC32 APIs. Details on API settings are provided in *MSP432 Peripheral Driver Library User's Guide* [DriverLib] and will not be repeated here.

- uint32_t CRC32_getResult(uint_fast8_t crcType)
- uint32_t CRC32_getResultReversed(uint_fast8_t crcType)

- void CRC32_set16BitData(uint16_t dataIn, uint_fast8_t crcType)
- void CRC32_set16BitDataReversed(uint16_t dataIn, uint_fast8_t crcType)
- void CRC32_set32BitData(uint32_t dataIn)
- void CRC32_set32BitDataReversed(uint32_t dataIn)
- void CRC32_set8BitData(uint8_t dataIn, uint_fast8_t crcType)
- void CRC32_set8BitDataReversed(uint8_t dataIn, uint_fast8_t crcType)
- void CRC32_setSeed(uint32_t seed, uint_fast8_t crcType)

Example: In this example a data array is fed into the CRC32 module and the 32-bit checksum is retrieved [[DriverLib](#)].

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
//  notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
//  notice, this list of conditions and the following disclaimer in the
//  documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,

```

```

//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//In the following very simple code example, an array of data is fed into
//the CRC32 module and the 32-bit calculation is retrieved.
//*****

//Statics:
static const uint8_t myData[15] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                  12, 13, 14, 15};

int main(void)
{
volatile uint32_t hwCalculatedCRC;
uint32_t ii;

```

```

//Halting the Watchdog
MAP_WDT_A_holdTimer();

//Setting the polynomial and feeding in the data
MAP_CRC32_setSeed(CRC32_POLY, CRC32_MODE);

for(ii=0;ii<15;ii++)
    MAP_CRC32_set8BitData(myData[ii], CRC32_MODE);

//Getting the result from the hardware module
hwCalculatedCRC = MAP_CRC32_getResult(CRC32_MODE);

//Pause for the debugger
__no_operation();
}

//*****

```

Example: In this example a data array of sixteen 16-bit values are sent to the CRC32 module. Also, software-based CRC-CCIT-BR algorithm is used to calculate the CRC checksum on the same block of data. The outputs of both methods are compared. If the two checksums agree, an LED is illuminated on P1.0.

```

//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//

```

```

//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****

//*****
//MSP432P401 Demo - CRC16, Compare CRC output with software-based
//algorithm

```

```

//
//Description: An array of 16 random 16-bit values are moved through the
//CRC module, as well as a software-based CRC-CCIT-BR algorithm.
Due to
//the fact that the software-based algorithm handles 8-bit inputs only,
//the 16-bit words are broken into 2 8-bit words before being run through
//(lower byte first). The outputs of both methods are then compared to
//ensure that the operation of the CRC module is consistent with the
//expected outcome.
If the values of each output are equal, set P1.0,
//else reset.
//
//MCLK = SMCLK = default DCO-1MHz
//
//      MSP432p401rpz
//      -----
//      /\|\|          |
//      | |            |
//      --|RST         |
//      |              |
//      |              P1.0|----> LED
//
//Wei Zhao
//Texas Instruments Inc.
//Jun 2014
//Built with Code Composer Studio V6.0
//*****

#include "msp.h"

const uint16_t CRC16_Init = 0xFFFF;

const uint16_t CRC16_Input[] = {
    0x0fc0, 0x1096, 0x5042, 0x0010,          //16 random 16-bit numbers
    0x7ff7, 0xf86a, 0xb58e, 0x7651,        //these numbers can be
    0x8b88, 0x0679, 0x0123, 0x9599,        //modified if desired
    0xc58c, 0xd1e2, 0xe144, 0xb691
};

```


460 10. MSP432 SYSTEM INTEGRITY

```
uint16_t CRC16_Result; //Holds results obtained through the CRC16 module
uint16_t SW_Results;   //Holds results obtained through SW

// Software Algorithm Function Declaration
uint16_t CCITT_Update(uint16_t, uint16_t);

int main(void)
{
uint16_t i;

WDTCTL = WDTPW | WDTHOLD;           //Stop WDT

//Configure GPIO
P1OUT &= ~BIT0;                     //Clear LED to start
P1DIR |= BIT0;                      //P1.0 Output

//First, use the CRC16 hardware module to calculate the CRC...
CRC16INIRES = CRC16_Init;           //Init CRC16 HW module

for(i=0;i<16;i++)
{
//Input random values into CRC Hardware
CRC16DIRB = CRC16_Input[i];         //Input data in CRC
__no_operation();
}

CRC16_Result = CRC16INIRES;          //Save results(per CRC-CCITT
//standard)

//Now use a software routine to calculate the CRC...
SW_Results = CRC16_Init;             //Init SW CRC

for(i=0;i<16;i++)
{
//First input lower byte
SW_Results = CCITT_Update(SW_Results, CRC16_Input[i] & 0xFF);

//Then input upper byte
SW_Results = CCITT_Update(SW_Results, (CRC16_Input[i] >> 8) & 0xFF);
```

```

}

//Compare data output results
if(CRC16_Result==SW_Results)           //if data is equal
    P1OUT |= BIT0;                      //set the LED
else
    P1OUT &= ~BIT0;                    //if not, clear LED

while(1);                              //infinite loop
}

//*****
//Software algorithm - CCITT CRC16 code
//*****

uint16_t CCITT_Update(uint16_t init, uint16_t input)
{
uint16_t CCITT = (unsigned char) (init >> 8) | (init << 8);
CCITT ^= input;
CCITT ^= (unsigned char) (CCITT & 0xFF) >> 4;
CCITT ^= (CCITT << 8) << 4;
CCITT ^= ((CCITT & 0xFF) << 4) << 1;
return CCITT;
}

//*****

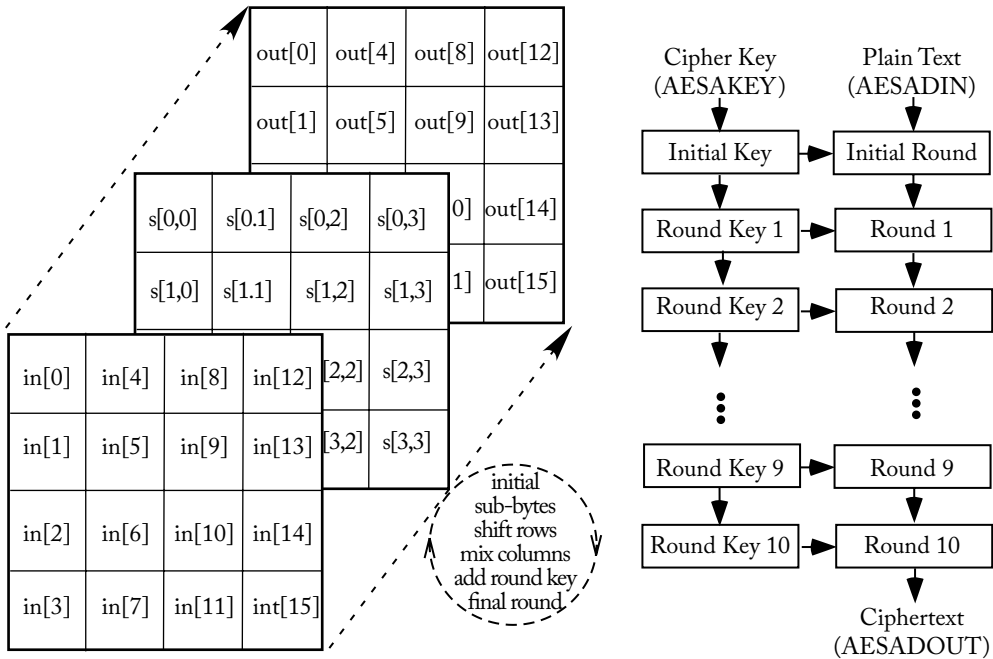
```

10.4 AES256 ACCELERATOR MODULE

The MSP432 is equipped with the AES256 Accelerator Module that allows encryption and decryption of data using the Rijndael cryptographic algorithm. The algorithm allows the encryption of a 128-bit plain text data block into a corresponding size cipher text block. The data may then be transmitted in an encrypted format and decrypted using a similar algorithm at the receiving end [FIPS, 2001, SLAU356A, 2015].

The data algorithm uses a 128-, 192-, or 256-bit cipher key to encrypt the plain text data block. The length of the cipher key determines the number of rounds (10, 12, or 14, respectively) of encryption performed on the plain text data to transform it into the cipher text block. The basic encryption process is shown in Figures 10.4a and 10.4b. The plain text 128-bit block is formatted into a state block. The state block then goes through a series of transformation rounds including an initial round, the sub-byte round, the shift rows round, the mix columns round, the add key

round, and the final round to encrypt the data. As shown in Figure 10.4b, a specific round key is derived from the original cipher key and used in a given round [FIPS, 2001, SLAU356A, 2015].



(a) AES256 algorithm.

(b) AES256 encryption process with a 128-bit key.

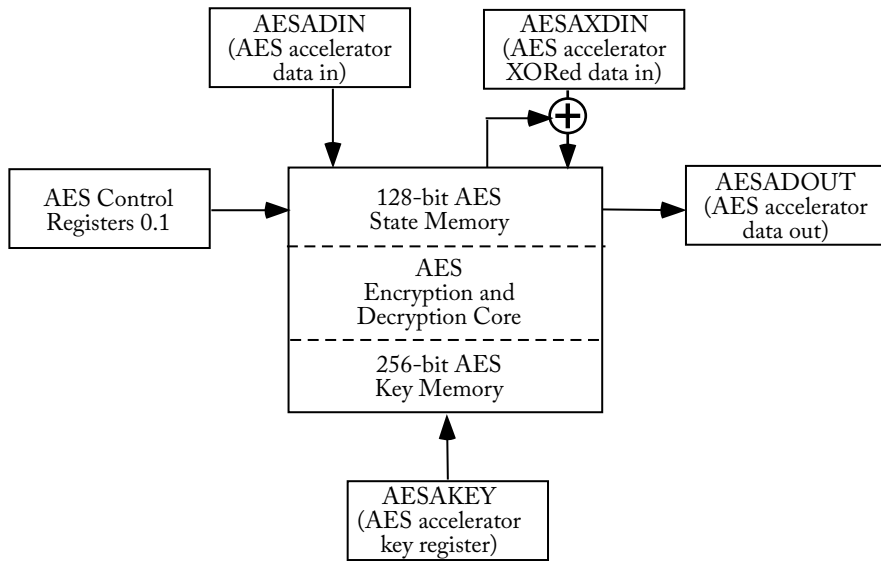
Figure 10.4: AES256 encryption process. (a) AES256 algorithm and (b) AES256 encryption process with 128-bit key. (Continues.)

A block diagram of the MSP432 AES256 Accelerator Module is provided in Figure 10.4c. Input plain text data for encryption may be stored in register AESADIN or AESAXDIN. Data input to AESAXDIN is XORed with the current state value. The operation of the AES256 Accelerator module is controlled by AES Control Registers 0 and 1. The AES key is provided to the AESAKEY register. The encrypted cipher text is output to the AESAOUT Register [SLAU356A, 2015].

10.4.1 REGISTERS

The AES256 system is supported by a complement of registers including [SLAU356A, 2015]:

- AESACTL0 AES accelerator control register 0



(c) AES256 block diagram.

Figure 10.4: (Continued.) AES256 encryption process. (c) MSP432 AES256 block diagram [SLAU356A, 2015].

- AESACTL1 AES accelerator control register 1
- AESASTAT AES accelerator status register
- AESAKEY AES accelerator key register
- AESADIN AES accelerator data in register
- AESADOUT AES accelerator data out register
- AESAXDIN AES accelerator XORed data in register
- AESAXIN AES accelerator XORed data in register (no trigger)

Details of specific register and bits settings are contained in *MSP432P4xx Family Technical Reference Manual* [SLAU356A, 2015] and will not be repeated here.

10.4.2 API SUPPORT

Texas Instruments provides extensive MSP432 AES256 support through a series of Application Program Interfaces (APIs). Provided below is a list of AES256 APIs. Details on API settings are

provided in *MSP432 Peripheral Driver Library User's Guide* [[DriverLib](#)] and will not be repeated here.

- void AES256_clearErrorFlag(uint32_t moduleInstance)
- void AES256_clearInterruptFlag(uint32_t moduleInstance)
- void AES256_decryptData(uint32_t moduleInstance, const uint8_t *data, uint8_t *decryptedData)
- void AES256_disableInterrupt(uint32_t moduleInstance)
- void AES256_enableInterrupt(uint32_t moduleInstance)
- void AES256_encryptData(uint32_t moduleInstance, const uint8_t *data, uint8_t *encryptedData)
- bool AES256_getDataOut(uint32_t moduleInstance, uint8_t *outputData)
- uint32_t AES256_getErrorFlagStatus(uint32_t moduleInstance)
- uint32_t AES256_getInterruptFlagStatus(uint32_t moduleInstance)
- uint32_t AES256_getInterruptStatus(uint32_t moduleInstance)
- bool AES256_isBusy(uint32_t moduleInstance)
- void AES256_registerInterrupt(uint32_t moduleInstance, void(*intHandler)(void))
- void AES256_reset(uint32_t moduleInstance)
- bool AES256_setCipherKey(uint32_t moduleInstance, const uint8_t *cipherKey, uint_fast16_t keyLength)
- bool AES256_setDecipherKey(uint32_t moduleInstance, const uint8_t *cipherKey, uint_fast16_t keyLength)
- void AES256_startDecryptData(uint32_t moduleInstance, const uint8_t *data)
- void AES256_startEncryptData(uint32_t moduleInstance, const uint8_t *data)
- bool AES256_startSetDecipherKey(uint32_t moduleInstance, const uint8_t *cipherKey, uint_fast16_t keyLength)
- void AES256_unregisterInterrupt(uint32_t moduleInstance)

Example: In this example the AES256 module is used to encrypt/decrypt data using a cipher key and APIs from DriverLib.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
//*****
//
//
//                               MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
```

```

//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//Below is a simple code example of how to encrypt/decrypt data using a
//cipher key with the AES256 module
//*****

int main(void)
{
//Load a cipher key to module
MAP_AES256_setCipherKey(AES256_MODULE, CipherKey,
                        AES256_KEYLENGTH_256BIT);

//Encrypt data with preloaded cipher key
MAP_AES256_encryptData(AES256_MODULE, Data, DataAESencrypted);

//Load a decipher key to module
MAP_AES256_setDecipherKey(AES256_MODULE, CipherKey,
                           AES256_KEYLENGTH_256BIT);

//Decrypt data with keys that were generated during encryption - takes
//214 MCLK cycles.
This function will generate all round keys needed for
//
decryption first and then the encryption process starts
MAP_AES256_decryptData(AES256_MODULE, DataAESencrypted,
                        DataAESdecrypted);
}

```

```
//*****
```

Example: In this example the AES256 module is used for encryption and decryption. The original plain text data is encrypted and then decrypted. The results are compared and if they agree an LED on P1.0 is illuminated. A UML activity diagram for the example is provided in Figure 10.5.

```
//*****
//    MSP432 DriverLib - v2_20_00_08
//*****
//
//--COPYRIGHT--,BSD_EX
//Copyright (c) 2013, Texas Instruments Incorporated
//All rights reserved.
//
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//- Redistributions of source code must retain the above copyright
//  notice, this list of conditions and the following disclaimer.
//- Redistributions in binary form must reproduce the above copyright
//  notice, this list of conditions and the following disclaimer in the
//  documentation and/or other materials provided with the distribution.
//
//Neither the name of Texas Instruments Incorporated nor the names of
//its contributors may be used to endorse or promote products derived
//from this software without specific prior written permission.
//
//THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
//"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
//LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
//FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
//COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
//INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
//BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
//OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
//ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
//TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
//USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
//DAMAGE.
// --COPYRIGHT--
```

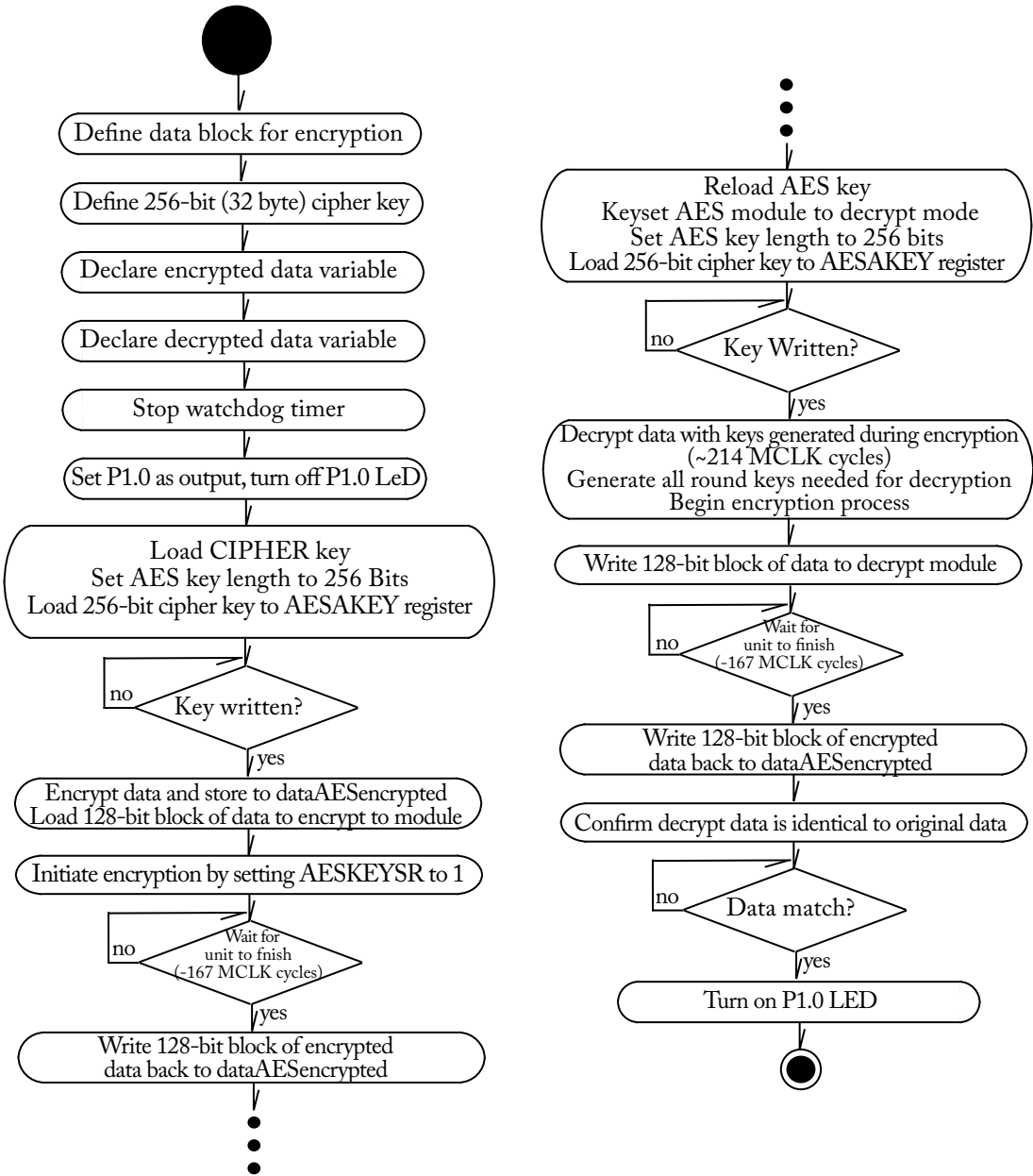



Figure 10.5: AES256 encryption/decryption process [SLAU356A, 2015].

```

//*****
//
//          MSP432 CODE EXAMPLE DISCLAIMER
//
//MSP432 code examples are self-contained low-level programs that
//typically demonstrate a single peripheral function or device feature
//in a highly concise manner. For this the code may rely on the device's
//power-on default register values and settings such as the clock
//configuration and care must be taken when combining code from several
//examples to avoid potential side effects.
Also see:
// http://www.ti.com/tool/mspdriverlib
//for an API functional library and:
// https://dev.ti.com/pinmux/
//for a GUI approach to peripheral configuration.
//
// --/COPYRIGHT/--
//*****
//MSP432P401 Demo - AES256 Encryption & Decryption
//
//Description: This example shows a simple example of encryption and
//decryption using the AES256 module.
//
//          MSP432p401rpz
//          -----
//          /|\|          |
//          | |          |
//          --|RST       |
//          |           |
//          |           P1.0|-->LED
//
//Key: 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
//Plaintext: 00112233445566778899aabbccddeeff
//Ciphertext: 8ea2b7ca516745bfeafc49904b496089
//
//Dung Dang
//Texas Instruments Inc.
//Nov 2013
//Built with Code Composer Studio V6.0

