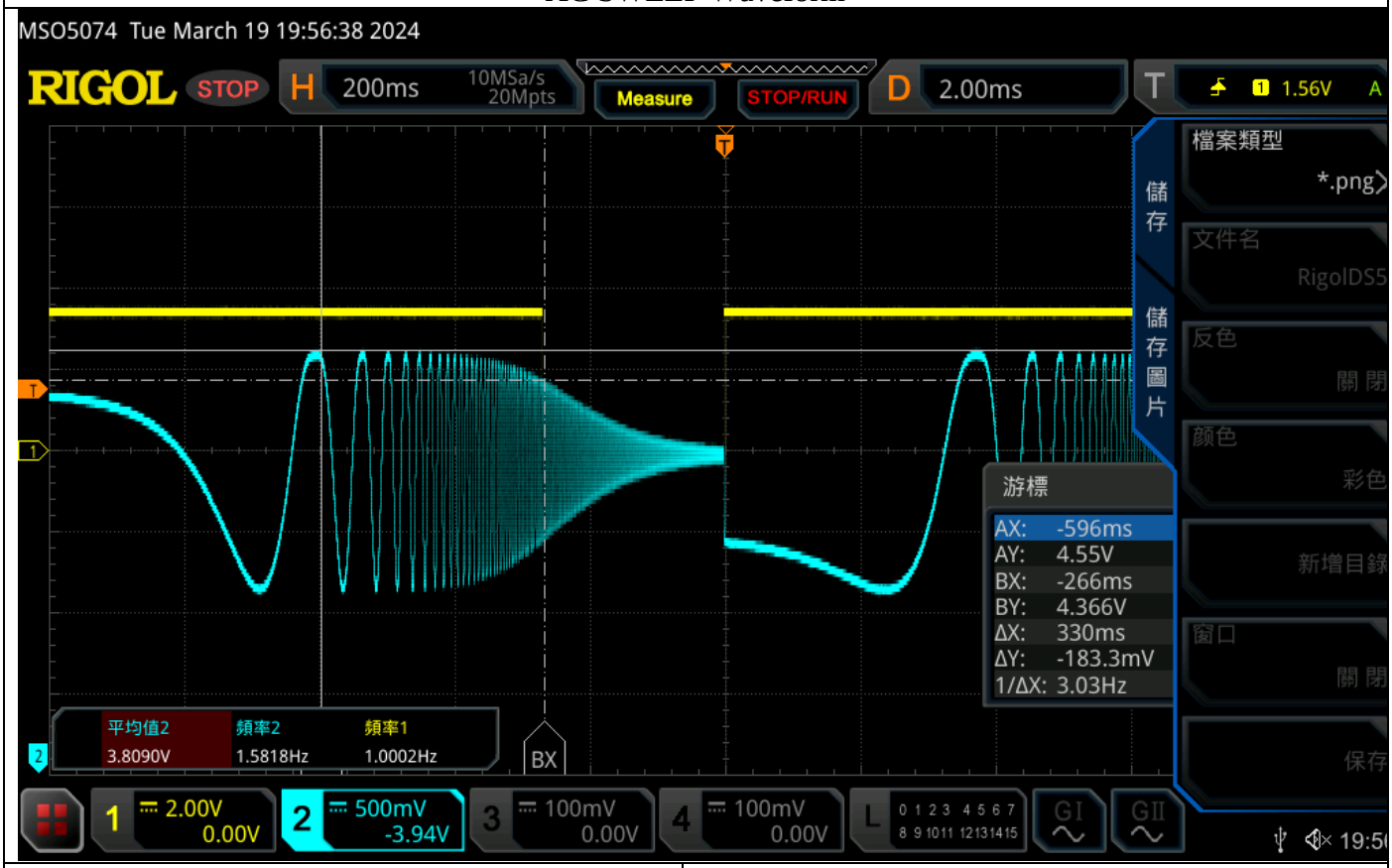
REPORT

Experiment 1: Photointerrupter characteristic

AC SWEEP Waveform



f_{H3dB} (Hz)	DC value of node out (Volt)
450	3.809

How does a photointerrupter work?

