MORGAN & CLAYPOOL PUBLISHERS

Microchip AVR® Microcontroller Primer

Programming and Interfacing, Third Edition

Steven F. Barrett
Daniel J. Pack

Synthesis Lectures on Digital Circuits and Systems

Mitchell A. Thornton, *Series Editor*

Microchip AVR® Microcontroller Primer: Programming and Interfacing

Third Edition

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.

Microchip AVR® Microcontroller Primer: Programming and Interfacing, Third Edition
Steven F. Barrett and Daniel J. Pack
2019

Synthesis of Quantum Circuits vs. Synthesis of Classical Reversible Circuits
Alexis De Vos, Stijn De Baerdemacker, and Yvan Van Rentergen
2018

Boolean Differential Calculus
Bernd Steinbach and Christian Posthoff
2017

Embedded Systems Design with Texas Instruments MSP432 32-bit Processor
Dung Dang, Daniel J. Pack, and Steven F. Barrett
2016

Fundamentals of Electronics: Book 4 Oscillators and Advanced Electronics Topics
Thomas F. Schubert and Ernest M. Kim
2016

Fundamentals of Electronics: Book 3 Active Filters and Amplifier Frequency
Thomas F. Schubert and Ernest M. Kim
2016

Bad to the Bone: Crafting Electronic Systems with BeagleBone and BeagleBone Black, Second Edition
Steven F. Barrett and Jason Kridner
2015

Fundamentals of Electronics: Book 2 Amplifiers: Analysis and Design
Thomas F. Schubert 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

Microchip AVR® Microcontroller Primer:
Programming and Interfacing, Third Edition
Steven F. Barrett and Daniel J. Pack

www.morganclaypool.com

ISBN: 9781681732046 paperback
ISBN: 9781681732053 ebook
ISBN: 9781681736235 hardcover

DOI 10.2200/S00803ED3V01Y201709DCS053

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS

Lecture #53
Series Editor: Mitchell A. Thornton, *Southern Methodist University*
Series ISSN
Print 1932-3166 Electronic 1932-3174

Microchip AVR® Microcontroller Primer: Programming and Interfacing

Third Edition

Steven F. Barrett
University of Wyoming, Laramie, WY

Daniel J. Pack
University of Tennessee Chattanooga, TN

SYNTHESIS LECTURES ON DIGITAL CIRCUITS AND SYSTEMS #53

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This textbook provides practicing scientists and engineers a primer on the Microchip AVR® microcontroller. The revised title of this book reflects the 2016 Microchip Technology acquisition of Atmel Corporation. In this third edition we highlight the popular ATmega164 microcontroller and other pin-for-pin controllers in the family with a complement of flash memory up to 128 KB. The third edition also provides an update on Atmel Studio, programming with a USB pod, the gcc compiler, the ImageCraft JumpStart C for AVR compiler, the Two-Wire Interface (TWI), and multiple examples at both the subsystem and system level. Our approach is to provide readers with the fundamental skills to quickly set up and operate with this internationally popular microcontroller. We cover the main subsystems aboard the ATmega164, providing a short theory section followed by a description of the related microcontroller subsystem with accompanying hardware and software to operate the subsystem. In all examples, we use the C programming language. We include a detailed chapter describing how to interface the microcontroller to a wide variety of input and output devices and conclude with several system level examples including a special effects light-emitting diode cube, autonomous robots, a multi-function weather station, and a motor speed control system.

KEYWORDS

microchip microcontroller, Microchip AVR®, ATmega164, microcontroller interfacing, embedded systems design

To our families

Contents

Preface

In 2006, Morgan & Claypool Publishers (M&C) released our textbook *Microcontrollers Fundamentals for Engineers and Scientists*. The purpose of this textbook was to provide practicing scientists and engineers a tutorial on the fundamental concepts and the use of microcontrollers. The textbook presented the fundamental concepts common to all microcontrollers. Our goal for writing this follow–on book is to present details on a specific microcontroller family, the Microchip AVR® ATmega164 Microcontroller family. This family includes the ATmega164 microcontroller which is equipped with four 8-bit input/output ports, a plethora of subsystems, and a 16K–bytes flash program memory. Other microcontrollers in the family include the pin–for–pin compatible ATmega324 (32K–bytes program memory), ATmega644 (64K–bytes program memory), and the ATmega1284 (128K–bytes program memory).

