Lab 07: Introduction to Raspberry Pi

UCR CS 122A

Pre-lab

Before coming to lab you **MUST** set up your Raspberry Pi Zero W so you will be able to SSH into it over a simple micro-USB cable. Follow these instructions to set up your Raspberry Pi. If you are working from your personal machine you can use these instructions to get connected. Note: It is recommended that you work from the lab machines for the labs.

Important: Make sure your ATMega1284 is powered at 3.3V.

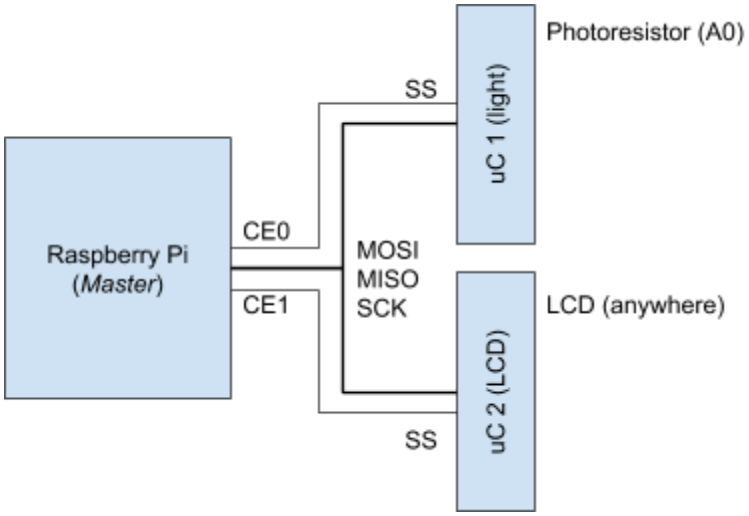
You will also want to read through the lab manual and wire your board up properly so you can start working once you get to the lab.

Introduction

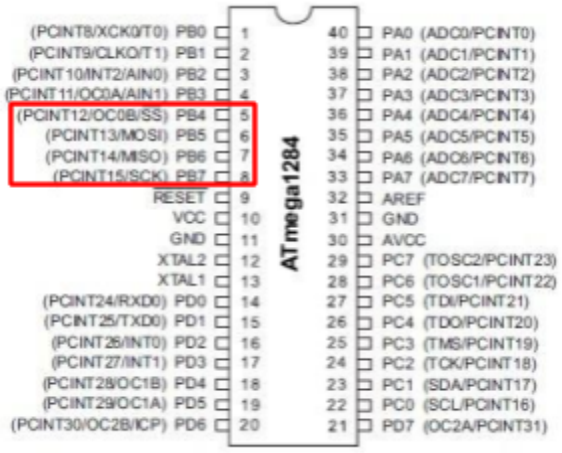
You will be setting up a system the consists of a Raspberry Pi communicating over SPI to two microcontrollers (see figure below). The Raspberry Pi will be set as the *master* and both microcontrollers will be set as *servants*.

- **Microcontroller 1** will have a photoresistor connected to PA0 and will be configured to send the data to the Raspberry Pi when requested.
- **Microcontroller 2** will have the LCD screen connected to it.
- The **Raspberry Pi** will request the data from microcontroller 1, calculate a running average of the light values, and send this value to microcontroller 2 to be displayed on the LCD.

The figure below shows the block diagram of how the system will look. MOSI, MISO, and SCK can be shared between the microcontrollers. SS for microcontroller 1 should be connected to CE0 (chip enable 0) on the Raspberry Pi Zero W and SS for microcontroller 2 should be connected to CE1 (chip enable 1) on the Raspberry Pi Zero W. This will allow you to address the microcontrollers separately.



Microcontrollers as *Servants*



Note: You will need to disconnect the SPI on the ATMega1284 when programming it.
When in *servant* mode, the `MISO` pin is configured as output while all other pins are configured as input.

Receiving data

You will want to set the microcontrollers to trigger an interrupt when they have received data from the master (Raspberry Pi). You will need to enable interrupts on SPI (`SPIE`) and enable global interrupts (`sei()`). See page 168 of the datasheet for more details.

Sending data

Since the SPI registers are set up in a virtual ring topology, when the master shifts data in on the `MOSI` line, the data register on the servant (`SPDR`) is shifted out on the `MISO` line. If the `MISO`

line is connected to the master, then it will receive the data that was on `SPDR` at the start of the transmission. This allows the servant to communicate data to the master.

