Introduction to Computer Science and Engineering
RPi Interfacing Lab

Introduction

As you look at the smart devices around you, many of those devices contain an embedded microprocessor running C or C++ interfaced with devices ranging from cameras, microphones, and speakers to cellular chips or WiFi chips that allow the computer to communicate with the outside world. The job market of embedded systems and edge computing often involves this fusion of sensors and computing functions.

In this lab, we will interface simple real-world sensors and devices, using some of the concepts in the first two digital labs.

You will also learn more about these concepts in CSE 2312 (Computer Organization and Assembly Language Programming), CSE 2315 (Discrete Structures), CSE 2441 (Digital Logic Design), CSE 3320 (Operating Systems), and CSE 3442/4342 (Embedded Systems I / II).

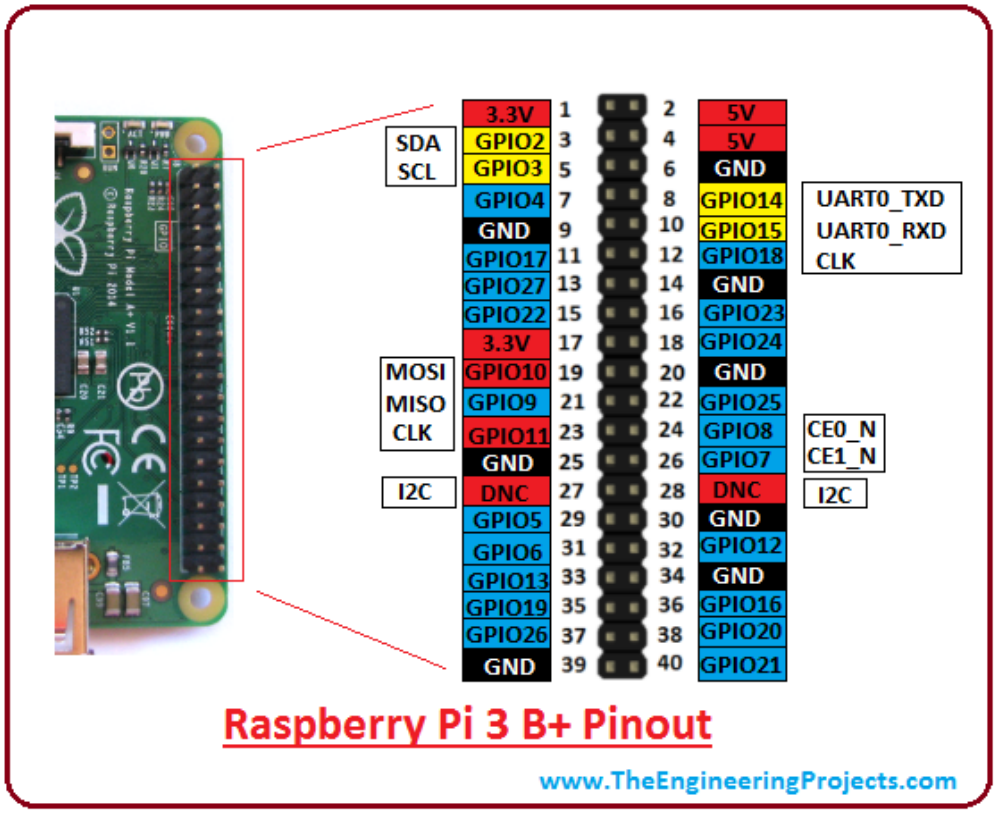
Virtual File System

In Linux, you may have heard the expression "everything is a file" in your first programming classes like CSE 1310 and CSE 1320. What does this really mean? For the purpose of this lab, consider the directory /sys/class/gpio. You can issue this command to get to this directory on your RPi:

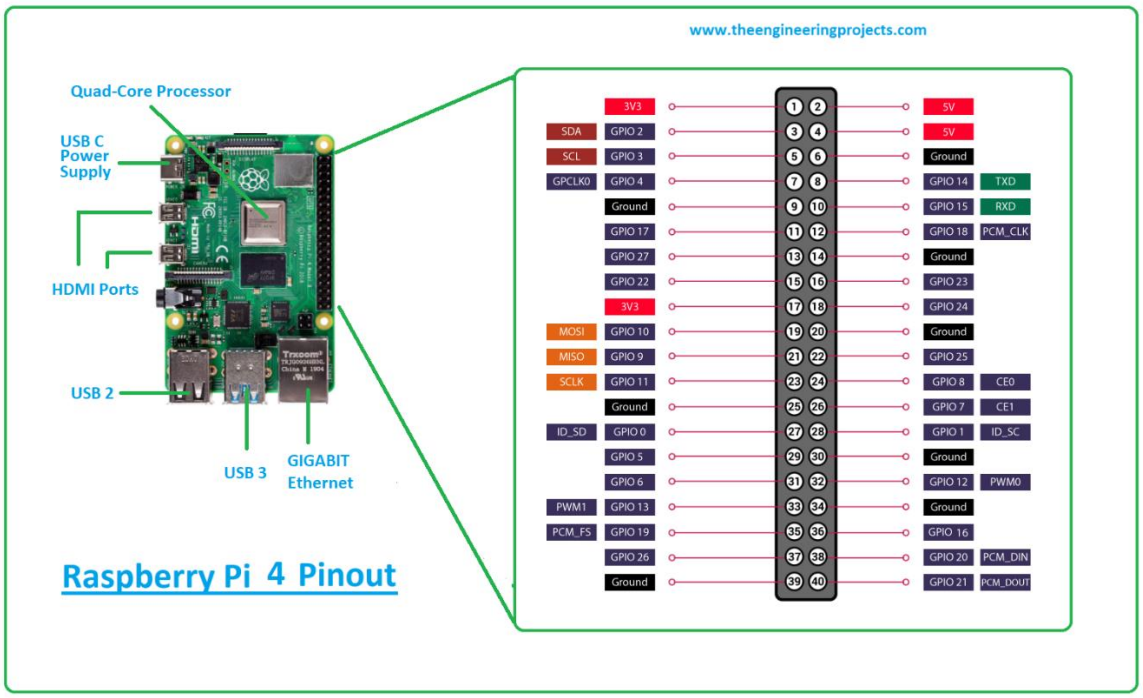
```
cd /sys/class/gpio
```

These are not files in the sense that "index.html" or "todo.txt" is a file, but rather special files (virtualization allows this abstraction) that when you use the fopen(), fprintf(), fscanf(), and fclose() functions in stdio.h, you are actually interacting with a general-purpose input-output (GPIO) device driver that allows you to control devices like LEDs and get the status of switches, like those used in the digital labs.

On the RPi 3b/3b+, you will find a 40 pin header when the top lid is removed as shown below:



The 40 pin header has a similar layout on the RPi 4.



Using the Dupont jumper leads with a socket on one end and pins on the other end, it is possible to interconnect pins on the header with contacts on the white breadboard used in the digital labs.

Warning:

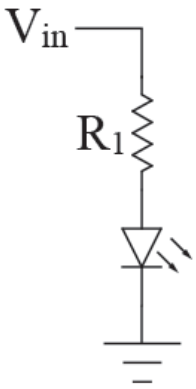
Before beginning, make sure that the USB connector on the white board is unconnected to the white board. The white board will be power only from the RPi 40 pin header during this lab.

If you chose to ignore this warning, you can back feed 5V from the USB connector to the 40 pin header of the RPi, damaging the RPi.

In other words, if you connect 5V to many of the pins on the 40 pin connector, you will damage the RPi.

Writing to an output device:

For our first experiment, let's connect a red LED to a pin on the RPi. Using a black wire, connect one of the GND pins on the RPi to the black wires on the white breadboard so that ground (0V) is connected. Then using the circuit below with R1 = 220 ohms as in the digital labs, connect Vin using any color wire except red or black to one of the blue colored GPIOx pins above, noting the value of x.