Why Microchip microcontrollers? There are many excellent international companies that produce microcontrollers. Some of the highlights of the Microchip AVR® microcontroller line include:

- high performance coupled with low power consumption,
- outstanding flash memory technology,
- reduced instruction set computer Harvard Architecture,
- single-cycle instruction execution,
- wide variety of operating voltages (1.8–5.5 VDC),
- architecture designed for the C language,
- one set of development tools for the entire AVR® microcontroller line, and
- in–system programming, debugging, and verification capability.

Although all of these features are extremely important, we have chosen to focus on the Microchip AVR® microcontroller line of microcontrollers for this primer for a number of other related reasons.

- The learning curve for Microchip microcontrollers is gentle. If this is your first exposure to microcontrollers, you will quickly come up to speed on microcontroller programming and interfacing. If you already know another line of processors, you can quickly apply your knowledge to this powerful line of 8-bit processors.

- It is relatively inexpensive to get started with the Microchip AVR® microcontroller line. The microcontrollers themselves are inexpensive, and the compilers and programming hardware and software are easily affordable.
- The AVR® microcontroller line provides a full range of processing power, from small, yet powerful 8-pin processors to complex 100-pin processors. The same compiler and programming hardware may be used with a wide variety of microcontrollers.
- Many of the AVR® microcontrollers are available in dual inline package (DIP), which makes them readily useable on a printed circuit board prototype (e.g., capstone design projects).
- Many of the microcontrollers in the AVR® microcontroller line are pin-for-pin compatible with one another. This allows you to easily move up and down the AVR® microcontroller line as your project becomes better defined.
- Microchip has documentation available for every microcontroller at your fingertips. Simply visit `www.microchip.com`. Furthermore, Microchip customer support is good and responsive.
- There is worldwide interest in the AVR® microcontroller line. We would be remiss to not mention AVR Freaks®. This is a dedicated, international group of AVR® microcontroller experts who share their expertise online with other high–power users and novices alike.

Approach of the book

If this is your first exposure to microcontrollers, we highly recommend that you read first our M&C textbook, *Microcontrollers Fundamentals for Engineers and Scientists*. It will provide you the background information necessary to fully appreciate the contents of this textbook. This textbook picks up where the first one left off. We have received permission from M&C to include some of the background material from the first textbook in this text to allow for a complete stand-alone product.

Our approach in this textbook is to provide you with the fundamental skills to quickly get set up and operate an Microchip microcontroller. We have chosen to use the AVR® ATmega164 as a representative sample of the AVR® microcontroller line (more on this processor later). The knowledge you gain on the ATmega164 can be easily translated to every other microcontroller in the AVR® microcontroller line.

The M&C textbooks are designed to be short tutorials on a given topic. Therefore, our treatment of each topic will provide a short theory section followed by a description of the related microcontroller subsystem with accompanying hardware and software to exercise the subsystem. In all examples, we use the C programming language. There are many excellent C compilers available for the Microchip AVR® microcontoller line. We have chosen the Atmel Studio integrated development platform (IDP) for its short learning curve and ease of use. We also provide

examples using the ImageCraft JumpStart C for AVR compiler www.imagecraft.com. We use the USB compatible Microchip AVR Dragon employing In–System Programming (ISP) techniques.

