ECE471: Embedded Systems – Homework 7
SPI, A/D and Temperature Probe

Due: Tuesday, 3 November 2015, 9:30AM EST

1. Use your Raspberry Pi for this homework.
   You will need an MCP3008 SPI A/D converter as well as a TMP36 temperature sensor (looks like a transistor) that I handed out in class. If you missed class, you can stop by my office to pick these up.

   You can view the datasheet for the MCP3008:
   `http://web.eece.maine.edu/~vweaver/classes/ece471_2014f/datasheets/MCP3008.pdf`
   You can view the datasheet for the TMP36:
   `http://web.eece.maine.edu/~vweaver/classes/ece471_2014f/datasheets/TMP35_36_37.pdf`

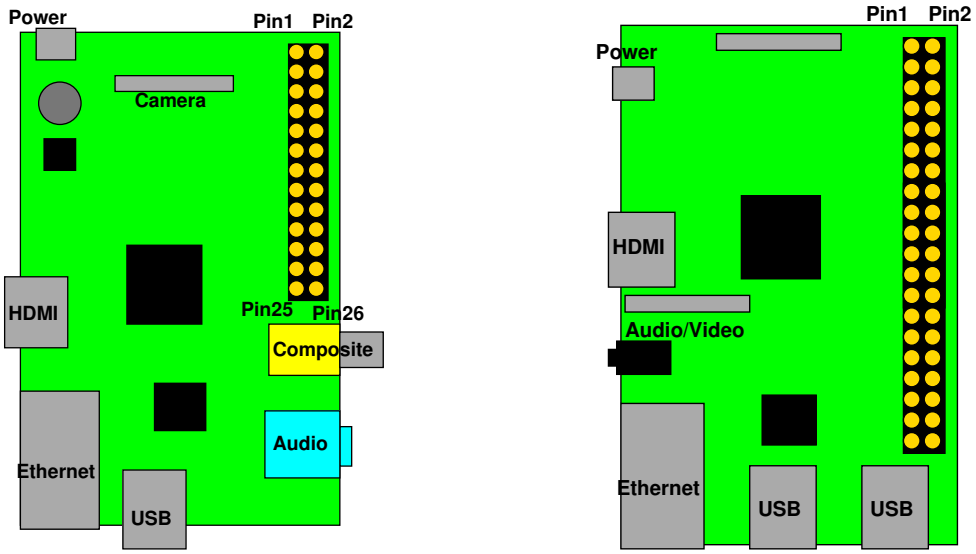

Figure 1: Location of header on Raspberry Pi Model B and B+

2. **Get the MCP3008 providing values over SPI** (3 points)

   First wire up the SPI device to the Raspberry Pi. You can use Figure 1 and Table 1 for guidance.

   (a) Put the MCP3008 on a breadboard and connect to your Pi:

   i. Connect `3.3V` on the Pi to V_{DD} (pin16) on the MCP3008.
   Also connect 3.3V to V_{REF} (pin15) on the MCP3008.

   ii. Connect `GND` on the Pi to `AGND` (pin14) on the MCP3008.
   Also connect GND to `DGND` (pin9) on the MCP3008.

   iii. Connect `SCLK` on the Pi to `CLK` (pin13) on the MCP3008.

   iv. Connect `MOSI` on the Pi to D_{IN} (pin11) on the MCP3008.

   v. Connect `MISO` on the Pi to D_{OUT} (pin12) on the MCP3008.

   vi. Connect `CE0` on the Pi to $\overline{CS}$ (pin10) on the MCP3008.

Table 1: Raspberry Pi Header Pinout

3.3V	1	2	5V
GPIO2 (SDA)	3	4	5V
GPIO3 (SCL)	5	6	GND
GPIO4	7	8	GPIO14 (UART_TXD)
GND	9	10	GPIO15 (UART_RXD)
GPIO17	11	12	GPIO18 (PCM_CLK)
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10 (MOSI)	19	20	GND
GPIO9 (MISO)	21	22	GPIO25
GPIO11 (SCLK)	23	24	GPIO8 (CE0)
GND	25	26	GPIO7 (CE1)
ID_SD (EEPROM)	27	28	ID_SC (EEPROM)
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

(b) For this first part, we will measure 0V on CH0 (pin1) and 3.3V on input CH1 (pin2). Hook up power and ground to those pins on the MCP3008.

(c) Enable SPI support in Linux running on your Pi. To confuse things, the name of the SPI driver has changed recently. If you try the below directions with `spi-bcm2835` and it doesn't work, then try using `spi-bcm2708` instead.

You can check check if the driver is already installed by running `lsmod` and seeing if `spi-bcm2835` is listed.

If not you need to install it. You can do this by `sudo modprobe spi-bcm2825`
`sudo modprobe spidev`
You can have these kernel modules loaded automatically at boot time by either running raspi-config / advanced / spi or else by editing `/etc/modules` and putting:
`spi-bcm2835`
`spidev`
In the file.
You may also have to comment out (by putting a # at the start of the line) the `spi-bcm2835` line in the file `/etc/modprobe.d/raspi-blacklist.conf`

You can look at `dmesg | grep spi` to see if SPI support was found and enabled

(d) Modify the `test_spi.c` file to read values from the MCP3008 and print them to the screen. See the classnotes for more details.

