**CG2271 Real Time Operating Systems**

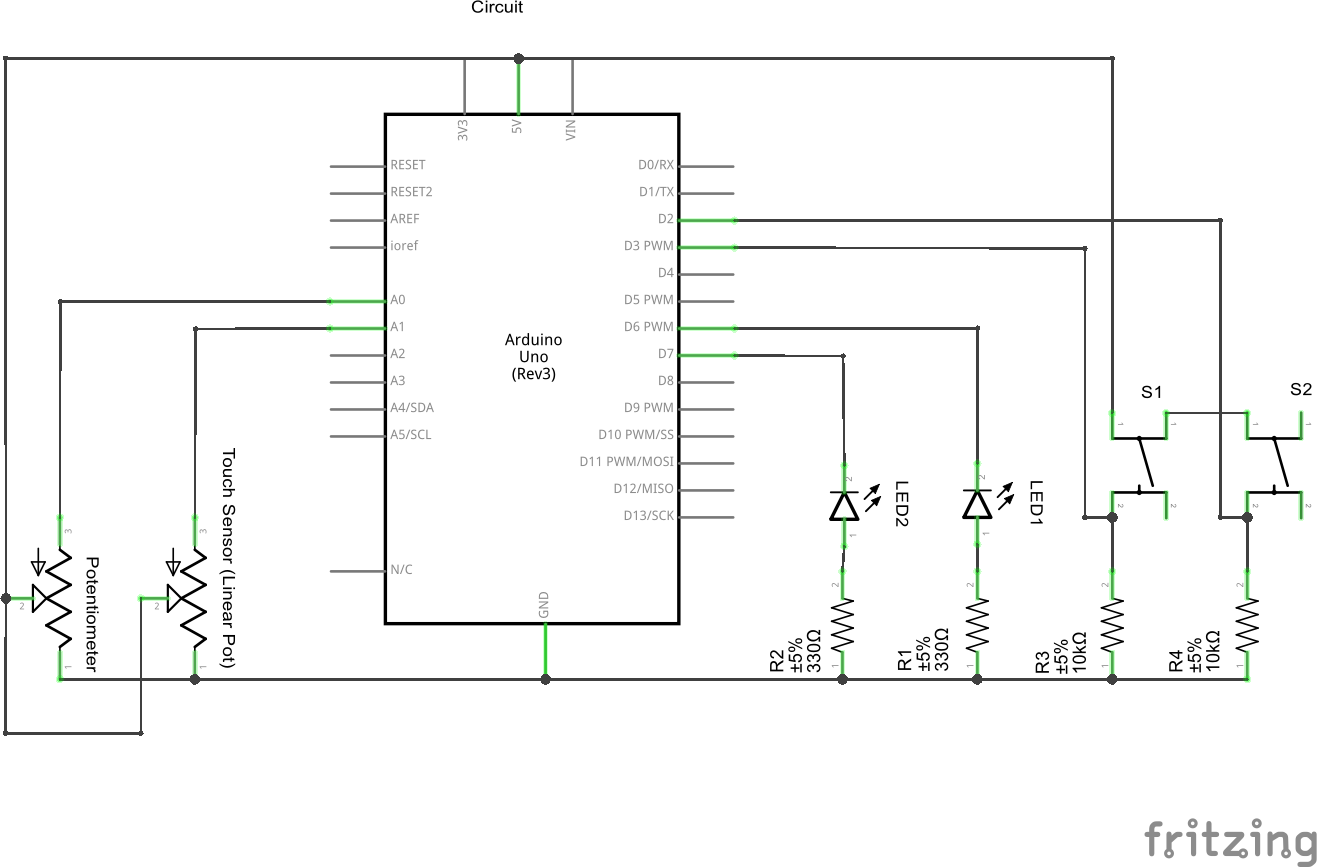