NEW IN THE THIRD EDITION

The third edition provides the following updated and expanded features:

- examples provided for the ATmega164. Easily ported to other compatible ATmega processors;
- multiple new, fully worked examples;
- ISP programming with the Atmel AVR Dragon;
- examples using serial UART configured LCD, voice synthesis chip, and serial monitor on host PC;
- tri–color light-emitting diode (LED) strip controlled by SPI system;
- detailed example using TWI (I2C) system;
- detailed heart beat monitor example using input capture system;
- controlling AC load using PowerSwitch Tail II;
- extended examples for 3D special effects LED cube, maze robots, a weather station, motor speed control circuit; and
- support provided for both the Atmel AVR Studio gcc compiler and the ImageCraft JumpStart C for AVR compiler.

Steven F. Barrett and Daniel J. Pack
Laramie, WY and Chattanooga, TN
September 2019

Acknowledgments

There have been many people involved in the conception and production of this book. In 2005, Joel Claypool of Morgan & Claypool Publishers, invited us to write a book on microcontrollers for his new series titled "Synthesis Lectures on Digital Circuits and Systems." The result was the book *Microcontrollers Fundamentals for Engineers and Scientists.* Since then we have been regular contributors to the series. Our goal has been to provide the fundamental concepts common to all microcontrollers and then apply the concepts to the specific microcontroller under discussion. We believe that once you have mastered these fundamental concepts, they are easily transportable to different processors. As with many other projects, he 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. He has provided many novice writers the opportunity to become published authors. His vision and expertise in the publishing world made this book possible. We thank Sara Kreisman of Rambling Rose Press, Inc. for her editorial expertise. 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.

We also thank Sparkfun, Adafruit, ImageCraft, and Microchip for their permission to use their copyrighted material and screenshots throughout the text. Several Microchip acknowledgments are in order.

- This book contains copyrighted material of Microchip Technology Incorporated replicated with permission. All rights reserved. No further replications may be made without Microchip Technology Inc's prior written consent.
- *Microchip AVR® Microcontroller Primer: Programming and Interfacing, Third Edition* is an independent publication and is not affiliated with, nor has it been authorized, sponsored, or otherwise approved by Microchip.

Most of all, we thank our families. Our work could not have come to fruition without the sacrifice and encouragement of our families over the past fifteen plus years. Without you, none of this would matter. We love you!

Steven F. Barrett and Daniel J. Pack
Laramie, WY and Chattanooga, TN
September 2019

CHAPTER 1

Microchip AVR® Architecture Overview

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

- provide an overview of the RISC architecture of the ATmega164;
- describe the different ATmega164 memory components and their applications;
- explain the ATmega164 internal subsystems and their applications;
- highlight the operating parameters of the ATmega164; and
- summarize the special ATmega164 features.

1.1 ATMEGA164 ARCHITECTURE OVERVIEW

In this section, we describe the overall architecture of the Microchip AVR ATmega164. We begin with an introduction to the concept of the reduced instruction set computer (RISC) and briefly describe the Microchip Assembly Language Instruction Set. We program mainly in C throughout the course of the book. We then provide a detailed description of the ATmega164 hardware architecture.

1.1.1 REDUCED INSTRUCTION SET COMPUTER

In our first Morgan & Claypool (M&C) textbook [Barrett and Pack, 2006], we described a microcontroller as an entire computer system contained within a single integrated circuit or chip. Microcontroller operation is controlled by a user-written program interacting with the fixed hardware architecture resident within the microcontroller. A specific microcontroller architecture can be categorized as accumulator-based, register-based, stack-based, or a pipeline architecture.

The Microchip ATmega164 is a register-based architecture. In this type of architecture, both operands of an operation are stored in registers collocated with the central processing unit (CPU). This means that before an operation is performed, the computer loads all necessary data for the operation to its CPU. The result of the operation is also stored in a register. During program execution, the CPU interacts with the register set and minimizes slower memory accesses for both instructions and data. Memory accesses are typically handled as background operations.

Coupled with the register-based architecture is an instruction set based on the RISC concept. A RISC processor is equipped with a complement of very simple and efficient basic operations. More complex instructions are built up from these basic operations. This allows for efficient program operation. The Microchip ATmega164 is equipped with 131 RISC-type instructions. Most can be executed in a single clock cycle. The ATmega164 is also equipped with additional hardware to allow for the multiplication operation in two clock cycles. In many other microcontroller architectures, multiplication typically requires many more clock cycles. For additional information on the RISC architecture, the interested reader is referred to Hennessy and Patterson [2003].

