REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

N/A

FUTURE WORK/APPLICATIONS

Digital to analog converters are frequently used in:
synthesizers
measuring devices
game controllers
music players
DAC’s can also be used for transferring analog voltages across telephone lines for dial up internet, or cable lines for cable internet.

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?

> **Because the compiled code is more inefficient than hand writing raw assembly code, the counter runs for a few extra clock cycles. Hence, lowering the frequency.**
> **When changing the optimization level, the frequency increases due to the elimination of extra instructions**

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.

> **No, it would not be fixed. The assembly code is still bloated.**

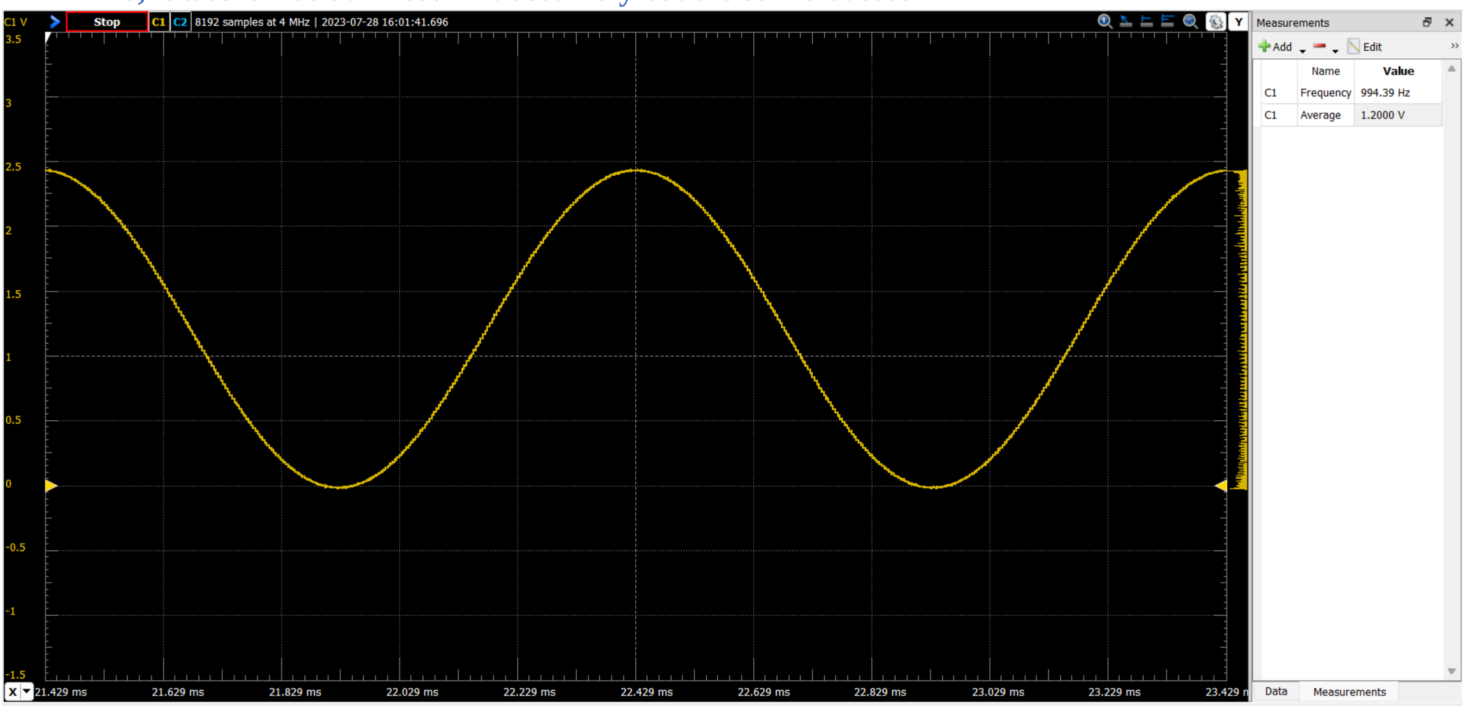

Figure 0: Waveform after using synchronous polling

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

> **As more data points are used, the smoother the waveform is.**

Miller, Steven
Class #: 11318
Anthony Stross
July 26, 2023

PSEUDOCODE/FLOWCHARTS

SECTION 1

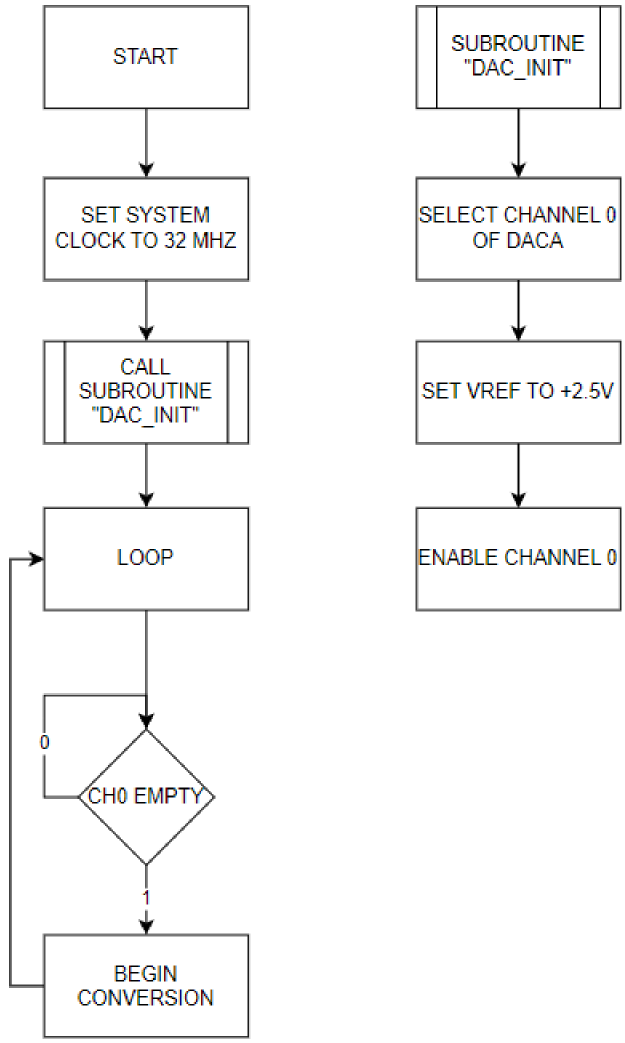

Figure 1: Flowchart for "lab8_1.C"

