CPE 301 – Embedded Systems Design Lab

Lab # 04 – Arduino IDE, GPIO & Debugging

Fall 2021

Objectives:

1. Install the Arduino IDE and run a simple example.
2. Learn the basic tools available for debugging embedded software.
3. Learn the basics of GPIO using both library functions, and register-level port manipulation.

Required Equipment:

1. Arduino Mega 2560
2. USB programming cable
3. Laptop or Lab PC with Arduino IDE installed
4. Laboratory Oscilloscope
5. 330 Ohm Axial Resistor
6. 3mm LED
7. Push-button
8. Breadboard
9. Jumper Kit

BEFORE THE LAB: If you intend to program your Arduino from your own laptop or PC, install the IDE before attending lab, and test to see if your computer recognizes the Arduino when it is plugged in. Then read the background information below.

Background:

Debugging

Embedded software development is significantly different from developing software to run on a typical PC. Traditional application-level development runs on top of an operating system with significant memory and processing resources. Embedded systems typically run in resource-constrained environments and perform tasks with real-time constraints. Debugging embedded applications can also be challenging as there is no traditional display on which to display status and error messages. Debugging utilities must be considered when developing an embedded application.

There are many ways of debugging an embedded system. First, a system may turn on and off an LED to signal the state of some flag. For example, if some condition has occurred, turn on an LED, otherwise turn it off. Another way to debug is to use serial messages. On an Arduino, writing to the default serial port will send the messages over USB to the connected computer. The computer can then display the messages with a serial terminal such as Putty or the Arduino Serial Monitor. For this lab, students may use the Serial Library to send debugging messages to a serial terminal. In order to enable serial communication, the Arduino code must call a function, Serial.begin(9600), in the setup function of the program. This initializes the Serial port and sets the baud rate to 9600. The specifics of this function and serial communication will be explored in later labs. To write a message to the serial port, call this

function: Serial.println("This is a message");. This will send the string parameter of the function over the serial port to the computer.

GPIO

General Purpose Input Output or GPIO is the most basic way a microcontroller has to interact with the rest of an embedded system. It is the control and feedback of a physical pin between two states: high or low. When a pin is configured as an output, the microcontroller can drive the pin high or low to signal another device, for example, turn an LED on or off. When a pin is configured as an input, the microcontroller can read a signal from another device, for example, the state of a button. In an 8-bit AVR microcontroller, physical pins are arranged as groups of 8 pins called "ports". Each pin on a port can be configured and controlled independently. A GPIO pin is controlled via a bit in three registers: PORT, PIN, and the DDR.

1. DDR (Data Direction Register) - controls whether a given pin is an input or an output. A '1' in the corresponding position of the DDR indicates that the pin is an output. A '0' in the position indicates that the pin is an input.
2. PORT – If the pin is configured as an output, this register controls the state of the pin. A '1' in this register drives the pin high. A '0' in this register drives the pin low. If the pin is configured as an input, this register controls the internal pullup resistor for this pin. If the corresponding bit is a '1', the pullup resistor is enabled. If the corresponding bit is a '0', the pullup resistor is disabled.
3. PIN – This register reflects the state of the pin, whether it is input or output. A '1' indicates that the pin has been driven high, a '0' indicates that the pin has been driven low.

The above register descriptions can be summarized in the table below:

GPIO Mode	DDR	Port	Pin
Input	0	0	State of pin
Input with Pullup	0	1	State of pin
Output	1	DATA (1 = High, 0 = Low)	State of pin

Masking

Masking is a logical operation on a piece of data. It is used for many purposes, but in the context of GPIO, it is used to manipulate a single bit or group of bits in a byte. When we set a pin as input or output, or control the state of an output or read the state of an input, we must read or write the entire byte of a register. If we use a mask, we can read or manipulate the register without changing the bits in a register that are not relevant. For example, the built-in LED on an Arduino Mega is connected to port b, pin 7 (PB7), in an active high configuration. This is the 8th bit, or bit 7 of the port B registers. To turn on an LED we want to place a '1' in the port register at bit 7. To do this, we "or" the port register with the mask 0x80 (or 0b1000 0000). This will set bit 7 to a '1' without changing any of the other bits. To turn off the LED, we "and" the port register with 0x7F (or 0b0111 1111). Remember from previous courses that an "and" operation sets 0's and an "or" operation sets 1's.

