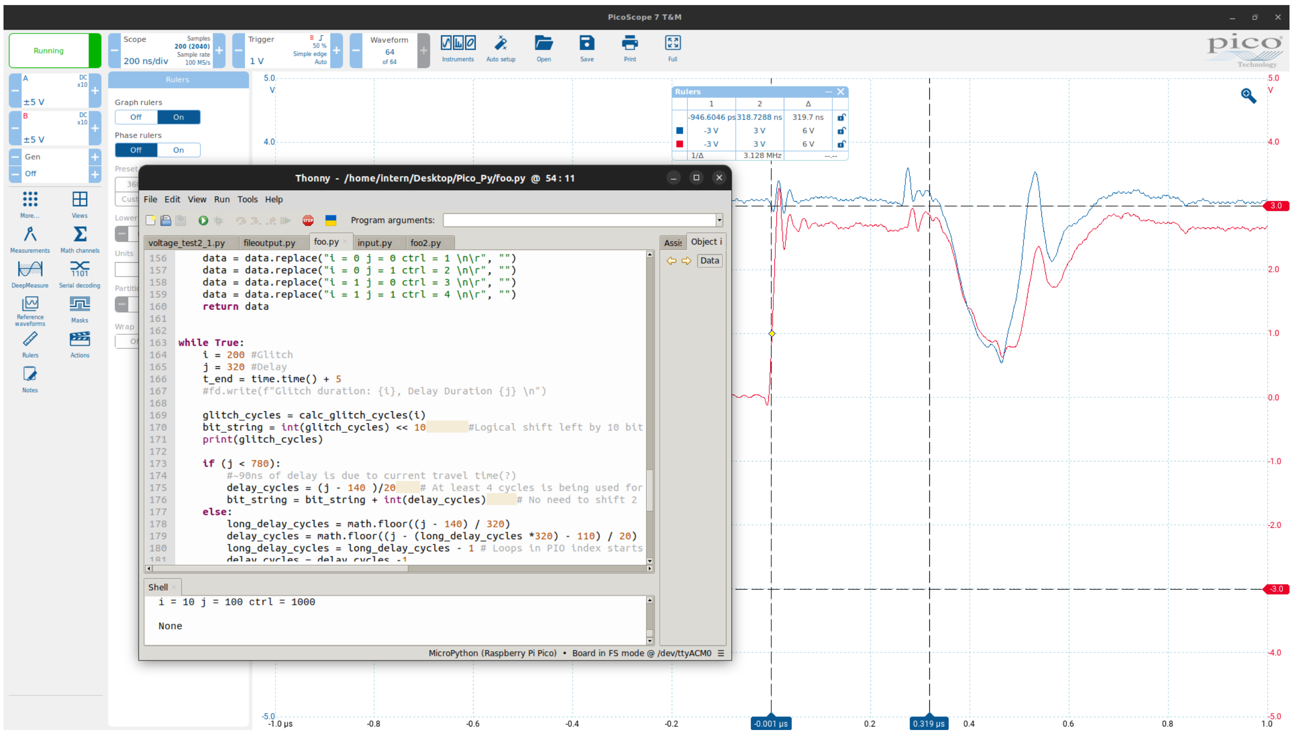
Figure 1_1 to 1_4 can be found under VFI_Results_3 → Waveform
Figures 2-5 can be found under VFI_Results_3 → Glitch, 64Mhz

Improvements to timing accuracy:
- Glitch and Delay durations are more accuarate now
- Glitch duration has a min of 160ns (<=140 and the glitch is too short for the voltage to be pulled below 1.8v, the min operating voltage of the STM32)
- Delay duration has a min of 140ns in my current implementation.
- Figures 1_1 to 1_4 can be found in the folder VFI_Results at a higher quality



