Raspberry GPIO

EET178 Lab #3

Goals:

- Learn how to interface physical components to a device
- Learn to create an embedded program in C on the raspberry

What you'll need:

- Your raspberry PI
- Breadboard, LEDs, switch, 330 ohm resistors for LED, Hookup wire – female to male – very important!!!

Background

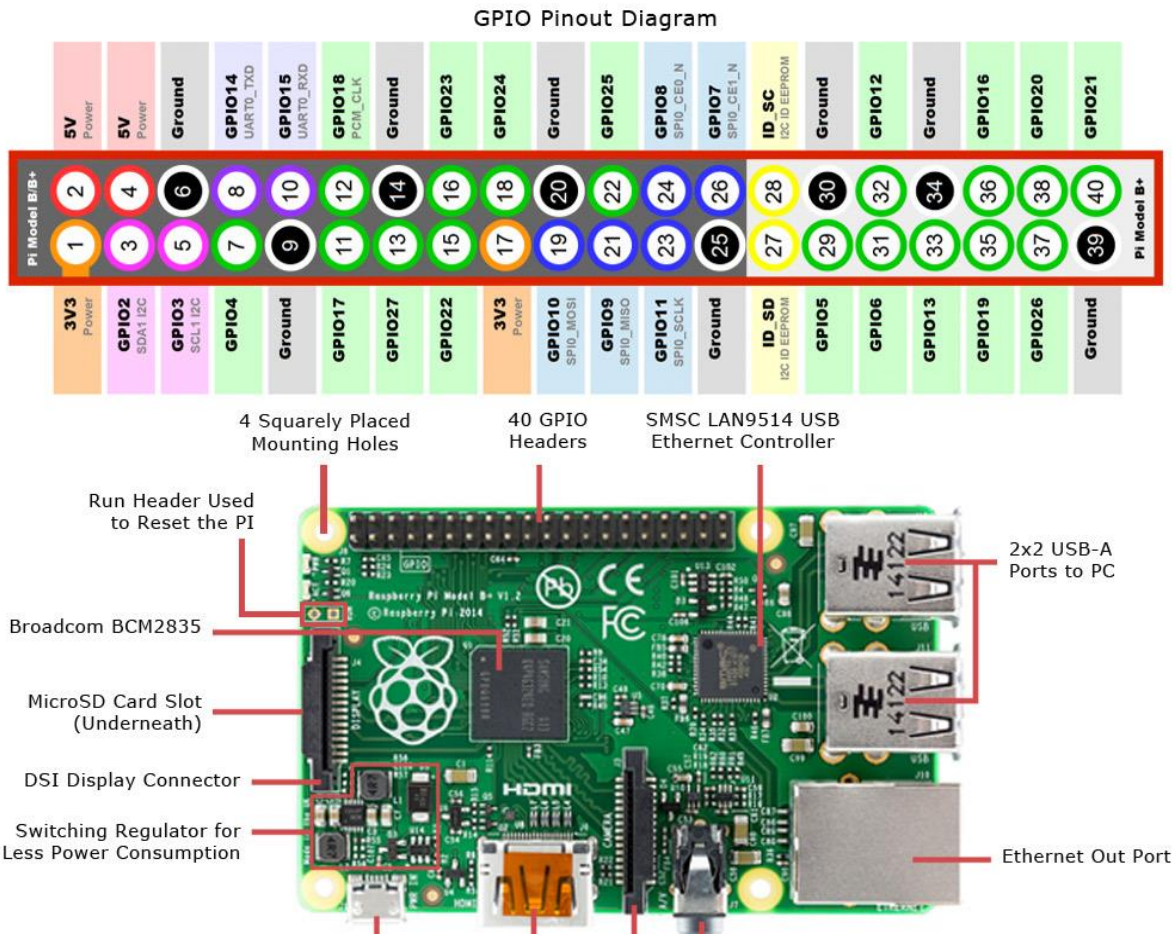
There are three parts to this lab activity

- Hookup and command line
- Shell programming
- C programming using a GPIO library

Raspberry GPIO

We are going to do some practical work on interfacing to the Raspberry PI's single wire input and outputs. The Raspberry has a header containing pins for general purpose input output (GPIO). These pins can be controlled by software programs. Some pins are also used for serial communication protocols such as serial-uart, SPI, and I2C. There are also power pins for 3.3V, 5V and ground. The header is male pins so you need a breakout cable designed for the PI or hookup wires that have female ends to go over the pins.

