REPORT

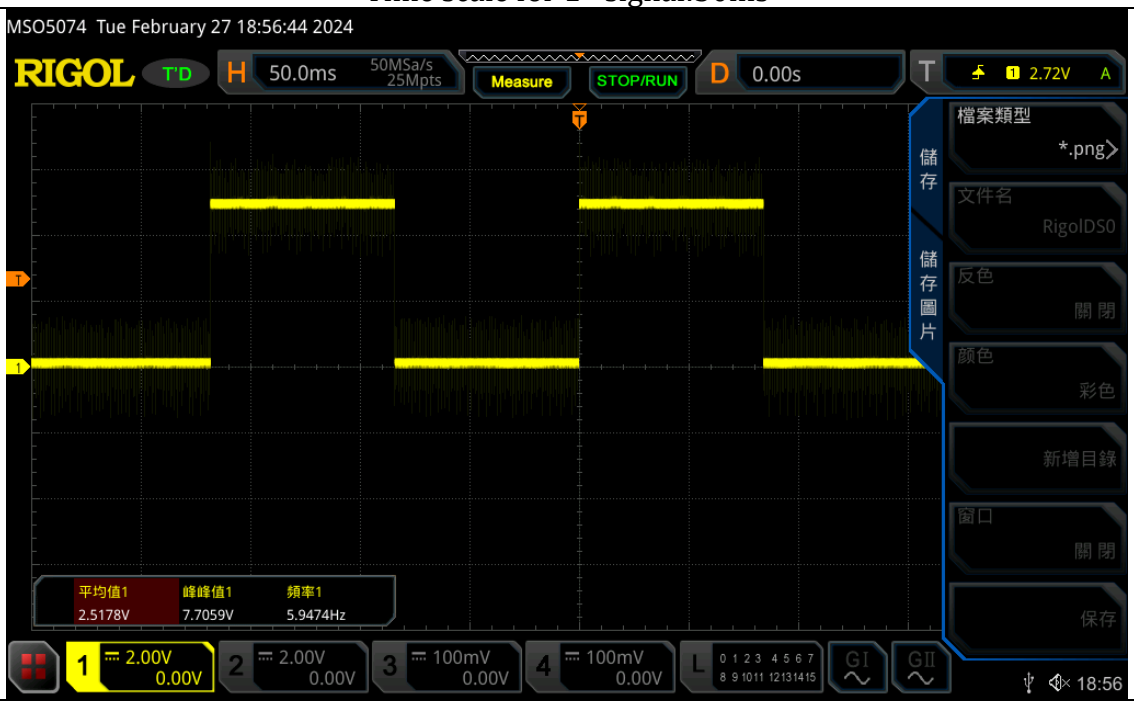
Experiment 1: Multi-task on the Arduino

(Answers depend on your ID)

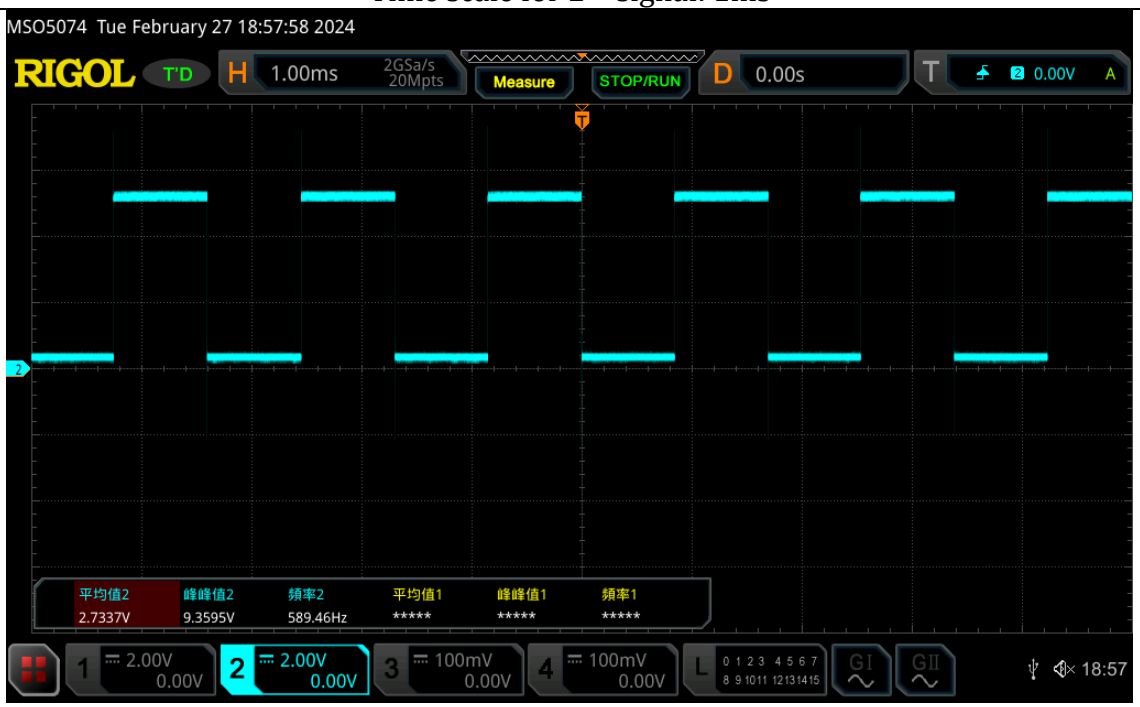
Your 1st signal frequency = 6 Hz
Your 2nd signal frequency = 600 Hz