SECTION 2a

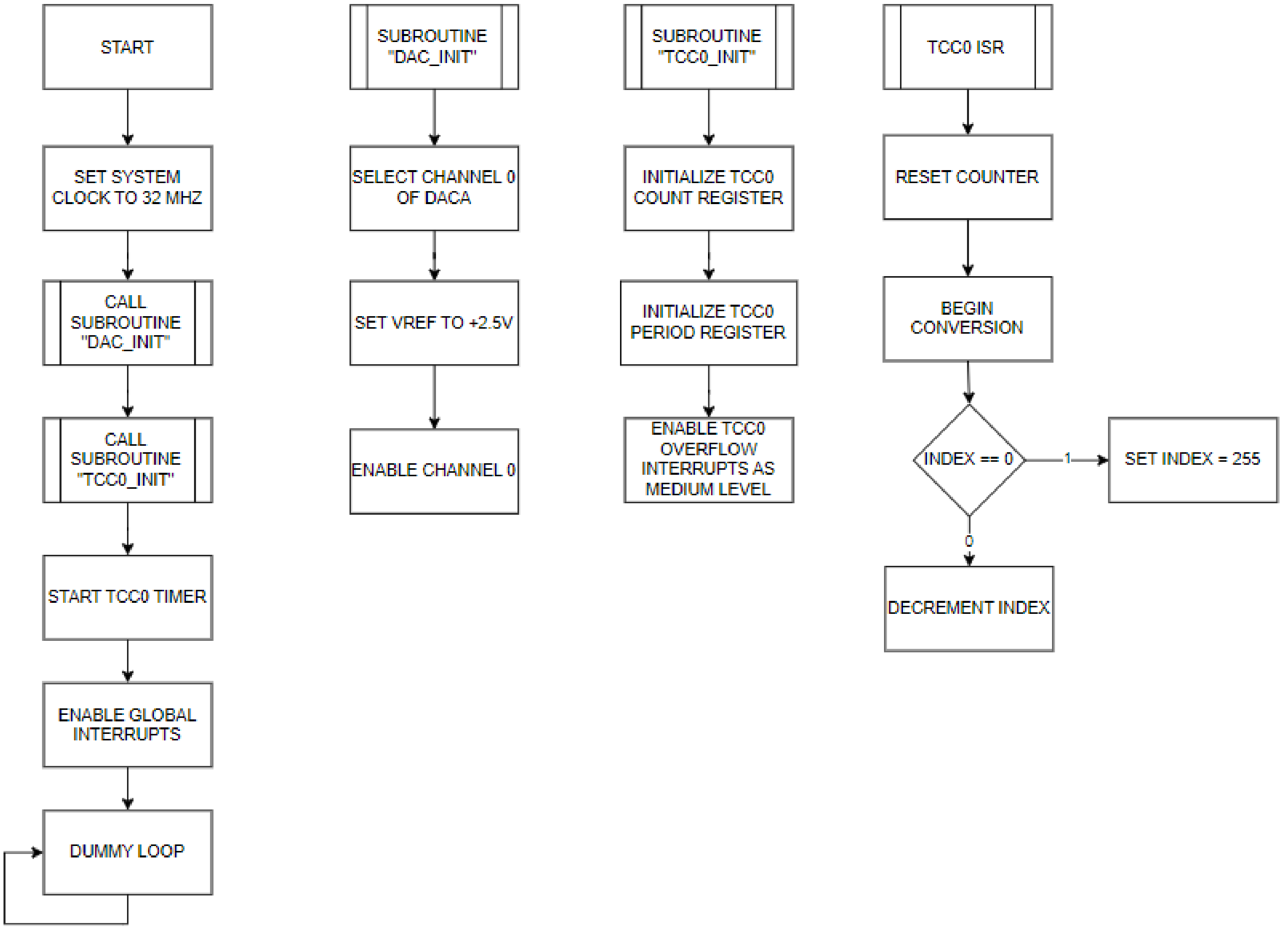

Figure 2: Flowchart for “lab8_2a.C”

