REQUIREMENTS NOT MET

I didn't finish section 3 or section 5.

PROBLEMS ENCOUNTERED

I got stuck for the longest time on section 3. I didn't know what other even we were supposed to have so I could never get that section to work. After being stuck on it for a long time I finally decided to see if I could do section 4 without section 3. I did, but then I figured that section 5 is the last section and would be building off of everything including 3 and I just hadn't gotten that to work yet.

FUTURE WORK/APPLICATIONS

Without even finishing the lab, I can already see how important DMA is. To be able to output data like we did without having to take time away from the CPU is an incredible feat. This is a tool that will be used to optimize code and make it better overall.

PRE-LAB EXERCISES

i. Why might you be unable to generate a desired frequency with this method of using an interrupt? Refer to the disassembly of the interrupt service routine. Additionally, temporarily change the optimization level of your compiler to -O1. Are the results any different? Why or why not?

I must have done something wrong because I did reach the desired frequency. My results are not different after O-1. I would say this is because we are already using Timer counters and interrupts so it is already on a set schedule where you can fine tune the frequency. There is very little to optimize that would affect the counter.

ii. Would a method of synchronous polling (i.e., a method with no interrupts) result in the same issue identified in the previous exercise? In other words, would the desired frequency not initially met now be achieved? Alter your program to check your answer, and then take a screenshot of the waveform generated, again denoting a precise frequency measurement of this waveform within the screenshot.

Again, I had already met the desired frequency before I used polling. However, when I changed to polling, my frequency went up. Also, when I changed to O-1, it went up even higher. Screenshot in Appendix.

iii. What is the correlation between the amount of data points used to recreate the waveform and the overall quality of the waveform?

If you have more data points, your curve will be finer tuned. There will be even more “steps” in your sine wave, but they will have smaller increments of voltage between them.

PSEUDOCODE/FLOWCHARTS

N/A

PROGRAM CODE

SECTION 1 (Initializing DAC)

```
#include <avr/io.h>



extern void clock_init(void);


int main(void)
{
	clock_init();
	dac_init();


    while (1)
    {
		/* Ensure the CH0DATA register is empty before starting a new conversion. */
		while(!(DACA.STATUS & DAC_CH0DRE_bm));

		/* Write digital data to the CH0DATA register. 0 = 0V, 0xFFF = VREF. */
		DACA.CH0DATA = 0x999;	//just used formula in section 29
    }
}


void dac_init(void)
{
	/* Use only channel 0 */
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc;

	/* Use AREFB (2.5V). Data is right-adjusted. */
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;

	/* Enable channel 0, as well as the overall DAC module. */
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
}
```

SECTION 2 (Had different parts. Consisted of generating a sine wave from a LUT different ways at different frequencies.)

Lab8_2a

```
#include <avr/io.h>
#include <avr/interrupt.h>

extern void clock_init(void);


int16_t array_position;

int main(void)
{
	clock_init();
	dac_init();
	tcc0_init();
	intr_init();

	array_position = 0;


    while (1)
    {}
}



void dac_init(void)
{
	/* Use only channel 0 */
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc;

	/* Use AREFB (2.5V). Data is right-adjusted. */
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;

	/* Enable channel 0, as well as the overall DAC module. */
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
}

void intr_init(void)
{
	/* Enable low level interrupts in the PMIC. */
	PMIC.CTRL = PMIC_LOLVLEN_bm;

	/* Enable interrupts globally. */
	sei();
}


void tcc0_init(void)
{
	//;set TCC0 period register
	// we want sine wave with a 1000Hz frequency
	// but we have 256 values for each frequency
	// so our PER = (1/1000Hz) /256 = 1/256000
	//;TCC0_PER = (fclk/prescalar) * (duration in seconds)
	//;                          32MH/1                              1/256000

	TCC0.CNT = 0;
```

```
	TCC0.PER = (32000000/1) / 256000; // = 125
	TCC0.CTRLA = TC_CLKSEL_DIV1_gc;

	/* Clear OVFIF to be safe */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	/* Set as low level interrupt */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;
}


ISR(TCC0_OVF_vect)
{
	//taking out the section below gets us to our target range of 1000Hz
	///* Ensure the CH0DATA register is empty before starting a new conversion. */
	//while(!(DACA.STATUS & DAC_CH0DRE_bm));

	/* Write digital data to the CH0DATA register. 0 = 0V, 0xFFF = VREF. */
	DACA.CH0DATA = sine[array_position];
	array_position++;

	if(array_position == 256)
	{
		array_position = 0;
	}
}
```

For polling and not using an ISR, main was the only thing that really changed:

```
int main(void)
{
	clock_init();
	dac_init();
	tcc0_init();

	array_position = 0;


	while (1)
	 {
		while(!(TCC0.INTFLAGS & TC0_OVFIF_bm));

		/* Write digital data to the CH0DATA register. 0 = 0V, 0xFFF = VREF. */
		DACA.CH0DATA = sine[array_position];
		array_position++;

		if(array_position == 256)
		{
		array_position = 0;
		}
	}
}
```