- Open the `/dev/spidev0.0` file for read/write access.
- Use `ioctl` to set the mode to `SPI_MODE_0`
- Use `ioctl` to set the bitsperword to 8.

- Use `ioctl` to set the max frequency to 100kHz.
- In an infinite loop, read the value of CH0 and CH1 once a second and print the voltages to the screen.
  - As described in class use `ioctl` to write 3 bytes: the first includes the start bit, the second says to use single-ended mode and is followed by 3 bits indicating the channel to read. The rest of the byte (and the next byte) should be 0.
  - This will return 3 bytes. The first byte can be ignored, the bottom 2 bits of the second byte are bits 9 and 8, and the third byte is the bottom 8 bits of the result.
  - Get the 10 bits as an integer, then use $V_{IN} = \frac{value \times V_{REF}}{1024}$ to convert to a floating-point voltage.
  - CH0 should be roughly 0V and CH1 should be 3.3V.

(e) Be sure to comment your code and check for errors!

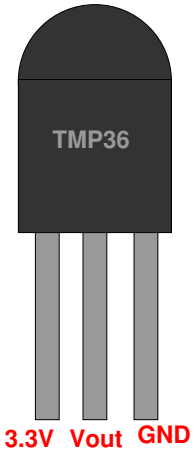

Figure 2: TMP36 Pinout

3. **Hook up the TMP36 to the SPI device** (3 points)

(a) Copy your test_spi.c file to display_temp.c

(b) Connect the TMP36 temperature probe to CH2.
Connect pin1 (3.3V) of the TMP36 to 3.3V
Connect pin2 (Vout) of the TMP36 to CH2 on the MCP3008
Connect pin3 (GND) of the TMP36 to ground
WARNING! the datasheet shows the pins from the *bottom* not the top. If you reverse the power/ground settings on the chip it will quickly heat up to 100+ degrees and will possibly be ruined! Follow the diagram in Figures 2 and you will be OK.

(c) Modify the code to print the current temperature as read by the probe, once a second. You can print degrees C or F as per your preference, but make sure the units are displayed.

(d) The temperature can be determined with the following equation:
$deg\_C = (100 \times voltage) - 50$

(e) Also the following might be useful:
$deg\_F = (deg\_C \times \frac{9}{5}) + 32$

(f) Be sure to comment your code!

4. **Something Cool** (1 point)

   Copy your code to `temp_cool.c` and modify it to do something cool.

   - Monitor the temperature, and after a while print the high/low temperatures recorded.
   - Monitor the temperature and print a message if a temp is exceeded (for example, print a message if someone touches the probe long enough to raise the temperature).
   - Note: Lab9 is probably going to be to display the temperature on the LED display so while it will be cool to do that, you can wait a few weeks.

5. **Questions** (2 points)

   Answer the following in the README file:

   (a) You are designing an embedded system for a car that controls the anti-lock brakes. The specification says that to work properly the brakes needed to start pulsing within 10ms. Would this be a hard, firm, or soft real-time task? Why?

   (b) You are designing another part of the car. The specification says that if you push the "tune" button on the stereo that it should switch stations within 1s. Would this be a hard, firm, or soft real-time task? Why?

   (c) You are working on the "info-tainment" system for the car, and it has a movie player for the backseat. The specification calls for the video decoder to be able to maintain a framerate of 60Hz. Is this a hard, firm, or soft real-time task? Why?

   (d) What is one disadvantage of SPI compared to i2c?

   (e) If you wanted to add a second temperature probe to this device, but at the end of a 50-foot long cable, would you still use a TMP36 sensor? Why or why not?

6. **Linux Fun** (1 point)

   Linux shells have what's known as job control. This isn't a question, but you can try it out and see how it works. To suspend something, press Control-Z. So if you are editing code in nano, try pressing Control-Z. It should bring you back to the prompt. To get back to pico again type `fg`. You can use this to run `make` while editing and going back without losing your place. You can also use `bg` to put programs running in the background, but be careful (you don't want that to happen to nano for example).

   Linux has very useful devices under /dev besides the i2c and spi nodes.

   (a) The first is `/dev/null`. What happens if you pipe a command to it? For example `ls > /dev/null`?

   (b) The next is `/dev/full`. What happens if you pipe a command to it? For example `ls > /dev/full`?

   (c) `/dev/zero` contains nothing but zeros. Why might that be useful?

   (d) What do you think `/dev/random` contains?

7. **Submitting your work**

- Run `make submit` which will create a `hw7_submit.tar.gz` file containing `Makefile`, `README`, `test_temp.c`, `temp_cool.c` and `display_temp.c`.
  You can verify the contents with `tar -tzvf hw7_submit.tar.gz`
- e-mail the `hw7_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!