**2. Blinky.cpp - D3**

#include "mraa.h"

#include <stdio.h>

int main()

{

mraa\_gpio\_context d\_pin = NULL;

d\_pin = mraa\_gpio\_init(13);

if (d\_pin == NULL) {

fprintf(stderr, "MRAA couldn't initialize GPIO, exiting");

return MRAA\_ERROR\_UNSPECIFIED;

}

if (mraa\_gpio\_dir(d\_pin, MRAA\_GPIO\_OUT) != MRAA\_SUCCESS) {

fprintf(stderr, "Can't set digital pin as output, exiting");

return MRAA\_ERROR\_UNSPECIFIED;

};

for (int i=10;i>0;i--) {

printf("LED OFF\n");

mraa\_gpio\_write(d\_pin, 0);

sleep(1);

printf("LED ON\n");

mraa\_gpio\_write(d\_pin, 1);

sleep(1);

}

return MRAA\_SUCCESS;

}

**3. For only Button - Button.cpp - D4**

#include "grove.h"

#include <stdio.h>

#include <unistd.h>

int main()

{

upm::GroveButton\* button = new upm::GroveButton(4);

int count = 5;

int button\_val=0;

while( count > 0 ) {

button\_val = button->value();

printf ("Program will exit after %d button presses\n", count);

printf ("Button value is: %d\n ", button\_val);

if (button\_val)

count--;

usleep(500000);

}

printf ("Exiting, bye!");

delete button;

}

**For both Button and Light (Button - D2 and Light - D3) Blinky.Cpp**

#include “mraa.h”

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define BUTTON\_PIN 2

#define LED\_PIN 13

int main()

{

mraa.init();

mraa.gpio.context button=mraa.gpio init(BUTTON\_PIN);

mraa\_gpio\_dir(button,MRAA\_GPIO\_IN);

mraa\_gpio\_context led = mraa\_gpio\_init(LED\_PIN);

mraa\_gpio\_dir(led, MRAA\_GPIO\_OUT);

for (;;) {

if (mraa\_gpio\_read((button)) mraa\_gpio\_write(led, 1);

else mraa\_gpio\_write(led, 0);

unsleep(100000);

}

mraa\_gpio\_close(button);

mraa\_gpio\_close(led);

mraa\_deinit();

return 0;

}

**4. Buzzer.cpp - D5**

#include <buzzer.hpp>

#include <stdio.h>

#include <unistd.h>

int main()

{

int chord[] = { DO, RE, MI, FA, SOL, LA, SI, DO, SI };

upm::Buzzer\* sound = new upm::Buzzer(5);

printf("Volume = %f\n", sound->getVolume());

sound->setVolume(0.6);

printf("Volume = %f\n", sound->getVolume());

fflush(stdout);

printf("\nPlaying notes, pausing for 0.1 seconds between notes...\n");

fflush(stdout);

for (int chord\_ind = 0; chord\_ind < 7; chord\_ind++) {

printf(" %d\n", sound->playSound(chord[chord\_ind], 500000) );

usleep(100000);

}

printf("Exiting, bbye!\n");

delete sound;

}

**5. Temperature.cpp - A0**

#include "mraa/aio.h"

#include <math.h>

#include <stdio.h>

#include <unistd.h>

#include "jhd1313m1.h"

#include "grove.h"

int main()

{

mraa\_aio\_context adc\_a0;

uint16\_t adc\_value = 0;

const int B=4275;

const int R0 = 100000;

adc\_a0 = mraa\_aio\_init(0);

if (adc\_a0 == NULL) {

return 1;

}

for (int i=10; i>0;i--) {

adc\_value = mraa\_aio\_read(adc\_a0); //Max value @ 5V = 1024

printf("ADC A0 read value : %d\n", adc\_value);

float R = 1023.0/((float)adc\_value)-1.0;

R = 100000.0\*R;

float temperature=1.0/(log(R/100000.0)/B+1/298.15)-273.15;

printf("Temperature value : %.2f Degree Celsius\n", temperature);

sleep(1);