SECTION 2b

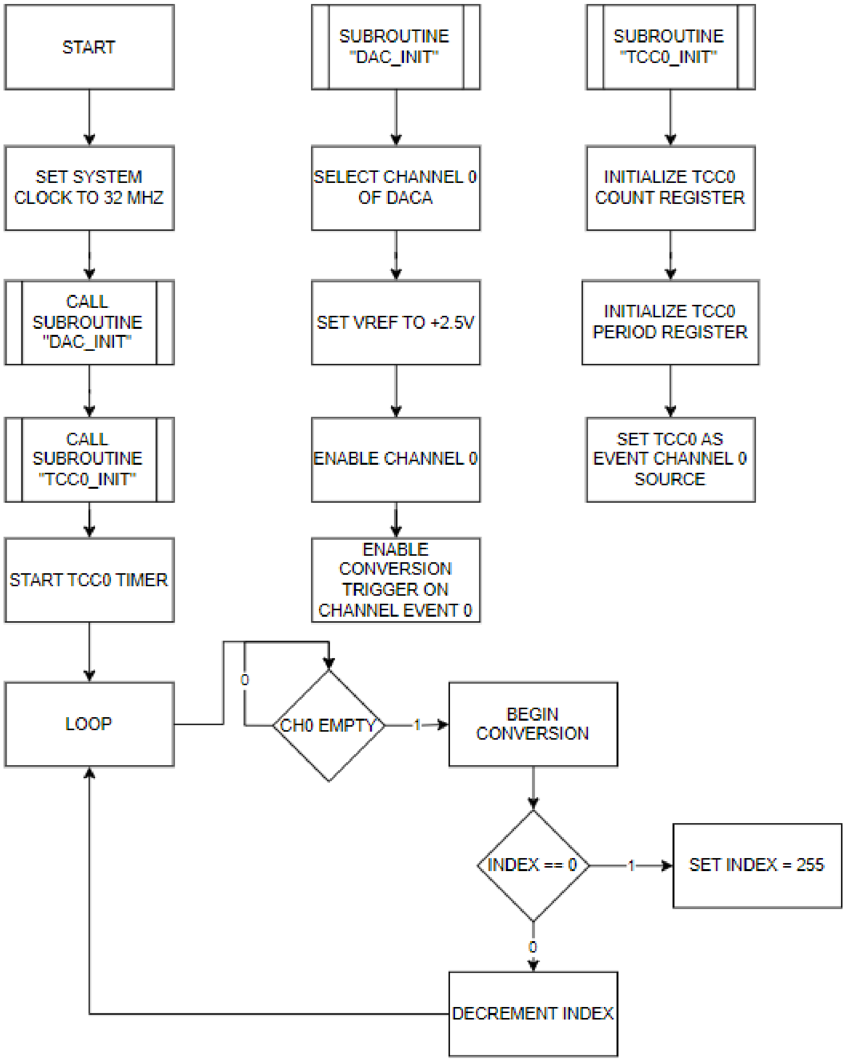

Figure 3: Flowchart for "lab8_2b.C"