```

```

//*****

#include "msp.h"
#include <stdint.h>

uint8_t Data[16] =
{0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff };

uint8_t CipherKey[32] =
{0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
 0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f };

uint8_t DataAESencrypted[16];          //Encrypted data
uint8_t DataAESdecrypted[16];         //Decrypted data

int main(void)
{
volatile uint32_t i;
uint16_t sCipherKey, tempVariable;

WDTCTL = WDTPW | WDTHOLD;             //Stop WDT

P1DIR |= BIT0;                        //P1.0 set as output
P1OUT &= ~BIT0;                       //Turn off P1.0 LED

//Step 1: Load cipher key
AESACTLO &= ~AESOP_3;                 //Set AES module to encrypt mode

//Set AES key length to 256 bits
AESACTLO = AESACTLO & (~(AESKL_1 + AESKL_2)) | AESKL__256BIT;

//Load 256-bit cipher key to the AESAKEY register
for(i = 0; i < 256/8; i = i + 2)
{
//Concatenate 2 8-bit blocks into one 16-bit block

```

```

sCipherKey =(uint16_t)(CipherKey[i]);
sCipherKey = sCipherKey |((uint16_t)(CipherKey[i + 1]) << 8);

//Load 16-bit key block to AESAKEY register
AESAKEY = sCipherKey;
}

//Wait until key is written
while((AESASTAT & AESKEYWR ) == 0);

//Step 2: Encrypt data and store to DataAESencrypted
//Load 128-bit block of data to encrypt to module
for(i = 0; i < 16; i = i + 2)
{
//Concatenate 2 8-bit blocks into one 16-bit block
tempVariable =(uint16_t)(Data[i]);
tempVariable = tempVariable |((uint16_t)(Data[i + 1]) << 8);

//Load 16-bit key block to AESADIn register
AESADIN = tempVariable;
}

//Initiate encryption by setting AESKEYWR to 1
AESASTAT |= AESKEYWR;

//Wait unit finished ~167 MCLK
while( AESASTAT & AESBUSY );

//Write 128-bit block of encrypted data back to DataAESencrypted
for(i = 0; i < 16; i = i + 2)
{
tempVariable = AESADOUT;
DataAESencrypted[i] = (uint8_t)tempVariable;
DataAESencrypted[i+1] = (uint8_t)(tempVariable >> 8);
}

//Step 3: Reload AES key
//Set AES module to decrypt mode

```

```

AESACTLO |= AESOP_1;

//Set AES key length to 256 bits
AESACTLO = AESACTLO & ~(AESKL_1 + AESKL_2) | AESKL__256BIT;

//Load 256-bit cipher key to the AESAKEY register
for(i = 0; i < 256/8; i = i + 2)
{
    //Concatenate 2 8-bit blocks into one 16-bit block
    sCipherKey =(uint16_t)(CipherKey[i]);
    sCipherKey = sCipherKey |((uint16_t)(CipherKey[i + 1]) << 8);

    //Load 16-bit key block to AESAKEY register
    AESAKEY = sCipherKey;
}

//Wait until key is written
while((AESASTAT & AESKEYWR ) == 0);

//Step 4: Decrypt data with keys that were generated during
//encryption takes 214 MCLK. This function will generate all round
//keys needed for decryption first and then the encryption process
//starts.

//Write 128-bit block of data to decrypt to module
for(i = 0; i < 16; i = i + 2)
{
    tempVariable = (uint16_t) (DataAESencrypted[i + 1] << 8);
    tempVariable = tempVariable | ((uint16_t) (DataAESencrypted[i]));
    AESADIN = tempVariable;
}

//Wait until finished ~167 MCLK
while(AESASTAT & AESBUSY);

//Write 128-bit block of encrypted data back to DataAESdecrypted
for(i = 0; i < 16; i = i + 2)
{

```

```

tempVariable = AESADOUT;
DataAESdecrypted[i] = (uint8_t)tempVariable;
DataAESdecrypted[i+1] =(uint8_t)(tempVariable >> 8);
}

//Step 4: Confirm decrypted data is identical to original data
for(i = 0; i < 16; i ++)
    if(DataAESdecrypted[i]!= Data[i])
        while(1); //Set breakpoint here for error

P1DIR |= BIT0;
P1OUT |= BIT0; //Turn on P1.0 LED = success
while(1);
}

//*****

```

10.5 LABORATORY EXERCISE: AES256

Develop an algorithm to encode a plain text block of data with the AES256 system, transmit the cipher text to another microcontroller, and then decrypt back to plain text at the receiving microcontroller.

10.6 SUMMARY

This chapter contained essential information about how to maintain the integrity of a microcontroller-based system. The chapter began with a discussion on electromagnetic interference (EMI), also known as noise. Design practices to minimize EMI were then discussed. The second section of the chapter discussed the concept of the Cyclic Redundancy Check or CRC. The final section covered the MSP432 advanced encryption standard module, the AES256.

10.7 REFERENCES AND FURTHER READING

Barrett, S. and Pack, D. 2004. *Embedded Systems: Design and Applications with the 68HC12 and HCS12*, Upper Saddle River, NJ, Prentice Hall. 450

DriverLib 454, 455, 464

Federal Information Processing Standards Publication 197 (FIPS-197). November 26, 2001. 461, 462

Microchip CRC Generating and Checking (AN370) Microchip, 2000. 452

MSP432 Peripheral Driver Library User's Guide. Texas Instruments, 2015.

MSP432P4xx Family Technical Reference Manual (SLAU356A). Texas Instruments, 2015. 452, 453, 454, 461, 462, 463, 468

Noise Reduction Techniques for Microcontroller-Based Systems (AN1705/D). Freescale Semiconductor, 2004. 450, 451

Understanding and Eliminating EMI in Microcontroller Applications (COP888). Texas Instruments, 1996. 450, 451

10.8 CHAPTER PROBLEMS

Fundamental

1. Describe sources of EMI.
2. Describe EMI coupling mechanisms.
3. Describe three strategies to combat EMI.
4. Describe specific techniques to combat EMI.
5. Describe defensive programming techniques.

Advanced

1. Sketch a UML activity diagram for the CRC algorithm.
2. What is the purpose of generating a CRC checksum?
3. What does a correct checksum indicate? An incorrect one?
4. Research common CRC polynomials. Sketch the corresponding LFSR for each polynomial.
5. What is the purpose of the AES256 subsystem?

Challenging

1. What are the advantages and disadvantages of using different encryption key lengths with the AES256?
2. Sketch a UML activity diagram for the AES256 encryption algorithm.

System Level Design

Objectives: After reading this chapter, the reader should be able to:

- define an embedded system;
- list multiple aspects related to the design of an embedded system;
- provide a step-by-step approach to design an embedded system;
- discuss design tools and practices related to embedded systems design;
- discuss the importance of system testing;
- apply embedded system design practices in the prototype of a MSP432-based system with several subsystems;
- develop a detailed design for a weather station including hardware layout and interface, structure chart, UML activity diagrams, and an algorithm coded in Energia;
- develop a detailed design for a submersible remotely operated vehicle (ROV) including hardware layout and interface, structure chart, UML activity diagrams, and an algorithm coded in Energia; and
- develop a detailed design for a four wheel drive (4WD) mountain maze navigating robot including hardware layout and interface, structure chart, UML activity diagrams, and an algorithm coded in Energia.

11.1 OVERVIEW

This chapter presents a step-by-step, methodical approach toward designing advanced embedded systems. We begin with a definition of an embedded system. We then explore the process of how to successfully (and with low stress) develop an embedded system prototype that meets established requirements. The overview of embedded system design techniques was adapted with permission from earlier Morgan & Claypool publications and several projects have been adapted for the MSP432 with permission. We also emphasize good testing techniques. We conclude the chapter with several extended examples. The examples illustrate the embedded system design process in the development and prototype of a weather station, a submersible remotely operated vehicle (ROV), and a four-wheel drive (4WD) mountain maze navigating robot.

11.2 WHAT IS AN EMBEDDED SYSTEM?

An embedded system is typically designed for a specific task. It contains a processor to collect system inputs and generate system outputs. The link between system inputs and outputs is provided by a coded algorithm stored within the processor's resident memory. What makes embedded systems design so challenging and interesting is the design must also provide for proper electrical interface for the input and output devices, potentially limited on-chip resources, human interface concepts, the operating environment of the system, cost analysis, related standards, and manufacturing aspects [Anderson, 2008]. Through careful application of this material the reader will be able to design and prototype embedded systems based on MSP432.

11.3 EMBEDDED SYSTEM DESIGN PROCESS

There are many available formal design processes. We concentrate on the steps that are common to most. We purposefully avoid formal terminology of a specific approach, and instead, concentrate on the activities that are accomplished during the development of a system prototype. The design process we describe is illustrated in Figure 11.1 using a Unified Modeling Language (UML) activity diagram. We discuss the UML activity diagrams later in this section.

11.3.1 PROJECT DESCRIPTION

The goal of the project description step is to determine what the system is ultimately supposed to do. Questions to raise and answer during this step include, but are not limited to, the following.

- What is the system supposed to do?
- Where will it be operating and under what conditions?
- Are there any restrictions placed on the system design?

To answer these questions, the designer interacts with the client to ensure clear agreement on what is to be done. The establishment of clear, definable system requirements may need considerable interaction between the designer and the client. It is essential that both parties agree on system requirements before proceeding further in the design process. The final result of this step is a detailed listing of system requirements and related specifications.

11.3.2 BACKGROUND RESEARCH

Once a detailed list of requirements has been established, the next step is to perform background research related to the design. In this step, the designer will ensure they understand all requirements and features required by the project. This will again involve interaction between the designer and the client. The designer will also investigate applicable codes, guidelines, protocols, and standards related to the project. This is also a good time to start thinking about the interface between different portions of the input and output devices peripherally connected to the processor. The

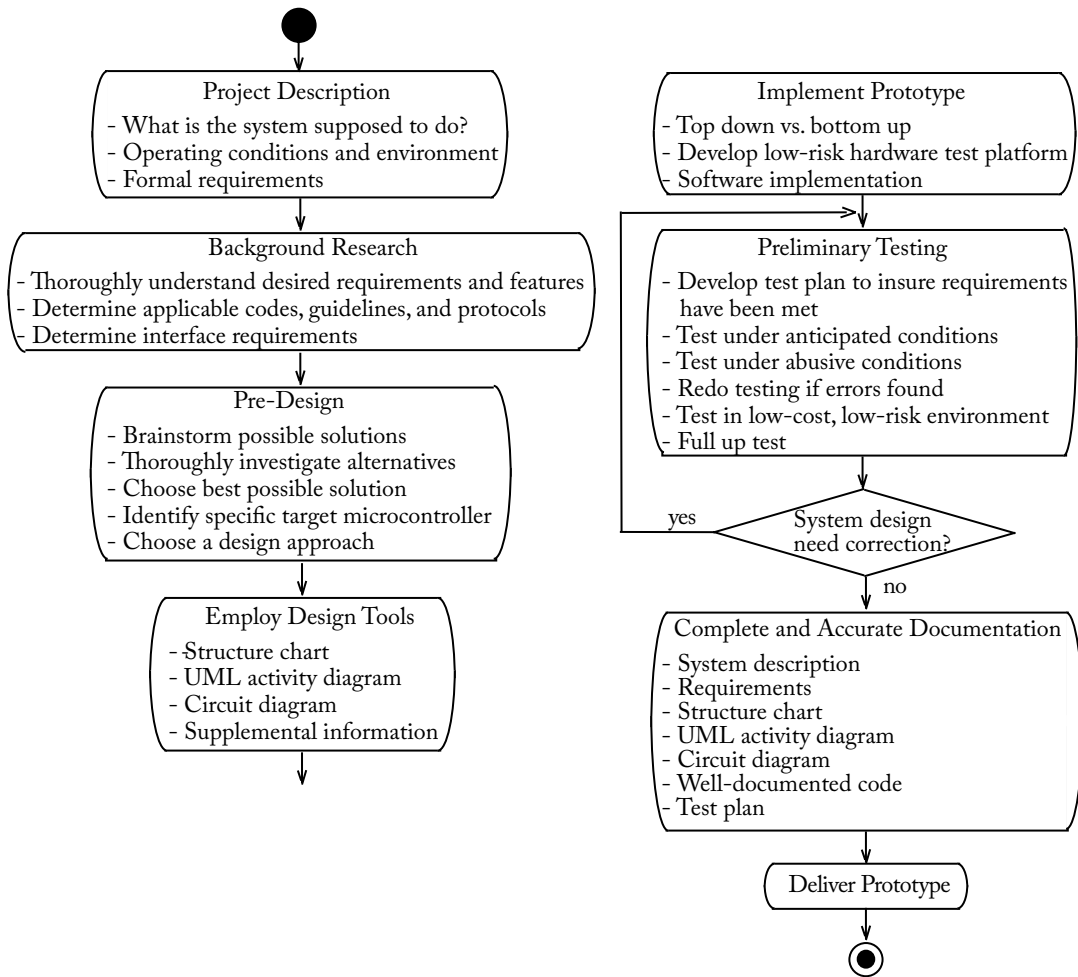


Figure 11.1: Embedded system design process.

ultimate objective of this step is to have a thorough understanding of the project requirements, related project aspects, and any interface challenges within the project.

11.3.3 PRE-DESIGN

The goal of the pre-design step is to convert a thorough understanding of the project into possible design alternatives. Brainstorming is an effective tool in this step. Here, a list of alternatives is developed. Since an embedded system involves hardware and/or software, the designer can investigate whether requirements could be met with a hardware only solution or some combination

of hardware and software. Generally speaking, a hardware only solution executes faster; however, the design is fixed once fielded. On the other hand, a software implementation provides flexibility but a slower execution speed. Most embedded design solutions will use a combination of both hardware and software to capitalize on the inherent advantages of each.

Once a design alternative has been selected, the general partition between hardware and software can be determined. It is also an appropriate time to select a specific hardware device to implement the prototype design. If a technology has been chosen, it is now time to select a specific processor. This is accomplished by answering the following questions.

- What processor systems or features (i.e., ADC, PWM, timer, etc.) are required by the design?
- What is the power requirement for the electronic system?
- How many input and output pins are required by the design?
- What type of memory components are required?
- What is the maximum anticipated operating speed of the processor expected to be?

Due to the variety of onboard systems, clock speed, and low cost; the MSP432 may be used in a wide array of applications typically held by microcontrollers and advanced processors.

11.3.4 DESIGN

With a clear view of system requirements and features, a general partition determined between hardware and software, and a specific processor chosen, it is now time to tackle the actual design. It is important to follow a systematic and disciplined approach to design. This will allow for low stress development of a documented design solution that meets requirements. In the design step, several tools are employed to ease the design process. They include:

- employing a top-down design, bottom up implementation approach;
- using a structure chart to assist in partitioning the system;
- using a Unified Modeling Language (UML) activity diagram to work out program flow; and
- developing a detailed circuit diagram of the entire system.

Let's take a closer look at each of these. The information provided here is an abbreviated version of the one provided in "Microcontrollers Fundamentals for Engineers and Scientists." The interested reader is referred there for additional details and an in-depth example [Barrett and Pack, 2005].

Top-down design, bottom-up implementation. An effective tool to start partitioning the design is based on the techniques of top-down design, bottom-up implementation. In this approach, you start with the overall system and begin to partition it into subsystems. At this point of the design, you are not concerned with how the design will be accomplished but how the different pieces of the project will fit together. A handy tool to use at this design stage is the structure chart. The structure chart shows how the hierarchy of system hardware and software components will interact and interface with one another. You should continue partitioning system activity until each subsystem in the structure chart has a single definable function. Directional arrows are used to indicate data flow in and out of a function.

UML Activity Diagram. Once the system has been partitioned into pieces, the next step is to work out the details of the operation of each subsystem previously identified. Rather than beginning to code each subsystem as a function, work out the information and control flow of each subsystem using another design tool: the Unified Modeling Language (UML) activity diagram. The activity diagram is simply a UML compliant flow chart. UML is a standardized method of documenting systems. The activity diagram is one of the many tools available from UML to document system design and operation. The basic symbols used in a UML activity diagram for a processor based system are provided in Figure 11.2 [Fowler, 2000].

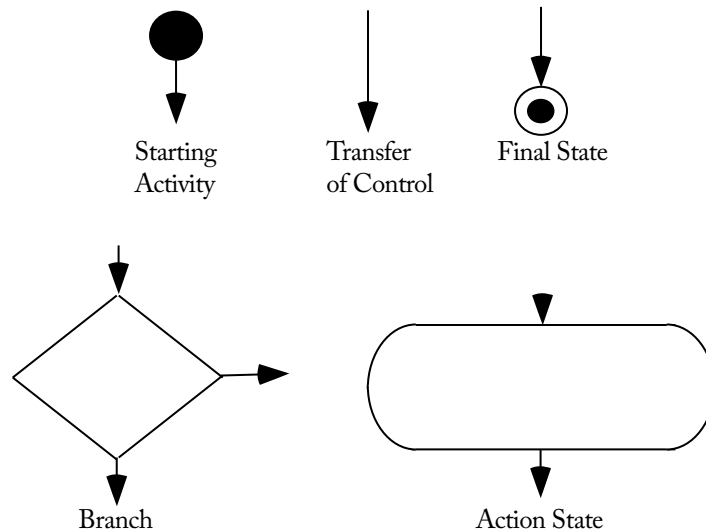


Figure 11.2: UML activity diagram symbols. Adapted from [Fowler, 2000].

To develop the UML activity diagram for the system, we can use a top-down, bottom-up, or a hybrid approach. In the top-down approach, we begin by modeling the overall flow of the algorithm from a high level. If we choose to use the bottom-up approach, we would begin at the

bottom of the structure chart and choose a subsystem for flow modeling. The specific course of action chosen depends on project specifics. Often, a combination of both techniques, a hybrid approach, is used. You should work out all algorithm details at the UML activity diagram level prior to coding any software. If you cannot explain system operation at this higher level first, you have no business being down in the detail of developing the code. Therefore, the UML activity diagram should be of sufficient detail so you can code the algorithm directly from it [Dale and Lilly, 1995].

In the design step, a detailed circuit diagram of the entire system is developed. It will serve as a roadmap to implement the system. It is also a good idea at this point to investigate available design information relative to the project. This would include hardware design examples, software code examples, and application notes available from manufacturers. As before, use a subsystem approach to assemble the entire circuit. The basic building block interface circuits discussed in the previous chapter may be used to assemble the complete circuit. At the completion of this step, the prototype design is ready for implementation and testing.

11.3.5 IMPLEMENT PROTOTYPE

To successfully implement a prototype, an incremental approach should be followed. Again, the top-down design, bottom-up implementation provides a solid guide for system implementation. In an embedded system design involving both hardware and software, the hardware system including the processor should be assembled first. This provides the software the required signals to interact with. As the hardware prototype is assembled on a prototype board, each component is tested for proper operation as it is brought online. This allows the designer to pinpoint malfunctions as they occur.

Once the hardware prototype is assembled, coding may commence. It is important to note that on larger projects software and hardware may be developed concurrently. As before, software should be incrementally brought online. You may use a top-down, bottom-up, or hybrid approach depending on the nature of the software. The important point is to bring the software online incrementally such that issues can be identified and corrected early on.

It is highly recommended that low cost stand-in components be used when testing the software with the hardware components. For example, push buttons, potentiometers, and LEDs may be used as low-cost stand-in component simulators for expensive input instrumentation devices and expensive output devices such as motors. This allows you to ensure the software is properly operating before using it to control the actual components.

11.3.6 PRELIMINARY TESTING

To test the system, a detailed test plan must be developed. Tests should be developed to verify that the system meets all of its requirements and also intended system performance in an operational environment. The test plan should also include scenarios in which the system is used in an

unintended manner. As before, a top-down, bottom-up, or hybrid approach can be used to test the system. In a bottom-up approach individual units are tested first.

Once the test plan is completed, actual testing may commence. The results of each test should be carefully documented. As you go through the test plan, you will probably uncover a number of run-time errors in your algorithm. After you correct a run-time error, the entire test plan must be repeated. This ensures that the new fix does not have an unintended effect on another part of the system. Also, as you process through the test plan, you will probably think of other tests that were not included in the original test document. These tests should be added to the test plan. As you go through testing, realize your final system is only as good as the test plan that supports it!

