Section 1: General Lab Info /0.5 points

Microprocessors II and Embedded Systems

EECE 4800

Lab 2: Interfacing with a Sensor Device on an Embedded Computer System

Prof. Yan Luo, TA. Ioannis Smanis

Group #1

Andrew MacGregor

Hand in: November 10, 2017

Due: November 10, 2017

Section 2: Contributions /1 points

1. **Group Member 1 - Andrew MacGregor (me)**

Wrote opencv code.
Figured out video interface.
Helped debug everything

2. Group Member 2 - Grayson Colwell

Wrote main file (including date-time code)
Helped figure out video interface
Helped figure out i2c bus.
Helped debug everything.

3. Group Member 3 - Jose Velis

Wrote all of the temperature sensor code.
Figured out i2c bus logic.
Helped debug everything.

Section 3: Purpose /0.5 points

The purpose of this project was to understand how to interface with peripherals through embedded Linux. i2c bus use was demonstrated with the temperature sensor, and the video device interface was demonstrated with the webcam. In the process of using these busses, we scratched the surface of Opencv and how to install and use libraries in general.

Section 4: Introduction /0.5 points

The final deliverable of this project is an application on the Galileo that uses the temperature sensor to control a webcam. The temperature sensor is connected to the Galileo on an i2c bus and the webcam is connected to the Galileo’s only USB port. If the temperature sensor reads a temperature above a certain threshold, the webcam takes a picture.

Section 5: Materials, Devices and Instruments /0.5 points

- Intel Galileo with wifi adaptor and with OpenCv and libmraa installed
- FTID/serial cable
- Breadboard
- Temperature sensor (TMP 102)
- Four wires to power the temperature sensor and connect it to the i2c bus
- USB 2.0 webcam

We did not end up using an oscilloscope or logic analyzer to debug our bus in this lab.

Section 6: Schematics /0.5 points

This project had the simplest circuit schematic so far. Figure 1 shows the connections made between the temperature sensor and the Galileo board. The webcam was connected the Galileo's only USB port, and the wall power supply was connected to the Galileo's power port.

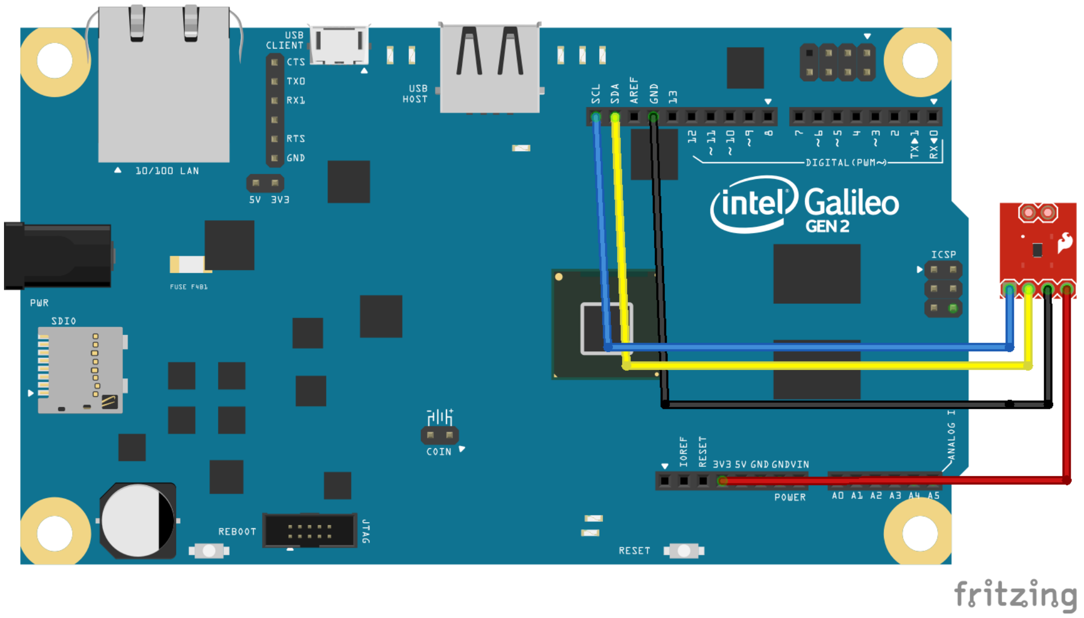

Figure 1: *Circuit drawing of our temperature sensor connection.*

Section 7: Lab Methods and Procedure /2 points

Hardware design:

Hardware design was minimal for this lab. The only design that went on was connecting the right ports on the Galileo to the temperature sensor Vcc, Gnd, i2c-SDA and i2c-SCL lines. Pull-up resistors for the i2c bus were already integrated into both the Galileo's i2c hardware and the temperature sensor. The webcam connection was trivial because it was just connecting the webcam's usb cable to the only USB port on the Galileo.