SECTION 3

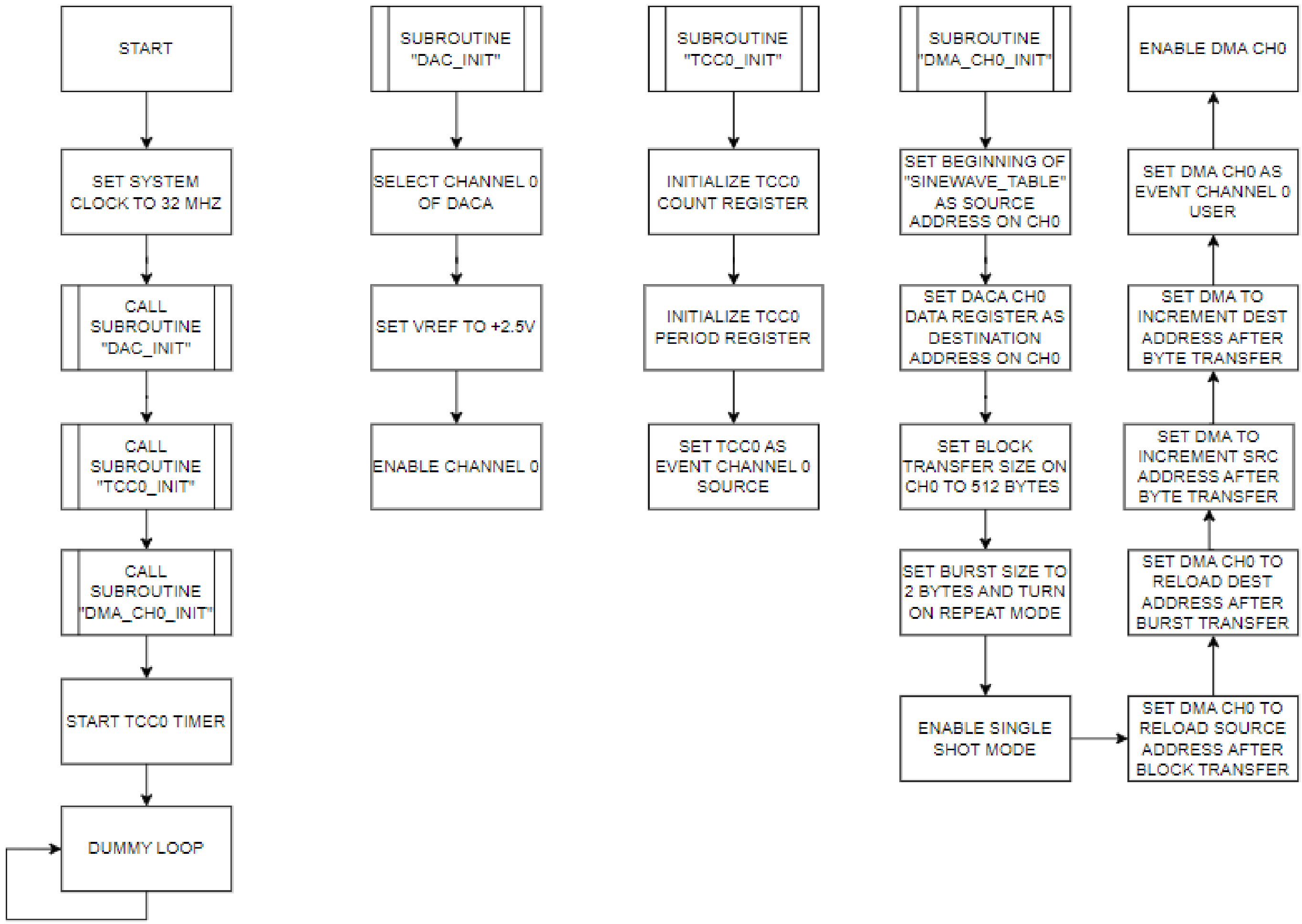

Figure 4: Flowchart for "lab8_3.C"

