SE3910 Lab 2: Basic IO with the Beagleboard

Due: March 24, 2014

1. Introduction

In most programming languages, you start out by writing a program which prints hello to the console. Having mastered this skill, you are then considered an expert programmer.

For embedded systems, it's often difficult to start saying hello world. So instead, we blink lights for our first projects. And we continue to use blinking lights even in our final versions of code. Think of how many devices you have in your house tell you that they are working properly by blinking lights.

2. Lab Objectives

- Explain how Embedded Linux can perform low level input and output through device drivers.
- Construct an embedded Linux program which performs low level Linux I/O to read and write to an output pin.
- Measure the systemic latency of input and output control using embedded Linux.
- Construct software which uses both polling and interrupts to detect input value changes.

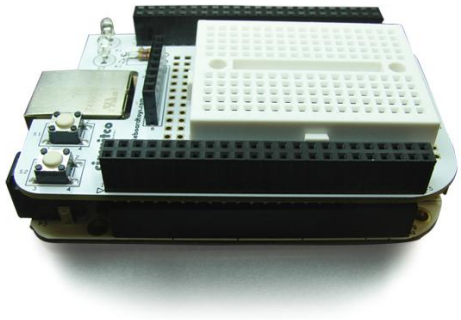
3. Laboratory Equipment Needed

- One Beaglebone Black embedded systems board
- One 10W 5 V power supply
- 2 ethernet networking cables
- 1 Beaglebone protoboard
- One Digilent Analog Discovery Oscilliscope

4. Part 1: Setting up our hardware

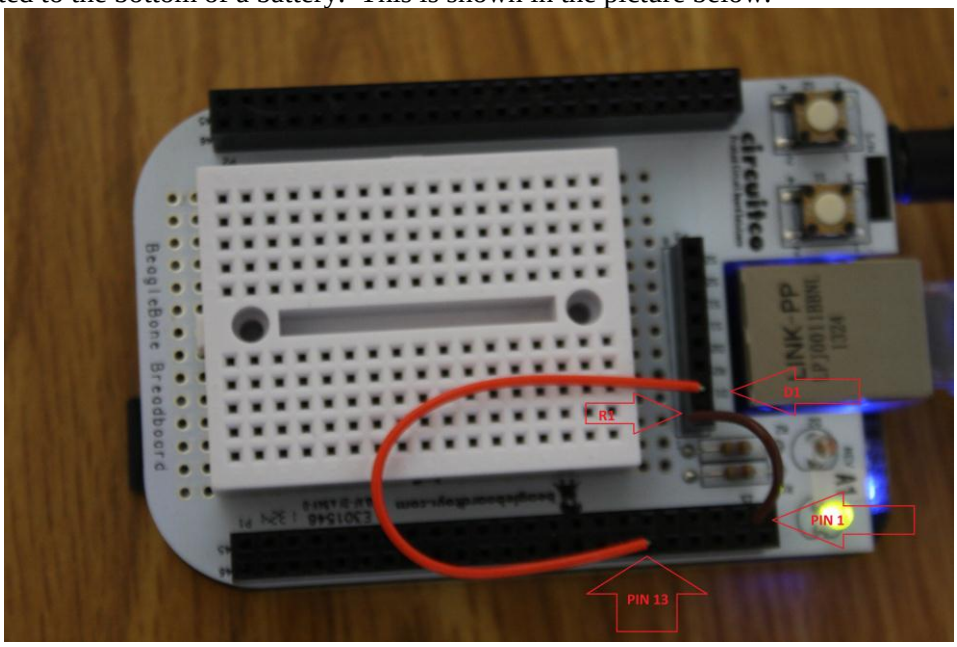
The first thing we want to do is to setup our hardware. The hardware involves what is referred to as a cape. A cape, in Beaglebone terminology, is a peripheral that can be attached to the beaglebone.

The first cape we are going to work with is the BeagleBone Breadboard cape.

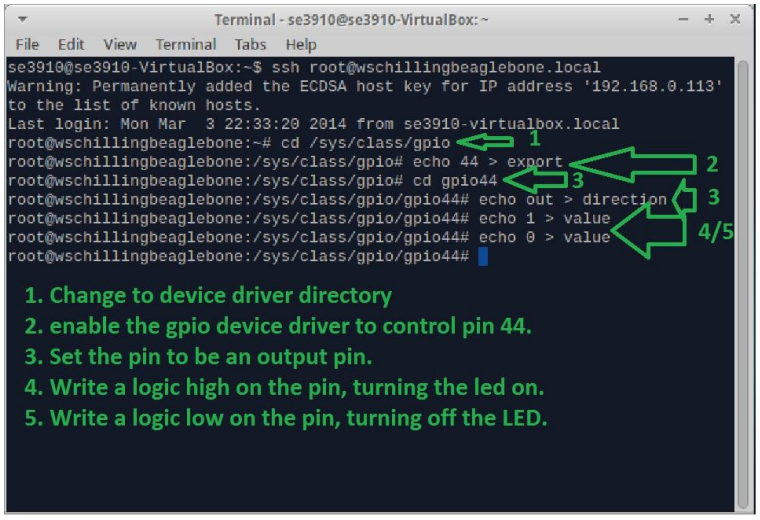


The Beaglebone breadboard cape allows us to prototype circuits on the surface of the cape. For this lab, we are going to use the cape to wire a switch and an LED. We will then use these setups to measure the performance of our system.

The first thing you want to do is to wire up LED D1 to Header P8 pin 12. This will allow the LED to turn on when a voltage is applied to this pin and to turn off when no voltage is applied. This is accomplished by adding two wires to the development board. The first wire connects the LED to the pin. The second wire connects the ground of the LED to the ground of the board, allowing electric current to flow. You can think of the first wire as being the wire connected to the top of a battery and the second wire as the wire connected to the bottom of a battery. This is shown in the picture below.



Once you have wired up the circuit, turn on your Beaglebone and connect via a secure connection (ssh) to the terminal. In the terminal window, issue the commands shown in the figure below. When executed, the command "echo 0 > value" and "echo 1 > value"

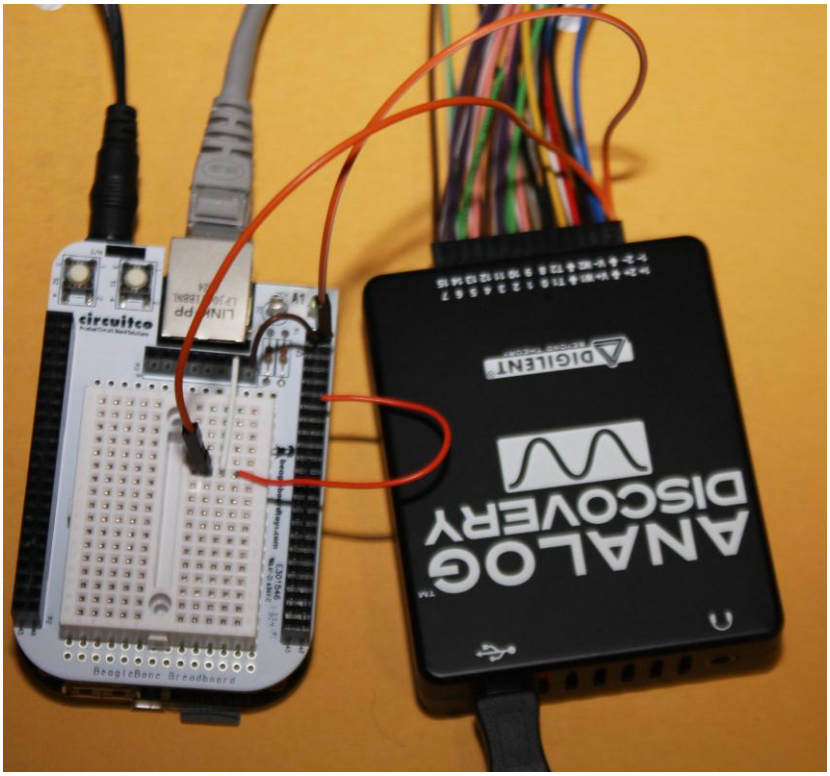


5. Part 1: The blinking light

The first piece of code we want to work with is a simple program which will turn a light on and off. The code for this is shown in Appendix A and is available for download from the course website. In essence, it will use the GPIO device driver to turn a light on and off. We will use this code

Upload the code to your Beagleboard and compile it to run. In essence, if you type the command **blinkLED 400 500000 25** the LED should blink 25 times at a rate of once per second. (The value of 500000 comes from the LED being on for 500ms and off for 500ms in each one second cycle.)

To verify that this is happening, connect your Digilent scope up to the board. You'll need to wire up the circuit slightly differently. Instead of going directly from pin 13 to D1, you will need to go to the breadboard and then run a second wire from the breadboard to the diode. On the Digilent Analog Discovery board, connect wire 1+ to this junction point and connect 1- to Pin2. Take a look at the photo below for better details.



This should give you a period of 1 second. With the oscilloscope, measure the period and record it. Also measure the frequency and record it. Also measure the period. In the same data date, calculate the frequency and period and record and determine the difference.

Now that you have done this, change the program so that the light blinks on and off 10 times per second. (Hint: You will need to determine the period.) Run this program and measure the result with the oscilloscope. Keep reducing this time period until your rate is 10000 times per second.

6. Part 2 Reading input

Output is great. But, what we would like to do now is to read input into the system. This is a little bit harder to do. Well, not really harder, but just different.

The program in appendix B allows a user to read input from a hardware device. It allows you to select a given GPIO port for input and to read from it. When the program executes, pushing a pushbutton will cause a message to be displayed on the terminal console.

To make this work, you will need to wire up a small circuit on the protoboard. The circuit consists of a resistor and a couple of wires.

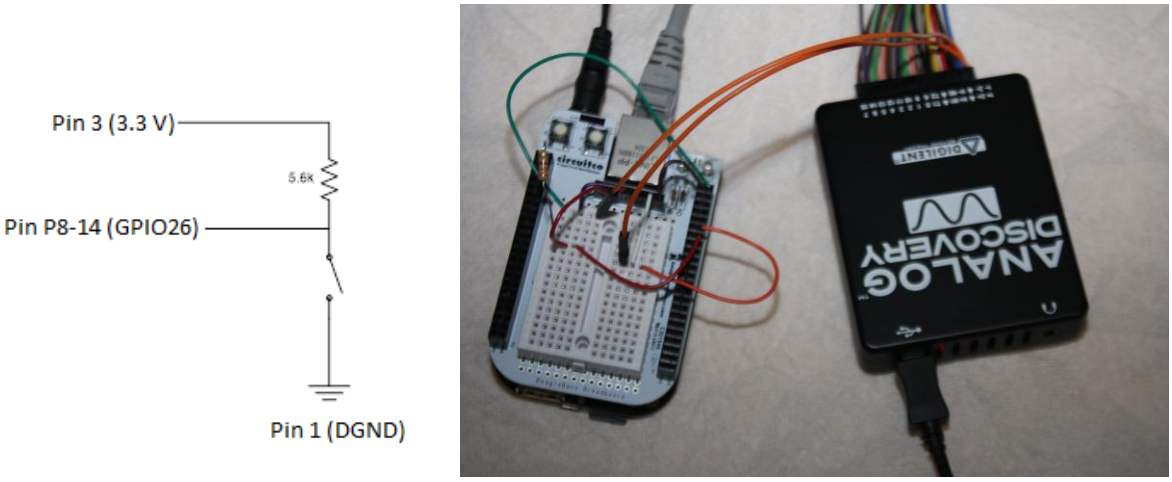


Figure 1 (Left)The IO circuit that you will need to construct. (Right) A picture of the wired up circuit. Note that the circuit from part 1 is still present on the protoboard.

Enter the program in Appendix B program and build it. Before running the program, open a second terminal connection to the machine and start the top program running in the shell. What is the top user of system resources right now? Now start the program and again look at system utilization. How much of the CPU resources is this program using to execute whether the button is pressed or not?

7. Part 3: Measuring latency

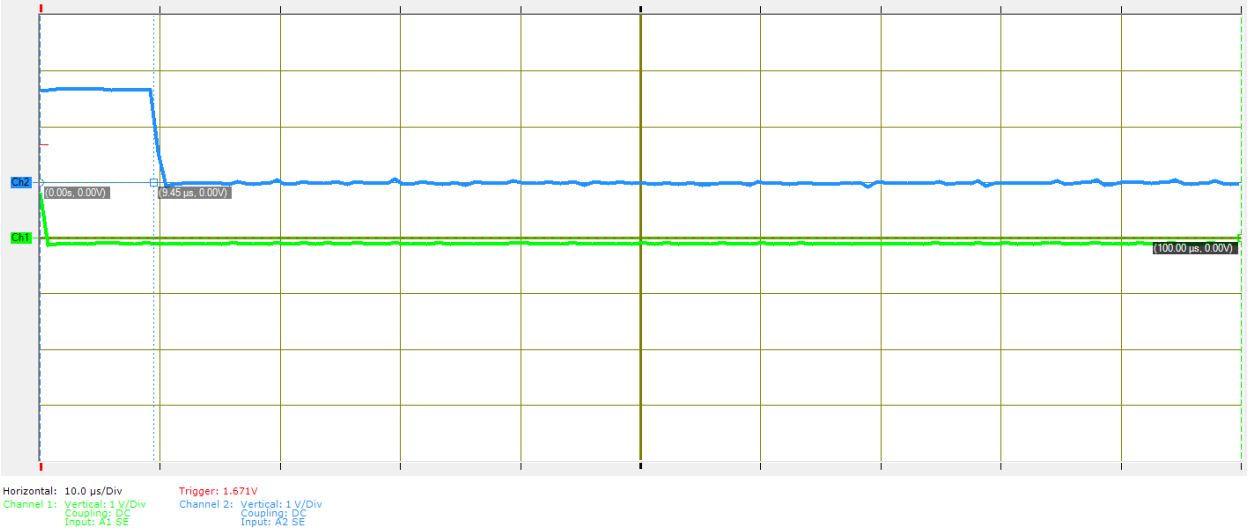
Latency can be defined as "the time that elapses between a stimulus and the response to it." As we design our computer system, there is always latency. In the computer system, the latency is caused by the delay between an input signal changing and the appropriate output action being taken.

We would like to determine the latency between a push button being pressed and the LED changing state. To do this, you will need to use the program given in Appendix C. This program will combine steps 1 and 2 together into a single program. You will read the input from pushbutton, and based upon whether the pushbutton is pressed or not, turn the LED either off or on.

Once you have completed this task, test the program using the setup that you now have. You should see that pressing the button turns off the light and releasing the button turns the light on.

To measure latency, we will need to change the input ever so slightly as well as connect your Analog discovery device to the unit. We will connect the second oscilloscope channel to the device so that we can see exactly when the pushbutton is pressed and when the LED changes state.

The figure below shows a sample capture of the oscilloscope in operation. Channel 1 (green) represents the output of the signal generator while channel 2 (blue) represents the output of the LED. Notice the slight, in this case 9.45 us, delay between the change in the input and the change in the output.



8. Interrupt driven input

In general, it seems as if polled input is very inefficient for our system. What we want to do is change our input so that it is interrupt driven. The code in appendix D shows an interrupt driven system. What we would like to do is modify the code from the previous step to make it interrupt driven. This should help with our latency as well as reduce the processor utilization. After getting the code to work, measure the latency and processor utilization of this program.

9. Deliverables / Submission

Each person will be responsible for submitting one report with the following contents:

1. Introduction -> What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Oscilloscope Captures
    a. For Each problem where you are to measure something on the system, include an oscilloscope capture of the output.
3. Measured Data in Tabular format
    a. For each step above, include a data table summarizing the latency and processor utilization for each step.

4. Things gone right / Things gone wrong -> This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
5. Conclusions -> What have you learned with this experience? What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file.

If you have any questions, consult your instructor.

10. Appendix A: Code to blink the light.

```
/****************************************************************
 * This program will allow on to turn a given port on and off at a user
 * defined rate.  The code was initially developed by Dingo_aus, 7 January 2009
 * email: dingo_aus [at] internode <dot> on /dot/ net
 * From http://www.avrfreaks.net/wiki/index.php/Documentation:LinuxGPIO#gpio_framework
 *
 * Created in AVR32 Studio (version 2.0.2) running on Ubuntu 8.04
 * Modified by Mark A. Yoder, 21-July-2011
 * Refactored and further modified by Walter Schilling, Summer 2012 and Winter 2013-2014.
 */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>

#define SYSFS_GPIO_DIR "/sys/class/gpio"

// create a variable to keep track of the port that is to be opened as a string.
static char ioPort[56];

/****************************************************************************
 * This method will setup the GPIO port to allow the user to perform output on the given port.
 *
 * @param uint32_t gpioPort - This is the number of the GPIO port that is to be opened.
 *
 ****************************************************************************/
void setupIOPort(uint32_t gpioPort)
{
        FILE* fp;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Using sysfs we need to write the 3 digit gpio number to /sys/class/gpio/export
        //This will create the folder /sys/class/gpio/gpio37
        if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL)
        {
                printf("Cannot open export file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write the value of our GPIO port to the file.
        sprintf(&set_value[0], "%d", gpioPort);
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);

        printf("...export file accessed, new pin now accessible\n");

        //SET DIRECTION
        //Open the LED's sysfs file in binary for reading and writing, store file pointer in fp
        // Strat by creating a string representing the port that needs to be opened.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/direction");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open direction file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);
```

```
        //Write our value of "out" to the file
        strcpy(set_value,"out");
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);
        printf("...direction set to output\n");
}

/****************************************************************************
 * This method will blink a given light a number of times at a given rate.
 *
 * @param  FILE *fp - This is a pointer to the io device driver which will be manipulated to
 *              turn the LED on and off.
 * @param  uint32_t onOffTime - This is the length of time that the led is to either be turned on
or turned off.
 *              It is given in nanoseconds.
 * @param  uint32_t numberOfBlinks - This is the number of on-off cycles that shall occur before
the loop exits.
 ****************************************************************************/
void blinkLight(FILE *fp, uint32_t onOffTime, uint32_t numberOfBlinks)
{
        //Integer to keep track of whether we want on or off
        uint8_t toggle = 0;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Run an infinite loop - will require Ctrl-C to exit this program
        while(numberOfBlinks > 0)
        {
                toggle = !toggle;
                if(toggle)
                {
                        //Set pointer to beginning of the file
                        rewind(fp);
                        //Write our value of "1" to the file
                        strcpy(set_value,"1");
                        fwrite(&set_value, sizeof(char), 1, fp);
                        fflush(fp);
                        printf("...value set to 1...\n");
                }
                else
                {
                        //Set pointer to beginning of the file
                        rewind(fp);
                        //Write our value of "0" to the file
                        strcpy(set_value,"0");
                        fwrite(&set_value, sizeof(char), 1, fp);
                        fflush(fp);
                        printf("...value set to 0...\n");
                }
                //Pause for a while
                usleep(onOffTime);
                numberOfBlinks--;
        }
}

/**
 * This is the main method for this program.  It will cause the LED to blink at the given rate.
 */

int main(int argc, char** argv)
{
        /* This variable is a file pointer to the file interface for the GPIO device driver
           which controls the output pin. */
        FILE *fp;
        uint32_t onOffTime;    // Time in micro sec to keep the signal on/off
        uint8_t gpioPort;   // This is the GPIO port that is to be used for this program.
        uint32_t numberOfBlinks;
```

```
        if (argc < 4) {
                printf("Usage: %s <port number> <on/off time in us> <Number of blinks>\n\n",
argv[0]);
                printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
                exit(-1);
        }

        // Convert the input into the appropriate parameters.
        gpioPort = atoi(argv[1]);
        onOffTime = atoi(argv[2]);
        numberOfBlinks = atoi(argv[3]);

        printf("\n*********************************\n"
                "*  Welcome to PIN Blink program  *\n"
                "*  ....blinking gpio %d         *\n"
                "*  ....period of %d us.........*\n"
                "*********************************\n", gpioPort, 2*onOffTime);

        setupIOPort(gpioPort);

        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/value");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open value file.\n");
                exit(1);
        }

        blinkLight(fp, onOffTime, numberOfBlinks);

        fclose(fp);
        return 0;
}
```

11. Appendix B: Input code

```
/*****************************************************************
 * This program will allow on to turn a given port on and off at a user
 * defined rate.  The code was initially developed by Dingo_aus, 7 January 2009
 * email: dingo_aus [at] internode <dot> on /dot/ net
 * From http://www.avrfreaks.net/wiki/index.php/Documentation:LinuxGPIO#gpio_framework
 *
 * Created in AVR32 Studio (version 2.0.2) running on Ubuntu 8.04
 * Modified by Mark A. Yoder, 21-July-2011
 * Refactrored and further modified by Walter Schilling, Summer 2012/ Winter 2013-2014.
 */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>

#define SYSFS_GPIO_DIR "/sys/class/gpio"

// create a variable to keep track of the port that is to be opened as a string.
static char ioPort[56];

/*****************************************************************************
 * This method will setup the GPIO port to allow the user to perform output on the given port.
 *
 * @param uint32_t gpioPort - This is the number of the GPIO port that is to be opened.
 *
 *****************************************************************************/
void setupIOPort(uint32_t gpioPort)
{
        FILE* fp;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Using sysfs we need to write the 3 digit gpio number to /sys/class/gpio/export
        //This will create the folder /sys/class/gpio/gpio37
        if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL)
        {
                printf("Cannot open export file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write the value of our GPIO port to the file.
        sprintf(&set_value[0], "%d", gpioPort);
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);

        printf("...export file accessed, new pin now accessible\n");

        //SET DIRECTION
        //Open the LED's sysfs file in binary for reading and writing, store file pointer in fp
        // Strat by creating a string representing the port that needs to be opened.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/direction");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open direction file.\n");
                exit(1);
        }

        //Set pointer to beginning of the file
        rewind(fp);
```

```
        //Write our value of "in" to the file
        strcpy(set_value,"in");
        fwrite(&set_value, sizeof(char), 2, fp);
        fclose(fp);
        printf("...direction set to input\n");
}


/****************************************************************************
 * This method will read the given pin and perform processing on it.
 *
 * @param FILE *fp - This is a pointer to the io device driver which will be read to determine
the state of the pin.
 ****************************************************************************/
void processPin(FILE *fp)
{
        char buffer[10];
        char prevState;

        prevState = '?';

        while (1==1)
        {
                fflush(fp);

                //Set pointer to beginning of the file
                rewind(fp);

                // Read in the state of the pin.
                fread(&buffer[0], 1, 1, fp);

                if (buffer[0]!=prevState)
                {
                        // The state is different.
                        // Print out the appropriate message.
                        if (buffer[0]=='0')
                        {
                                printf("The button is pressed.\n");
                        }
                        else
                        {
                                printf("The button is not pressed.\n");
                        }
                        prevState = buffer[0];
                }
        }
}

/**
 * This is the main method for this program.  It will cause the program to read the state of the
IO pin and write it out to the console.
 */
int main(int argc, char** argv)
{
        /* This variable is a file pointer to the file interface for the GPIO device driver
           which controls the input pin. */
        FILE *fp;
        uint8_t gpioPort;   // This is the GPIO port that is to be used for this program.

        if (argc < 2) {
                printf("Usage: %s <port number> \n\n", argv[0]);
                printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
                exit(-1);
        }

        // Convert the input into the appropriate parameters.
    gpioPort = atoi(argv[1]);
```

```
        printf("\n*********************************\n"
               "*  Welcome to GPIO Input Demo program.  *\n"
               "*  ....Reading gpio %d         *\n"
               "*********************************\n", gpioPort);
        // Setup the IO port.
        setupIOPort(gpioPort);

        // Open the device driver for processing.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/value");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
               printf("Cannot open value file.\n");
               exit(1);
        }
    processPin(fp);

        // Shutdown everything.
        fclose(fp);
        return 0;
}
```

12. Appendix C: Controlled Blinking

```
/****************************************************************
 * This program will allow on to turn a given port on and off at a user
 * defined rate.  The code was initially developed by Dingo_aus, 7 January 2009
 * email: dingo_aus [at] internode <dot> on /dot/ net
 * From http://www.avrfreaks.net/wiki/index.php/Documentation:LinuxGPIO#gpio_framework
 *
 * Created in AVR32 Studio (version 2.0.2) running on Ubuntu 8.04
 * Modified by Mark A. Yoder, 21-July-2011
 * Refactrored and further modified by Walter Schilling, Summer 2012 / Winter 2013-2014.
 */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>

#define SYSFS_GPIO_DIR "/sys/class/gpio"

// create a variable to keep track of the port that is to be opened as a string.
static char ioPort[56];

/****************************************************************************
 * This method will setup the GPIO port to allow the user to perform output on the given port.
 *
 * @param uint32_t gpioPort - This is the number of the GPIO port that is to be opened.
 *
 ****************************************************************************/
void setupOutputPort(uint32_t gpioPort)
{
        FILE* fp;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Using sysfs we need to write the 3 digit gpio number to /sys/class/gpio/export
        //This will create the folder /sys/class/gpio/gpio37
        if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL)
        {
                printf("Cannot open export file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write the value of our GPIO port to the file.
        sprintf(&set_value[0], "%d", gpioPort);
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);

        printf("...export file accessed, new pin now accessible\n");

        //SET DIRECTION
        //Open the LED's sysfs file in binary for reading and writing, store file pointer in fp
        // Strat by creating a string representing the port that needs to be opened.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/direction");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open direction file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);
```

```
        //Write our value of "out" to the file
        strcpy(set_value,"out");
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);
        printf("...direction set to output\n");
}

/****************************************************************************
 * This method will setup the GPIO port to allow the user to perform input on the given port.
 *
 * @param uint32_t gpioPort - This is the number of the GPIO port that is to be opened.
 *
 ****************************************************************************/
void setupInputPort(uint32_t gpioPort)
{
        FILE* fp;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Using sysfs we need to write the 3 digit gpio number to /sys/class/gpio/export
        //This will create the folder /sys/class/gpio/gpio37
        if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL)
        {
                printf("Cannot open export file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write the value of our GPIO port to the file.
        sprintf(&set_value[0], "%d", gpioPort);
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);

        printf("...export file accessed, new pin now accessible\n");

        //SET DIRECTION
        //Open the LED's sysfs file in binary for reading and writing, store file pointer in fp
        // Strat by creating a string representing the port that needs to be opened.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/direction");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open direction file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write our value of "in" to the file
        strcpy(set_value,"in");
        fwrite(&set_value, sizeof(char), 2, fp);
        fclose(fp);
        printf("...direction set to input\n");
}


/****************************************************************************
 * This method will read the given pin and perform processing on it.
 *
 * @param FILE *fp - This is a pointer to the io device driver which will be read to determine
the state of the pin.
 ****************************************************************************/
void processPin(FILE *ifp, FILE *ofp)
{
```

```
	char buffer[10];
	//create a variable to store whether we are sending a '1' or a '0'
	char set_value[5];
	char prevState;

	prevState = '?';

	while (1==1)
	{
		fflush(ifp);

		//Set pointer to begining of the file
		rewind(ifp);

		// Read in the state of the pin.
		fread(&buffer[0], 1, 1, ifp);

		if (buffer[0]!=prevState)
		{
			// The state is different.
			// Print out the appropriate message.

			//Set pointer to begining of the file
			rewind(ofp);


			if (buffer[0]=='0')
			{
				//printf("The button is pressed.\n");

				//Write our value of "1" to the file
				strcpy(set_value,"0");
			}
			else
			{
				//printf("The button is not pressed.\n");

				//Write our value of "1" to the file
				strcpy(set_value,"1");
			}

			fwrite(&set_value, sizeof(char), 1, ofp);
			fflush(ofp);

			prevState = buffer[0];
			usleep(10000);
		}
	}
}

/**
 * This is the main method for this program.  It will cause the program to read the state of the
IO pin and write it out to the console.
 */
int main(int argc, char** argv)
{
	/* This variable is a file pointer to the file interface for the GPIO device driver
	   which controls the input pin. */
	FILE *fp;
	FILE *ofp;
	FILE *ifp;
	uint8_t gpioInputPort;    // This is the GPIO port that is to be used for input into this
program.
	uint8_t gpioOutputPort;       //   This is the GPIO port that is to be used for output
from this program.

	if (argc < 3) {
		printf("Usage: %s <input port number> <output port number>\n\n", argv[0]);
```

```
                printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
                exit(-1);
        }

        // Convert the input into the appropriate parameters.
        gpioInputPort = atoi(argv[1]);
        gpioOutputPort = atoi(argv[2]);

        printf("\n**********************************\n"
                "*  Welcome to LED follower program Demo program. *\n"
                "*  ....Reading gpio %d Controlling GPIO %d       *\n"
                "**********************************\n", gpioInputPort, gpioOutputPort);
        // Setup the IO ports.
        setupInputPort(gpioInputPort);
        setupOutputPort(gpioOutputPort);

        // Open the device driver for processing.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioInputPort, "/value");

        if ((ifp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open value file.\n");
                exit(1);
        }

        // Open the device driver for processing.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioOutputPort, "/value");

        if ((ofp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open value file.\n");
                exit(1);
        }


        processPin(ifp, ofp);

        // Shutdown everything.
        fclose(ifp);
        fclose(ofp);
        return 0;
}
```

13. Appendix D: Interrupt Controlled LED

```
/*****************************************************************
 * This program will allow on to turn a given port on and off at a user
 * defined rate.  The code was initially developed by Dingo_aus, 7 January 2009
 * email: dingo_aus [at] internode <dot> on /dot/ net
 * From http://www.avrfreaks.net/wiki/index.php/Documentation:LinuxGPIO#gpio_framework
 *
 * Created in AVR32 Studio (version 2.0.2) running on Ubuntu 8.04
 * Modified by Mark A. Yoder, 21-July-2011
 * Refactored and further modified by Walter Schilling, Summer 2012 / Winter 2013-2014.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>
#include <signal.h>    // Defines signal-handling functions (i.e. trap Ctrl-C)
#include <stdint.h>

#define SYSFS_GPIO_DIR "/sys/class/gpio"
#define SYSFS_GPIO_DIR "/sys/class/gpio"
#define POLL_TIMEOUT (3 * 1000) /* 3 seconds */
#define MAX_BUF 64

// create a variable to keep track of the port that is to be opened as a string.
static char ioPort[56];

static int keepgoing = 1;     // Set to 0 when ctrl-c is pressed

/*****************************************************************************
 * This method will setup the GPIO port to allow the user to perform output on the given port.
 *
 * @param uint32_t gpioPort - This is the number of the GPIO port that is to be opened.
 *
 *****************************************************************************/
void setupOutputPort(uint32_t gpioPort)
{
        FILE* fp;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Using sysfs we need to write the 3 digit gpio number to /sys/class/gpio/export
        //This will create the folder /sys/class/gpio/gpio37
        if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL)
        {
                printf("Cannot open export file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write the value of our GPIO port to the file.
        sprintf(&set_value[0], "%d", gpioPort);
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);

        printf("...export file accessed, new pin now accessible\n");

        //SET DIRECTION
        //Open the LED's sysfs file in binary for reading and writing, store file pointer in fp
        // Strat by creating a string representing the port that needs to be opened.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/direction");
```

```
        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open direction file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write our value of "out" to the file
        strcpy(set_value,"out");
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);
        printf("...direction set to output\n");
}

/****************************************************************************
 * This method will setup the GPIO port to allow the user to perform input on the given port.
 *
 * @param uint32_t gpioPort - This is the number of the GPIO port that is to be opened.
 *
 ****************************************************************************/
void setupInputPort(uint32_t gpioPort)
{
        FILE* fp;
        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];

        //Using sysfs we need to write the 3 digit gpio number to /sys/class/gpio/export
        //This will create the folder /sys/class/gpio/gpio37
        if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL)
        {
                printf("Cannot open export file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write the value of our GPIO port to the file.
        sprintf(&set_value[0], "%d", gpioPort);
        fwrite(&set_value, sizeof(char), 3, fp);
        fclose(fp);

        printf("...export file accessed, new pin now accessible\n");

        //SET DIRECTION
        //Open the LED's sysfs file in binary for reading and writing, store file pointer in fp
        // Strat by creating a string representing the port that needs to be opened.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioPort, "/direction");

        if ((fp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open direction file.\n");
                exit(1);
        }

        //Set pointer to begining of the file
        rewind(fp);

        //Write our value of "in" to the file
        strcpy(set_value,"in");
        fwrite(&set_value, sizeof(char), 2, fp);
        fclose(fp);
        printf("...direction set to input\n");
}
```

```
/*****************************************************************************
 * This method will read the given pin and perform processing on it.
 *
 * @param FILE *fp - This is a pointer to the io device driver which will be read to determine
the state of the pin.
 *****************************************************************************/
int processPin(int gpio_fd, FILE *ofp, uint32_t gpioInputPort)
{
        struct pollfd fdset[2];
        int timeout, rc;
        int nfds = 2;
        char buf[MAX_BUF];
        int len;

        //create a variable to store whether we are sending a '1' or a '0'
        char set_value[5];
        char prevState;

        prevState = '?';
        timeout = POLL_TIMEOUT;
        printf("AAAA");


        while (keepgoing) {
                memset((void*)fdset, 0, sizeof(fdset));

                fdset[0].fd = STDIN_FILENO;
                fdset[0].events = POLLIN;

                fdset[1].fd = gpio_fd;
                fdset[1].events = POLLPRI;

                rc = poll(fdset, nfds, timeout);

                if (rc < 0) {
                        printf("\npoll() failed!\n");
                        return -1;
                }

                if (rc == 0) {
                        printf(".");
                }

                if (fdset[1].revents & POLLPRI) {
                        lseek(fdset[1].fd, 0, SEEK_SET);  // Read from the start of the file
                        len = read(fdset[1].fd, buf, MAX_BUF);
                        printf("\npoll() GPIO %d interrupt occurred, value=%c, len=%d\n",
                                 gpioInputPort, buf[0], len);


                        if (buf[0]!=prevState)
                        {
                                if (buf[0]=='0')
                                {
                                        //printf("The button is pressed.\n");

                                        //Write our value of "1" to the file
                                        strcpy(set_value,"0");
                                }
                                else
                                {
                                        //printf("The button is not pressed.\n");

                                        //Write our value of "1" to the file
                                        strcpy(set_value,"1");
                                }
```

```
                                fwrite(&set_value, sizeof(char), 1, ofp);
                                fflush(ofp);

                                prevState = buf[0];
                        }
                }
                if (fdset[0].revents & POLLIN) {
                        (void)read(fdset[0].fd, buf, 1);
                        printf("\npoll() stdin read 0x%2.2X\n", (unsigned int) buf[0]);
                }

                fflush(stdout);
        }
}

/****************************************************************
 * signal_handler
 ****************************************************************/
// Callback called when SIGINT is sent to the process (Ctrl-C)
void signal_handler(int sig)
{
        printf( "Ctrl-C pressed, cleaning up and exiting..\n" );
        keepgoing = 0;
}

/****************************************************************
 * gpio_export
 ****************************************************************/
int gpio_export(unsigned int gpio)
{
        int fd, len;
        char buf[MAX_BUF];

        fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
        if (fd < 0) {
                perror("gpio/export");
                return fd;
        }

        len = snprintf(buf, sizeof(buf), "%d", gpio);
        write(fd, buf, len);
        close(fd);

        return 0;
}

/****************************************************************
 * gpio_unexport
 ****************************************************************/
int gpio_unexport(unsigned int gpio)
{
        int fd, len;
        char buf[MAX_BUF];

        fd = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
        if (fd < 0) {
                perror("gpio/export");
                return fd;
        }

        len = snprintf(buf, sizeof(buf), "%d", gpio);
        write(fd, buf, len);
        close(fd);
        return 0;
}

/****************************************************************
 * gpio_set_dir
```

```
 ***************************************************************/
int gpio_set_dir(unsigned int gpio, unsigned int out_flag)
{
        int fd, len;
        char buf[MAX_BUF];

        len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR  "/gpio%d/direction", gpio);

        fd = open(buf, O_WRONLY);
        if (fd < 0) {
                perror("gpio/direction");
                return fd;
        }

        if (out_flag)
                write(fd, "out", 4);
        else
                write(fd, "in", 3);

        close(fd);
        return 0;
}

/***************************************************************
 * gpio_set_value
 ***************************************************************/
int gpio_set_value(unsigned int gpio, unsigned int value)
{
        int fd, len;
        char buf[MAX_BUF];

        len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

        fd = open(buf, O_WRONLY);
        if (fd < 0) {
                perror("gpio/set-value");
                return fd;
        }

        if (value)
                write(fd, "1", 2);
        else
                write(fd, "0", 2);

        close(fd);
        return 0;
}

/***************************************************************
 * gpio_get_value
 ***************************************************************/
int gpio_get_value(unsigned int gpio, unsigned int *value)
{
        int fd, len;
        char buf[MAX_BUF];
        char ch;

        len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

        fd = open(buf, O_RDONLY);
        if (fd < 0) {
                perror("gpio/get-value");
                return fd;
        }

        read(fd, &ch, 1);

        if (ch != '0') {
```

```
                *value = 1;
        } else {
                *value = 0;
        }

        close(fd);
        return 0;
}


/****************************************************************
 * gpio_set_edge
 ****************************************************************/

int gpio_set_edge(unsigned int gpio, char *edge)
{
        int fd, len;
        char buf[MAX_BUF];

        len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/edge", gpio);

        fd = open(buf, O_WRONLY);
        if (fd < 0) {
                perror("gpio/set-edge");
                return fd;
        }

        write(fd, edge, strlen(edge) + 1);
        close(fd);
        return 0;
}

/****************************************************************
 * gpio_fd_open
 ****************************************************************/

int gpio_fd_open(unsigned int gpio)
{
        int fd, len;
        char buf[MAX_BUF];

        len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

        fd = open(buf, O_RDONLY | O_NONBLOCK );
        if (fd < 0) {
                perror("gpio/fd_open");
        }
        return fd;
}

/****************************************************************
 * gpio_fd_close
 ****************************************************************/

int gpio_fd_close(int fd)
{
        return close(fd);
}

/****************************************************************
 * Main
 ****************************************************************/
int main(int argc, char **argv, char **envp)
{
        /* This variable is a file pointer to the file interface for the GPIO device driver
           which controls the input pin. */
        FILE *fp;
        FILE *ofp;
```

```
        FILE *ifp;
        uint8_t gpioInputPort;   // This is the GPIO port that is to be used for input into this
program.
        uint8_t gpioOutputPort;       //  This is the GPIO port that is to be used for output
from this program.

        int gpio_fd; //, rc;

        if (argc < 3) {
                printf("Usage: %s <input port number> <output port number>\n\n", argv[0]);
                printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
                exit(-1);
        }


        // Set the signal callback for Ctrl-C
        signal(SIGINT, signal_handler);


        // Convert the input into the appropriate parameters.
         gpioInputPort = atoi(argv[1]);
         gpioOutputPort = atoi(argv[2]);

        printf("\n**********************************\n"
                "*  Welcome to LED follower program Demo program. *\n"
                "*  ....Reading gpio %d Controlling GPIO %d       *\n"
                "**********************************\n", gpioInputPort, gpioOutputPort);

        gpio_export(gpioInputPort);
        gpio_set_dir(gpioInputPort, 0);
        gpio_set_edge(gpioInputPort, "both");  // Can be rising, falling or both
        gpio_fd = gpio_fd_open(gpioInputPort);

        // Setup the IO ports.
        setupOutputPort(gpioOutputPort);

        // Open the device driver for processing.
        sprintf(&ioPort[0], "%s%s%d%s", SYSFS_GPIO_DIR, "/gpio", gpioOutputPort, "/value");

        if ((ofp = fopen(&ioPort[0], "rb+")) == NULL)
        {
                printf("Cannot open value file.\n");
                exit(1);
        }

        processPin(gpio_fd, ofp, gpioInputPort);

//***************************************************************************



        gpio_fd_close(gpio_fd);
        fclose(ofp);

        return 0;
}
```