ECE471: Embedded Systems – Homework 4

A/D Converter and System Busses

Due: Thursday, 14 November 2013, 5PM EST

Background:

- Use your Gumstix board for this homework. You will need 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 one up.
- First check that the TPS65950 Power Management chip (which provides A/D inputs) on your Gumstix is working properly.

  As described in class, the interface for this chip is handled by the "hardware monitoring" hwmon driver through files under the `/sys/class/hwmon` directory.

  - Run:

    ```
    cat /sys/class/hwmon/hwmon0/device/in12_input
    ```

    and it should report the value of the 3.3 Voltage rail (in millivolts).
  - Run:

    ```
    cat /sys/class/hwmon/hwmon0/device/temp1_input
    ```

    and it should show you the temperature of the CPU in degrees Celsius.
  - Run:

    ```
    cat /sys/class/hwmon/hwmon0/device/in2_input
    ```

    which is an unconnected input. It will likely be floating somewhere around 100mV or so.
- Hook up the TMP36 to the Gumstix A/D input #2.

  The datasheet for the TMP36 can be found here: `http://www.analog.com/en/mems-sensors/digital-temperature-sensors/tmp36/products/product.html`

  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 diagrams in Figures 1 and 2 and you will be OK.

  Connect pin1 (3.3V) of the TMP36 to pin 2 (3.3V) on the Tobi
  Connect pin2 (Vout) of the TMP36 to pin 34 (ADCIN2) on the Tobi
  Connect pin3 (GND) of the TMP36 to pin 37 (AGND) on the Tobi

  Test your setup by running

  ```
  cat /sys/class/hwmon/hwmon0/device/in2_input
  ```

  which should now report a voltage proportional to the temperature.

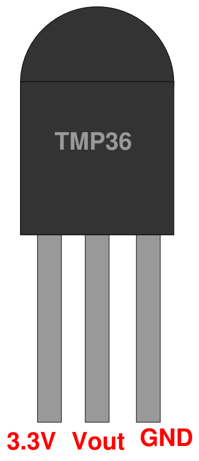

Figure 1: TMP36 Pinout

1. **Part 1: Reading the Temperature with Linux**

   Download the template code from the ECE471 website:
   `http://web.eece.maine.edu/~vweaver/classes/ece471_2013f/ece471_hw4_code.tar.gz`

   Uncompress it with `tar -xzvf ece471_hw4_code.tar.gz`

   Modify the provided `test_temp.c` file so that it continually reads in the temperature from the sensor and prints it to the console using `printf` in an infinite loop (you can press control-C to break out of the program). The output should be the temperature in Fahrenheit and should print one decimal after the decimal point.

   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`.

   Be sure to Comment your code!

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

   You can use the high-level `fopen()` to open the file and `fscanf()` to read the value from the `/sys/class/hwmon/hwmon0/device/in2_input` file.

   You can use the `sleep()` or `usleep()` functions to delay between updates.

   The temperature can be determined with the following equation:
   $deg\_C = (100 \times \frac{voltage}{1000}) - 50$

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

2. **Part 2: Writing to the Display**

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

   `cp test_temp.c display_temp.c`.

   Then work on the `display_temp.c` code.

   The goal is to make the display show the temperature in Fahrenheit, updated once per second. You may re-use code from HW#3 for updating the display.

