Designing with Microcontrollers

An Online Continuing Education Course for Engineers

Course Number: E-3052

Credit: 3 Hours / 3 PDH / 3 CPD

Designing with Microcontrollers

Mark Allen Strain, P.E.

Introduction

In my course entitled "The Basics of Microcontrollers" I discussed the architecture of microcontrollers. I showed how the central processing unit fetches instructions (or a program) from memory and decomposes the instructions into components that the control unit and the arithmetic logic unit can use to perform the desired operation or function. Here I will discuss how to design a simple circuit incorporating a microcontroller with a small footprint, small pin count, and a small amount of internal memory (both program and data memory). I will give program examples using the C programming language.

Microcontrollers are simply microprocessors that include program and data memory and peripherals such as general-purpose input/output ports, timers, serial communications controllers, analog-to-digital converter, etc.

For this course I will utilize the Atmel AVR series of microcontrollers, specifically the Atmel ATtiny2313A series with 2048 bytes of internal flash program memory and 128 bytes of internal data memory. The Atmel AVR series is one of several different processor options a developer can use. Other example microcontrollers include, but are not limited to the Microchip PIC, Texas Instruments MSP430, Intel 8051, STMicroelectronics STM8, Freescale 68HC11, and multiple versions of the ARM core from many vendors.

Design Considerations

When designing a microcontroller-based system, there are three things that need to be considered: the microcontroller, the compiler, and the device programmer.

Microcontroller

First of all, consider the microcontroller. The processor must be sized appropriately to the desired task. Consider the following parameters:

- bus width (8-bit, 16-bit, 32-bit)
- processor speed
- amount of program and data memory
- amount of input/output pins
- peripherals
- power consumption

The examples in this course perform basic operations, such as controlling an LED, reading a button, and utilizing a timer peripheral and some interrupts. A microcontroller with an 8-bit architecture is sufficient. The 8-bit, 16-bit, and 32-bit architecture nomenclature refers to the width of the bus within the core of the microprocessor. That is, how many bits the core can process at once. An 8-bit machine is sufficient for simple systems, such as the ones exemplified

in this course, as well as thermostats, toys that have LEDs and buttons that need to be controlled and read. Microprocessors with larger bus-widths are used for devices that need more computing horsepower such as cell phones, GPS navigation devices, and MP3 players.

The processor speed will determine how fast an instruction can be executed and how much data can be processed during a given slice of time. For example, a device transmitting and receiving data via a USB port will need to have a faster processor speed (and probably larger bus-width) than a device simply controlling a couple of LEDs and responding to a button. Most microprocessors are clocked by an external clock (or oscillator) which determines the speed of the master clock. Some low-power, low-horsepower microcontrollers may be clocked by an internal oscillator circuit that requires no external components like a crystal or capacitors. The microcontroller used in the examples in this course utilize an internal oscillator circuit contained within the chip.

Memory is also a consideration, both program memory and data memory. Program memory is where the program (or set of instructions specific to the task at hand) is stored. Program memory is persistent and is maintained over power cycles. Data memory is used by a program during execution for the stack and to store variables. Both program memory and data memory may be internal to the microcontroller or external to the chip and accessed by an address/data bus. The program memory must be large enough to store the binary (or compiled source code) for the project. The data memory must be large enough to contain the stack and for all of the variables during program execution.

The number of input/output pins depends on the application or the system that the microcontroller will control. Output pins may control an LED, a motor, a relay, or some pins on an external peripheral device (such as a communications controller). Input pins may be used to read an individual button, or a keypad. In some systems, input/output ports may not be required at all.

Peripherals are those subsystems that interface with the microprocessor and pass data to and from the processor and memory. Peripherals may include systems such as communication devices, such as a UART (universal asynchronous receiver transmitter) or a USB (universal serial bus) controller, timers, pulse-width modulators, and analog-to-digital converters.
Power may or may not be a major consideration. If the processor is controlling or monitoring an industrial process, like monitoring and controlling the temperature of a room, then the device will most likely be plugged into the building's power source. In this case a few extra milliwatts is not a major consideration. However, if the processor is going to control a small handheld device, like a garage door opener remote control or a keyless entry remote, then power consumption is a major concern.