The Microchip ATmega164 is equipped with 32 general-purpose 8-bit registers that are tightly coupled to the processor's arithmetic logic unit within the CPU. Also, the processor is designed following the Harvard Architecture format. That is, it is equipped with separate, dedicated memories and buses for program instructions and data information. The register-based Harvard Architecture coupled with the RISC-based instruction set allows for fast and efficient program execution and allows the processor to complete an assembly language instruction every clock cycle. Microchip indicates the ATmega164 can execute 20 million instructions per second when operating at a clock speed of 20 MHz and with the supply voltage between 4.5 and 5.5 VDC.

1.1.2 ASSEMBLY LANGUAGE INSTRUCTION SET

An instruction set is a group of instructions a machine "understands" how to execute. A large number of instructions provide flexibility but require more complex hardware. Thus, an instruction set is unique for a given hardware configuration and cannot be used with another hardware configuration. Microchip has equipped the ATmega164 with 131 different instructions.

For the most efficient and fast execution of a given microcontroller, assembly language should be used. Assembly language instructions are written to efficiently interact with a specific microcontroller's resident hardware. To effectively use the assembly language, the programmer must be thoroughly familiar with the low-level architecture details of the controller. Furthermore, the learning curve for a given assembly language is quite steep and lessons learned do not always transfer to another microcontroller. This is part of the reason programmers prefer to use a high-level language over an assembly language.

We program the Microchip ATmega164 using the C language throughout the text. The C programming language allows for direct control of microcontroller hardware at the register level while being portable to other microcontrollers in the AVR® microcontroller line. When a C program is compiled during the software development process, the program is first converted to assembly language and then to the machine code for the specific microcontroller.

We must emphasize that programming in C is not better than assembly language or vice versa. Both approaches have their inherent advantages and disadvantages. We have chosen to use C in this textbook for the reasons previously discussed.

1.1.3 ATMEGA164 ARCHITECTURE OVERVIEW

We have chosen the ATmega164 as a representative of the Microchip AVR line of microcontrollers. Lessons learned with the ATmega164 may be easily adapted to all other processors in the AVR microcontroller line. A block diagram of the Microchip ATmega164's architecture is provided in Figure 1.1.