Miller, Steven
Class #: 11318
Anthony Stross
July 26, 2023

SECTION 4

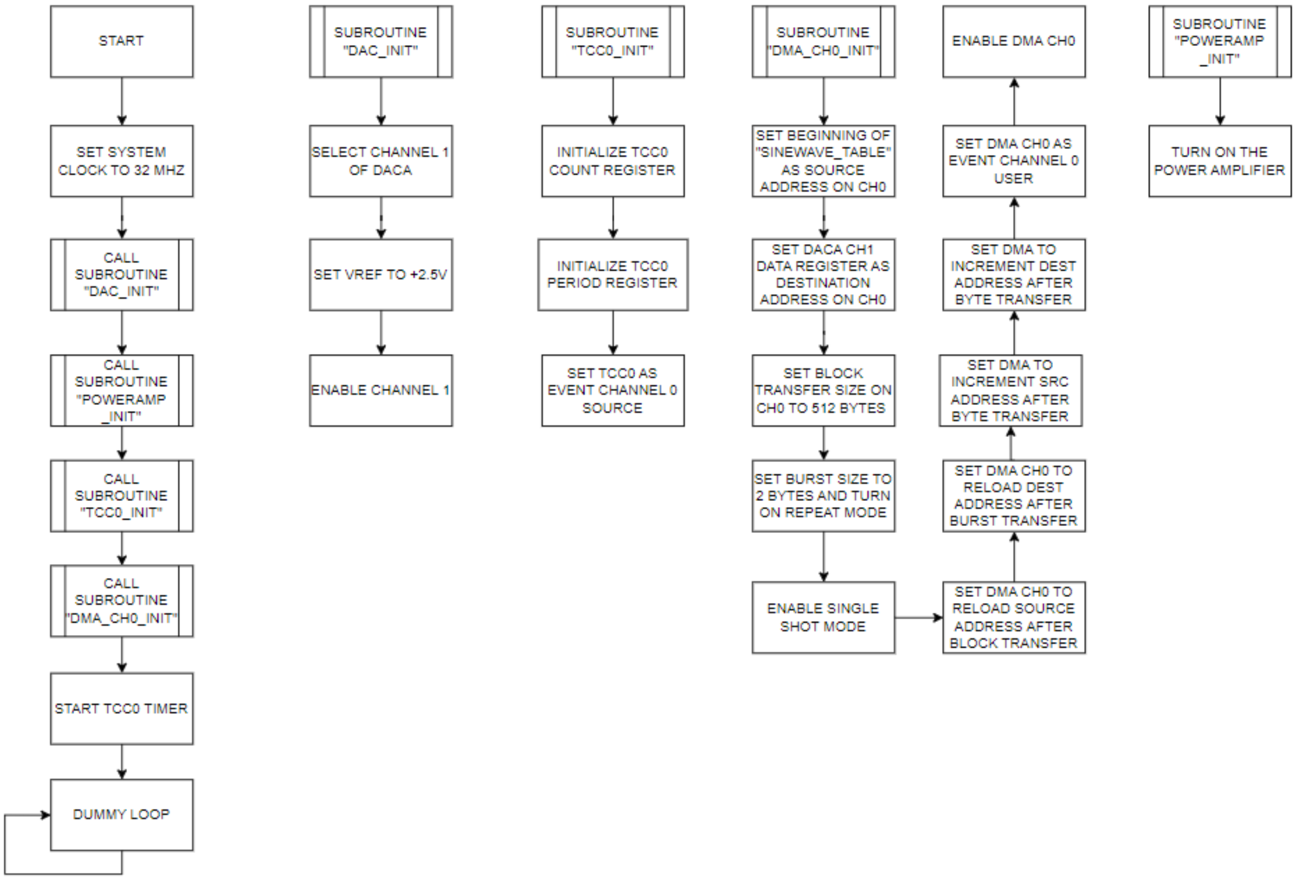

Figure 5: Flowchart for "lab8_4.C"

