Section 1: General Lab Info */0.5 points*

"Microprocessor Systems II & Embedded Systems"

"EECE.4800"

"Lab3: Controlling an i2c Device"

"Instructors Yan Luo, TA; Ionnis Smanis"

"Group #10"

"Derek Teixeira"

"Due date November 20, 2017"

"Handed in November 20, 2017"

Section 2: Contributions /1 points

1. Group Member 1 – Derek Teixeira (Me)

Had the same role as last time, being the team manager and handling all of the equipment and being at all of the team meetings. Main responsibility was getting the Temperature sensor to communicate with the Galileo board. Eventually went to figuring out of how to Write to and read from the Texas Instruments TMP 102 device. Able to discover OpenCV libraries and help get the webcam part of the project started. Did the physical parts of wiring the circuitry on the Galileo board with both the USB camera and the 102 TMP gauge.

2. Group Member 2 – Hans-Edward Hoene

Hans wrote most of the code for the web cam capture and the ability to send and receive files from the computer to the Galileo board. Was the first person to get the camera to work and realized it wasn't taking correct pictures. On presentation day, assembled all the code into nice separate files with C files and header files.

3. Group Member 3 – Kyle Marescalchi

Kyle was in charge of testing the code for the web cam device. Kyle was able to fix the portion of having negative temperature readings from having an OR gate instead of a + sign. Kyle suggested using shifts in order to read from the TMP 102 device when operating over i2c.

Section 3: Purpose /0.5 points

Lab 3 introduces the class to new abilities that the Galileo board has built into it. These consist of having the ability to control an i2c device and the ability to control devices running on USB. The Temperature sensor made by Texas Instruments, is the i2c device used in this lab for undergraduate students. The Galileo has built in i2c support and even has the ability to not need pullup resistors when using an i2c device. The other device is a simple USB webcam that plugs into the USB 2.0 port on the side of the Galileo board. Both of these devices can be control and manipulated by the little but very power piece of equipment called Intel Galileo gen2. This lab shows other abilities that the simple Galileo board can execute.

Section 4: Introduction /0.5 points

The premise of lab3 “Controlling an i2c Device” is to learn the basics of i2c and have that device give live readings to the user on the Galileo board’s Linux terminal. Once the groups can figure out how to send data back and forth from this i2c device, a web cam is then to be used to take a picture on temperature change. The camera should take a picture when the temperature on the TMP 102 goes over a user set threshold. Learning the essentials to i2c reading and writing, and also the basics of OpenCV; a C library for computer vision, were very important for completing this lab3.

Section 5: Materials, Devices and Instruments /0.5 points

- Analog Discovery 2, Model# 210-321, S/N# 210321AZE323, Operating Voltage (5 volts) Was sometimes used to check voltages, square wave pulses, and supply voltages.
- Breadboard R.S.R Electronics
- Intel Galileo Gen 2 Board, S/N# FZGL50400CW, operating voltage is 3.3 or 5 volts. Used to write and read temperate from the TMP 102 and to capture pictures from a webcam.
- USB-to-UART cable, has 6 pins and is used for serial communication between the Galileo and a PC USB port. FTDI drivers needed for communication.
- Two standard length cat-5 ethernet cables.
- Temperature Sensor- TMP 102 from Texas Instruments. Operates around 3.3 volts and has 12-bit resolution for results.
- Webcam that uses USB 2.0

Section 6: Schematics */0.5 points*

Basic configuration for the Lab3 circuitry. As shown the SDA from the TMP 102 goes into the A4/SDA port of the Galileo and the SCL goes into the A5/SCL port of the Galileo board. The USB 2.0 port on the Galileo board is used to power and send signals to the webcam. The ethernet port is used to send and received files between the Galileo board and the PC controlling the Linux terminal. The TMP 102 cannot be connected to over 3.6 volts or it will get burnt out, therefore it is connected to the 3.3 V port from the Galileo board. Both the ground pin and the AD0 from the TMP 102 are connected to the ground pin on the Galileo. Having AD0 connected to ground gives our i2c device the set address of hex 48(100 1000).

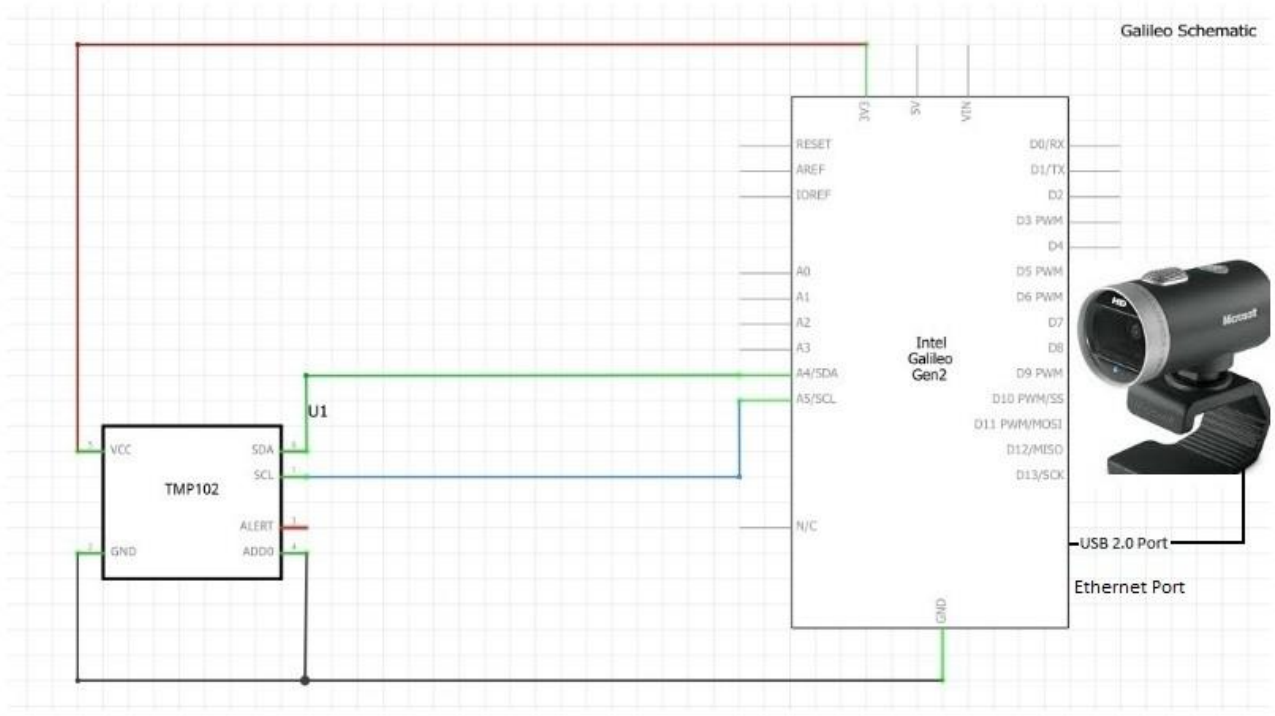

Figure 1: Schematic connecting the Temp Sensor to Galileo board, also showing the webcam input/output

Section 7: Lab Methods and Procedure /2 points

Hardware design:

The Hardware design is all based off of the schematic that is scene on the previous page. It was really basic for this lab number 3 and the main things to account for with the temperature sensor are the SDA, SCL, VCC, Ground and ADO. The SDA goes into A4 and the SCL goes into A5 on the Galileo board. The voltage gets tied to the 3.3 V output on the Galileo while the TMP 102 ground connects to the Galileo ground. When grounding the address pin of the TMP 102, it will make the address 48 in hexadecimal; This is shown in figure 2. The webcam just needed to be plugged into the USB 2.0 on the side of the Galileo, nothing was done with the aux part of the cord. The only thing from the TMP 102 that was not in use was the ALT pin which stands for alert.

Table 12. Address Pin and Slave Addresses

DEVICE TWO-WIRE ADDRESS	A0 PIN CONNECTION
1001000	Ground
1001001	V+
1001010	SDA
1001011	SCL

Figure 2: Assigning the address of the i2c device (TMP 102)

Software design:

First thing to start off with was detecting the i2c device and figuring out if video0 was showing when looking for the webcam connection. The gathered information is shown in the results section for commands to find out if an i2c and a webcam are present. Once this is figured out the group was able to start trying to write and read to the i2c device. If we look at the TMP 102 data sheet, we can see how to write to the i2c device.

Once the data is written to the i2c, we now need to read from the TMP 102. This is done by reading 12-bits of resolution through two 8-bit registers (the same register just read twice). First, the bits are shifted to the left 4 to get to the 12-bits total, then it is shifted to the right 4 to make the last 4 bits able to be read. This can be seen in figure 7. Both the read command the write command code can be seen below the datasheet descriptions of the TMP 102.

The main way our program runs, is to have the user set a threshold with their body heat. The sample of code can be seen in figure 9 and allows ten samples to be taken to set the threshold temperature. This is done in order to get an accurate threshold, no matter the room or surrounding environment.

Setting up the camera was the easy part but learning OpenCV was the difficult part. All the camera needed was to be plugged into the USB 2.0 port and it was read as being video0. We had to use a lot of header files, including some CPP files in order to compile our camera side of the code. We needed 5 different functions from the OpenCV library to capture and store a photo onto the SD card. They can be seen in figure 8.

Last but not least was being able to take a picture when the temperature was over the threshold reading from the user. This can be seen in the main code section in the appendix. The main.c will take a picture and save it when the user threshold is over the set point in the first five seconds of the program. Main.c contains a count system where when the number of pictures is equal or greater than the PIC_COUNTER, the camera finishes and ends the program.

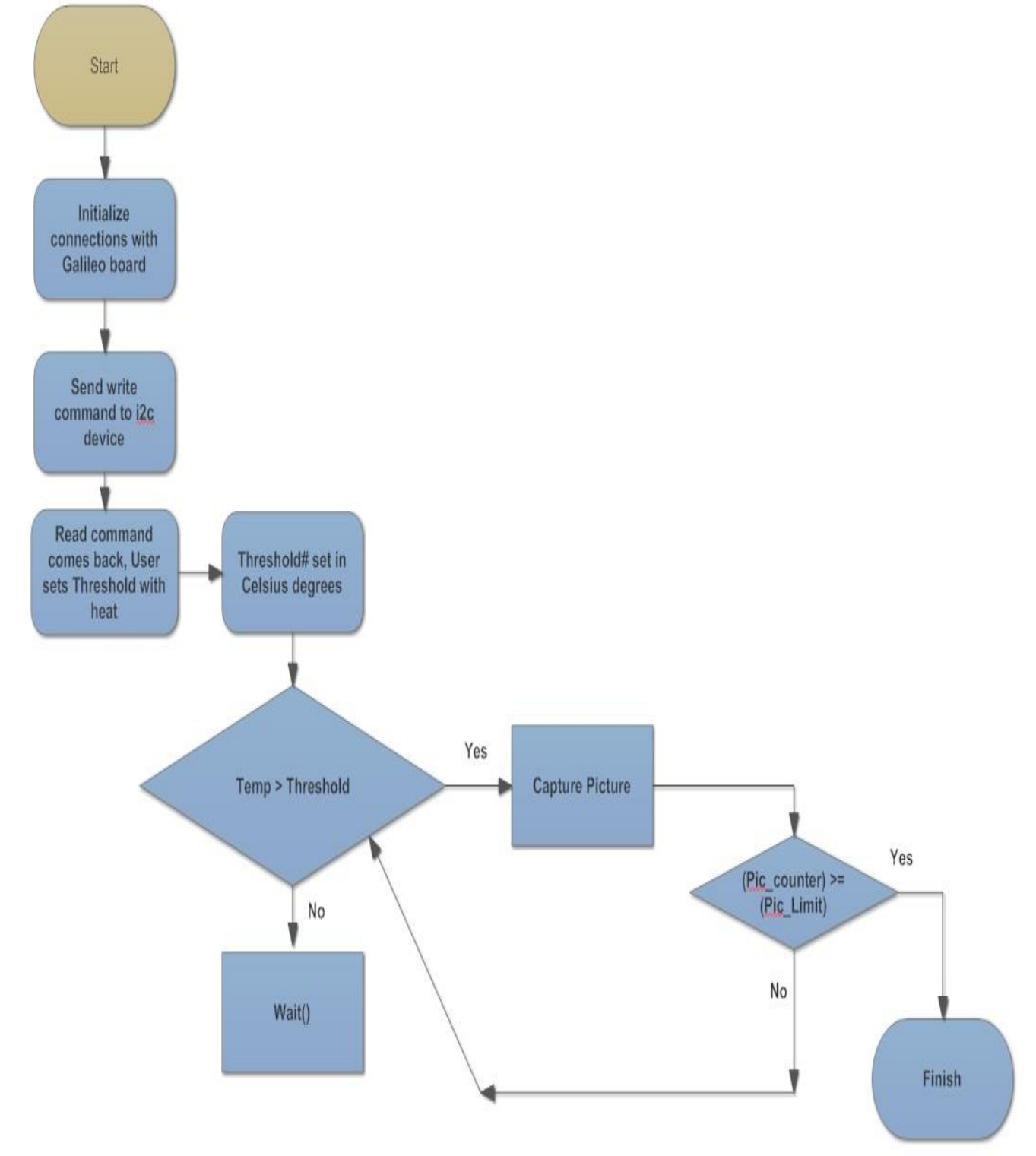

Figure 3; Flowchart implementation of program for lab3

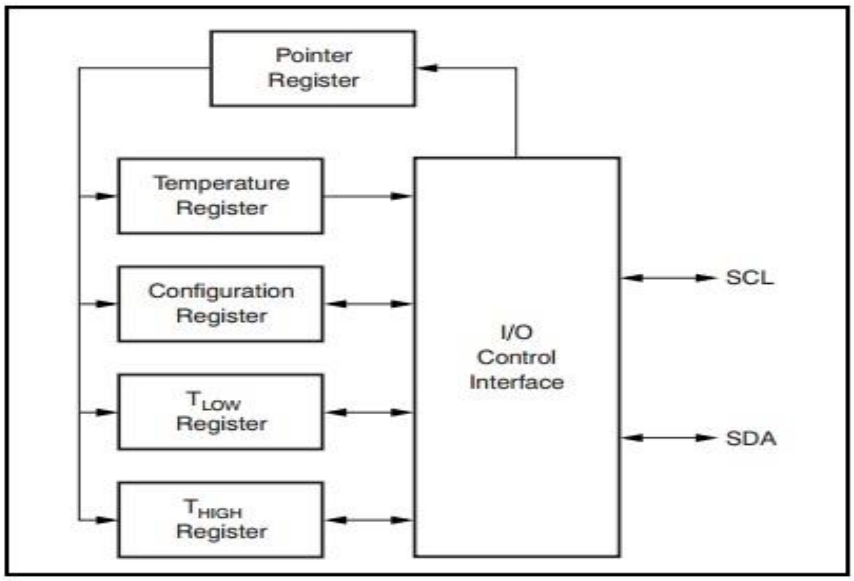

Figure 8. Internal Register Structure

Table 1. Pointer Register Byte

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	Register Bits	

Table 2. Pointer Addresses

P1	P0	REGISTER
0	0	Temperature Register (Read Only)
0	1	Configuration Register (Read/Write)
1	0	T_{LOW} Register (Read/Write)
1	1	T_{HIGH} Register (Read/Write)

Figure 4: Writing to the TMP 102 i2c device, we send all zeroes and write the the TMP 102.

```
int InitTempDevice(int adapter_number) {
	// opens i2c device and returns file handle
	// returns <0 if error

	int handle;		// handle to the temperature sensor; will be returned
	char filename[50];	// to access temperature sensor

	sprintf(filename, "/dev/i2c-%d", adapter_number);
	handle = open(filename, O_RDWR);// gets handle to temperature sensor
	ioctl(handle, I2C_SLAVE, ADDRESS); // set as I2C slave, which is at ADDRESS
	write(handle, 0, 1);// set pointer register byte to zero to read from register

	return handle;		// return the handle for further communicate
}
```

Figure 5: Code section for writing to the i2c device

Table 3. Byte 1 of Temperature Register(1)

D7	D6	D5	D4	D3	D2	D1	D0
T11 (T12)	T10 (T11)	T9 (T10)	T8 (T9)	T7 (T8)	T6 (T7)	T5 (T6)	T4 (T5)

(1) Extended mode 13-bit configuration shown in parenthesis.

Table 4. Byte 2 of Temperature Register(1)

D7	D6	D5	D4	D3	D2	D1	D0
T3 (T4)	T2 (T3)	T1 (T2)	T0 (T1)	0 (T0)	0 (0)	0 (0)	0 (1)

(1) Extended mode 13-bit configuration shown in parenthesis.

Figure 6: Reading the temp, we only need up until D4 on the second register for 12-bit mode

```
double readTemp(int handle) {
	// return temperature in celsius

	unsigned char buffer[2];// bytes that shall be read will go here
	unsigned int temp;

	read(handle, buffer, 2);// read 2 bytes from temperature register
	temp = (buffer[0] << 4) + (buffer[1] >> 4);	// shift bytes for 12-bit value
	return (double)temp * 0.0625;//[the resolution] in Celsius
}
```

Figure 7: Code section for the read command from the i2c

```
void takePicture(unsigned int id) {

	char filename[200];
	CvCapture *capture;
	IplImage *image;

	/*
		1) Establish file name
		2) capture frame
		3) retrieve data from frame
		4) save image to file
		5) release capture
		6) release image
	*/

	sprintf(filename, "%s/%u.jpg", DEST_FOLDER, id);//Folder[DEST_FOLDER]/[id].jpg
	capture = cvCaptureFromCAM(CV_CAP_ANY);	// capture frame
	image = cvQueryFrame(capture);			// grabs and retrieves data
	cvSaveImage(filename, image, 0);	// save image to file as JPG
	cvReleaseCapture(&capture);		// release capture
	cvReleaseImage(&image);				// release image

	return;
}
```

Figure 8: Code to setup camera capture, save, and release

```
double sampleTemp(int handle) {

	unsigned int i;
	double sum;

	sum = 0;
	for (i = 0; i < NUM_SAMPLES; i++) {
		sum += readTemp(handle);
	}

	return sum / NUM_SAMPLES;
}
```

Figure 9: Code for setting the threshold before the pictures can be taken. NUM_SAMPLES = 10

Section 8: Trouble Shooting /1 points

ISSUE #1

The first issue was at the beginning of trying to connect the i2c device and the USB webcam. All of the connections made sense but the group was not able to figure out if it was connected and communicating with the Galileo board. The reason we actually figured out that it was working was when another group asked to borrow our TMP 102 chip. This showed that our address, 48, was not showing up in the grid when using the "i2detect -r 0" command. The same thing happened with the camera when looking for video0. The photo for the i2c detect and LS to find the video0 can be found in the results section.

ISSUE #2

Not the biggest issue but one that was concerning on the day of the demo presentation. When Hans went in to clean up the code to make it easier to present, he took out a + symbol and added an OR gate to the i2c.c code. This didn't seem like an issue, but when testing the program multiple times afterwards, the Celsius temperate was showing up as negative. Kyle was doing the debugging and found out that maybe the OR gate was taking the value as a negative signed integer value. We put back a + sign to read the 12-bits and everything went back to working; crisis everted.

ISSUE #3

Biggest issue we faced was not knowing that our webcam was broken from the start. Its connections and actions were functioning normally but the pictures were coming out discolored or not normal looking. This was confusing at first considering we didn't know if it was hardware or software related. In order to figure out that the camera was bad, Derek took home the galileo program and ran it with a different, better performing webcam. The pictures on the different webcam were coming out fine and we sent an email to the TA explaining our problem. We received a different webcam and were able to capture normal colored photos just by changing the camera. It was essential to test the camera before editing any part of our code.

Section 9: Results	*/0.5 points*

TERMINAL SHOTS FROM DEMO PRESENTATION

```
root@galileo:~/Documents/from PC# make main
gcc -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/usr/local/lib/ -Wall main.c i2c.c pic.c -o ./gal.out
_video -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_stitching
root@galileo:~/Documents/from PC# ./gal.out
Get ready to put hand on the sensor...
Put hand on temperature sensor. Do not remove until instructed to do so.
Now take your hand off the sensor.
Threshold: 25.81 degrees Celsius
Program will begin in 5 seconds...

Your picture is being taken. Temperature (C) = 25.82
Your picture is being taken. Temperature (C) = 25.83
Your picture is being taken. Temperature (C) = 25.84
25.8
```

Figure 10: Terminal screen shots of take pictures on demo day

```
root@galileo:/sys/class# i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     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 -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
root@galileo:/sys/class# i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     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:/sys/class# i2cdetect -l
i2c-0   i2c             intel_qrk_gip_i2c                       I2C adapter
root@galileo:/sys/class# i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     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:/sys/class#
```

Figure 11: Shows when the TMP 102 device was connected or discounted, address 48

```
imrtest0         ptyp7               rtc0       tty28  tty54  ttypc   vcs5
initctl          ptyp8               shm        tty29  tty55  ttypd   vcs6
input            ptyp9               snd        tty3   tty56  ttype   vcsa
kmem             ptypa               spidev1.0  tty30  tty57  ttypf   vcsa1
kmsg             ptypb               stderr     tty31  tty58  ttyq0   vcsa2
log              ptypc               stdin      tty32  tty59  ttyq1   vcsa3
loop-control     ptypd               stdout     tty33  tty6   ttyq2   vcsa4
loop0            ptype               tty        tty34  tty60  ttyq3   vcsa5
loop1            ptypf               tty0       tty35  tty61  ttyq4   vcsa6
mem              ptyq0               tty1       tty36  tty62  ttyq5   video0
mmcblk0          ptyq1               tty10      tty37  tty63  ttyq6   zero
mmcblk0p1        ptyq2               tty11      tty38  tty7   ttyq7
root@galileo:/dev# [  207.591277] usb 1-1: USB disconnect, device number 2

root@galileo:/dev# cd /dev
root@galileo:/dev# ls
autofs           mmcblk0p2           ptyq3      tty12  tty39  tty8    ttyq8
block            mqueue              ptyq4      tty13  tty4   tty9    ttyq9
bus              net                 ptyq5      tty14  tty40  ttyGS0  ttyqa
char             network_latency     ptyq6      tty15  tty41  ttyS0   ttyqb
console          network_throughput  ptyq7      tty16  tty42  ttyS1   ttyqc
core             null                ptyq8      tty17  tty43  ttyp0   ttyqd
cpu              port                ptyq9      tty18  tty44  ttyp1   ttyqe
cpu_dma_latency  ppp                 ptyqa      tty19  tty45  ttyp2   ttyqf
disk             ptmx                ptyqb      tty2   tty46  ttyp3   uio0
esramtest0       pts                 ptyqc      tty20  tty47  ttyp4   uio1
fd               ptyp0               ptyqd      tty21  tty48  ttyp5   urandom
full             ptyp1               ptyqe      tty22  tty49  ttyp6   vcs
fuse             ptyp2               ptyqf      tty23  tty5   ttyp7   vcs1
hpet             ptyp3               ram0       tty24  tty50  ttyp8   vcs2
hugepages        ptyp4               random     tty25  tty51  ttyp9   vcs3
i2c-0            ptyp5               rfkill     tty26  tty52  ttypa   vcs4
iio:device0      ptyp6               rtc        tty27  tty53  ttypb   vcs5
imrtest0         ptyp7               rtc0       tty28  tty54  ttypc   vcs6
initctl          ptyp8               shm        tty29  tty55  ttypd   vcsa
input            ptyp9               snd        tty3   tty56  ttype   vcsa1
kmem             ptypa               spidev1.0  tty30  tty57  ttypf   vcsa2
kmsg             ptypb               stderr     tty31  tty58  ttyq0   vcsa3
log              ptypc               stdin      tty32  tty59  ttyq1   vcsa4
loop-control     ptypd               stdout     tty33  tty6   ttyq2   vcsa5
loop0            ptype               tty        tty34  tty60  ttyq3   vcsa6
loop1            ptypf               tty0       tty35  tty61  ttyq4   zero
mem              ptyq0               tty1       tty36  tty62  ttyq5
mmcblk0          ptyq1               tty10      tty37  tty63  ttyq6
mmcblk0p1        ptyq2               tty11      tty38  tty7   ttyq7
root@galileo:/dev# [  237.240172] usb 1-1: new high-speed USB device number 3 using ehci-pci
[  237.971906] uvcvideo: Found UVC 1.00 device USB2.0 Camera (1e4e:0110)
[  237.988966] input: USB2.0 Camera as /devices/pci0000:00/0000:00:14.3/usb1/1-1/1-1:1.0/inpu
t/input3
```

Figure 12: Shows when the camera was connected or disconnected, Video0

Section 10: Appendix

```
#include <linux/i2c-dev.h>// access i2c adapter from linux program; this may be incorrect
library

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "i2c.h"
#include "pic.h"

int main() {

	unsigned int pic_counter;	// # of pictures taken
	int temp_sensor_handle;
	double temp, temp_threshold;


	/*declare and initialise variables here*/

	/* PART 1 - COMPLETE FIRST OBJECTIVE */
	// Note: I2C on A4 (SDA) and A5 (SCL) of galileo
	temp_sensor_handle = InitTempDevice(ADAPTER_NUMBER);	// get I2C handle to temp
sesnor
	pic_counter = 0;

	/* protocol to determine temperature threshold dynamically */
	puts("Get ready to put hand on the sensor...");
	sleep(5);
	puts("Put hand on temperature sensor. Do not remove until instructed to do so.");
	sleep(5);
	temp_threshold = sampleTemp(temp_sensor_handle);
	puts("Now take your hand off the sensor.");

	printf("Threshold: %2.2lf degrees Celsius\nProgram will begin in 5
seconds...\n\n", temp_threshold);
	sleep(5);

	/* PART 2 - COMPLETE SECOND AND THIRD OBJECTIVES */
	// infinite loop - exit from inside
	while (1) {

		temp = sampleTemp(temp_sensor_handle);// read temperature via I2C to temp
sensor

		if (temp > temp_threshold) {
			// temperature is above threshold, so take picture and update
counter

			++pic_counter;
			printf("\rYour picture is being taken. Temperature (C) = %2.2lf\n_",
temp);
			takePicture(pic_counter);	// stores as [pic_counter
value].jpg
```

```
                        if (pic_counter >= PICTURE_LIMIT) {
                                // if enough pictures have been taken, exit
                                return 0;
                        }

                } else {

                        printf("\r%2.2lf", temp);
                }


        }

        return 0;               // the code shall never reach this point

} // end main
```