Lab8_2b:

```
#include <avr/io.h>
#include <avr/interrupt.h>

extern void clock_init(void);


int16_t array_position;


int main(void)
{
	clock_init();
	dac_init();
	tcc0_init();
	intr_init();

	array_position = 0;


    while (1)
    {
		while(!(DACA.STATUS & DAC_CH0DRE_bm));

		/* Write digital data to the CH0DATA register. 0 = 0V, 0xFFF = VREF. */
		DACA.CH0DATA = sine[array_position];
		array_position++;

		if(array_position == 256)
		{
		array_position = 0;
		}
    }
}



void dac_init(void)
{
	/* Use only channel 0 */
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc | DAC_CH0TRIG_bm;

	/* Use AREFB (2.5V). Data is right-adjusted. */
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;

	/* Enable channel 0, as well as the overall DAC module. */
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;

	/* make a DAC conversion start when Event Channel 0 is triggered */
	DACA.EVCTRL = DAC_EVSEL_0_gc;
}

void intr_init(void)
{
	/* Enable low level interrupts in the PMIC. */
	PMIC.CTRL = PMIC_LOLVLEN_bm;

	/* Enable interrupts globally. */
	sei();
}
```

```
void tcc0_init(void)
{
	//;set TCC0 period register
	// we want sine wave with a 1000Hz frequency
	// but we have 256 values for each frequency
	// so our PER = (1/1760Hz) /256 = 1/256000
	//;TCC0_PER = (fclk/prescalar) * (duration in seconds)
	//;                          32MH/1                              1/459560
	//this didn't give a value withing 2% so I manually changed the PER

	TCC0.CNT = 0;
	TCC0.PER = 72;//(32000000/1) / 459560; // = 69 but didn't work
	TCC0.CTRLA = TC_CLKSEL_DIV1_gc;

	/* Clear OVFIF to be safe */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	/* Set as low level interrupt */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
}
```

SECTION 3 (Using DMA)

This section is not finished. I feel like it is close but I could not get it to work properly.

```
#include <avr/io.h>
#include <avr/interrupt.h>

extern void clock_init(void);


int16_t array_position;

int main(void)
{
	clock_init();
	tcc0_init();
	dma_init();
	dac_init();
	intr_init();



    while (1)
    {
    }
}



void dac_init(void)
{
	/* Use only channel 0 */
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc | DAC_CH0TRIG_bm;

	/* Use AREFB (2.5V). Data is right-adjusted. */
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;

	/* Enable channel 0, as well as the overall DAC module. */
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;

	/* make a DAC conversion start when Event Channel 0 is triggered */
	DACA.EVCTRL = DAC_EVSEL_0_gc;
}

void intr_init(void)
{
	/* Enable low level interrupts in the PMIC. */
	PMIC.CTRL = PMIC_LOLVLEN_bm;

	/* Enable interrupts globally. */
	sei();
}


void tcc0_init(void)
{
	//;set TCC0 period register
	// we want sine wave with a 1000Hz frequency
	// but we have 256 values for each frequency
	// so our PER = (1/1760Hz) /256 = 1/256000
	//;TCC0_PER = (fclk/prescalar) * (duration in seconds)
```

```
	//;                          32MH/1                              1/459560
	//this didn't give a value withing 2% so I manually changed the PER

	TCC0.CNT = 0;
	TCC0.PER = 72; //(32000000/1) / 459560; // = 69 but didn't work
	TCC0.CTRLA = TC_CLKSEL_DIV1_gc;

	/* Clear OVFIF to be safe */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	/* Set as low level interrupt */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
}


void dma_init(void)
{
	/* reset entire DMA periph */
	DMA.CTRL |= DMA_RESET_bm;

	DMA.CH0.CTRLA = DMA_CH_SINGLE_bm | DMA_CH_BURSTLEN_2BYTE_gc; // I think I need to change this to
2byte

	DMA.CH0.ADDRCTRL = DMA_CH_SRCRELOAD_BLOCK_gc | DMA_CH_SRCDIR_INC_gc | DMA_CH_DESTRELOAD_NONE_gc |
DMA_CH_DESTDIR_FIXED_gc;

	DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_DACA_CH0_gc;//DMA_CH_TRIGSRC_DACA_CH0_gc  or
DMA_CH_TRIGSRC_TCC0_OVF_gc change to TC interrupt

	DMA.CH0.TRFCNT = (uint16_t)(sizeof(sine));

	DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)sine);
	DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)sine) >> 8);
	DMA.CH0.SRCADDR2 = (uint8_t)(((uint32_t)((uintptr_t)sine)) >> 16);

	DMA.CH0.DESTADDR0 = (uint8_t)((uintptr_t)&DACA.CH0DATA);
	DMA.CH0.DESTADDR1 = (uint8_t)(((uintptr_t)&DACA.CH0DATA) >> 8);
	DMA.CH0.DESTADDR2 = (uint8_t)(((uint32_t)((uintptr_t)&DACA.CH0DATA)) >> 16);



	/* enable CH0 */
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;

	/* enable entire DMA periph */
	DMA.CTRL = DMA_ENABLE_bm;

}



ISR(TCC0_OVF_vect)
{
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;

	DMA.CH0.CTRLA |= DMA_CH_TRFREQ_bm;
}
```

SECTION 4 (Using our speaker)