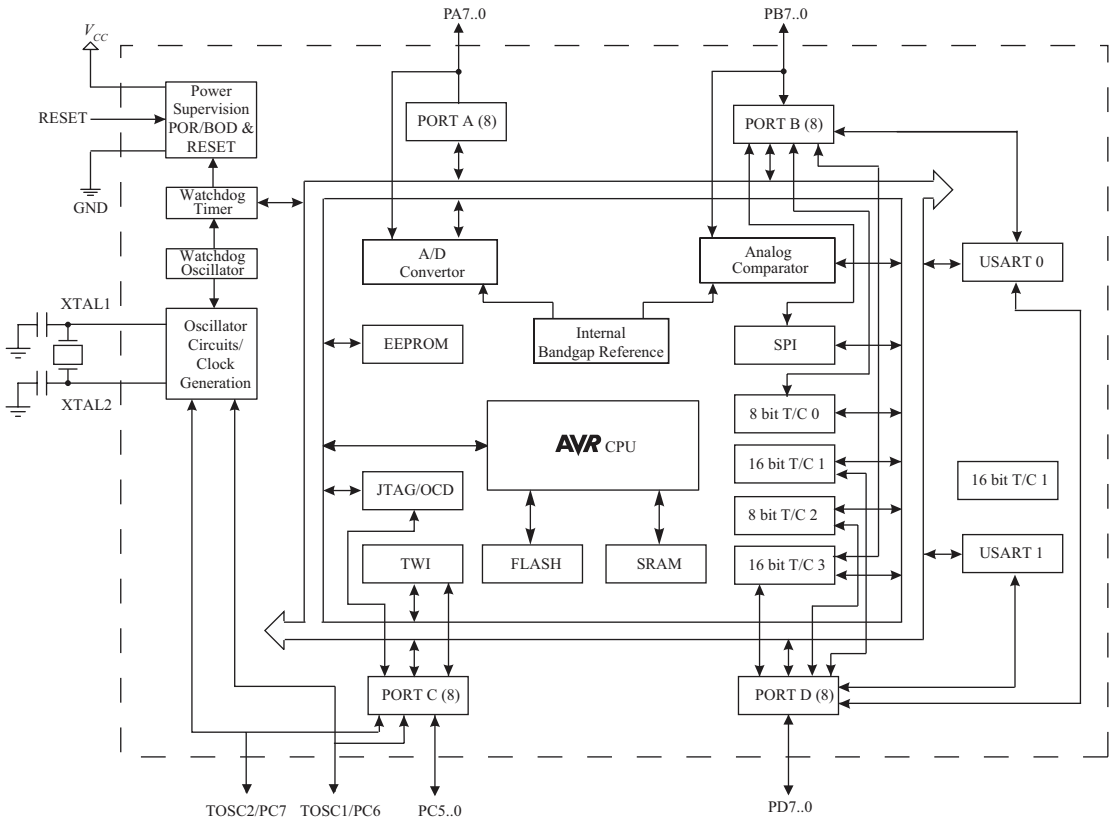


Figure 1.1: Microchip AVR ATmega164 block diagram. Figure used with permission of Microchip. All rights reserved.

As can be seen from the figure, the ATmega164 has external connections for power supplies (VCC, GND, AVCC, and AREF), an external time base (XTAL1 and XTAL2) input pins to drive its clocks, processor reset (active low RESET), and four 8-bit ports (PA0-PA7, PC0-PC7, PB0-PB7, and PD0-PD7), which are used to interact with the external world. These ports may be used as general purpose digital input/output (I/O) ports or they may be used for their alternate functions. The ports are interconnected with the ATmega164's CPU and internal subsystems via an internal bus. The ATmega164 also contains a timer subsystem, an analog-to-digital

converter (ADC), an interrupt subsystem, memory components, and a serial communication subsystem.

In the next several sections, we briefly describe each of these internal subsystems shown in the figure. Detailed descriptions of selected subsystem operation and programming appear in latter parts of the book. Since we cannot cover all features of the microcontroller due to limited space, we focus on the primary functional components of the microcontroller to fulfill the purpose of this book as a basic primer to the ATmega164.

1.2 NONVOLATILE AND DATA MEMORIES

The ATmega164 is equipped with three main memory sections: flash electrically erasable programmable read-only memory (EEPROM), static random access memory (SRAM), and byte-addressable EEPROM for data storage. We discuss each memory component in turn.

1.2.1 IN-SYSTEM PROGRAMMABLE FLASH EEPROM

Bulk programmable flash EEPROM is typically used to store programs. It can be erased and programmed as a single unit. Also, should a program require a large table of constants, it may be included as a global variable within a program and programmed into flash EEPROM with the rest of the program. Flash EEPROM is nonvolatile, meaning memory contents are retained when microcontroller power is lost. The ATmega164 is equipped with 16 KB of onboard reprogrammable flash memory. This memory component is organized into 8 K locations, with 16 bits at each location.

The flash EEPROM is in-system programmable. In-system programmability (ISP) means the microcontroller can be programmed while resident within a circuit. It does not have to be removed from the circuit for programming. Instead, a host personal computer (PC), connected via a specialized programming cable and pod, downloads the program to the microcontroller. Alternately, the microcontroller can be programmed outside its resident circuit using a flash programmer board. We employ the AVR Dragon for ISP programming the ATmega164. This development board is readily available from a number of suppliers. It may be used to program many different microcontrollers in the ATmega and ATtiny product families.