!!!! BE CAREFUL – YOU CAN DAMAGE YOUR PI IF YOU SHORT 5V or 3.3V to Ground !!!!!!!!!!!!!!!!!!!!!!!!!
DON'T HAVE LOOSE WIRES AND PARTS NEAR THE PINS
The GPIO pins themselves seem to be more tolerant of static discharge than the FPGA boards however.



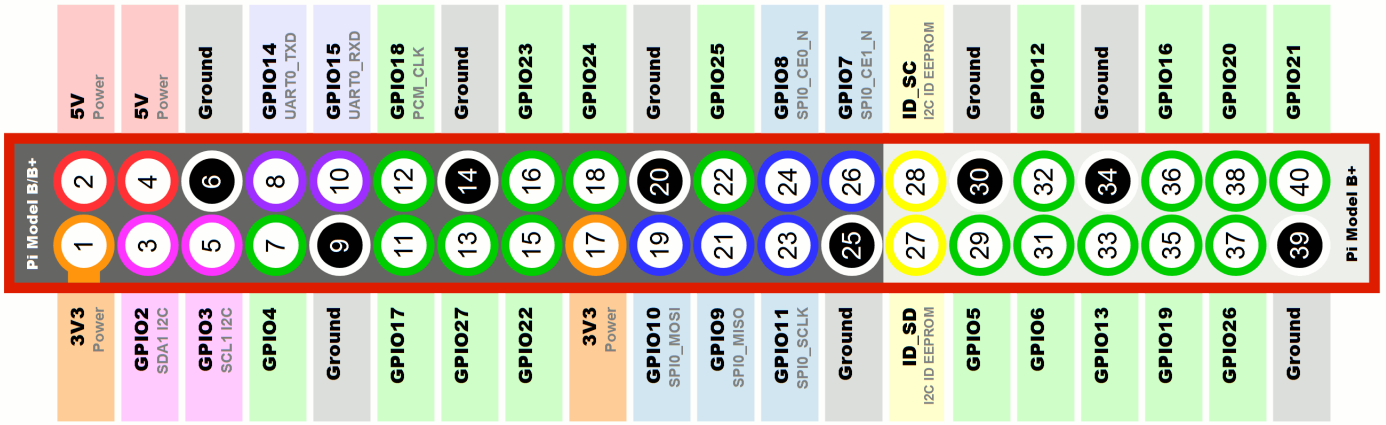
Warnings and Tips

DON'T SHORT OUT ANY PINS!!!!!!!!

The maximum current from the 3.3V pin is 50 mA

The ideal current for sinking or sourcing from a GPIO pin is 3 mA however if you only have one or two LEDs attached to the entire board, a single pin can sink/source around 10 mA. If have a lot of LEDs then you'll be using a transistor to drive the LEDs like you did on the mini-oscope digital lab.

The pins have two numbering systems:



The pin number on the header – good for finding the right pin to connect to

The Broadcom CPU/GPU pin – the one you actually use in your program

For example header pin 11 == GPIO 17

Command line and Shell programming

First you'll interface to the GPIO pins using the command line interface and then create a short shell program (or script). When you are working with the command line, sometimes you'll want to automate the steps into a script so you don't have to retype them every time.

The shell script uses a different syntax which doesn't resemble the C syntax you've been learning so you'll just copy the script here.

