ECE471: Embedded Systems – Homework 6

Direct (bit-bang) i2c Access

Due: Tuesday, 28 October 2014, 5PM EDT

1. Use your Raspberry Pi for this homework. In addition you will need one 4x7 segment LED displays you used in Homework 5.

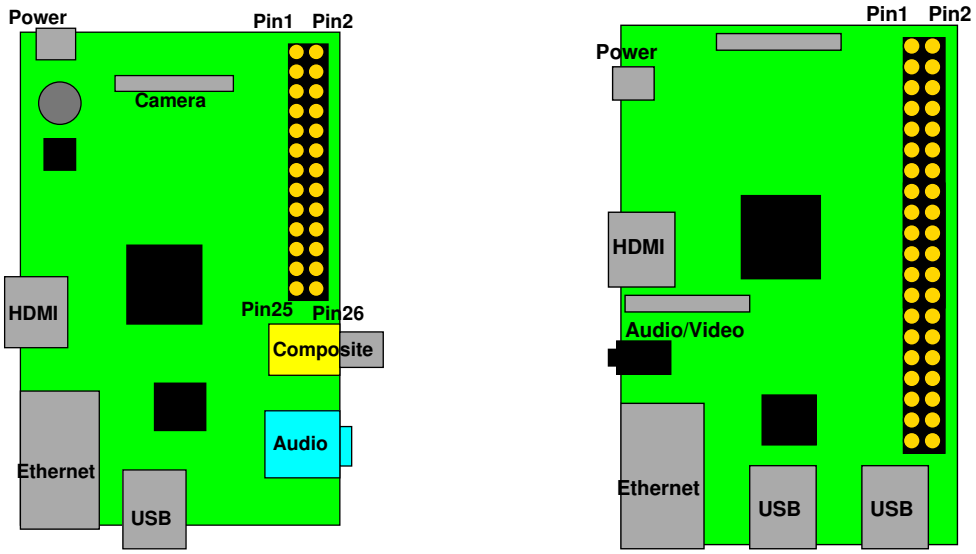

Figure 1: Location of header on Raspberry Pi Model B and B+

2. You will need to hook up the display just as you did in Homework 5. You can use Figure 1 and Table 1 for guidance.

   (a) Hook the display to your pi on a breadboard:
   Connect `3.3V` on the Pi to + on the LED Display.
   Connect `GND` on the Pi to − on the LED Display.
   Connect `SDA` on the Pi to `D` on the LED Display.
   Connect `SCL` on the Pi to `C` on the LED Display.

3. **Getting the Code**

   The display is run by a ht16k33 chip. You can get the datasheet here:
   `http://www.adafruit.com/datasheets/ht16K33v110.pdf`

   Download the template code from the ECE471 website:
   `http://web.eece.maine.edu/~vweaver/classes/ece471_2014f/ece471_hw6_code.tar.gz`

   Uncompress it with `tar -xzvf ece471_hw6_code.tar.gz`

4. **Enable bit-bang i2c support** (6 points)

   Modify the provided `i2c-bitbang.c` file. Running `make` should build your code.

   Comment your code!

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

The provided code does various things for you. It provides code that enables the GPIOs, opens file descriptors to the GPIO value files, etc.

You will need to fill in various functions in the file before it will work.

(a) `clear_SDA()`
This routine should change the GPIO write direction to output and then write a 0 to the sda_fd filedescriptor. You can use the provided `gpio_set_write()` routine to change the write direction: `gpio_set_write(SDA_GPIO);`

(b) `clear_SCL()`
This routine should change the GPIO write direction to output and then write a 0 to the scl_fd filedescriptor. You can use the provided `gpio_set_write()` routine to change the write direction: `gpio_set_write(SCL_GPIO);`

(c) `read_SDA()`
This routine should change the GPIO direction to input (You can use the provided `gpio_set_read(SDA_GPIO);`). Be sure to rewind to the sda_fd with `lseek(sda_fd,0,SEEK_SET);` before reading. Read a byte and return 0 or 1.
Remember to convert from ASCII to decimal!

(d) `read_SCL()`
This routine should be like `read_SDA()` but for SCL instead. Be careful when cut and pasting that you convert all the sda to scl.

(e) `i2c_start_bit()`
This routine should send an i2c start bit. If you recall, this means sda goes from high to low

while scl is high. You might also want to run I2C_delay() to make sure scl stays high for long enough.

NOTE! Remember that i2c is an open collector bus where the bus is pulled high with resistors (for this homework we are using the special GPIOs that already have resistors in place on the motherboard). This means that to set sda and scl high you do *not* write 1 to them, but rather you let the outputs float. The simplest way to do this is to issue a read request (such as read_SDA() or read_SCL()) and ignore the read value.

(f) `i2c_stop_bit()`
This routine should send an i2c stop bit. This is much like the previous start bit routine, only sda goes from low to high while scl is high.

(g) `i2c_read_bit()`
Let sda go high and wait a delay. Let scl go high. Now read sda, it will have the value. Delay, then pull scl low.

(h) `i2c_write_bit()`
Pull scl low. Then set sda to the value you want; let it float for high or set it to low for zero.
Delay. Then let scl go high. Delay. Then pull scl low.

Once you have all of the above implemented, running (as root or with sudo) `./i2c-bitbang` should light up the display, alternating all on and all off.

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

Copy your `i2c-bitbang.c` code over to `i2c-cool.c` and modify it to do something extra. It can just be the same blinking ECE 471 or whatever you did for HW5, or you can try out something different this time.

Alternately, hook up the i2c bus to a logic analyzer (your analog discovery board?) and send a plot of one i2c transaction along with your homework submission.

Put into the README some notes about what your code does.

6. **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 1st. 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) Name one aspect of an embedded system that might contribute to missing a deadline.

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

   (a) You can use the `cat` command to dump a text file to the screen.
   Run `cat /proc/interrupts` which will give you status on the interrupts that have happened on your Pi since it was turned on.

   List a name of one of the interrupt sources.

   (b) There is a command called `yes`. Run it and see what it does. Why do you think a utility like this exists?

8. **Submitting your work**

- Run `make submit` which will create a `hw6_submit.tar.gz` file containing `Makefile`, `README`, `i2c-bitbang.c`, and `i2c-cool.c`
  You can verify the contents with `tar -tzvf hw6_submit.tar.gz`
- e-mail the `hw6_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!