Issue the following command to see what special files exist for GPIO control:

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

Now, substituting the value of x below with the pin number, type the following command to "export" the pin for control:

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

Issue the following command to see what special files now exist for GPIO control, noting the changes:

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

Now, look in the new directory and see the new special files:

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

Next, tell the OS we want the pin to be an output, so it can drive the LED:

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

These two echos are writing to the special files named "export" and "direction" and together are equivalent to this C function:

```
void gpioOutput(int pin)
{
    FILE* file;
    char str[35];
    file = fopen("/sys/class/gpio/export", "w");
    fprintf(file, "%d", pin);
    fclose(file);
    sprintf(str, "/sys/class/gpio/gpio%d/direction", pin);
    file = fopen(str, "w");
    fprintf(file, "out");
    fclose(file);
}
```

Now, turn the LED on and off by issuing the following commands:

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

or

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

This is equivalent to the following C function:

```
void gpioWrite(int pin, int value)
{
    FILE* file;
    char str[35];
    sprintf(str, "/sys/class/gpio/gpio%d/value", pin);
    file = fopen(str, "w");
    fprintf(file, "%d", value);
    fclose(file);
}
```

Reading from an input device:

For our second experiment, let's connect a slide switch to a GPIO pin so we can read the status from software.

Using a black wire, connect one of the GND pins on the RPi to the black wires on the white breadboard so that ground (0V) is connected (you should already have your ground wire in place after the previous section). Next, using a red wire, connect one of the 3.3V pins on the RPi to the red wires on the white breadboard so that power (3.3V) is connected.

Warning:

Make absolutely sure not to connect anything to the 5V pins on the RPi 40 pin header. If the 5V gets connected to any of the GPIO pins on the RPi 40 pin header, the RPi can be damaged.

Again, never use the 5V pins for this lab.

After verifying that neither of the 5V pins on the 40 pin header are connected and that the USB connector on the white breadboard is disconnected, connect the center pin of a slide switch, using any color other than red and black, to any GPIOy pins above, noting the value of y.

Issue the following command to see what special files exist for GPIO control:

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

Now, substituting the value of x below with the pin number, type the following command to "export" the pin for control:

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

Issue the following command to see what special files now exist for GPIO control, noting the changes:

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

Now, look in the new directory and see the new special files:

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

Next, tell the OS we want the pin to be an input, so we can read the status of the switch:

```
echo "in" > /sys/class/gpio/gpioy/direction
```

These two echos are writing to the special files named "export" and "direction" and together are equivalent to this C function:

```
void gpioInput(int pin)
{
    FILE* file;
    char str[35];
    file = fopen("/sys/class/gpio/export", "w");
    fprintf(file, "%d", pin);
    fclose(file);
    sprintf(str, "/sys/class/gpio/gpio%d/direction", pin);
    file = fopen(str, "w");
    fprintf(file, "in");
    fclose(file);
}
```

Now, read the status of the slide switch by issuing the following commands:

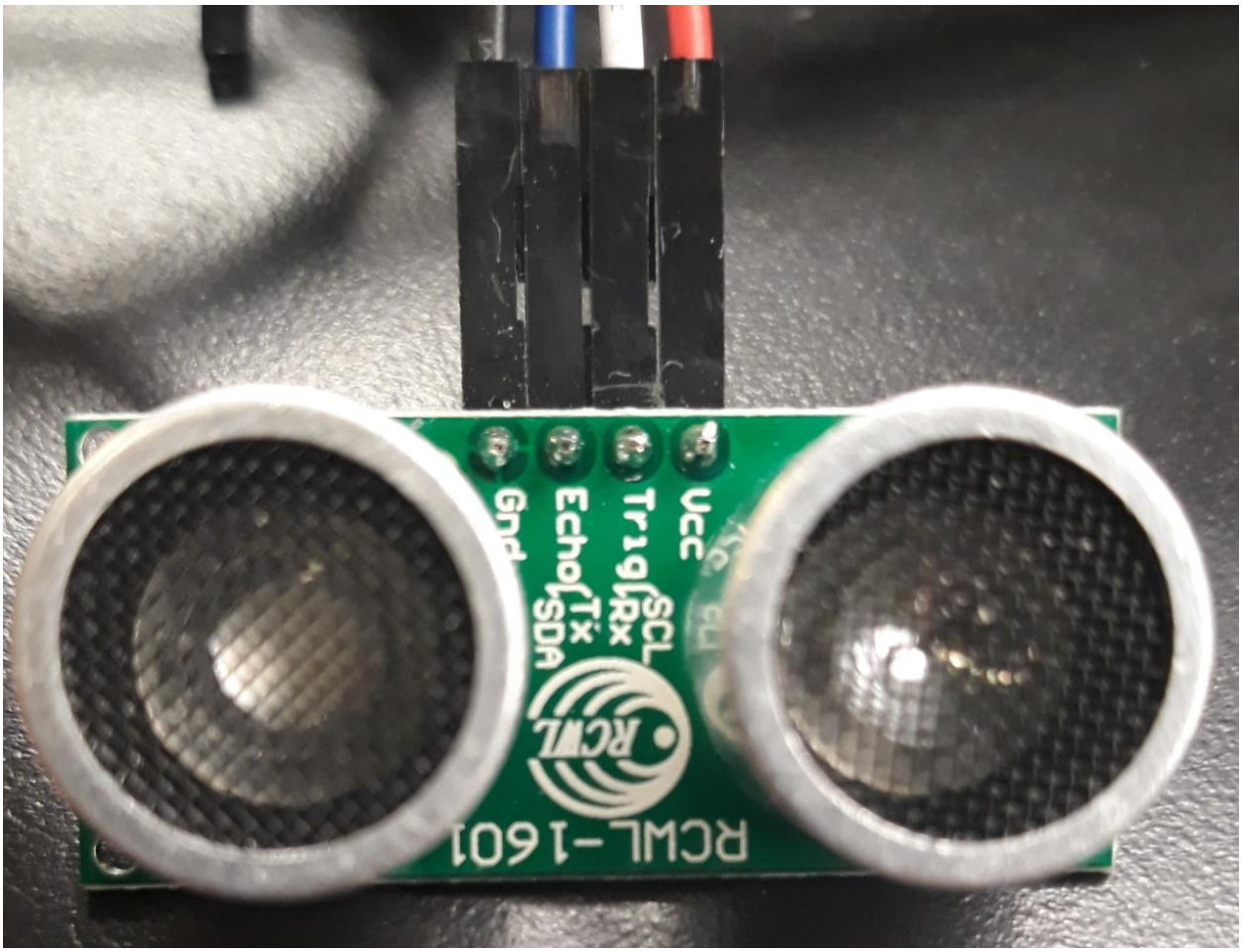
```
cat /sys/class/gpio/gpioy/value
```

This is equivalent to the following C function:

```
int gpioRead(int pin)
{
    FILE* file;
    int result;
    char str[30];
    sprintf(str, "/sys/class/gpio/gpio%d/value", pin);
    file = fopen(str, "rw");
    fscanf(file, "%d", &result);
    fclose(file);
    return result;
}
```

Ultrasonic range finder:

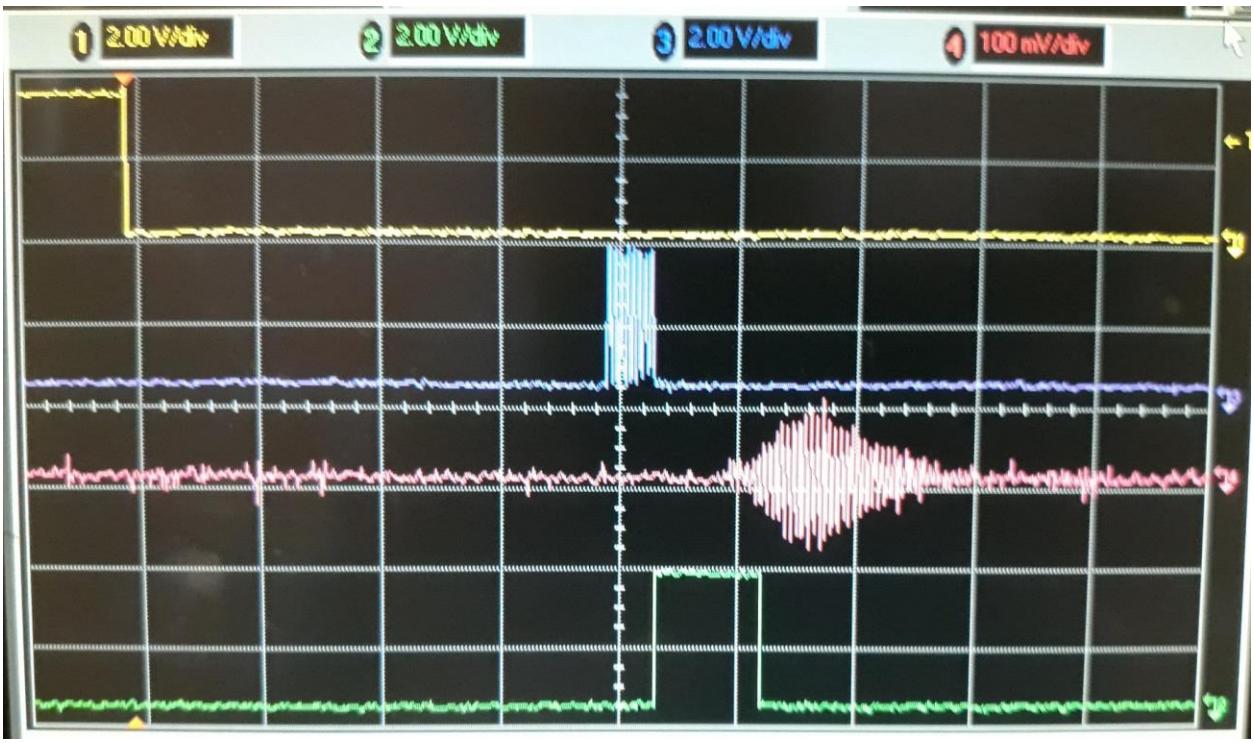
The RWCL-1601 ultrasonic range finder is a simple device that is both an output and an input device. The device is shown below:



The VCC (red wire) and Gnd (black wire) connect to 3.3V and GND on the RPi 40 pin header. The Trig input (white wire) connects to a GPIOx pin. The Echo output (blue wire) connects to a GPIOy. To make things simple, x and y can be treated like the LED (x) and switch (y) in the above sections.

The operation of this device is simple. Software pulses the Trig input high and then low. Then the pin goes low, the module sends an ultrasonic (above human hearing range) that travels out one of the round silver-colored transducers and sets the Echo output high. This ping then bounces off an object and is received on the other round transducer. When the signal is received, the Echo output is set to low.

Using an instrument called an oscilloscope, which captures all of this signal in time, you get the following picture:



Looking at the above oscilloscope capture, let's go through the series of events from left to right (time is the horizontal axis).

1. Software pulses the Trig input (yellow trace) high and then low.

2. The module sends a short ping of sound (blue trace) and sets the Echo output high (green trace).

3. The ping travels until it hits a target and then returns. The longer the distance, the longer the Echo output (green trace) is high.

4. After the round-trip delay (sound travels around 340m/s), the signal is received at the module (pink trace).

5. When the signal is received, the Echo output goes low (green trace).

In the figure above, the Trig input is driven by an output pin (GPIOx) on the RPi header and the Echo output drives an input pin (GPIOy) on the RPi. Software measures the time that Echo output is high to determine distance.

The blue and pink traces can be measured with test equipment and are shown for clarity, but they are not available for software to "see".

RPi Interfacing Lab Worksheet **Name _____**

Course/Section _____

Please complete the following steps to complete the RPi interfacing lab:

1. Make sure that the USB connector on the white breadboard is not connected. Failing to do this could damage the RPi in later steps.

2. Connect a black wire from a GND connection on the RPi header to the ground/0V (black wires) on the white breadboard.

3. Connect a red wire from a 3.3V connection on the RPi header to the red wires on the white breadboard. Make absolutely sure not to use the 5V connections on the RPi header. Again, only use the 3.3V connections! Failing to do this could damage the RPi in later steps.

