USC Viterbi School of Engineering
Ming Hsieh Department of Electrical and Computer Engineering

EE 459Lx - Embedded Systems Design Laboratory

The Microchip ATmega328P Microcontroller

by Allan G. Weber

1 Introduction

This document is a short introduction to the architecture of the Microchip ATmega328P microcontroller and provides some information on using it in EE 459 projects. Additional documents on the EE 459 web site describe using the C software development system. For more complete information on any of the topics below, see the full Microchip datasheet or programming manual.

Note: The ATmega328P microcontroller used to be made by Atmel until Atmel was acquired by Microchip Technology in 2016. Some documents on the EE459 web site may still refer to it as Atmel ATmega328P, but it's the same microcontroller and only the company name has changed.

The Microchip ATmega328P is one member of the Microchip AVR 8-bit microcontroller family. It is the microcontroller that is used on the Arduino Uno although this document is meant to describe how to use it as a "bare metal" chip just by itself.

Each member of the ATmega family has different amounts of RAM, ROM, I/O ports, etc. Depending on the number of external pins required they may come in packages with more than a hundred pins, or with as few as eight. The ATmega328P was selected for the EE 459 class for a variety of reasons:

- Availability of both the chips and development software.
- Available in 28-pin DIP (dual-inline package) that fits into available IC sockets. Other members of the family are available in 20-pin and 40-pin packages.
- Enough TTL compatible[1] I/O pins (21) to handle most EE 459 project tasks.
- FLASH memory for easy and fast reprogramming.

2 Hardware

The ATmega328P contains the following components:

- 32kb of FLASH memory for program storage.
- 2kb of RAM memory.
- 1kb of EEPROM memory
- Two 8-bit and one 16-bit timer/counters. These can count internal clock cycles or external events and generate an interrupt when reaching a specified count value.
- 6 channels of 10-bit analog-to-digital converter (ADC).
- Serial communications port. This can be used to communicate to the COM port of a computer.
- I^2C interface port for communication with other I^2C compatible ICs

[1]The term "TTL compatible" means the I/O pins can be interfaced to logic gates in the various TTL logic families such as the very common 74LS and 74HCT integrated circuits. The I/O pins produce compatible voltage levels and they can sink or source the necessary amount of current.

- 21 lines of general purpose I/O.

Not all of these functions are available at the same time. Most of the pins on the chip are connected to multiple functional units and it is up to the programmer to decide what a particular pin does. For example, a pin might be used as a general purpose I/O line, or it might be ADC input, but it can't be both simultaneously.

2.1 Minimum Connections

In order to make the microcontroller operate the following connections must be made.

2.1.1 Power and Ground

The power supply voltage (5 volts) must be connected to the VCC input on pin 7. The ground connections are on pins 8 and 22.

2.1.2 Clock

Some sort of clock signal must be provided in order for the microcontroller to operate. On the ATmega328P the clock can come from one of three different sources. The selection of the clock source is done by programming fuse bits in the chip.

A TTL-compatible clock signal can be generated externally by other logic and connected to the XTAL1 input (pin 9.) This probably the easiest way to generate the clock for the EE 459 projects. The lab has a supply of DIP oscillators in some of the more common frequencies. These output a TTL level square wave that can be fed directly into the microcontroller and to other chips.

Alternatively, the processor can generate a clock if a crystal is connected to the XTAL1 and XTAL2 inputs. This method uses a plain crystal, not the DIP crystal oscillators as described above.

The third method uses an internal oscillator that runs at approximately 8MHz. This is probably the least accurate way to generate a clock. Do not use this method if your project requires a clock running close to a specified frequency. The advantage of using the internal clock is that you do not need to provide any external signal and other functions are now available on pin 9. For example it can now be used as Port B bit 6 (PB6) thus giving the microcontroller 22 I/O pins.

In applications where the UART0 serial communications interface is being used, the choice of clock frequency determines the baud rates that can be used for transmitting and receiving serial data. The accuracy of the frequency of the baud rate depends on the clock frequency used for the microcontroller. If a high degree of accuracy is required, an external oscillator of the correct frequency will be needed. See Sec. 2.4 for more information.