1.2.2 BYTE-ADDRESSABLE EEPROM

Byte-addressable memory is used to permanently store and recall variables during program execution. It too is nonvolatile. That is, it retains its contents even when power is not available. It is especially useful for logging system malfunctions and fault data during program execution. It is also useful for storing data that must be retained during a power failure but might need to be changed periodically. Examples where this type of memory is used are found in applications to store system parameters, electronic lock combinations, and automatic garage door electronic unlock sequences. The ATmega164 is equipped with 512 B of byte-addressable EEPROM.

1.2.3 STATIC RANDOM ACCESS MEMORY

SRAM is volatile. That is, if the microcontroller loses power, the contents of SRAM memory are lost. It can be written to and read from during program execution. The ATmega164 is equipped with 1000 B (actually 1120) of SRAM. A small portion (96 locations) of the SRAM is set aside for the general-purpose registers used by the central processing unit (CPU) and also for the input/output (I/O) and peripheral subsystems aboard the microcontroller. A complete ATmega164 register listing and accompanying header file is provided in the Appendices. During program execution, RAM is used to store global variables, support dynamic memory allocation of variables, and provide a location for the stack (to be discussed later).

1.2.4 PROGRAMMABLE LOCK BITS

To provide for memory security from tampering, the ATmega164 is equipped with memory lock bits. These lock bits are programmed using the Microchip AVR Dragon. The lock bits may be configured for the following options:

- no memory lock features enabled;
- no further programming of memory is allowed using parallel or serial programming techniques; or
- no further programming or verification of memory is allowed using parallel or serial programming techniques.

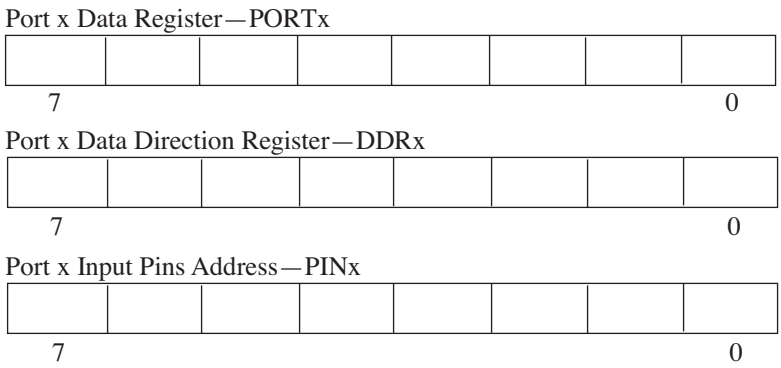
1.3 PORT SYSTEM

The Microchip ATmega164 is equipped with four 8-bit general-purpose, digital I/O ports designated PORTA, PORTB, PORTC, and PORTD. All of these ports also have alternate functions, which are described later. In this section, we concentrate on the basic digital I/O port features.

As shown in Figure 1.2, each port has three registers associated with it:

- Data Register (PORTx) used to write output data to the port,
- Data Direction Register (DDRx) used to set a specific port pin to either output (1) or input (0), and
- Input Pin Address (PINx) used to read input data from the port.

Figure 1.2b describes the settings required to configure a specific port pin to either input or output. If selected for input, the pin may be selected for either an input pin or to operate in the high-impedance (Hi-Z) mode. In Hi-Z mode, the input appears as high impedance to a particular pin. If selected for output, the pin may be further configured for either logic low or logic high.



(a)

DDxn	PORTxn	I/O	Comment
0	0	Input	Tri-state (Hi-Z)
0	1	Input	Source current if externally pulled low
1	0	Output	Output Low (Sink)
1	1	Output	Output High (Source)

x: port designator (A, B, C, D)
n: pin designator (0–7)

(b)

Figure 1.2: ATmega164 port configuration registers: (a) port-associated registers and (b) port pin configuration.