Arduino Pins & AVR Ports

Vocabulary around pins and ports can be confusing for the uninitiated. The Atmel 2560 is an AVR microcontroller commonly used in embedded systems. Atmel defines all of its ports and pins relative to the microcontroller in the 2560 datasheet. An embedded system engineer would use these definitions when designing a system using the 2560. These definitions can be found on the diagram below, and a more detailed description of the registers can be found on page 399 of the datasheet.

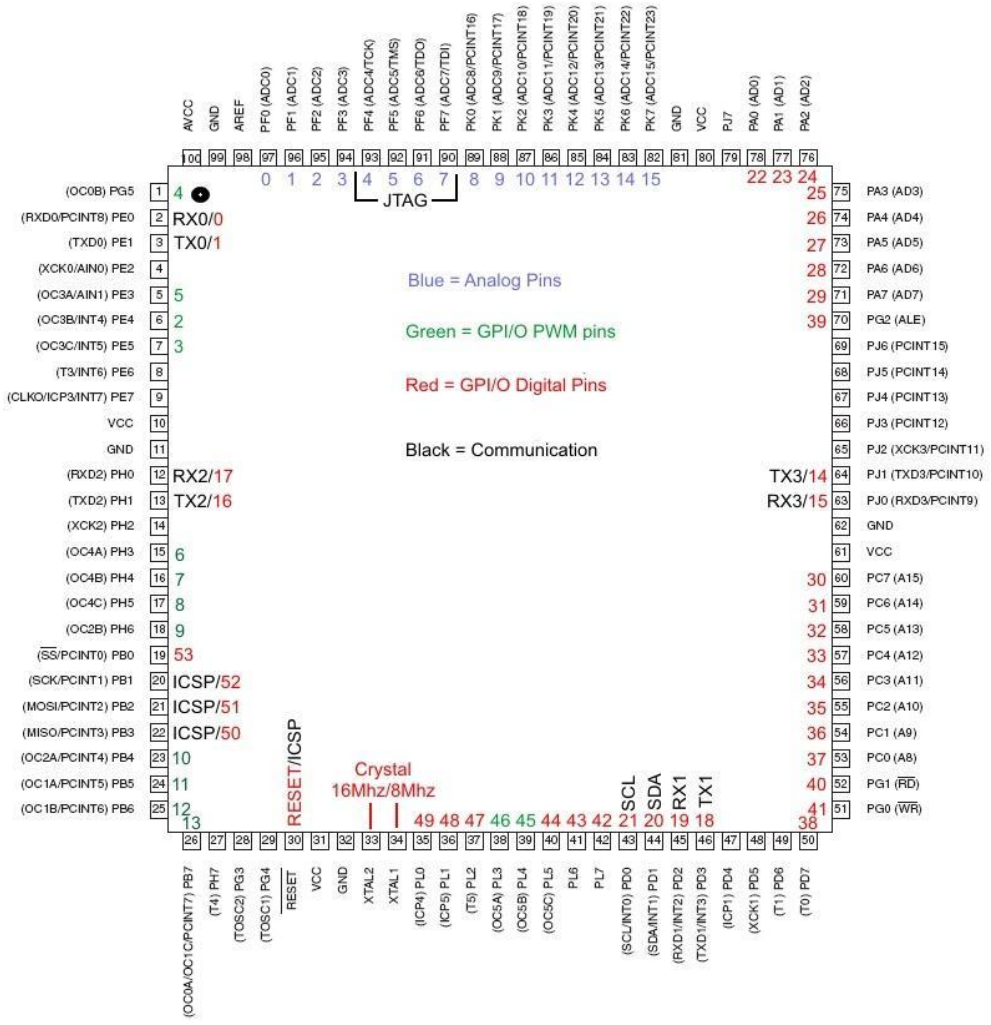

Figure 1 - Atmel 2560 Pinout

The Arduino Mega is a development board which uses an Atmel 2560 as its microcontroller. Arduino has added a layer of abstraction so that the definition of various pins matches those of other Arduino boards, such as the Arduino Uno and Nano which use an Atmel 328. This allows users to swap out different Arduino boards with minimal changes to the code. For example, on