Section 1: General Lab Info */0.5 points*

EECE.4800 Microprocessors II and Embedded System Design

11024

Lab 4: Multithreaded Programming

Professor Yan Luo

Group number: 1

Jose Velis

December 13, 2017

December 21, 2017

Section 2: Contributions */1 points*

1. Group Member 1 – Jose Velis

   Responsible for temperature sensor I2C implementation. Light troubleshooting and finding the temperature threshold for camera activation. Some help with ADC and PWM implementation and configuration. Basis for menu.

2. Group Member 2 – Grayson Colwell

   Responsible for main program logic implementation and some camera code (date/time). Troubleshooting. Setting up Galileo for file transfers and compiling. Also for merging and modifying Lab 3 and 2 (PIC communication protocol/ Servo control) code to implement for the requirements of the lab. Most of the modifications were for handling the threads.

3. Group Member 3 – Andy MacGregor

   Responsible for implementing opencv library and the rest of camera code. Some help with temperature sensor implementation. This consisted of clearing up how the mraa library works and what functions do. Troubleshooting and setting up Galileo for file transfers and compiling. Also for setting up the server code, and threading code. Helped with main loop logic as well.

Section 3: Purpose */0.5 points*

The purpose of the lab was to implement a variety of different sensor devices. These sensor devices included light intensity, temperature, and a USB camera. The lab also required that the data collected from these sensors be sent to a web server. The pictures were required to be stored on the Galileo locally. The end goal was a multithreaded client-sensor program that has an individual thread for communication with the sensors, another thread for web server activities, and finally a thread to control user selection and handling of variables.

Section 4: Introduction */0.5 points*

In this lab, I²C was used to communicate with a temperature sensor. The TMP102 temperature sensor was connected to the Galileo via the SDA and SCL pins. The device operates at 3.3V. on the Galileo a simple command interface was used to select which function to do. Connected to the Galileo was also a USB camera. The camera software was implemented using the opencv library. This library handles the communications protocol and camera functions. In addition to the lab 3 requirements, the pthread library was used to implement various parts of the program using different threads to handle different parts. A thread running on the Galileo was responsible for sending user input and receiving data from the PIC16. A communications protocol was used (from lab 2) to send command signals to the PIC16 microcontroller. The microcontroller was used to control a servo and to respond to other, more software based, requests. The Intel Galileo Gen 2 board was used as the master and the PIC16 as the slave. As mentioned, the PIC16 was simply receiving, acknowledging, and transmitting a response back to the Galileo through a simple bus protocol. The protocol was implemented through the Galileo GPIO pins and the PIC16 GPIO pins. The bus protocol was designed as required by the lab manual. Another thread was responsible for sending certain data to a web server.

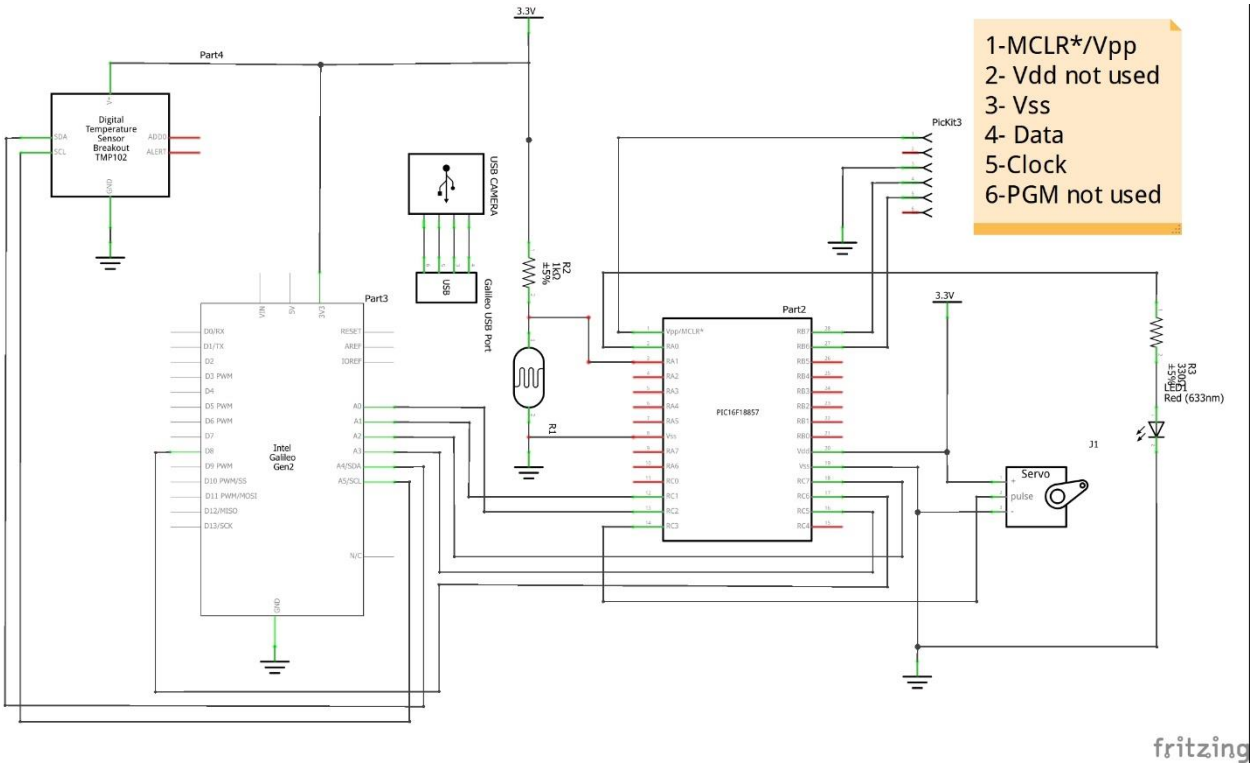
Another thread was responsible for showing the user the options that the program has (menu), as described in the lab manual. Lastly, a thread was used to check the status of the sensors.

Section 5: Materials, Devices and Instruments */0.5 points*

- FTDI cable, operates at 5V
- Breadboard & wiring
- Lab oscilloscope and power supply
- Intel Galileo Gen 2 embedded computer system
- USB Camera
- TMP102 i2c temperature sensor
- Microchip PicKit, Version 3, used to program the PIC device, operates at 5V
- Microchip PIC16F18857 microcontroller, 28 dip device, operates at 3.3 volts

- MPLAB X IDE with XC8 compiler
- Servo motor, SG90, operates between 4.8v -5V
- FTDI cable, operates at 5V
- Photoresistor, measures light intensity, operates between 0V-5V
- Resistors, 1kΩ and 330Ω
- LED

Section 6: Schematics */0.5 points*



Section 7: Lab Methods and Procedure /2 points

Hardware Design:

This lab simply involved rebuilding the previously used sensors. The light sensor and servo were connected to the PIC, like in lab 2. While the temperature sensor was connected to the Galileo. The camera was connected to Galileo via USB.

Software Design:

The software was design around the pthread, mraa and opencv libraries. The mraa library was used to communicate with the TMP102 temperature sensor and the opencv for the camera. Pthreads was used to initialize threads, control data flow, and for accessing global variables. A basic flow chart can be seen in figure 1.

The basic code on the Galileo is as follows:

- Initialize all libraries and their objects
- Start threads, start i2c and camera interfaces, this is done through functions in the pthread, mraa and opencv libraries, respectively.
- Prompt user for a command
- Send to command to the thread that send the user input out to the desired function
- Based on user input:

    - Obtain the temperature or take a picture based on a set temperature threshold
    - Change the temperature threshold
- Upload data via HTTP POST method
- Re-do

Below is also some calculations that helped with timing and speed of the servo rotation.

```
Timer2
Period
calculation
              ========================================


              //Calculate pwm frequency for servo
              //choose prescale of 64 just because


              TargetPeriod = (PR2 + 1) * 4 * 1/_XTAL_FREQ * Timer2Prescale
                TargetPeriod = 20ms
                Timer2Prescale = 64
                _XTAL_FREQ = 1M
              PR2 = 20ms / (4 * 1/1M * 64)
              PR2 = 77









              Timer2 Duty Cycle Calculation
              ============================================


              PulseWidth = CCPR4 * 64 / (_XTAL_FREQ/4)


              //I substituted a bunch of values in for CCPR4 and found that values giving a
              pulse
              //width between 1 and 2ms were ...


              15 <= CCPR4  <= 31
```

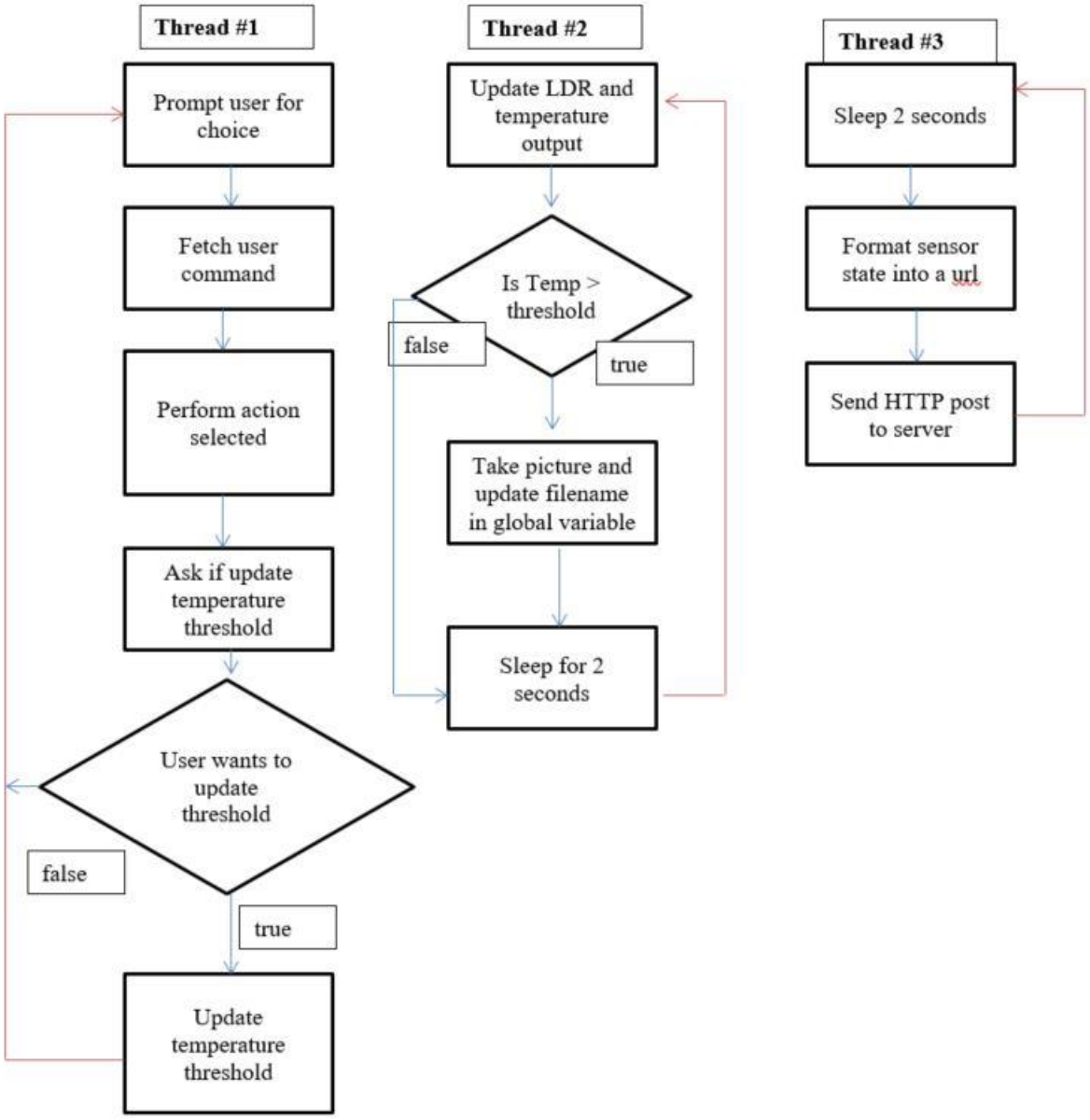


Figure 1 – Software flow chart

Some sample code for the TMP102 sensor is show below:

```
#ifndef
TEMP_SENSOR_H
                #define TEMP_SENSOR_H
```

```
#include "mraa.hpp"


#define TEMPERATURE_REGISTER 0x00    //According to table 1 in the TMP102
Datasheet
#define CONFIG_REGISTER 0x01         //These values will determine what
register we are
#define T_LOW_REGISTER 0x02                 //communicating with..not sure
if we them.....
#define T_HIGH_REGISTER 0x03


#define TMP102Address 0x48
using namespace mraa;




double get_temp(); // temperature in C


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;
}
#endif
```

Below is a code snippet of the camera file, this code is what initializes the camera and also checks to see if the camera is connected. The code also starts to set the file saving mechanisms:

```
#include
"camera.h"
```

```
#include "opencv2/opencv.hpp"


using namespace cv;
using namespace std;


//read image from filename
//if successful, save as a file -- .png and retrun true
//if not, return false.
bool capture_and_save_image(char* filename)
{
  //init webcam on video0 interface
  VideoCapture ourCam;
  ourCam.open(0);


  Mat image;
  bool camIsThere = false;


  //read a frame from the vid camera into image
  camIsThere = ourCam.read(image);


  //test if camera not connected
  if(!camIsThere)
    return false;
  //test if a blank image was grabbed
  if(image.empty())
    return false;
```

Below is a snippet of the pthread creating of threads and the pthread_mutex_lock and unlock commands used for accessing global variables such as the temperature threshold and filename.

```
int
main()
```

```
{
  ////init shared stuff
  pthread_mutex_init(&myData_m, NULL);
  pthread_mutex_init(&gpio_m, NULL);


  //init data structure
  myData.picOnline = false; //true if pic is online
  myData.adcData = 0; //value of PIC adc
  strncpy(myData.fileName, DEFAULT_FILENAME, MAX_FILENAME); //filename of last


  //init GPIO pins
  fileHandleGPIO_4 = openGPIO(GP_4, GPIO_DIRECTION_OUT);
       fileHandleGPIO_5 = openGPIO(GP_5, GPIO_DIRECTION_OUT);
       fileHandleGPIO_6 = openGPIO(GP_6, GPIO_DIRECTION_OUT);
       fileHandleGPIO_7 = openGPIO(GP_7, GPIO_DIRECTION_OUT);
       fileHandleGPIO_S = openGPIO(Strobe, GPIO_DIRECTION_OUT);


       //test
       //testMyBus();


  ////create threads
  pthread_t command_thread;
  pthread_t sensor_control_thread;
  pthread_t net_thread;


  ////run threads
  pthread_create(&command_thread, NULL,commandLoop, NULL);
  pthread_create(&sensor_control_thread, NULL, sensor_control, NULL);
  pthread_create(&net_thread, NULL, serverPostLoop, NULL);



  pthread_join(command_thread, NULL);


  //jump out if the command loop exits
  printf("Main thread stopped\n");
  pthread_exit(NULL);
```

```
        }
```

Pthread mutex command:

```
        //print
 the current
 temperature
 reading
                                pthread_mutex_lock(&myData_m);
                                printf("Current Temperature settings:\n\ttemp =
                %lf\n\tthreshold = %lf\n", cur_temp, temperature_threshold);
                                pthread_mutex_unlock(&myData_m);


                                //ask the user to change the temp threshold
                                printf("Would you like to change the Temperature
                Threshold?\n (1 = yes)\n");
                                scanf(" %d", &temp_thresh_input);
                                if (temp_thresh_input == 1)
                                {
                                        pthread_mutex_lock(&myData_m);
                                        printf("Enter the new Temperature Threshold:\n");
                                        scanf("%lf", &temperature_threshold);
                                        pthread_mutex_unlock(&myData_m);
                                }
                        } while (input != 11);
                }
```

Section 8: Trouble Shooting */1 points*

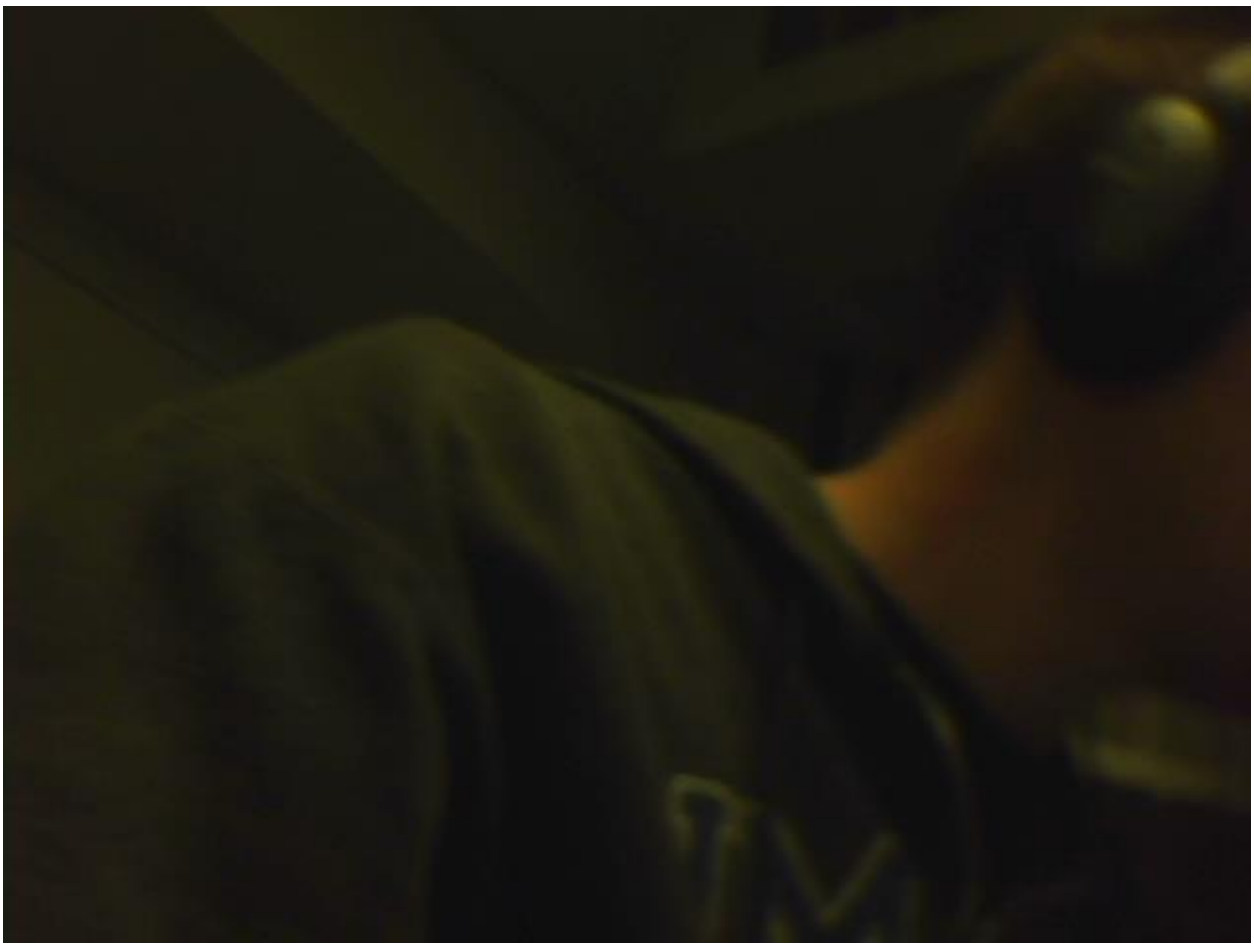
Issue 1: Learning the pthread commands was a bit overwhelming. But with the documentation and examples this was overcome.

Issue 2: Connectivity issues to the web server sometimes hindered the ability to test the HTTP POST uploads. The only thing that could be done was test if the files were at least being stored locally, until the server came back up.

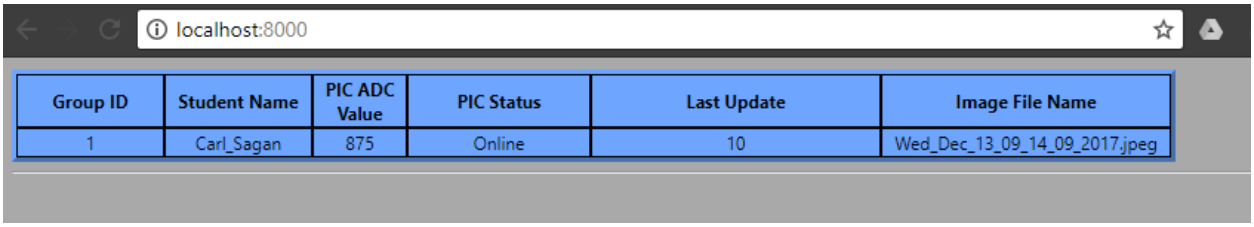
Issue 3: It was tricky to know exactly what and when each variable had to me accessed with the pthread_mutex. To help overcome this, a list was made of variables that were needed in other threads. If other variables arose they were handled as they came along.

Section 9: Results /0.5 points

Below is a sample picture from the USB camera.

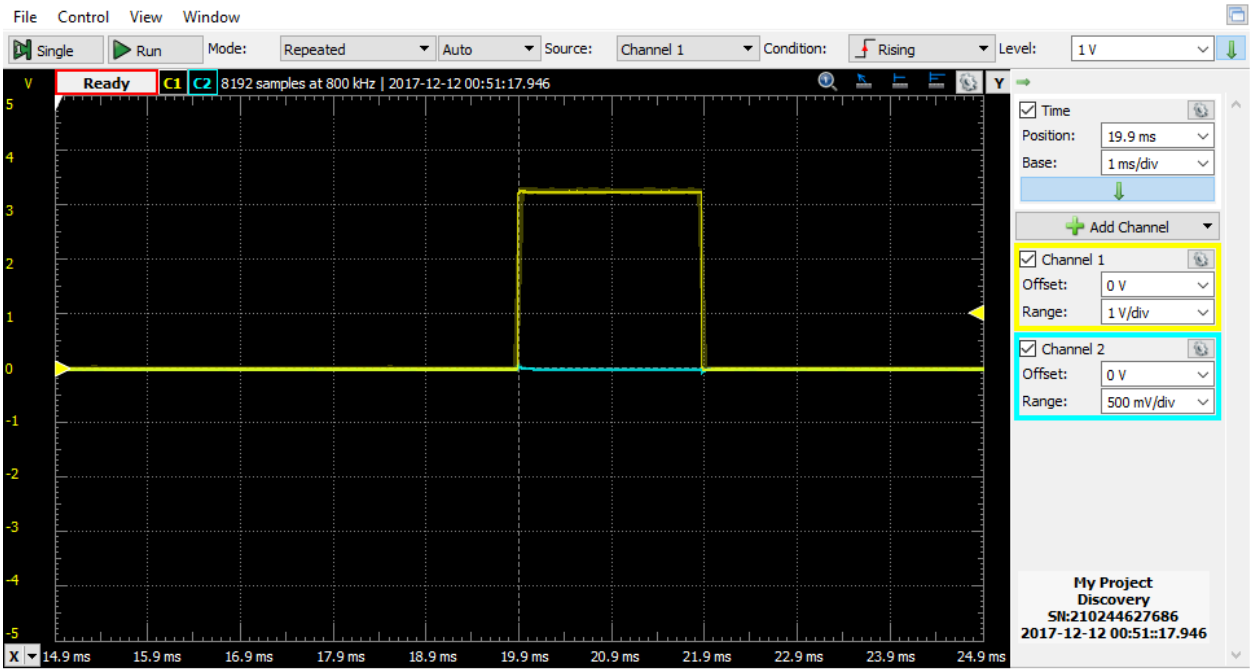


Below you can also see a post to the web server:



Group ID	Student Name	PIC ADC Value	PIC Status	Last Update	Image File Name
1	Carl_Sagan	875	Online	10	Wed_Dec_13_09_14_09_2017.jpeg

Also, the output of the PWN is shown below, this is a max pulse or 100% duty cycle:



Below is a i2c detect command ran on the Galileo to see if the temperature sensor was 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#
```