ME210 Introduction to Mechatronics
Lab 1

Winter 2026

Julia Jiang

Part 1: The Arduino as a Controllable Signal Source

1.3

A unity gain buffer cannot swing to its rails (0V and 5V) and instead swings from a bit above 0V and a bit below 5V. The potentiometer can operate full swing, so connecting a unity gain buffer would decrease the operating voltage range. Since we are mapping the voltage to a large range in frequency, a small change in voltage will have a big impact on the frequency range.

1.5

resistor size = 270 Ω

the current the arduino pin takes in is 20mA

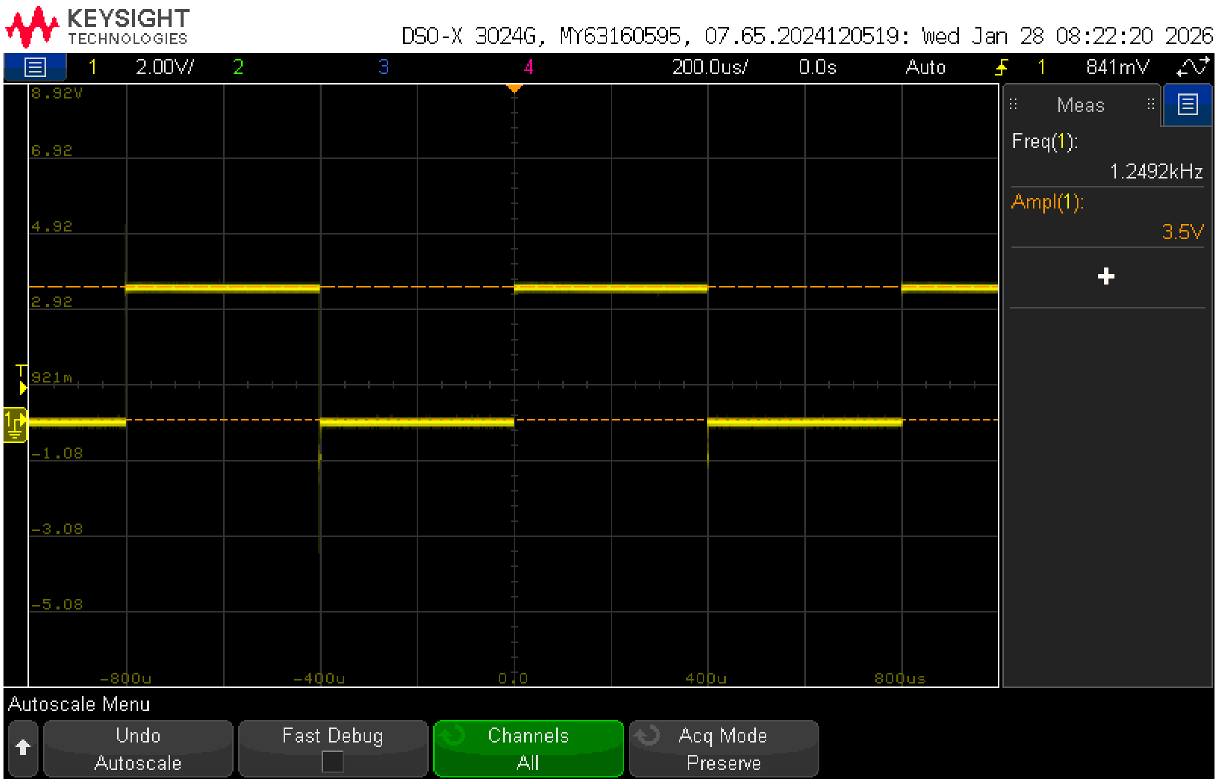
$R_L \geq \frac{5V}{20mA} = 250\Omega$

∴ we choose 270 Ω since that is the closest resistor value that fits the specifications

```
#include "TimerInterrupt.h"
#include "ISR_Timer.h"
#define USE_TIMER_1   true
#define USE_TIMER_2   false
#define OUTPUT_PIN 2
volatile bool toggleState = LOW;
long currentFreq = 12500;

void TimerHandler(){
  //after timer expires, change led state
  toggleState = !toggleState;
  digitalWrite(OUTPUT_PIN, toggleState);
}
void setup(){
  pinMode(OUTPUT_PIN, OUTPUT);
  ITimer1.init();
}
void loop(){
    int potValue = analogRead(A0);
    long newFreq = map(potValue, 0, 1023, 100, 25000); // ×2 for toggle
    if (newFreq != currentFreq){
        currentFreq = newFreq;
        ITimer1.setFrequency(currentFreq, TimerHandler);
    }
}
```

