Lab 1: Intro to Microcontroller Development, Simple GPIO and IDE Programming

OBJECTIVES

This lab will introduce you to the concept of developing with a microcontroller while focusing on the use of General Purpose Input/Output (GPIO) pins. You will be introduced to an Integrated Development Environment (IDE) where you will make a simple program to control an output device using a simple input device.

THIS Lab expects that you have an understanding of programming and simple circuit design. If at any time you are lost, please ask for clarification.

REQUIRED MATERIALS

- Epiphany-DAQ board
- Wire Jumpers
- Switch
- LED
- Resistors
- Breadboard
- Multimeter (if needed)

DISCUSSION

In this section we will review common concepts for developing a simple circuit and program as this lab will review. Please read the sections that you may be unfamiliar with from the following list:

- Digital I/O
- Number Bases
- Ports and Registers
- Masking
- Breadboard Testing

Digital I/O

When building a device, such as a robot, it is often desired to use a microcontroller to control the behavior of the system. Though microcontrollers may have many different abilities, the simplest and most common is the use of digital (binary) input and output signals. A digital signal may only be in the form of one out of two possibilities. Various descriptions may be given to those possibilities:

Table 1. Digital Signal Descriptions

On	Off
High	Low
Vcc	Ground
5V (3.3V)	0V
1	0
True	False

Some examples of digital inputs may be in the form of a Switch, digital IR Sensor, or digital Tilt Switch. These types of devices typically have three wires in the form of a power and ground source, as well as a single output source acting as the digital signal. A digital input may enter a microcontroller through GPIO pins that are typically grouped in the form of 8-bit ports. As the input enters the microcontroller through a port/pin, software may be used to detect a change in the pin and react accordingly.

While digital inputs may be used by the microcontroller to change the behavior of the system, digital outputs are typically the result of the behavior change. When something happens in the system, digital outputs may be used to cause an external affect. Examples of digital outputs are in the form of LEDs or Mechanical Switches/Relays.

When working with digital inputs, some considerations should be made in regards to the voltage and current levels of the digital signal as well as the tolerance of the microcontroller pins. If a pin is 3.3V tolerant, it is best not to supply the pin with 5V. This means, as an input, you should limit the signal to the tolerance of the input pins of the microcontroller. This may require a change in circuitry, or simply choosing the right sensor to output the desired voltage.

As for outputs, the consideration should be placed in what the output capabilities of the microcontroller are, as well as the required voltage and current levels of the device being supplied the signal. If you are outputting a digital signal to an LED, it needs to be at the correct voltage and current to power the LED with the desired brightness.

Number Bases

When programming microcontrollers, it is typically required to understand binary and hexadecimal number systems, which are base-2 and base-16 systems respectively. Remember that our normal number system that we use is decimal, base-10 number system. There are many resources on how bases work through the internet, however, we will review the concepts.

When working with number systems, they are typically designated in text by placing a subscript following the value. The subscript value is set to the base of the system. For example, a decimal value may be given as 37_{10}. To represent a number from a base to decimal, the following equation may be used:

$$a = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0$$

The letter $\boldsymbol{b}$ in the equation represents the base of the number, while $\boldsymbol{a_n}$ represents the digit in place of the number.

When dealing with binary number systems, a digit is simply either a 0 or a 1. A single digit in binary is often

called a **bit**. To represent a number from base-10 as we are familiar with, it will take many bits. Each bit from right to left is a higher order of the base, i.e., $2^2\ 2^1\ 2^0$. See the following for an example:

$123_{10} = 0111\ 1011_2$
$123_{10} = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

Hexadecimal systems are handled in the same format as binary, however at base-16, using values from **0 to 9** and **A – F**. Notice that a hexadecimal value represents decimal values from 0 to 15. It is important to note a commonality between binary and hexadecimal. Binary numbers are made of multiple bits. Four bits grouped together form a **nibble** and also may be used to form a hexadecimal value. Eight bits grouped together form a **byte** and are also formed of **two** hexadecimal values. See a continuation of the previous example below:

$123_{10} = 0111\ 1011_2$
$123_{10} = 7\ B_{16}$

PORTS AND REGISTERS

To access ports in the microcontroller, the IDE typically allows you to type the name and letter designation of the port. For example when reference PORT E:

PORTE

Typically, ports have registers that tell various things about the port. The three primary registers used are **DIRECTION (DIR), IN,** and **OUT**. Registers are setup to reference each bit of the port, therefore, in an 8-bit port setup, the register will be 8 bits. Understand that these registers will change the value of **every** bit of the port. Each bit of the value set to the register represents the value or affect that register has to a specific pin. For example, if I want to reference **pin 2** only, my hex value to represent that pin will be **0x04**.