all Arduino boards, the on-board LED is connected to digital pin 13. However, on the Mega, digital pin 13 is actually port B, bit 7, while on the Uno and the Nano, digital pin is connected to port B, bit 5. Using the library function, digitalWrite, to write pin 13 high will turn on the LED on all Arduino Boards. We will not be using the Arduino Library functions nor pin definitions for the majority of the course, but understanding these different definitions will greatly aid the understanding of datasheets, diagrams, and example code. The Arduino Mega "pinout" below shows both the Atmel Port definitions alongside the Arduino Pin definitions.

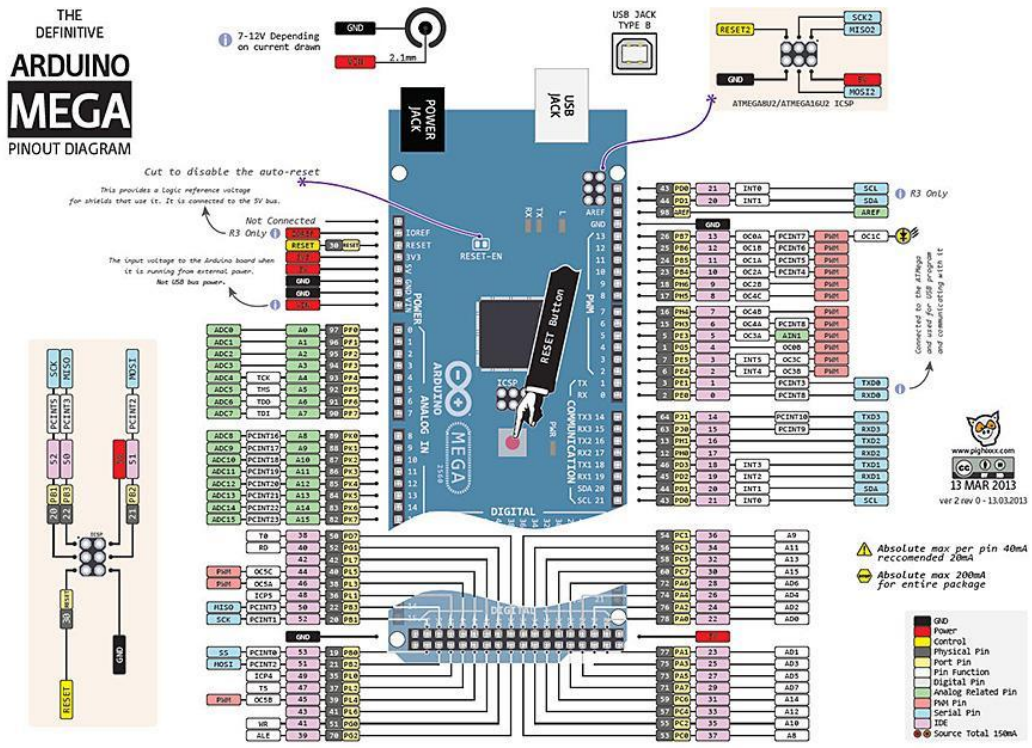

Figure 2 - Arduino Mega Pinout

Procedure:

Part 1 – Library GPIO

Open the blink example in the Arduino IDE by clicking File>Examples>Basics>Blink. Compile and upload the code to your Arduino by clicking the right arrow shaped button at the top of the IDE.