A photointerrupter consists of two main components: a light source (typically an LED) and a light detector (usually a phototransistor or photodiode). These two components are arranged in a way that the light from the source can reach the detector. When **an object causes changes in the amount of light reaching the detector, there's a corresponding change in the output signal.**

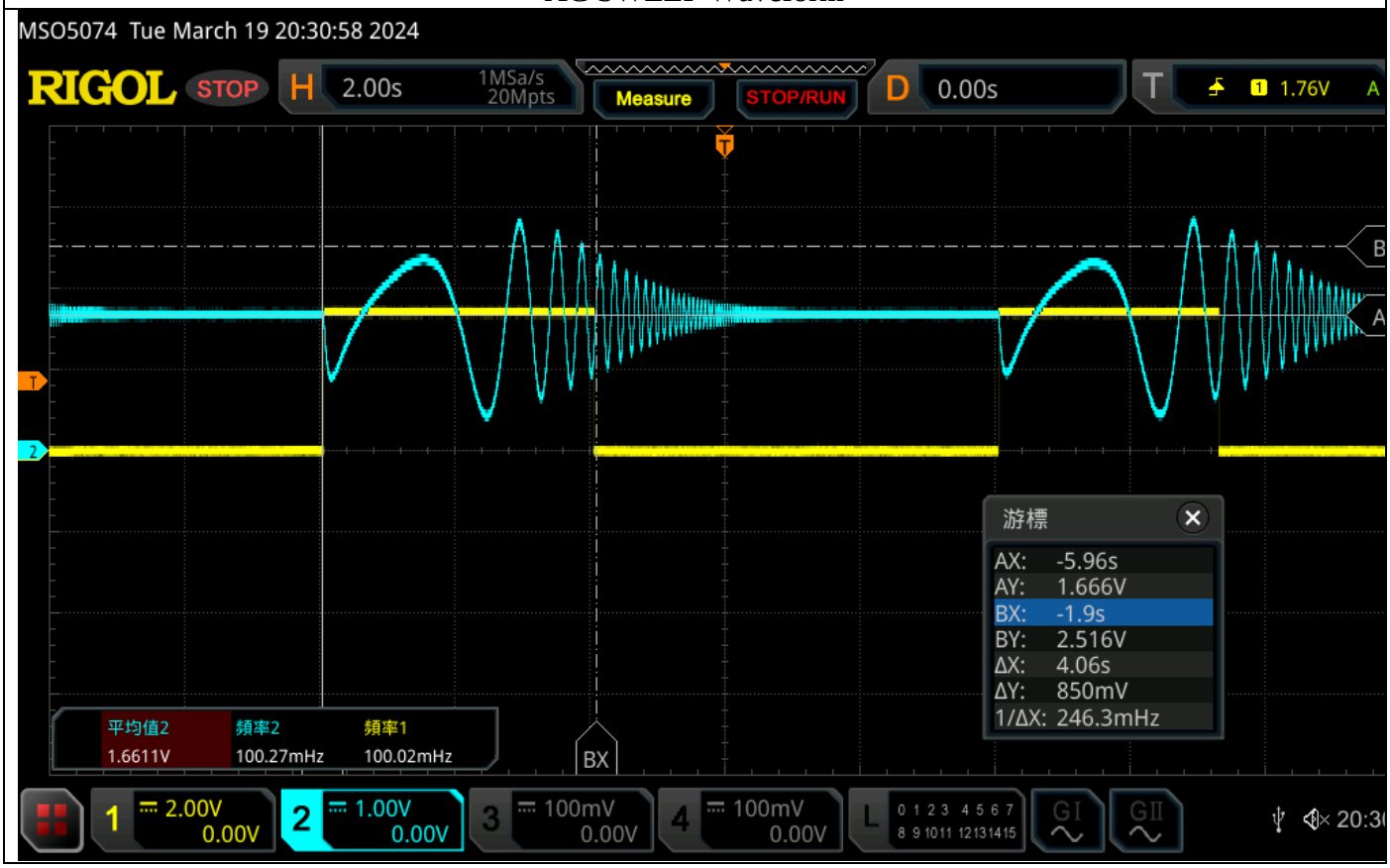
Observation:

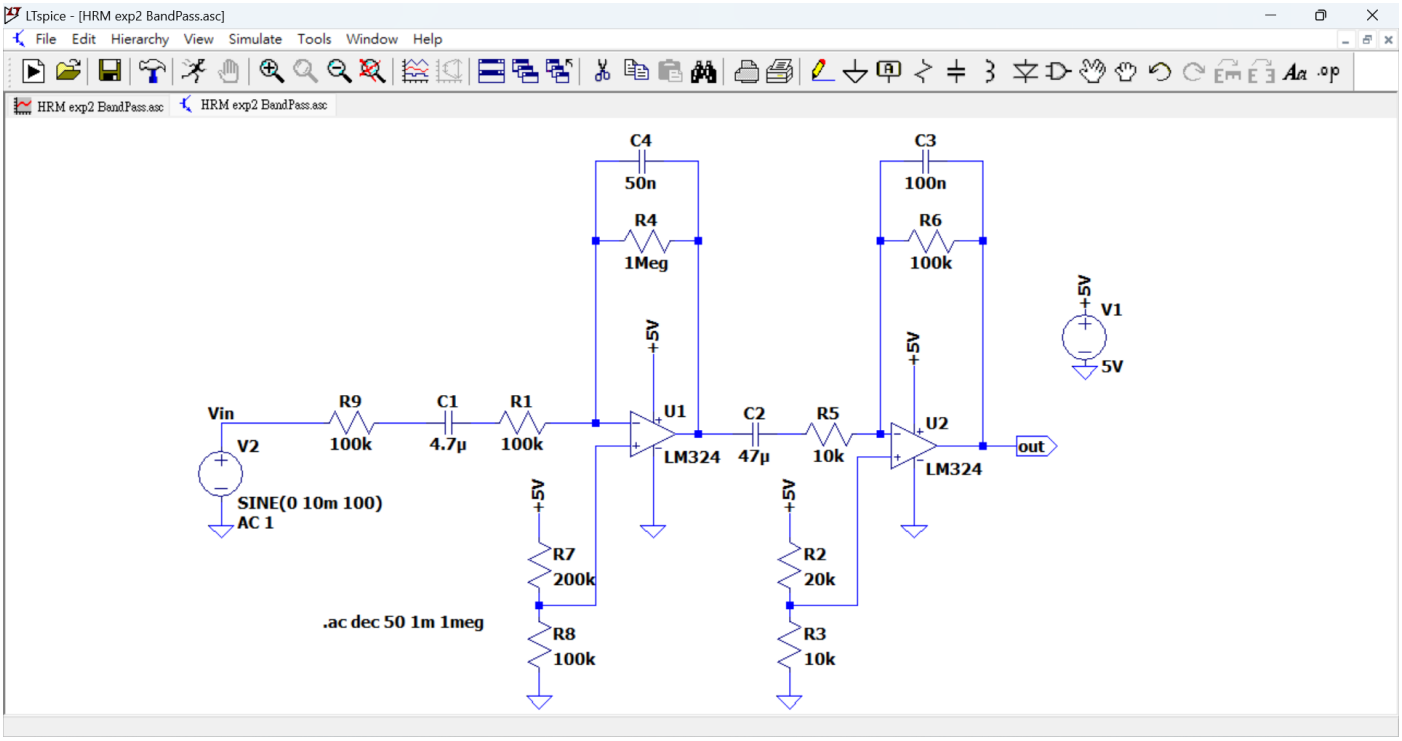
The transistor on the photointerruptor acts as a **lowpass** filter, **allowing frequencies below 450 Hz to pass through unattenuated.**