In more detail, the **DIR** register specifies the direction that each and every pin of the port is set to, in regards to input or output. On most chips, the default direction is input. On Atmel chips, to designate a pin as input, the **DIR** register value for that pin must be set to a zero. To designate a pin as output, the **DIR** register value for that pin must be set to a one.

The **IN** register specifies a value read into the port from an external source. The **OUT** register specifies a value to be sent out from the port. In programming, a port register may be set to a value with the following command.

PORTD.OUT = 0x01;

This effectively says that pin 1 should have an output of a high value while all other pins are low. The following table reviews each standard register.

Table 2. Standard PORT registers

PORTx.DIR	Set the Direction of all pins for the port
PORTx.IN	Holds the input value read for the port
PORTx.OUT	Set the output value for all pins of the port

Occasionally, registers have extra specifications that allow further modification of their values. They are typically in the form of **SET, CLR,** or **TGL.** These modifiers are used in situations where it is desired to affect only a single pin of a port, and not every pin as the standard **DIR**, **IN**, and **OUT** registers do. Which pin is modified still uses the process of changing the value of a bit representing the pin in either binary or hexadecimal.

When a value is used with the **SET** modifier, it designates which pin of the port should be set high. When using the **CLR** modifiers, this value designates which pin of the port should be set low (cleared). The **TGL** modifier simply says to toggle whichever pin the value references. A few examples are below:

PORTD.DIRSET = 0x02
PORTD.DIRCLR = 0x11
PORTD.OUTSET = 0x03

The first example sets **pin 1** of PORTD Direction register to high. The second example sets **pin 4** and **pin 0** of PORTD Direction register to low. The final example sets **pin 1** and **pin 0** of PORTD Output register to high.

Table 3. PORT Registers with Modifiers

PORTx.DIRSET	Set the Direction of specific pins for the port to **output**
PORTx.DIRCLR	Set the Direction of specific pins for the port to **input**
PORTx.DIRTGL	Toggle the Direction of specific pins for the port to **the opposite of current value**
PORTx.OUTSET	Set the Output Value of specific pins for the port to **high**
PORTx.OUTCLR	Set the Output Value of specific pins for the port to **low**
PORTx.OUTTGL	Toggle the Output Value of specific pins for the port to **the opposite of current value**

Masking

When working with PORTs, it is commonly desired to get a value of a single pin in the port. To do this, we do something called **AND MASKING**. To do this, you must

first generate a hexadecimal byte (two hex characters) that represents the bit that you are interested in, creating your MASK. Then, if you simply AND the PORT value with the MASK, the result will be the value of the pin/bit in question.

PORTD.IN & 0x08

The example will give the result of whatever is held in **pin/bit 3.**

Breadboard Testing

When testing circuits, you may want to use a breadboard. Breadboards come in many sizes and flavors, however, you may be using one that is similar to that in Figure 1.