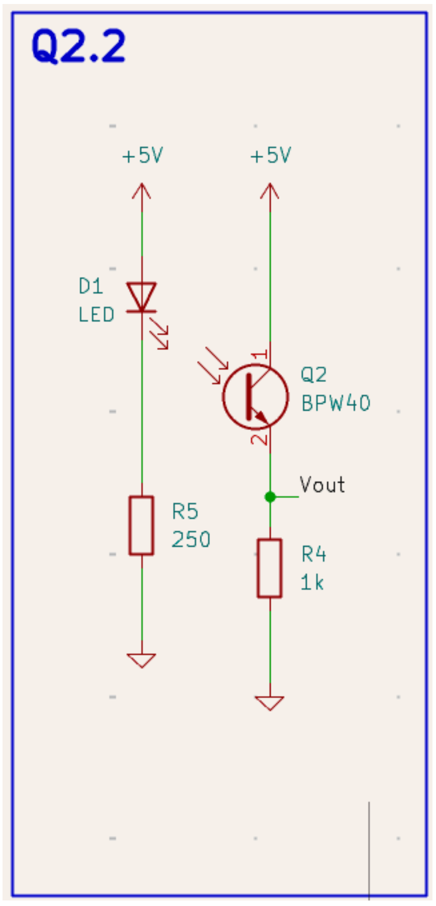
1.6 Toggling the LED at 1.25kHz



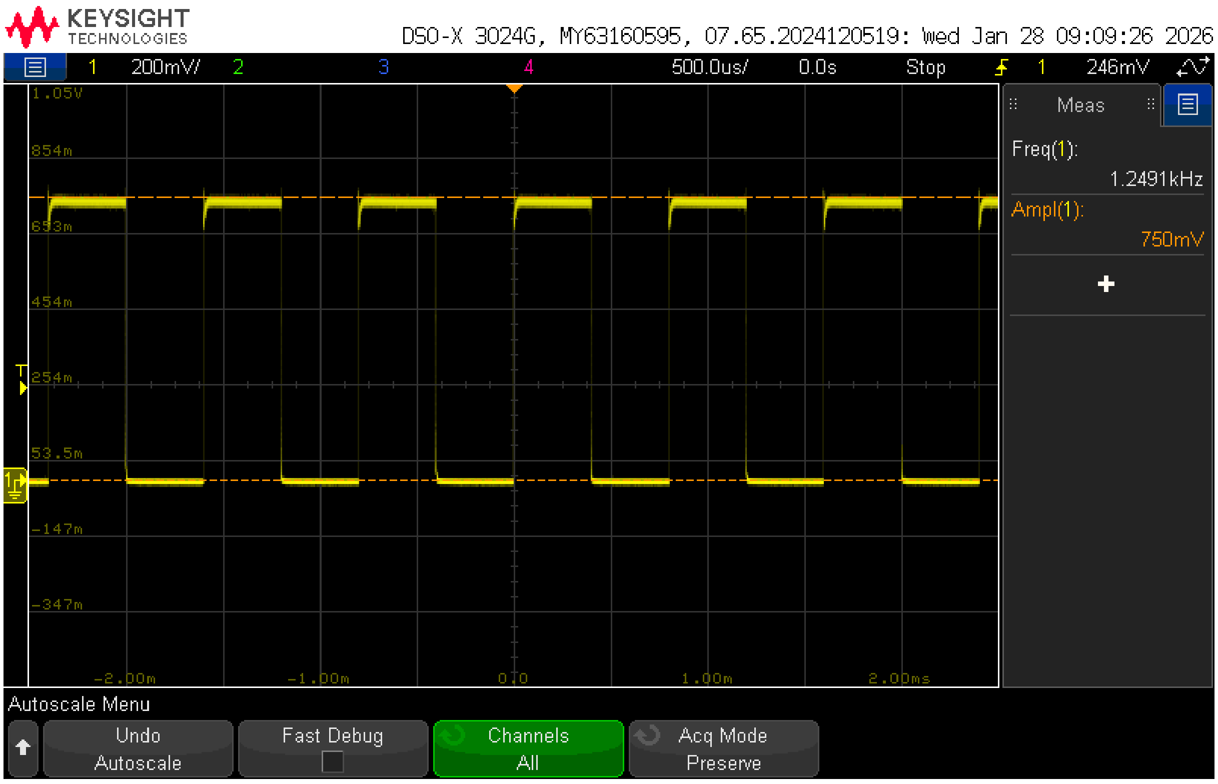
```
#include "TimerInterrupt.h"
#include "ISR_Timer.h"
#define USE_TIMER_1   true
#define USE_TIMER_2   false
#define OUTPUT_PIN 2
volatile bool toggleState = LOW;
void TimerHandler(){
  toggleState = !toggleState;
  digitalWrite(OUTPUT_PIN, toggleState);
}
void setup(){
  pinMode(OUTPUT_PIN, OUTPUT);
  ITimer1.init();
  ITimer1.setFrequency(2500, TimerHandler); //1.25kHz * 2
}
void loop(){
}
```