Once testing is complete, you should accomplish another level of testing where you intentionally try to “jam up” the system. In other words, try to get your system to fail by trying combinations of inputs that were not part of the original design. A robust system should continue to operate correctly in this type of an abusive environment. It is imperative that you design robustness into your system. When testing on a low-cost simulator is complete, the entire test plan should be performed again with the actual system hardware. Once this is completed you should have a system that meets its requirements!

11.3.7 COMPLETE AND ACCURATE DOCUMENTATION

With testing complete, the system design should be thoroughly documented. Much of the documentation will have already been accomplished during system development. Documentation will include the system description, system requirements, the structure chart, the UML activity diagrams, documenting program flow, the test plan, results of the test plan, system schematics, and properly documented code. To properly document code, you should carefully comment all functions describing their operation, inputs, and outputs. Also, comments should be included within the body of the function describing key portions of the code. Enough detail should be provided such that code operation is obvious. It is also extremely helpful to provide variables and functions within your code names that describe their intended use.

You might think that comprehensive system documentation is not worth the time or effort to complete it. Complete documentation pays rich dividends when it is time to modify, repair, or update an existing system. Also, well-documented code may be often reused in other projects: a method for efficient and timely development of new systems.

In the next sections we provide detailed examples of the system design process for a weather station, a submersible robot, and a four wheel drive robot capable of navigating through a mountainous maze.

11.4 WEATHER STATION

In this project, we design a weather station to sense wind direction and ambient temperature. The wind direction will be displayed on LEDs arranged in a circular pattern. The wind direction and

temperature will also be transmitted serially via the SPI from the microcontroller to an MMC/SD flash memory card for data logging.

11.4.1 REQUIREMENTS

The requirements for this system include:

- design a weather station to sense wind direction and ambient temperature;
- wind direction should be displayed on LEDs arranged in a circular pattern; and
- wind direction and temperature should be transmitted serially from the microcontroller to an MMC/SD card for storage.

11.4.2 STRUCTURE CHART

To begin, the design process, a structure chart is used to partition the system into definable pieces. We employ a top-down design/bottom-up implementation approach. The structure chart for the weather station is provided in Figure 11.3. The three main microcontroller subsystems needed for this project are the SPI for serial communication to the MMC/SD card, the ADC14 system to convert the analog voltage from the LM34 temperature sensor and weather vane into digital signals, and the wind direction display. The system is partitioned until the lowest level of the structure chart contains “doable” pieces of hardware components or software functions. Data flow is shown on the structure chart as directed arrows.

11.4.3 CIRCUIT DIAGRAM

Analog sensors: The circuit diagram for the weather station is provided in Figure 11.4. The weather station is equipped with two input sensors: the LM34 to measure temperature and the weather vane to measure wind direction. Both of the sensors provide an analog output that is fed to the MSP432 on P5.2 (J2, pin 12) and P5.0 (J2, pin 13). The LM34 provides 10 mV output per degree Fahrenheit. The weather vane provides a voltage output from 0–3.3 VDC for different wind directions, as shown in Figure 11.4. The weather vane must be oriented to a known direction with the output voltage at this direction noted. We assume that 0 VDC corresponds to North.

Wind direction display: There are eight different LEDs to drive for the wind direction indicator. An interface circuit is required for each LED as shown in the figure.

MMC/SD flash memory card: The system also includes an MMC/SD flash memory card as a data logger. The card is interfaced to the MSP432 via the serial peripheral interface (SPI). An easy method to construct an interface between the MSP432 and the SD card is via an SD card reader and breakout board. For this example, the card reader available from 43oh.com was used. A 6-pin header was soldered to the breakout board J5 connector. The interface between the MSP432 and the 43oh.com breakout board is shown in Figure 11.4.

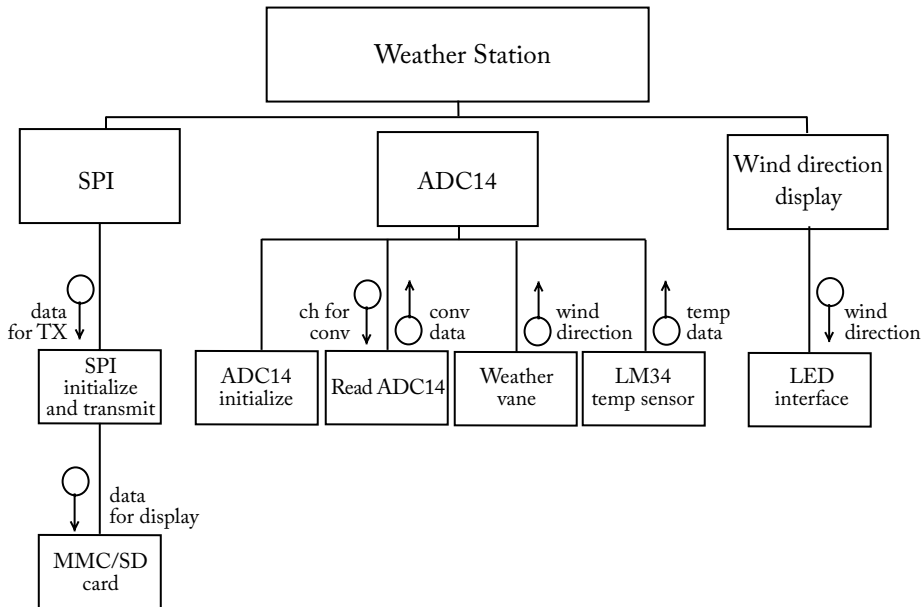


Figure 11.3: Weather station structure chart.

11.4.4 UML ACTIVITY DIAGRAMS

The UML activity diagram for the main program is provided in Figure 11.5. After initializing the subsystems, the program enters a continuous loop where temperature and wind direction are sensed and displayed on the LCD and the LED display. The sensed values are then transmitted via the SPI to the MMC/SD card. The system then enters a delay, which determines how often the temperature and wind direction parameters are updated.

11.4.5 MICROCONTROLLER CODE

For quick prototyping the first version of the code for this project is rendered in Energia. After initializing the system, the code continuously loops and reads temperature and wind direction data, displays the data to the LED array, and stores the data to the MMC/SD card. A delay should be inserted in the loop to determine how often the weather data should be collected. During development code status is sent to the serial monitor. Printing to the serial monitor is enabled with the variable “troubleshoot.”

```

//*****
//weather station
// - Equipped with Sparkfun weather meters (SEN-08942)

```

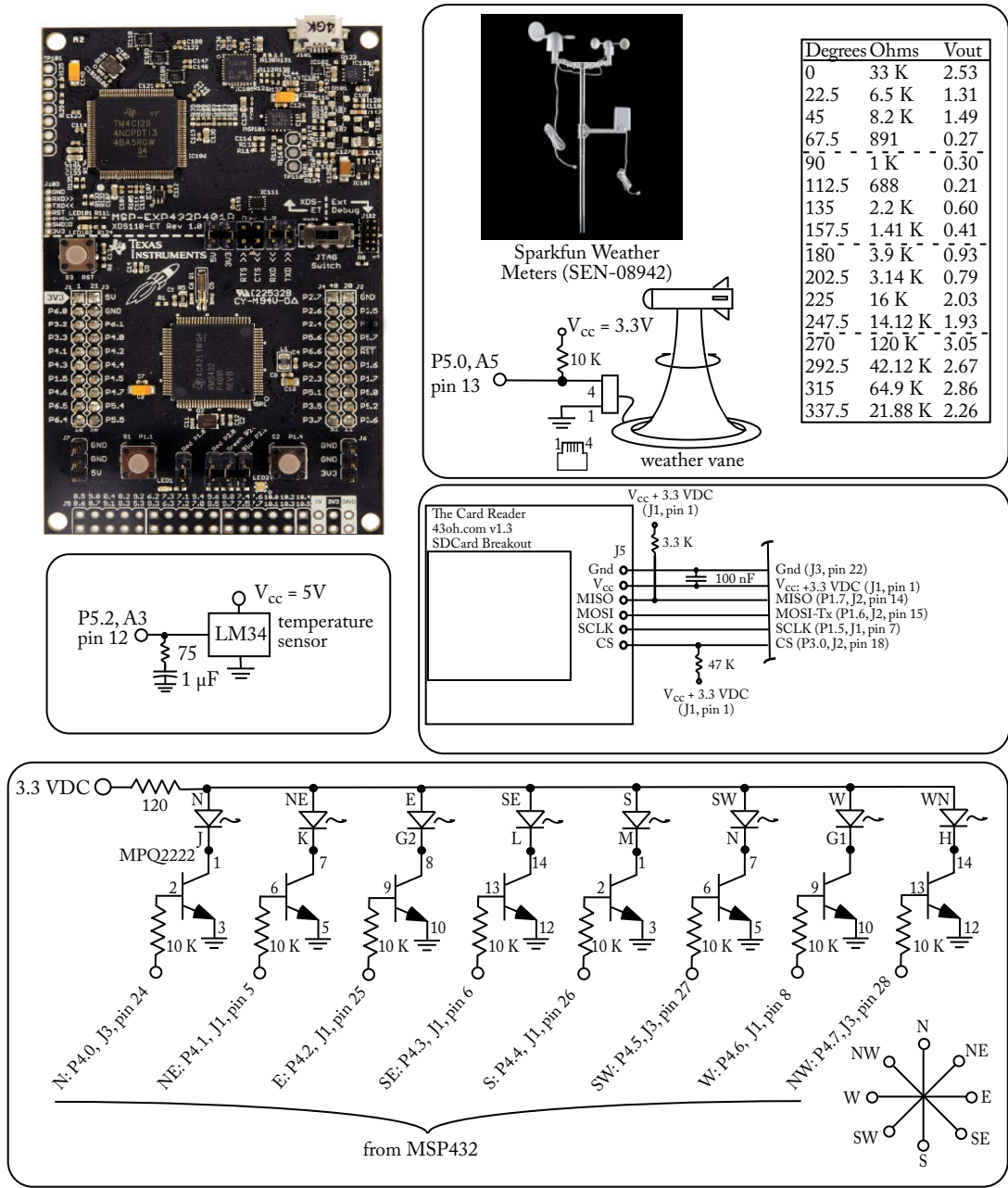



Figure 11.4: Circuit diagram for weather station.

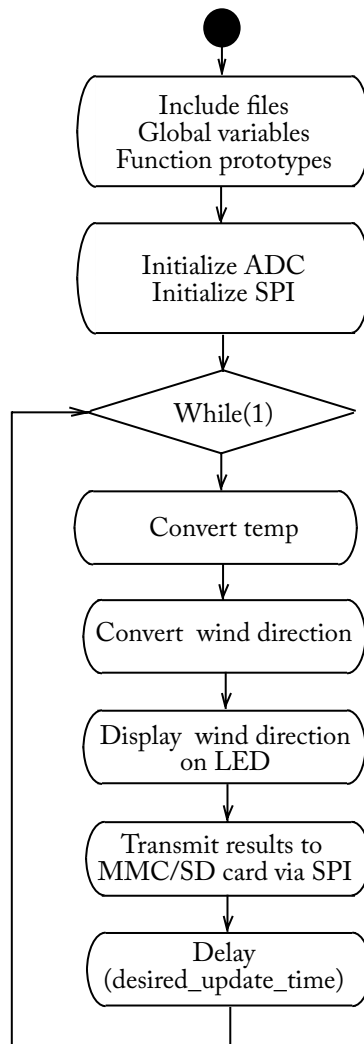


Figure 11.5: Weather station UML activity diagram.

```

// - rain gauge
// - anemometer
// - wind vane
//- SD card datalogger
// - Originally developed by Tom Igoe for the Arduino processor
//   24 November 2010.
// - Modified for use with the MSP432 LaunchPad by Martin Valencia
//   February 2016, Element 14 post "Interfacing SD Card with MSP432."
// - Interface details between SD card and MSP432 adapted from
//   "Interfacing an SD-Card to the LaunchPad - A Walkthrough
//   Tutorial by 43oh," December 21, 2013.
// - The SD card is interfaced to the MSP432 via the SPI. SPI
//   connections:
//   - MOSI - pin 15
//   - MISO - pin 14
//   - CLK - pin 7
//   - CS - pin 18 (also referred to as SS pin)
// - Software modifications:
//   - Update file: Sd2PinMap.h with pin assignments provided above.
//   - Change CS (Chip Select) to pin 18 in code below.
// - SD card datalogger
//   - Created 24 Nov 2010
//   - Modified 9 Apr 2012
//   - by Tom Igoe
//   - SD datalogger code is in the public domain.
//- LM34 temperature sensor
//- Sparkfun LCD-09067, serial enabled 16x2 LCD, 3.3 VDC
//This code is in the public domain.
//*****

#include <SPI.h>                //include files for SD card datalogger
#include <SD.h>

//analog input pins
#define wind_dir      13        //analog pin - weather vane
#define temp_sensor   12        //analog pin - LM34 temp sensor

//digital output pins - LED indicators
#define N_LED         24        //digital pin - N LED

```

```
#define NE_LED      5           //digital pin - NE LED
#define E_LED      25          //digital pin - E LED
#define SE_LED     6           //digital pin - SE LED
#define S_LED      26          //digital pin - S LED
#define SW_LED     27          //digital pin - SW LED
#define W_LED      8           //digital pin - W LED
#define NW_LED     28          //digital pin - NW LED

int wind_dir_value;           //declare variable for wind dir
int temp_value;               //declare variable for temp
int troubleshoot = 1;        //1: serial monitor prints

const int chipSelect = 18;

float wind_direction_float;
float temp_value_float;

void setup()
{
  //LED indicators
  pinMode(N_LED, OUTPUT);      //config pin for digital out - N LED
  pinMode(NE_LED, OUTPUT);     //config pin for digital out - NE LED
  pinMode(E_LED, OUTPUT);      //config pin for digital out - E LED
  pinMode(SE_LED, OUTPUT);     //config pin for digital out - SE LED
  pinMode(S_LED, OUTPUT);      //config pin for digital out - S LED
  pinMode(SW_LED, OUTPUT);     //config pin for digital out - SW LED
  pinMode(W_LED, OUTPUT);      //config pin for digital out - W LED
  pinMode(NW_LED, OUTPUT);     //config pin for digital out - NW LED

  //SD card chip select
  pinMode(18, OUTPUT);         //config pin for digital out - SD CS

  //Serial monitor - open serial communications
  if(troubleshoot == 1) Serial.begin(9600);

  //SD card initialization
  if(troubleshoot == 1) Serial.print("Initializing SD card...");

  //see if the card is present and can be initialized:
```

```

    if(!SD.begin(chipSelect))
    {
        if(troubleshoot == 1) Serial.println("Card failed, or not present");
        // don't do anything more:
        return;
    }
    if(troubleshoot == 1) Serial.println("card initialized.");
}

void loop()
{
    // make a string for assembling the data to log:
    String dataString = "";

    //read two sensors and append to the string
    //analog read returns value between 0 and 1023
    wind_dir_value    = analogRead(wind_dir);
    temp_value        = analogRead(temp_sensor);

    if(troubleshoot == 1)Serial.println(wind_dir_value);
    if(troubleshoot == 1)Serial.println(temp_value);

    //LM34 provides 10 mV/degree
    temp_value = (int)(((temp_value/1023.0) * 3.3)/.010);
    if(troubleshoot == 1)Serial.println(temp_value);

    dataString += String(wind_dir_value);
    dataString += ",";
    dataString += String(temp_value);
    dataString += ",";
    delay(10000);           //data update cycle time

    //Open the file.
    Note that only one file can be open at a time,
    File dataFile = SD.open("datalog1.txt", FILE_WRITE);

    // if the file is available, write to it:
    if (dataFile)

```

```

    {
    dataFile.println(dataString);
    dataFile.close();
    // print to the serial port too:
    if(troubleshoot == 1)Serial.println(dataString);
    }
// if the file isn't open, pop up an error:
else
{
if(troubleshoot == 1) Serial.println("error opening datalog1.txt");
}

//display wind direction
display_wind_direction(wind_dir_value);
}

//*****

void display_wind_direction(unsigned int wind_dir_int)
{
float wind_dir_float;
                                //convert wind direction to float
wind_dir_float = wind_dir_int/1023.0 * 3.3;

if(troubleshoot == 1)Serial.println(wind_dir_float);

//N - LED0
if((wind_dir_float <= 2.56)&&(wind_dir_float > 2.50))
{
digitalWrite(N_LED, HIGH); digitalWrite(NE_LED, LOW);
digitalWrite(E_LED, LOW); digitalWrite(SE_LED, LOW);
digitalWrite(S_LED, LOW); digitalWrite(SW_LED, LOW);
digitalWrite(W_LED, LOW); digitalWrite(NW_LED, LOW);
}

//NE - LED1
if((wind_dir_float > 1.46)&&(wind_dir_float <= 1.52))
{
digitalWrite(N_LED, LOW); digitalWrite(NE_LED, HIGH);

```

```
digitalWrite(E_LED, LOW);    digitalWrite(SE_LED, LOW);
digitalWrite(S_LED, LOW);    digitalWrite(SW_LED, LOW);
digitalWrite(W_LED, LOW);    digitalWrite(NW_LED, LOW);
}

//E - LED2
if((wind_dir_float > 0.27)&&(wind_dir_float <= 0.33))
{
digitalWrite(N_LED, LOW);    digitalWrite(NE_LED, LOW);
digitalWrite(E_LED, HIGH);   digitalWrite(SE_LED, LOW);
digitalWrite(S_LED, LOW);    digitalWrite(SW_LED, LOW);
digitalWrite(W_LED, LOW);    digitalWrite(NW_LED, LOW);
}

//SE - LED3
if((wind_dir_float > 0.57)&&(wind_dir_float <= 0.63))
{
digitalWrite(N_LED, LOW);    digitalWrite(NE_LED, LOW);
digitalWrite(E_LED, LOW);    digitalWrite(SE_LED, HIGH);
digitalWrite(S_LED, LOW);    digitalWrite(SW_LED, LOW);
digitalWrite(W_LED, LOW);    digitalWrite(NW_LED, LOW);
}

//S - LED4
if((wind_dir_float > 0.9)&&(wind_dir_float <= 0.96))
{
digitalWrite(N_LED, LOW);    digitalWrite(NE_LED, LOW);
digitalWrite(E_LED, LOW);    digitalWrite(SE_LED, LOW);
digitalWrite(S_LED, HIGH);   digitalWrite(SW_LED, LOW);
digitalWrite(W_LED, LOW);    digitalWrite(NW_LED, LOW);
}

//SW - LED5
if((wind_dir_float > 2.0)&&(wind_dir_float <= 2.06))
{
digitalWrite(N_LED, LOW);    digitalWrite(NE_LED, LOW);
digitalWrite(E_LED, LOW);    digitalWrite(SE_LED, LOW);
digitalWrite(S_LED, LOW);    digitalWrite(SW_LED, HIGH);
digitalWrite(W_LED, LOW);    digitalWrite(NW_LED, LOW);
}
```

```

}

//W - LED6
if((wind_dir_float > 3.02)&&(wind_dir_float <= 3.08))
{
    digitalWrite(N_LED, LOW);    digitalWrite(NE_LED, LOW);
    digitalWrite(E_LED, LOW);    digitalWrite(SE_LED, LOW);
    digitalWrite(S_LED, LOW);    digitalWrite(SW_LED, LOW);
    digitalWrite(W_LED, HIGH);   digitalWrite(NW_LED, LOW);
}

//NW - LED7
if((wind_dir_float > 2.83)&& (wind_dir_float <= 2.89))
{
    digitalWrite(N_LED, LOW);    digitalWrite(NE_LED, LOW);
    digitalWrite(E_LED, LOW);    digitalWrite(SE_LED, LOW);
    digitalWrite(S_LED, LOW);    digitalWrite(SW_LED, LOW);
    digitalWrite(W_LED, LOW);    digitalWrite(NW_LED, HIGH);
}
}

//*****

```

11.4.6 PROJECT EXTENSIONS

The control system provided above has a set of very basic features. Here are some possible extensions for the system.

- Equip the weather station with an LCD display.
- In addition to the wind vane, the Sparkfun weather meters (SEN-08942) include a rain gauge and an anemometer. Add these features to the weather station.
- Extend the eight LED display to sixteen LEDs.

