Sean Webster

Microprocessors II

Lab 3: Building Linux Kernel and Controlling an I2C Device

Group 8 – Sean Webster, Eric Craaybeek, Munib Elkhatib

Handed in 11/28/2016 - Due 11/28/2016

**Contributions to this lab:**

* Eric Craaybeek
  + Created OpenCV code
  + Troubleshooted OpenCV code
  + Soldered TMP102 sensor
  + Created Schematic
* Munib Elkhatib
  + Worked on I2C Code
  + Troubleshooted I2C code
  + Created main loop code
  + Created flow charts
* Sean Webster
  + Created I2C Code
  + Troubleshooted I2C code
  + Troubleshooted main loop
  + Took pictures

**Purpose:**

The purpose of this lab was to understand I2C bus protocol, be able to control an I2C device using Linux on a Galileo board, and be able to capture, store and process camera images on Linux.

**Introduction:**

After learning how to interface a sensor device with the Galileo GPIO ports, adding a couple of new devices to the system was necessary to delve deeper into embedded Linux. A TMP102 was used to further explore the I2C bus protocol, an important communication method in embedded Linux.

**Materials and Devices:**

* TMP102 Breakout board
* Jumper wires
* Galileo board
* Serial cable
* Ethernet cable
* Galileo power cable
* USB Web camera

**Schematic:**

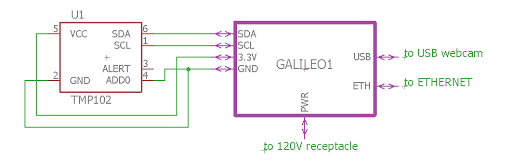


Figure 1 – Schematic of whole lab.

**Lab Methods and Procedure:**

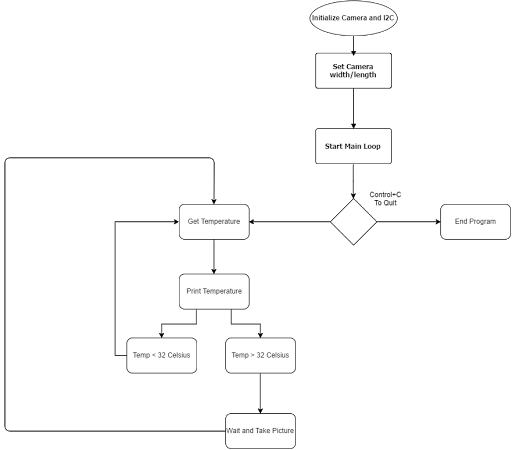
****

Figure 2 – program flowchart for the program.

**Code Explanation:**

1. GetTemp() Function
   1. Initialize variables
      1. I – iterator
      2. Err – err code
      3. Fd – file directory
      4. Dev – directory for I2C bus file
      5. Addr – I2C address for device
   2. Open file at address dev
      1. If error
         1. Return error and print error message
   3. Set I2C device at addr as slave, return err as return code
      1. If err
         1. Print error code
   4. For loop
      1. Output enable | read input i
      2. Write command to file to output to I2C
      3. Delay
      4. Read from I2C file into value[i]
      5. If err
         1. Print error
      6. Delay
   5. Set MSB to 4th byte read
   6. Set LSB to 2nd byte read
   7. OR LSB and shifted MSB to create one value
   8. Set value to Celsius with temp\*.0625
   9. Print temp
   10. Close file
   11. Delay
   12. If temp >= 32
       1. Return that temp is what we want
   13. If temp < 32
       1. Reset hasTaken to allow for another picture to be taken
       2. Return 0
2. TakePic() Function
   1. Takes arguments int i2 and CvCapture\* camera
   2. Initial I to 0 to allow for loop iteration
   3. Print ‘taking picture’
   4. Initialize character array for file path to save image
   5. While loop
      1. Initialize image and take picture
      2. Save image
      3. I++
   6. Return
3. Main loop
   1. Create car for camera
   2. Set camera var to any camera
   3. Set up camera dimensions (640x480)
   4. Detect errors
      1. If error
         1. Print ‘no camera detected!’
   5. If no error
      1. While loop
         1. Initialize temperature flag as GetTemp() function return
         2. If temperature flag triggered and no picture taken yet
            1. Take picture
            2. Set hasTaken to 1
            3. Increase iterator loop

The I2C code revolved around reading from the I2C dev file and writing the value to it. DetectI2C and readI2C were helpful Linux commands that helped get started into the world of I2C. Ioctrl() is the most important I2C command, as it controls most I2C IO. The output from the TMP102 sensor came out as 3 nibbles of data, with use not needing the last nibble because we didn’t need that kind of resolution.

The OpenCV code was a bit more simplified with all of the library calls it had. It would initialize a camera, the picture dimensions, etc. Then you would use the cvSaveImage() function to save the image that you took with cvQueryFrame().

**Troubleshooting:**

* Problem: unsure of I2C address
  + Solution: Used I2Cdetect function built into Linux to determine connected I2C device addresses
* Problem: Networking folder deletes itself every time Galileo shutdown
  + No solution found
* Problem: Saved picture too dark.
  + Solution: looped image taking process, which fixed problem, almost like a sort of exposure timer.
* Problem: Temperature would stay the same after image taken, taking many images.
  + Solution: Created variable that prevented image from being taken until temperature fell

**Results:**

* Pictures came out nicely. Even nicer after fiddling with the lense.
* Adding hasTaken variable really improved picture taking process, allowing for single pictures to be taken.

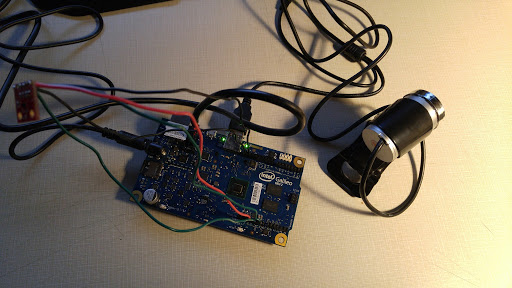


Figure 3 – Picture of lab configuration

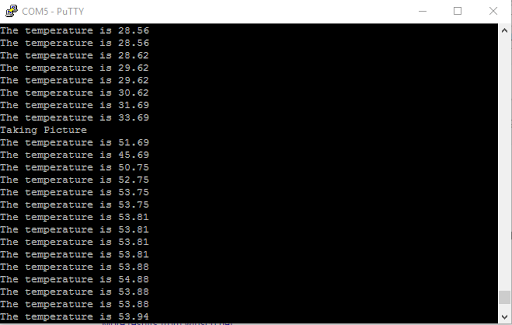


Figure 4 – Screen capture of running program.



Figure 5 – Image taken by web camera, from code in figure 4.

**Appendix:**

/\*

\* File: Building Linux Kernel and Controlling an I2C Device

\*

\* Authors: Munib Elkhatib, Sean Webster, Eric Craaybeek

\*

\* 11/14/2016

\*

\*/

#include <opencv2/objdetect/objdetect.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include "cv.h"

#include "highgui.h"

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <linux/i2c-dev.h>

#include <fcntl.h>

#define BW\_THRESHOLD 16

using namespace std;

using namespace cv;

//prototype functions

void TakePic(int i2, CvCapture\* camera);

int GetTemp();

//var to protect from taking two pics in a row

int hasTaken = 0;

int main()

{

//create var for camera

CvCapture\* capture = 0;

//set camera var to any camera, set up camera dimensions

capture = cvCaptureFromCAM( -1 ); //0=default, -1=any camera, 1..99=your camera

cvSetCaptureProperty( capture, CV\_CAP\_PROP\_FRAME\_WIDTH, 640 ); //175

cvSetCaptureProperty( capture, CV\_CAP\_PROP\_FRAME\_HEIGHT, 480);//144

//error detection

if(!capture) cout << "No camera detected" << endl;

if( capture )

{

int i=0;

//start main loop

while(1)

{

int temperature\_flag;

//if get temp returns 1, take picture

temperature\_flag = GetTemp();

if(temperature\_flag==1 && hasTaken ==0)

{

//take the picture and wait a few seconds

hasTaken = 1;

TakePic(i,capture);

i++;

}

}

}

}

//get temperature function

int GetTemp()

{

int i, err, fd;

unsigned char command[2]; //i2c command

unsigned char value[4]; //i2c return data

useconds\_t delay = 2000;

//i2c address

char \*dev = "/dev/i2c-0";

int addr = 0x48;

fd = open(dev, O\_RDWR ); //open file

if(fd < 0)

{

perror("Opening i2c device node\n");

return 1;

}

err = ioctl(fd, I2C\_SLAVE, addr); // Set i2c device at addr as slave, return err as return code

if(err < 0)

{

perror("Selecting i2c device\n");

}

for(i = 0; i < 4; i++)

{

command[0] = 0x40 | ((i + 1) & 0x03); // output enable | read input i

command[1]++;

err = write(fd, &command, 2);

usleep(delay);

// the read is always one step behind the selected input

err = read(fd, &value[i], 1);

if(err != 1)

{

perror("reading i2c device\n");

}

usleep(delay);

}

//convert data to celcius

unsigned char MSB;

unsigned char LSB;

MSB = value[3];

LSB = value[2];

float temp = (MSB << 8 | LSB) >> 4;

temp = temp\*.0625;

printf("The temperature is %.2f\n", temp);

//close file

close(fd);

usleep(500000);

//return value used with take picture condition

if (temp>=32)

return 1;

if (temp<32)

{

hasTaken=0;

return 0;

}

}

//take picture function

void TakePic(int i2, CvCapture\* camera)

{

int i = 0;

printf("Taking Picture\n");

//path to save picture

char path[100];

sprintf(path,"/media/mmcblk0p1/project3/pic%d.jpg",i2);

//loop for taking picture

while(i<5) {

IplImage\* picture = cvQueryFrame( camera ); //initialize image and take picture

cvSaveImage( path , picture); //save image

i++;

}

return;

}