}

mraa\_aio\_close(adc\_a0);

printf("Exiting .. Bbye!");

return MRAA\_SUCCESS;

}

**6. TouchInterrupt.cpp - D4**

#include "ttp223.h"

#include <stdio.h>

#include <unistd.h>

void touchISR (void\*);

int count = 5;

void touchISR(void\*)

{

count--;

printf("\nHello World from ISR, will exit after %d touch events", count);

fflush(stdout);

}

int main()

{

upm::TTP223\* touch = new upm::TTP223(4);

touch->installISR(mraa::EDGE\_FALLING, &touchISR, NULL);

printf("\nWelcome, waiting for touch event.\nWill exit after 5 events");

fflush(stdout);

while(count>0);

printf("\nExiting .. Bbye!");

delete touch;

}

**7. Light.cpp - A0**

#include "grove.h"

#include <stdio.h>

#include <unistd.h>

int main()

{

upm::GroveLight\* light = new upm::GroveLight(0);

for (int i=20;i>0;i--) {

printf(" Light value is %f which is roughly %d lux \n", light->raw\_value(), light->value());

fflush(stdout);

sleep(1);

}

printf("Exiting .. bbye!");

delete light;

}

**8. Mic.cpp - A0**

#include "mic.h"

#include <stdio.h>

#include <unistd.h>

#include <signal.h>

#include <sys/time.h>

int is\_running = 1;

uint16\_t buffer [128];

upm::Microphone \*mic = NULL;

void sig\_handler(int signo)

{

printf("got signal\n");

if (signo == SIGINT) {

is\_running = 0;

}

}

int main(int argc, char \*\*argv)

{

mic = new upm::Microphone(0);

if (signal(SIGINT, sig\_handler) == SIG\_ERR)

printf("\ncan't catch SIGINT\n");

thresholdContext ctx;

ctx.averageReading = 0;

ctx.runningAverage = 0;

ctx.averagedOver = 2;

while (is\_running) {

int len = mic->getSampledWindow (2, 128, buffer);

if (len) {

int thresh = mic->findThreshold (&ctx, 30, buffer, len);

mic->printGraph(&ctx);

if (thresh) {

// do something ....

}

}

}

printf ("exiting application\n");

delete mic;

return 0;

}

**9. LCD.cpp - last I2C which is close to D5**

#include "jhd1313m1.h"

#include <stdio.h>

#include <unistd.h>

int main(void)

{

upm::Jhd1313m1 \*lcd;

lcd = new upm::Jhd1313m1(0, 0x3E, 0x62);

printf("Display text on LCD\n");

lcd->setCursor(0,0);

lcd->write("Batch 4");

lcd->setCursor(1,2);

lcd->write("welcome !");

printf("Sleeping for 5 seconds\n");

sleep(5);

printf("Starting Color loop...\n");

for (int i = 5; i>0 ;i--){

lcd->setColor(255,220,220);

sleep(1);

lcd->setColor(0,255,0);

sleep(1);

lcd->setColor(0,0,125);

sleep(1);

}

printf("Exiting .. bbye!\n");

delete lcd;

return 0;

}

**10 . RotaryAngle.cpp - A1**

#include "grove.h"

#include <stdio.h>

#include <unistd.h>

int main()

{

upm::GroveRotary\* knob = new upm::GroveRotary(1);

while( 1 ) {

float abs\_value = knob->abs\_value(); // Absolute raw value

float abs\_deg = knob->abs\_deg(); // Absolute degrees

float abs\_rad = knob->abs\_rad(); // Absolute radians

float rel\_value = knob->rel\_value(); // Relative raw value

float rel\_deg = knob->rel\_deg(); // Relative degrees

float rel\_rad = knob->rel\_rad(); // Relative radians

printf("Absolute: %4d raw %5.2f deg = %3.2f rad Relative: %4d raw %5.2f deg %3.2f rad\n",

(int16\_t)abs\_value, abs\_deg, abs\_rad, (int16\_t)rel\_value, rel\_deg, rel\_rad);

sleep(1);

}

delete knob;

}