11.5 SUBMERSIBLE ROBOT

The area of submersible robots is fascinating and challenging. To design a robot is quite complex (yet fun). To add the additional requirement of waterproofing key components provides an additional level of challenge. (Water and electricity do not mix!) In this section we provide the construction details and a control system for a remotely operated vehicle, an ROV. Specifically, we develop the structure and control system for the SeaPerch style ROV, as shown in Figure 11.6.

By definition, an ROV is equipped with a tether umbilical cable that provides power and control signals from a surface support platform. An Autonomous Underwater Vehicle (AUV) carries its own power and control equipment and does not require surface support [[Seaperch](#)].

Details on the construction and waterproofing of an ROV are provided in the excellent and fascinating *Build Your Own Underwater Robot and Other Wet Projects* by Harry Bohm and Vickie Jensen. For an advanced treatment, please see *The ROV Manual—A User Guide for Remotely Operated Vehicles* by Robert Crist and Robert Wernli, Sr. There is a national-level competition for students based on the SeaPerch ROV. The goal of the program is to stimulate interest in the next generation of marine related engineering specialties [[Seaperch](#)].

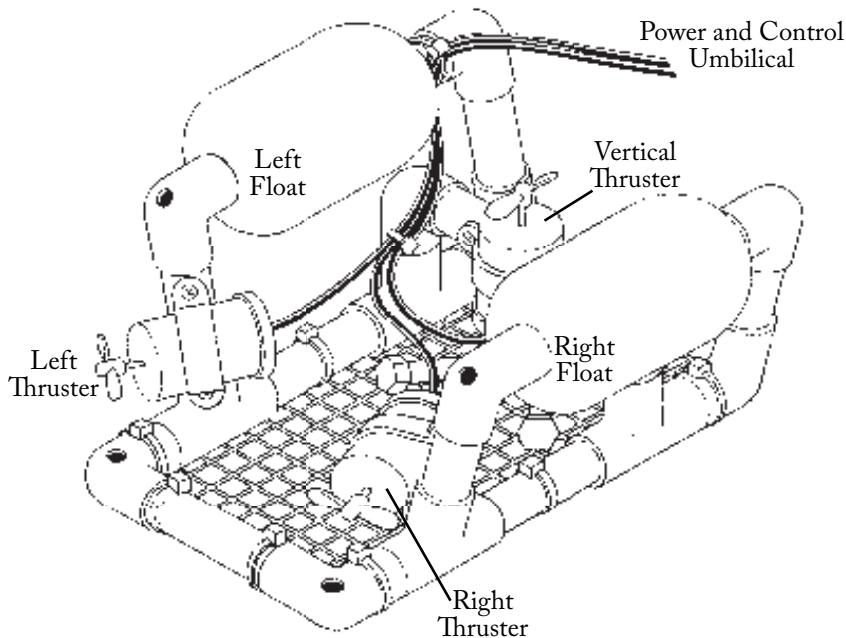


Figure 11.6: SeaPerch ROV. (Adapted and used with permission of Bohm and Jensen, West Coast Words Publishing.)

11.5.1 APPROACH

This is a challenging project; however, we take a methodical, step-by-step approach to successful design and construction of the ROV. We complete the design tasks in the following order:

- determine requirements;
- design and construct ROV structure;

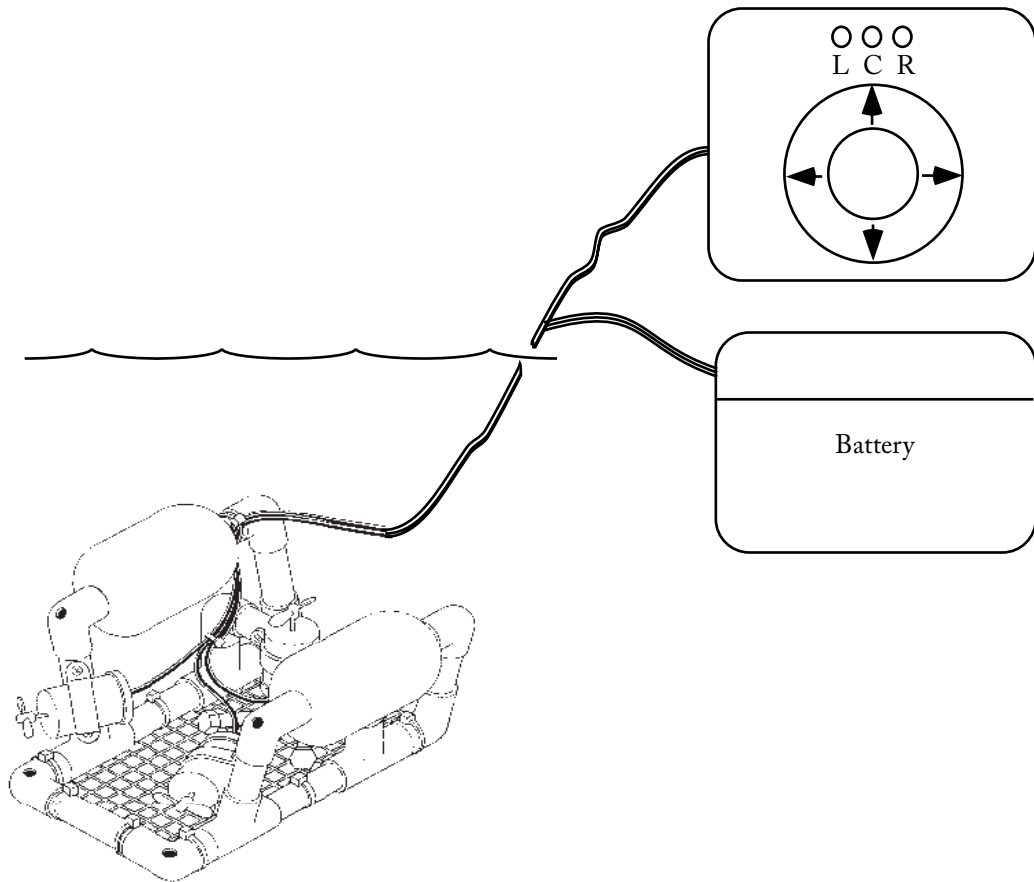


Figure 11.7: Power and control are provided remotely to the SeaPerch ROV. (Adapted and used with permission of Bohm and Jensen, West Coast Words Publishing.)

- design and fabricate control electronics;
- design and implement control software using Energia;
- construct and assemble a prototype; and
- test the prototype.

11.5.2 REQUIREMENTS

The requirements for the ROV system include:

- develop a control system to allow a three thruster (motor or bilge pump) ROV to move forward, left (port), and right (starboard);
- the ROV will be pushed down to a shallow depth via a vertical thruster and return to surface based on its own, slightly positive buoyancy;
- ROV movement will be under joystick control;
- light-emitting diodes (LEDs) are used to indicate thruster assertion;
- all power and control circuitry will be maintained in a surface support platform, as shown in Figure 11.7; and
- an umbilical cable connects the support platform to the ROV.

11.5.3 ROV STRUCTURE

The ROV structure is shown in Figure 11.8. The structure is constructed with 0.75-in PVC piping. The structure is assembled quickly using “T” and corner connectors. The pieces are connected using PVC glue or machine screws. The PVC pipe and connectors are readily available in hardware and home improvement stores.

The fore or bow portion of the structure is equipped with plexiglass panels to serve as mounting bulkheads for the thrusters. The panels are mounted to the PVC structure using ring clamps. Either waterproofed electric motors or submersible bilge pumps are used as thrusters. A bilge pump is a pump specifically designed to remove water from the inside of a boat. The pumps are powered from a 12 VDC source and have typical flow rates from 360 to over 3,500 gallons per minute. They range in price from U.S. \$20–80 (www.shorelinemarinedevelopment.com). Details on waterproofing electric motors are provided in *Build Your Own Underwater Robot and Other Wet Projects*. We use three Shoreline Bilge Pumps rated at 600 gallons per minute (GPM). They are available online from www.walmart.com.

The aft or stern portion of the structure is designed to hold the flexible umbilical cable. The cable provides a link between the MSP432 based control system and the thrusters. Each thruster may require up to 1-2 amps of current. Therefore, a 4-conductor, 16 AWG, braided (for flexibility) conductor cable is recommended. The cable is interfaced to the bilge pump leads using soldered connections or Butt connectors. The interface should be thoroughly waterproofed using caulk. For this design the interface was placed within a section of PVC pipe equipped with end caps. The resulting container is filled with waterproof caulk.

Once the ROV structure is complete, its buoyancy is tested. This is accomplished by placing the ROV structure in water. The goal is to achieve a slightly positive buoyancy. With positive buoyancy the structure floats. With neutral buoyancy the structures hovers beneath the surface. With negative buoyancy the structure sinks. A more positive buoyancy may be achieved by attaching floats or foam to the structure tubing. A more negative buoyancy may be achieved by adding weights to the structure [Bohm and Jensen, 2012].

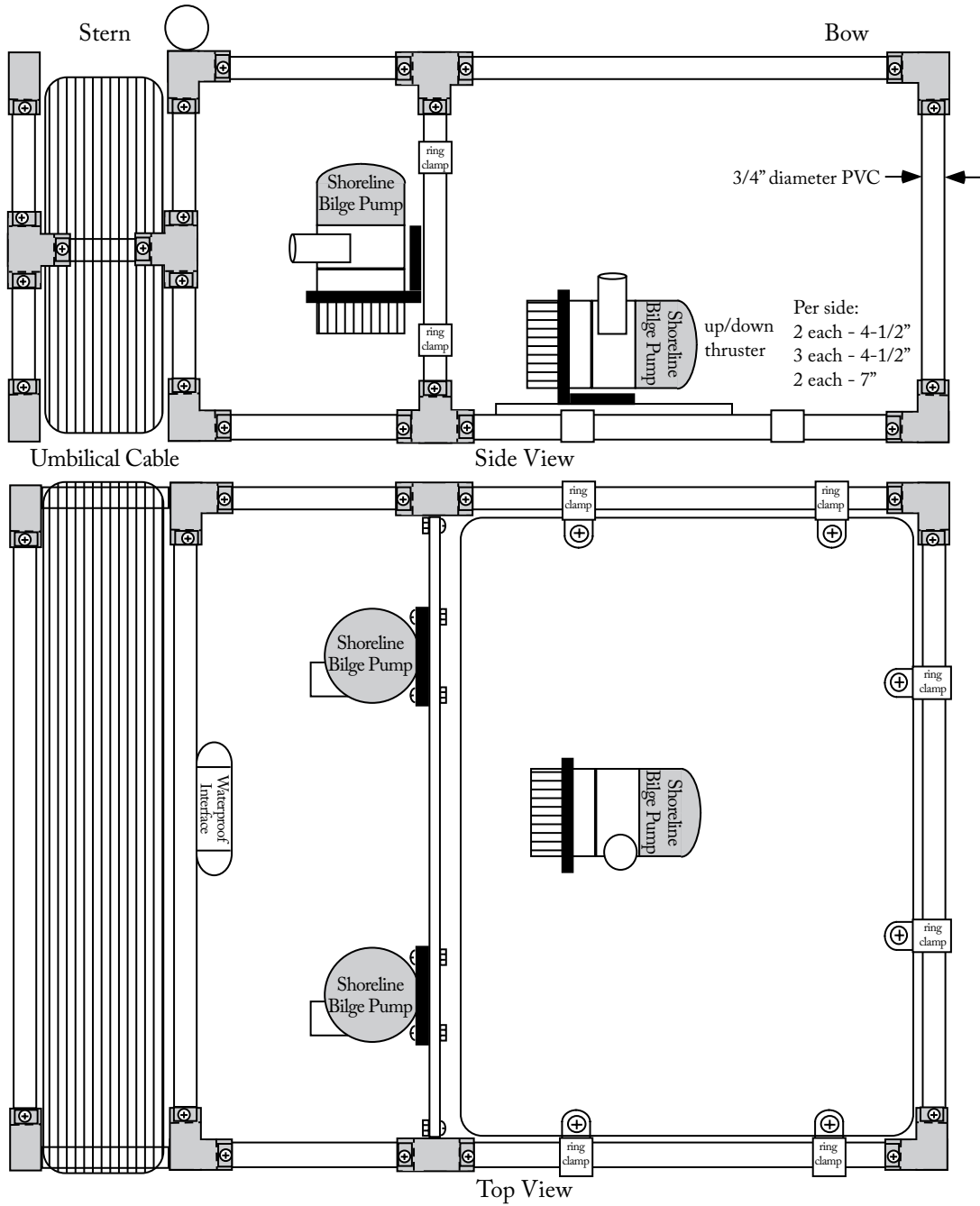


Figure 11.8: SeaPerch ROV structure.

11.5.4 STRUCTURE CHART

The SeaPerch structure chart is provided in Figure 11.9. As can be seen in the figure, the SeaPerch control system will accept input from the five position joystick (left, right, select, up, and down). We use the Sparkfun thumb joystick (Sparkfun COM-09032) mounted to a breakout board (Sparkfun BOB-09110), as shown in Figure 11.11. A MSP432 compatible booster pack is constructed from two 2×10 edge connectors and a prototype builder $1.6'' \times 2.7''$ epoxy glass PCB (Jameco #105100). The joystick breakout board is mounted to this PCB.

The joystick schematic and connections to MSP432 are provided in Figures 11.10 and 11.11.

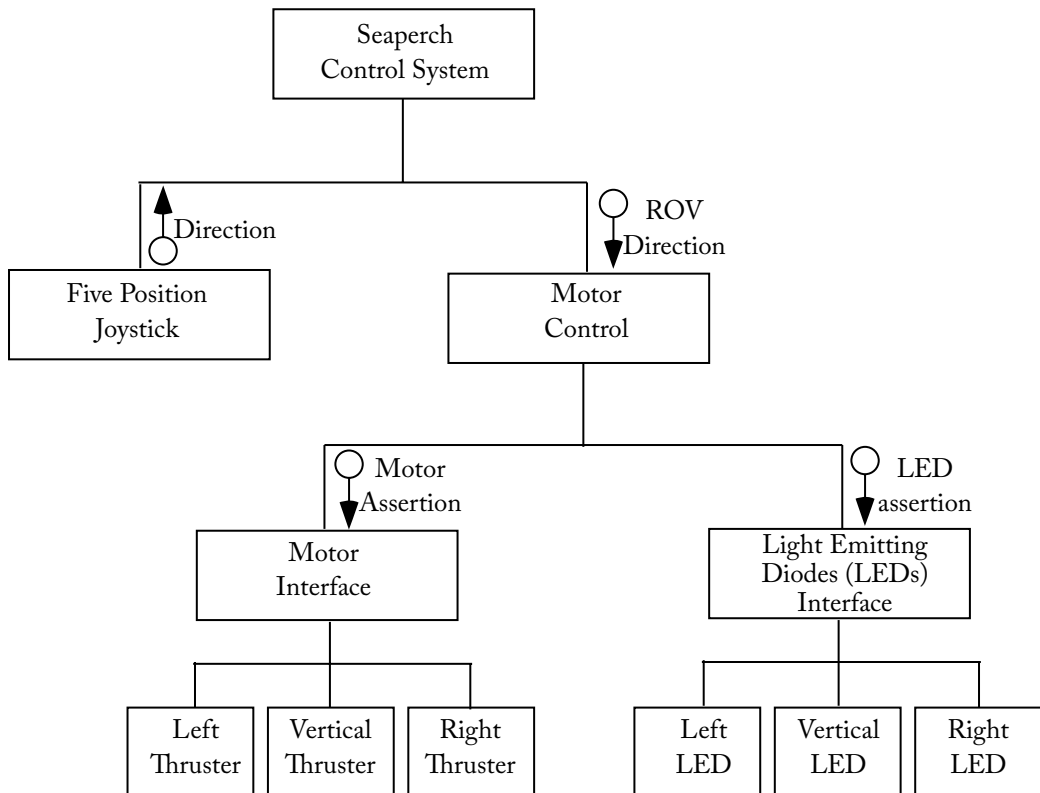


Figure 11.9: SeaPerch ROV structure chart.

In response to user joystick input, the SeaPerch control algorithm will issue a control command indicating desired ROV direction. In response to this desired direction command, the motor control algorithm will issue control signals to assert the appropriate thrusters and signals to illuminate appropriate LEDs.

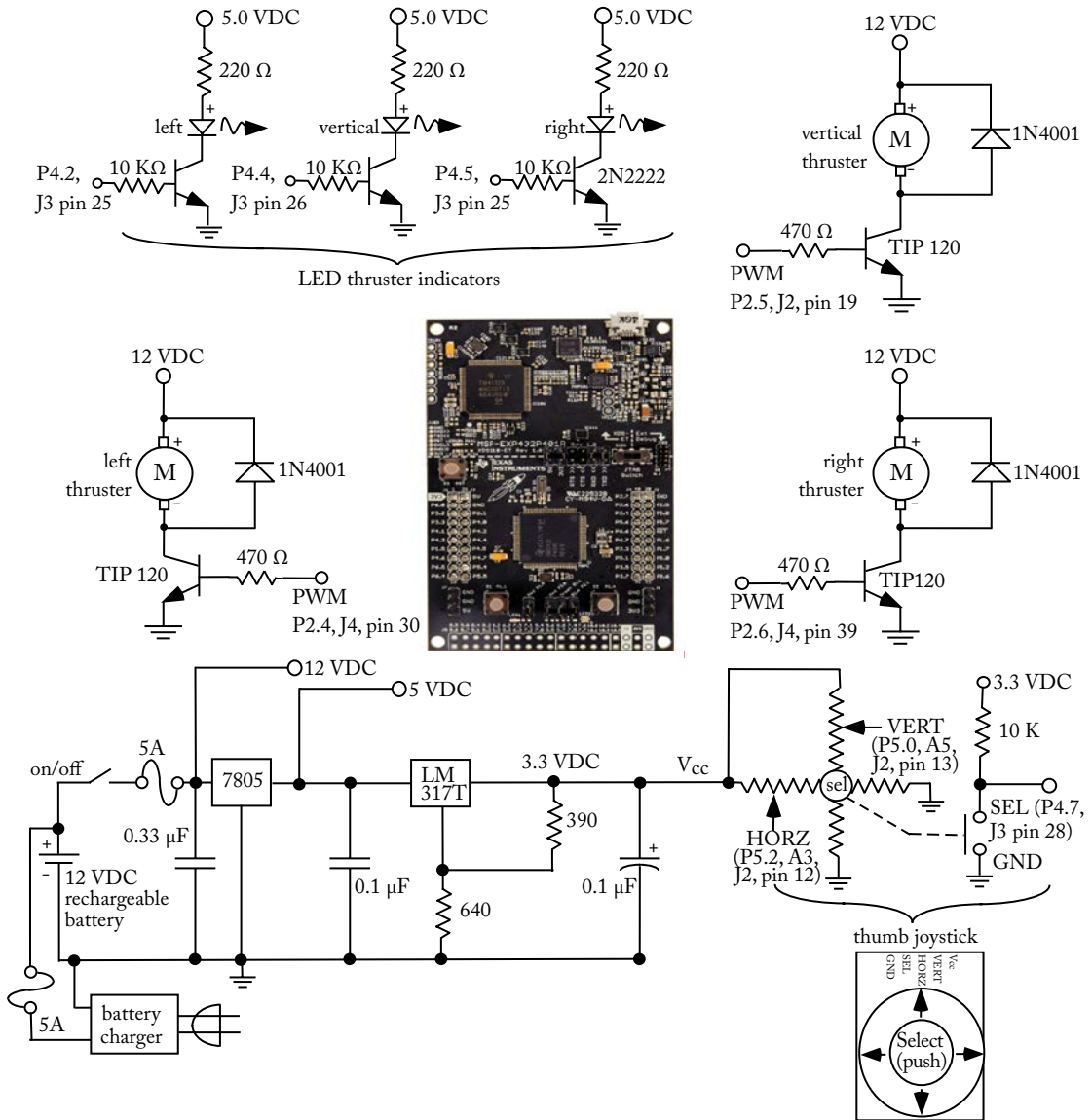


Figure 11.10: SeaPerch ROV interface control.