Power consumption needs to be considered from two angles: the amount of power consumed at runtime and while the processor is sleeping. Also, different processors have different levels of power saving modes. Most will let the developer turn off unused peripherals. Some modes will actually halt the clock and resume due to an external signal (like a button push). This power saving feature is useful for devices (such as remote controls) that do not need to do anything until a button is pushed; it then can perform its intended operation and then go back to sleep.

Compiler

The second consideration when designing a microcontroller-based system is what development tools are available. Software needs to be written (whether in assembly or in a higher-level language such as the C programming language). This software needs to be compiled and/or assembled into a binary file that can be loaded onto the device.

The compiler, assembler and linker tool chain need to be considered. Some simple projects can be done in assembly. Most assembler tool chains are free. Most C programs if written properly can be very compact, using very little memory. The use of a compiler (such as a C compiler) allows for ease of design and prevents the developer from having to use processor-specific assembly code instructions. This allows more complex programs to be more easily maintained. Even some of the tiniest microcontrollers are supported by some of the available C compiler tool chains.

Some C compilers have a monetary cost and require some sort of licensing, while others are free (or open source). The tool chain used in this course is from the GNU suite of tools, specifically, WinAVR. It is open source and free.

The linker is usually (almost always) bundled with the compiler/assembler. The job of the linker is to link all of the object code together into a single programming file.

Device Programmer

The third consideration when designing a microcontroller-based system is how to program the system. The software that is written is assembled, compiled and linked, creating a single binary file that needs to be written to the device's nonvolatile memory. The development PC containing the file to be programmed transfers this file to the device programmer via one of the ports of the PC: serial, parallel or USB. The device programmer then transfers the file to the microcontroller's program memory. The file is transferred from the device programmer to the microcontroller usually via a serial interface, like a SPI (serial peripheral interface) or JTAG (Joint Technical Architecture Group) interface.

The device programmer used for proving the examples in this course is the USBASP programmer. It is an inexpensive programmer (supported by the AVRDUDE command line interface) that interfaces to the PC via a USB port.

For high volume production runs, device programmers are not the best solution. Once a system is in production, the program memory chips (usually flash memory) are pre-programmed at the factory or by a third-party by a multi-chip programmer. Or, if a microcontroller is utilized with internal flash memory, the microcontroller may be programmed by at the factory or by a third-party.

Microcontroller

The microcontroller used for the exercises in this course is Atmel ATtiny2313A microcontroller. It is a low power, 8-bit, reduced instruction set (or RISC) microcontroller. The one used here comes in a 20-pin DIP (dual in-line) package which makes it easy to insert into a breadboard for experimentation. It has the following features:

- 2048 bytes internal flash (in-system programmable)
- 128 bytes of internal RAM
- internal oscillator
- 18 programmable input/output lines
- an 8-bit timer with separate prescaler and compare mode
- a 16-bit timer with separate prescaler, compare and capture modes
- 4 pulse-width modulation (PWM) channels
- on-chip analog comparator
- serial communications controller
- USART (universal synchronous/asynchronous receiver transmitter)
- low-power idle, standby, and power down modes
- 1.8 - 5.5 volt operation