Part 2: The Phototransistor

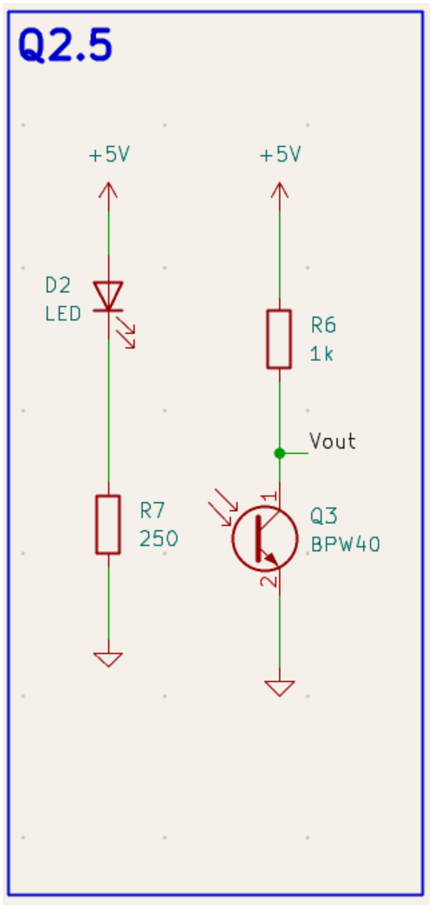
2.2 Sourcing Configuration



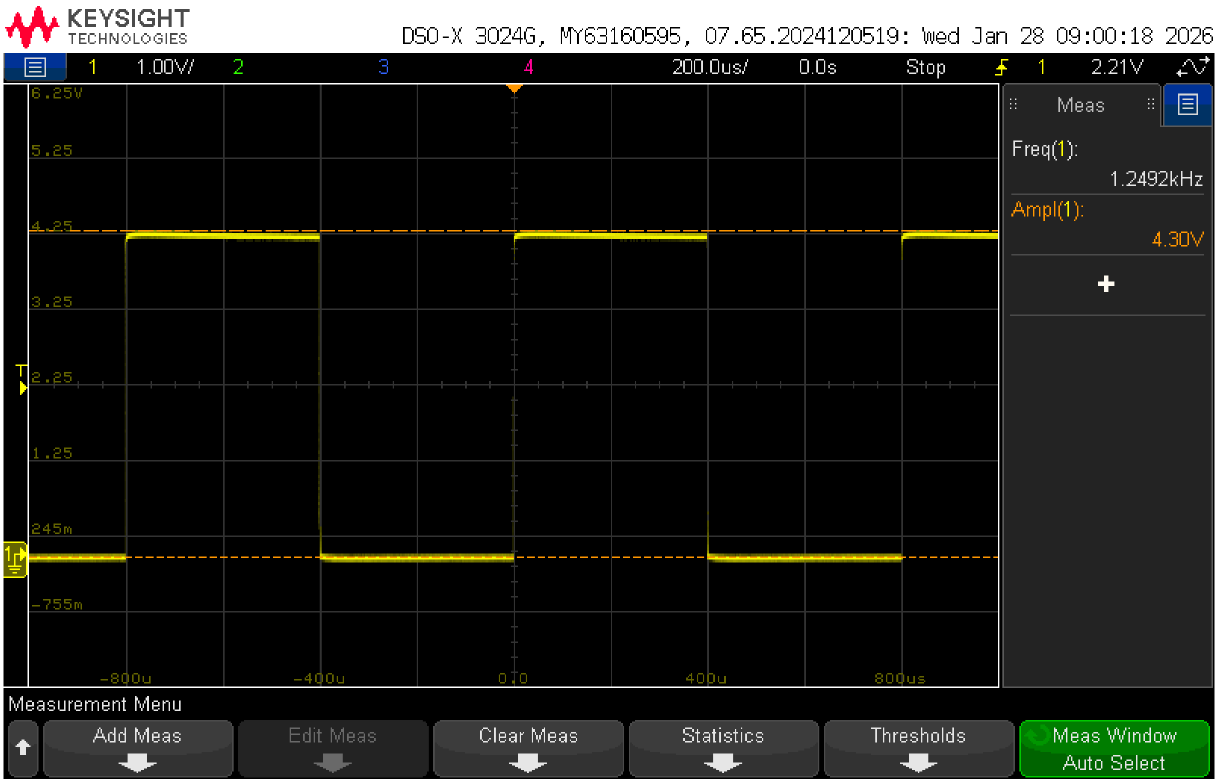
2.3 Sourcing Configuration Waveform



2.5 Sinking Configuration



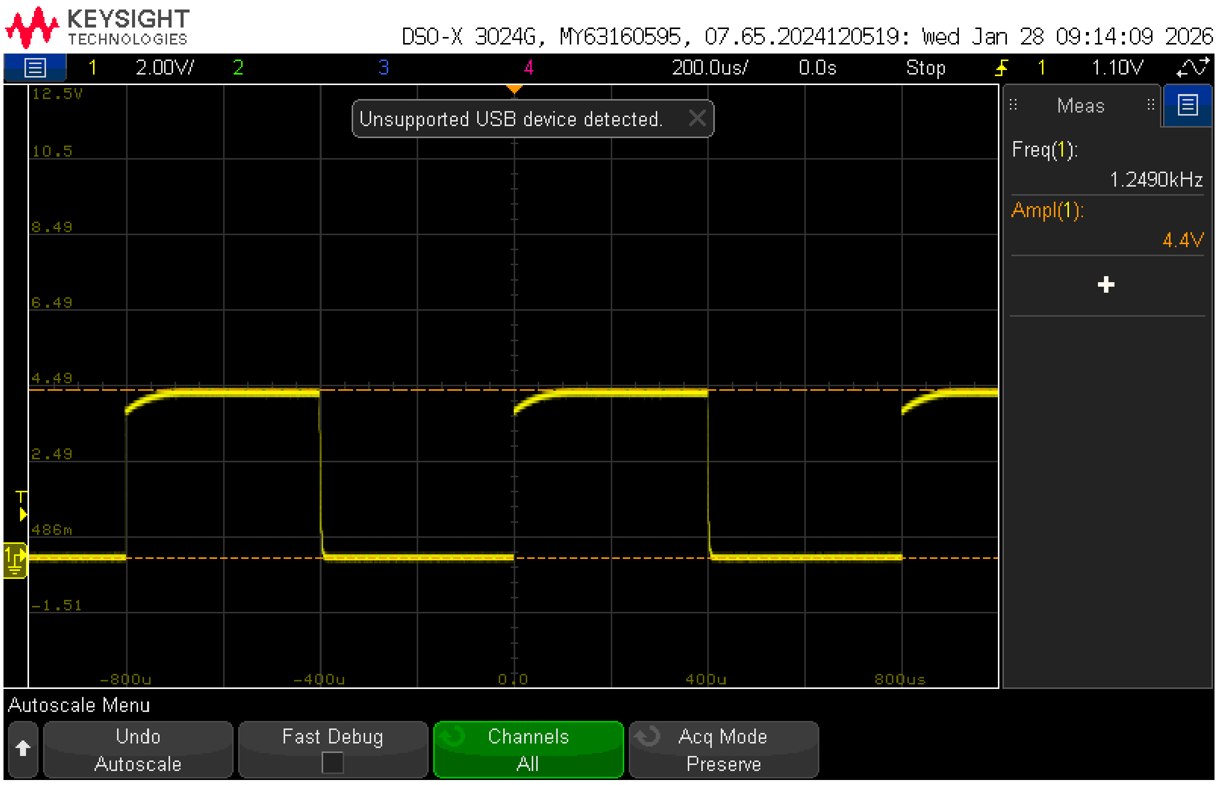
2.6 Sinking Configuration Waveform



2.7 Difference between Sourcing and Sinking Waveforms

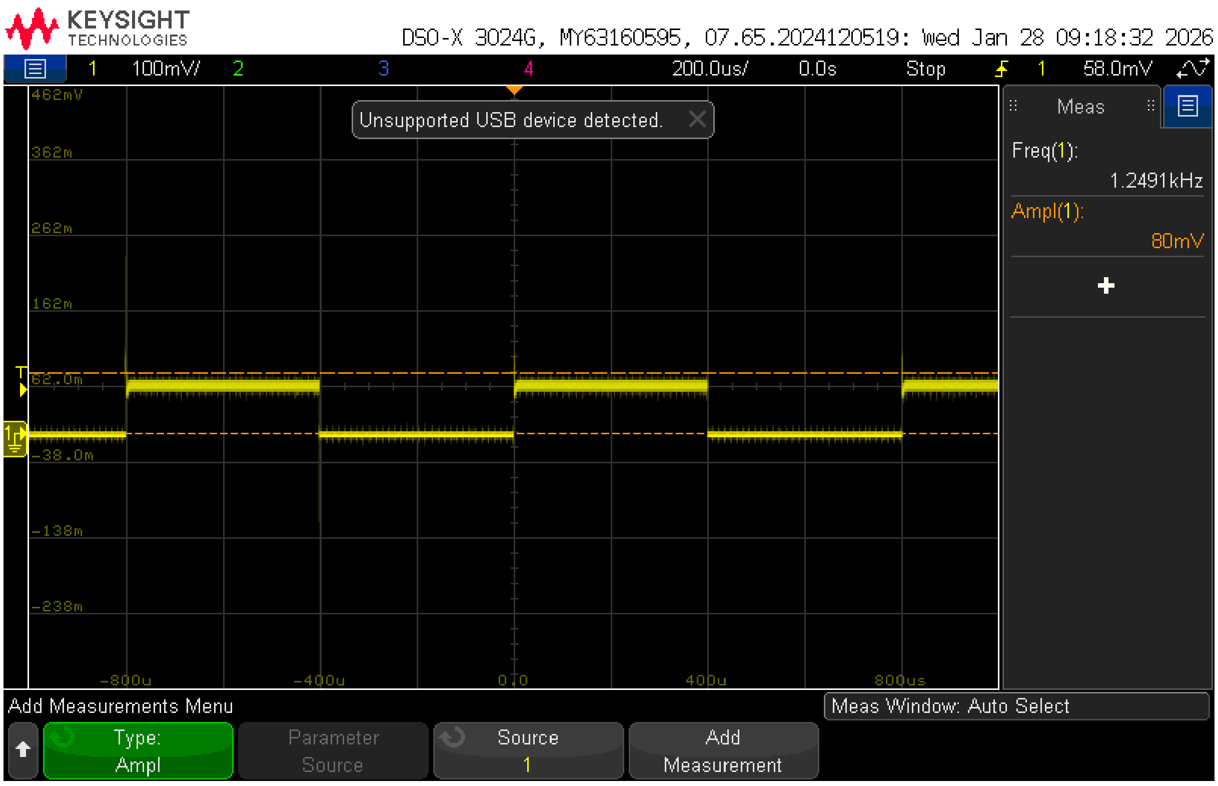
The waveforms have different voltages because the resistors are in different places. In the sourcing configuration, the waveform measures voltage across the resistor directly but in the sinking configuration the voltage is measured at the node below the load resistor. Either way, the voltages across the resistors are about the same and the currents through the phototransistor are about the same, which makes sense because the current depends on the distance of the IR LED from the phototransistor.