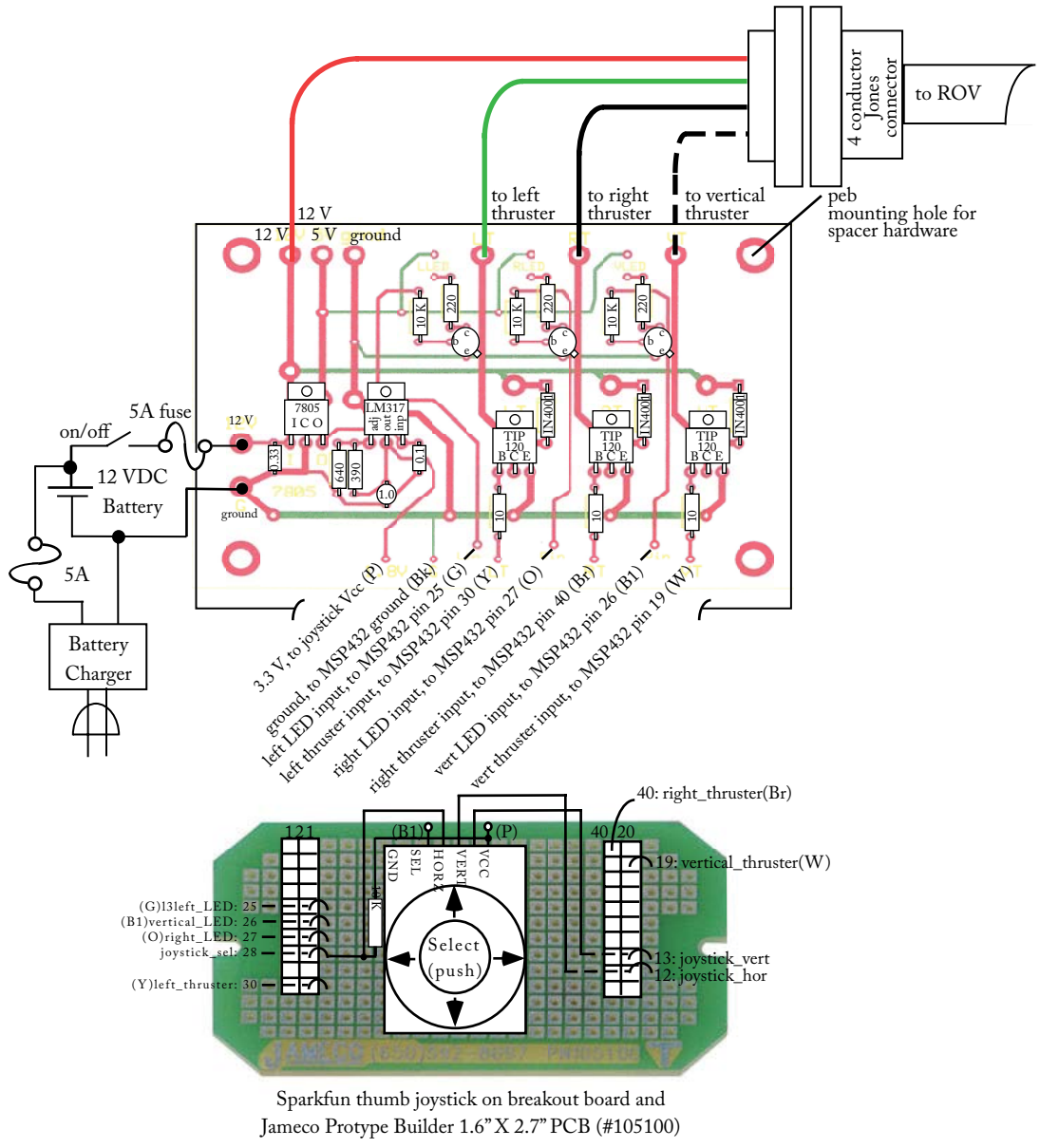


Figure 11.11: SeaPerch ROV printed circuit board interface.

11.5.5 CIRCUIT DIAGRAM

The circuit diagram for the SeaPerch control system is shown in Figure 11.10. The thumb joystick is used to select desired ROV direction. The thumb joystick contains two built-in potentiometers (horizontal and vertical). A reference voltage of 3.3 VDC is applied to the VCC input of the joystick. As the joystick is moved, the horizontal (HORZ) and vertical (VERT) analog output voltages will change to indicate the joystick position. The joystick is also equipped with a digital select (SEL) button. The SEL button is used to activate an ROV dive using the vertical thruster. The joystick is interfaced to MSP432, as shown in Figure 11.10.

There are three LED interface circuits connected to MSP432 header pins P4.2, P4.4, and P4.5. The LEDs illuminate to indicate the left, vertical and right thrusters have been asserted. As previously mentioned, the prime mover for the ROV are three bilge pumps. The left and right bilge pumps are driven by pulse width modulation channels (MSP432 P2.4 and P2.6) via power NPN Darlington transistors (TIP 120), as shown in Figure 11.10. The vertical thrust is under digital pin control P2.5 equipped with NPN Darlington transistor (TIP 120) interface. Both the LED and the pump interfaces were discussed in an earlier chapter.

The interface circuitry between the MSP432 LaunchPad and the bilge pumps is mounted on a printed circuit board (PCB) within the control housing. The interface between MSP432, the PCB, and the umbilical cable is provided in Figure 11.11.

11.5.6 UML ACTIVITY DIAGRAM

The SeaPerch control system UML activity diagram is shown in Figure 11.12. After initializing the MSP432 pins the control algorithm is placed in a continuous loop awaiting user input. In response to user input, the algorithm determines desired direction of ROV travel and asserts appropriate control signals for the LED and motors.

11.5.7 MSP432 CODE

In this example we use the thumb joystick to control the left and right thruster (motor or bilge pump). The joystick provides a separate voltage from 0–3.3 VDC for the horizontal (HORZ) and vertical (VERT) position of the joystick. We use this voltage to set the duty cycle of the pulse width modulated (PWM) signals sent to the left and right thrusters. The select pushbutton (SEL) on the joystick is used to assert the vertical thruster. The analog read function (`analogRead`) is used to read the X and Y position of the joystick. A value from 0–1023 is reported from the analog read function corresponding to 0–3.3 VDC. After the voltage readings are taken they are scaled to 3.3 VDC for further processing. Joystick activity is divided into multiple zones (0–8) as shown in Figure 11.13. The joystick signal is further processed, consistent with the joystick zone selected.

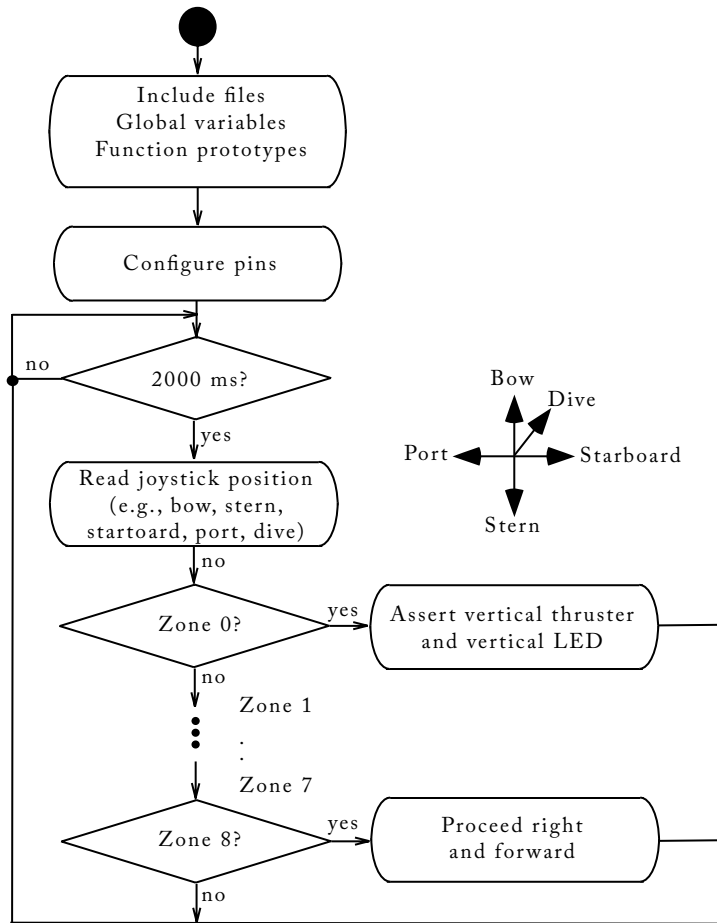


Figure 11.12: SeaPerch ROV UML activity diagram.

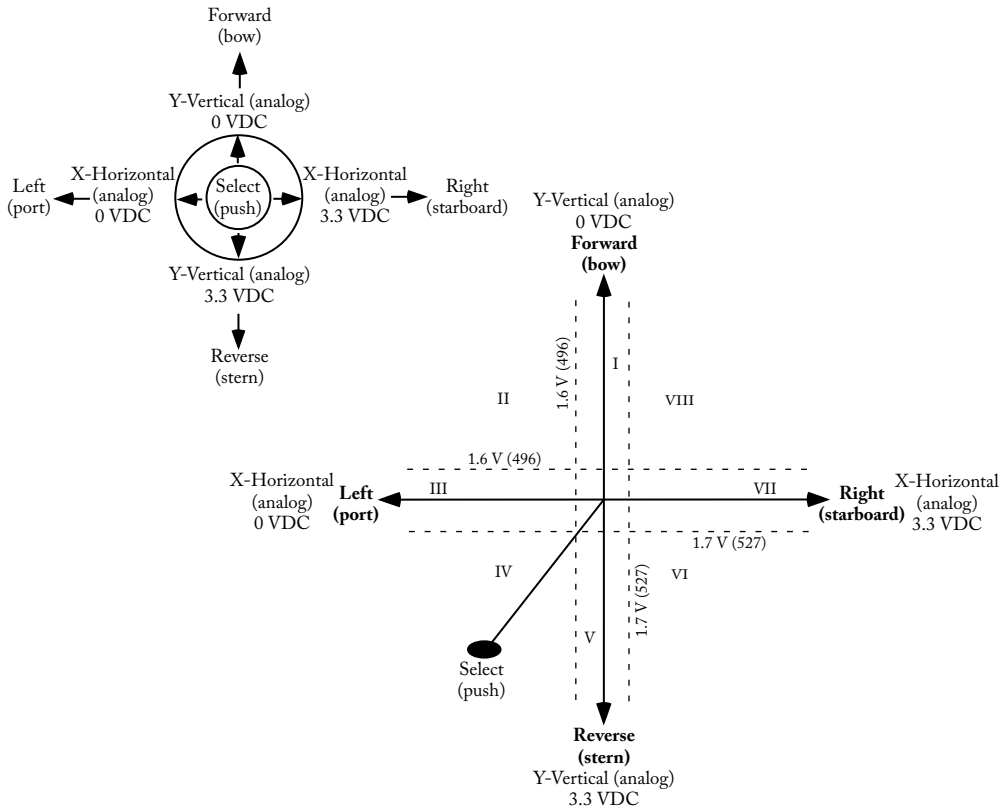


Figure 11.13: Joystick position as related to thruster activity.

```

//*****
//ROV
//In response to joystick input, the SeaPerch control algorithm issues
//a control command indicating desired ROV direction.
In response to
//desired direction command, the motor control algorithm issues
//control signals to assert the appropriate thrusters and LEDs.
//
//This code example is in the public domain.
//*****

//analog input pins
#define joystick_hor      12      //analog pin - joystick horizontal in

```

```

#define joystick_ver      13      //analog pin - joystick vertical in

//digital input pin
#define joystick_sel      28      //digital pin - joystick select in

//digital output pins - LED indicators
#define left_LED          25      //digital pin - left LED out
#define vertical_LED      26      //digital pin - vertical LED out
#define right_LED         27      //digital pin - right LED out

//thruster outputs
#define left_thruster     30      //digital pin - left thruster
#define right_thruster    40      //digital pin - right thruster
#define vertical_thruster 19      //digital pin - vertical thruster

int joystick_hor_value;      //horizontal joystick value
int joystick_ver_value;     //vertical joystick value
int joystick_sel_value;     //joystick select value
int joystick_thrust_on;     //1: thrust on; 0: off
int troubleshoot = 1;       //1: serial monitor prints

void setup()
{
  //LED indicators
  pinMode(left_LED,      OUTPUT); //config pin for digital out - left LED
  pinMode(vertical_LED, OUTPUT); //config pin for digital out - vert LED
  pinMode(right_LED,    OUTPUT); //config pin for digital out - right LED

  //joystick select input
  pinMode(joystick_sel, INPUT);   //config pin for digital in - joystick sel

  //thruster outputs
  pinMode(left_thruster,  OUTPUT); //config digital out - left thruster
  pinMode(vertical_thruster, OUTPUT); //config digital out - vertical thruster
  pinMode(right_thruster, OUTPUT); //config digital out - right thruster

  //Serial monitor - open serial communications
  if(troubleshoot == 1) Serial.begin(9600);
}

```

```
void loop()
{
//set update interval
delay(1000);

//turn off LEDs
digitalWrite(left_LED,    LOW);        //left LED    - off
digitalWrite(vertical_LED, LOW);        //vertical LED - off
digitalWrite(right_LED,   LOW);        //right LED   - off

//read hor and vert joystick position
//analog read returns value between 0 and 1023
joystick_hor_value = analogRead(joystick_hor);
joystick_ver_value = analogRead(joystick_ver);

//Print sensor values to Serial Monitor
if(troubleshoot == 1)
{
  Serial.print("Joystick H: ");
  Serial.println(joystick_hor_value);
  Serial.print("Joystick V: ");
  Serial.println(joystick_ver_value);
}

//Read vertical thrust
joystick_thrust_on = digitalRead(joystick_sel); //vertical thrust?

//*****
//vertical thrust - active low pushbutton on joystick
//*****
if(joystick_thrust_on == 0)
{
  digitalWrite(vertical_thruster, HIGH);
  digitalWrite(vertical_LED, HIGH);
  if(troubleshoot == 1) Serial.println("Thrust is on!");
}
else
{
```

```

digitalWrite(vertical_thruster, LOW);
digitalWrite(vertical_LED, LOW);
if(troubleshoot == 1) Serial.println("Thrust is off!");
}

//*****
//process different joystick zones
//*****
//Case 0: Joystick in null position
//Inputs:
// X channel between 1.60 to 1.70 VDC - null zone (496 to 527)
// Y channel between 1.60 to 1.70 VDC - null zone (496 to 527)
//Output:
// Shut off thrusters
//*****

if((joystick_hor_value > 496)&&(joystick_hor_value < 527)&&
    (joystick_ver_value > 496)&&(joystick_ver_value < 527))

{
    if(troubleshoot == 1) Serial.println("Zone 0");

    if(troubleshoot == 1)
    {
        Serial.print("Joystick H: ");
        Serial.println(joystick_hor_value);
        Serial.print("Joystick V: ");
        Serial.println(joystick_ver_value);
        Serial.print("Thrust: ");
        Serial.println(joystick_thrust_on);
        Serial.println("");
        Serial.println("");
    }

    //assert thrusters to move forward
    analogWrite(left_thruster, 0);
    analogWrite(right_thruster, 0);

    //assert LEDs

```

```

digitalWrite(left_LED, LOW);      //de-assert left LED
digitalWrite(right_LED, LOW);     //de-assert right LED
}

//*****
//*****
//process different joystick zones
//*****
//Case 1:
//Inputs:
// X channel between 1.60 to 1.70 VDC - null zone (496 to 527)
// Y channel <= 1.60 VDC (496)
//Output:
// Move forward - provide same voltage to left and right thrusters
//*****

if((joystick_hor_value > 496)&&(joystick_hor_value < 527)&&
    (joystick_ver_value <= 496))
{
    if(troubleshoot == 1) Serial.println("Zone 1");

    //scale joystick vertical to value from 0 to 1
    joystick_ver_value = 496 - joystick_ver_value;

    if(troubleshoot == 1)
    {
        Serial.print("Joystick H: ");
        Serial.println(joystick_hor_value);
        Serial.print("Joystick V: ");
        Serial.println(joystick_ver_value);
        Serial.print("Thrust: ");
        Serial.println(joystick_thrust_on);
        Serial.println("");
        Serial.println("");
    }

    //assert thrusters to move forward
    analogWrite(left_thruster, joystick_ver_value);
    analogWrite(right_thruster, joystick_ver_value);
}

```

```

//assert LEDs
digitalWrite(left_LED, HIGH);           //assert left LED
digitalWrite(right_LED,HIGH);          //assert right LED
}

//*****
//Case 2:
//Inputs:
// X channel <= 1.60 VDC (496)
// Y channel <= 1.60 VDC (496)
//Output:
// Move forward and bare left
// - Which joystick direction is asserted more?
// - Scale PWM voltage to left and right thruster accordingly
//*****

if((joystick_hor_value <= 496)&&(joystick_ver_value <= 496))
{
  if(troubleshoot == 1) Serial.println("Zone 2");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = 496 - joystick_hor_value;
  joystick_ver_value = 496 - joystick_ver_value;

  if(troubleshoot == 1)
  {
    Serial.print("Joystick H: ");
    Serial.println(joystick_hor_value);
    Serial.print("Joystick V: ");
    Serial.println(joystick_ver_value);
    Serial.print("Thrust: ");
    Serial.println(joystick_thrust_on);
    Serial.println("");
    Serial.println("");
  }

  //assert thrusters and LEDs

```

```

if(joystick_hor_value > joystick_ver_value)
{
  analogWrite(left_thruster, (joystick_hor_value - joystick_ver_value));
  analogWrite(right_thruster, joystick_hor_value);

  //assert LEDs
  digitalWrite(left_LED, HIGH);          //assert left LED
  digitalWrite(right_LED, HIGH);        //assert right LED
}
else
{
  analogWrite(left_thruster, joystick_ver_value);
  analogWrite(right_thruster, (joystick_ver_value - joystick_hor_value));

  //assert LEDs
  digitalWrite(left_LED, HIGH);          //assert left LED
  digitalWrite(right_LED, HIGH);        //assert right LED
}
}

//*****
//Case 3:
//Inputs:
// X channel <= 1.60 VDC (496)
// Y channel between 1.60 to 1.70 VDC - null zone (496 to 527)
//Output:
// Bare left
//*****

if((joystick_hor_value <= 496)&&(joystick_ver_value > 496)&&
(joystick_ver_value < 527))
{
  if(troubleshoot == 1) Serial.println("Zone 3");

  //scale joystick vertical to value from 0 to 1
  joystick_hor_value = 496 - joystick_ver_value;

  if(troubleshoot == 1)

```



```

    {
    Serial.print("Joystick H: ");
    Serial.println(joystick_hor_value);
    Serial.print("Joystick V: ");
    Serial.println(joystick_ver_value);
    Serial.print("Thrust: ");
    Serial.println(joystick_thrust_on);
    Serial.println("");
    Serial.println("");
    }

//assert thrusters
analogWrite(left_thruster, 0);
analogWrite(right_thruster, joystick_hor_value);

//assert LEDs
digitalWrite(left_LED, LOW);           //de-assert left LED
digitalWrite(right_LED, HIGH);        //assert right LED
}

//*****
//Case 4:
//Inputs:
// X channel <= 1.60 VDC (496)
// Y channel >= 1.70 VDC (527)
//Output:
// Bare left to turn around
//*****

if((joystick_hor_value <= 496)&&(joystick_ver_value >= 527))
{
    if(troubleshoot == 1) Serial.println("Zone 4");

    //scale joystick horizontal and vertical to value from 0 to 1
    joystick_hor_value = 496 - joystick_hor_value;
    joystick_ver_value = joystick_ver_value - 527;

    if(troubleshoot == 1)
    {

```

```

Serial.print("Joystick H: ");
Serial.println(joystick_hor_value);
Serial.print("Joystick V: ");
Serial.println(joystick_ver_value);
Serial.print("Thrust: ");
Serial.println(joystick_thrust_on);
Serial.println("");
Serial.println("");
}

//assert thrusters and LEDs
if(joystick_hor_value > joystick_ver_value)
{
  analogWrite(left_thruster, 0);
  analogWrite(right_thruster, (joystick_hor_value-joystick_ver_value));

  //assert LEDs
  digitalWrite(left_LED, LOW);          //de-assert left LED
  digitalWrite(right_LED, HIGH);       //assert right LED
}
else
{
  analogWrite(left_thruster, 0);
  analogWrite(right_thruster, (joystick_ver_value-joystick_hor_value));

  //assert LEDs
  digitalWrite(left_LED, LOW);          //de-assert left LED
  digitalWrite(right_LED, HIGH);       //assert right LED
}
}

//*****
//Case 5:
//Inputs:
// X channel between 1.60 to 1.70 VDC - null zone (496 to 527)
// Y channel >= 1.70 VDC (527)
//Output:
// Move backward - provide same voltage to left and right thrusters
//*****