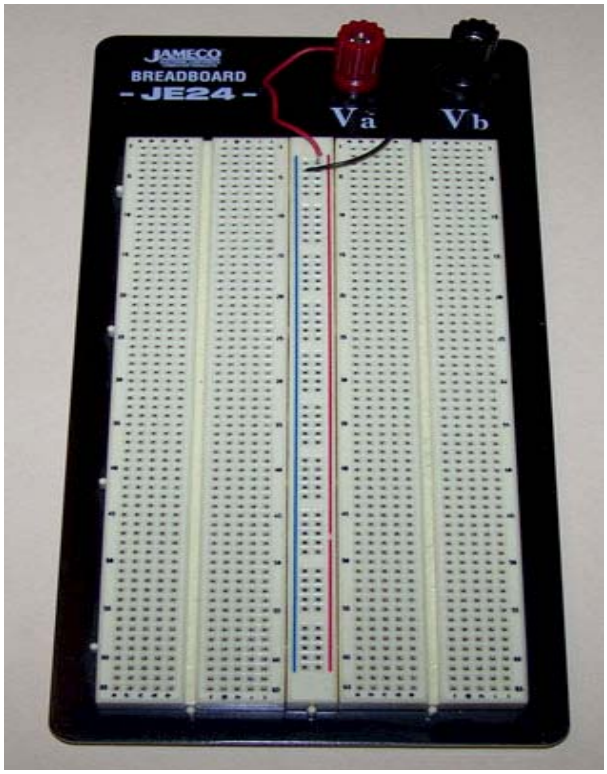

Figure 1 Breadboard

When using a breadboard, you must understand how each hole is grouped and connected to each other. A close-up of a breadboard is given in Figure 2.

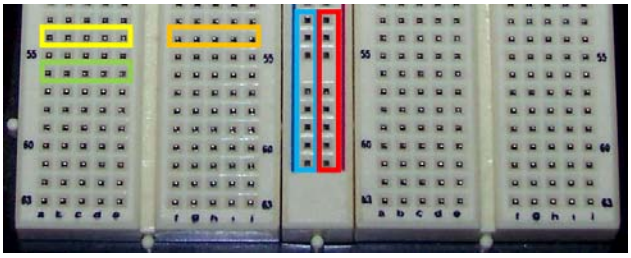

Figure 2 Breadboard Groupings

The board has two types of regions called socket strips and bus strips. Socket strips consist of groups of five holes arranged in a horizontal manner, as shown by the Yellow, Orange, and Green rectangles. Each hole in a group is shorted together forming a node. For example, this means that all five holes in the Yellow rectangle are connected together. If you place Vcc in the left hole, the Vcc value will also be in the right hole.

Each socket strip is separate from each other. This means that all pins in the Orange rectangle are shorted together, however, they have no connection to the Yellow rectangle. It is possible to connect a single wire between each socket strip to make a connection.

Bus strips, designated by the Blue and Red rectangles, are groupings of five pins arranged in a vertical manner. Even though these strips are setup in groups of five holes, every single hole in the column is shorted together. However, they are not shorted to the next column, i.e., Blue column and Red column are not connected. Bus strips are typically used for supply sections of the board with power and ground.

LAB PROCEDURE

Epiphany-DAQ board and Atmel Studio

NOTE: The below section describes using an Atmel based chip. If you have already received your desired microcontroller, you may use it and demonstrate the following requirements. Your TA may or may not have the knowledge of your board to directly help you, but feel free to ask for assistance.

Today you will be introduced to the Epiphany-DAQ board for a simple development test program. The board has two sets of pins on the left and right side of the board; 16 pins for each set. Below is the pinout for the board:

Table 5. Digital Signal Descriptions

Pin Label	*Name*	*Pin Label*	*Name*
0	PC1	**G**	GND
1	PD0	**AI0-**	ADC0 -
2	PD1	**AI0+**	ADC0 +
3	PD2	**G**	GND
4	PD3	**AI1-**	ADC1 -
5	PD4	**AI1+**	ADC1 +
6	PD5	**G**	GND
7	PR1	**AI2-**	ADC2 -
8	PR0	**AI2+**	ADC2 +
9	PE3	**G**	GND
10	PE2	**AI3-**	ADC3 -
11	PE1	**AI3+**	ADC3 +
12	PE0	**G**	GND
2.5	2.5V	**AO-**	DAC -
5.0	5.0V	**AO+**	DAC +
G	GND	**G**	GND

Any name in the form of **Px#** designates a **PORT,** the letter designation given to that port, and the pin number (i.e., PD5 references PORT D pin 5). Any name in the form of ADC is an Analog to Digital input. There are four analog inputs on this board. Each analog port has a negative and positive source. Names in the form of DAC are Digital to Analog outputs. Both ADCs and DACs will be discussed in a later lab.

Figure 3 shows the physical Epiphany-DAQ board. The left and right screw terminals match the various pins described in Table 4. When you flip over the board, you will find labels next to each terminal. When connecting wires to the Epiphany-DAQ board, you must screw the terminal all the way down to make a tight connection to the wire. Also take care not to allow a power supply wire and ground wire to touch.