2.9 10k Ω



The 10k Ω resistor forces V_0 to hit the supply rails since the current through the phototransistor (around 0.7mA) wants to stay constant due to the constant IR LED emmission and the resistance is large. Thus, the the current is limited to an amount where V_0 hits the supply rail.

2.11 100 Ω



The waveform is the same as with the 1k /ohm load resistor since V_o doesn't hit the supply rails and the same amount of current can pass through the phototransistor.

Part 3: Op-Amps

3.1

Inverting op amp gain = 10

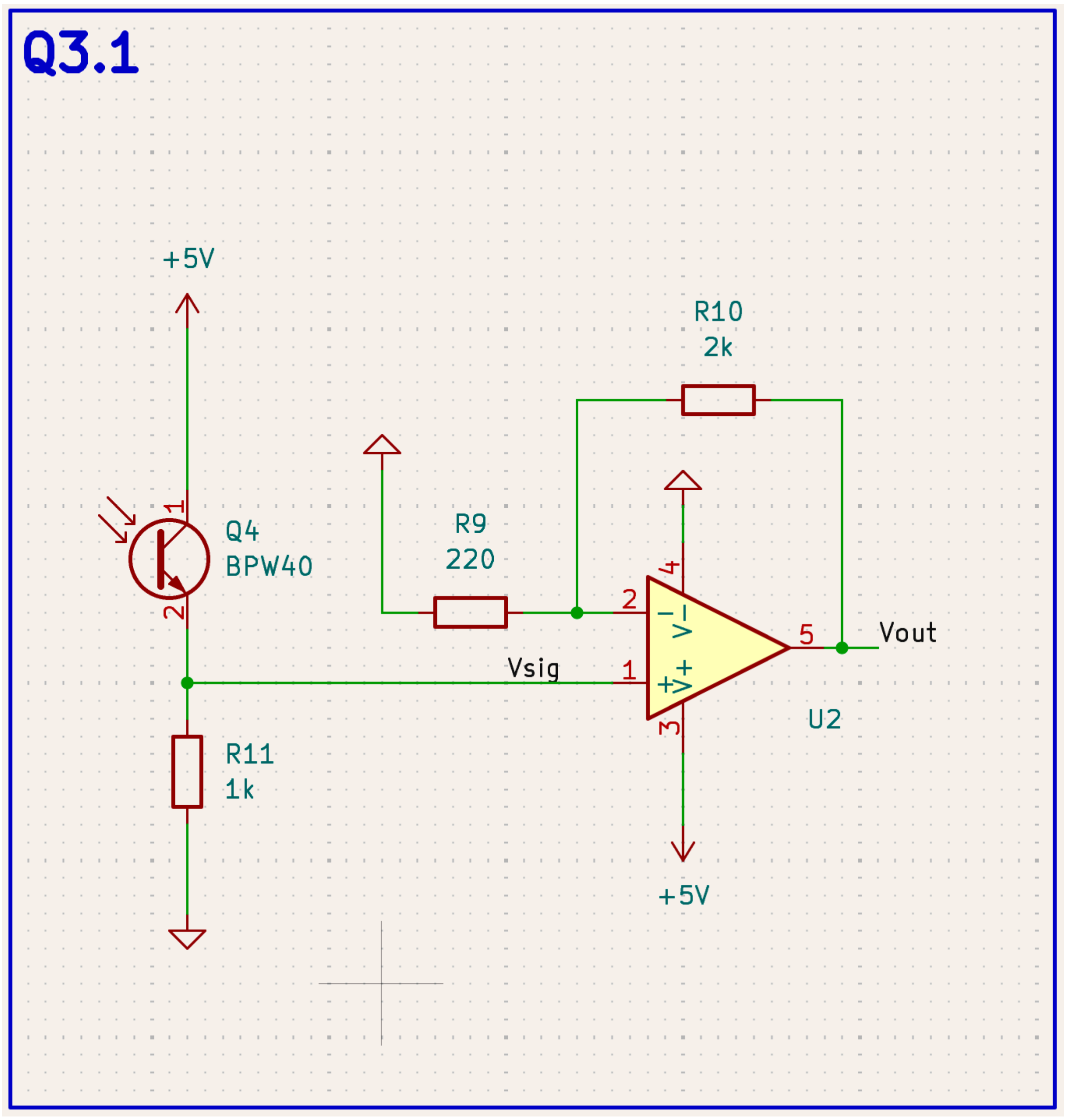
$G = 1 + \frac{R_1}{R_2} = 10$

$\frac{R_1}{R_2} = 9$

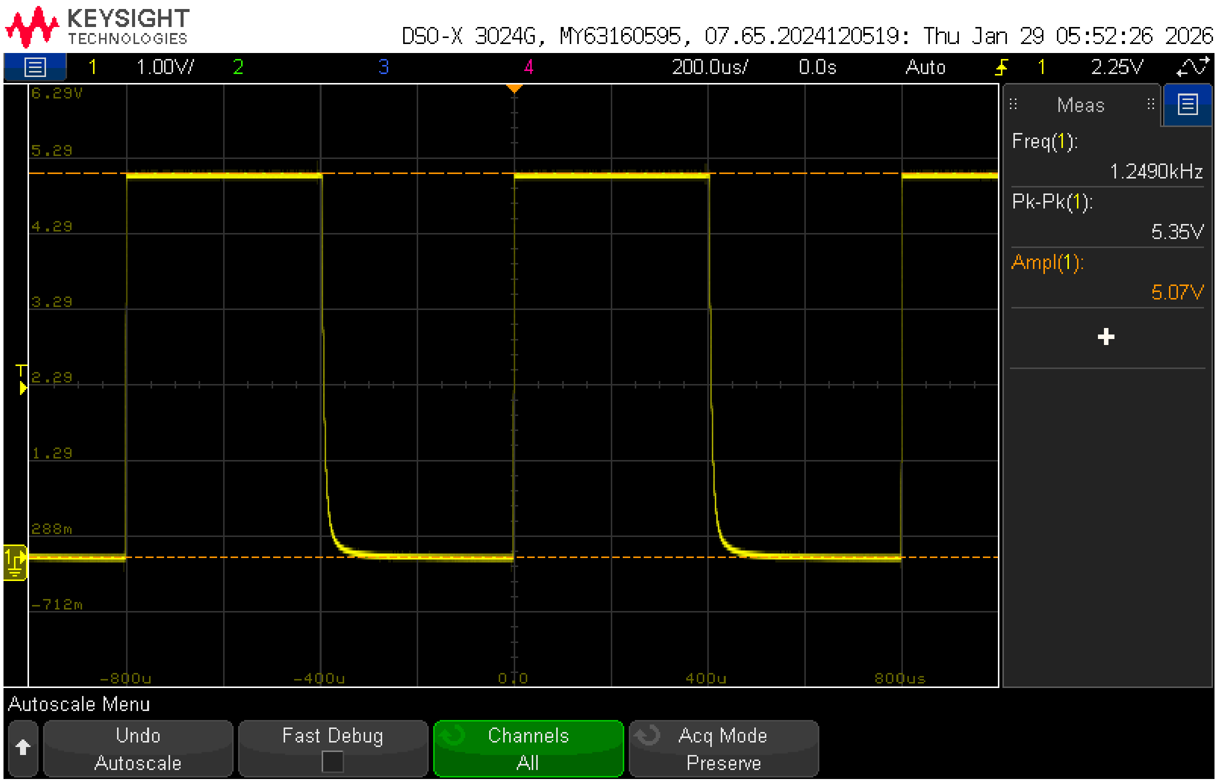
∴ we choose the closest resistors that produce this ratio

