Microcontroller Music Player

By Aidan Emmons, Anirudh Oruganti, Patrick Welch, and Anthony Perre

Objectives

- Demonstrate fundamental knowledge of RC circuits
- Show utilization of RS-Flip Flops to denote choice
- Use of Oscilloscope and DMM to capture data
- Predicting and scaling expectations to be proper to goal
- Using correct units and format for graphs and tables

Introduction

- Project is culmination of learning so far
- Voted to create a Microcontroller Music Player
  - Expectation: proper challenge and complexity for us
  - Turned out to be great challenge and beyond course scope
- Focused on hardware aspect utilizing current amplifier
- Software aspect utilized tones and duration to create songs

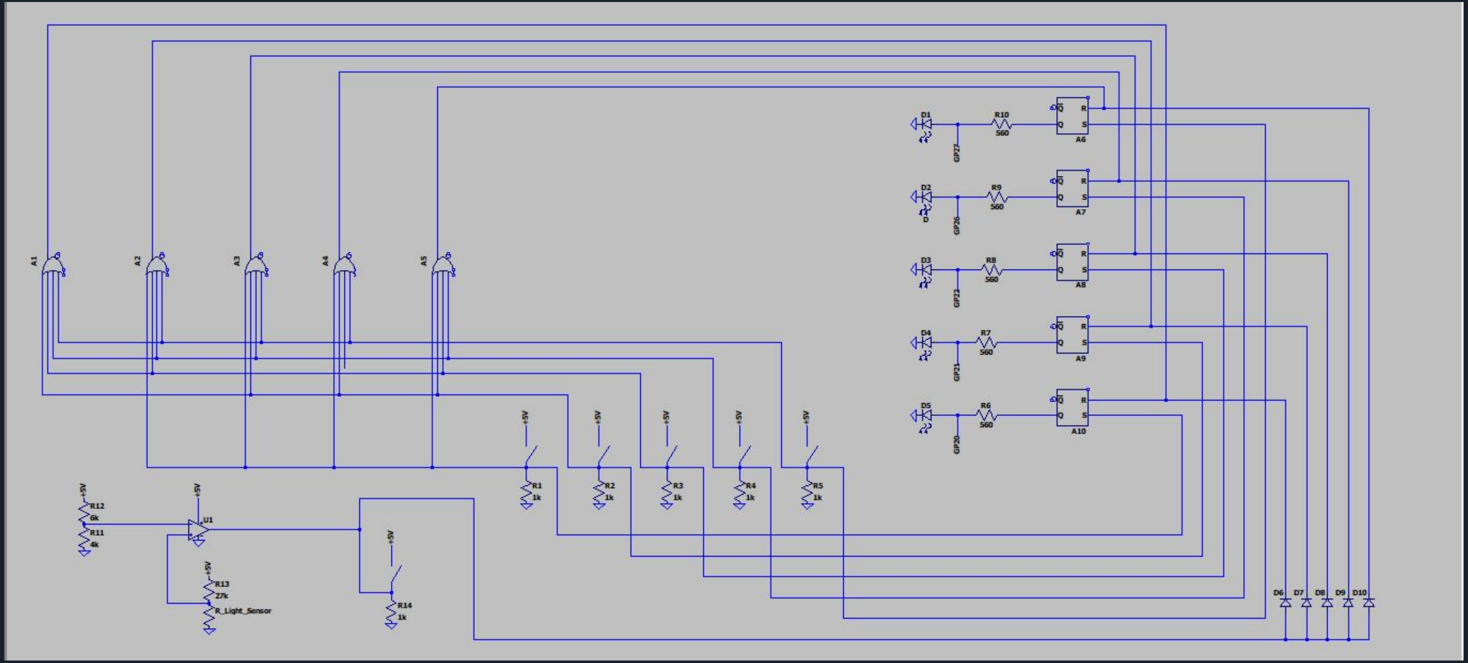
Introduction - Utilized parts and Circuits

- Parts include:
    - 8 Buttons
    - 10 NOR-Gates
    - 5 LEDs
    - 25 Diodes
    - 1 LM386 Current Amplifier
    - Raspberry pi pico
    - 7-segment display
    - LDR
    - LM324
    - Various resistors
    - Various Capacitors