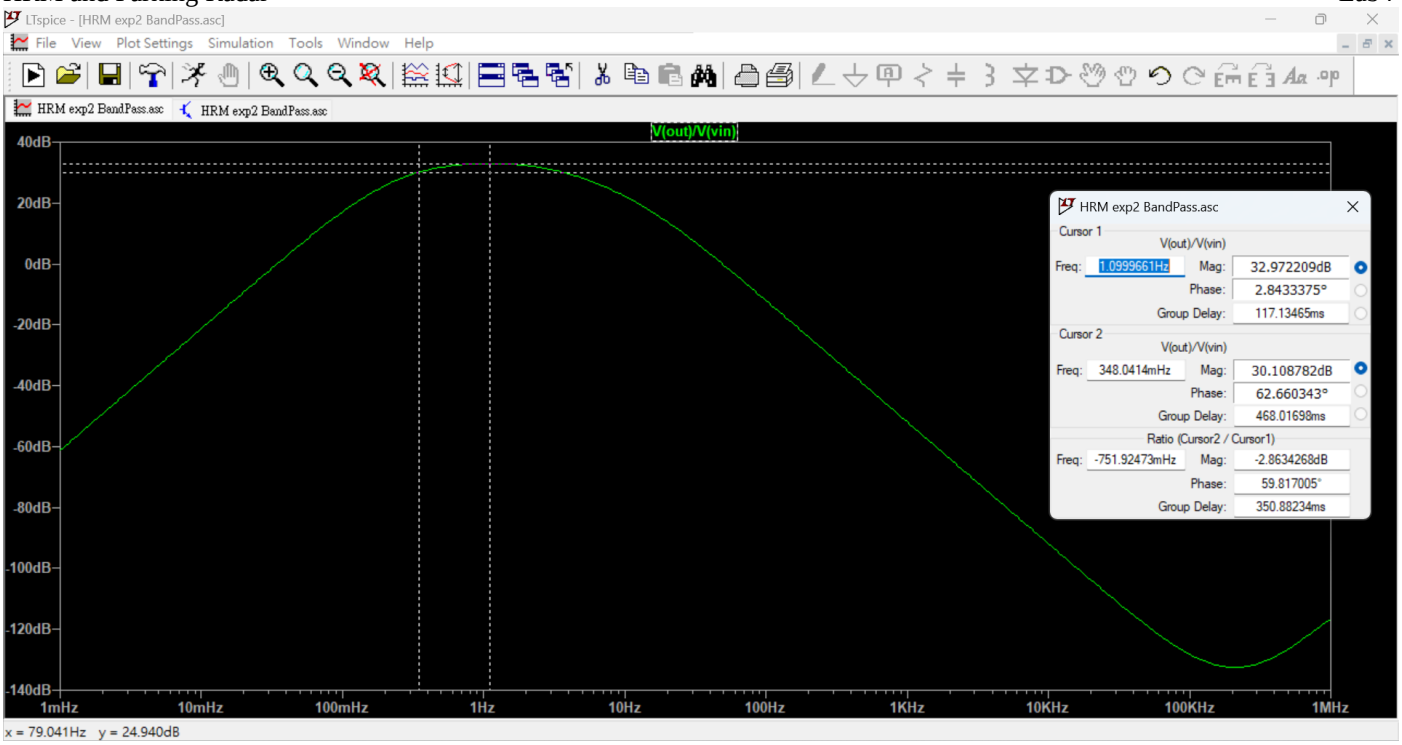
Experiment 2: Active band pass filter

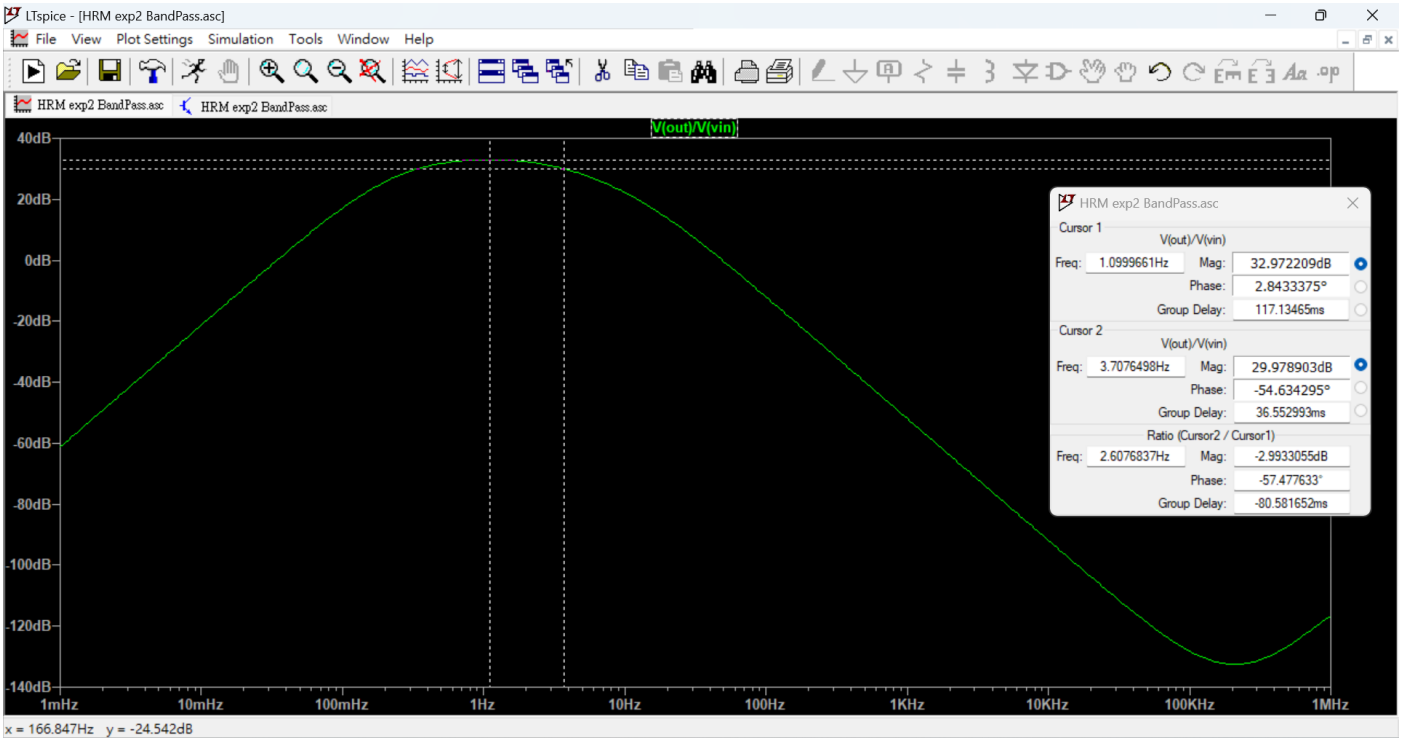
Specified Node DC value			
U1,V+ (V)	U1,Vout (V)	U2,V+ (V)	U2,Vout (V)
1.65	1.65	1.66	1.66