Pretty much the only change between this and 2b is that you need to use the channel 1 for the DAC and you need to set PortC pin 8. So I won't show everything. I based this section off of section 2 and not section 3 because I couldn't get my section 3 to work properly right away.

```
//in main

	PORTC.DIRSET = 0x80;
	PORTC.OUTSET = 0x80;


void dac_init(void)
{
	/* Use only channel 1 */
	DACA.CTRLB = DAC_CHSEL_SINGLE1_gc | DAC_CH1TRIG_bm;

	/* Use AREFB (2.5V). Data is right-adjusted. */
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;

	/* Enable channel 0, as well as the overall DAC module. */
	DACA.CTRLA = DAC_CH1EN_bm | DAC_ENABLE_bm;

	/* make a DAC conversion start when Event Channel 0 is triggered */
	DACA.EVCTRL = DAC_EVSEL_1_gc;
}


//In tcc0_init
	EVSYS.CH1MUX = EVSYS_CHMUX_TCC0_OVF_gc;
```

SECTION 5 (Create the Piano)

I didn't have time to finish this section. I got caught up on the previous sections, but I know how I would go about doing this. I will put what code I have but it will have pseudo code in it to get my ideas across. I will also explain it shortly here.

Section 5 would be similar to section 3 & section 4. I would create a usart_init() function so I would be able to communicate to Putty through my keyboard. I also would change my tcc0_init() to use a new variable "frequency" to calculate the period. Then I would constantly poll the usart receiver to see what key has been pushed. Based on what key was pushed I would have some logic that would change the value of the frequency variable. This logic would also check if it should use the sine wave or the triangle wave.

```
#include <avr/io.h>
#include <avr/interrupt.h>

extern void clock_init(void);

int16_t frequency;
int16_t array_position;


int main(void)
{
	clock_init();
	tcc0_init();
	dma_init();
	dac_init();
	intr_init();
	usartd0_init();

    while (1)
    {
    }
}



void dac_init(void)
{
	/* Use only channel 1 */
	DACA.CTRLB = DAC_CHSEL_SINGLE1_gc | DAC_CH1TRIG_bm;

	/* Use AREFB (2.5V). Data is right-adjusted. */
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;

	/* Enable channel 0, as well as the overall DAC module. */
	DACA.CTRLA = DAC_CH1EN_bm | DAC_ENABLE_bm;

	/* make a DAC conversion start when Event Channel 0 is triggered */
	DACA.EVCTRL = DAC_EVSEL_1_gc;
}

void intr_init(void)
{
	/* Enable low level interrupts in the PMIC. */
	PMIC.CTRL = PMIC_LOLVLEN_bm;

	/* Enable interrupts globally. */
```

```
	sei();
}


void tcc0_init(void)
{
	//;set TCC0 period register
	// we want sine wave with a 1000Hz frequency
	// but we have 256 values for each frequency
	// so our PER = (1/1760Hz) /256 = 1/256000
	//;TCC0_PER = (fclk/prescalar) * (duration in seconds)
	//;                          32MH/1                              1/459560
	//this didn't give a value withing 2% so I manually changed the PER

	TCC0.CNT = 0;
	TCC0.PER = 32000000/(frequency * 256); //(32000000/1) / 459560; // = 69 but didn't work
	TCC0.CTRLA = TC_CLKSEL_DIV1_gc;

	/* Clear OVFIF to be safe */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	/* Set as low level interrupt */
	TCC0.INTCTRLA = TC_OVFINTLVL_LO_gc;

	EVSYS.CH1MUX = EVSYS_CHMUX_TCC0_OVF_gc;
}


void dma_init(void)
{
	/* reset entire DMA periph */
	DMA.CTRL |= DMA_RESET_bm;

	DMA.CH0.CTRLA = DMA_CH_SINGLE_bm | DMA_CH_BURSTLEN_2BYTE_gc; // I think I need to change this to
2byte

	DMA.CH0.ADDRCTRL = DMA_CH_SRCRELOAD_BLOCK_gc | DMA_CH_SRCDIR_INC_gc | DMA_CH_DESTRELOAD_NONE_gc |
DMA_CH_DESTDIR_FIXED_gc;

	DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_DACA_CH0_gc;//DMA_CH_TRIGSRC_DACA_CH0_gc  or
DMA_CH_TRIGSRC_TCC0_OVF_gc change to TC interrupt

	DMA.CH0.TRFCNT = (uint16_t)(sizeof(sine));

	DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)sine);
	DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)sine) >> 8);
	DMA.CH0.SRCADDR2 = (uint8_t)(((uint32_t)((uintptr_t)sine)) >> 16);

	DMA.CH0.DESTADDR0 = (uint8_t)((uintptr_t)&DACA.CH0DATA);
	DMA.CH0.DESTADDR1 = (uint8_t)(((uintptr_t)&DACA.CH0DATA) >> 8);
	DMA.CH0.DESTADDR2 = (uint8_t)(((uint32_t)((uintptr_t)&DACA.CH0DATA)) >> 16);



	/* enable CH0 */
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;

	/* enable entire DMA periph */
	DMA.CTRL = DMA_ENABLE_bm;

}
```

```
void usartd0_init()
{
	//BSEL = 1
	//BSCALE = -4

	/* Configure the UART frame. */
	USARTD0.CTRLC = ( USART_CMODE_ASYNCHRONOUS_gc | USART_PMODE_DISABLED_gc | USART_CHSIZE_8BIT_gc );
//need to use 8 bit

	/* Initialize the baud rate */
	USARTD0.BAUDCTRLA = 1;
	USARTD0.BAUDCTRLB = (-4<<4); //Should really or the high part of 1, but that is 0

	/* Enable the transmitter and receiver */
	USARTD0.CTRLB = ( USART_TXEN_bm | USART_RXEN_bm );

	USARTD0.CTRLA = 0b010000; //make low level. couldn't find the bm

}



ISR(TCC0_OVF_vect)
{
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;

	DMA.CH0.CTRLA |= DMA_CH_TRFREQ_bm;
}


ISR(USARTD0_RXC_vect)
{
	while(!(USARTD0.STATUS & USART_DREIF_bm)); //wait

	if(USARTD0.DATA == 's')
	{
		//change to triangle wave and sine wave
		//I would make this a boolean

	}
	else if(USARTD0.DATA == 'W')
	{
		frequency = 1046.5;

	}
	else if(USARTD0.DATA == '3')
	{
		frequency = 1108.73;
	}
	else if(USARTD0.DATA == 'E')
	{
		frequency = 1174.66;
	}
	else if(USARTD0.DATA == '4')
	{
		frequency = 1244.51;
	}
	//Keep going. I would probably change to a case statement to make it shorter
}
```