2.1.3 Reset

The reset input ($\overline{\text{RESET}}$, pin 1) must be in the high state for the processor to operate normally. This pin has an internal pull-up and does not have to be externally pulled-up to VCC in order for the processor to operate normally.

2.1.4 SPI Programming

The Flash memory on the ATmega328P is programed using connections to the reset input and three other pins: PB3, PB4 and PB5. These three I/O pins can be used for other purposes as long as the design allows the programming hardware to have sole access to these pins during the programming process. Make sure that none of these pins is used as in input from some source that will continue to drive a signal at the 328P while the reset line is in the low state.

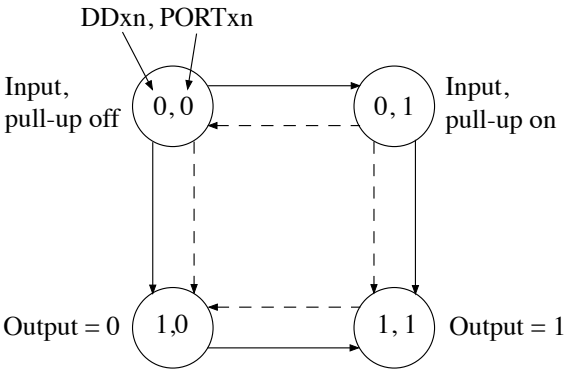

Figure 1: State of port bits when changing between input and output

2.2 I/O Ports

When used with an external clock, the ATmega328P has 21 pins that can be configured for general purpose I/O. Many of these can also be used for other purposes such as analog-to-digital conversion, timers, etc. All the I/O port bits are capable of sourcing or sinking current to drive higher-current devices like LEDs.

For each port there are three registers that control the actions of the individual bits of the port:

Data Direction Register (DDRx) - These registers determine whether the pins for that port are serving as inputs or outputs. Initially, or upon a reset signal, the bits in the DDRs are all zero which makes the corresponding I/O port bits inputs. To use a I/O port bit as an output, the corresponding bit in the DDR must be set to a one.

Port Output Register (PORTx) - When an I/O bit is configured as an output the corresponding bit in the PORTx register control the state of the output. If the bit is a one, the output line will go high. If the bit is a zero, the output line will go low. When an I/O bit is configured as in input, the bits in the PORTx register determine whether the internal pull-up resister is enabled. Writing a one to a bit in PORTx register turns on the corresponding pull-up resistor, and writing a zero turns it off.

Port Input Register (PINx) - The PINx registers are read-only registers and are used when the pin is configured to be an input. The value of the bit in the register indicates the logic level on the corresponding pin. If the pin is in the high state the value in the register is a one. If the pin is in the low state the value is a zero.

A potential problem can sometimes occur in a design that requires switching bits in a port between input and output. Since the direction of the port bit and the output state of the port bit are controlled by bits in separate registers it's not possible to switch both conditions at the same time. They must be changed individually and this can lead to the port bit briefly being in an undesirable intermediate state.

This situation is illustrated by the state diagram in Fig. 1. When changing a port bit from being an input with the pull-up resistors off to a high output, the port bit must briefly be either an input with pull-ups on, or a low output as shown by the solid lines between the states. Same would apply when making the reverse change. Similarly, when changing a port bit from being an input with pull-up on to a low output (or the reverse), the port bit must briefly be either an input with pull-up off, or a high output as shown by the dashed lines in the figure.

2.2.1 Port B (PB)

Port B on the ATmega328P has seven usable pins (PB0 through PB5 and PB7). A eighth bit, PB6, shares a pin with the XTAL1 input. If the chip is configured for an external clock, this pin is not available for I/O. Three of the pins (PB3, PB4 and PB5) are use for the SPI interface for programming the Flash memory. These pins should not be used as inputs connected to sources that will continue to drive signals at the 328P while in the reset state.

2.2.2 Port C (PC)

Port C on the 328P has six pins (PC0 through PC5). A seventh bit, PC6, shares a pin with the RESET input. By changing the configuration fuse settings this bit can be use for I/O. Most of the pins in PC are shared with the analog-to-digital converter so if the ADC function is used one or more pins will not be available for general purpose I/O. In addition, PC4 and PC5 are use for the I^2C interface and will not be available for general I/O if I^2C is used.