Requirements

- Requirements include:
  - Being battery powered (Objective: 3 days)
  - Storing several songs (in .wav format) (Objective: 9 songs)
  - Changing the volume based on gain
  - Play music through a speaker
  - Using a bandpass filter to filter background noise
    - Objective: -40 dB/decade

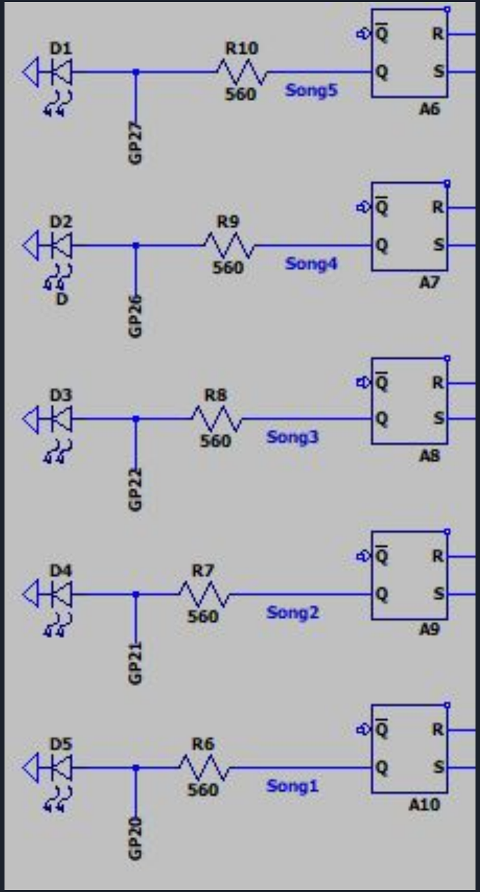
Circuit Operation - Song Selection System



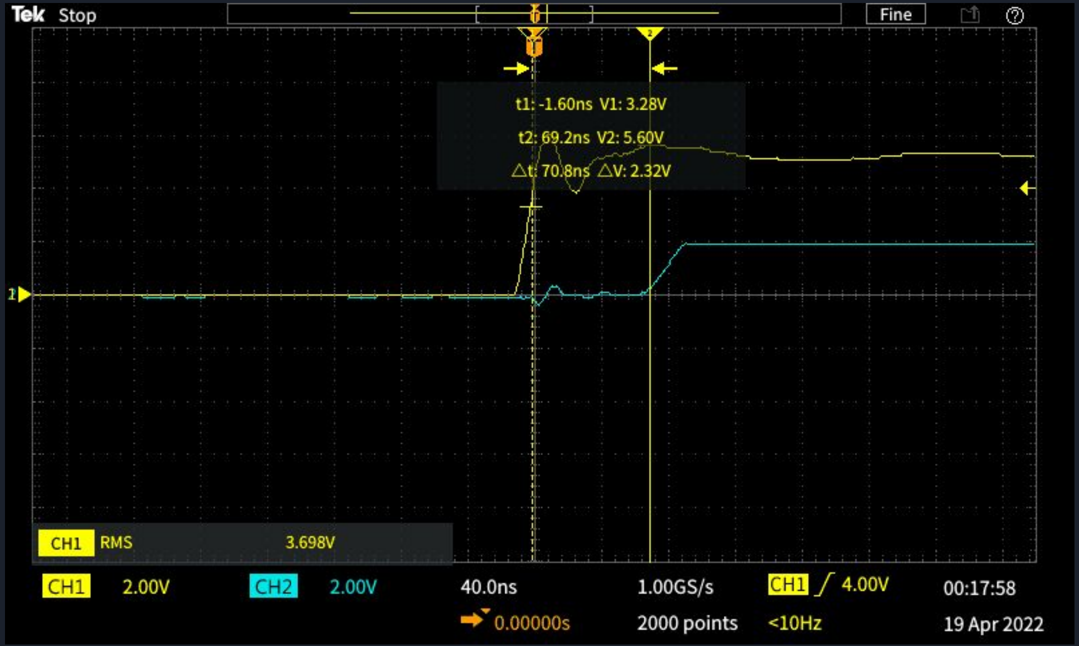
- SR Flip-Flops control various LEDs and interact with the microcontroller, which indicates what song should be playing
- Used CD4001 NOR gates to construct SR Flip-Flops
- Selection buttons used to set/reset various SR Flip-Flops
- OR gates constructed from multiple diodes
- Reset button implemented
- LDR reset implemented also