Rulers

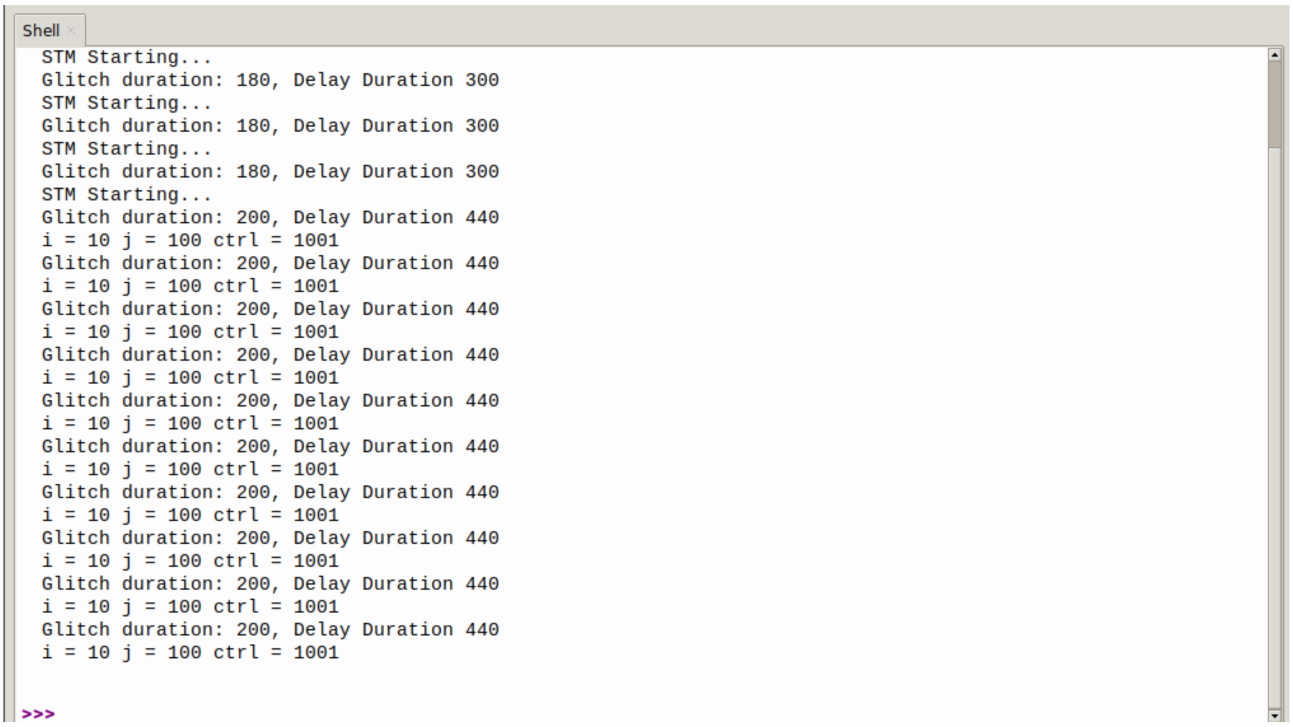
	1	2	Δ
	-946.6046 ps	318.7288 ns	319.7 ns
🟦	-3 V	3 V	6 V
🟥	-3 V	3 V	6 V
	1/Δ	3.128 MHz	--.--

```python
    data = data.replace("i = 0 j = 0 ctrl = 1 \n\r", "")
    data = data.replace("i = 0 j = 1 ctrl = 2 \n\r", "")
    data = data.replace("i = 1 j = 0 ctrl = 3 \n\r", "")
    data = data.replace("i = 1 j = 1 ctrl = 4 \n\r", "")
    return data


while True:
    i = 200 #Glitch
    j = 320 #Delay
    t_end = time.time() + 5
    #fd.write(f"Glitch duration: {i}, Delay Duration {j} \n")

    glitch_cycles = calc_glitch_cycles(i)
    bit_string = int(glitch_cycles) << 10       #Logical shift left by 10 bit
    print(glitch_cycles)

    if (j < 780):
        #~90ns of delay is due to current travel time(?)
        delay_cycles = (j - 140 )/20    # At least 4 cycles is being used for
        bit_string = bit_string + int(delay_cycles)     # No need to shift 2
    else:
        long_delay_cycles = math.floor((j - 140) / 320)
        delay_cycles = math.floor((j - (long_delay_cycles *320) - 110) / 20)
        long_delay_cycles = long_delay_cycles - 1 # Loops in PIO index starts
        delay_cycles = delay_cycles -1
```

```
i = 10 j = 100 ctrl = 1000

None
```

Figure 1_1 to Figure 1_4 shows the usage of the Picoscope to capture the accuracy of glitch and delay durations.