2.2.3 Port D (PD)

Port D on the 328P has eight pins (PD0 through PD7). Two of the pins, PD0 and PD1, are shared with the serial communications interface and can not be used as I/O if the USART0 functions are used.

2.3 Timer/Counters

The ATmega328P contains three timers:

Timer/Counter0 - an 8-bit counter.

Timer/Counter1 - a 16-bit counter.

Timer/Counter2 - an 8-bit counter similar to Timer/Counter0 but with asynchronous clocking capability.

The internal timers can be used to count events and generate an interrupt when a specified number of events has occurred. A common use of a timer is to implement a delay function by counting the number of internal clock cycles that occur. The example on the class web site in program `at328-2.c` and discussed below uses the 16-bit timer but the the procedure is similar for the 8-bit timers.

To implements a delay first set the timer for "Clear Timer on Compare Match" (CTC) mode using Output Compare Register A (OCR1A). The mode is set using four bits: WGM12 and WGM13 in TCCR1B, and WGM10 and WGM11 in TCCR1A. In most situations enabling the CTC interrupt is also required. This is done by setting the OCIE1A bit in the TIMSK1 register. In this mode the counter counts up to the value in OCR1A, generates an interrupt, clears the count and starts counting up again.

Use the rate of the internal clock to calculate what count value the counter will need to count to. If the maximum value exceeds the range of the timer's 16-bit register (greater then 65,535), determine what value to use in the prescaler to divide the internal clock by (8, 64, 256 or 1024) before it reaches the timer. The maximum count value, after any prescaling, is loaded into the Output Capture Register (OCR1A). The prescaler is controlled by bits CS10, CS11 and CS12 in TCCR1B. The action of setting the prescaler bits to something other than all zeros starts the timer counting. To turn the timer off, set the prescaler bits to all zero.

When the counter reaches the maximum count value, it generates an interrupt, resets the count value to zero and continues to count. The user program should service the interrupt and take whatever action is necessary. Keep in mind that the counter does not stop and wait for the interrupt to be serviced. It continues to count regardless of when or if the user program services the interrupt.

2.4 Serial Communications Interface

The ATmega328P contains a Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) that can be used to interface to the "COM" port on a PC or to another device. The data is transmitted from the ATmega328P on the TxD pin, and data is received on the RxD pin. If the external device is using RS-232 voltage levels for signaling, an external RS-232 transceiver chip like a Maxim MAX232 is needed to translate between the voltage levels the ATmega328P uses and RS-232 positive and negative voltages.

The USART0 transmitter (TxD) and receiver (RxD) use the same pins on the chip as I/O ports PD1 and PD0. This means that applications that use the USART0 can not also use these two bits in Port D. It is not necessary to set any bits in the DDRD register in order to use the USART0

The baud rates for sending and receiving serial data are derived from the main clock that drives the processor. The baud rate is determine by the value in the Baud Rate Register (UBBR) which is determined by the following equation.

$$\text{UBRR} = \frac{f_{osc}}{16 \times BAUD} - 1$$

where f_{osc} is the processor clock rate and $BAUD$ is the desired baud rate.

Since the baud rate is the result of dividing the clock by some integer value, in order to get an accurate baud rate it may be necessary to select an oscillator with a frequency that is not one of the normal sounding rates like 8MHz, 10MHz and 12MHz since these will not generate baud rates at exactly the correct frequency.

For more detailed information on using the serial interface, see the document "Using the Serial Communications Interface" on the class web site.

2.5 I^2C Interface

The ATmega328P has the ability to communicate to other IC's using the IIC or I^2C serial interface. This is a two line bi-directional interface designed for medium speed communications between ICs on a board. One line is the clock, the other is for data. Numerous ICs are available on the market that have I^2C interface such as EEPROMs, temperature sensors, real-time-clocks, RAM, etc.

On the 328P, the I^2C interface is available on pins 27 and 28. These pins are shared with both the ADC function and by Port C (PC4 and PC5). If the I^2C interface is enabled then any other function on those pins is not available.

If the pins mentioned above are not available for use, it is still possible to implement an I^2C interface by using software to perform all the I^2C transactions. Several I^2C libraries are available from Internet sites, most of which will probably have to be modified to work with the 328P.