SECTION 5

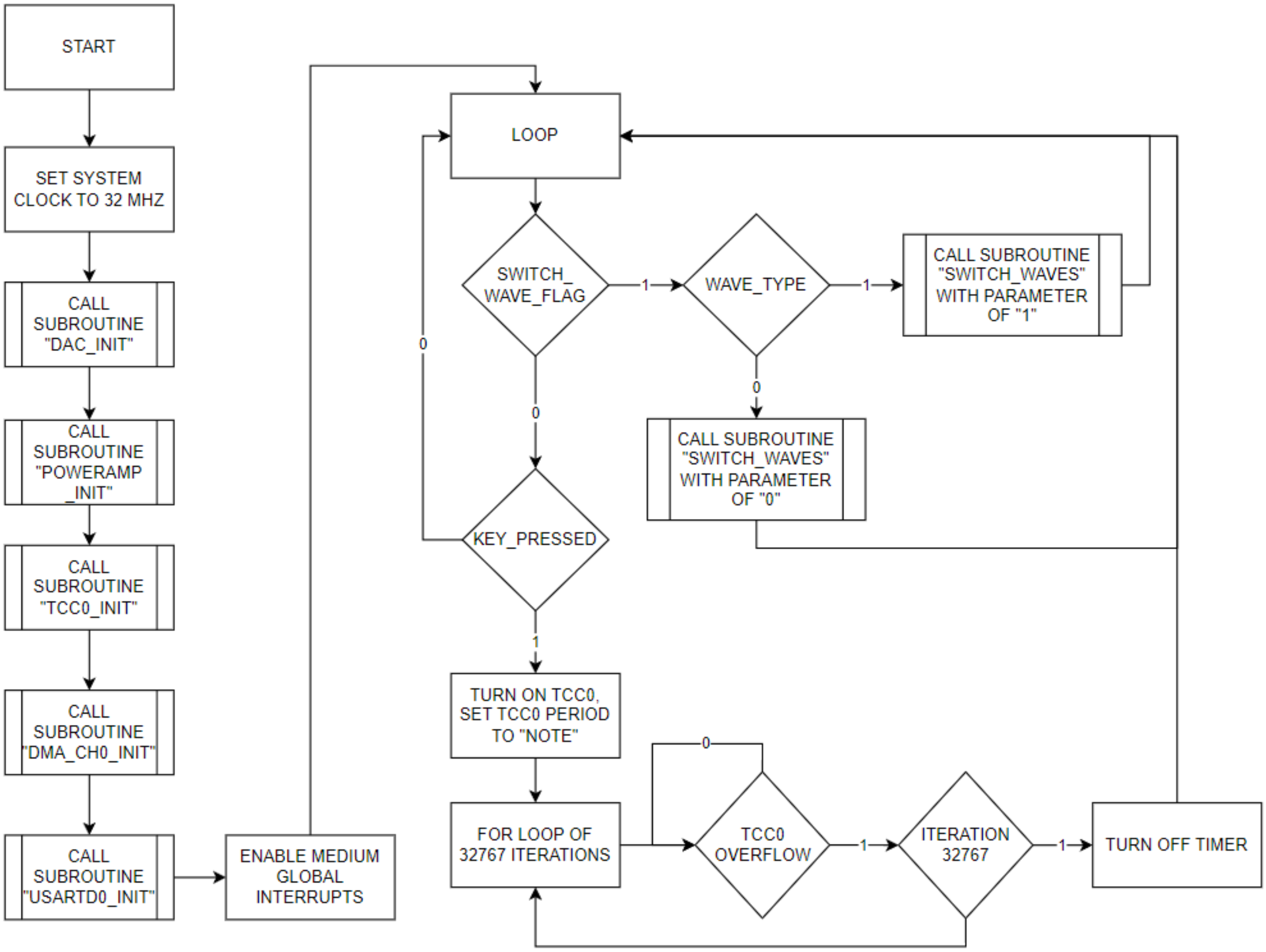

Figure 6: Flowchart for main routine of “lab8_5.C”