```

```

if((joystick_hor_value > 496)&&(joystick_hor_value < 527)&&
    (joystick_ver_value >= 527))
{
    if(troubleshoot ==1) Serial.println("Zone 5");

    //scale joystick vertical to value from 0 to 1
    joystick_ver_value = joystick_ver_value - 527;

    if(troubleshoot == 1)
    {
        Serial.print("Joystick H: ");
        Serial.println(joystick_hor_value);
        Serial.print("Joystick V: ");
        Serial.println(joystick_ver_value);
        Serial.print("Thrust: ");
        Serial.println(joystick_thrust_on);
        Serial.println("");
        Serial.println("");
    }

    //assert thrusters
    analogWrite(left_thruster, 0);
    analogWrite(right_thruster, joystick_ver_value);

    //assert LEDs
    digitalWrite(left_LED, LOW);           //de-assert left LED
    digitalWrite(right_LED, HIGH);        //assert right LED
}

//*****
//Case 6:
//Inputs:
// X channel >= 1.70 VDC (527)
// Y channel >= 1.70 VDC (527)
//Output:
// Bare left to turn around
//*****

```

```
if((joystick_hor_value >= 527)&&(joystick_ver_value >= 527))
{
  if(troubleshoot == 1) Serial.println("Zone 6");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = joystick_hor_value - 527;
  joystick_ver_value = joystick_ver_value - 527;

  if(troubleshoot == 1)
  {
    Serial.print("Joystick H: ");
    Serial.println(joystick_hor_value);
    Serial.print("Joystick V: ");
    Serial.println(joystick_ver_value);
    Serial.print("Thrust: ");
    Serial.println(joystick_thrust_on);
    Serial.println("");
    Serial.println("");
  }

  //assert thrusters and LEDs
  if(joystick_hor_value > joystick_ver_value)
  {
    analogWrite(left_thruster, (joystick_hor_value-joystick_ver_value));
    analogWrite(right_thruster, 0);

    //assert LEDs
    digitalWrite(left_LED, HIGH);           //assert left LED
    digitalWrite(right_LED, LOW);          //de-assert right LED
  }
  else
  {
    analogWrite(left_thruster, (joystick_ver_value-joystick_hor_value));
    analogWrite(right_thruster, 0);

    //assert LEDs
    digitalWrite(left_LED, HIGH);           //assert left LED
    digitalWrite(right_LED, LOW);          //de-assert right LED
  }
}
```

```

    }
  }

  /*******
  //Case 7:
  //Inputs:
  // X channel >= 1.70 VDC (527)
  // Y channel between 1.60 to 1.70 VDC - null zone (496 to 527)
  //Output:
  // Bare right
  /*******

  if((joystick_hor_value >= 527)&&(joystick_ver_value > 496)&&
    (joystick_ver_value < 527))
  {
    if(troubleshoot == 1) Serial.println("Zone 7");

    //scale joystick vertical to value from 0 to 1
    joystick_hor_value = joystick_hor_value - 527;

    if(troubleshoot == 1)
    {
      Serial.print("Joystick H: ");
      Serial.println(joystick_hor_value);
      Serial.print("Joystick V: ");
      Serial.println(joystick_ver_value);
      Serial.print("Thrust: ");
      Serial.println(joystick_thrust_on);
      Serial.println("");
      Serial.println("");
    }

    //assert thrusters
    analogWrite(left_thruster, joystick_hor_value);
    analogWrite(right_thruster, 0);

    //assert LEDs
    digitalWrite(left_LED, HIGH);      //assert left LED
    digitalWrite(right_LED, LOW);     //de-assert right LED
  }
}

```

```

}

//*****
//Case 8:
//Inputs:
// X channel >= 1.70 VDC (527)
// Y channel <= 1.60 VDC (496)
//Output:
// Move forward and bare right
// - Which joystick direction is asserted more?
// - Scale PWM voltage to left and right thruster accordingly
//*****

if((joystick_hor_value >= 527)&&(joystick_ver_value <= 496))
{
  if(troubleshoot == 1) Serial.println("Zone 8");

  //scale joystick horizontal and vertical to value from 0 to 1
  joystick_hor_value = joystick_hor_value - 527;
  joystick_ver_value = 496 - joystick_ver_value;

  if(troubleshoot == 1)
  {
    Serial.print("Joystick H: ");
    Serial.println(joystick_hor_value);
    Serial.print("Joystick V: ");
    Serial.println(joystick_ver_value);
    Serial.print("Thrust: ");
    Serial.println(joystick_thrust_on);
    Serial.println("");
    Serial.println("");
  }

  //assert thrusters and LEDs
  if(joystick_hor_value > joystick_ver_value)
  {
    analogWrite(left_thruster, joystick_hor_value);
    analogWrite(right_thruster, (joystick_hor_value-joystick_ver_value));
  }
}

```

```

    //assert LEDs
    digitalWrite(left_LED, HIGH);          //assert left LED
    digitalWrite(right_LED, HIGH);        //assert right LED
  }
else
{
    analogWrite(left_thruster, (joystick_ver_value-joystick_hor_value));
    analogWrite(right_thruster, joystick_ver_value);

    //assert LEDs
    digitalWrite(left_LED, HIGH);          //assert left LED
    digitalWrite(right_LED, HIGH);        //assert right LED
  }
}
}

//*****

```

11.5.8 CONTROL HOUSING LAYOUT

A Plano Model 1312-00 water-resistant field box is used to house the control circuitry and rechargeable battery. The battery is a rechargeable, sealed, lead-acid battery, 12 VDC, with an 8.5 amp-hour capacity. It is available from McMaster-Carr (#7448K82). A battery charger (12 VDC, 4-8 amp-hour rating) is also available (#7448K67). The layout for the ROV control housing is provided in Figure 11.14.

The control circuitry consists of two connected plastic panels as shown in Figure 11.14. The top panel has the on/off switch, the LED thruster indicators (left, dive, and right), an access hole for the joystick, and a 1/4-in jack for the battery recharge cable.

The lower panel is connected to the top panel using aluminum spacers, screws, and corresponding washers, lock washers, and nuts. The lower panel contains the MSP432 equipped with the thumb joystick booster pack assembly. The MSP432 LaunchPad is connected to the lower panel using a Jameco stand off kit (#106551). The MSP432 LaunchPad is interfaced to the thrusters via interface circuitry described in Figures 11.10 and 11.11. The interface printed circuit board is connected to the four conductor thruster cable via a Jones connector.

11.5.9 FINAL ASSEMBLY TESTING

The final system is tested a subassembly at a time. The following sequence is suggested.

- Recheck all waterproofed connections. Reapply waterproof caulk as necessary.

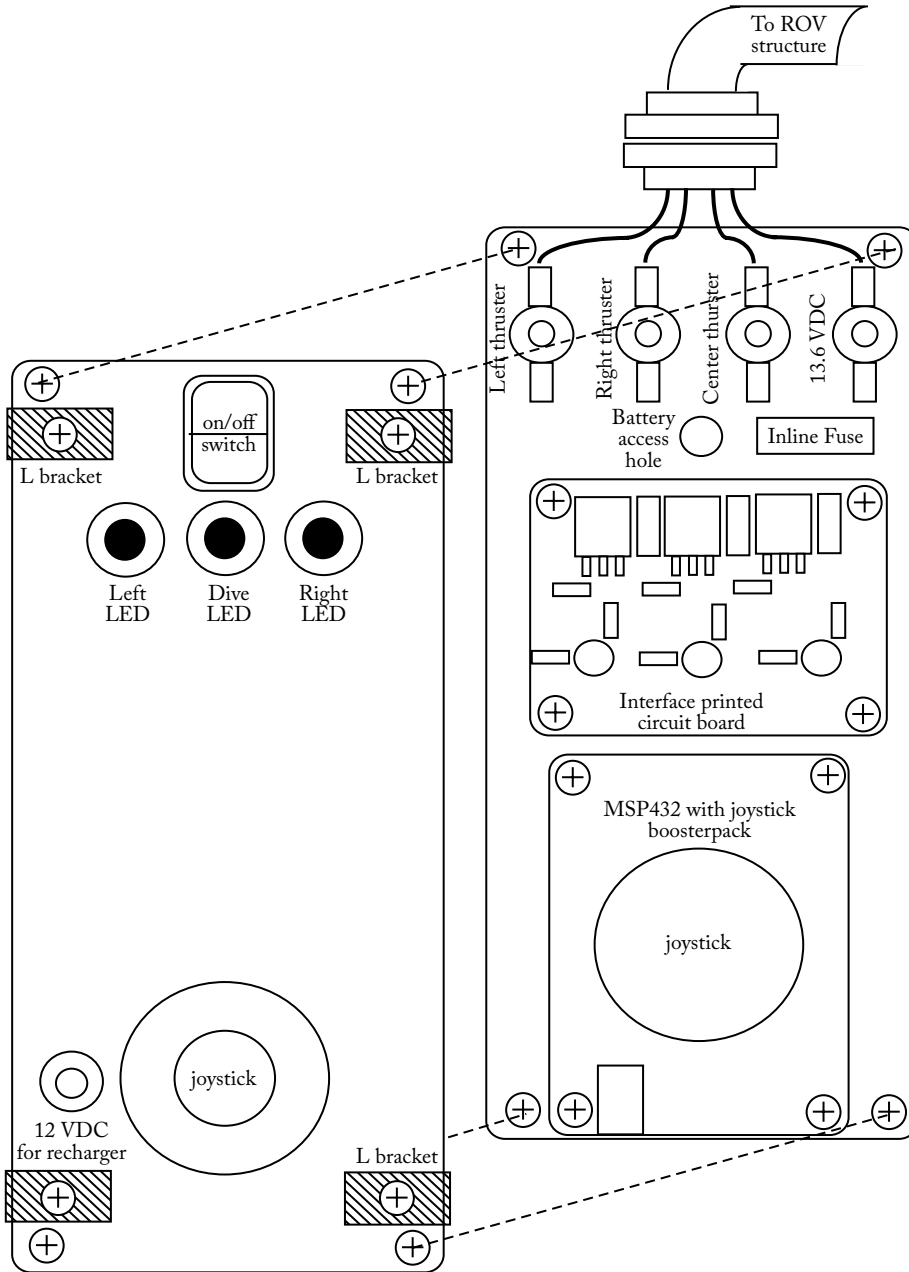


Figure 11.14: ROV control housing layout.

- With power applied, the voltage regulators aboard the printed circuit board should be tested for proper voltages.
- With the MSP432 LaunchPad disconnected, test each LED indicator (left, dive, and right). This is accomplished by applying a 3.3 VDC signal in turn to P4.2 (left LED), P4.4 (vertical LED), and P4.5 (right LEDs).
- In a similar manner each thruster (left, right, and vertical) may be tested. If available, a signal generator may be used to generate a pulse width modulated signal to test each thruster.
- The output voltages from the thumb joystick may be verified at header P5.0 and P5.2 and also the select pushbutton at header P4.7.
- With the software fully functional, the circuit board may be connected to MSP432 LaunchPad for end-to-end testing.

11.5.10 FINAL ASSEMBLY

The fully assembled ROV is shown in Figure 11.15.

11.5.11 PROJECT EXTENSIONS

The control system provided above has a set of very basic features. Here are some possible extensions for the system.

- Provide a powered dive and surface thruster. To provide for a powered dive and surface capability, the ROV must be equipped with a vertical thruster equipped with an H-bridge to allow for motor forward and reversal. This will require a reversible, waterproof motor. This modification is given as an assignment at the end of the chapter.
- Left and right thruster reverse. Currently, the left and right thrusters may only be powered in one direction. To provide additional maneuverability, the left and right thrusters could be equipped with an H-bridge to allow bi-directional motor control. This will require a reversible, waterproof motors. This modification is given as an assignment at the end of the chapter.
- Proportional speed control with bi-directional motor control. Both of these advanced features may be provided by driving the H-bridge circuit with PWM signals. This modification is given as an assignment at the end of the chapter.

11.6 MOUNTAIN MAZE NAVIGATING ROBOT

In this project we extend the Dagu Magician maze navigating project described in Chapter 3 to a three-dimensional mountain pass. We use a robot equipped with four motorized wheels. Each of



Figure 11.15: ROV fully assembled. (Photo courtesy of J. Barrett, Closer to the Sun International.)

the wheels is equipped with an H-bridge to allow bidirectional motor control. Two of the wheels are equipped with encoders to track wheel rotation.

11.6.1 DESCRIPTION

For this project, a DF Robot 4WD mobile platform kit was used (DFROBOT ROB0003, Jameco #2124285). The robot kit is equipped with four powered wheels. As in the Dagu Magician project, we equip the DF Robot with three Sharp GP2Y0A21YKOF IR sensors, as shown in Figure 11.16. The robot is placed in a three-dimensional maze with reflective walls modeled after a mountain pass. The goal of the project is for the robot to detect wall placement and navigate through the maze. The robot will not be provided with any information about the maze. The control algorithm for the robot is hosted on MSP432.

11.6.2 REQUIREMENTS

The requirements for this project are simple: the robot must autonomously navigate through the maze without touching maze walls as quickly as possible. Furthermore, the robot must be able to safely navigate through the rugged maze without becoming “stuck” on maze features.

11.6.3 CIRCUIT DIAGRAM

The circuit diagram for the robot is provided in Figure 11.17. The three IR sensors (left, middle, and right) are mounted on the leading edge of the robot to detect maze walls. The sensors' outputs are fed to three separate analog-to-digital (ADC) channels. The robot motors are driven by PWM channels via an H-bridge. The robot is powered by a 7.5 VDC battery pack (5 AA batteries) which is fed to a 3.3 VDC and 5 VDC voltage regulator. Alternatively, the robot may be powered by a 7.5 VDC power supply rated at several amps. In this case, the power is delivered to the robot by a flexible umbilical cable. The circuit diagram includes the wheel encoders, a liquid crystal display, LEDs to indicate wall detection, and the H-bridge circuit interface. The H-bridge circuit was discussed early in the text. We use four PWM channels to control the forward and reverse action of the motor. The PWM signals are interfaced to the H-bridge via LM324 op amps configured as comparators. The op amps in this configuration translate the 3.3 VDC signals from MSP432 to 5 VDC signals and also boost the sync/source current capability. In the configuration shown, the same control signal is sent to left paired motors and the right paired motors. Other configurations may be used. The printed circuit board (PCB) layout for the robot is provided in Figure 11.18. The assembled robot is shown in Figure 11.19.

11.6.4 STRUCTURE CHART

The structure chart for the robot project is shown in Figure 11.20.

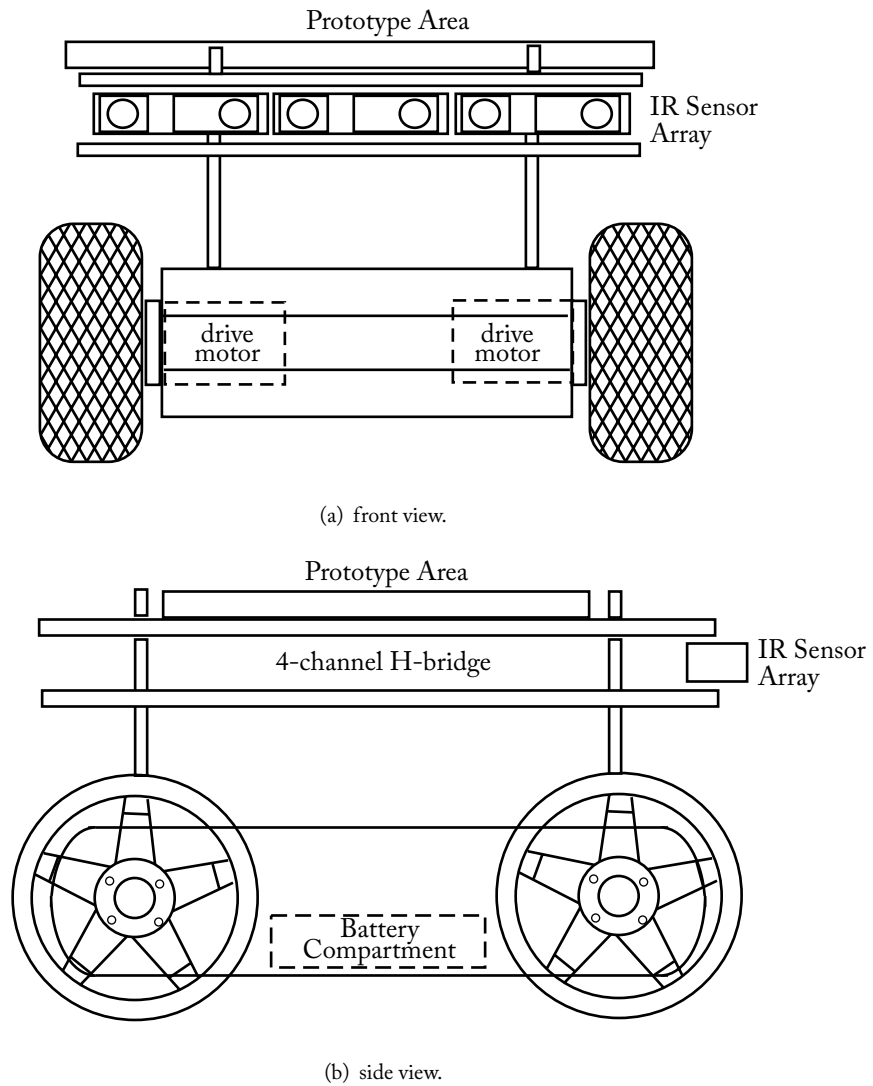


Figure 11.16: Robot layout.

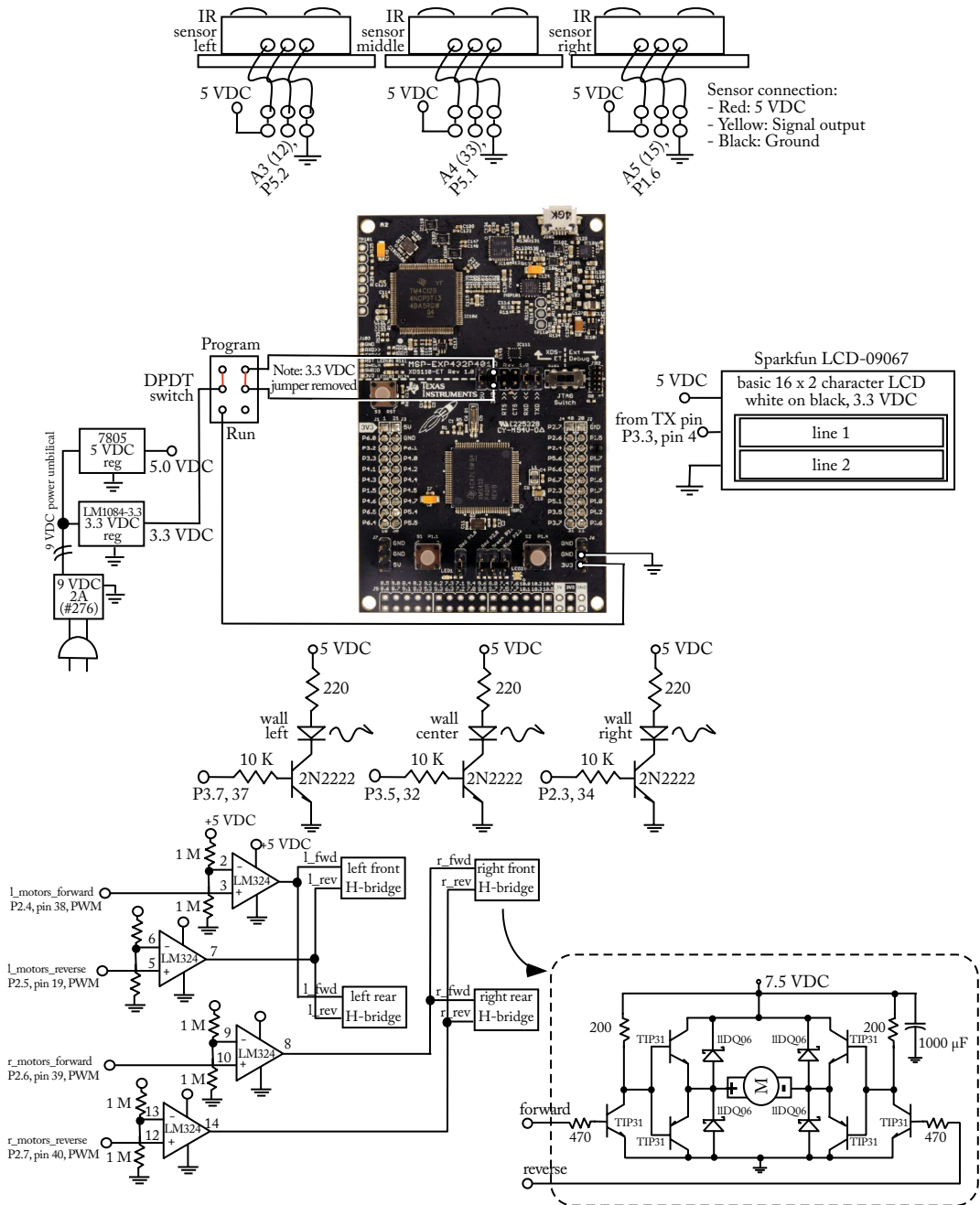


Figure 11.17: Robot circuit diagram.