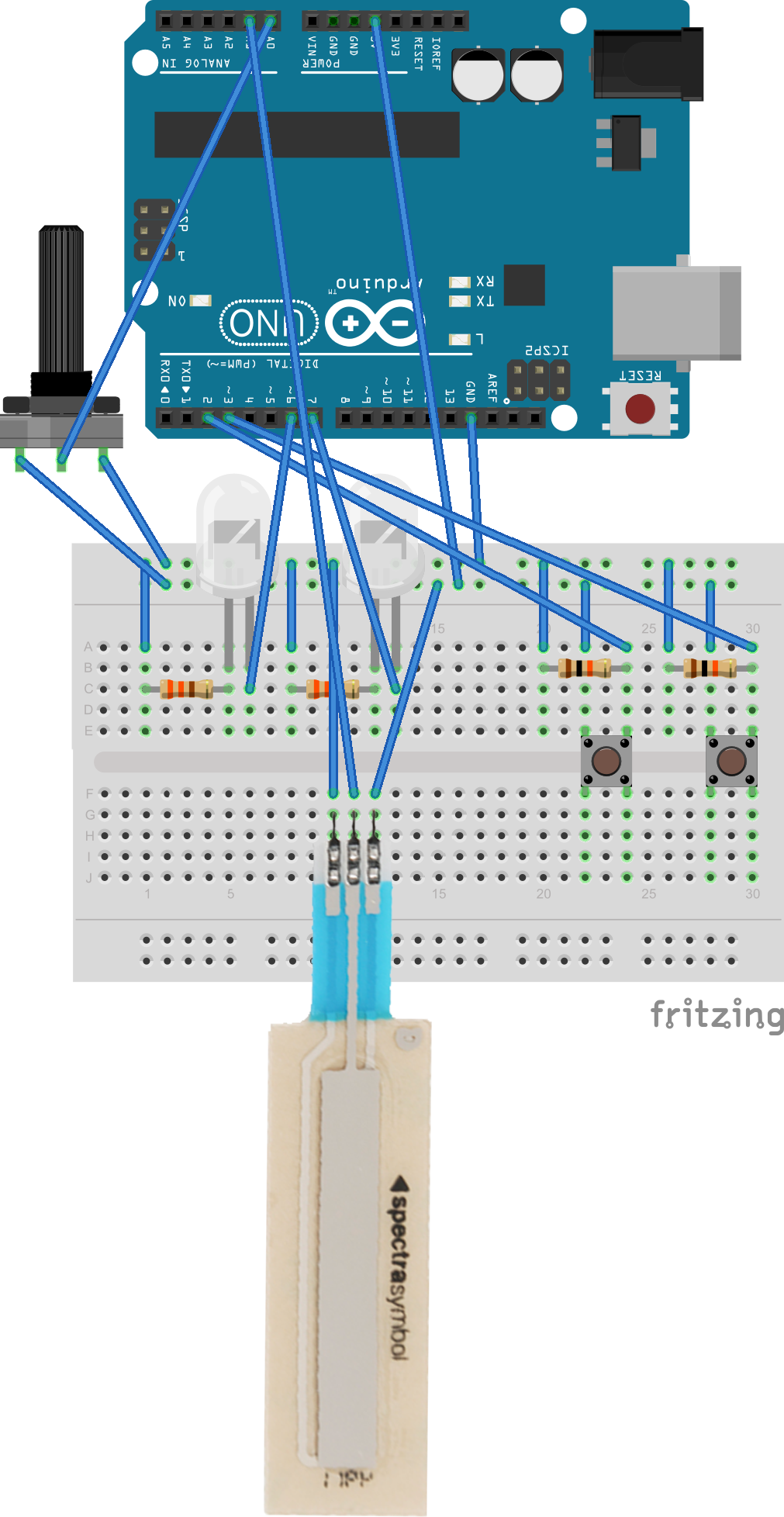
**Lab 2 – Real Time Architectures**