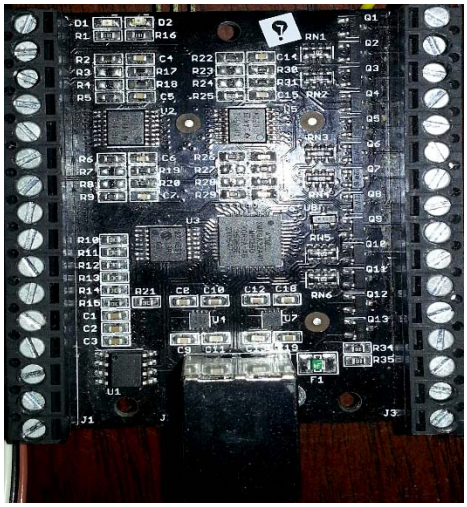

Figure 3 Epiphany-DAQ Board

The board is programmed and powered via a standard USB Type B cable. To program the board, you need the Atmel Studio IDE and the ATmega Xmega Bootloader software (chip45boot2), each located below:

http://www.atmel.com/Microsite/atmel_studio6/
http://www.chip45.com/avr_bootloader_atmega_xmega_chip45boot2.php

Once both applications are installed, ask your TA for the current library and solution file for the Epiphany-DAQ board. The project solution file that you receive may be used to create applications for the board. It includes various functions and libraries that will simplify certain functions. It is in your best interest to review any library or function that you use to better understand the code. If you do not understand the functions, you may ask your TA for clarification.

Switch and LED circuit

Once all software has been installed, it is time to create a simple switch and LED circuit for testing. You should first collect a push button switch, LED, two resistors, and a few jumper wires.

For this lab, you should create a simple pull-up switch using the circuit shown in Figure 4.

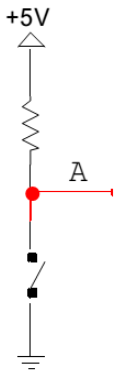

Figure 4 Pull-up Switch Circuit

Any digital signal that acts as a switch type device will typically be used in the form of the above circuit. The +5V represents whatever power supply will generate the correct tolerance for the microcontroller pins. The Red line with the label A is the digital source leaving the switch and connecting to the microcontroller input pin. Using the provided push button switch and resistor, **build the above circuit**. Once this circuit is made, **Choose any of the pins connected to Port D and run the digital input to that pin**.

Once the switch circuit is complete, begin the LED circuit. For the time being, we will create what is known as an Active High LED circuit. Discussion on what an Active Low LED circuit may be given at a later time. See the circuit diagram, in Figure 5, for the LED circuit:

Figure 5 Active-High LED Circuit

The output of the microcontroller should enter the LED circuit at the label B(H). Pay attention to the direction of the LED, noticing that the Anode side is on the top and the Cathode side is on the bottom. When working with a physical LED, the long lead is the Anode, the short lead is the Cathode. This may also be noticed by the physical flat edge of the LED, which also represents the Cathode side. Using the provided LED and resistor, **build the above circuit**. Once this circuit is made, **Choose any of the pins connected to Port D (other than that chosen for the switch) and run the digital output from that pin to the LED.**

Creating the Program

It is now time to create a simple program. In the top right of the IDE, right click on the project name, as shown by the red box in Figure 6, and choose Add -> New Item. In the new window, Choose C File and give the file a relevant name for the lab. Now, open the **blank.c** file (template for this lab), copy the code, and paste it into your newly created C File.

