Lab 6

Due Dates:

Submit this lab and the extended learning (optional) to TEACH by Monday at 11:59pm.
You must get checked off by a TA by the beginning of lab 7.

Prelab:

1. Download the zipped Skeleton files (from tekbots.com) for this lab. Take a look at main.c.
   What is PortA for?
   What is PortC for?
   What is PortE for?

2. Complete the following table. Assume every number is positive. In other words, the left-most bit is just another bit. It is not a sign bit.

	Binary	Decimal
~(0b11000101)		
0b00111001 & 0b01100011		
0b10111100 | 0b00011001		

3. Look up an "ASCII" chart online and fill in the following table.

Character	Binary	Decimal
'A'		
'a'		
'0' (zero)		
'9'		
'z'		
'Z'		

Embedded Programming Overview:

Based on your point of view, the differences between coding for embedded devices and coding for computers can be considered as very different or as very similar. Since you have just spent several weeks learning about application programming, we will approach the concept of embedded program as though it was very similar. This means, that unless told otherwise, you can assume that the coding works exactly as you have learned before. The syntax is the same, the functions work the same, and the functions compile the same.

Input and Output:

For the last several weeks you have been learning to enter data for your program using the keyboard and displaying information on the screen. This has been done with a variety of functions

including printf(), scanf(), puts(), and getchar(). However many embedded devices do not have screens and keyboards. The Wunderboard was designed to have some basic inputs and outputs that nearly every embedded system in the world has: buttons and light emitting diodes (LEDs).
A computer has lots of different ports used for lots of different functions: USB ports, PS2 Ports, Monitor ports, eSata, etc... Embedded systems have ports, too. However, usually these ports are 'lower level.' For example, rather than simply plugging in a keyboard, embedded ports often use voltages and read logic 1 and logic 0.
The Wunderboard has several different devices pre-connected to its ports, figure 19. You will learn more about each port as you use the Wunderboard more. In this lab, you will need to use PORTA and PORTC, switches and LEDs respectively.

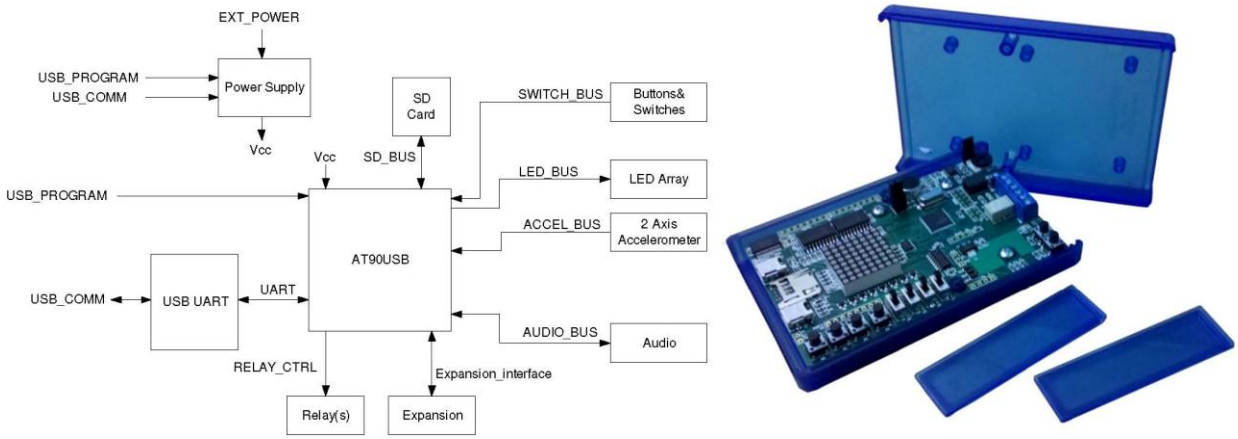

Figure 19: Wunderboard Block Diagram

Bits and Bytes

We will write binary numbers starting with 0b, for example 0b00110101, so that you know the following number is written in the binary format. It is important to understand that in computers everything binary. Numbers, text, pictures, everything is stored as binary. This is important because using binary you can do things that you cannot do in other formats. For example, you can treat 0b00101010 as a number, 42, or as a random sequence of bits. If you want to do math, it's a number, if you want to control LEDs it's just bits.

Figure 20 shows how each of the bits in a byte are 'mapped' to the bits on a port for the Wunderboard. The order of the bits is important! The most significant bit (MSB) is written on the left while the least significant bit (LSB) is written on the right.

```
                MSB                       LSB
          0b  0  1  0  0  1  0  1  0
Port bit/pin  7  6  5  4  3  2  1  0
```

Figure 20

Port Control

To use the ports on the AT90USB646, there are 3 variables you need to know about; PORTx, PINx, and DDRx. We will learn more about DDRx in later labs. The 'x' in these variables should be

replaced by the port you want to work with, for example when working with port C, you would use PORTC, PINC, and DDRC. These variables must be in all capital letters, remember that the C language is case sensitive.