Waveform of 1st and 2nd signals (It should contain appropriate measurements)

Time scale for 1st signal:50ms



Time scale for 2nd signal: 1ms



(optional) Attach your code here. Ref. a better way to paste your code in Word file)

```
bool Freq6Hz_Out = LOW, Freq600Hz_Out = LOW;
int Freq6Period = 167, Freq600Period = 1667;
unsigned long Prevmillis = 0, Prevmicros = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(millis()-Prevmillis > Freq6Period/2){
    Prevmillis = millis();
    if(Freq6Hz_Out == LOW)Freq6Hz_Out = HIGH;
    else Freq6Hz_Out= LOW;
  }
  if(micros()-Prevmicros > Freq600Period/2){
    Prevmicros = micros();
    if(Freq600Hz_Out == LOW)Freq600Hz_Out = HIGH;
    else Freq600Hz_Out= LOW;
  }
  digitalWrite(2,Freq6Hz_Out);
  digitalWrite(3,Freq600Hz_Out);
}
```

My ID is 111511076 so my 1st signal should be **6Hz (Period:167ms)**, 2nd signal should be **600Hz (Period: 1667us)**.

According to Arduino's official website, "No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function". Therefore, the **"delay()" function is not viable if we want to multi-task**. That is why I used "millis()" and "micros()" here.

millis(): returns how many milliseconds have passed since the Arduino board started running.

micros(): returns how many microseconds have passed since the Arduino board started running.

millis() was enough for the 6Hz signal, but the **600Hz required higher timing accuracy, so I used micros() for it instead.** The variables "Prevmillis" and "Prevmicros" are "unsigned long" because storing time tends to take up more bits. Once the difference between millis() and Prevmillis exceeds half of the period (167ms / 2), the signal is inverted, and Prevmillis is set to the current return value of millis(). The same logic applies to the 600Hz signal except it uses micros(). This generates square waves of 6Hz and 600Hz respectively.

Experiment 2: Button debounce

(Recording all results of your design. Range:0~9. Direction: up and down)

Video link

https://youtu.be/PBNDiiBtpCw?si=rivhLud5mEQOwX02

What is debouncing?

Debouncing refers to a technique used to **ensure that electronically noisy signals cause a single action for each press or release**. If we don't debounce the buttons, a single press will result in multiple triggers even though we only want it to trigger once.

What is pull-up / pull-down?

Pull-up/Pull-down is the process of giving the input pin a well-defined state when they are not interacted with.

Pull-up resistors are used when you want the unconnected state of a pin to be read as "HIGH". Simply **connect one end of the resistor to Vcc and the other end to the button**. In a button circuit, when the button is **not pressed**, the input pin connected to the button would read **HIGH** due to the pull-up resistor. **Pressing the button would connect the pin to ground, changing its state to LOW**.

Pull-down configuration is the exact opposite. We connect one end of the resistor to ground and the other end to the button. That would make the pin connected to the button read LOW when not pressed.

What happens if we don't use pull-up / pull-down resistors?

The button will be **"floating".** This means that **its state is not being actively driven to a high (logic level 1) or low (logic level 0) voltage level.** This would make the behavior of our circuit unpredictable.