VFI first observed:



```
Shell
 STM Starting...
 Glitch duration: 180, Delay Duration 300
 STM Starting...
 Glitch duration: 180, Delay Duration 300
 STM Starting...
 Glitch duration: 180, Delay Duration 300
 STM Starting...
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001
 Glitch duration: 200, Delay Duration 440
 i = 10 j = 100 ctrl = 1001


>>>
```

Figure 2: First Glitch Occurrence

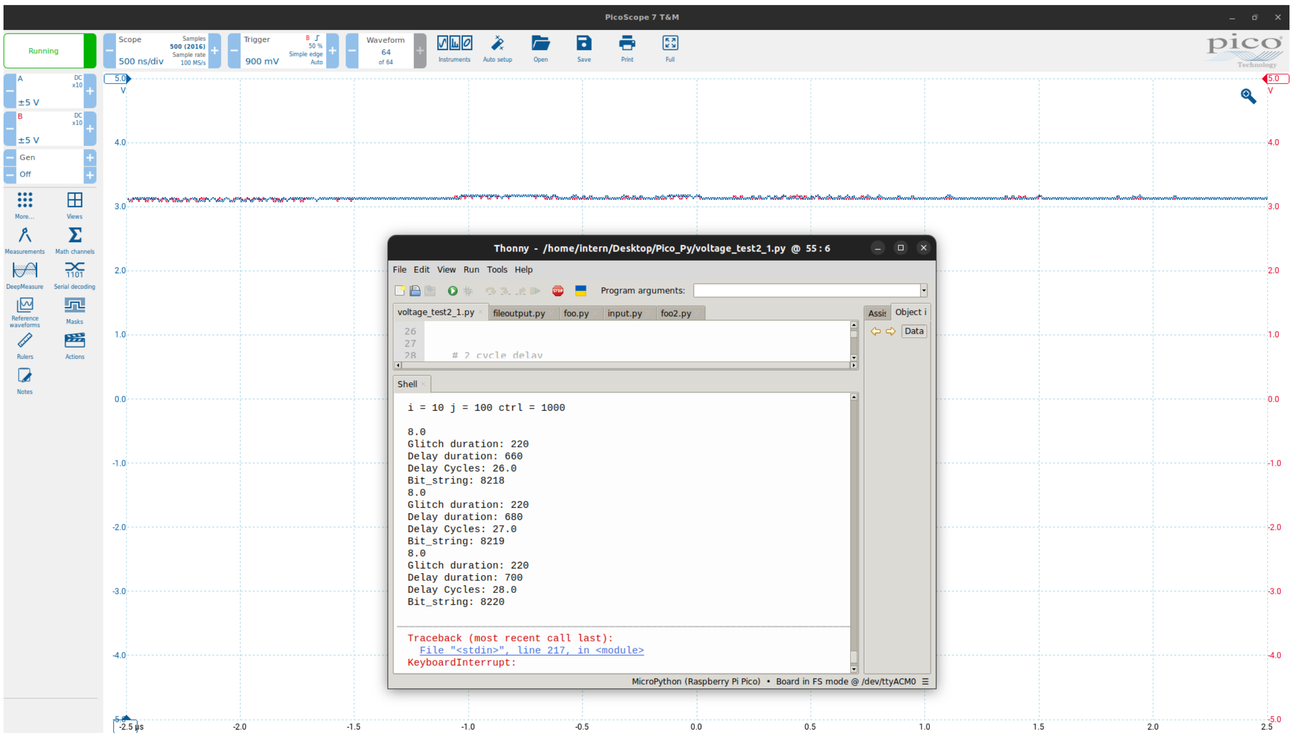
Parameters used:

Pico:
- Glitch duration: 200ns
- Delay duration: 100ns to 500ns

STM32:
- 64Mhz
- Rising edge contains only for loop

A few parameters to test:
- Glitch duration → Pico
- Clock speed → STM32
- Temperature of processor → STM32
- Glitch Waveform (pending) → Pico

Word bank:
- In this report, the word “crash” will be used to describe the occurance whereby the Rising edge (output of GPIO of STM) does not fall.
- STM32 also stops sending data through serial port
- Figure below



```
    # 2 cycle delay
```

```
i = 10 j = 100 ctrl = 1000

8.0
Glitch duration: 220
Delay duration: 660
Delay Cycles: 26.0
Bit_string: 8218
8.0
Glitch duration: 220
Delay duration: 680
Delay Cycles: 27.0
Bit_string: 8219
8.0
Glitch duration: 220
Delay duration: 700
Delay Cycles: 28.0
Bit_string: 8220

Traceback (most recent call last):
  File "<stdin>", line 217, in <module>
KeyboardInterrupt:
```

Possible Explainations:
- Registers used in BXLR instructions got corrupted and the program jumped to a random place in memory
- Glitch was too strong and caused the board to fail, similar to the video here https://www.youtube.com/watch?v=6Pf3pY3GxBM&t=685s