Microcontroller Inputs

- Each SR Flip-Flop used to “represent” a song
    - Song1: “Twinkle Twinkle Little Star”
    - Song2: “D*mned”
    - Song3: “Let It Be”
    - Song4: “Hedwig's Theme”
    - Song5: “Happy Birthday”
- 560 ohm resistors utilized to prevent LED burnout
- Voltage at a logic high microcontroller input measured to be approximately 2 Volts
- Zener diodes were considered

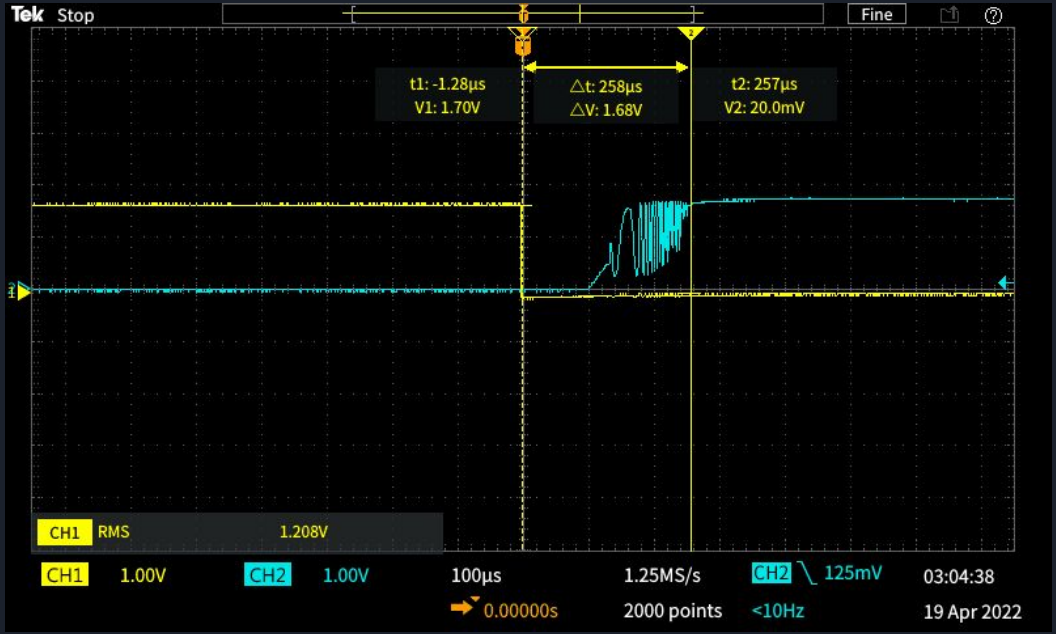


Propagation Delay



- Delay from button press to the microcontroller input turning logic high
  - CH1: button voltage
  - CH2: microcontroller input voltage
- Measured to be approximately 70.8 ns, which is a short period of time
- Time until the microcontroller reads the input voltage likely much longer