APPENDIX

LUTs used:

```
uint16_t sine[256] =
{
	0x800,0x832,0x864,0x896,0x8c8,0x8fa,0x92c,0x95e,0x98f,0x9c0,
	0x9f1,0xa22,0xa52,0xa82,0xab1,0xae0,0xb0f,0xb3d,0xb6b,0xb98,
	0xbc5,0xbf1,0xc1c,0xc47,0xc71,0xc9a,0xcc3,0xceb,0xd12,0xd39,
	0xd5f,0xd83,0xda7,0xdca,0xded,0xe0e,0xe2e,0xe4e,0xe6c,0xe8a,
	0xea6,0xec1,0xedc,0xef5,0xf0d,0xf24,0xf3a,0xf4f,0xf63,0xf76,
	0xf87,0xf98,0xfa7,0xfb5,0xfc2,0xfcd,0xfd8,0xfe1,0xfe9,0xff0,
	0xff5,0xff9,0xffd,0xffe,0xfff,0xffe,0xffd,0xff9,0xff5,0xff0,
	0xfe9,0xfe1,0xfd8,0xfcd,0xfc2,0xfb5,0xfa7,0xf98,0xf87,0xf76,
	0xf63,0xf4f,0xf3a,0xf24,0xf0d,0xef5,0xedc,0xec1,0xea6,0xe8a,
	0xe6c,0xe4e,0xe2e,0xe0e,0xded,0xdca,0xda7,0xd83,0xd5f,0xd39,
	0xd12,0xceb,0xcc3,0xc9a,0xc71,0xc47,0xc1c,0xbf1,0xbc5,0xb98,
	0xb6b,0xb3d,0xb0f,0xae0,0xab1,0xa82,0xa52,0xa22,0x9f1,0x9c0,
	0x98f,0x95e,0x92c,0x8fa,0x8c8,0x896,0x864,0x832,0x800,0x7cd,
	0x79b,0x769,0x737,0x705,0x6d3,0x6a1,0x670,0x63f,0x60e,0x5dd,
	0x5ad,0x57d,0x54e,0x51f,0x4f0,0x4c2,0x494,0x467,0x43a,0x40e,
	0x3e3,0x3b8,0x38e,0x365,0x33c,0x314,0x2ed,0x2c6,0x2a0,0x27c,
	0x258,0x235,0x212,0x1f1,0x1d1,0x1b1,0x193,0x175,0x159,0x13e,
	0x123,0x10a,0xf2,0xdb,0xc5,0xb0,0x9c,0x89,0x78,0x67,
	0x58,0x4a,0x3d,0x32,0x27,0x1e,0x16,0xf,0xa,0x6,
	0x2,0x1,0x0,0x1,0x2,0x6,0xa,0xf,0x16,0x1e,
	0x27,0x32,0x3d,0x4a,0x58,0x67,0x78,0x89,0x9c,0xb0,
	0xc5,0xdb,0xf2,0x10a,0x123,0x13e,0x159,0x175,0x193,0x1b1,
	0x1d1,0x1f1,0x212,0x235,0x258,0x27c,0x2a0,0x2c6,0x2ed,0x314,
	0x33c,0x365,0x38e,0x3b8,0x3e3,0x40e,0x43a,0x467,0x494,0x4c2,
	0x4f0,0x51f,0x54e,0x57d,0x5ad,0x5dd,0x60e,0x63f,0x670,0x6a1,
	0x6d3,0x705,0x737,0x769,0x79b,0x7cd,0x800
};
uint16_t triangle[256] =
{
	0x20,0x40,0x60,0x80,0xa0,0xc0,0xe0,0x100,0x120,0x140,
	0x160,0x180,0x1a0,0x1c0,0x1e0,0x200,0x220,0x240,0x260,0x280,
	0x2a0,0x2c0,0x2e0,0x300,0x320,0x340,0x360,0x380,0x3a0,0x3c0,
	0x3e0,0x400,0x420,0x440,0x460,0x480,0x4a0,0x4c0,0x4e0,0x500,
	0x520,0x540,0x560,0x580,0x5a0,0x5c0,0x5e0,0x600,0x620,0x640,
	0x660,0x680,0x6a0,0x6c0,0x6e0,0x700,0x720,0x740,0x760,0x780,
	0x7a0,0x7c0,0x7e0,0x800,0x81f,0x83f,0x85f,0x87f,0x89f,0x8bf,
	0x8df,0x8ff,0x91f,0x93f,0x95f,0x97f,0x99f,0x9bf,0x9df,0x9ff,
	0xa1f,0xa3f,0xa5f,0xa7f,0xa9f,0xabf,0xadf,0xaff,0xb1f,0xb3f,
	0xb5f,0xb7f,0xb9f,0xbbf,0xbdf,0xbff,0xc1f,0xc3f,0xc5f,0xc7f,
	0xc9f,0xcbf,0xcdf,0xcff,0xd1f,0xd3f,0xd5f,0xd7f,0xd9f,0xdbf,
	0xddf,0xdff,0xe1f,0xe3f,0xe5f,0xe7f,0xe9f,0xebf,0xedf,0xeff,
	0xf1f,0xf3f,0xf5f,0xf7f,0xf9f,0xfbf,0xfdf,0xfff,0xfdf,0xfbf,
	0xf9f,0xf7f,0xf5f,0xf3f,0xf1f,0xeff,0xedf,0xebf,0xe9f,0xe7f,
	0xe5f,0xe3f,0xe1f,0xdff,0xddf,0xdbf,0xd9f,0xd7f,0xd5f,0xd3f,
	0xd1f,0xcff,0xcdf,0xcbf,0xc9f,0xc7f,0xc5f,0xc3f,0xc1f,0xbff,
	0xbdf,0xbbf,0xb9f,0xb7f,0xb5f,0xb3f,0xb1f,0xaff,0xadf,0xabf,
	0xa9f,0xa7f,0xa5f,0xa3f,0xa1f,0x9ff,0x9df,0x9bf,0x99f,0x97f,
	0x95f,0x93f,0x91f,0x8ff,0x8df,0x8bf,0x89f,0x87f,0x85f,0x83f,
	0x81f,0x800,0x7e0,0x7c0,0x7a0,0x780,0x760,0x740,0x720,0x700,
	0x6e0,0x6c0,0x6a0,0x680,0x660,0x640,0x620,0x600,0x5e0,0x5c0,
	0x5a0,0x580,0x560,0x540,0x520,0x500,0x4e0,0x4c0,0x4a0,0x480,
	0x460,0x440,0x420,0x400,0x3e0,0x3c0,0x3a0,0x380,0x360,0x340,
	0x320,0x300,0x2e0,0x2c0,0x2a0,0x280,0x260,0x240,0x220,0x200,
	0x1e0,0x1c0,0x1a0,0x180,0x160,0x140,0x120,0x100,0xe0,0xc0,0xa0,0x80,0x60,0x40,0x20,0x0}
```