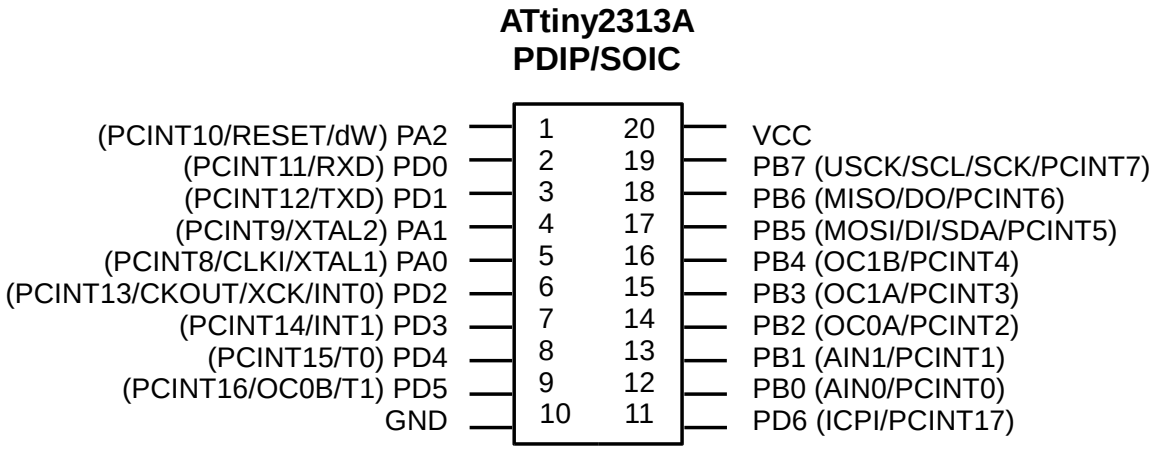


Figure 1 - ATtiny2313A pinout

Another microcontroller in the Atmel AVR series is the ATtiny10. The ATtiny10 device is a powerful microcontroller for its size. It is barely larger than the head of a small nail, but has internal program flash and data memories, and many powerful peripherals, including a 16-bit timer with two PWM channels. Its programming interface is different than the ATtiny2313A processor. It uses a 2-wire serial programming instead of a 3-wire serial interface.

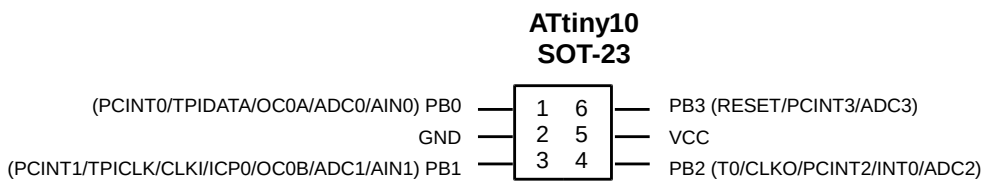


Figure 2 - ATtiny10 pinout

Architecture

The ATtiny2313A is a member of the Atmel AVR series of microcontrollers. It is a reduced instruction set computer (RISC). This means that it has fewer instructions than an x86 processor (for example), but each instruction is powerful, thereby requiring fewer instructions for a complete instruction set. The RISC processors are more code efficient than older-model processors. Many of the instructions may be executed in a single clock cycle.

The AVR employs a Harvard architecture. Therefore, the core interfaces to program and data memories using separate busses. With a Harvard architecture program and data memories can be accessed simultaneously. While an instruction is being executed, the next instruction is fetched from the program memory. Many microcontrollers employ a Harvard architecture to speed processing (with a lower clock rate) by allowing simultaneous access of program memory and data memory. Since the data and program memory have separate busses to the core, there are no bus collision problems. This feature makes a processor designed with a Harvard architecture faster than a similar processor designed with a von Neumann architecture (where data and program instructions are accessed from the same memory device across the same bus).