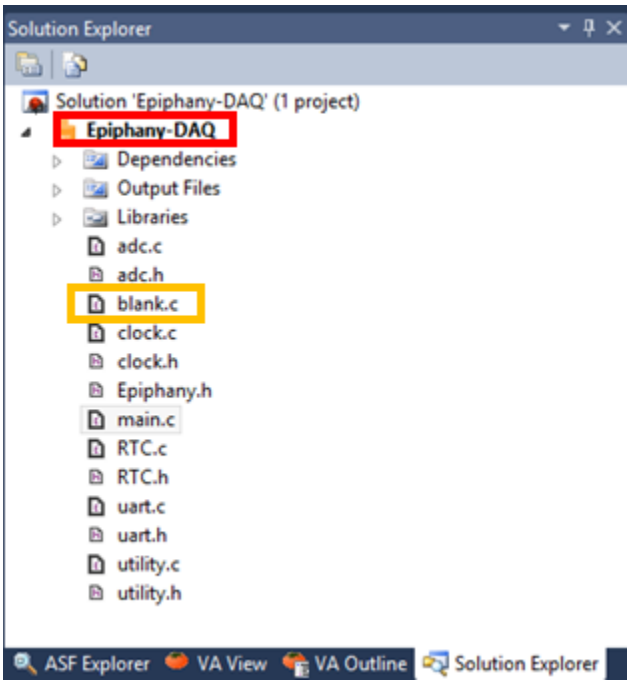

Figure 6 Atmel Studio Solution Explorer

To allow for the microcontroller to communicate with the previously created switch and LED, two ports must be configured. Going back to the previous section, create two separate hexadecimal bytes to represent which pin is being set as output and which is input (NOTE: this can be done in binary as well).

Using the correct register for the desired port and pin, write the code required to accurately define the required directions. This code should be placed in the setup function. Hint: This should only take 1 to 2 lines.

Now, creating a simple **IF/ELSE** statement, check the desired input pin using **AND MASKING**. If the pin is

high, set the desired output pin for the LED to high, otherwise, set the output pin to low. **This code should be placed in the loop function.**

To compile the program, first select the blank.c file in the solution explorer. Locate the properties window below the solution explorer, shown in Figure 7. Click on the Build Action option, select the drop down box, and change the option from "Compile" to "None". Repeat these steps for the file you created for this lab, except change the option from "None" to "Compile".

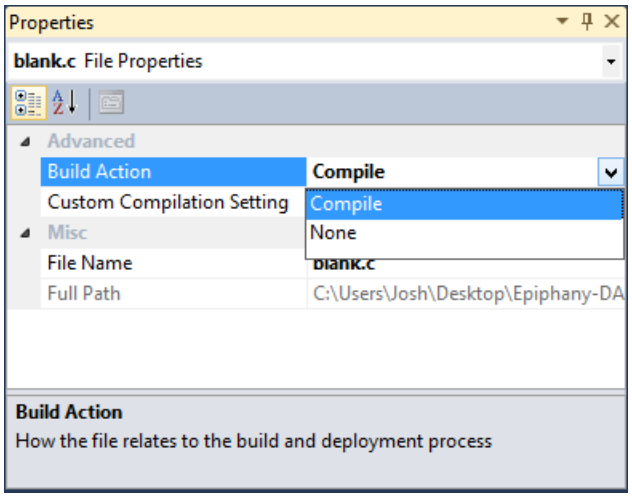

Figure 7 Atmel Studio File Properties

Now, view the menu bar at the top, and choose **Build -> Build Solution**. If no errors occur, you are ready to continue.

Programming the Board

Once the program is complete, open **chip45boot2**, as shown in Figure 8. Choose the correct COM Port for your board. Set the **Buadrate as high as you can**. Click the **Select Flash Hexfile**, button and navigate to the folder you saved the solution program to. Navigate through until you get to the debug folder where the hex file is located.

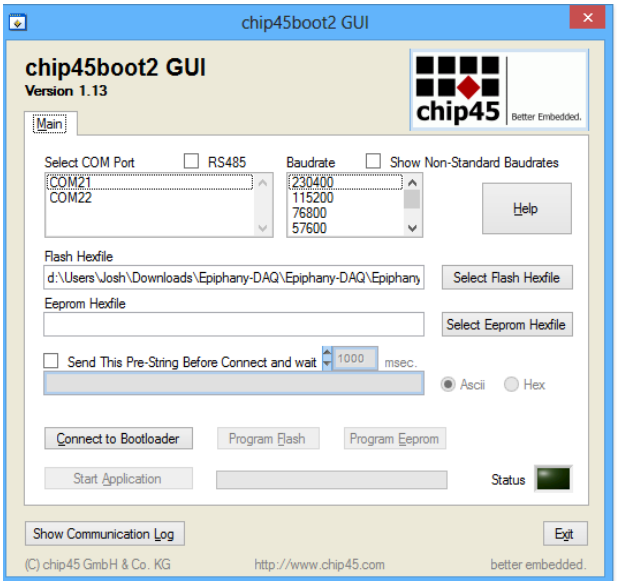

Figure 8 chip45boot2 GUI for Programming

Select the **Connect to Bootloader** button. If the Status light goes Green, then press the **Program Flash** button, and finally the **Start Application** button once the device is programmed. Begin Testing!!