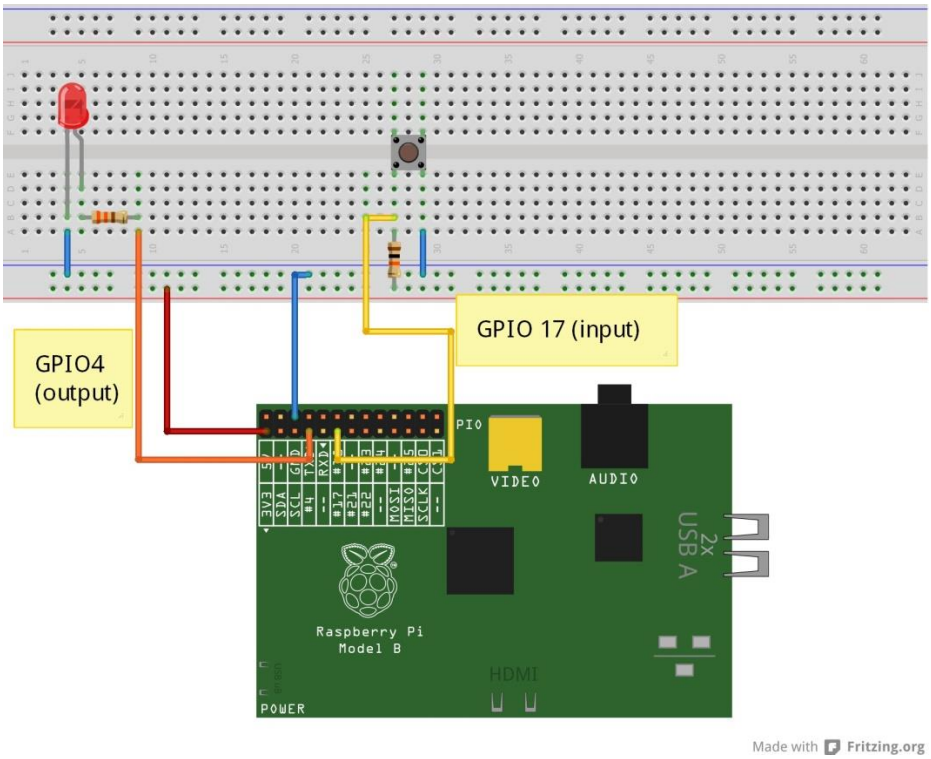
C programming

The next level of automation is to write a program. Someone has written a library which gives you functions to call. These functions are kind of like the math functions you briefly used in section 1 of zybooks. This library is almost identical to the way you will program microcontrollers in the fall. This section serves as a good introduction to that.

Command Line: Instructions/Procedures

1. Read your raspberry PI users guide section on the GPIO. You can read the material on interfacing to the GPIO with python but since we are C++ and microcontroller focused, we won't be using python. For more information, google "raspberry PI gpio" and any of the top 10 tutorials are good except that 99% of internet tutorials use python. They are helpful understanding the connections and general program flow – but again we are going to use C++ with a quick diversion off into shell scripts.

2. Connect an LED/resistor to GPIO4 and a switch with a pull up to GPIO17 (alternatively you can use any pair of GPIO pins if 4 or 17 seem to be not working).



3. Make the LED turn on and off. Open up a terminal window. You should be in your home directory. We will use the shell commands "echo" and the output redirect '>'. Type:

```
ls /sys/class/gpio
```

What do you see in that directory?

```
echo 4 > /sys/class/gpio/export
```

```
ls /sys/class/gpio
```

The echo command sends the value '4' to a special file (not one you can actually see). It tells the operating system that GPIO pin 4 is now active and can be set to 5V (1) or 0V 0).

What do you see now in that directory after running the 'ls' command?. You can "cd" into the new directory to see what new files were created to handle activities on GPIO4.

```
echo "out" > /sys/class/gpio/gpio4/direction
```

```
echo 1 > /sys/class/gpio/gpio4/value
```

What we did there was the following:

1. Told the operating system to enable GPIO4 – the echo command sent the value 4 to a file called export. Now this looks like a file but it's really a method to supply values to the operating system kernel. Think of it like an ordering window at a fast food restaurant. It has a location in the directory structure but it really isn't a file. This operation created a directory called gpio4 in the /sys/class/gpio directory that's needed for the next step

2. Told the operating system to set GPIO4 to be an output.

3. Told the operating system to turn GPIO4 "on" – a logical 1, a voltage of 3.3V

Now type:

```
echo 0 > /sys/class/gpio/gpio4/value
```

You just turned the pin "off" or set it to ground. Now clean up by typing:

```
echo 4 > /sys/class/gpio/unexport
```

If you go look into the /sys/class/gpio directory, you'll see the GPIO4 subdirectory gone. This releases the resource for any other users, terminal windows, or programs to use. If you ever get a device or resource busy message for a GPIO pin from any operation, type this command into a terminal window, substituting the relevant GPIO number for the one you are using.

TROUBLESHOOTING: IF YOU KEEP GETTING A "DEVICE OR RESOURCE BUSY" ERROR MESSAGE, TRY THE UNEXPORT COMMAND AND IF THAT DOESN'T WORK TRY A DIFFERENT PIN (SOMETIMES GPIO PINS GET DAMAGED)

Shell Programming: Instructions/Procedures

1. Let's automate those command line operations with a shell script. A shell script is a quick program that engineers and technicians use to automate linux commands. Some companies actually look for shell programming skills on resumes. The shell script programming language is not C++. It has variables and loops but the syntax is much different. It uses a while loop but the syntax is a little different – but you just need to enter the script as-is.