**Answer Book**

|  |  |
| --- | --- |
| **Name: Ang Kah Min, Kelvin** | **Matric No: A0111890** |
| **Name: Sitti Maryam Binte Rashid Ridza** | **Matric No: A0112675** |

Question 1 (3 marks)

Paste a sketch of your circuit below.



Question 2 (3 marks)

This behaviour is caused by the mismatch between the number of bits read by the ADC and the number of bits the DAC accepts.

In binary, when a number goes from 0 to 1023, the lower 8-bits will cycle from 0 to 255 four times as shown:

00**000000** – 0 is interpreted as 0

…

00**111111** – 255 is interpreted as 255

01**000000** – 256 is interpreted as 0

…

01**111111** – 511 is interpreted as 255

10**000000** – 512 is interpreted as 0

…

10**111111** – 767 is interpreted as 255

11**000000** – 768 is interpreted as 0

…

11**111111** – 1023 is interpreted as 255

This results in gradual increase in brightness between 0 to 255, 256 to 511, 512 to 767 and 768 to 1023.

There will be a sudden drop in brightness between 255 and 256, 511 and 512, and 767 and 768.

Therefore there is a need to scale the 10-bits input into an 8-bit output. This is done using the remap(int) function in the next question.

Question 3 (5 marks)

int remap(int val) {

return (int)(val\*255.0/1023.0);

}

Question 4 (3 marks)

Minimum value observed: 0

Maximum value observed: 1023

Question 5 (5 marks)

int remapTouch(int touch) {  
 return 125 + (int)(touch\*375.0/1023.0);  
}

Question 6 (3 marks)

#include <avr/io.h>

#include <Arduino.h>

#define polledPin 2

#define analogOut 6

#define analogChannel 0

void setup() {

pinMode(2, INPUT);

pinMode(7, OUTPUT);

Serial.begin(9600);

}

void flashPin7(int delayVal) {

digitalWrite(7, HIGH);

delay(delayVal);

digitalWrite(7, LOW);

delay(delayVal);

}

int remap(int val) {

return (int)(val\*255.0/1023.0);

}

int remapTouch(int touch) {

return 125 + (int)(touch\*375.0/1023.0);

}

void loop() {

int val = analogRead(0);

int valRemapped = remap(val);

int touch = analogRead(1);

int touchRemapped = remapTouch(touch);

Serial.print(val);

Serial.print(" ");

Serial.print(valRemapped);

Serial.print(" ");

Serial.print(touch);

Serial.print(" ");

Serial.print(touchRemapped);

Serial.println();

analogWrite(6, valRemapped);

flashPin7(touchRemapped);

}

// Note: Do not modify main. It has been written to work correctly with

// the Arduino library. Modify only setup() and loop(), though you may

// add new functions.

int main(void) {

init();

setup();

while(1) {

loop();

if(serialEventRun)

serialEventRun();

}

return 0;

}

The loop function is modified to remap the touch variable from [0 to 1023] to [125 to 500]. The remapped value is then stored in touchRemapped before being passed to the flashPin7(int) function. The function will then use the remapped value as the delay after switching the LED on as well as switching it off. The delay(500) is removed as it will cause an unnecessary delay between each LED blink.

Question 7 (6 marks)

#include <avr/io.h>

#include <Arduino.h>

#define polledPin 2

#define analogOut 6

#define analogChannel 0

static int val = 0;

static int valRemapped = 0;

static int touch = 0;

static int touchRemapped = 125;

void setup() {

pinMode(2, INPUT);

pinMode(3, INPUT);

pinMode(7, OUTPUT);

Serial.begin(9600);

}

void flashPin7(int delayVal) {

digitalWrite(7, HIGH);

delay(delayVal);

digitalWrite(7, LOW);

delay(delayVal);

}

int remap(int val) {

return val \* (int)(255.0 / 1023.0);

}

int remapTouch(int touch) {

return 125 + (int)(touch \* 375.0 / 1023.0);

}

void readPotentiometer() {

val=analogRead(0);

valRemapped = remap(val);

}

void readTouch() {

touch=analogRead(1);

touchRemapped = remapTouch(touch);

}

void outputBrightness() {

analogWrite(6, valRemapped);