- PORTx: This register controls what is outputted to the pins of a port. If a PORTx register bit is set to a '1', the Wunderboard will set the pin to a '1' (3.3V). If a PORTx register bit is set to a '0' the Wunderboard will set the pin to a '0' (0V). The example below sets up port C as all outputs and then sets the upper 4 bits to 1 (ON) and the lower 4 bits to 0 (OFF).
  ```
  PORTC = 0b11110000;
  ```
- PINx: This register lets you read what signal is being inputted to the port. If you read the PINx register, there will be a '0' for pins that are around 0V and a 1 for pins around 3.3V. The example below sets up port A as all inputs and then reads in the value from the port.
  ```
  value = PINA;
  ```

Install The Embedded Environment

To be able to program for embedded systems, the code environment must be installed. The Wunderboard system contains a microcontroller from Atmel based on the company's 8bit AVR computer architecture. Many different compilers exist for this family of tools, but to be consistent the manual uses the avr-gcc tool set. The tool set works the same as the gcc tool chain used earlier in the course but instead of producing a file that runs on the computer it produces a 'hex' file for use on the Wunderboard. Trivia: This is an example of a cross compiler.

For installing the software on Windows, two options are available. One option is to download a copy of 'WinAVR' and install it. Another option is to use a tool set called 'Portable WinAVR' that is run from a USB thumb drive or the main hard drive of a computer. The labs will assume the use of Portable WinAVR.

1. The first step to prepare the tool chain is to install the software and drivers for the Wunderboard itself. To do this, download the ATMEL F.L.I.P. (nothing to do with the engr machine named flip) program from the link on the lab webpage. We recommend FLIP 3.4.2 with the JRE included. Run the installer and install it in the default location. Using a USB cable, connect the Wunderboard programming port to the PC. Press both the reset and hwb buttons at the same time and release the reset button first, then the hwb button. Windows will activate the 'Installing New Hardware' wizard. When asked, browse to the F.L.I.P. directory and install the drivers for the Wunderboard microcontroller, the AT90USB647.

   NOTE: When using certain versions of Windows (Windows 64bit as an example) the default driver from Atmel may not work. If this is the case, use the alternate driver on the lab webpage. Only use this if Windows requires you to.

2. Download the Portable WinAVR zip file from the lab webpage. Once it is downloaded, it must be unzipped into the 'root' of a drive. A USB flash drive or hard drive is acceptable to use. For example, the files should be in F:\, not C:\Documents and Settings\user\PortableWinAVR. NOTE: The files must be in the root of the drive. The two '.bat' files show in the root and a directory called Portable will exist (eg. Z:\Portable). This lab will assume that you have unzipped the files to your 'Z' drive engineering directory.

3. Once the files have been unzipped, browse to the directory and run the file called 'CommandShell.bat.' A window will appear with a command line in it. This command line operates the same as on flip for the most part.

Compile and Test Your First Program

After navigating to the directory, run the make all command. The tool chain will run and process the file. To test the code, it needs to be downloaded to the Wunderboard.

1. To download the code to the Wonderboard, connect its programming port to the PC. There are two USB connectors, make sure that you use the programming connector.

2. Once connected press both the Reset_n and HWB_n buttons at the same time. Release the Reset_n button first, then the HWB_n button. If you have not installed the driver for your Wunderboard before this, your computer might request a driver for the board. If this happens, refer to the instructions earlier in this lab.

3. Once the board is connected and readied for programming, run the command make **programwindows** in the command shell. This will cause a program to download your .hex file to your board. Once the download finishes, your board should automatically run your program.

Demonstrate to a TA that the sample code is working on your Wunderboard. Get checked off.

Edit Your Code

Your sample code displays a single pattern of lights. Edit the code so that the lights that are displayed correspond to the switches and push buttons on the Wunderboard.
HINT: Earlier in this document there is an example showing which variable is used to 'read in' values and what variable you need to 'write out' values. You need to figure out which ports (A, B, C, D, E, or F) you will use.

Demonstrate and Submit Code

When your code is submitted, it will be processed both to ensure it compiles / runs correctly and to evaluate its comments. Comment your code using // and /* */ so that a year from now you will understand what each line is for.
When ready, submit your source code to TEACH, Lab6. Your file should be named main.c

Study Questions:

None this week.

Lab 6 Summary:

TASK	Completed?
Prelab	4pts.
Sample code displays a single pattern of lights on Wunderboard	4pts.
Edited code allows switches and push buttons to control lights on Wunderboard	4pts.
Edited code is well commented	2pts.
Study Questions from Lab5	4pts.

Extended Learning:

Once you have proven that you can read in inputs, edit your program so that when the switch on port A pin 7 (PORTA.7) is flipped, the LEDs count up. The count should be slow enough that you can see it operate. When the LEDs get to 0b11111111 they should 'roll over' to 0b00000000. The LEDs should increment in a binary fashion, (ie. 0b00000010 -> 0b00000011 -> 0b00000100). The switch should 'pause' the count, not reset it to 0 when it is not activated.
HINT: You may want to call the function **_delay_ms()** to control how fast your system counts.

Your file should be named main.c
Demonstrate your program to a TA at the beginning of Lab7.
Submit your code to TEACH Lab7Extra by next Monday at 11:59pm.