Port pins are usually configured at the beginning of a program for either input or output, and their initial values are then set. Usually, all eight pins for a given port are configured simultaneously, even if all eight pins may not be used during a program execution. A code example to configure a port is shown below. Note that because we are using the C programming language with a compiler include file, the register contents are simply referred to by name. Note that the data direction register (DDRx) is first used to set the pins as either input or output, and then the data register (PORTx) is used to set the initial value of the output port pins. It is a good design practice to configure unused microcontroller pins as outputs. This prevents them from acting as a source for noise input.

```
//*************************************************************
//initialize_ports: provides initial configuration for I/O ports
//*************************************************************

void initialize_ports(void)
{
DDRA=0xfc;  //set PORTA[7:2] as output, PORTA[1:0]
            //as input (1111_1100)
PORTA=0x03; //initialize PORTA[7:2] low, PORTA[1:0]
            //current source
DDRB=0xa0;  //PORTB[7:4] as output, set PORTB[3:0] as input
PORTB=0x00; //disable PORTB pull-up resistors
DDRC=0xff;  //set PORTC as output
PORTC=0x00; //initialize low
DDRD=0xff;  //set PORTD as output
PORTD=0x00; //initialize low
}
```

To read the value from a port pin configured as input, the following code could be used. Note the variable used to read the value from the input pins is declared as an unsigned char because both the port and this variable type are 8 bits wide. A brief introduction to programming in C is provided in Chapter 2.

```
unsigned char   new_PORTB;            //new values of PORTB

:
:
:

new_PORTB = PINB;                     //read PORTB
```

1.4 PERIPHERAL FEATURES INTERNAL SUBSYSTEMS

In this section, a brief overview of the peripheral features of the ATmega164 is provided. It should be emphasized that these features are the internal subsystems contained within the confines of the microcontroller chip. These built-in features allow complex and sophisticated tasks to be accomplished by the microcontroller.

1.4.1 TIME BASE

The microcontroller is a complex synchronous state machine. It responds to program steps in a sequential manner as dictated by a user-written program. The microcontroller sequences through a predictable fetch, decode, and execute sequence. Each unique assembly language program instruction issues a series of signals to control the microcontroller hardware to accomplish instruction related operations.

The speed at which a microcontroller sequences through these actions is controlled by a precise time base, called the clock. The clock source is routed throughout the microcontroller to be used as a common time base for all peripheral subsystems. The ATmega164 may be clocked internally, using a user-selectable resistor capacitor (RC) time base, operating at approximately 8 MHz or externally using a timing crystal or resonator. The RC internal time base is selected using programmable fuse bits. We show its use in the application section.

To provide for a wider range of frequency selections, an external source may be used. The external time sources, in order of increasing accuracy and stability, are an external RC network, a ceramic resonator, or a crystal oscillator. The system designer chooses the time base frequency and clock source device appropriate for the application at hand. Generally speaking, a microcontroller is operated at the lowest possible frequency for a given application since clock speed is linearly related to power consumption. The clock source may be divided down by a user selectable value of 1, 2, 4, 8, 16, 32, 64, 128, or 256. The clock divide by value is set into the clock prescaler select (CLKPS) register or a divide by 8 value may be set with a fuse.

1.4.2 TIMING SUBSYSTEM

The ATmega164 is equipped with a complement of timers that allows the user to generate a precision output signal, measure the characteristics (period, duty cycle, frequency) of an incoming digital signal, or count external events. Specifically, the ATmega164 is equipped with two 8-bit timer/counters and one 16-bit counter. We discuss the operation, programming, and application of the timing system in Chapter 6.

1.4.3 PULSE WIDTH MODULATION CHANNELS

A pulse width modulated (PWM) signal is characterized by a fixed frequency and a varying duty cycle. Duty cycle is the percentage of time a repetitive signal is logic high during the signal

period. It may be formally expressed as

$$\text{duty cycle (\%)} = (\text{on time/period}) \times (100\%).$$

The ATmega164 family is equipped with up to six PWM channels. The PWM channels coupled with the flexibility of dividing the time base down to different PWM subsystem clock source frequencies allow the user to generate a wide variety of PWM signals, from relatively high-frequency, low-duty cycle signals to relatively low-frequency, high-duty cycle signals. PWM signals are used in a wide variety of applications, including controlling the position of a servo motor and controlling the speed of a DC motor. We discuss the operation, programming, and application of the PWM system in Chapter 6.