$R_1 = 2k\Omega$

$R_2 = 220k\Omega$


Q3.1
+5V
R10
2k
Q4
BPW40
R9
220
2
1
5
4
3
Vout
Vsig
U2
R11
1k
+5V

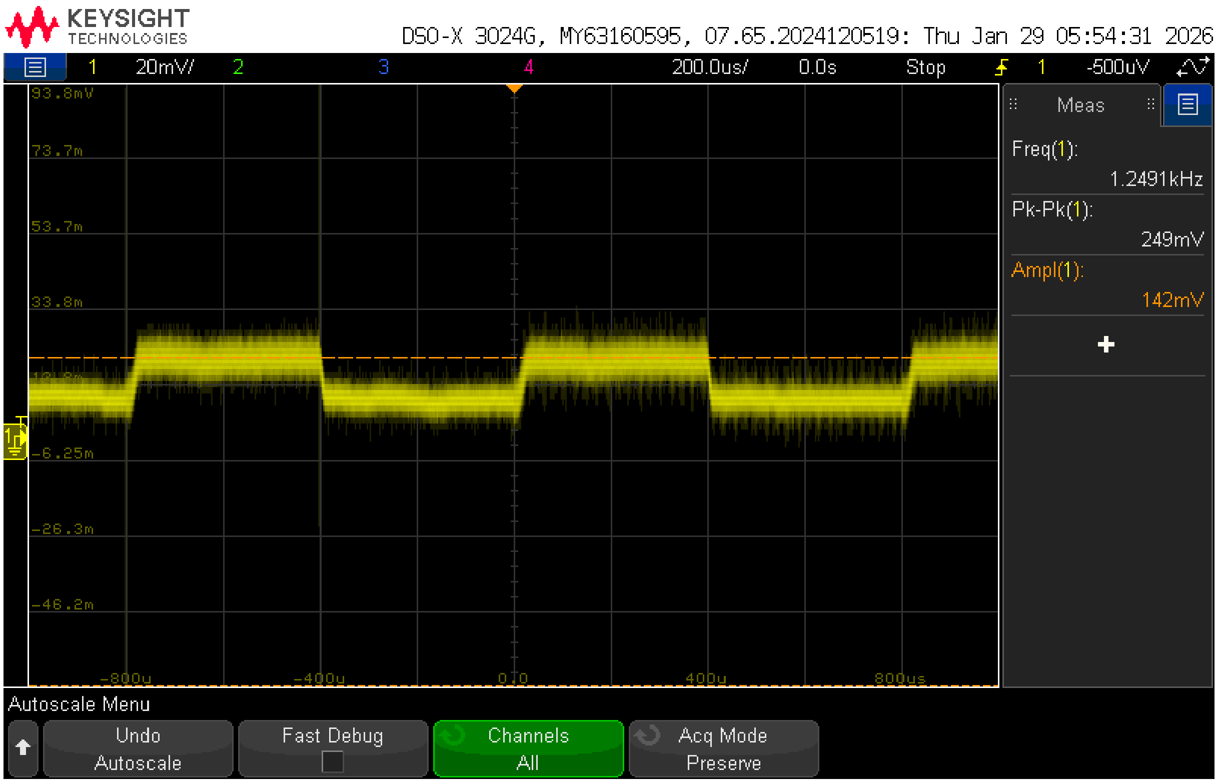
3.3


KEYSIGHT TECHNOLOGIES
DSO-X 3024G, MY63160595, 07.65.2024120519: Thu Jan 29 05:52:26 2026
1 1.00V/ 2 3 4 200.0us/ 0.0s Auto 1 2.25V
Meas
Autoscale Menu
Undo Autoscale
Fast Debug
Channels All
Acq Mode Preserve

Freq(1): 1.2490kHz
Pk-Pk(1): 5.35V
Ampl(1): 5.07V

3.4 Reversing Phototransistor Polarity

The signal is noiser and weaker, which makes sense because the phototransistor sensitivity decreases.


KEYSIGHT TECHNOLOGIES
DSO-X 3024G, MY63160595, 07.65.2024120519: Thu Jan 29 05:54:31 2026
1 20mV/ 2 3 4 200.0us/ 0.0s Stop 1 -500uV
Meas
Autoscale Menu
Undo Autoscale
Fast Debug
Channels All
Acq Mode Preserve