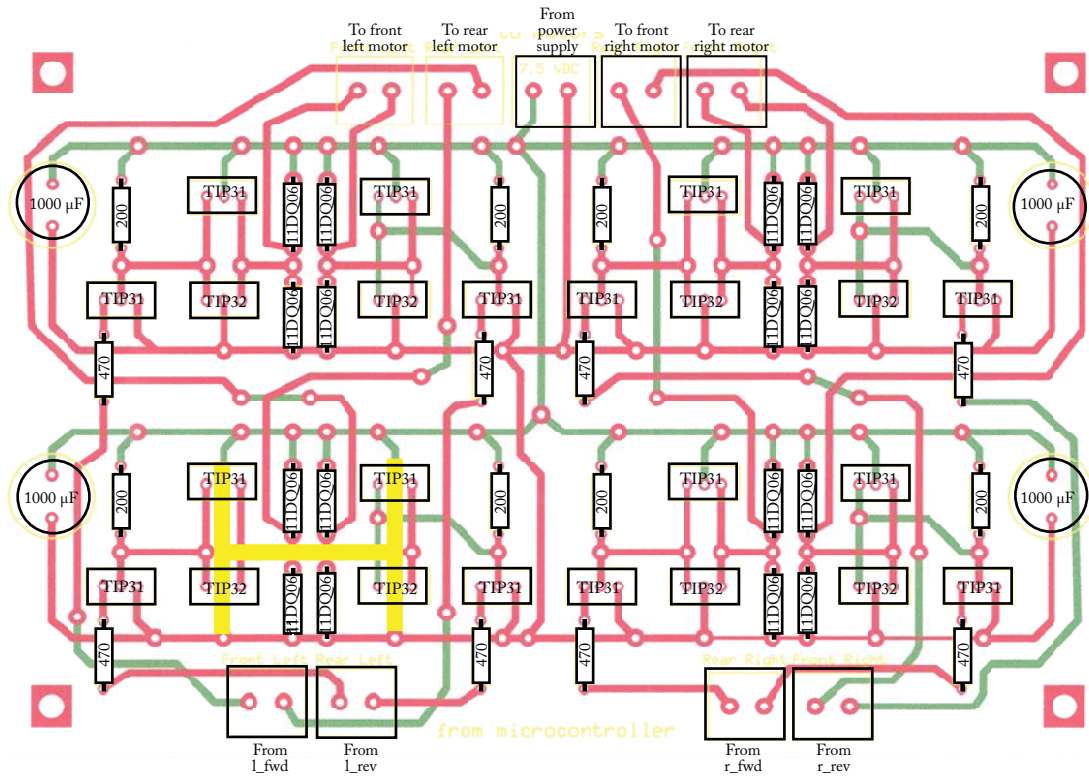


Figure 11.18: 4WD robot PCB.

11.6.5 UML ACTIVITY DIAGRAMS

The UML activity diagram for the robot is provided in Figure 11.21.

11.6.6 4WD ROBOT ALGORITHM CODE

The code for the robot may be adapted from that of the Dagu Magician robot. With the robot wheel motors equipped with an H-bridge, slight modifications are required to the robot turning code. For example, when forward robot movement is desired, PWM signals are sent to both of the left and right forwards signals and a logic zero signal to the left and right reverse signals. To render a left robot turn, a PWM signal is sent to the left_reverse control line and a logic zero to the left_forward control line. Also, a PWM signal is sent to the right_forward control line and a logic zero to the right_reverse control line. The signals are held in this configuration until the program completes the loop cycle. This time can be controlled using the delay function.

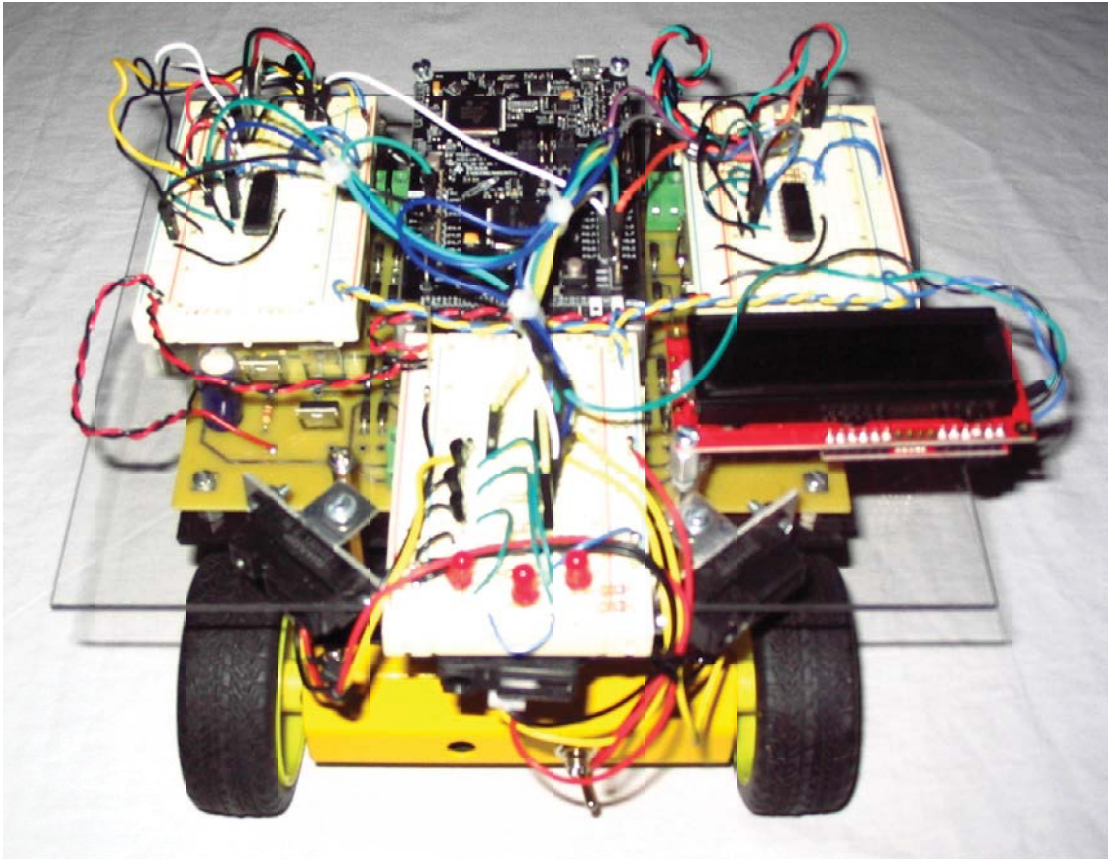


Figure 11.19: 4WD robot.

```

//*****
//robot
//
//Three IR sensors (left, middle, and right) are mounted on the leading
//edge of the robot to detect maze walls.
The sensors' outputs are
//fed to three separate ADC channels on pins 12, 33, and 13.
//
//The robot is equipped with:
// - serial LCD at Serial 1 accessible at:
//   - RX: P3.2, pin 3

```

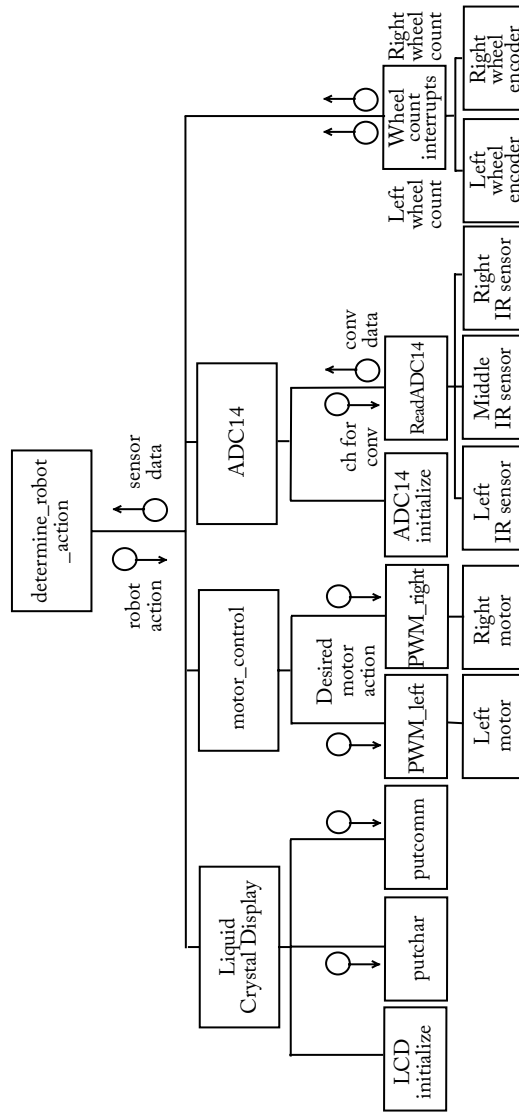


Figure 11.20: Robot structure diagram.

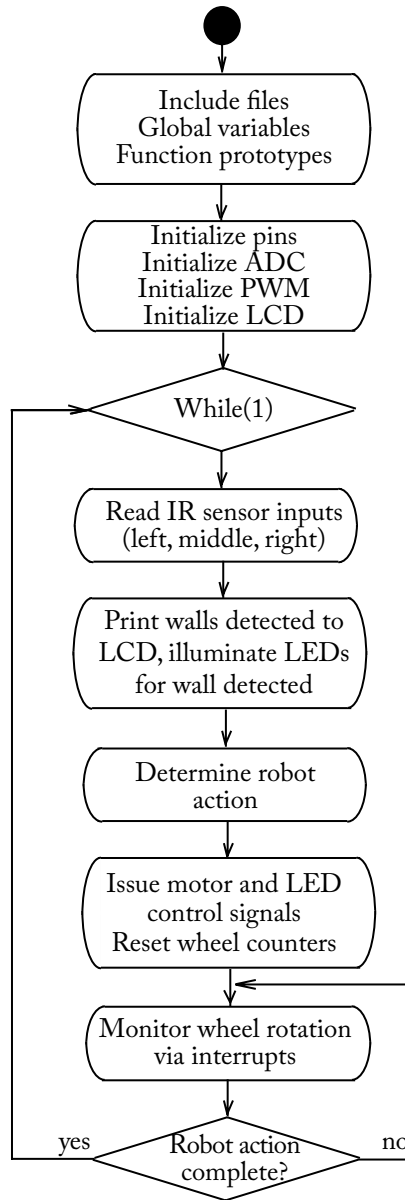


Figure 11.21: Abbreviated robot UML activity diagram. The “determine robot action” consists of multiple decision statements.

```

//      - TX: P3.3, pin 4
// - LEDs to indicate wall detection: 37, 32 and 34
// - Robot motors are driven by PWM channels via an H-bridge.
//      - the same control signal is sent to left paired motors
//        and the right paired motors.
//      - For example, when forward robot movement is desired,
//        PWM signals are sent to both of the left and right forwards
//        signals and a logic zero signal to the left and right reverse signals.
//      - To render a left robot turn, a PWM signal is sent to the
//        left_reverse control line and a logic zero to the left_forward
//        control line.
Also, a PWM signal is sent to the right_forward
//        control line and a logic zero to the right_reverse control line.
//        The signals are held in this configuration until the wheel
//        encoders indicate the turns have been completed.
The wheel
//        encoders provide ten counts per revolution.
// - A separate interrupt is used to count left and right wheel counts.
//
//This example code is in the public domain.
//*****

//analog input pins
#define left_IR_sensor    12    //analog pin - left IR sensor
#define center_IR_sensor  33    //analog pin - center IR sensor
#define right_IR_sensor   13    //analog pin - right IR sensor

//digital output pins
//LED indicators - wall detectors
#define wall_left         37    //digital pin - wall_left
#define wall_center       32    //digital pin - wall_center
#define wall_right        34    //digital pin - wall_right

//motor outputs
#define l_motors_forward  38    //digital pin - left motors forward
#define l_motors_reverse  19    //digital pin - left motors reverse
#define r_motors_forward  39    //digital pin - right motors forward
#define r_motors_reverse  40    //digital pin - right motors reverse

```

```
int left_IR_sensor_value;           //sensor value
int center_IR_sensor_value;         //declare variable for left IR sensor
int right_IR_sensor_value;          //declare variable for center IR sensor
                                     //declare variable for right IR sensor

int troubleshoot;                   //asserts troubleshoot statements

void setup()
{
  troubleshoot = 1;

                                     //enable serial monitor
  if(troubleshoot == 1) Serial.begin(9600);

  //Initialize serial channel 1 to 9600 BAUD and wait for port to open
  //Serial LCD, 3.3 VDC connected to P3.3, pin 4 Sparkfun LCD-09052
  Serial1.begin(9600);
  delay(1000);                       //allow LCD to boot up

                                     //LED indicators - wall detectors
  pinMode(wall_left,  OUTPUT);        //configure pin for digital output
  pinMode(wall_center, OUTPUT);       //configure pin for digital output
  pinMode(wall_right, OUTPUT);        //configure pin for digital output

                                     //motor outputs - PWM
  pinMode(l_motors_forward, OUTPUT);  //configure pin for digital output
  pinMode(l_motors_reverse, OUTPUT);  //configure pin for digital output
  pinMode(r_motors_forward, OUTPUT);  //configure pin for digital output
  pinMode(r_motors_reverse, OUTPUT);  //configure pin for digital output
}

void loop()
{
  //read analog output from IR sensors
  left_IR_sensor_value  = analogRead(left_IR_sensor);
  center_IR_sensor_value = analogRead(center_IR_sensor);
  right_IR_sensor_value = analogRead(right_IR_sensor);

  //Print sensor values to Serial Monitor
  if(troubleshoot == 1)
```

```
{
  Serial.print("Left IR sensor: ");
  Serial.println(left_IR_sensor_value);
  Serial.print("Center IR sensor: ");
  Serial.println(center_IR_sensor_value);
  Serial.print("Right IR sensor: ");
  Serial.println(right_IR_sensor_value);
  Serial.println("");
}

//to LCD
Serial1.write(254);          //Command to LCD
delay(5);
Serial1.write(1);           //Cursor to home position
delay(5);

Serial1.write(254);          //Command to LCD
delay(5);
Serial1.write(128);          //Cursor to home position
delay(5);
Serial1.write("Left  Ctr  Right");
delay(50);
Serial1.write(254);          //Command to LCD
delay(5);
Serial1.write(192);          //Cursor to line 2, position 1
delay(5);
Serial1.print(left_IR_sensor_value);
delay(5);
Serial1.write(254);          //Command to LCD
delay(5);
Serial1.write(198);          //Cursor to line 2, position 1
delay(5);
Serial1.print(center_IR_sensor_value);
delay(5);
Serial1.write(254);          //Command to LCD
delay(5);
Serial1.write(203);          //Cursor to line 2, position 1
delay(5);
Serial1.print(right_IR_sensor_value);
```

```

delay(5);

delay(500);

//robot action table row 0 - robot forward
if((left_IR_sensor_value < 300)&&(center_IR_sensor_value < 300)&&
    (right_IR_sensor_value < 300))
{
    //wall detection LEDs
    digitalWrite(wall_left,  LOW);    //turn LED off
    digitalWrite(wall_center, LOW);    //turn LED off
    digitalWrite(wall_right,  LOW);    //turn LED off
    //motor control
    analogWrite(l_motors_forward, 64); //0(off)-255(full speed)
    analogWrite(l_motors_reverse,  0); //0(off)-255(full speed)
    analogWrite(r_motors_forward, 64); //0(off)-255(full speed)
    analogWrite(r_motors_reverse,  0); //0(off)-255(full speed)

    if(troubleshoot == 1) Serial.print("Table row 0 \n\n");
}

//robot action table row 1 - robot forward
else if((left_IR_sensor_value < 300)&&(center_IR_sensor_value < 300)&&
    (right_IR_sensor_value > 300))
{
    //wall detection LEDs
    digitalWrite(wall_left,  LOW);    //turn LED off
    digitalWrite(wall_center, LOW);    //turn LED off
    digitalWrite(wall_right,  HIGH);   //turn LED on
    //motor control
    analogWrite(l_motors_forward, 64); //0(off)-255(full speed)
    analogWrite(l_motors_reverse,  0); //0(off)-255(full speed)
    analogWrite(r_motors_forward, 64); //0(off)-255(full speed)
    analogWrite(r_motors_reverse,  0); //0(off)-255(full speed)

    if(troubleshoot == 1) Serial.print("Table row 1 \n\n");
}

```

```

//robot action table row 2 - robot right
else if((left_IR_sensor_value < 300)&&(center_IR_sensor_value > 300)&&
        (right_IR_sensor_value < 300))
{
    //wall detection LEDs
    digitalWrite(wall_left, LOW); //turn LED off
    digitalWrite(wall_center, HIGH); //turn LED on
    digitalWrite(wall_right, LOW); //turn LED off

    //motor control
    analogWrite(l_motors_forward, 64); //0(off)-255(full speed)
    analogWrite(l_motors_reverse, 0); //0(off)-255(full speed)
    analogWrite(r_motors_forward, 0); //0(off)-255(full speed)
    analogWrite(r_motors_reverse, 64); //0(off)-255(full speed)

    if(troubleshoot == 1) Serial.print("Table row 2 \n\n");
}

//robot action table row 3 - robot left
else if((left_IR_sensor_value < 300)&&(center_IR_sensor_value > 300)&&
        (right_IR_sensor_value > 300))
{
    //wall detection LEDs
    digitalWrite(wall_left, LOW); //turn LED off
    digitalWrite(wall_center, HIGH); //turn LED on
    digitalWrite(wall_right, HIGH); //turn LED on

    //motor control
    analogWrite(l_motors_forward, 0); //0(off)-255(full speed)
    analogWrite(l_motors_reverse, 64); //0(off)-255(full speed)
    analogWrite(r_motors_forward, 64); //0(off)-255(full speed)
    analogWrite(r_motors_reverse, 0); //0(off)-255(full speed)

    if(troubleshoot == 1) Serial.print("Table row 3 \n\n");
}

//robot action table row 4 - robot forward
else if((left_IR_sensor_value > 300)&&(center_IR_sensor_value < 300)&&
        (right_IR_sensor_value < 300))
{

```

```

//wall detection LEDs
digitalWrite(wall_left, HIGH); //turn LED on
digitalWrite(wall_center, LOW); //turn LED off
digitalWrite(wall_right, LOW); //turn LED off
//motor control
analogWrite(l_motors_forward, 64); //0(off)-255(full speed)
analogWrite(l_motors_reverse, 0); //0(off)-255(full speed)
analogWrite(r_motors_forward, 64); //0(off)-255(full speed)
analogWrite(r_motors_reverse, 0); //0(off)-255(full speed)

if(troubleshoot == 1) Serial.print("Table row 4 \n\n");

}

//robot action table row 5 - robot forward
else if((left_IR_sensor_value > 300)&&(center_IR_sensor_value < 300)&&
(right_IR_sensor_value > 300))
{
//wall detection LEDs
digitalWrite(wall_left, HIGH); //turn LED on
digitalWrite(wall_center, LOW); //turn LED off
digitalWrite(wall_right, HIGH); //turn LED on
//motor control
analogWrite(l_motors_forward, 64); //0(off)-255(full speed)
analogWrite(l_motors_reverse, 0); //0(off)-255(full speed)
analogWrite(r_motors_forward, 64); //0(off)-255(full speed)
analogWrite(r_motors_reverse, 0); //0(off)-255(full speed)

if(troubleshoot == 1) Serial.print("Table row 5 \n\n");

}

//robot action table row 6 - robot right
else if((left_IR_sensor_value > 300)&&(center_IR_sensor_value > 300)&&
(right_IR_sensor_value < 300))
{
//wall detection LEDs
digitalWrite(wall_left, HIGH); //turn LED on
digitalWrite(wall_center, HIGH); //turn LED on

```

```

digitalWrite(wall_right, LOW);          //turn LED off
                                        //motor control
analogWrite(l_motors_forward, 64);     //0(off)-255(full speed)
analogWrite(l_motors_reverse, 0);      //0(off)-255(full speed)
analogWrite(r_motors_forward, 0);      //0(off)-255(full speed)
analogWrite(r_motors_reverse, 64);     //0(off)-255(full speed)

if(troubleshoot == 1) Serial.print("Table row 6 \n\n");
}

//robot action table row 7 - robot reverse
else if((left_IR_sensor_value > 300)&&(center_IR_sensor_value > 300)&&
        (right_IR_sensor_value > 300))
{
                                        //wall detection LEDs
digitalWrite(wall_left, HIGH);         //turn LED on
digitalWrite(wall_center, HIGH);       //turn LED on
digitalWrite(wall_right, HIGH);        //turn LED on
                                        //motor control
analogWrite(l_motors_forward, 64);     //0(off)-255(full speed)
analogWrite(l_motors_reverse, 0);      //0(off)-255(full speed)
analogWrite(r_motors_forward, 0);      //0(off)-255(full speed)
analogWrite(r_motors_reverse, 64);     //0(off)-255(full speed)

if(troubleshoot == 1) Serial.print("Table row 7 \n\n");
}
}

//*****

```

11.6.7 MOUNTAIN MAZE

The mountain maze is constructed from plywood, chicken wire, expandable foam, plaster cloth, and Bondo. A rough sketch of the desired maze path is first constructed. Care is taken to ensure the pass is wide enough to accommodate the robot. The maze platform is constructed from 3/8-in plywood on 2 by 4-in framing material. Maze walls are also constructed from the plywood and supported with steel L brackets.