1. Verify that the code is working – does the on-board Arduino LED blink? Does it blink at the expected rate? What is the frequency of the signal from the code?
2. Connect the lab oscilloscope to pin 13 on the Arduino observe the output signal. What is the measured frequency of the signal?
3. Modify the code in the example to blink faster, and check output signal. What is the frequency of the signal?
4. Modify the code in the example to blink as fast as possible and observe the signal. What is the fastest frequency you can attain with the library functions?
5. Modify the example code to output the signal on both pin 13 and pin 7.
6. Connect an LED and a resistor in series to pin 7, and configure it such that the LED is on when the pin is driven high. Is this LED active high or active low?
7. Connect an LED and a resistor in series to pin 7, and configure it such that the LED is off when the pin is driven high. Is this LED active high or active low?

Part 2 – Register-Level Output

Download example_1.ino from webcampus, this example demonstrates manipulating GPIO output at the register level.

1. What is the frequency of the signal on PB7 without modifying the code?

2. Modify the code to switch the state of PB7 as fast as possible. What is the fastest frequency you can attain using low level register access functions?
3. What is limiting the maximum output frequency of the code? Are we approaching the clock frequency of the Arduino Mega?
4. What is the function of the line in the setup function? What does the DDR register do?
5. Explain what the 'or' operation and the 'and' operation in the loop function are doing to the port register. What does the port register do?

Part 3 – Preprocessor Definitions and Function Wrapping

Download example_2.ino and example_3.ino from webcampus, these examples demonstrate two different ways to implement GPIO at the register level.

1. What is the difference between example_1.ino and example_2.ino?
2. What is the difference between example_2.ino and example_3.ino?
3. After compiling, the Arduino IDE lists the memory consumption for both program data, and variable data in the terminal window at the bottom of the IDE, compare the memory consumption of all three examples.
4. Modify example_2.ino and example_3.ino to maximize the frequency output on PB7. What are the frequencies achieved?. Is there any difference in the maximum frequency output between examples 1 through 3? What explains the frequency differences?
5. Is there any difference in readability between the three examples?

Part 4 – Register-Level Input

Download example_4.ino from webcampus. This example code demonstrates how to implement register-level input. Connect a push button to PB4 such that PB4 is grounded when the button is pressed, and open the Arduino Serial Monitor by clicking the magnifying glass at the top right corner of the IDE. Be sure to set the baud rate so that it matches the baud rate set in example_4.ino.

1. Why do we need to enable the pullup resistor on PB4? What would happen if the pullup resistor was not enabled?
2. Test your assumption from question #1 by modifying the example to disable the pullup resistor. Is the input reliable? Does it reflect the state of the pin? What is the state of the pin when the button is not pressed? (Remember the 3rd state in tristate devices)
3. What is the sampling frequency of the example program? How often is the input checked?

Part 5 – Putting Everything Together

Write an Arduino program to read the state of PK2, output this state on PD0, and output the complement of this state on PE3. Write the program to minimize the time delay between a change on PK2 reflecting onto PD0 and PE3. Attach a button to PK2, and an LED (with a series resistor) to PD0.

1. Use an oscilloscope to measure the propagation delay between PK2 and PD0.
2. Use an oscilloscope to show the difference between PD0 and PE3

References

1. Atmel 2560 Pinout - http://4.bp.blogspot.com/-Sj4AEX-ndNo/UeqPDlJo3DI/AAAAAAAAJN4/uN5tE84jGPI/s1600/MEGA2560.jpg
2. Arduino Mega Pinout - https://arduino-info.wikispaces.com/file/view/Mega2-900.jpg/421499040/Mega2-900.jpg

3. Atmel 2560 Datasheet - http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf