Lab 1 Report

Jinzhi Cai Yifei Che, Ryan Anthony Kane

February 5, 2020

Contents

1 Introduction 4

2 Download, Build an Image, and Boot the Beaglebone from an SD Card 5
2.1 Setup . . . 5
2.2 Procedure . . . 5
2.3 Result . . . 5

3 Copy the image to the eMMC and Boot the Beaglebone from the eMMC 6
3.1 Setup . . . 6
3.2 Procedure . . . 6
3.3 Result . . . 6

4 GPIO with the Beaglebone using the command line interface 7
4.1 Setup . . . 7
4.2 Procedure . . . 7
4.3 Result . . . 7

5 GPIO with the Beaglebone using C++ code 8
5.1 Setup . . . 8
5.2 Procedure . . . 8
5.3 Result . . . 8

6 Test the speed of GPIO in Linux c++ using the flip-a-bit program 9
6.1 Setup . . . 9
6.2 Procedure . . . 9
6.3 Result . . . 9

7 PWM with the Beaglebone using the command line interface 10
7.1 Setup . . . 10
7.2 Procedure . . . 10
7.3 Result . . . 10

8 PWM with the Beaglebone using c++ code 11
8.1 Setup . . . 11
8.2 Procedure . . . 11
8.3 Result . . . 11

9 ADC (Analog In) with the Beaglebone using the command line interface 13
9.1 Setup . . . 13
9.2 Procedure . . . 13
9.3 Result . . . 13

10 ADC (Analog In) with the Beaglebone using c++ code 14
10.1 Setup . . . 14
10.2 Procedure . . . 14
10.3 Result . . . 14

11 Embedded System 16
11.1 Setup . . . 16
11.2 Procedure . . . 16
11.3 Result . . . 16

12 Connection to iot_lab network **17**
12.1 Setup 17
12.2 Procedure 17
12.3 Result 17

13 Sending Data Over a Socket **18**
13.1 Setup 18
13.2 Procedure 18
13.3 Result 18

14 Conclusion **19**

15 Appendix A: C++ Gpio Class Code **19**

1 Introduction

Before beginning with the following experiments, we will first explore what board we will be using, the Beaglebone Green Wireless. A Beaglebone is a small, low-power open source computer capable of running different Linux flavors. When it comes to hardware, the Beaglebone Green Wireless has six main components shown in the table below.

Component	Functionality
Sitara AM3358BZCZ100	The main processor
Micron 512MB DDR3L or Kingston 512mB DDR3	The Dual Data Rate RAM memory
TPS65217C PMIC	Distributes the board power rails to various components on the board
SMSC Ethernet PHY	Physical network interface
Micron eMMC	Onboard MMC chip that holds up to 4GB of storage
Wilink 8 Module	Adds Bluetooth and WLAN support on the 2.4 GHz range

Table 1: Summary of Revision History

Now that we are familiar with the hardware that the board offers, we can continue to the remaining experiments.

2 Download, Build an Image, and Boot the Beaglebone from an SD Card

2.1 Setup

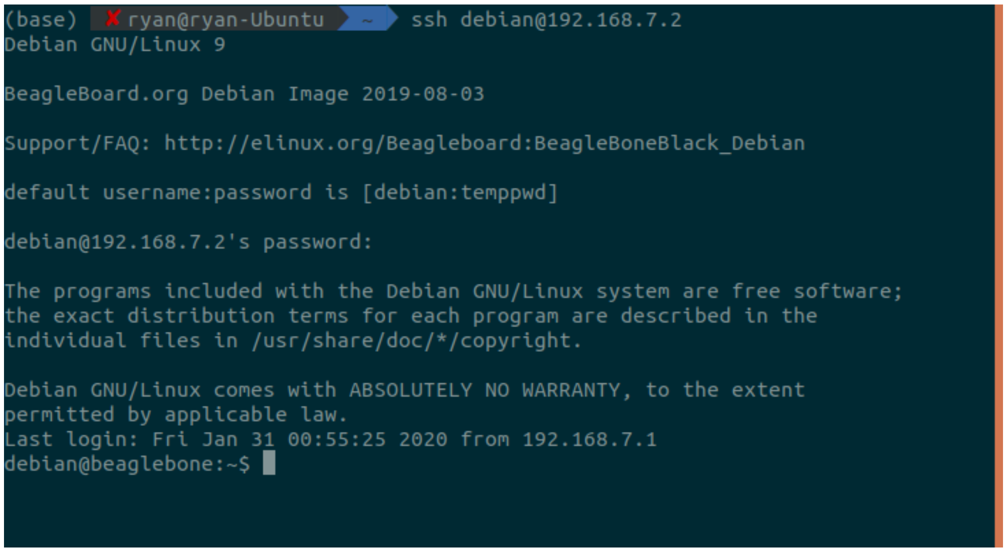
In the remainder of this lab, debian will be used on the Beaglebone. In order to run the latest debian image on the board, an SD card must be inserted into the board with the system image on it. The only setup for this experiment is having an SD card with a minimum of 4GB of storage on it.

2.2 Procedure

First, the latest system image can be obtained at https://beagleboard.org/latest-images. Here the system image downloaded is the "Debian X.X 4GB SD IoT". While the LXQT images can also be installed, the beaglebone green wireless cannot take advantage of their GUI because it does not have an HDMI connection. After the image is downloaded, it has to be extracted from an .img.xz file to a .img file. After this is done, the image is then flashed to the SD card using Etcher. In Etcher select the SD Card, press flash and wait for the process to finish. Now, make sure the board is powered down by unplugging it from the computer if it is plugged in. Then insert the SD card and plug it back into the computer while holding the user button until it is booted successfully.

2.3 Result

After the board is booted properly, it can be accessed via SSH from a terminal on the connected computer via USB. This can be done by running "ssh debian@192.168.6.2" or "ssh debian@192.168.7.2" depending on the system. If done properly, you will be prompted for a password, which by default is "temppwd". This means the board is successfully booted off the SD card and we are ready to continue to the next experiment.



```
(base)  ✗ ryan@ryan-Ubuntu  ~  ssh debian@192.168.7.2
Debian GNU/Linux 9

BeagleBoard.org Debian Image 2019-08-03

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

debian@192.168.7.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jan 31 00:55:25 2020 from 192.168.7.1
debian@beaglebone:~$
```

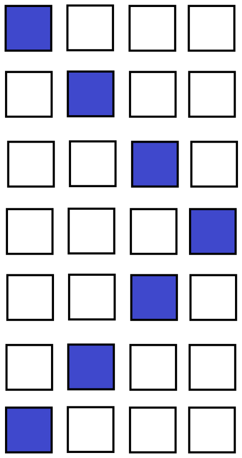
3 Copy the image to the eMMC and Boot the Beaglebone from the eMMC

3.1 Setup

In this experiment, the system image on the SD card previously booted off of by the board will be flashed to the onboard eMMC so the SD card is no longer needed by the board. This can be avoided if you are fine with keeping that SD card in the Beaglebone permanently. No setup is required for this step.

3.2 Procedure

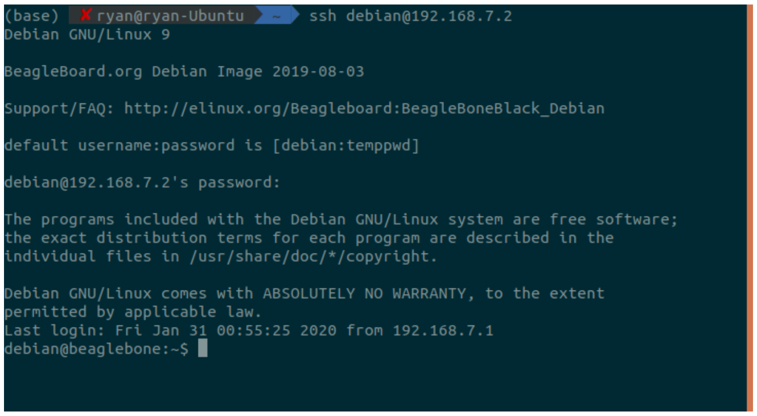
The first step to flash the system image to the eMMC is to SSH into the board while it is booted off of the SD card. Then, open the file uENV.txt in the /boot directory. Inside of the file, uncomment the line "cmdline=init=/opt/scripts/tools/eMMC-flasher-v3.sh". This causes the SD card image to flash to the eMMC. Now, power down the board and reboot it while holding the user button until the LED sequence near the USB port lights up in such a way repetitively.



Once the process is finished, the board will power down by itself. Finally, disconnect the board from the computer, remove the SD card and reconnect.

3.3 Result

After the board is rebooted, it can be accessed again via SSH from the computer terminal as done in the previous experiment. Congratulations the board now boots off of the onboard eMMC image!

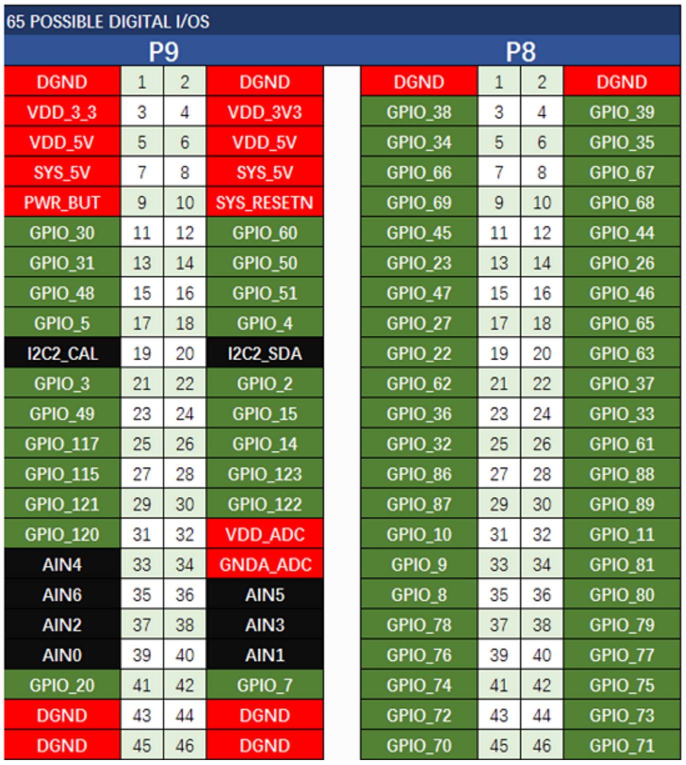


```
(base)  ✗ ryan@ryan-Ubuntu  ~  ssh debian@192.168.7.2
Debian GNU/Linux 9

BeagleBoard.org Debian Image 2019-08-03

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

debian@192.168.7.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jan 31 00:55:25 2020 from 192.168.7.1
debian@beaglebone:~$
```

4 GPIO with the Beaglebone using the command line interface

4.1 Setup

Now, in this experiment the Beaglebone GPIO ports will be controlled and tested using the terminal interface over SSH. To set up this experiment, the Analog Discovery was connected to a GPIO pin on the beaglebone. The pin diagram for the Beaglebone green wireless is shown below. Pin 0 of the Analog Discovery is connected to GPIO pin 48 of the Beaglebone and pin 1 on the Analog Discovery is connected to GPIO pin 49. Open staticIO in Waveforms so that the pin values can be seen.



65 POSSIBLE DIGITAL I/OS

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3_3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_50	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_5	17	18	GPIO_4	GPIO_27	17	18	GPIO_65
I2C2_CAL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	GPIO_123	GPIO_86	27	28	GPIO_88
GPIO_121	29	30	GPIO_122	GPIO_87	29	30	GPIO_89
GPIO_120	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

4.2 Procedure

After SSHing into the Beaglebone, navigate to the /sys/class/gpio directory. Here the possible GPIO ports are shown as directories. To make sure pin 48 and 49 are available, run "export gpio48" and "export gpio49" in terminal. Now, navigate into the gpio48 directory. Inside of this directory there are two files that control the GPIO port. First, the direction of the pin can be set by writing in or out into the "direction" file. To set it as an output, execute "echo out > direction". Now, running "echo 1 > value" sets the gpio pin to high, and 0 will set it to low. To read the value being sent by the Analog Discovery on pin 49, navigate up a directory level and into the gpio49 directory. Here, to set it as an input execute "echo in > direction" in terminal. After that, the value can be read using "cat value".

4.3 Result

When gpio48/value is set to 1, the Analog Discovery pin 0 in staticIO reflects it, showing high, and vice versa for 0. Setting pin 1 on the Analog Discovery to low cause "cat value" in the gpio49 directory to be 0 and vice versa for high. This means we can correctly manipulate and read the Beaglebone GPIOs from the command line

5 GPIO with the Beaglebone using C++ code

5.1 Setup

Since we don't always want to navigate to, read, and set pin values from the command line, C++ code will be developed to do this for us. This will let us programmatically operate the GPIO pins easier. The only setup required for this experiment is to connect two more GPIO pins to the analog discovery, and to make sure g++ is installed on the Beaglebone so the C++ code can be compiled into an executable.

5.2 Procedure

Writing this C++ code comes down to using file I/O to control the files used in the previous experiment. For this, fstreams are used to write and read the value and direction files. The full C++ class code can be found in Appendix A. After creating the C++ class it is implemented in the main.cpp file in order to setup two pins and inputs, and two as outputs. The input pins read the values sent from the Analog Discovery, and the values are reflected on the two pins set as outputs. The main.cpp file is shown below.

```
#include "gpioHandle.h"
#include <iostream>
#include <cstdlib>
#include <unistd.h>


int main(){
        gpioHandle gpio;
        int res;
        res = gpio.initializePin(60,0,0);
        res = gpio.initializePin(117,0,0);
        res = gpio.initializePin(48,1,0);
        res = gpio.initializePin(49,1,0);

        int value;
        while(1) {
                value = gpio.getPinVal(48);
                gpio.setPinVal(60, value);
                value = gpio.getPinVal(49);
                gpio.setPinVal(117, value);
        }
}
```

5.3 Result

After the executable is run, the values set on the Analog Discovery are correctly reflected on the output pins of the board. This can be verified using staticIO in Waveforms.

6 Test the speed of GPIO in Linux c++ using the flip-a-bit program

6.1 Setup

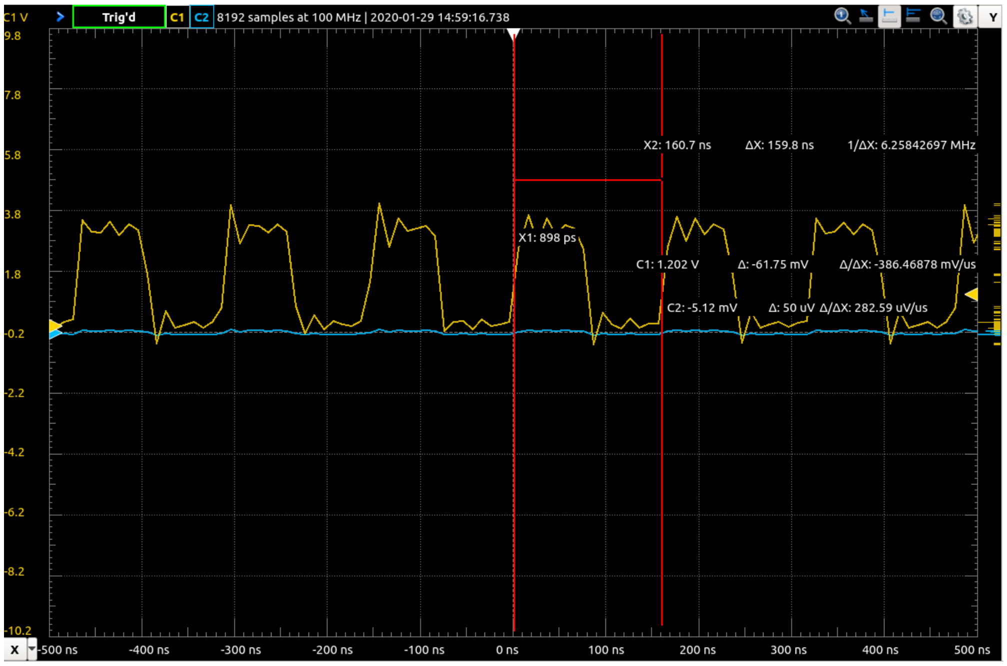
We used the C++ code implemented for GPIO support in the previous section and add the feature to control the behavior of flip-a-bit program, then we used the analog discovery to monitor the outcome and frequency of the flip-a-bit.

6.2 Procedure

We write a main program using our previously written GPIO class that runs a loop setting a GPIO output pin high and then low as fast as possible. Inside the code, we have the bit flipping between 0 and 1 in an infinite while loop. The program results in the Logic to produce a square wave at the output of the selected GPIO pin.

6.3 Result

In the analog discovery logic, we observe that the frequency of flip-a-bit is 5kHz. The reason why we get the 5kHz frequency instead of 1Ghz is that the time interval corresponds to how quickly the kernel of the OS is updated.



7 PWM with the Beaglebone using the command line interface

7.1 Setup

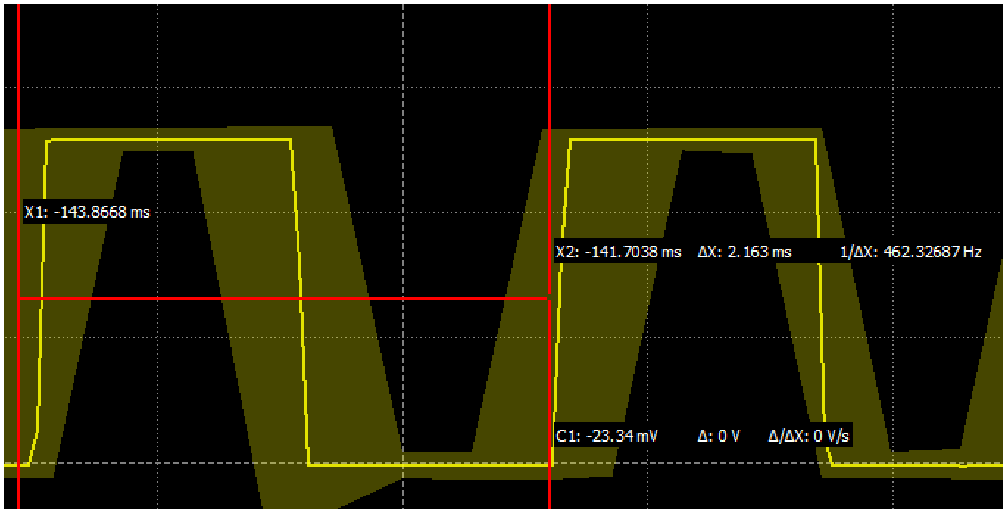
Beaglebone has several PWM pins built inside the hardware, and we can connect the PWM pins. As soon as we finish connecting PWN pins, in order to control the PWM signals, we can use the command line interface to control the behavior. Later, we need to use type in command in the terminal to set the duty cycles, frequency, and period to control the value of PWMs. In the end, we need to use analog discovery to observe the duty cycle of the PWM.

7.2 Procedure

Firstly, we look at the Beaglebone Pinout Diagram to find the PWM pins. As the diagram indicates, each three PWMs are grouped as one, and two of them are a pair. We determine that we are going to use P9_42 as our PWM pin. It's located in the folder "/sys/class/pwm/pwm-1:0". As we are in this directory, we type in the command "config-pin -q P9_42" and "Config-pin -q P9_42 pwm", now, this pin is set up as a PWM pin. Then we can manually set the value of duty cycle, frequency, and period by echo values to those variables. The corresponding duty cycle is shown in the analog discovery.

7.3 Result

After we set up the pin P9_42 as a PWM pin, we manually set the duty cycle to 1000, 10000 to the period, now the duty cycle is 10% since 1000/10000 is ten percent. About ten percent of each period is high, which shows the duty cycle of PWM observed in analog discovery.



8 PWM with the Beaglebone using c++ code

8.1 Setup

Beaglebone has several PWM pins built inside the hardware, and we can connect the PWM pins. As soon as we finish connecting PWN pins, in order to control the PWM signals, we need to use the C++ code to control the behavior. As the instruction indicates, we don't have to configure the pins as PWM with code since we assume the pin is already configured as PWM by using the command line in the previous section.

8.2 Procedure

Firstly, we need to construct a C++ code to control the PWM behavior of beaglebone. In the code, we define certain variables such as period, pinMode, etc. Then we determine we are going to use slot 21(1:1), 22(1:0) as our PWMs. In the code, the pwmHandle function contains the directories of these 3 PWMs and enable the pwm as well as assigns value to period for PWMs. pwnHnadle function disables the PWM. setPinVal function will set the value of duty cycle. In the main class, we pick PWM0, and inside an infinite while loop, we can increment the value of duty cycle. As soon as the duty cycle hits 100%, it goes back to 0% since "i%=max".

8.3 Result

After compiling our C++ code, and execute the object file, we can observe that the duty cycle of the PWM is varying from 0% to 100% then back to 0% in the waveform.

```
#ifndef PWM_HANDLE
#define PWM_HANDLE

#include <iostream>
#include <fstream>

using namespace std;

// pinmode definitions
#define PINMODE_OUT      0
#define PINMODE_IN       1
#define period 1000000
#define PWM0 0
#define PWM1 1
#define PWM2 2

class pwmHandle
{
        public:

        pwmHandle(int pin);
        ~pwmHandle(); // unexport all pins opened by this class only
        int setPinVal(int duty_cycle_in);
        // retuns -1 when pin is unusable, else returns 0

        private:
        string pwms[3][2]={{"P9_42","-0\:0"},{"P9_22","-1\:0"},{"P9_21","-1\:1"}};
        string name;
        ofstream duty_cycle;
        ofstream en;
        char *buffer;   // for sprintfs
        string path;
};

#endif

#include "pwmHandler.h"
#include <cstring>
#include <unistd.h>
using namespace std;
pwmHandle::pwmHandle(int pin){
        std::string cmd(string("config-pin ")+pwms[pin][0]+string(" pwm"));
        system(cmd.c_str());
        path="/sys/class/pwm/pwm";
        path+=pwms[pin][1];
        ofstream pinMode;
        pinMode.open(path+"/period");
        pinMode << period << endl;
        pinMode.close();
        en.open(path+"/enable");
        en << "1" << endl;
        en.close();
        cout << path+"/enable" << endl;
}
pwmHandle::~pwmHandle(){
        duty_cycle.open(path+"/duty_cycle");
        en.open(path+"/enable");
        duty_cycle << "0"<< endl;
        en << "0"<< endl;
        duty_cycle.close();
        en.close();
}
int pwmHandle::setPinVal(int duty_cycle_in){
        char duty_cycle_value[21];
        sprintf(duty_cycle_value, "%d", duty_cycle_in);

        duty_cycle.open(path+"/duty_cycle");
        duty_cycle << duty_cycle_value<< endl;
        duty_cycle.close();
}
```

9 ADC (Analog In) with the Beaglebone using the command line interface

9.1 Setup

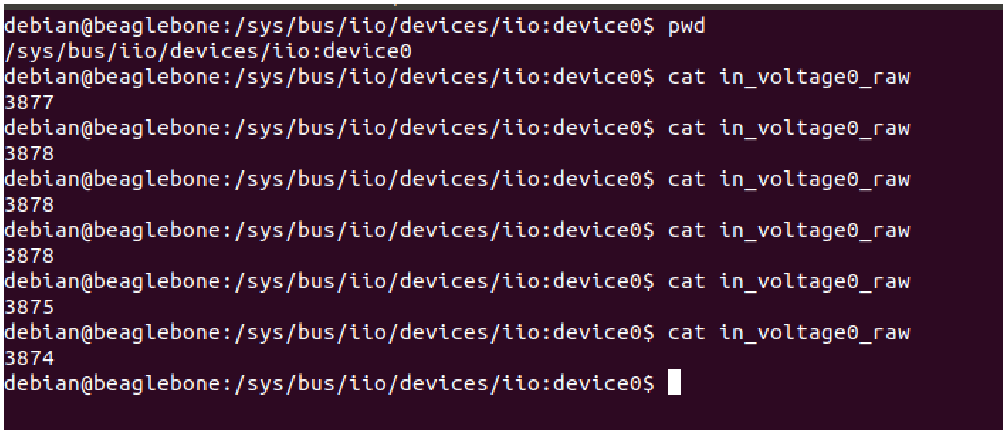
In this section we will use the ADC with BeagleBone to measure the voltage using a potentiometer.The BeagleBone has six ADC pins and six analog input pins. The maximum voltage of ADC pins is 1.8 Volts between GND_ADC and VDD_ADC. We also need a 1 potentiometer we adjust voltage and several cables to connect the hardware.

9.2 Procedure

The potentiometer has 3 terminals. The terminal 1 and 3 connect to the end points of GND_ADC and VDD_ADC. The second terminal is connected to the analog discovery pin. As the scroller scrolls, the resistance values across terminals 1 and 2 and terminals 2 and 3 change. Thus, the potentiometer acts like a voltage divider.

9.3 Result

The value of the potentiometer is printed out in the terminal after we compile and run the file.



```
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ pwd
/sys/bus/iio/devices/iio:device0
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ cat in_voltage0_raw
3877
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ cat in_voltage0_raw
3878
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ cat in_voltage0_raw
3878
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ cat in_voltage0_raw
3878
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ cat in_voltage0_raw
3875
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ cat in_voltage0_raw
3874
debian@beaglebone:/sys/bus/iio/devices/iio:device0$ 
```

10 ADC (Analog In) with the Beaglebone using c++ code

10.1 Setup

In this section we will use the ADC with BeagleBone to measure the voltage using a potentiometer.The BeagleBone has six ADC pins and six analog input pins. The maximum voltage of ADC pins is 1.8 Volts between GND_ADC and VDD_ADC. We also need a 1 potentiometer we adjust voltage and several cables to connect the hardware. Moreover, this time we need to construct a C++ file to read the value from the ADC pin and print the value out in the terminal.

10.2 Procedure

We build our C++ code to take value form the ADC pin and print the value out in the terminal. Basically, we construct the ADC class similar to the PWM class. Now the ADC class contains the functions "adcHandler", " adcHandler", and "getPinVal". adcHandler will get the path of the ADC device. getPinVal will get the value of duty cycle. In the main class, we create an infinite while loop that keeps it running. Then we can observe the ADC voltage value in the terminal after we compile and run the file. By scrolling the spin of the potentiometer, we can see the value varies.

10.3 Result

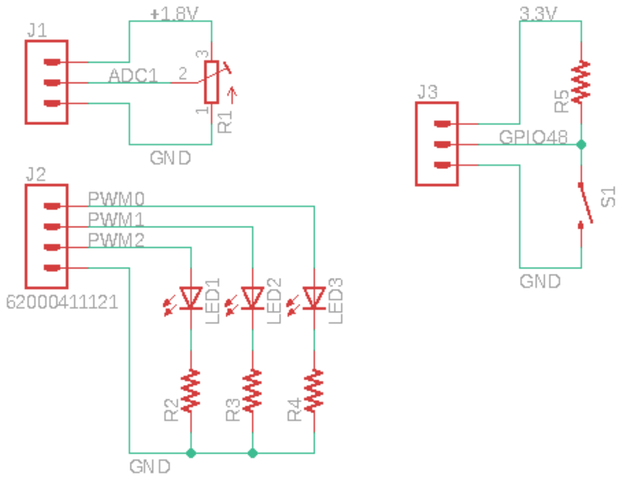
Then the value printed out will be manually countrolled by the spin.

```
#ifndef ADC_HANDLE
#define ADC_HANDLE

#include <iostream>
#include <fstream>

#define ADC_PATH "/sys/bus/iio/devices/iio:device0/in_voltage%d_raw"

using namespace std;
class adcHandle
{
        public:
                adcHandle(int device);
                ~adcHandle(); // unexport all pins opened by this class only
                int getPinVal(); // retuns -1 when pin is unusable, else returns 0

        private:
                string path;
};

#endif

#include "adcHandler.h"
using namespace std;
adcHandle::adcHandle(int device){
        if (device>7)return;
        char char_path[55];
        sprintf(char_path, ADC_PATH, device);
        path=char_path;
}
adcHandle::~adcHandle(){
} // unexport all pins opened by this class only
int adcHandle::getPinVal(){
        ifstream getVal;
        getVal.open(path);
        int result =0;
        getVal >> result;
        getVal.close();
        return result;
} // retuns -1 when pin is unusable, else returns 0
```

11 Embedded System

11.1 Setup

In this section,we use all the class we made in the previous lab to create a embedded system that can react base on the input form the ADC and present the result in the screen and the LED light. The whole lab contain three component. The connection is in following form.

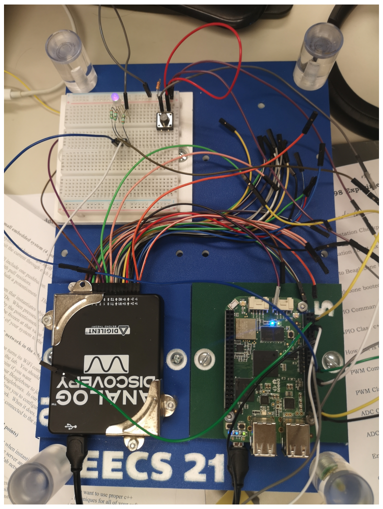


11.2 Procedure

Connecting the circuit base on the image. The program contain three region. The first region is set up. The purpose for this region is to set up all the peripheral require in the program. The second region and the third region all contain in the while loop. The second region is to write the value to the PWM module and the third region is use to read the value from the ADC and save to selected channel.

11.3 Result

Even though the brightness for each channel is different which is due to the characteristic of different color LED, it still will be able to get the different brightness for different channel base on the ADC value.



12 Connection to iot_lab network

12.1 Setup

In this section, we will use the connmanctl to make the beaglebone connect to the iot_lab wifi network. The requirement for this lab is following.

- Beaglebone Green
- Connmanctl
- Working wifi router

12.2 Procedure

- Open connmanctl
  » $ connmanctl
- Scan wifi
  » connman$ scan wifi
- Check services avaliable
  » connman$ services
  lot_lab wifi_dc85de828967_38303944616e69656c73_managed_psk
- Start agent
  » connman$ agent on
- Connect to the WIFI
  » connman$ connect wifi_dc85de828967_38303944616e69656c73_managed_ps
- Key in the password
  Agent RequestInput wifi_dc85de828967_38303944616e69656c73_managed_psk
  Passphrase = [Type=psk, Requirement=mandatory]
  Passphrase?
- Ifconfig check connection
  » $ ifconfig

12.3 Result

After type the ifconfig....

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.2.19.112  netmask 255.255.224.0  broadcast 10.2.31.255
        inet6 fe80::52ee:62b2:c8ce:7be8  prefixlen 64  scopeid 0x20<link>
        ether d4:25:8b:dd:3a:bf  txqueuelen 1000  (Ethernet)
       RX packets 105  bytes 6602
       RX errors 0  dropped 0  overruns 0  frame 0
       TX packets 560  bytes 402
       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

13 Sending Data Over a Socket

13.1 Setup

In this section, we will use the TCP socket to make the beaglebone connect to the laptop via wifi network. The requirement for this lab is following.

- Beaglebone Green
- Working wifi router

13.2 Procedure

The server program contain two parts. The server handler is opening a listen port on the network socket. It also help the main class to transmit and receive the data. The main class is to read in the data that get from the TCP connection and send information back to the client.

```
Server foo(9090);
foo.listen();
std::cout<<"Got \"";
std::cout<<foo.receive(12);
std::cout<<"\" from the client\n";
std::cout<<"Sending \"Hello Client\" to the client\n";
foo.send("Hello Client");
```

13.3 Result

After execute this program in the, client is netcat. Connect to the server and it do reply the the welcome message.

14 Conclusion

Embedded system is also a improtant part in the people's life. By running this lab, we understand the basic structure for a embedded system. The system we design, even it is very simple, but can apply to many different area. As the computer system that the most close the physical world, a lot of the debugging work is not only in the software layer. It create new challenge to the engineers to uncover a system that have problem.

15 Appendix A: C++ Gpio Class Code

```
#include "gpioHandle.h"
#include <cstring>
#include <unistd.h>


gpioHandle::gpioHandle()
{
}


gpioHandle::~gpioHandle()
{
}


//exports pin and sets mode
int gpioHandle::initializePin(const int pin, const int mode, const bool  initialValue) {
        char pinNum[21];
        std::string fordirection("/direction");
        std::string path("/sys/class/gpio/gpio");
        sprintf(pinNum, "%d", pin);
        path += pinNum;
        ofstream exportPin;
        exportPin.open("/sys/class/gpio/export");
        if(!exportPin.is_open()) {
                cout << "Error! exportPin not open. \n";
                return (-1);
        }
        exportPin << pin;
        exportPin.close();


        path += fordirection;
        usleep(100000);
        ofstream pinMode;
        pinMode.open(path);
        if(!pinMode.is_open()) {
                cout << "Error! pinMode not open \n";
                return (-1);
        }
        if (PINMODE_OUT == mode) {
                pinMode << "out";
        } else {
                if (PINMODE_IN == mode) {
                        pinMode << "in";
                } else {
                        cout << "ERROR!!! \n";
                        return (-1);
                }
        }
        pinMode.close();
        usleep(100000);
        // cout << "Pin " << pin << " initialized to mode "; //debug only
        switch(mode){
                case(PINMODE_OUT):
                cout << "OUTPUT"; break;
                case(PINMODE_IN):
                cout << "INPUT"; break;
                default:
                cout << "INVALID PINMODE"; break;
        }
        cout << " with an intial value of " << initialValue << endl;
        return(0);
}


// changes pin mode
int gpioHandle::setPinMode(const int pin, const int mode, const bool initialValue) {
        char pinNum[21];
        std::string fordirection("/direction");
        std::string path("/sys/class/gpio/gpio");
        sprintf(pinNum, "%d", pin);
        path += pinNum;
        path += fordirection;
        ofstream pinMode;
        pinMode.open(path);
        if(!pinMode.is_open()) {
                cout << "Error! \n";
                return (-1);
        }

        if (PINMODE_OUT == mode) {
                pinMode << "out";
        } else {
                if (PINMODE_IN == mode) {
                        pinMode << "in";
                } else {
                        cout << "ERROR!!! \n";
                        return (-1);
                }
        }
        pinMode.close();


        usleep(100000);
        // cout << "Pin " << pin << " set to mode "; //debug only
        switch(mode){
                case(PINMODE_OUT):
                cout << "OUTPUT"; break;
                case(PINMODE_IN):
                cout << "INPUT"; break;
                default:
                cout << "INVALID PINMODE"; break;
        }
        // cout << " with an intial value of " << mode << endl;
        return(0);
}

// retuns -1 when pin is unusable, else returns 0
int gpioHandle::setPinVal(const int pin, const bool val) {
        char pinNum[21];
        std::string forvalue("/value");
        std::string path("/sys/class/gpio/gpio");;
        sprintf(pinNum, "%d", pin);
        path += pinNum;
        path += forvalue;
        fstream setVal;
        setVal.open(path);
        if(!setVal.is_open()) {
                cout << "Error! \n";
                return (-1);
        }


        // cout << "value is: " << (int)val << " Non cast: " << val << endl;
        setVal << val;
        // cout << "Pin " << pin << " set to " << val << endl;// debug only
        setVal.close();
        usleep(100000);
        return(0);
}

// returns -1 when pin is unusable, else retuns value
int gpioHandle::getPinVal(const int pin){
        char pinNum[21];
        std::string forvalue("/value");
        std::string path("/sys/class/gpio/gpio");
        sprintf(pinNum, "%d", pin);
        path += pinNum;
        path += forvalue;
        ifstream getVal;
        getVal.open(path);
        if(!getVal.is_open()) {
                cout << "Error! \n";
                return (-1);
        }
        int value;
        getVal >> value;
        getVal.close();
        usleep(100000);
        return(value);
```

```
}

// returns true if pin is usable
bool gpioHandle::pinOpen(const int pin) {
        char pinNum[21];
        std::string path("/sys/class/gpio/gpio");
        sprintf(pinNum, "%d", pin);
        path += pinNum;
        ofstream usePin;
        usePin.open(path);
        if(usePin.is_open()) {
                cout << "The pin " << pin << " is usable!";
                return (true);
        } else {
                cout << "The pin " << pin << " is unusable!";
                return(false);
        }
        usePin.close();
        usleep(100000);
        // cout << "Pin " << pin << " opened!" << endl; // debug only
}

// unexports pin
int gpioHandle::pinClose(const int pin) {
        char pinNum[21];
        int pinNumber = pin;
        std::string path("/sys/class/gpio/gpio");
        sprintf(pinNum, "%d", pin);
        path += pinNum;
        ofstream unexportPin;
        unexportPin.open("/sys/class/gpio/unexport");
        if(!unexportPin.is_open()) {
                cout << "Error! \n";
                return (-1);
        }
        unexportPin << pinNumber;
        unexportPin.close();
        // cout << "Pin " << pin << " closed!" << endl; // debug only
        usleep(100000);
        return(0);
}
```