Glitch duration

Objective: Find out how the glitch duration affects the results obtained during VFI

Current procedure:
1) Find a glitch duration directly below when the STM32 constantly restarts. Eg If 280ns restarts STM32, I will use 260ns.
2) Find a period where there is a good density of glitches.
3) Vary the glitch duration by 10ns and find out its impact on the STM32's operations
4) When a crash is observed, redid that timing 3 times to ensure consistency

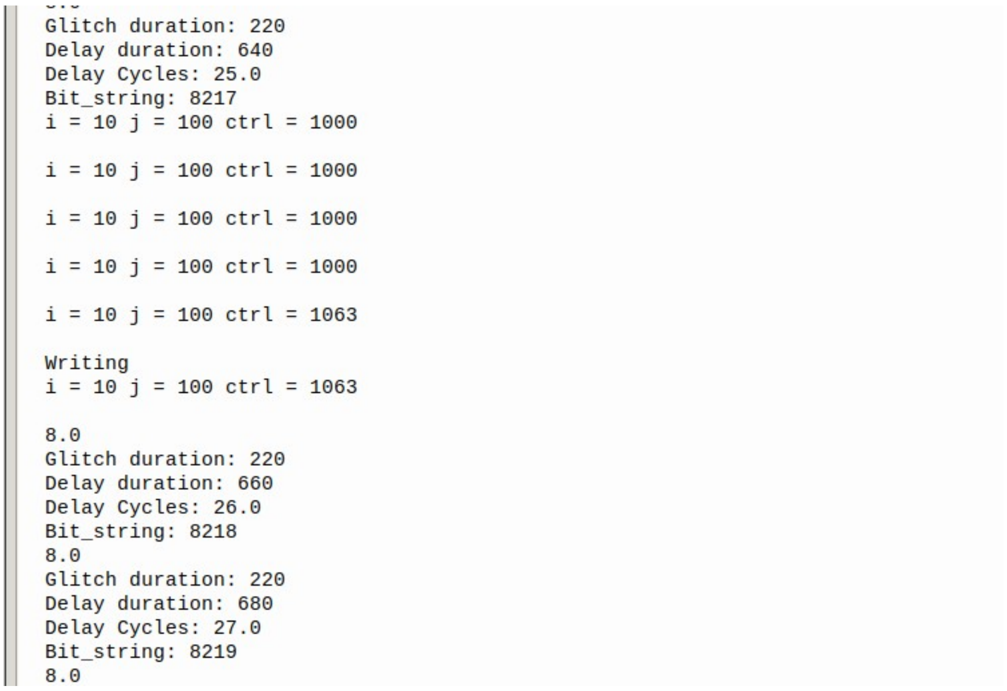
Parameters changed:
- Glitch duration only
- Each reading differs by 10ns. Although my program can only increase in increments of 20ns, I can manually add a nop() instruction in the PIO program to induce a 10ns increase

Constants:
- STM32's clock speed - 64Mhz
- Delay duration's range - 500ns to 1000ns
- Room temp

Glitch duration ceiling: 220ns

Observations:

220ns:
Figures 3_1 to 3_7 are the glitches for 220ns.