AC SWEEP Waveform









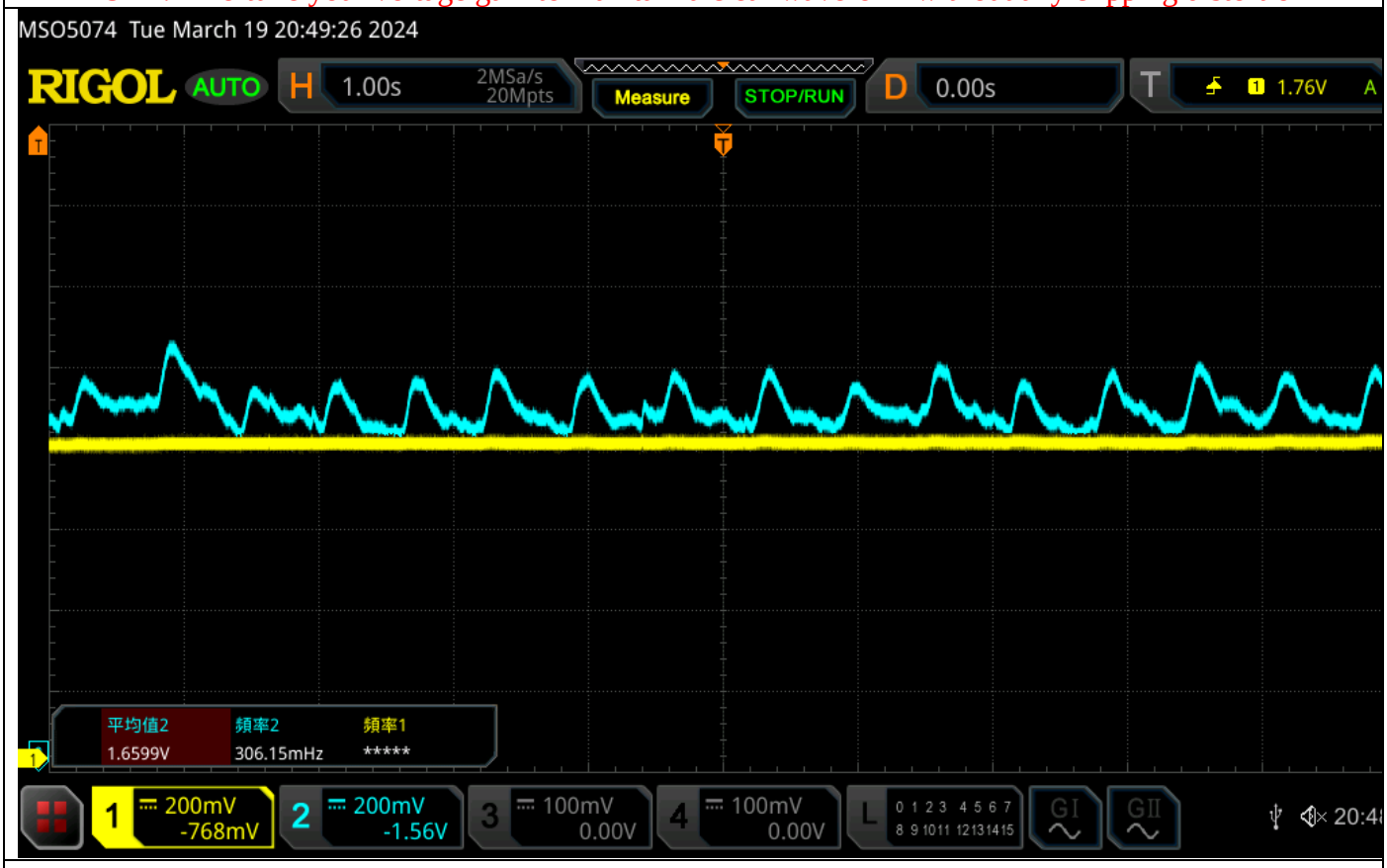
Observation:

According to my LTspice simulation, this bandpass filter has a **lower cutoff frequency** of approximately **0.35Hz** and an **upper cutoff frequency** of approximately **3.71Hz**. It's **passband gain** is around **33dB**, which will allow us to greatly amplify the signal coming from the photointerrupter.