}

void outputBlink() {

flashPin7(touchRemapped);

}

void outputSerial() {

Serial.print(val);

Serial.print(" ");

Serial.print(valRemapped);

Serial.print(" ");

Serial.print(touch);

Serial.println();

Serial.print(touchRemapped);

Serial.println();

}

void loop() {

while(1) {

if (digitalRead(2)) {

readPotentiometer();

}

outputBrightness();

if (digitalRead(3)) {

readTouch();

}

outputBlink();

outputSerial();

}

}

int main(void) {

init();

setup();

while(1) {

loop();

if(serialEventRun)

serialEventRun();

}

return 0;

}

The variables val, valRemapped, touch and touchRemapped are moved to file scope to allow different functions to access them. The program now uses digitalRead(int) to determine if the pushbuttons are depressed. The values will be read from the potentiometer and the touch sensor if their respective pushbuttons are depressed at the point where the program tries to read their values.

This is an example of a round-robin architecture because the tasks are run in strict order:

1. Read pushbutton at pin 2.
2. Read and process value from Potentiometer (if pushbutton is depressed).
3. Output processed value to first LED.
4. Read pushbutton at pin 3.
5. Read and process value from Touch Sensor (if pushbutton is depressed).
6. Output the processed value to blink the second LED.
7. Output the new readings to the serial port.

This is the simplest way to implement the microcontroller model:

1. Read Data from Sensor
2. Process Data
3. Output to Actuator

However, there is a limitation in this program. Since the readings are only taken in discrete intervals, it is possible for a button press to be ignored (which is an example of data loss). For example, if a pushbutton is quickly depressed and released within the duration of an LED blink, the program will not register the button press. We will solve the limitations of this program by using interrupts to handle button inputs in the next section.

Question 8 (12 marks)

#include <avr/io.h>

#include <Arduino.h>

#define polledPin 2

#define analogOut 6

#define analogChannel 0

static int val = 0;

static int valRemapped = 0;

static int touch = 0;

static int touchRemapped = 125;

int remap(int val) {

return val \* 255.0 / 1023.0;

}

int remapTouch(int touch) {

int lowestVal = 100;

int highestVal = 200;

return 125 + touch \* 375.0 / 1023.0;

}

void readPotentiometer() {

val=analogRead(0);

valRemapped = remap(val);

}

void readTouch() {

touch=analogRead(1);

touchRemapped = remapTouch(touch);

}

void outputBrightness() {

analogWrite(6, valRemapped);

}

void outputBlink() {

flashPin7(touchRemapped);

}

void outputSerial() {

Serial.print(val);

Serial.print(" ");

Serial.print(valRemapped);

Serial.print(" ");

Serial.print(touch);

Serial.println();

Serial.print(touchRemapped);

Serial.println();

}

void setup() {

attachInterrupt(0, readPotentiometer, RISING);

attachInterrupt(1, readTouch, RISING);

pinMode(2, INPUT);

pinMode(3, INPUT);

pinMode(7, OUTPUT);

Serial.begin(9600);

}

void flashPin7(int delayVal) {

digitalWrite(7, HIGH);

delay(delayVal);

digitalWrite(7, LOW);

delay(delayVal);

}

void loop() {

while(1) {

outputBrightness();

outputBlink();

outputSerial();

}

}

int main(void) {

init();

setup();

while(1) {

loop();

if(serialEventRun)

serialEventRun();

}

return 0;

}

The functions readPotentiometer() and readTouch() are now ISRs and are attached to INT0 and INT1 respectively using the attachInterrupt(…) function in setup(). In this program, the interrupts are triggered on the RISING edge (when the button is released).

The output functions outputBrightness(), outputBlink() and outputSerial() continue to run in a round-robin fashion in loop().

This program overcomes the previous limitations of losing data if the button press duration is too short (as in the previous question). Now, whenever INT0 or INT1 is triggered, the program will pause whatever it is doing and perform the required reading and processing functions.