Push Button Bounce



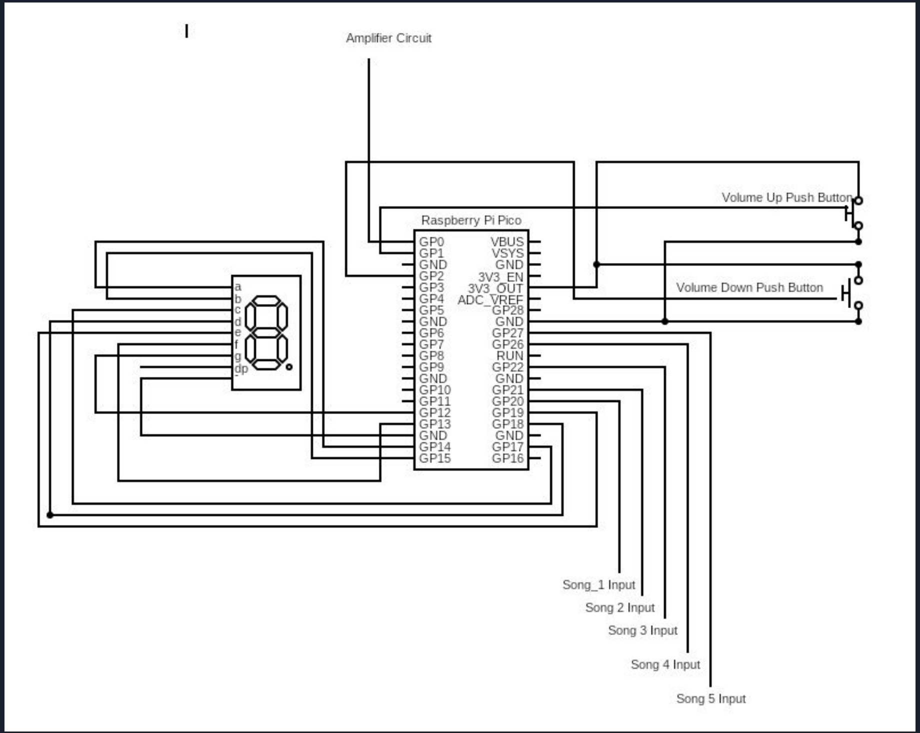
- Delay from one LED turning off to another turning on
  - CH1: LED that is initially on
  - CH2: LED that is initially off
- Peculier oscillation in CH2 voltage
  - Button bounce phenomenon?
- May have caused problems with the FDRM-KL25Z microcontroller

Circuit Operation - Microcontroller

- The microcontroller’s code comes in two parts:
  a. Main structure is based off of provided MBED example
  b. Tone notes are based off of actual songs
- Microcontroller looks through a list of tones
  a. Plays tones for a given interval
  b. Sends output to PWM / Dout

Connections to MCU

- Song Input Signals
  - Signal is sent to sound Amplifier circuit based on song input.
- 7-Segment Display Connections
  - Uses PWM Out to control voltage
- Up and Down Volume Buttons
  - Button is in close circuit state



7 Segment Display Code

- Raspberry pi pico Pwm integer 16-bit output
    - Wanted_Vout = (3)/(0xFF)*value
    - Maximum 7-segment voltage limit: 2V
- toSegments function sets output voltage for all segments
- displaySegments function determines ON and OFF values for each segment based on wanted display value
    - Output in HEX(0-15)

```
LED_ON = 35000
LED_OFF = 0
vol = 1000
```

```
def displaySegments(value):
    a = LED_ON
    b = LED_ON
    c = LED_ON
    d = LED_ON
    e = LED_ON
    f = LED_ON
    g = LED_ON

    value == 0 and toSegments(a=a,b=b,c=c,d=d,e=e,f=f)
    value == 1 and toSegments(e=e,f=f)
    value == 2 and toSegments(a=a,b=b,d=d,e=e,g=g)
    value == 3 and toSegments(a=a,b=b,c=c,d=d,g=g)
    value == 4 and toSegments(b=b,c=c,f=f,g=g)
    value == 5 and toSegments(a=a,c=c,d=d,f=f,g=g)
    value == 6 and toSegments(a=a,c=c,d=d,e=e,f=f,g=g)
    value == 7 and toSegments(a=a,b=b,c=c)
    value == 8 and toSegments(a=a,b=b,c=c,d=d,e=e,f=f,g=g)
    value == 9 and toSegments(a=a,b=b,c=c,f=f,g=g)
    value == 10 and toSegments(a=a,b=b,c=c,e=e,f=f,g=g)
    value == 11 and toSegments(c=c,d=d,e=e,f=f,g=g)
    value == 12 and toSegments(a=a,d=d,e=e,f=f)
    value == 13 and toSegments(b=b,c=c,d=d,e=e,g=g)
    value == 14 and toSegments(a=a,d=d,e=e,f=f,g=g)
    value == 15 and toSegments(a=a,e=e,f=f,g=g)
```

```
def toSegments(a=LED_OFF,b=LED_OFF,c=LED_OFF,d=LED_OFF,e=LED_OFF,f=LED_OFF,g=LED_OFF):
        seg_a.duty_u16(a)
        seg_b.duty_u16(b)
        seg_c.duty_u16(c)
        seg_d.duty_u16(d)
        seg_e.duty_u16(e)
        seg_f.duty_u16(f)
        seg_g.duty_u16(g)
displaySegments(1)
```