For more information on using I^2C, see the document "Using the IIC Interface" on the class web site. This document also contains information on using the oscilloscopes in the lab to debug I^2C transactions.

2.6 A/D Conversion

The 328P has an internal 10-bit analog-to-digital converter for converting analog voltages to a binary value. The ADC inputs can be used for a variety of tasks such as sampling a voltage to determine the position of an input control. The ADC generates a value from 0 to 1023 for input levels between ground and whatever upper reference voltage is selected. For simple measurements, the upper reference voltage can be the ADC supply voltage input (AVCC) which is normally connected to the chips VCC supply voltage.

The ADC can accept input from any one of 6 pins depending on the bits in control register. It can only do a conversion of one input at a time. The following example shows how to make the ADC sample a voltage between ground and AVCC on input ADC3 (pin 26). The conversion process requires an internal clock that can be generated from the microcontroller clock by configuring a prescalar to divide it down to reach a frequency between 50kHz and 200kHz.

The following lines set the source of the reference voltage to AVCC, set the output to have the 8 most significant bits of the 10-bit result stored in the ADCH register, set the prescaler to divide the clock by 128, set the channel to be ADC3, and enable the ADC.

```
ADMUX |= (1 << REFS0);       // Set reference to AVCC
ADMUX &= ~(1 << REFS1);

ADMUX |= (1 << ADLAR);       // Left adjust the output

ADCSRA |= (7 << ADPS0);      // Set the prescalar to 128

ADMUX |= (3 << MUX0);        // Set the channel to 3

ADCSRA |= (1 << ADEN);       // Enable the ADC
```

Once the ADC is configured, the following loop starts a conversion, waits for it to finish and then loads the eight-bit value into the variable `x`.

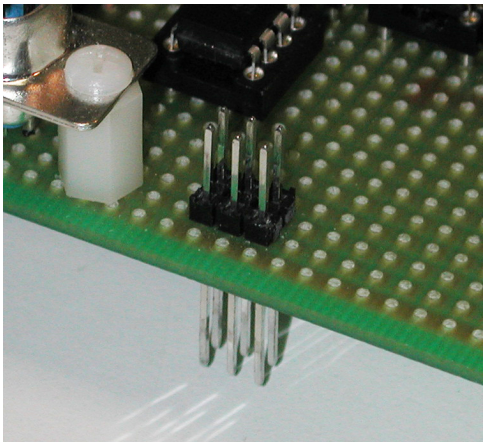

Figure 2: 6-pin ISP header mounted on a project board

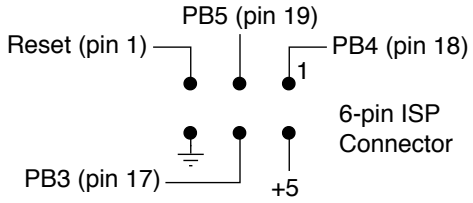

Figure 3: View of connections to the 6-pin ISP header from the **component side** of the board

```
while (1) {
    ADCSRA |= (1 << ADSC);  // Start a conversion

    while (ADCSRA & (1 << ADSC)); // wait for conversion complete

    x = ADCH;               // Get converted value
}
```

See the full 328P manual for more information on using the ADC.

2.7 SPI Programming Interface

In order to program the ATmega328P a project board needs to have a six-pin header on the board (Fig. 2) with connections to power, ground, reset and PB3, PB4 and PB5. Figure 3 shows the connections that need to be made. This is the view of the connector from the top of the board (component side). This six-pin header mates with the connector on the programming modules provided in each EE459 toolkit. When mounting the header on your board make sure to leave some room around it for the connector to mate with it. The cable to the connector comes in from the side where the power and ground pins are located so it's best to leave extra space on this side of the connector for the cable.

3 Software

There are several ways to create the code that gets programmed into the FLASH memory of the processor. The "avr-gcc" software for compiling C programs, linking, and downloading the executable program to the ATmega328P is based on the "gcc" compiler package and is available for free from various Internet sites for Macs and Windows systems. The installation notes for installing the avr-gcc software used in EE109 should work for setting up systems to do programming in EE459 More detailed information on acquiring, installing and using the software for the 328P is discussed in a separate document available on the EE459 web site.