Freq(1): 1.2491kHz
Pk-Pk(1): 249mV
Ampl(1): 142mV

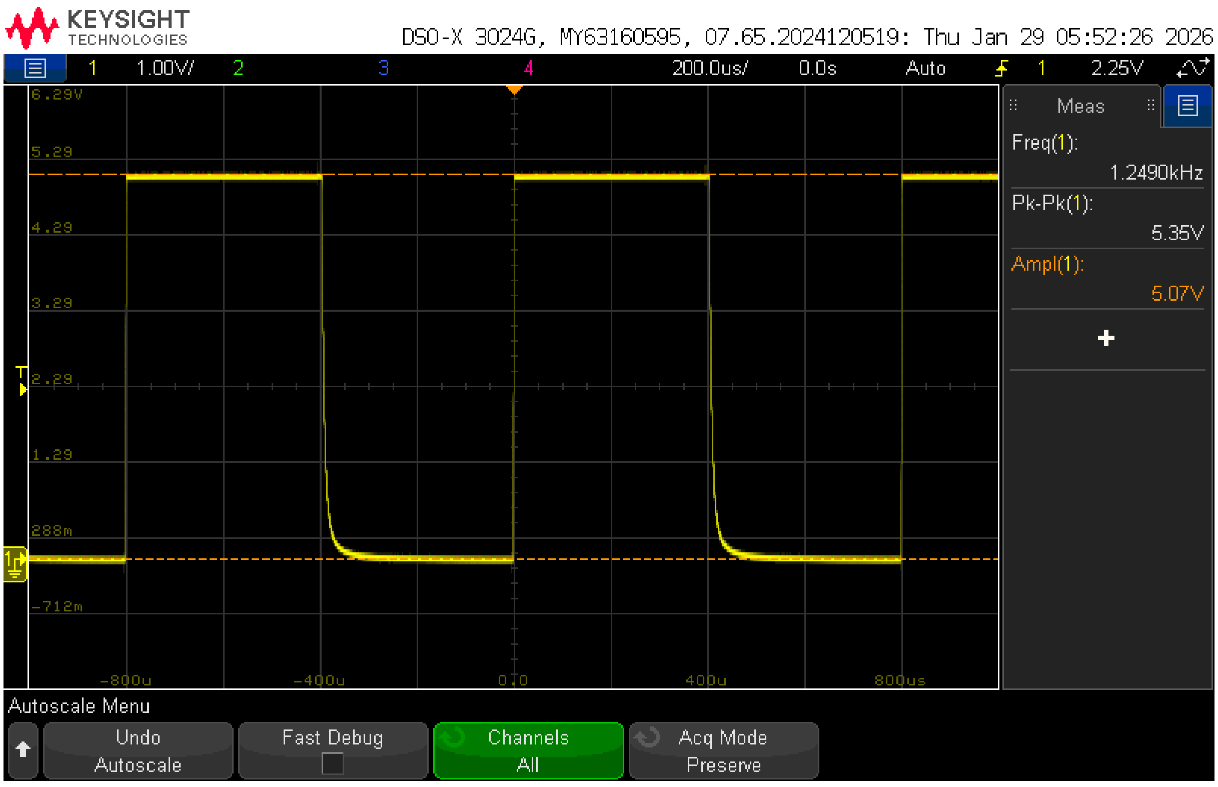
3.5 Measured vs. Calculated Gain

Measured gain = 5V

Calculated gain $= \frac{5V}{0.75V} = 6.67\frac{V}{V}$ -> 5V (rails op amp)

This agrees with the calculated gain since the output voltage from the phototransistor is around 0.75V and a gain of 10 means the op amp saturates at the supply rails, making the output 5V.

3.6 Oscilloscope Settings



Freq(1): 1.2490kHz

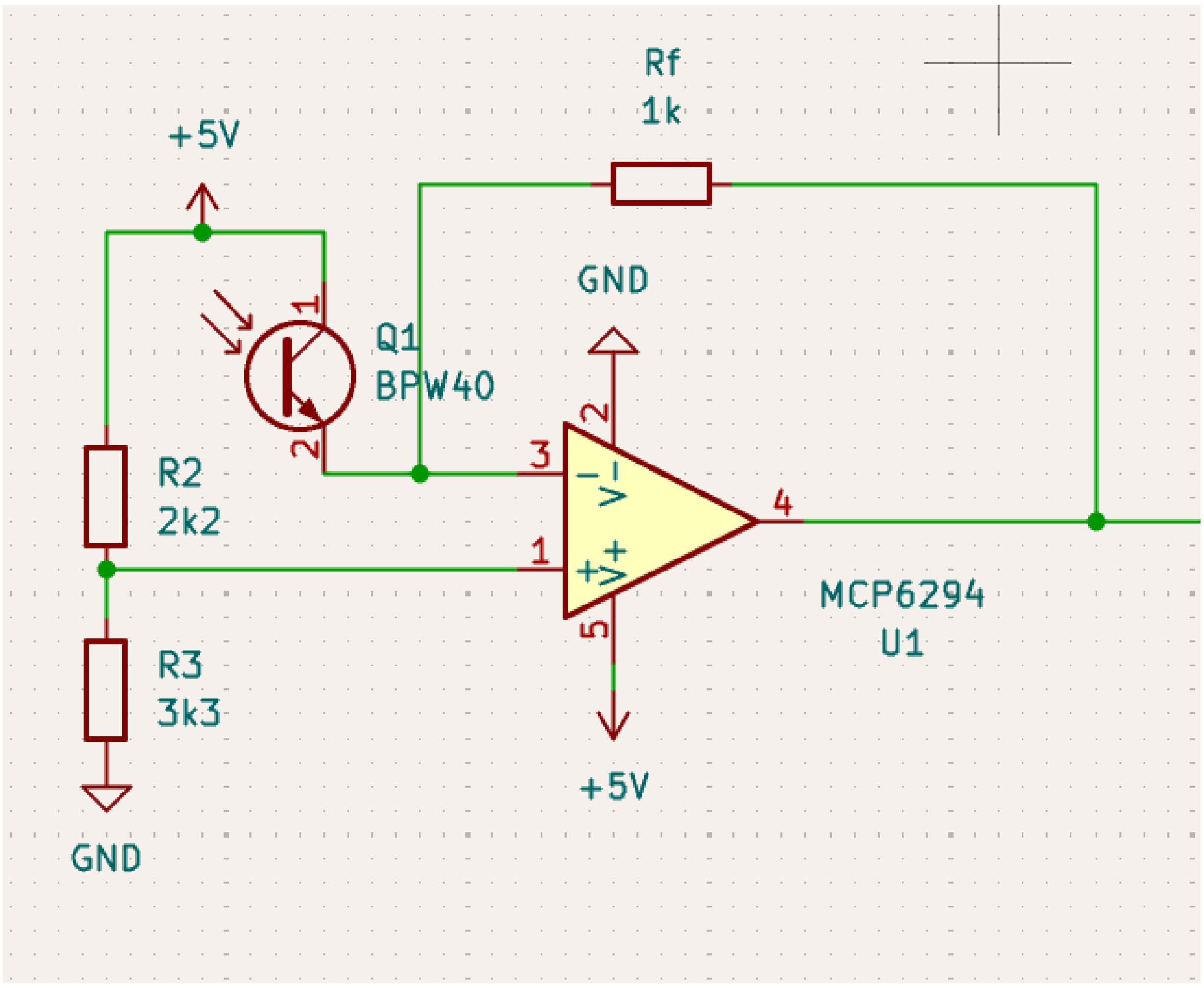
Pk-Pk(1): 5.35V

Ampl(1): 5.07V

I used Amplitude on the oscilloscope to measure the value of HI on the op amp. I chose this since peak-to-peak was not measuring to the top of the waveform.