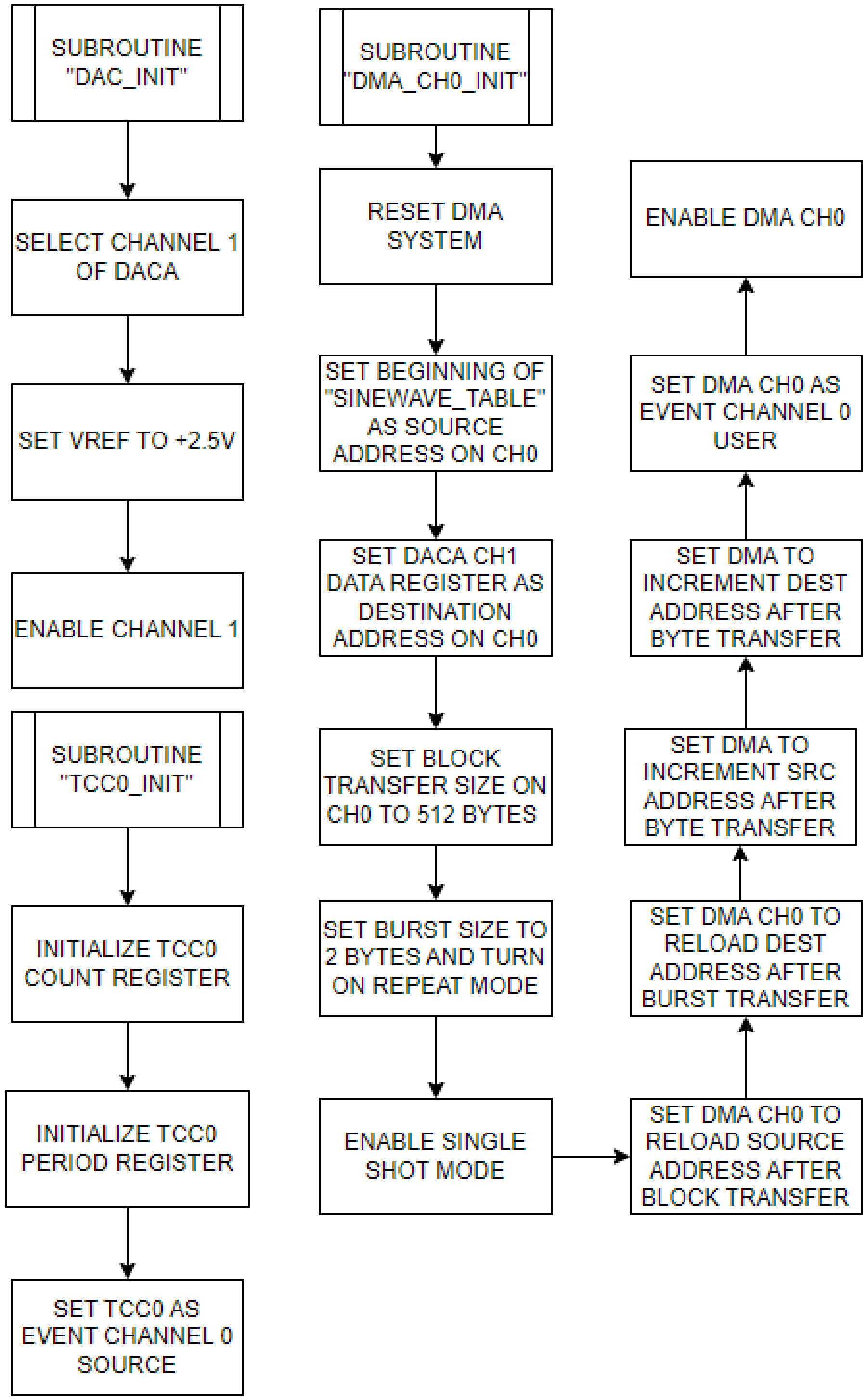

Figure 7: Flowcharts for DAC, TCC0, and DMA initialization subroutines for “lab8_5.C”

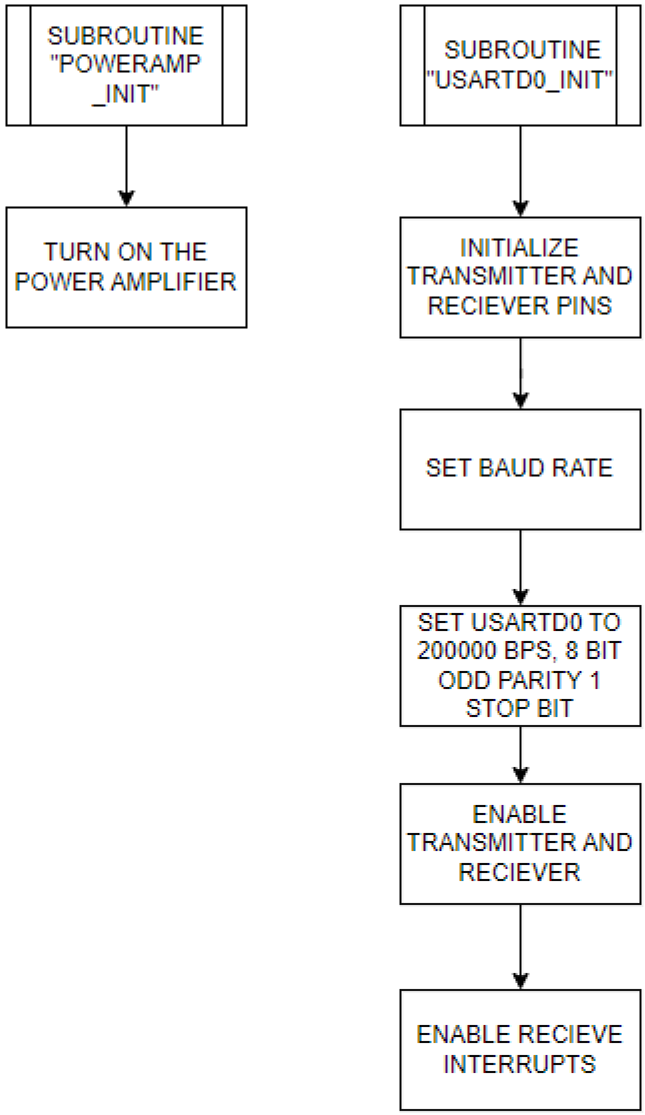

Figure 8: Flowcharts for USART and power amplifier initialization subroutines for "lab8_5.C"