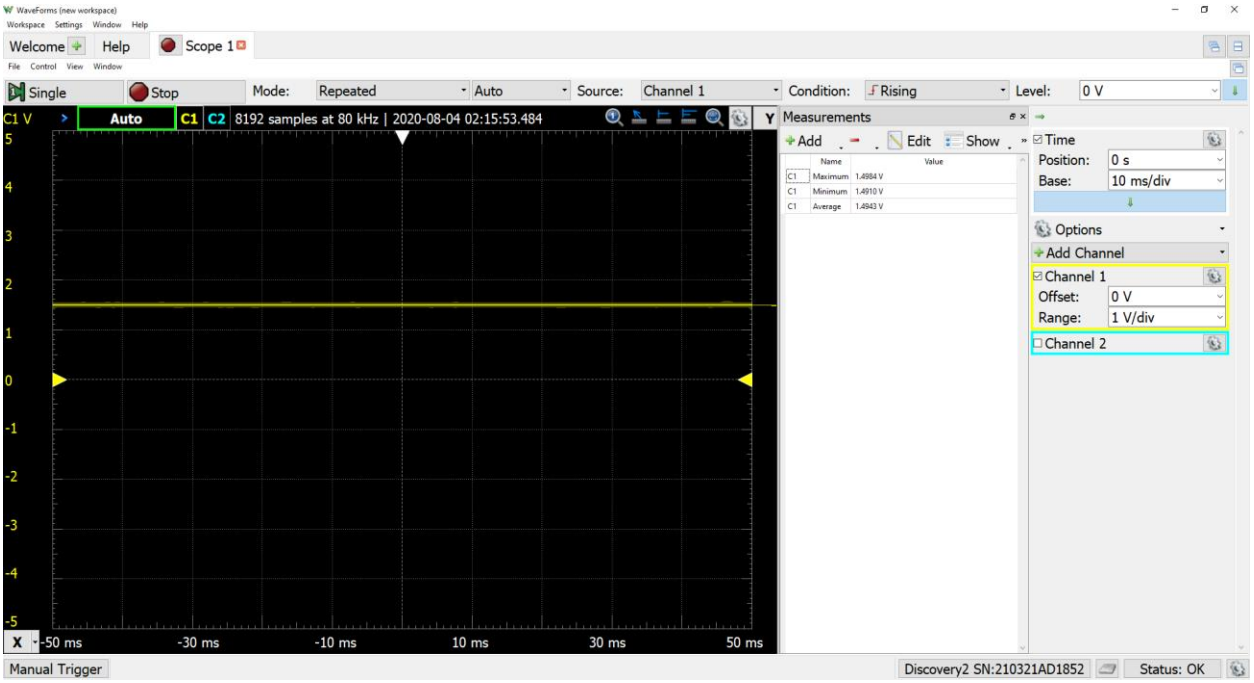

Figure 1: Section 1 screenshot. Constant 1.5V

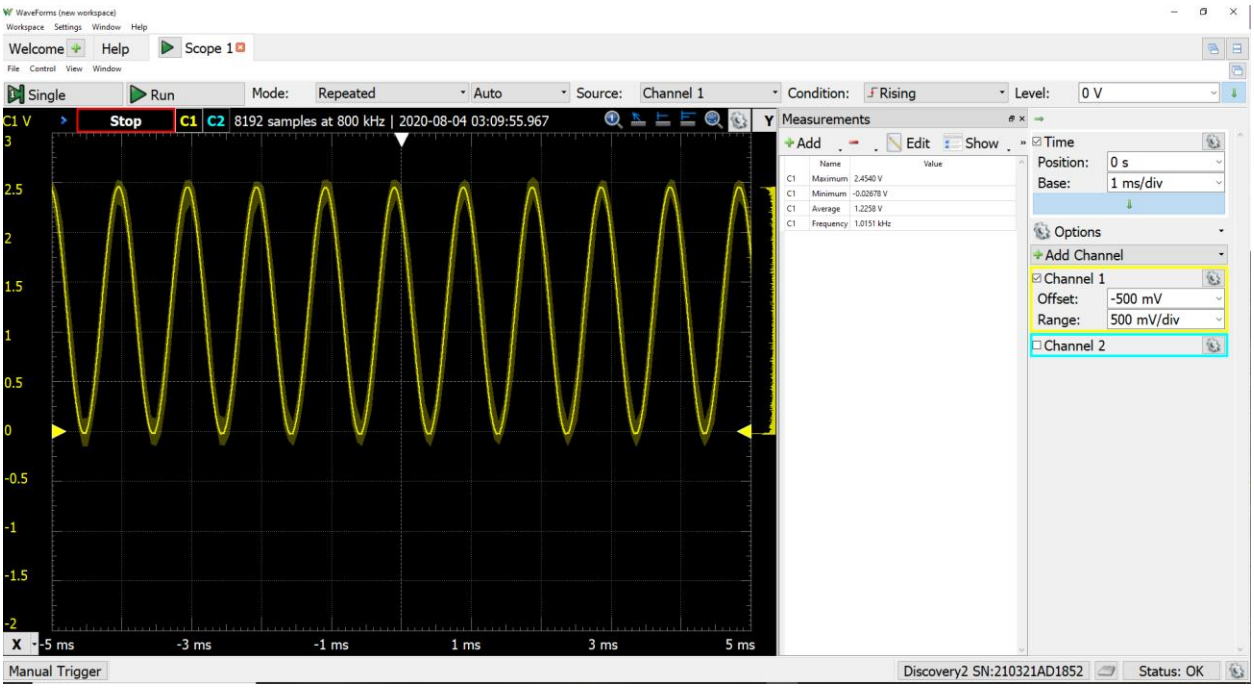

Figure 2: Section 2 screenshot. 1000Hz

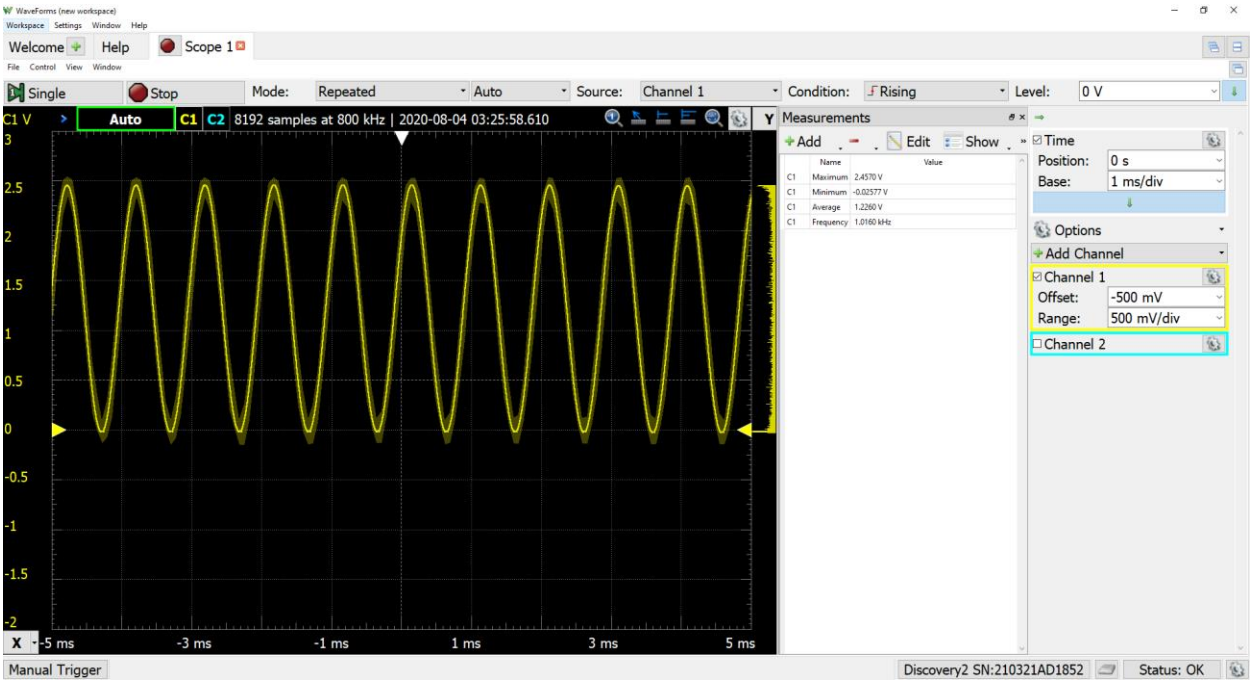

Figure 3:Section 2 screenshot. This was the output when the clock was set at a desired frequency of 1100Hz

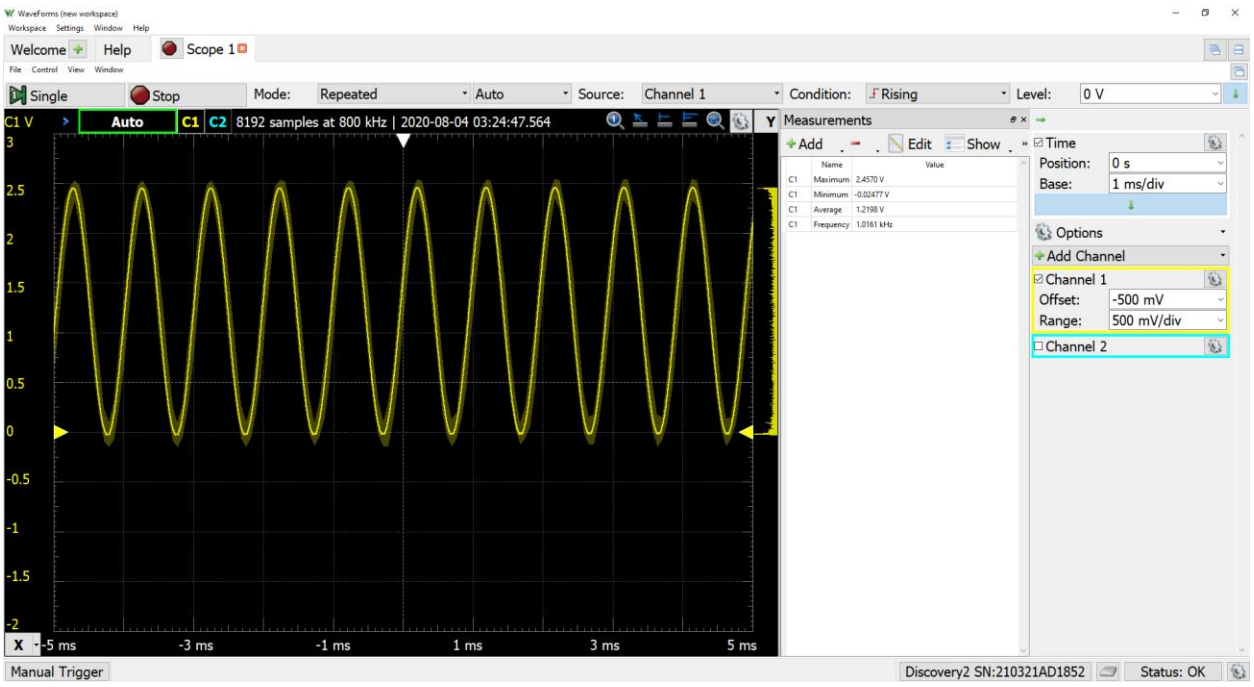

Figure 4: Section 2 Screenshot with the clock period at 1. Smallest it can be.

For section 2:

So, I was a little confused during the section where we are trying to increase the target frequency by 100Hz until it couldn't meet the desired accuracy. I didn't know if our +-2% was still related to our 1000Hz frequency or if now was +-2% of the new desired frequency. I didn't stress too much though because when I made my desired frequency 1100Hz, my resulting frequency was almost the same as my 1000Hz frequency. In fact, I could not really make the resulting frequency go much higher (only around 3Hz). Even with the Period of my timer at 1, I got around 1017Hz which is still within +-2% of our original 1000Hz. So, if our accuracy is based on the new frequency, 1100Hz was the highest I could go. And if our accuracy is always based on 1000Hz, I could not get +2% higher. The above graphs show that my frequency sort of leveled off around 1015Hz.