Part 4: Transresistive Amplifiers

4.1 Transresistive Amplifier

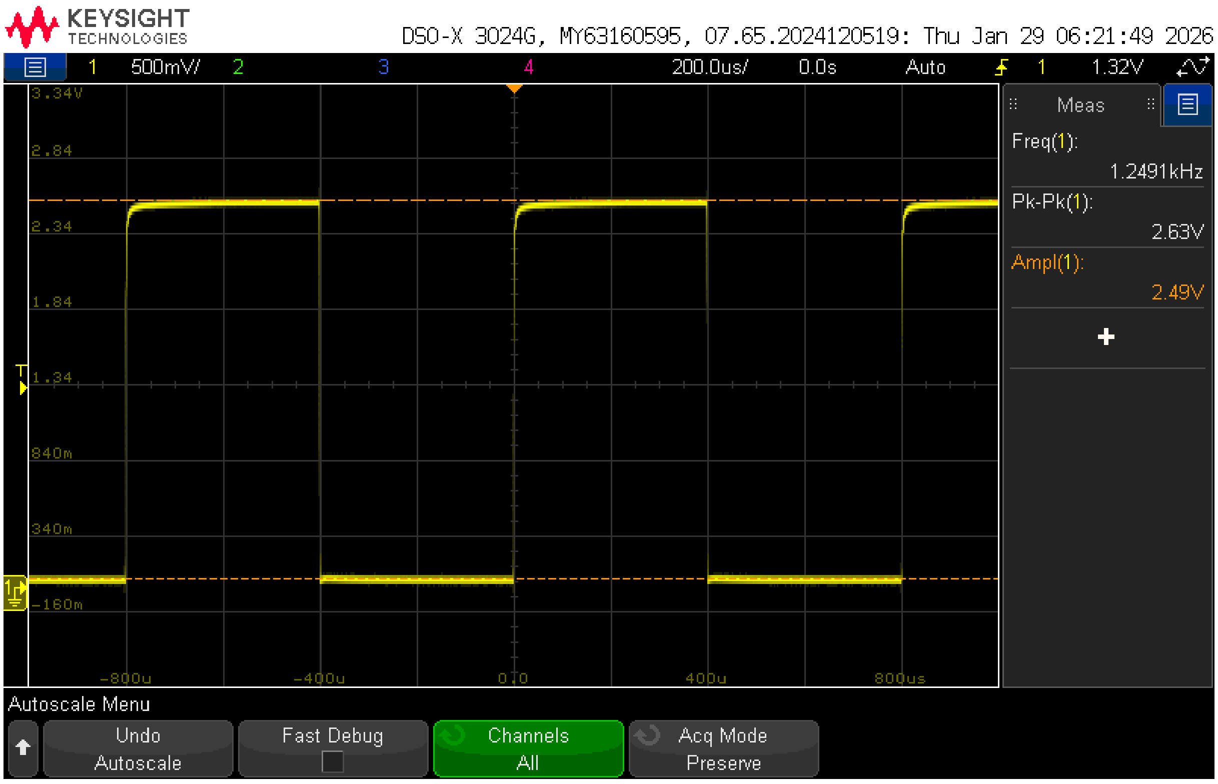


V_CE being 2.5V is not ideal for the phototransistor because from the datasheet, the typical operating voltage is 5V.

4.2

The output of the amplifier at 1mW/cm^2 should be low and the output at 5mW/cm ^{2} should be high.

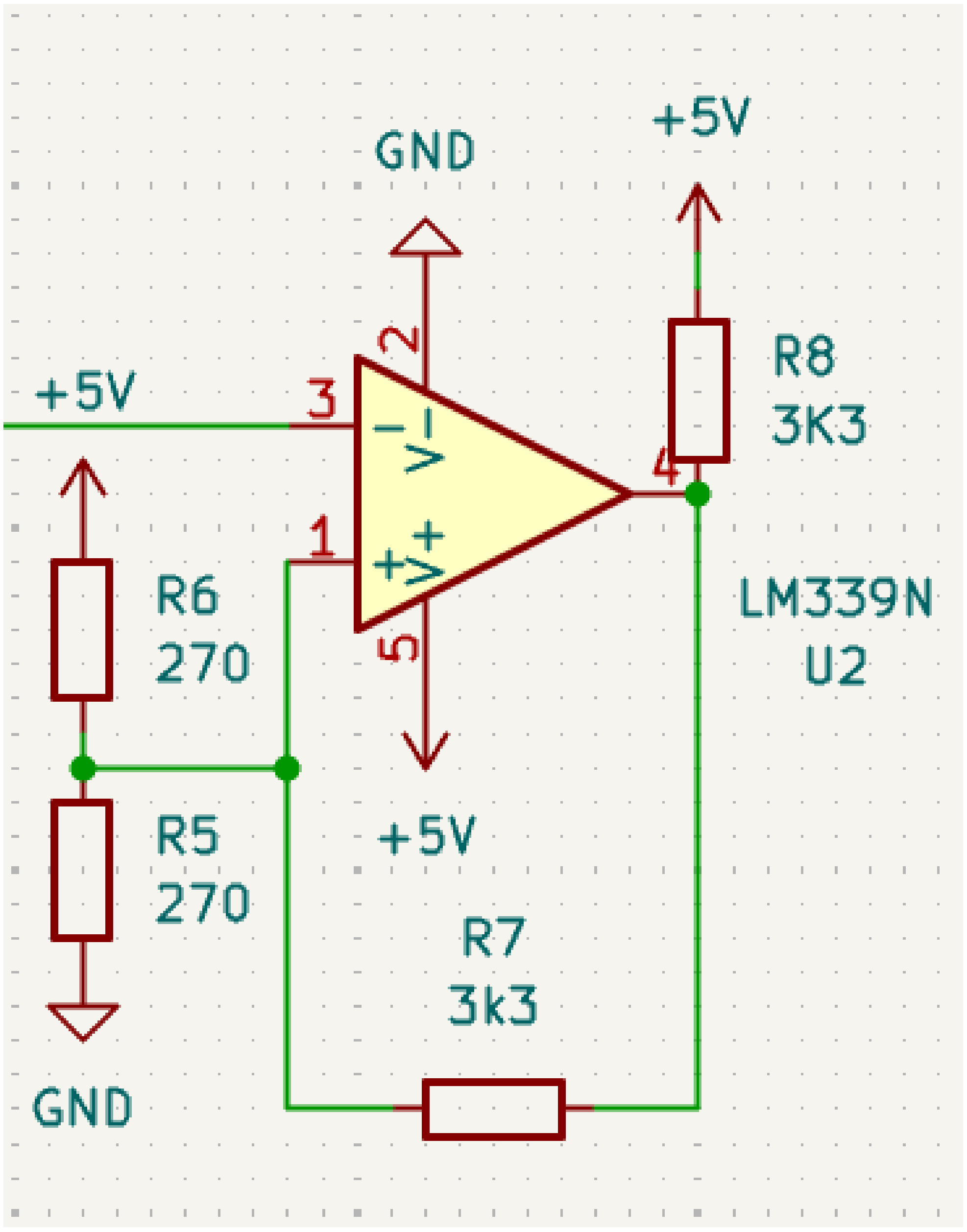
4.4 Op-Amp Output Waveform



The output HI is only around half the voltage of the output HI in part 3. We can make the signal amplitude larger by putting the IR LED closer to the phototransistor.