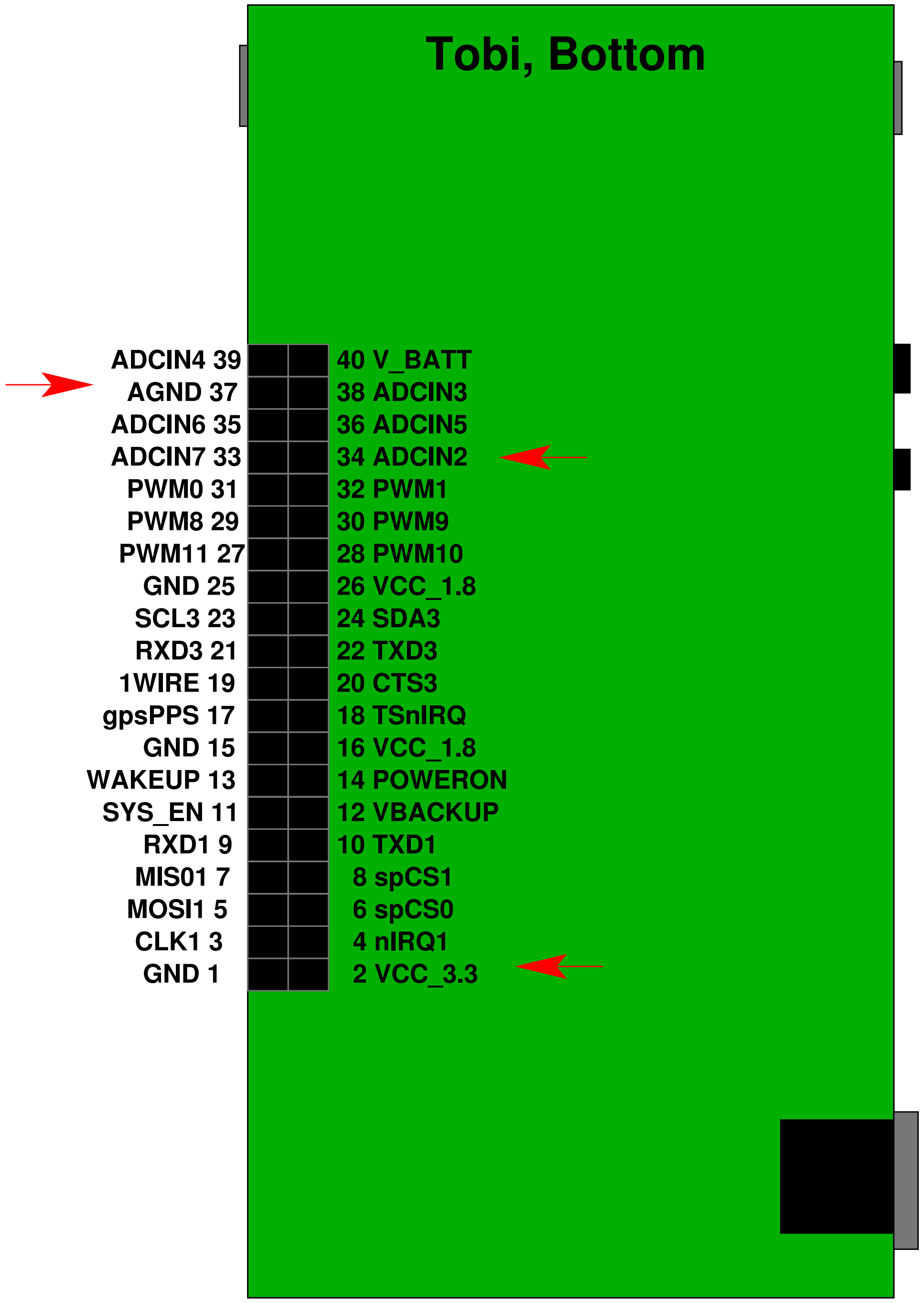

Figure 2: Tobi board 40-pin header pinout

Your code should handle four cases:

(a) Temperatures between 0 and 99.9 degrees. These should be displayed as two digits, a decimal point, another digit, and then a degree symbol (which is just a crude circle made of the top 4 segments on the display).

(b) Temperatures between -99 and -1 degrees. These should display a minus sign and then two digits of temperature, then the degree symbol.

(c) Temperatures between 100 and 999 degrees should print three digits of temperature, then the degree symbol.

(d) Invalid temperatures that won't fit the display (and errors reading the thermometer) should be reported in a method that isn't a valid temperature. It is your choice how to indicate this.

To test the above you can first write the display code (maybe as a separate function) and hard-code the value to display. Then once it works on all of the possibilities, then hook it up to your temperature reading code.

3. **Part 3: Editing the README**

   Edit the `README` file to have your name and answer the following questions.

   (a) Describe what your code in Part 2 does if the temperature is out of range.

   (b) 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?

   (c) As per the previous question, you want to add a temperature sensor at the end of a 50-foot cable. Of the busses described in class, name one which would be well suited for this task. List 3 features of the bus you pick and why they suit this application well.

4. **Submitting your work**

   - Run `make submit` which will create a `hw4_submit.tar.gz` file containing `Makefile`, `README`, `test_temp.c` and `display_temp.c`.
     You can verify the contents with `tar -tzvf hw4_submit.tar.gz`
   - e-mail the `hw4_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!