Software design:

I^{2}C

We chose to use libmraa to control the i2c bus. Mraa is Intel's Linux library for low speed IO communication. With the mraa c++ bindings, i2c usage is trivial. We created a new i2c object on i2c-0 and set the address to 0x48. 0x48 was the address that we found the temperature sensor to have when using i2cdetect as shown in Appendix A1.

Temperature sensor

To control the temperature sensor, all we did was read two bytes from the default register on the sensor. We address the sensor at its i2c address (0x48). The two bytes we read contain the left justified 12 bit temperature value. To convert to degrees C, we treat it like a signed 16 bit integer, then shift it right 4 bits (to remove the left justification) then multiply by 0.0625 (because each bit in the temperature value corresponds to 0.0625 degrees C. Code snippet is shown in Appendix A2.

Opencv/Webcam

We used opencv to access the webcam. Every time a new image needed to be captured, we used a cv::VideoCapture object connected to the video0 interface to access the webcam. We used the *read* function to read a single frame from the camera into a Mat object. Next, we saved the file as a .jpeg with a filename related to the current date and time. Code snippet is shown in Appendix A1.

Date

We used standard C functions from string.h and time.h to format a filename based on the current date and time. We used the time() function to retrieve the current date and time, then used asctime() and localtime() to format the time struct into a string. These functions return strings with inconvenient characters for filenames, so we replace them with underscore characters.

Main logic

In the main loop, there was one function used to capture & save an image, and there was one function used to read the temperature. Our main loop is based on commands input from a user keyboard. If cmd=1 is input, the temperature is read, and if the temperature is above a certain threshold (30 degrees Celsius in our case), a picture is taken and saved with the webcam. If cmd=2 is input, a picture is taken and saved without checking the temperature. Flow chart is shown below in Figure 2.