4. Take a green LED (not a blue LED) and a 220 ohm resistor (red-red-brown) in series as shown in the circuit above and connect them across the red and blue distribution strip. It should glow brightly. 3.3V is not enough voltage potential to illuminate the blue LED.

5. Connect a red LED and a 220 ohm resistor (red-red-brown) in series as show In the circuit shown earlier and connect them through a different colored wire (not red or black) and connect to any of the GPIOx pins on the RPi header.

6. Using the steps in the "Writing to an output device" section, verify that the LED turns on and off using echo commands. Screen capture the shell commands for submission later.

7. Connect a slide switch center contact through a different colored wire (not red or black or the color in 5) and connect to any of the GPIOy pins on the RPi header.

8. Using the steps in the "Reading from an input device" section, read out the switch status using echo and cat commands. Screen capture the shell commands for submission later.

9. Write a program, led_pb.c, compiled with GCC on the RPi 3b/3b+, that performs the following steps using the provided sysfs_gpio files:

a. Include the sysfs_gpio.h file
b. Initializes the LED pin to be an output by calling gpioOutput()
c. Initializes the switch pin to be an input by calling gpioInput()
d. Turns on the LED using gpioWrite()
e. Enters a while loop that calls gpioRead() and waits for the switch to read 1
f. Turns off the LED using gpioWrite()

Note, you would compile the program from the terminal with:

```
gcc -o led_pb led_pb.c sysfs_gpio.c
```

Using GPIO requires elevated privilege, so you should run the program with

sudo ./led_pb

10. Take a picture of your setup and disassemble the circuit.

11. Connect the ultrasonic sensor directly to 3.3V, GND, GPIOx (trigger input), and GPIOy (echo output) referencing the "Ultrasonic range finder" section.

12. Write a program, sonar.c, compiled with GCC on the RPi 3b/3b+, that performs the following steps using the provided sysfs_gpio files:

a. Include the sysfs_gpio.h file
b. Initializes the TRIGGER pin to be an output by calling gpioOutput()
c. Initializes the ECHO pin to be an input by calling gpioInput()
d. Sets Trig signal high (1) and then low (0)
e. Zero a count variable
f. Enter a while loop that waits until the Echo signal goes high
    There is no code in the while loop
g. The program exits the first loop
h. Enter another while loop that waits until the Echo signal goes low
    The while loop contains code that increments the count variable
i. The program exits the second loop
j. The count value is now proportional to the distance traveled
k. Print the raw count value
l. Convert the value to a distance (cm) by empirical means
    (the desk is about 4 feet or 122 cm in length)

Aside: How can this work? It takes a finite amount of time to execute each line of C code. For each time step (h) checks the Echo signal and finds it is still high, the count variable is incremented.

13. Run the program and take a screen capture of the program running for submission later.

14. Take a picture of the sensor connected to the RPi.

15. Lab checkout steps:

a. Show your working circuit and program to the grader.

b. Create a file, lastname_netid_lab6.zip, that includes the following files:

- A JPEG image of your working circuit from steps 1-10

- The shell command capture from step 6

- The shell command capture from step 8

- The source code your program from step 9 (as a .c file)

- The output of your program from step 9

- A JPEG image of your ultrasonic sensor connected to the RPi from steps 11-14

- The source code your program from step 12 (as a .c file)

- The output of your program from step 13

- Note: images do not need specific file names

c. Upload the zip file to Canvas.

d. After you have your pictures and send the zip file, dismantle your circuit and return everything to the box on the shelf. Do not remove the slides switches or the pre-loaded red and black wires.

Thank you for attending the lab. We hope some of this material was new and interesting to you.

Drs. Losh and Eary

Lab 6 Rubric

Prerequisite:
-40: Did not complete familiarization quiz on time

Assignment:
-20: No picture of LED and switch circuit
-20: No picture of ultrasonic sensor
-20: No source code for LED and switch circuit (as a .c file)
-20: No source code for ultrasonic sensor (as a .c file)
-10: No photos of output from the command line
-10: Wrong file name