Microcontroller 1

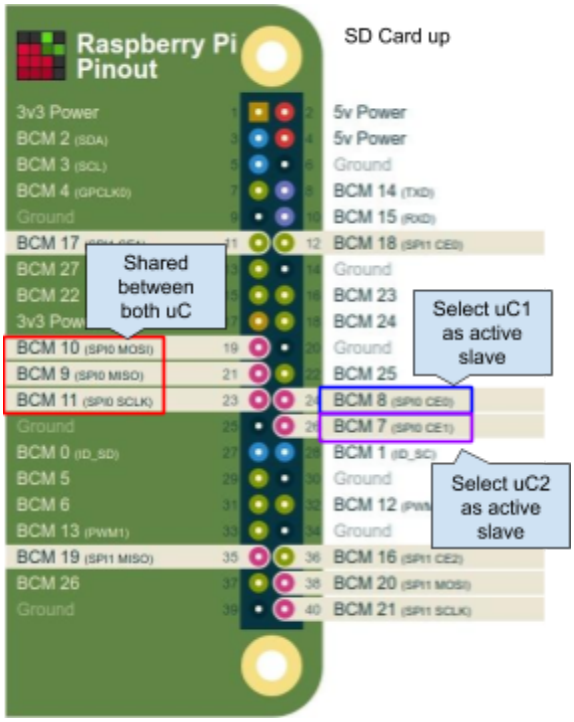
This circuit will have a simple SM (or inline code if desired) to check the ADC every 1 second and update the current local light value. This value will then be put on the `SPDR` register. When the Raspberry Pi "requests" data from this microcontroller it will send a junk value (0xFF) over SPI and receive the light value that is stored on `SPDR`. Later will we will add commands to request data.

Microcontroller 2

This circuit will be displaying the average light value on the LCD screen. When the Raspberry Pi sends data over SPI, the ISR should fire which will take the data sent on `SPDR` (average light value) and populate a local variable with that value. No data needs to be sent back to the Raspberry Pi. The current average light value should be displayed on the LCD.

Raspberry Pi as *Master*

The Raspberry Pi will be configured as the *master* and will be controlling the entire system. You will be reading the current light value from microcontroller 1 (using `CE0`), calculating the average value using Welford's algorithm, and sending that data to microcontroller 2 (using `CE1`).



To set up the Raspberry Pi to access SPI you will need to:

1) Edit `/boot/config.txt`
2) Make sure `dtparam=spi=on` is not commented out
3) Reboot
4) You should now see `/dev/spidev0.0`
   ```
   $ ls /dev/spidev0*
   /dev/spidev0.0  /dev/spidev0.1
   ```

To set up your Raspberry Pi environment:

1. Install **virtualenv**
   ```
   $ sudo apt install virtualenv
   ```
2. Install **git**
   ```
   $ sudo apt install git
   ```
3. Install vim (or another command line text editor of your choice)
4. Set up **virtualenv** for **python3.7**
   ```
   $ virtualenv -p /usr/bin/python3.7 <env_name>
   $ source <env_name>/bin/activate
   ```
   You will now see `(<env_name>)` in front of your username.
   ```
   $ pip install spidev
   ```

You will want to activate your `virtualenv` whenever you work on this lab.

You will be writing a Python script for the Raspberry Pi Zero W to communicate over SPI to the atmega1284's. This script will use the spidev library you just installed (`import spidev`) and implement the following function.

- `createSPI(device)` will create and open and SPI on the CE specified by device (0 or 1)

```
def createSPI(device):
    # Create the SPI object
    # Open the SPI object on bus 0 for the specified device (CE)
    Spi.max_speed_hz = 1000000 # Set the max speed to 1 MHz
    spi.mode = 0 # Set the mode to 0
    return spi
```

You will want to include the following at the bottom of the file. If this is the main file running
`$ python <this_file>`
execute forever (`while True:`) or until keyboard interrupt (ctrl-C).

```
if __name__ == '__main__':
    # Create SPI for uC 1 (light)
    lightSPI = createSPI(0)
    # Create SPI for uC 2 (LCD)
    lcdSPI = createSPI(1)
    try:
        while True:
            # Receive data from microcontroller 1
            # Calculate running average using Welford's algorithm
            average = WelfordsAlgorithm(newLightValue)
            # Send new average to microcontroller 2
            # Delay for 1 second
    except KeyboardInterrupt:
        # Close all open SPI (spi.close())
        exit()
```

Welford's algorithm for computing a running average. This should be defined in the same script.

```
_numValues = 0
_mean = 0
_s = 0
def WelfordsAlgorithm(newLightValue):
    global _numValues
    global _mean
    global _s
    _numValues += 1
    if _numValues == 1:
        _mean = newLightValue
        _s = 0
    else:
        _oldMean = _mean
        _mean = _oldMean + (newLightValue-_oldMean) / _numValues
        _s = _s + ((newLightValue-_oldMean) * (newLightValue-_mean))
   return _mean
```

Exercises

1) Complete the system described above.
2) Update the SPI writing and reading functions to create a command structure that is shared across both microcontrollers. Instead of constantly putting the light data on the SPDR as in the previous exercise, the microcontroller will only put the light data on SPDR after it has received a request (command 0x10). The master will then follow the command with a 0xFF command to receive the light data that was placed on the SPDR

register.

Command (From master)	Action (on servant)	Description
0x10	Put light value on SPDR (or 0xFF if no data)	Master requests light value from servant
0x20	Next data on sent is the average, confirm by placing received data on SPDR	Master sending average light value next
0x40	Next data sent is current light value, confirm by placing received data back on SPDR	Master sending current light value
0xFF	None	“Junk” value used to poll data from Servant

3) (Challenge) Set up this process up to start on boot (systemd). Now, when you reboot your device, it should immediately check if the microcontrollers are connected to the SPI ports and start running this process.

Submission

Each student must submit their source files (`.c, .py`) and test files (`.gdb`) according to instructions in the lab submission guidelines.

```
$ tar -czvf [cslogin]_lab7.tgz turnin/
```

Don’t forget to commit and push to Github before you logout!