Using your nano text editor, enter the following text and create a file called "flash_test.sh".

```
#!/bin/bash
echo 4 > /sys/class/gpio/export
sleep 0.5
echo "out" > /sys/class/gpio/gpio4/direction
COUNTER=0
while [ $COUNTER -lt 100 ]; do
   echo 1 > /sys/class/gpio/gpio4/value
   let COUNTER=COUNTER+1
   sleep 0.5
   echo 0 > /sys/class/gpio/gpio4/value
   sleep 0.5
done
echo 4 > /sys/class/gpio/unexport
```

Save the file.

Some of those commands look like what you were typing in the shell a few minutes ago. Here's an explanation of the other lines:

```
#!/bin/bash          #tells the operating system what kind of script this is – it's a bash shell script
echo …….             #just a message sent to the screen for the user
sleep 0.5            #tells the operating system to wait for half a second for the last command to do its work
echo …….
COUNTER=0            #create and set a variable to 0, no type needed, no need to specify float or int
while [ $COUNTER -lt 100 ]; do
                     #Just like a C++ loop but use [] instead of ()
                     # shell scripts don't use == or > or <, instead they use –lt for "less than"
                     # shell scripts don't use {} to surround code, it uses "do" and "done"
                     # this syntax means while counter is less than 100 keep doing the lines between "do" and "done"
   echo …….
   let COUNTER=COUNTER+1      #same as counter=counter+1
```

```
    sleep 0.5                   #wait ½ second before turning the led off in the next line
    echo …….
    sleep 0.5                   #wait ½ second before turning the led back on at the top of the while loop
done
echo ……….
```

2. To be able to execute this file as a shell script (or program) you need to change the permissions to make it executable. Until you do that, it's just a text file with a funny extension. Type:

```
chmod 755 flash_test.sh
```

```
./flash_test.sh
```

You should see LED flash on and off for 100 times before the shell prompt comes back. You can only run this in the directory you created the script in.

TROUBLESHOOTING: IF YOU KEEP GETTING A "DEVICE OR RESOURCE BUSY" ERROR MESSAGE, TRY THE UNEXPORT COMMAND AND IF THAT DOESN'T WORK TRY A DIFFERENT PIN (SOMETIMES GPIO PINS GET DAMAGED).
IF YOU PRESSED CTRL-C TO STOP THE PROGRAM IN THE MIDDLE IT WILL LEAVE THE GPIO PIN AS 'ACTIVE' AND YOU WILL HAVE TO 'UNEXPORT' IT AGAIN BEFORE YOU CAN RUN ANYTHING

C programming: Instructions/Procedures

You can use the above commands in a C program. Someone has developed a library to give programmers functions that interface to the GPIO. We're using the library explained here - http://wiringpi.com/

It is based on Arduino microcontroller programming. We won't get into any depth here. The programs are given to you here. All you need to do is make a few changes using what you learned in zybooks.

1. The library should be already installed on your raspberry. Type:

```
gpio -v
```

If you see lots of information about pins and GPIO, then the package is already installed. If you get an error, then install wiringPI:

ONLY DO THIS IF WIRING PI IS NOT INSTALLED!!!

Navigate to your home directory. Follow the steps at this website, you may need to run the build command with sudo:

http://wiringpi.com/download-and-install/

2. Let's program!

Here is a sample program. It uses a few things we haven't covered in class yet but I'll explain them as we go along. Using your text editor or the geany IDE, create the following file called flash.c

```
#include <wiringPi.h>  //tell the compiler you are using this library
#include <stdio.h>    //this is in all our programs

#define LED_GPIO  4    //a new type of statement that lets us use "LED GPIO"
                       //instead of '4' in the code below. If we ever need
                      //to change a pin, you only have to change this line

int main() {
   wiringPiSetupGpio();                // use GPIO and not header pin numbers
   printf("Starting LED flashing on GPIO %d\n", LED_GPIO);
   printf("Press CTRL+C to stop\n");
   pinMode(LED_GPIO, OUTPUT);          // GPIO4 is an output pin
   unsigned long int i;                //declare a counter variable
   while(1) {                          // loop forever
      digitalWrite(LED_GPIO, HIGH);    // LED on

      for(i=0; i<5000000; i++) { }      // a delay loop

      digitalWrite(LED_GPIO, LOW);     // LED off

      for(i=0; i<5000000; i++) { }      // a delay loop

   }
   return 0;
}
```

