ECE471: Embedded Systems – Homework 3
i2c interface and LED Display

Due: Thursday, 29 October 2013, 5PM EDT

1. Use your Gumstix board for this homework. You will need to have a 40-pin header attached to your Tobi board. If you haven't already, please drop it off for soldering. When you get your board back you will also be loaned a kit providing:
   - One 4x7 segment LED display `http://www.adafruit.com/products/880`
   - One Bi-Directional Level Shifter `http://www.adafruit.com/products/757`
   - One Small Breadboard
   - 12 wires

2. First you will need to hook up the LED display, via the level-shifter, to the i2c bus on the gumstix board. The level shifter is needed because the display works best at 5V (or even 3.3V) while the Tobi i2c interface is running at 1.8V.

   The 40-pin Tobi header pinout is shown in Figure 1. **Careful! Most pin references found online expect you to be looking from the top, but our connector is on the bottom so the pinout is mirrored**.

   (a) First hook up the level shifter:
   Connect both `GND` pins on the shifter to one of the `GND` pins on the Tobi.
   Connect `HV` (high voltage) on the shifter to `V_BATT` (5V) on the Tobi.
   Connect `LV` (low voltage) on the shifter to `VCC_1.8` on the Tobi.
   Connect `A4` on the shifter to `SDA3` on the Tobi.
   Connect `A3` on the shifter to `SCL3` on the Tobi.

   (b) Next hook the display to the level shifter:
   Connect `GND` to − on the LED Display.
   Connect `V_BATT` to + on the LED Display.
   Connect `B4` on the shifter to `D` on the LED Display.
   Connect `B3` on the shifter to `C` on the LED Display.

   To test that everything is working, as root (or using sudo) run the command:

   `i2cdetect -y -r 3`

   and it should give the following output:

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

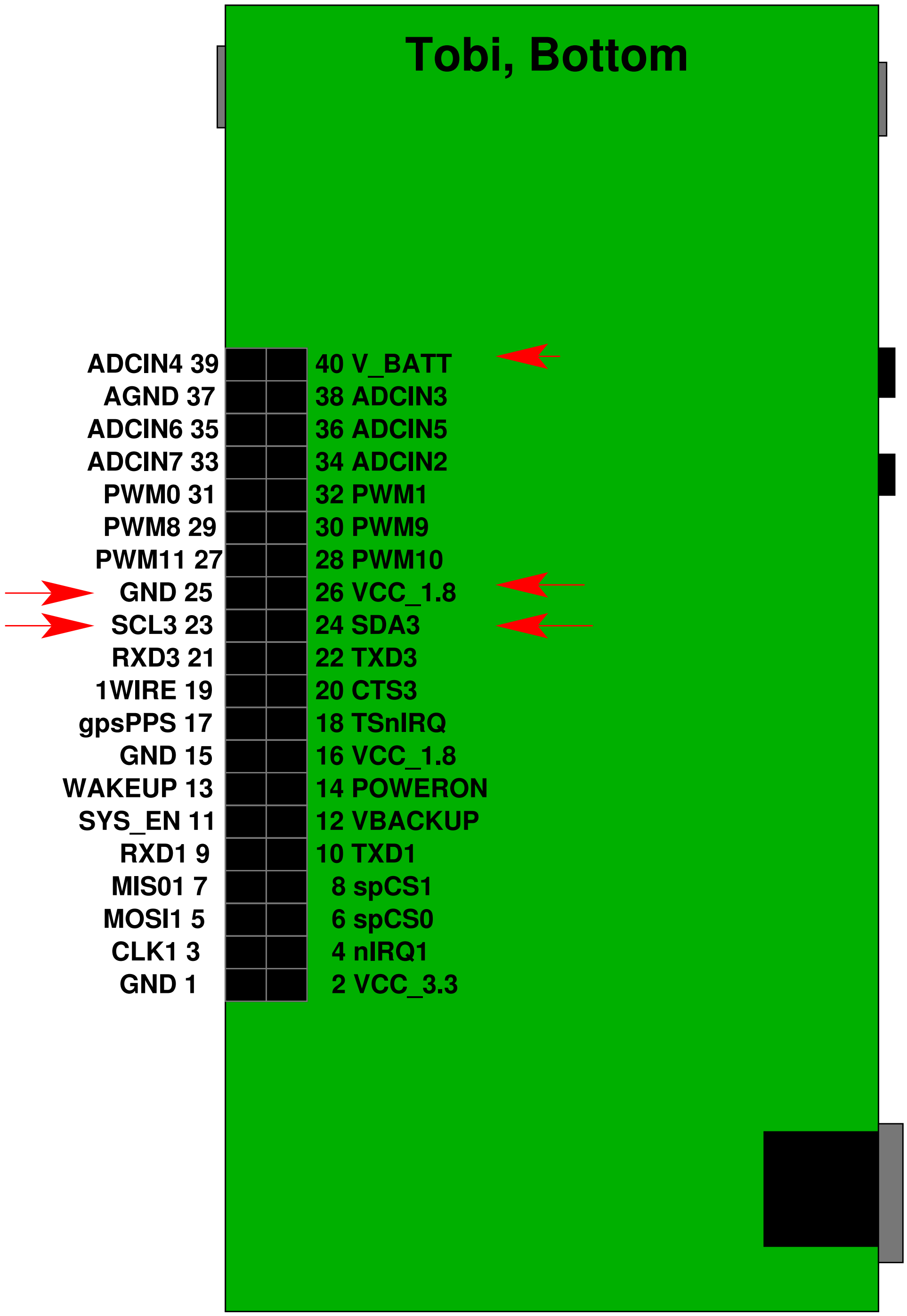


Figure 1: 40-pin header output on Tobi board

The important part is that the 70 appears under column 0. This means Linux can see your slave device at address 0x70.

3. **Talking to the Display via Linux**

   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_2013f/ece471_hw3_code.tar.gz`

   Uncompress it with `tar -xzvf ece471_hw3_code.tar.gz`

   Modify the provided `display_test.c` file. Running `make` should build your code. If you get clock skew errors, either set your clock to current time or else run `make clean` before running `make`.

   Comment your code!

   Make sure you check any function calls for errors and report them back to the user.

   Here are the steps needed to talk to the device:

   (a) Open the device using the `open()` function. This returns an integer file descriptor for the device, or -1 on error. The call will look something like

   `fd = open("/dev/i2c-3", O_RDWR);`

   where `O_RDWR` means open for reading and writing.

   (b) Set the slave address. An IOCTL is used for this.

   `result=ioctl(fd, I2C_SLAVE, 0x70);`

   where I2C_SLAVE is defined in the linux/i2c-dev.h header, fd is the file descriptor from earlier, and 0x70 is the slave address. A negative value is returned on error.

   (c) Commands to the device are described starting on page 10 of the data sheet. Commands are an 8-bit value, with the command type in the top 4 bits and the data in the low 4 bits.

   (d) Send a command to activate the oscillator on the device by modifying the "System Setup Register". The high 4 command bits should be 0x2 and the low 4 bits should be 0x1. Write this 8-bit value to the device. The code will look something like this:

   ```
   unsigned char buffer[17];

   buffer[0]=(0x2<<4)|(0x1);
   result=write(fd, buffer, 1);
   ```

   This says to write to file descriptor `fd` 1 byte from a pointer pointing to the beginning of the buffer array. The return value is how many bytes were successfully written.

   Feel free to use C pre-processor defines to make the constants for the commands and data easier to read.

   (e) Next turn on the display, with blinking disabled. Do this via the "Display Setup Register"

   (f) Next set the brightness. A value from 10 to 15 is probably best. Do this via the "Display Dimming Data Input". You will need to get the proper values from the data sheet.

(g) Finally write out what to display. For this simple test case we will write all 1s to make the display completely light up.