(optional) Attach your code here.)

```
#define down 13
#define up 12
int num = 0;
void setup() {
  // put your setup code here, to run once:
  pinMode(2,OUTPUT); //A
  pinMode(3,OUTPUT); //B
  pinMode(4,OUTPUT); //C
  pinMode(5,OUTPUT); //D
  pinMode(6,OUTPUT); //E
  pinMode(7,OUTPUT); //F
  pinMode(8,OUTPUT); //G
  pinMode(9,OUTPUT); //Dot
  pinMode(12,INPUT); //Up
  pinMode(13,INPUT); //Down
}

void loop() {
  // put your main code here, to run repeatedly:
  if (digitalRead(up) == HIGH){
    delay(50);
    if(digitalRead(up) == LOW) num++;
  }
  if (digitalRead(down) == HIGH){
    delay(50);
    if(digitalRead(down) == LOW) num--;
  }
  if(num > 9) num = 0;
  if(num < 0) num = 9;
```

Section 1: Button Control

When the button is pressed, we enter the first if statement, then we wait 50ms to avoid race condition. Once the user releases the button, we enter the 2nd if statement and the number is incremented.

The final 2 if statements are there to ensure that the number wrap around.

```
switch(num){
  case 0:
    digitalWrite(2,LOW); // A
    digitalWrite(3,LOW); //B
    digitalWrite(4,LOW); //C
    digitalWrite(5,LOW); // D
    digitalWrite(6,LOW); // E
    digitalWrite(7,LOW); // F
    digitalWrite(8,HIGH); // G
    digitalWrite(9,HIGH); // Dot
  break;
```

Section 2: Seven-Seg Switch Statement

The switch statement is used to control the seven-seg display. There are 10 cases in this switch statement all implemented in the same way as case 0. I will not include them all in the report because that will take up a lot of space while providing little actual value. The main thing to take note here is that the seven-segment display is in **common anode configuration so the segments will light up when the pin connected to them is LOW.**

Experiment 3: PWM Control

(Recording all results of your design.)

Video link

https://youtu.be/KV90QjEmP5c?si=achCfyYfr9p3tImc

(optional) Attach your code here. Ref.)

```
int Brightness = 0;
int state = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(3,OUTPUT); //PWM
  pinMode(12,INPUT); //Up
  pinMode(13,INPUT); //Down
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (digitalRead(up) == HIGH){
    delay(100);
    if(digitalRead(up) == LOW)
      state++;
      Serial.println(state);
  }
  if (digitalRead(down) == HIGH){
    delay(100);
    if(digitalRead(down) == LOW)
      state--;
      Serial.println(state);
  }
  if(state > 4) state = 4;
  if(state < 0) state = 0;
```

Section 1: Setup & Button Control

Essentially the same as exp2, the buttons are debounced. This means that the brightness is only increased upon button release and is only done once per button press.

The "digitalWrite()" function previously used only supports 2 voltage levels, 5V and 0V. To achieve different levels of brightness, we want to be able to freely choose between any voltage level within range of [0,5]. In other words, we require an analog signal.

```
  switch(state){
    case 0: Brightness = 255; break;
    case 1: Brightness = 250; break;
    case 2: Brightness = 225; break;
    case 3: Brightness = 190; break;
    case 4: Brightness = 50; break;
  }
  analogWrite(3,Brightness);
}
```

Section 2: Switch Statement

Arduino uses PWM(Pulse Width Modulation) to simulate an analog signal. The **"analogWrite()" function takes 2 parameters, the 1st is the output pin. Only ports with '~' are capable of PWM, so only ports ~3, ~5, ~6, ~9, ~10, ~11 on the Uno board are compatible with this function.** The 2nd parameter dictates the duty cycle of the PWM signal. There is an internal counter that counts from 0 to 255, then resets to 0 and starts counting again. The number parameter provided tells Arduino to set the output as "HIGH" before counting to the given number.

Examples:

i. analogWrite(3, 127); means that pin3 will output HIGH while the number of the counter is 0~127; it will output LOW during 127~255. This produces a 50% duty cycle signal.
ii. analogWrite(3, 64); means that pin3 will output HIGH while the number of the counter is 0~64; it will output LOW during 64~255. This produces a 25% duty cycle signal.
iii. analogWrite(3, 0); means that pin3 will always be LOW.
iv. analogWrite(3, 0); means that pin3 will always be HIGH.

To get the desired duty cycle you simply multiply the duty cycle you desire by 255 and pass the result in as parameter.
Ex:

$$Desired\ Duty\ Cycle: 75\%$$

$$analogWrite()\ Param = 255 \times 75\% = 191.25 \cong 191$$

Since the seven-segment display is in common anode configuration, the segments will light up when the pin connected to them is LOW. Therefore, LOWER duty cycle corresponds to HIGHER brightness. **According to this, I set case 0 as completely off and case 4 as the highest brightness.**

References

1. http://arduino.cc/en/Tutorial/BlinkWithoutDelay
2. https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/
3. https://docs.arduino.cc/built-in-examples/digital/Debounce/