With the basic structure complete, the maze walls are covered with chicken wire. The chicken wire is secured to the plywood with staples. The chicken wire is then covered with plaster cloth (Creative Mark Artist Products #15006). To provide additional stability, expandable foam is sprayed under the chicken wire (Guardian Energy Technologies, Inc. Foam It Green 12). The mountain scene is then covered with a layer of Bondo for additional structural stability. Bondo is a two-part putty that hardens into a strong resin. Mountain pass construction steps are illustrated in Figure 11.22. The robot is shown in the maze in Figure 11.23

11.6.8 PROJECT EXTENSIONS

- Modify the turning commands such that the PWM duty cycle and the length of time the motors are on are sent in as variables to the function.
- Develop a function for reversing the robot.
- Equip the robot wheels with encoders to control the length of a turn.
- Equip the motor with another IR sensor that looks down toward the maze floor for “land mines.” A land mine consists of a paper strip placed in the maze floor that obstructs a portion of the maze. If a land mine is detected, the robot must deactivate it by moving slowly back and forth for 3 s and flashing a large LED.
- Develop a four-wheel drive system which includes a tilt sensor. The robot should increase motor RPM (duty cycle) for positive inclines and reduce motor RPM (duty cycle) for negative inclines.
- Equip the robot with an analog inertial measurement unit (IMU) to measure vehicle tilt. Use the information provided by the IMU to optimize robot speed going up and down steep grades.

11.7 LABORATORY EXERCISE: PROJECT EXTENSIONS

The weather station presented earlier in the chapter has a set of very basic features. Implement the following extensions to the system.

- Equip the weather station with an LCD display.
- In addition to the wind vane, the Sparkfun weather meters (SEN-08942) include a rain gauge and anemometer. Add these features to the weather station.
- Extend the 8 LED display to 16 LEDs.



Figure 11.22: Mountain maze.



Figure 11.23: Robot in maze. (Photo courtesy of J. Barrett, Closer to the Sun International.)

11.8 SUMMARY

In this chapter, we discussed the design process and related tools, and applied the process to real world designs. It is essential to follow a systematic, disciplined approach to embedded systems design to successfully develop a prototype that meets established requirements.

11.9 REFERENCES AND FURTHER READING

Anderson, M. Help wanted: embedded engineers why the United States is losing its edge in embedded systems. *IEEE-USA Today's Engineer*, Feb 2008. 476

Barrett, S. and Pack, D. 2012. *Atmel AVR Processor Primer: Programming and Interfacing*, 2nd ed., San Rafael, CA, Morgan & Claypool Publishers. DOI: [10.2200/s00427ed1v01y201206dcs039](https://doi.org/10.2200/s00427ed1v01y201206dcs039).

- Barrett, S. and Pack, D. 2005. *Embedded Systems Design and Applications with the 68HC12 and HCS12*. Upper Saddle River, NJ, Pearson Prentice Hall. Print. 478
- Barrett, S. and Pack, D. 2006. *Processors Fundamentals for Engineers and Scientists*, San Rafael, CA, Morgan & Claypool Publishers. www.morganclaypool.com
- Bohm, H. and Jensen, V. 2012. *Build Your Own Underwater Robot and Other Wet Projects*, 11th ed., Vancouver, B.C. Canada, Westcoast Words. 494
- Christ, R. and Wernli, Sr. R. 2014. *The ROV Manual—A User Guide for Remotely Operated Vehicle*. 2nd ed., Oxford. U.K., Butterworth-Heinemann imprint of Elsevier.
- Dale, N. and Lilly, S. C. 1995. *Pascal Plus Data Structures*, 4th ed., Englewood Cliffs, NJ, Jones and Bartlett. 480
- Fowler, M. and Scott, K. 2000. *UML Distilled A Brief Guide to the Standard Object Modeling Language*, 2nd ed., Boston, MA, Addison-Wesley. 479
- Seaperch*, www.seaperch.com 492

11.10 CHAPTER EXERCISES

Fundamental

1. What is an embedded system?
2. What aspects must be considered in the design of an embedded system?
3. What is the purpose of the structure chart, UML activity diagram, and circuit diagram?
4. Why is a system design only as good as the test plan that supports it?
5. During the testing process, when an error is found and corrected, what should now be accomplished?
6. Discuss the top-down design, bottom-up implementation concept.
7. Describe the value of accurate documentation.
8. What is required to fully document an embedded systems design?
9. For the Dagu Magician robot, modify the PWM turning commands such that the PWM duty cycle and the length of time the motors are on are sent in as variables to the function.

10. For the Dagu Magician robot, equip the motor with another IR sensor that looks down for “land mines.” A land mine consists of a paper strip placed in the maze floor that obstructs a portion of the maze. If a land mine is detected, the robot must deactivate it by rotating about its center axis three times and flashing a large LED while rotating.
11. For the Dagu Magician robot, develop a function for reversing the robot.

Advanced

1. Provide a powered dive and surface thruster for the SeaPerch ROV. To provide for a powered dive and surface capability, the ROV must be equipped with a vertical thruster equipped with an H-bridge to allow for motor forward and reversal.
2. Provide a left and right thruster reverse for the SeaPerch ROV. Currently, the left and right thrusters may only be powered in one direction. To provide additional maneuverability, the left and right thrusters could be equipped with an H-bridge to allow bi-directional motor control.
3. Provide proportional speed control with bi-directional motor control for the SeaPerch ROV. Both of these advanced features may be provided by driving the H-bridge circuit with PWM signals.
4. For the 4WD robot, modify the PWM turning commands such that the PWM duty cycle and the length of time the motors are on are sent in as variables to the function.
5. For the 4WD robot, equip the motor with another IR sensor that looks down for “land mines.” A land mine consists of a paper strip placed in the maze floor that obstructs a portion of the maze. If a land mine is detected, the robot must deactivate it by rotating about its center axis three times and flashing a large LED while rotating.
6. For the 4WD robot, develop a function for reversing the robot.
7. For the 4WD robot, develop a four wheel drive system which includes a tilt sensor. The robot should increase motor RPM (duty cycle) for positive inclines and reduce motor RPM (duty cycle) for negatives inclines.
8. Equip the robot with an inertial measurement unit (IMU) to measure vehicle tilt. Use the information provided by the IMU to optimize robot speed going up and down steep grades.
9. Develop an embedded system controlled dirigible/blimp (www.microflight.com, www.rc-toys.com).

Challenging

1. Develop a trip odometer for your bicycle (Hint: use a Hall Effect sensor to detect tire rotation).

2. Develop a timing system for a four lane Pinewood Derby track.
3. Develop a playing board and control system for your favorite game (Yahtzee, Connect Four, Battleship, etc.).
4. You have a very enthusiastic dog that loves to chase balls. Develop a system to launch balls for the dog.
5. Construct the UML activity diagrams for all functions related to the weather station.
6. It is desired to updated weather parameters every 15 min. Write a function to provide a 15 minute delay.
7. Add one of the following sensors to the weather station:
 - anemometer
 - barometer
 - hygrometer
 - rain gauge
 - thermocouple

You will need to investigate background information on the selected sensor, develop an interface circuit for the sensor, and modify the weather station code.

8. Modify the weather station software to also employ the 138 × 110 LCD. Display pertinent weather data on the display.
9. Modify the 4WD robot to generate voice output (Hint: Use an ISD 4003 Chip Corder).
10. Develop an embedded system controlled submarine (www.seaperch.org).
11. Equip the MSP432 with automatic cell phone dialing capability to notify you when a fire is present in your home.

Authors' Biographies

DUNG DANG

Dung Dang has served as an applications engineer for Texas Instruments since 2007. He has served in various positions with the MSP430 and MSP432 microcontroller product lines and now serves as the MSP432 Platform Marketing Manager. He is an advocate of open-source platforms to allow ready adoption of microcontroller innovations in education and industry. He is the technical founder of the TI LaunchPad ecosystem. His service has taken him worldwide for customer field training and support. On a daily basis he collaborates with teams in Germany, India, China, and the U.S. Dung Dang holds an MSEE degree from Saint Mary's University at San Antonio, concentrating on embedded systems and image processing. He served as a Research Assistant at Saint Mary's for two years.

DANIEL J. PACK

Daniel J. Pack is the Dean of the College of Engineering and Computer Science at the University of Tennessee, Chattanooga (UTC). Prior to joining UTC, he was Professor and Mary Lou Clarke Endowed Chair of the Electrical and Computer Engineering Department at the University of Texas, San Antonio, after serving as Professor (now Professor Emeritus) of Electrical and Computer Engineering at the United States Air Force Academy (USAF), CO, where he served as Director of the Academy Center for Unmanned Aircraft Systems Research.

He received a Bachelor of Science degree in Electrical Engineering, a Master of Science degree in Engineering Sciences, and a Ph.D. degree in Electrical Engineering from Arizona State University, Harvard University, and Purdue University, respectively. He also spent a year as a visiting scholar at the Massachusetts Institute of Technology–Lincoln Laboratory. Dr. Pack has co-authored seven textbooks on embedded systems (including *68HC12 Microcontroller: Theory and Applications* and *Embedded Systems: Design and Applications with the 68HC12 and HCS12*) and published over 130 book chapters, technical journal/transactions, and conference papers on unmanned systems, cooperative control, robotics, pattern recognition, and engineering education. He is the recipient of a number of teaching and research awards including Carnegie U.S. Professor of the Year Award, Frank J. Seiler Research Excellence Award, Tau Beta Pi Outstanding Professor Award, Academy Educator Award, and Magoon Award. He is a member of Eta Kappa Nu (Electrical Engineering Honorary), Tau Beta Pi (Engineering Honorary), IEEE (senior member), and the American Society of Engineering Education.

He is a registered Professional Engineer in Colorado and currently serves as Editor-at-Large for *Journal of Intelligent & Robotic Systems* and as Associate Editor for *IEEE Systems Journal*. His research interests include unmanned aerial vehicles, intelligent control, automatic target recognition, robotics, and engineering education.

STEVEN F. BARRETT

Steven F. Barrett, Ph.D., P.E., received a B.S. in Electronic Engineering Technology from the University of Nebraska Lincoln (Omaha campus) in 1979, a M.E. in Electrical Engineering from the University of Idaho at Moscow in 1986, and a Ph.D. in Electrical Engineering from The University of Texas at Austin in 1993. He was formally an active duty faculty member at the United States Air Force Academy, Colorado and now serves as the Associate Dean of Academic Programs and professor of electrical and computer engineering at the University of Wyoming. He is a member of IEEE (senior) and Tau Beta Pi (chief faculty advisor). His research interests include digital and analog image processing, computer-assisted laser surgery, and embedded controller systems. He is a registered Professional Engineer in Wyoming and Colorado and serves on the Wyoming State Board of Professional Engineers and Surveyors. He has co-written several textbooks on microcontrollers and embedded systems. In 2004, Barrett was named "Wyoming Professor of the Year" by the Carnegie Foundation for the Advancement of Teaching and in 2008 was the recipient of the National Society of Professional Engineers (NSPE) Professional Engineers in Higher Education, Engineering Education Excellence Award.

Index

- AC device control, 159
- AC interfacing, 159
- ADC, 10
- ADC conversion, 360
- ADC process, 360
- ADC, SAR converter, 369
- ADC14, 367
- AES accelerator, 11
- AES256 Accelerator Module, 461
- ALU, 3
- annunciator, 157
- arithmetic operations, 78
- ASCII, 402

- background research, 476
- Bardeen, Brattain and Schockley, 2
- bare metal, 70
- battery capacity, 233
- battery operation, 233
- battery, primary and secondary, 235
- Baud rate, 401
- bilge pump, 158
- binary number system, 194
- bit twiddling, 80
- Boone, Gary, 3
- BoosterPacks, 12
- bottom-up approach, 479

- CCS Cloud, 15
- checksum, 213
- Clock System, 8
- clock system, 274
- Code Composer Studio, 15
- code re-use, 481
- comments, 66
- COMP E, 392
- comparator, 392
- CRC checksum, 452
- CRC generator, 11
- CRC polynomial, 452
- CRC32, 213
- CRC32 module, 453
- current sink, 95
- current source, 95

- DAC converter, 367
- Dagu Magician robot, 47
- data integrity, 452
- data logging, 198
- DC fan, 158
- DC motor, 135
- decoder, 166
- design, 478
- design process, 476
- DF robot, 518
- digital-to-analog converter (DAC), 7
- Direct Memory Access (DMA), 10, 208
- documentation, 481
- dot matrix display, 127
- Driver Library, 16
- duty cycle, 271

- Educational Booster Pack MkII, 162, 185
- Educational BoosterPack MKII, 373
- EEPROM, 4
- elapsed time, 274
- electrical specifications, 94
- electromagnetic interference (EMI), 450
- electrostatic discharge (ESD), 450
- embedded system, 476
- EMI noise suppression, 450
- EMI reduction strategies, 450
- emulator, 12
- encoder, absolute, 108
- encoder, incremental, 108
- encoder, quadrature, 109
- encoding, 366
- Energia, 15, 24
- Energia Development Environment, 25
- Energia MT, 24
- EnergyTrace, 12
- enhanced Universal Serial Communication Interface (eUSCI), 400
- ENIAC, 2
- eUSCI A module, 400
- eUSCI B module, 400
- fireworks, 178
- flash memory, 7
- floating point unit, 10
- frequency, 270
- full duplex, 402
- function body, 69
- function call, 68
- function prototypes, 68
- functions, 67
- Grove Starter Kit, 164, 185
- gyroscope, 113
- H-bridge, 140, 142
- hardware multiplier, 10
- Harvard architecture, 196
- HC CMOS, 97
- I²C module, 432
- I2C, 8
- ideal op-amp, 118
- if-else, 85
- include files, 67
- inertial measurement unit, 113
- inertial measurement unit (IMU), 113
- input capture, 272
- input devices, 99
- input/output ports, 8
- integrated circuit, 2
- interrupt handler, 74
- interrupt processing, 339
- interrupt service routine (ISR), 339, 343
- interrupt system, 339
- interrupt theory, 339
- interrupts, 339
- interrupts, MSP432, 341
- interval timer, 289
- IR sensor, 109
- IR sensors, 48
- IrDA, 10
- IrDA protocol, 400
- joystick, 109, 496, 499
- jumper isolation block, 12
- keypad, 101, 142
- Kilby, Jack, 2
- laser light show, 137
- LaunchPad, 11
- LCD, serial, 134
- LDO regulator, 227
- LED biasing, 121
- LED cube, 165, 169

- LED cube, construction, 166
- LED, seven segment, 123
- LED, tri-state, 125
- light-emitting diode (LED), 121
- linear feedback shift register (LFSR), 453
- liquid crystal display (LCD), 133
- logical operations, 79
- loop, 81
- loop(), 25
- low power modes, 7
- low-power operating modes, 225

- main program, 76
- Mauchly and Eckert, 2
- MAX232, 402
- maze, 47
- memory address bus, 192
- memory concepts, 192
- memory data bus, 192
- memory map, 200
- memory map, MSP432, 200
- memory operations, 194
- memory pointers, 198
- memory, flash, 200
- microcontroller, 1, 3
- MMC/SD, 217
- MMC/SD card, 197, 217, 482
- MOSFET, 139
- motor operating parameters, 140
- motor, vibrating, 158
- mountain maze, 516, 531
- MSPWare, 16
- multitasking, 24

- noise, 450
- non-volatile memory, 196
- NRZ format, 402
- Nyquist rate, 361
- octal buffer, 166

- op-amp, 118
- operating mode transitions, 230
- operating modes, 229
- operating parameters, 93
- operational amplifier, 118
- operators, 77
- optical encoder, 107
- optical isolation, 157
- output compare, 272
- output device, 121
- output timer, 273

- parity, 402
- part numbering, 11
- PCM, 228
- period, 271
- photodiode, 117
- pin assignments, 27
- pointers, 198
- port configuration, 69
- Power Control Module (PCM), 226
- Power Supply System (PSS), 226
- PowerSwitch Tail II, 159
- pre-design, 477
- preliminary testing, 480
- program constants, 74
- program constructs, 81
- programming in C, 63
- project description, 476
- prototyping, 480
- PSS, 227
- pulse width modulation (PWM), 271

- quantization, 363

- RAM, 4, 196
- RAM memory, 7
- real-time clock (RTC), 10, 325
- resets, 338
- resolution, 364
- RF connectivity, 7

- Rijndael algorithm, 461
- RISC architecture, 6
- robot IR sensors, 47
- robot platform, 48
- robot steering, 47
- robot, autonomous, 518
- robot, submersible, 491
- ROM, 4, 196
- ROV, 491
- ROV buoyancy, 494
- ROV control housing, 514
- ROV structure, 494
- RS-232, 402
- RTC C, 325

- sampling, 361
- SeaPerch, 491, 499
- SeaPerch control system, 499
- SeaPerch ROV, 492
- sensor, analog, 109
- sensor, digital, 107
- sensor, flex, 109
- sensor, level, 113
- sensor, ultrasonic, 113
- sensors, 107
- serial communications, 399
- serial peripheral interface, 417
- serial peripheral interface (SPI), 8
- servo motor, 135
- servos, Futaba, 137
- setup(), 25
- simplex communication, 401
- sketch, 26, 30
- sketchbook, 26
- solenoid, 150
- sonalert, 157
- speech chip, SPO-512, 410
- SPI, 417
- SPI features, 418

- SPI hardware, 419
- SPI operation, 417
- SPI registers, 421
- stepper motor, 136, 150
- strip LED, 38
- switch, 87
- switch debouncing, 101
- switch interface, 99
- switch, hexadecimal, 99
- switches, 99

- tact switch, 99
- test plan, 480
- TI Resource Explorer, 15
- time base, 272
- Timer 32, 300
- Timer A, 309
- timer channels, 8
- timing parameters, 270
- TMS 1000, 3
- top-down approach, 479
- top-down design, bottom-up implementation, 479
- transducer interface, 115
- transistor, 2

- UART, 8, 403
- UART character format, 406
- UART features, 403
- UART interrupts, 407
- UART module, 404
- UART registers, 408
- ultra-low power, 225
- ultra-low power consumption, 7
- UML, 479
- UML activity diagram, 50, 479
- Unified Modeling Language (UML), 478
- UNIVAC I, 2

Universal Serial Communication Interfaces
(USCI), 8

vacuum tube, 2

variable size, 75

variables, 75

volatile memory, 196

von Neumann architecture, 196

watchdog timer, 289

WDT modes, 289

weather station, 481

while, 82