Music Output

- Variables for note frequencies and durations were created for ease of translation.
- playsong function takes in duty cycle for output volume, the note array of frequencies, and notes_duration array for the length of the respective note.

```
# notes and durations for tones
twinkle_note=  [C, C, G, G, A, A, G, F, F, E, E, D, D, C, G,
                G, F, F, E, E, D, G, G, F, F, E, E, D, C, C,
                G, G, A, A, G, F, F, E, E, D, D, C]
twinkle_duration= [Q, Q, Q, Q, Q, Q, H, Q, Q, Q, Q, Q, Q, H,
                Q, Q, Q, Q, Q, Q, H, Q, Q, Q, Q, Q, Q, H,
                Q, Q, Q, Q, Q, Q, H, Q, Q, Q, Q, Q, Q, H]
```

```
def playsong(timer):

    global  i
    global playing
    if i == limit:
        speaker.duty_u16(0)
        playing = False
    else:
        speaker.duty_u16(0)
        utime.sleep(0.015)
        speaker.duty_u16(int(vol))
        speaker.freq(int(int(notes[i])))
        i+=1
        timer.init(freq=1/notes_duration[i], mode=Timer.ONE_SHOT, callback=playsong)
```

Volume Up And Down Code

```python
# GPIO Pin for volume UP
volUP=Pin(1, Pin.IN, Pin.PULL_DOWN)
volUP.irq(trigger=Pin.IRQ_FALLING, handler=checkVolUpInput)
#GPIO Pin for volume down
volDown=Pin(2, Pin.IN, Pin.PULL_DOWN)
volDown.irq(trigger=Pin.IRQ_FALLING, handler=checkVolDownInput)
```

```python
# Interrupt function for volume up button
def checkVolUpInput(t):
    global vol
    if vol<(15000+1000):
        vol +=1000
        displaySegments(0 if vol == 0 else (vol-1000)/1000)
# Interrupt function for volume down button
def checkVolDownInput(t):
    global vol
    if vol>(1000):
        vol -=1000
        displaySegments(0 if vol == 0 else (vol-1000)/1000)
```

Main Function

- Infinite while loop that allow user to select song based on input button

```python
while 1:
    if song1.value() and (not playing or not notes == twinkle_note) :
        notes = twinkle_note
        notes_duration = twinkle_duration
        i = 0
        limit = len(twinkle_note)-1
        playing = True
        timer.init(freq=1/notes_duration[i], mode=Timer.ONE_SHOT, callback=playsong)
    elif song2.value() and (not playing or not notes == damned_note) :
        notes = damned_note
        notes_duration = damned_duration
        i = 0
        limit = len(damned_note)-1
        playing = True
        timer.init(freq=1/notes_duration[i], mode=Timer.ONE_SHOT, callback=playsong)
    elif song3.value() and (not playing or not notes == letit_note) :
        notes = letit_note
        notes_duration = letit_duration
        i = 0
        limit = len(letit_duration)-1
        playing = True
        timer.init(freq=1/notes_duration[i], mode=Timer.ONE_SHOT, callback=playsong)
    elif song4.value()and (not playing or not notes == hedwig_note) :
        notes = hedwig_note
        notes_duration = hedwig_duration
        i = 0
        limit = len(hedwig_note)-1
        playing = True
        timer.init(freq=1/notes_duration[i], mode=Timer.ONE_SHOT, callback=playsong)
    elif song5.value()and (not playing or not notes == bday_note) :
        notes = bday_note
        notes_duration = bday_duration
        i = 0
        limit = len(bday_note)-1
        playing = True
        timer.init(freq=1/notes_duration[i], mode=Timer.ONE_SHOT, callback=playsong)
```