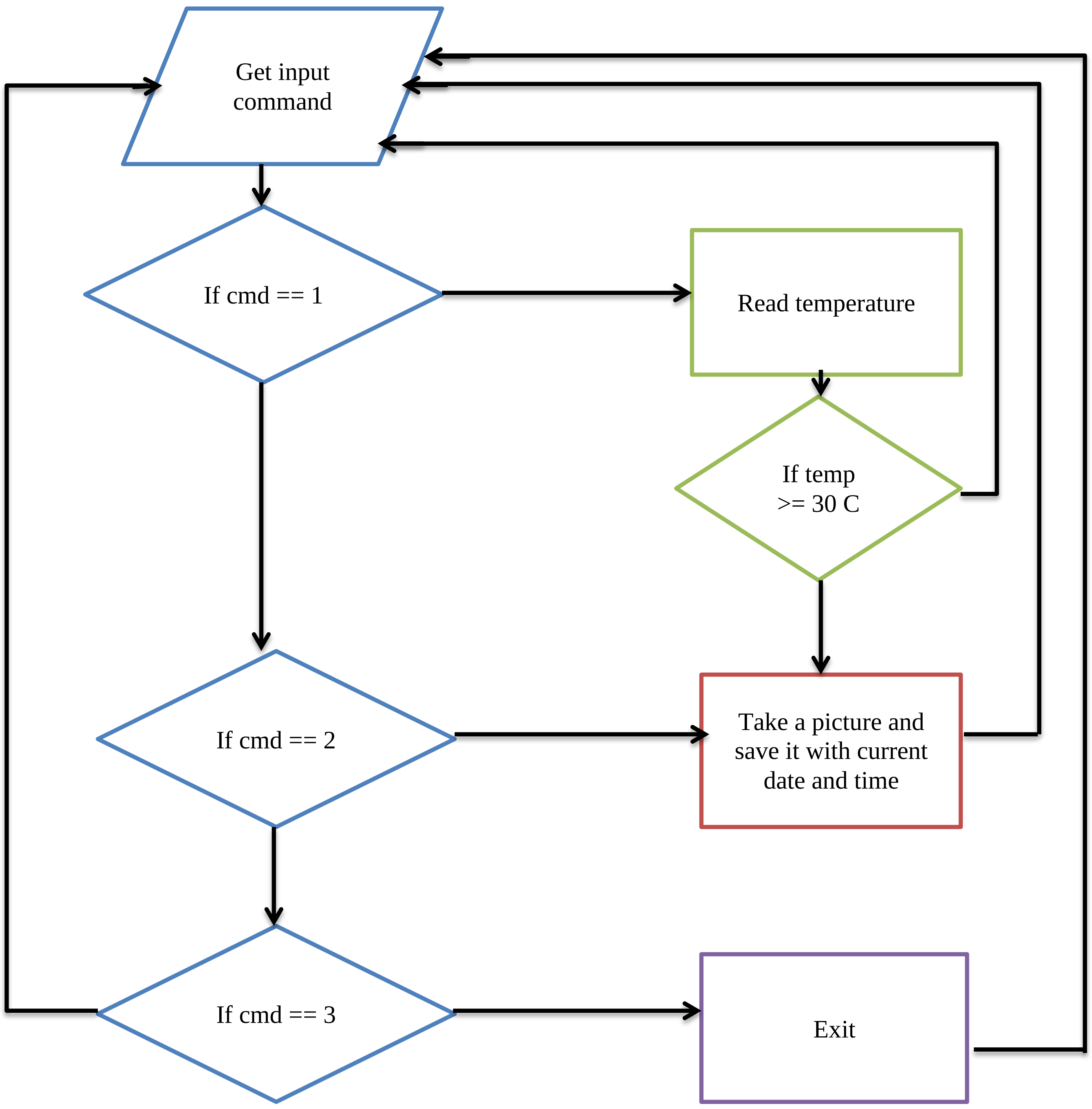

***Figure 2**: Software flow chart for main loop*

Section 8: Trouble Shooting /1 *points*

Issue 1: We were having trouble compiling code that included openCV headers.

We solved this issue by editing /usr/lib/pkgconfig/opencv.pc like Ioannis instructed on piazza.

Issue 2: We were having trouble viewing the first images that the openCV module.

We solved the issue by trying a different set of compression parameters to save the picture as a jpeg instead of a png. That let us look at the files.

Issue 3: We couldn't transfer files to Windows saved with a filename using the unedited string return of asctime().

We debugged the code and found that this function was returning a string with ':' and '\n' characters. We replaced these symbols with '_' characters in the filenames and had no problem saving it then.

Issue 4: The temperature sensor returned values that were way too high.

We found out that the units of the value returned by the temperature sensor were equal to 0.0625 °C. To convert to °C, we multiplied the value returned by 0.0625.

Section 9: Results /0.5 points

To discover the temp sensor on the i2c-0 bus, we used the command line too i2cdetect. Output of this function is shown in Appendix A1.

We tested our temp sensor and webcam incrementally. We started off by trying to make minimal simple reads using both sensors. After both sensors worked making simple measurements, we found a reasonably temperature threshold to control the camera. It turns out that Ball Hall 424 was about 27°C normally, and when the sensor is covered by a warm finger, the temperature rises to about 31°C. We chose to use 30°C as our threshold.

As a result, we ended up taking many pictures of ourselves with our finger on the temperature sensor. Here is a typical picture taken.

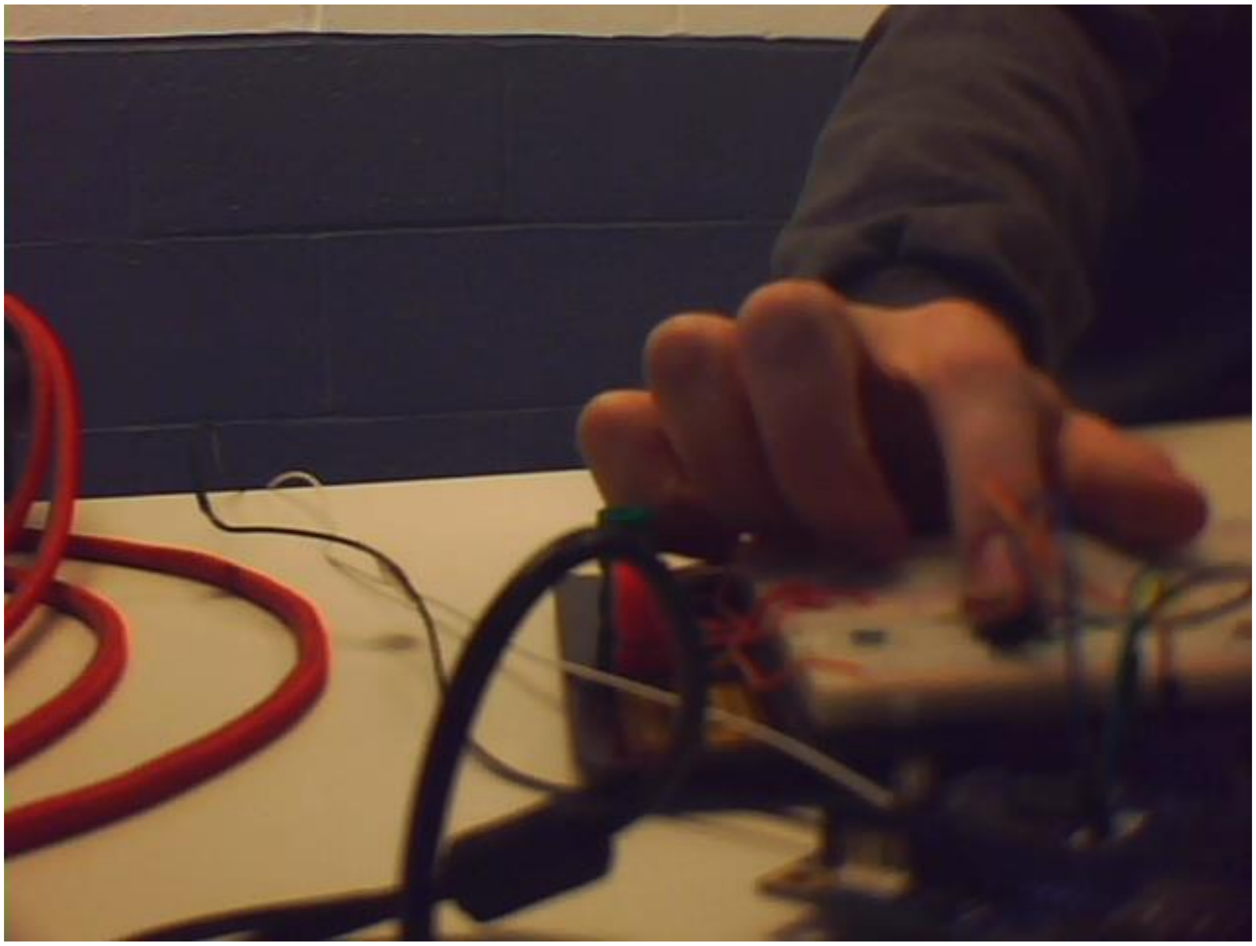

Figure 3: *Webcam-captured image of one of us with our finger on the temperature sensor.*

A1. i2cdetect output with temperature sensor connected

```
root@galileo:~/Lab3# i2cdetect -y -r 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- UU UU UU -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- UU 48 -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
root@galileo:~/Lab3#
```

A2. Function to get temperature

```
double get_temp(){
      I2c i2c(0);
      i2c.address(TMP102Address);

uint8_t dataReg [2];

int buffer = i2c.read(dataReg,2); // read two bytes from the registers

int temperature = ((dataReg[0]<<8 | dataReg[1]) >> 4);

return temperature*0.0625;
}
```

A3. Function to capture/save image (without error checking)

```
bool capture_and_save_image(char* filename)
{
  //init webcam on video0 interface
  VideoCapture ourCam;
  ourCam.open(0);
  Mat image;

 //read a frame from the vid camera into image
  ourCam.read(image);
  //make jpeg format parameters
  vector<int> compression_params;
  compression_params.push_back(IMWRITE_JPEG_QUALITY);
  compression_params.push_back(95);

  //make filename
  string fn = string(filename) + string(".jpeg");

  //try to save to file
  imwrite(fn, image, compression_params);
  return true;
}
```