Part 5: Comparators

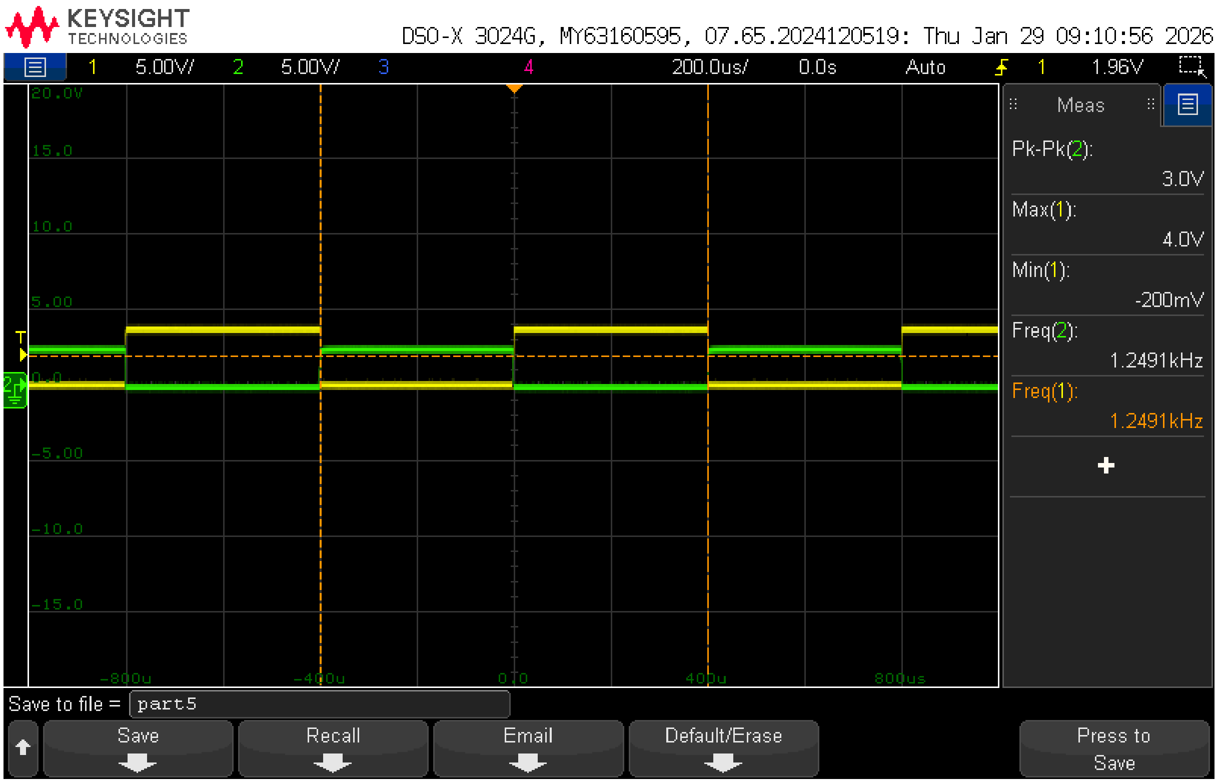
5.2 Comparator Schematic



5.3

Trip points: 2.4V, 2.6V

5.5 Output of Op Amp and Output of Comparator Graph



Pk-Pk(2): 3.0V
Max(1): 4.0V
Min(1): -200mV
Freq(2): 1.2491kHz
Freq(1): 1.2491kHz

Part 6: Capturing the Output

6.2 Incrementing Counter

```
#include "TimerInterrupt.h"
#include "ISR_Timer.h"

#define PIN_SIGNAL_IN 2
#define OUTPUT_PIN 3 //frequency is toggled here

volatile int counter = 0;
void CountFallingEdges(){
  counter++;
}

void setup(){
  Serial.begin(9600);
  pinMode(OUTPUT_PIN, OUTPUT);
  pinMode(PIN_SIGNAL_IN, INPUT);
  ITimer1.init();
  ITimer1.setFrequency(currentFreq, TimerHandler);
  attachInterrupt(digitalPinToInterrupt(PIN_SIGNAL_IN), CountFallingEdges,
FALLING);
}
void loop(){
  //potentiometer logic
  int potValue = analogRead(A0);
  long newFreq = map(potValue, 0, 1023, 100, 25000); // ×2 for toggle
  if (newFreq != currentFreq){
      currentFreq = newFreq;
      ITimer1.setFrequency(currentFreq, TimerHandler);
  }
}
```

6.4 Calculating Frequency

```
#include "TimerInterrupt.h"
#include "ISR_Timer.h"

#define PIN_SIGNAL_IN 2
#define OUTPUT_PIN 3 //frequency is toggled here

volatile int counter = 0;
volatile bool toggleState = LOW;
unsigned long curr_time = 0;
long currentFreq = 1250;

void CountFallingEdges(){
  counter++;
}
unsigned long CalculateFreq(int edge_counts, unsigned long time_interval){
  counter = 0; // reset counter
  return edge_counts * 1000.0 / time_interval;
}
void setup(){
  Serial.begin(9600);
  pinMode(OUTPUT_PIN, OUTPUT);
  pinMode(PIN_SIGNAL_IN, INPUT);
  ITimer1.init();
  ITimer1.setFrequency(currentFreq, TimerHandler);
  attachInterrupt(digitalPinToInterrupt(PIN_SIGNAL_IN), CountFallingEdges,
FALLING);
}
void loop(){
  // calculate detected frequency logic
  unsigned long interval = millis() - curr_time;
  if (interval > 2000){
    Serial.println(CalculateFreq(counter, interval));
    curr_time = millis(); // update to current time
  }
  //potentiometer logic
  int potValue = analogRead(A0);
  long newFreq = map(potValue, 0, 1023, 100, 25000); // ×2 for toggle
  if (newFreq != currentFreq){
      currentFreq = newFreq;
      ITimer1.setFrequency(currentFreq, TimerHandler);
  }
}
```

Final Code

(using potentiometer to change frequency of LED -> Reading the Output of the Comparator and Calculating Frequency)

```
#include "TimerInterrupt.h"
#include "ISR_Timer.h"

#define PIN_SIGNAL_IN 2
#define OUTPUT_PIN 3 //frequency is toggled here

volatile int counter = 0;
volatile bool toggleState = LOW;
unsigned long curr_time = 0;
long currentFreq = 1250;

void TimerHandler(){
  toggleState = !toggleState;
  digitalWrite(OUTPUT_PIN, toggleState);
}
void CountFallingEdges(){
  counter++;
}
unsigned long CalculateFreq(int edge_counts, unsigned long time_interval){
  counter = 0; // reset counter
  return edge_counts * 1000.0 / time_interval;
}
void setup(){
  Serial.begin(9600);
  pinMode(OUTPUT_PIN, OUTPUT);
  pinMode(PIN_SIGNAL_IN, INPUT);
  ITimer1.init();
  ITimer1.setFrequency(currentFreq, TimerHandler);
  attachInterrupt(digitalPinToInterrupt(PIN_SIGNAL_IN), CountFallingEdges,
FALLING);
}
void loop(){
  // calculate detected frequency logic
  unsigned long interval = millis() - curr_time;
  if (interval > 2000){
    Serial.println(CalculateFreq(counter, interval));
    curr_time = millis(); // update to current time
  }

  //potentiometer logic
  int potValue = analogRead(A0);
  long newFreq = map(potValue, 0, 1023, 100, 25000); // ×2 for toggle
  if (newFreq != currentFreq){
      currentFreq = newFreq;
      ITimer1.setFrequency(currentFreq, TimerHandler);
    }
}
```