1.4.4 SERIAL COMMUNICATIONS

The ATmega164 is equipped with a host of different serial communication subsystems, including the universal synchronous and asynchronous serial receiver and transmitter (USART), the serial peripheral interface (SPI), and the two-wire serial interface (TWI). What all of these systems have in common is the serial transmission of data. In a serial communications transmission scheme, data are sent a single bit at a time from transmitter to receiver.

Serial USART

The serial USART is used for full duplex (two-way) communication between a receiver and transmitter. This is accomplished by equipping the ATmega164 with independent hardware for the transmitter and receiver. The USART is typically used for asynchronous communication. That is, there is not a common clock between the transmitter and receiver to keep them synchronized with one another. To maintain synchronization between the transmitter and receiver, framing start and stop bits are used at the beginning and end of each data byte in a transmission sequence.

The ATmega164 USART is quite flexible. It has two independent channels designated USART0 and USART1. It has the capability to be set to a variety of data transmission rates known as the baud (bits per second) rate. The USART may also be set for data bit widths of 5–9 bits with one or two stop bits. Furthermore, the ATmega164 is equipped with a hardware-generated parity bit (even or odd) and parity check hardware at the receiver. A single parity bit allows for the detection of a single bit error within a byte of data. The USART may also be configured to operate in a synchronous mode. The flexible configuration of the USART system allows it to be used with a wide variety of peripherals including serial configured liquid crystal displays (LCDs) and a speech chip. The USART is also used to communicate microcontroller status to a host PC. We discuss the operation, programming, and application of the USART in Chapter 3.

Serial Peripheral Interface

The ATmega164 serial peripheral interface (SPI) can also be used for two-way serial communication between a transmitter and a receiver. In the SPI system, the transmitter and receiver share a common clock source. This requires an additional clock line between the transmitter and receiver but allows for higher data transmission rates, as compared with the USART.

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. The transmitter is designated the master because it provides the synchronizing clock source between the transmitter and the receiver. The receiver is designated as the slave. The SPI may be used to communicate with external peripheral devices such as large seven segment displays, digital-to-analog converts (DACs), and LED strips. We discuss the operation, programming, and application of the SPI in Chapter 3.

Two-Wire Serial Interface

The TWI subsystem allows the system designer to network a number of related devices (microcontrollers, transducers, displays, memory storage, etc.) together into a system using a two-wire interconnecting scheme. The TWI allows a maximum of 128 devices to be connected together. Each device has its own unique address and may both transmit and receive over the two-wire bus at frequencies up to 400 kHz. This allows the device to freely exchange information with other devices in the network within a small area.

1.4.5 ANALOG-TO-DIGITAL CONVERTER

The ATmega164 is equipped with an eight-channel ADC subsystem. The ADC converts an analog signal from the outside world into a binary representation suitable for use by the microcontroller. The ATmega164 ADC has 10-bit resolution. This means that an analog voltage between 0 and 5 V will be encoded into one of 1024 binary representations between $(000)_{16}$ and $(3FF)_{16}$. This provides the ATmega164 with a voltage resolution of approximately 4.88 mV. It is important to emphasize the ADC clock frequency must be set to a value between 50 and 200 kHz to obtain accurate results. We discuss the operation, programming, and application of the ADC in Chapter 4.

1.4.6 INTERRUPTS

The normal execution of a program follows a designated sequence of instructions. However, sometimes, this normal sequence of events must be interrupted to respond to high-priority faults, and status generated from both inside and outside the microcontroller. When these higher-priority events occur, the microcontroller must temporarily suspend its normal operation and execute event specific actions called an interrupt service routine. Once the higher priority event has been serviced, the microcontroller returns and continues processing the normal program.