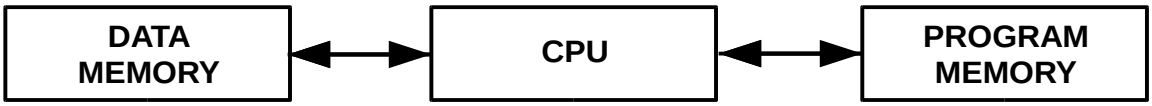
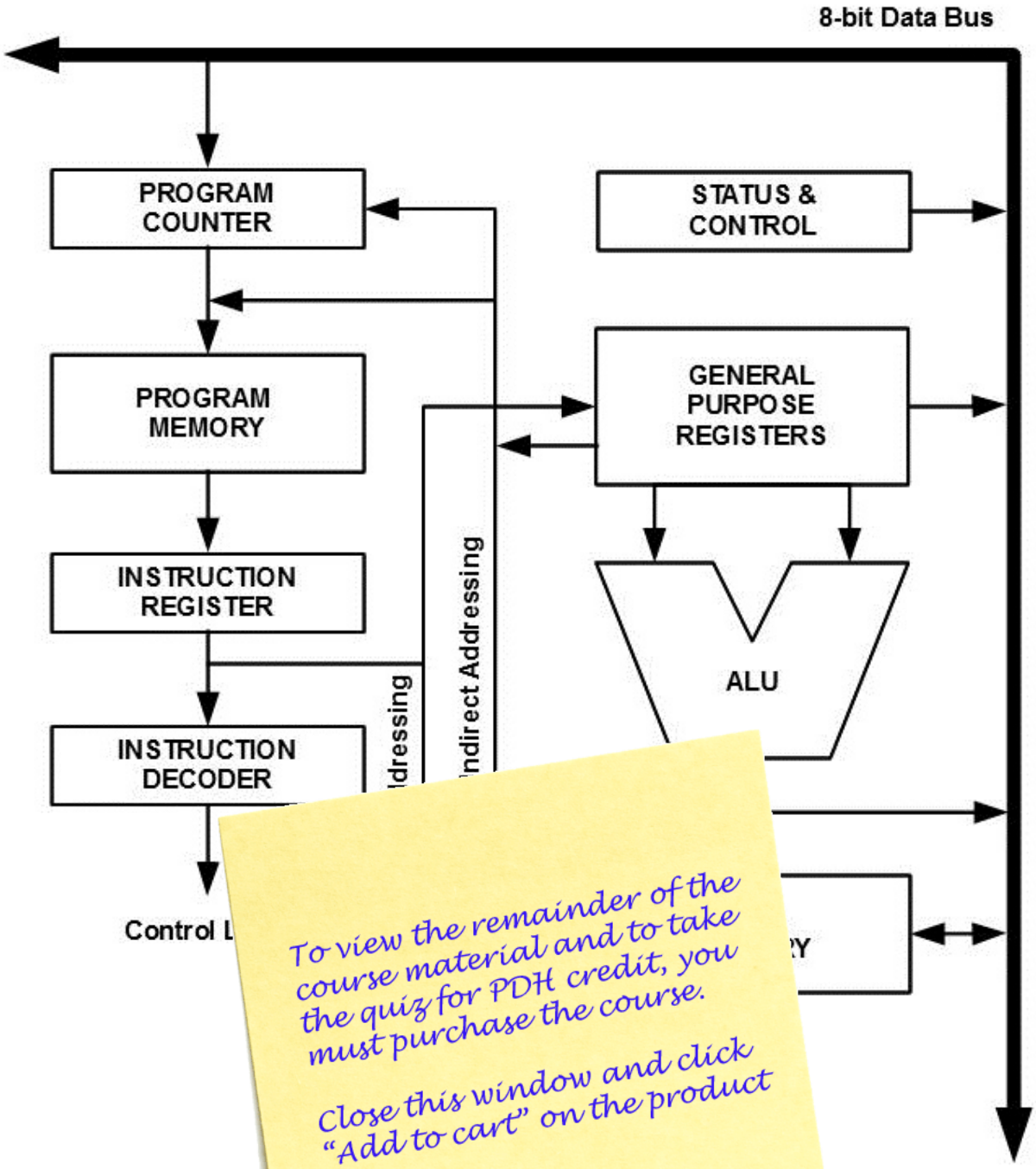


Figure 3 - Harvard Architecture

The AVR core includes 32x8 bit general purpose registers that are directly connected to the arithmetic logic unit (ALU). The main purpose of the central processing unit (CPU), or core, is to maintain correct program execution. The core accesses program and data memories, performs calculations, controls peripherals and handles interrupts.



The 32 registers in the c ock cycle. Two registers from the registe perform an operation (like an add) ar gister file all in a single clock cycle. The ALU supports perations between registers or between a constant and a register. The stack is ted in SRAM, and must be initialized in the reset routine.