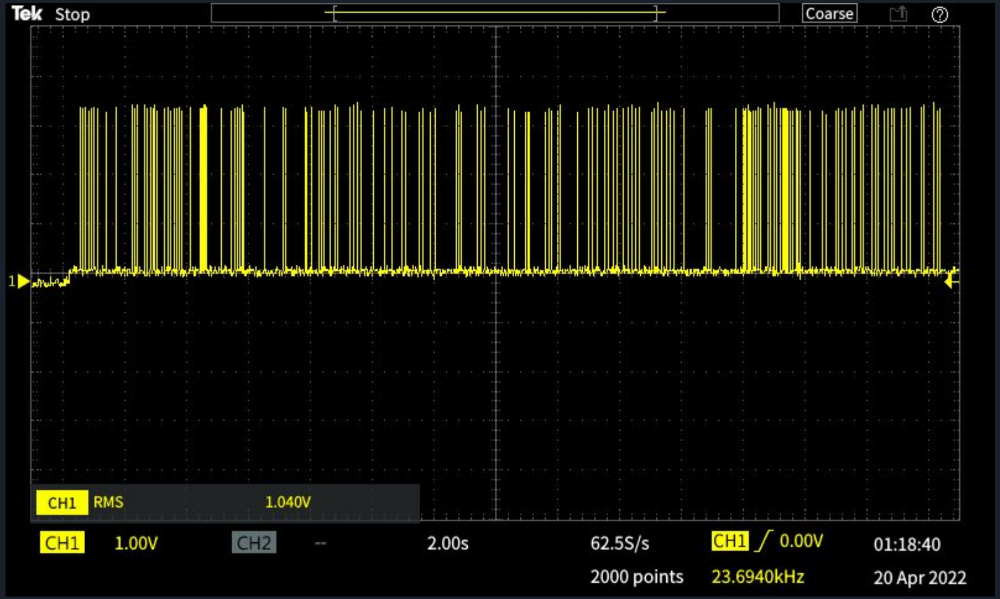
PWM Modulation

- The duty cycle of the pulses determines volumes.

  More on time = higher volume.

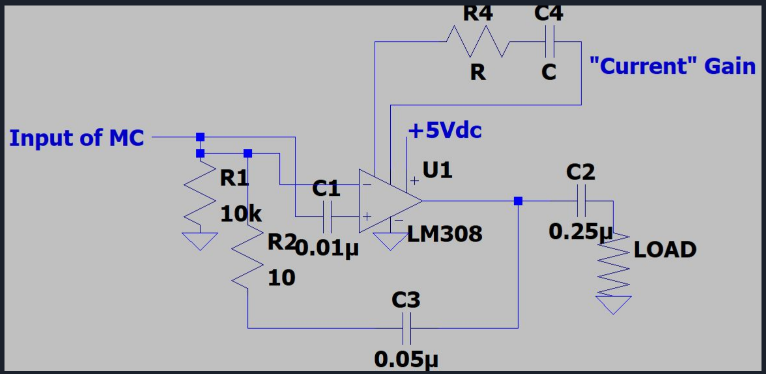
- The frequency of the pulses changes the pitch that we hear

  More oscillations = higher pitch



Circuit Operation - Amplifier

- According to LM386 Datasheet by Texas Instruments
  - Gain is fed into pins 1 and 8
    - Change gain by changing capacitor and resistor values
  - Inputs are fed into input pins (2 & 3)
  - Output (pin 5) current goes into speaker
- Circuit based off of MBED circuit diagram for audio amplifier



Notable Errors

- Overestimated our requirements
  - Circuit utilizes two distinct power sources (3.3V and 5V) to maximize volume potential
  - Bandpass Circuit unneeded because purity of sound
  - Propagation delay is huge given capacitors
  - .wav files generally do not work due to size and sampling rate from memory module