The "for loop" in this program simply tells the computer to waste a bunch of time doing i++ and checking that it's still under 5000000 (a very big number). The computer works very very fast but each time it does that, it does take a few nanoseconds. The pseucode looks like this:

Keep doing the following lines forever or until ctrl-c is pressed

Turn the LED attached to the pin ON

Kill time by doing all this math until variable i gets big enough

Turn the LED off

Kill time by doing all this math until variable i gets big enough

You can use an IDE to enter and run the program or you can do it on the command line:

Command line method

To compile and run – type:

```
gcc flash.cpp -o flash -lwiringPi

./flash
```

IDE method

You can use the geany IDE to develop your code on the raspberry. You will have to modify a line under set build commands to make it work. See this video tutorial: https://youtu.be/mu9NOsS5aUY

You may have to put "sudo" in front of the command. It depends on how your raspberry is setup.

7. Lets review some of that program you just entered. There are four important functions that got called. We may not have covered functions in class yet so I'll do some explaining here. These functions are described at this page - http://wiringpi.com/reference/core-functions/

void wiringPiSetupGpio();

> void means it doesn't return any values to the program. It just does it's thing.
>
> The () means it doesn't take any inputs
>
> Calling this function means that any numbers used in the other functions refer to the GPIO header pin numbers and not the actual header pin numbers. For example 17 would mean GPIO17 not header pin 17 even though GPIO17 is at header pin 11. Refer to the chart at the top.
>
> ***Example usage: `wiringPiSetupGpio();`***

void pinMode(int pin, int mode);

> void means it doesn't return any values to the program
>
> (int pin, int mode) means it takes two input numbers and they are both of integer type. The first one is the GPIO pin you want to set and the second number is a value that sets the pin to be an input or an output. When you include the wiringpi library, it creates an integer constant of INPUT=0 and OUTPUT=1 so that it's easier to just use INPUT and OUTPUT instead of remembering what 0 and 1 mean.
>
> A GPIO pin has to be either an input or an output and this function sets that
>
> ***Example usage: `pinMode(LED_GPIO, OUTPUT);`***
>
> This one sets GPIO4 to be an output. If you look above LED_GPIO was defined to be 4.

void digitalWrite(int pin, int value);

> void means it doesn't return any values to the program
>
> This function sets the pin to the value you provide. When you include the wiringpi library, it defines HIGH to be 1 (3.3 volts) and LOW to be 0 (0 volts)
>
> ***Example usage: `digitalWrite(4, HIGH);`***

This will set GPIO4 to 3.3V.

int digitalRead(int pin);

the "int" at the front means when this function is called, it returns an integer value

This function reads the voltage on the pin you specify. If the voltage is 0 V, then a 0 is returned. If the voltage is 3.3V, then an integer 1 is returned.

Example usage: `x = digitalRead(17);`

If 3.3V is connected to GPIO17, then variable x will be set to the value 1 after this function executes.

Exercises

1. Change the timing so that the flash rate is one second on, one second off. Try increasing the number in the for loop.

2. What does the "digitalWrite" function do and what arguments does it take? What voltage levels are HIGH and LOW?

3. Use the internet to find documentation for all the functions that wiringPi provides (try wiringpi.com)

4. Find a wiringPI function to replace the for loops that are used as a time delay and repeat exercise 1 above (hint: look at the reference section of the wiringpi web page - http://wiringpi.com/reference/timing/). Find the delay function and implement it in place of the for loop.

void delay (unsigned int howLong)

example usage: delay(1000); //a 1000 millisecond delay or 1 S

5. There is a wiringPi function called "int digitalRead(int pin)". Given a GPIO pin number as the argument, it will return an integer 1 or an integer 0 depending on the voltage on the pin. Write a new program that reads the voltage on the pin and turns the LED on or off. You will need to create a new integer variable to store the result of the digitalRead and use that variable in an if statement that controls the LED.

Instructor Checkpoints (demonstrate)

Usually these are done in person but to facilitate remote signoffs, you can create a video, using your phone most likely of the following items. Each part should include you showing the code on a screen, running the command or program and then showing the LED blinking. You should narrate what you are doing. When done either create a youtube link or upload to D2L (not sure about size limits and mp4 seems to work best). You can also copy to your google drive and share the file with me.

1. **Demonstrate the shell script blinking**
2. **Show the version of the program that has the for loop replaced by the delay function**
3. **Demonstrate the program from step 5 in the exercises, that uses the button.**