To do this, write the "Display Data Address Pointer" which for our case is 0, then followed by 16-bytes holding the two 8-bit values for each of the 8 rows. It is easiest to just write all 17 bytes at once.

```
unsigned char buffer[17];
int i;

buffer[0]=0x0;
for(i=0;i<16;i++) buffer[1+i]=0xff;
write(fd,buffer,17);
```

(h) Now close the file descriptor with `close(fd);` and exit the program. The display will keep displaying the last thing written to it.

4. **More complicated Display Routines**

For this part of the homework, modify `display_final.c`. First copy your `display_test.c` file over as a starting point:

`cp display_test.c display_final.c`.

Then work on the `display_final.c` code.

The goal is to make the display show the following pattern:

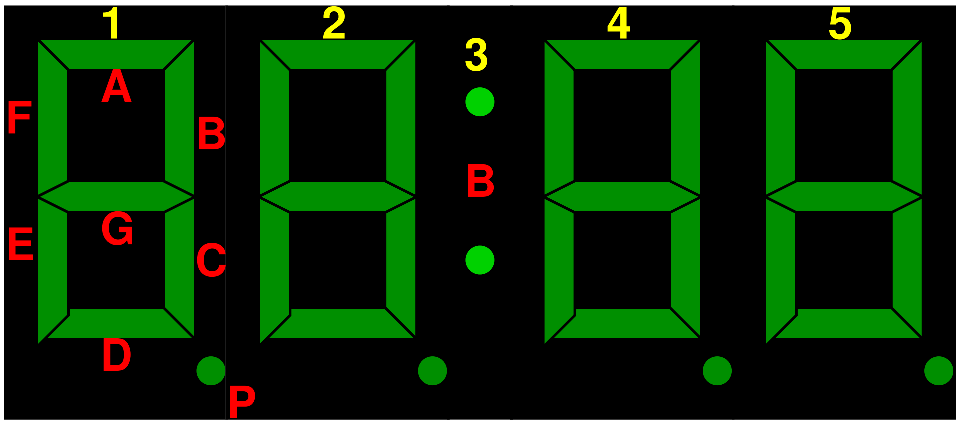
- `ECE`
  pause for 0.5 seconds
- `471`
  pause for 0.5 seconds
- Then display something of your choice (not blank) for 10s. Feel free to be clever, but no bonus points for extra cleverness.
- Then loop back to the beginning to display ECE again, and repeat until broken out of with Control-C.

The display mapping for each LED segment is shown in Figure 2.

As an example, to display the letter 'E' in the far left column, you would do:

```
unsigned char buffer[17];

buffer[0]=0x00;  // offset pointer
buffer[1]=0x79;  // Column 1, Segments ADEFG
buffer[2]=0x00;  //  next 8 bits of column 1, not connected
buffer[3]=0x00;  // Column 2, empty
buffer[4]=0x00;  //  next 8 bits of column 2, not connected
...
write(fd,buffer,17);
```



byte 0 = 0x00 (display pointer offset)
byte 1 = (1P, 1G ,1F, 1E, 1D, 1C, 1B, 1A)
byte 2 = 0x00
byte 3 = (2P, 2G, 2F, 2E, 2D, 2C, 2B, 2A)
byte 4 = 0x00
byte 5 = (X, X, X, X, X, X, X, 3:, X)
byte 6 = 0x00
byte 7 = (4P, 4G, 4F, 4E, 4D, 4C, 4B, 4A)
byte 8 = 0x00
byte 9 = (5P, 5G, 5F, 5E, 5D, 5C, 5B, 5A)
byte10–byte16 = 0x00

Figure 2: LED Display Segment Mapping

The Linux/C function to pause (sleep) for a period of time is `usleep()`. The parameter is the amount of time to sleep in micro-seconds.

5. **Editing the README**

   Edit the `README` file to have your name, and then a brief description of what your `display_final.c` code is displaying.

   (For example, "ECE, pause for .5s, 471, pause for .5s, my cell-phone number scrolling back and forth for 5s").

6. **Submitting your work**

   - Run `make submit` which will create a `hw3_submit.tar.gz` file containing `README`, `display_test.c` and `display_final.c`.
     You can verify the contents with `tar -tzvf hw3_submit.tar.gz`
   - e-mail the `hw3_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!