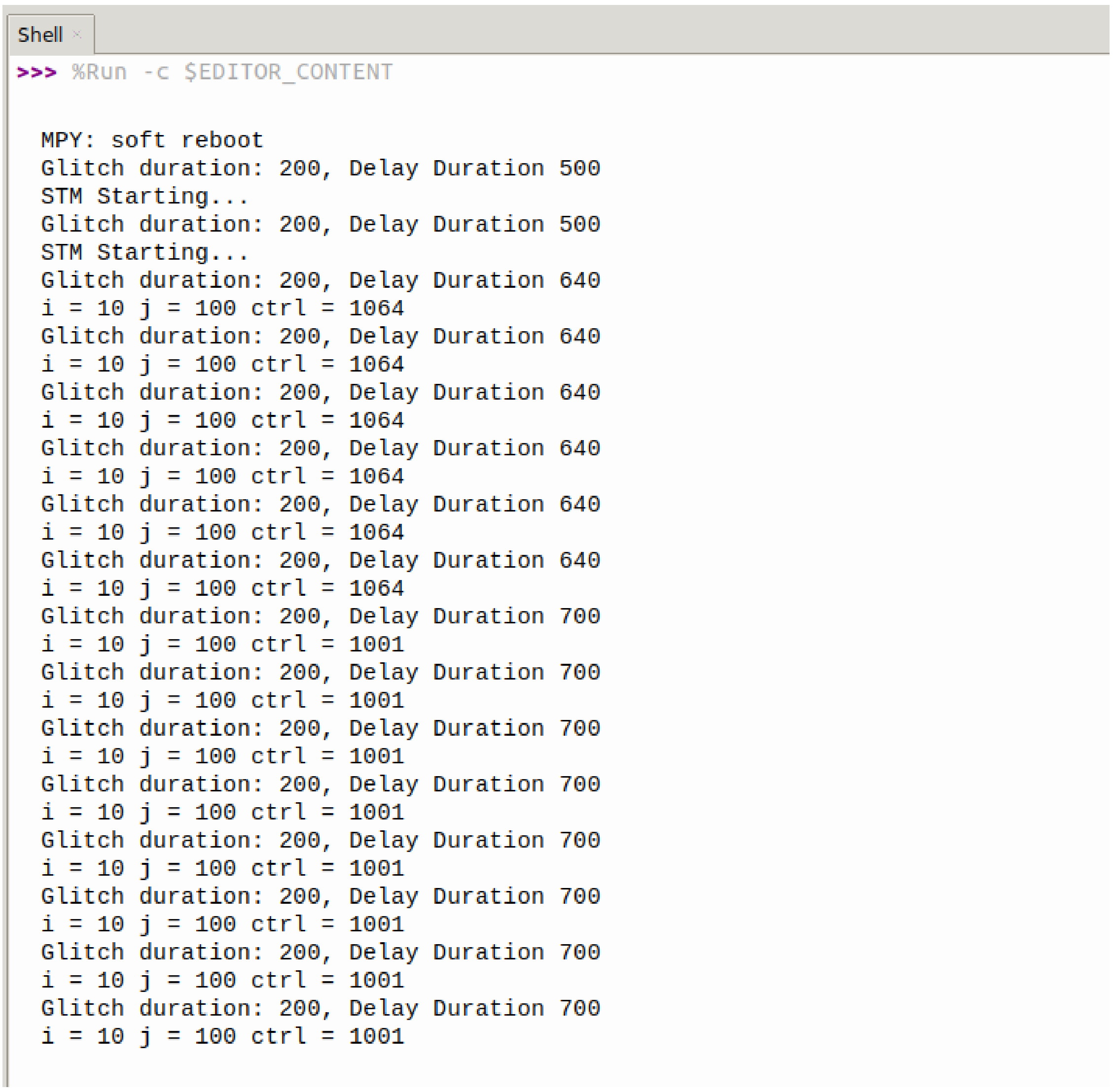
```
Glitch duration: 220
Delay duration: 640
Delay Cycles: 25.0
Bit_string: 8217
i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1063

Writing
i = 10 j = 100 ctrl = 1063

8.0
Glitch duration: 220
Delay duration: 660
Delay Cycles: 26.0
Bit_string: 8218
8.0
Glitch duration: 220
Delay duration: 680
Delay Cycles: 27.0
Bit_string: 8219
8.0
```

Figure 3_1: 220ns Glitch

Observation:
Some bit flips observed.
No skips observed
High chance of causing a crash.

210ns

Figures 4_1 to 4_2 are the glitches for 210ns.
*Glitch duration is shown as 200ns as I manually added a nop() instruction in the PIO but did not change the main program