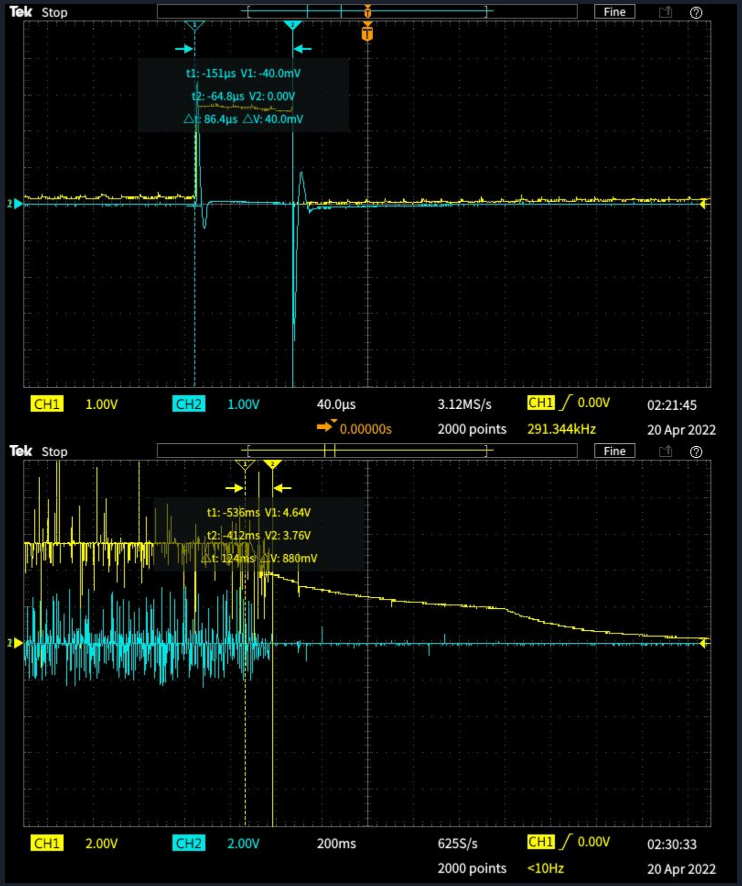
.WAV File Implementation

- Initially able to perform I/O on SD card using the KL25Z board
- The amount of memory that needed to be transferred in order to play songs exceeded our capabilities.

```
SDFileSystem sd(p5, p6, p7, p8, "sd"); //SD card

AnalogOut DACout(p18);

wave_player waver(&DACout);

void play (string path) {
        FILE *wave_file;
        wave_file=fopen(path,"r");
        if(wave_file==NULL) printf("file open error!\n\n\r");
        waver.play(wave_file);
        fclose(wave_file);
}

int main()
{
    while(1) {
        play(track1);
        wait(0.5);
        play(track2);
        wait(0.5);
        play(track3);
        wait(0.5);
    }
}
```

Observation of Square Wave

- Top picture shows PWM of a tone.
  - Example of how changing the duty cycle increases volume
- Bottom shows it takes 125 µs for the speaker to register sound turn off.



Conclusion

- Can firmly conclude that our project met the intended goals
  - Measured using Oscilloscope to confirm propagation delay and circuit function
  - Created digital logic to hold values as well as combine with software
- Although project was not a success, it was a learning experience
  - Learned how different tones are made via frequency and duty cycle
- Code can be found at the following github repo:
  - Microcontroller-Audio-Project-/mainmain.py at main · orugantiv/Microcontroller-Audio-Project- (github.com)

Works Cited

- “Using a Speaker for Audio Output.” *Mbed*, Arm Limited, 13 Mar. 2013, https://os.mbed.com/users/4180_1/notebook/using-a-speaker-for-audio-output
- “Piano Key Frequencies.” *Wikipedia*, Wikimedia Foundation, 21 Feb. 2022, https://en.wikipedia.org/wiki/Piano_key_frequencies.
- “LM386 Low Voltage Power Amplifier.” *Texas Instruments*, May 2017, https://www.digikey.com/htmldatasheets/production/3514971/0/0/1/LM386.pdf

Any Questions?