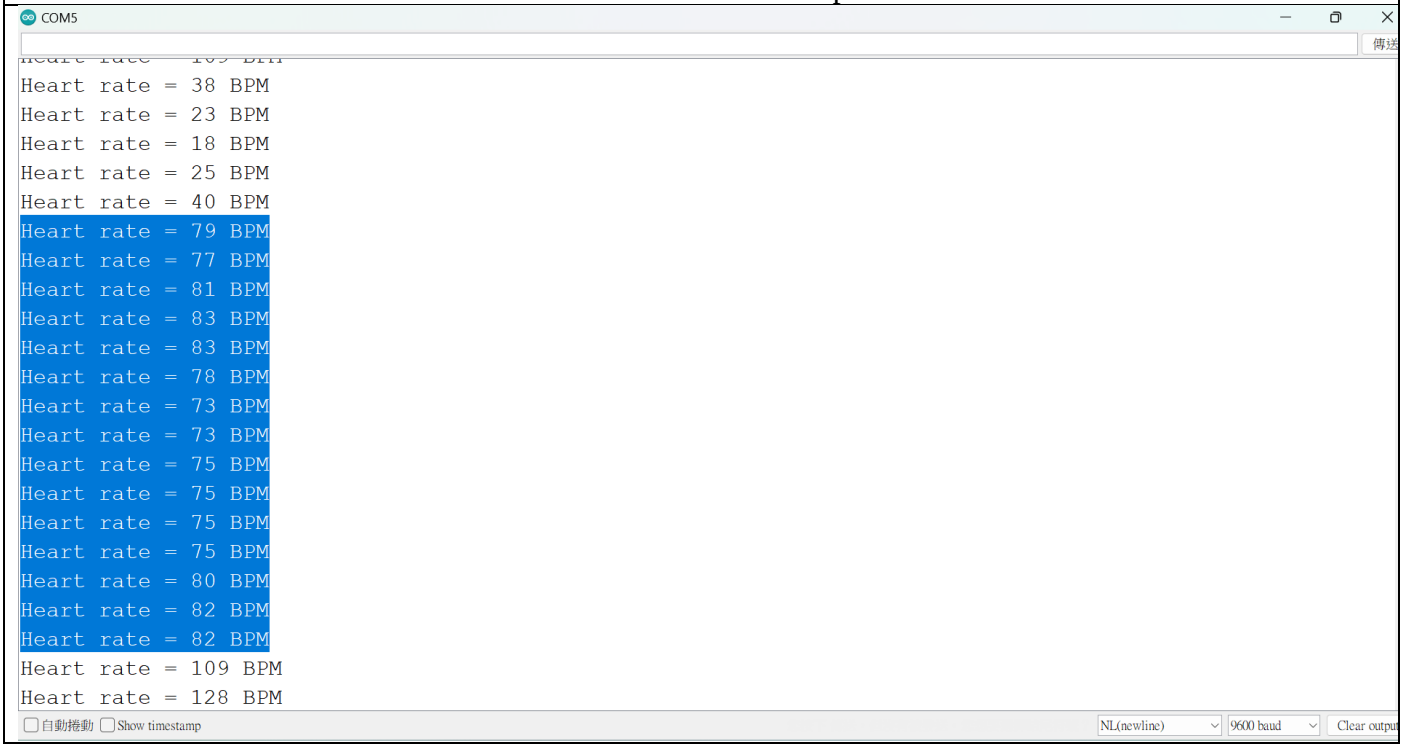
Experiment 3: Heart Rate Monitor

sensor_out and amp_out Waveform

NOTE: Fine-tune your voltage gain to maintain a clear waveform without any clipping distortion



Arduino serial monitor capture



Observation:

The Channel 1 (Yellow) signal is the original unamplified signal from the photointerrupter. It's extremely small in amplitude, therefore it's swings are barely visible on screen. The Channel 2 (Blue) signal is the result of the photointerrupter's signal passed through the bandpass filter from exp2. **Undesired frequencies are attenuated and the frequency containing the information of my heart rate is amplified by approximately 33dB.**

Experiment 4: Parking Radar

Video link

https://www.youtube.com/shorts/xK1ARda4v2s

(optional) Attach your code here.

```
int trigPin = 11;
int echoPin = 12;
int speakerpin = 6;
long duration, cm ;

void setup() {
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(speakerpin, OUTPUT);
}

void loop()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
  cm = (duration/2) / 29.1;

  Serial.print(cm);
  Serial.println(" cm");

  if (cm <= 5) {
    analogWrite(speakerpin, 200);
    delay (25);
    digitalWrite(speakerpin, 0);
    delay (25);
  }
  if (cm > 5 && cm <= 10) {
    analogWrite(speakerpin, 200);
```

```
    delay (50);
    digitalWrite(speakerpin, 0);
    delay (50);
  }
   if (cm > 10 && cm <= 15) {
    analogWrite(speakerpin, 200);
    delay (100);
    digitalWrite(speakerpin, 0);
    delay (100);
  }
  if (cm > 15 && cm <= 20) {
    analogWrite(speakerpin, 200);
    delay (250);
    digitalWrite(speakerpin, 0);
    delay (250);
  }
  if (cm > 15 && cm <= 20) {
    digitalWrite(speakerpin, 0);
    delay (100);
  }
}
```

Code Explanation:

The code used here is fairly intuitive. I utilized the **analogWrite() function to generate a PWM signal of around 490 Hz** and pass it to the speaker. The delay() function is used to control the interval between beeps. Smaller time intervals are used as the distance between the HC-SR04 and the approaching object gets smaller.

How does the function "analogWrite()" work?

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, 255); 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 × 75% = 191.25 ≅ 191

Reference:

1. ROHM Semiconductor: https://www.rohm.com/products/faq-search/faqId/279
2. ElectronicsTutorials: https://www.electronics-tutorials.ws/filter/filter_7.html
3. Ray 的 Arduino 教學: https://sites.google.com/view/rayarduino/%E8%B6%85%E9%9F%B3%E6%B3%A2%E8%B7%9D%E9%9B%A2%E6%84%9F%E6%B8%AC%E5%99%A8