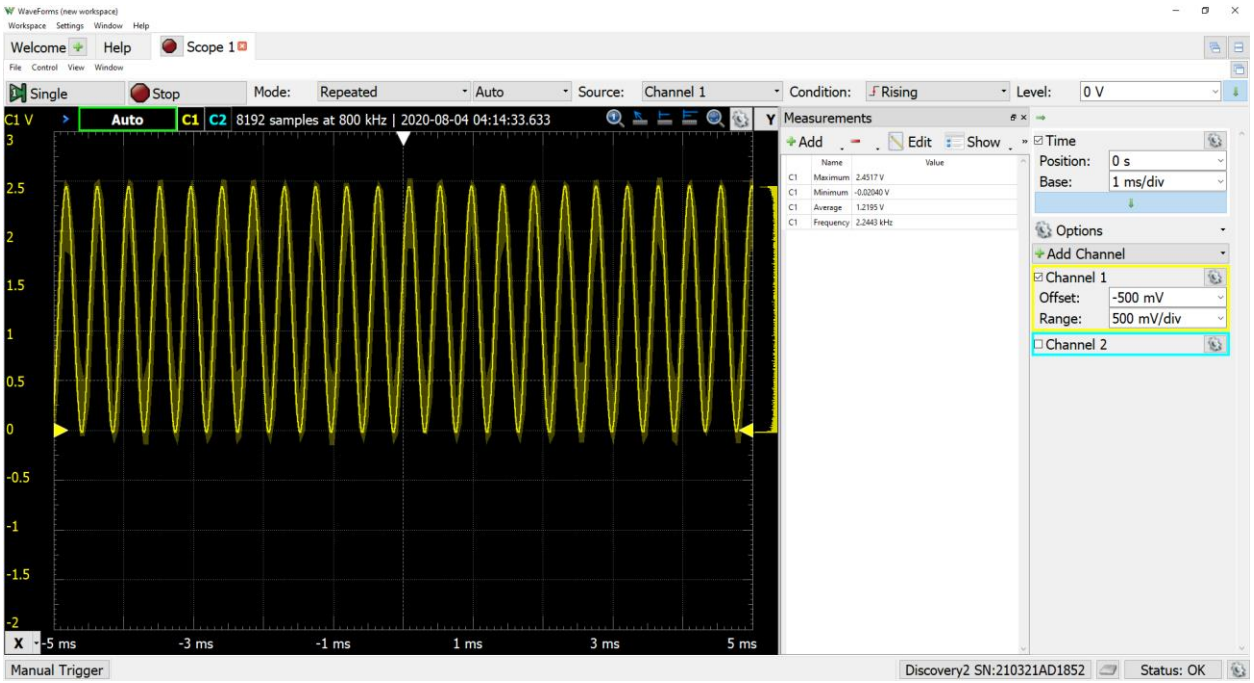

Figure 5: Section 2 Polling

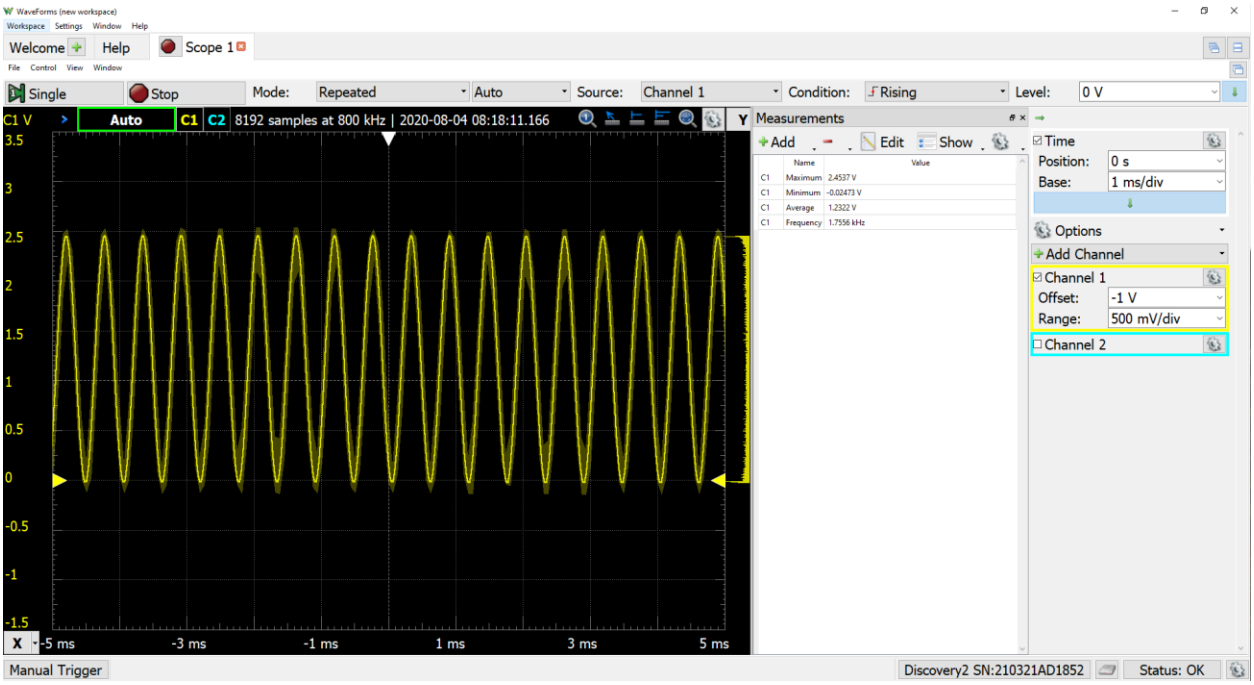

Figure 6: Section 2 Events