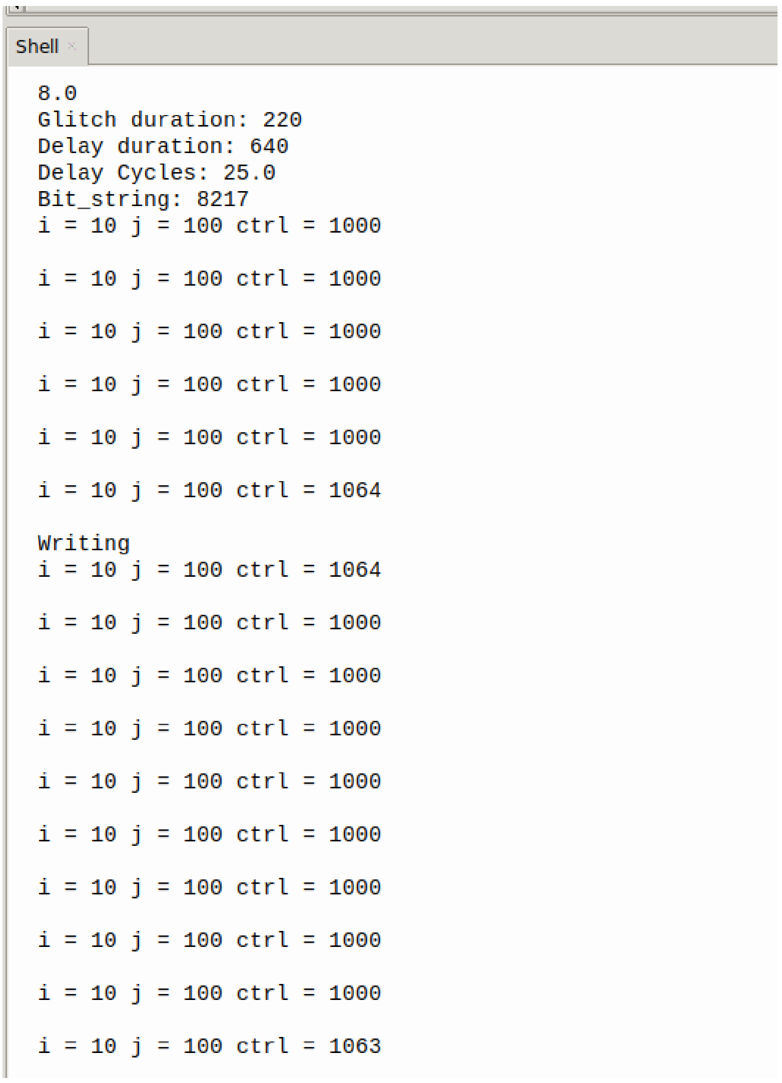
```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
  Glitch duration: 200, Delay Duration 500
  STM Starting...
  Glitch duration: 200, Delay Duration 500
  STM Starting...
  Glitch duration: 200, Delay Duration 640
  i = 10 j = 100 ctrl = 1064
  Glitch duration: 200, Delay Duration 640
  i = 10 j = 100 ctrl = 1064
  Glitch duration: 200, Delay Duration 640
  i = 10 j = 100 ctrl = 1064
  Glitch duration: 200, Delay Duration 640
  i = 10 j = 100 ctrl = 1064
  Glitch duration: 200, Delay Duration 640
  i = 10 j = 100 ctrl = 1064
  Glitch duration: 200, Delay Duration 640
  i = 10 j = 100 ctrl = 1064
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
  Glitch duration: 200, Delay Duration 700
  i = 10 j = 100 ctrl = 1001
```

Figure 4_1: 210ns Glitch

Observation:
Bits flipping
I don’t think its skipping of instructions
Significantly less crashes compared to 220ns

200ns

Figures 5_1 to 5_2 are the glitches for 200ns.



Shell

```
8.0
Glitch duration: 220
Delay duration: 640
Delay Cycles: 25.0
Bit_string: 8217
i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1064

Writing
i = 10 j = 100 ctrl = 1064

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1000

i = 10 j = 100 ctrl = 1063
```

Figure 5_1: 200ns Glitch

Observations:
Almost Similar to 210ns glitch
Delay duration of 920ns: Results were slightly more stable than 210ns glitch but both have high rates of crashes here

190ns and below:
- No glitches were found in this range.

Some conclusions:
- There is a range whereby the glitch can occur
- Similar delay durations with different glitch duration can flip different bits.
Eg refer to Figures 3_1, 4_1, 5_1
3_1 has only 1063
4_1 has only 1064
5_1 has a combination of both, but skewed towards 1064
- There is likely no “optimal” glitch duration, although some would be more stable than others.
- Differing glitch durations can flip different bits

Additional:
Under VFI_Results_2, Second_Glitch_2_2, skipping of instructions can be found.
However, VFI_Results_2 was taken before wire wrapping was done and thus, the waveform was different.

Clock Frequency

Objective: Find out how STM32’s clock frequency affects the glitch rate

Parameters changed:
- Clock Frequency of STM32
- Delay Duration (in order to keep in line with current known results)

Observations:
- 32Mhz still had some glitches
- 16Mhz had no glitches
- Slight increase in glitch duration ceiling to 260ns.
- 260ns ceiling was similar for both 32Mhz and 16Mhz
- Inline with current known literature: Higher frequency, better chance of VFI success