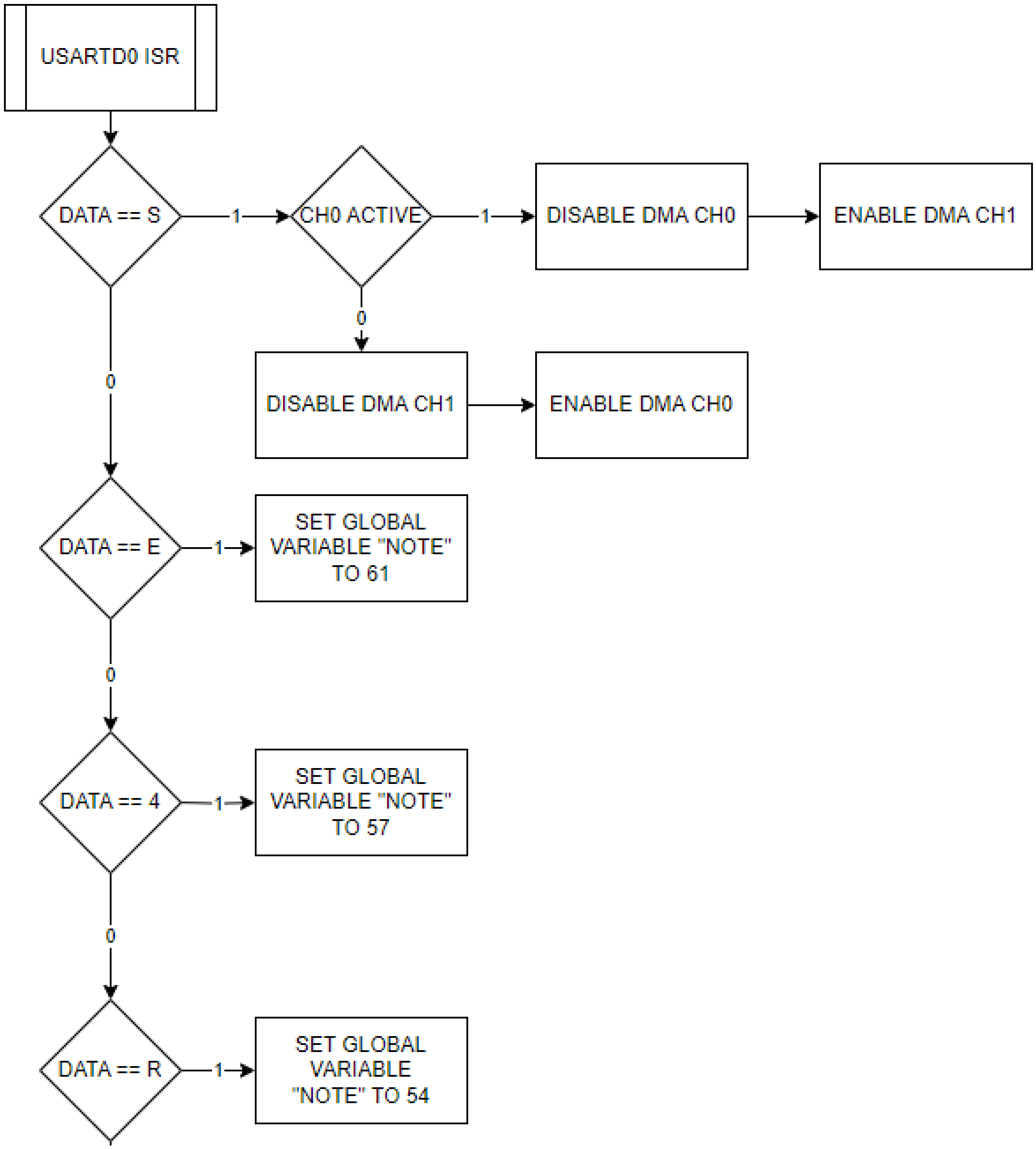

Figure 9: Flowchart for USARTD0 ISR for "lab8_5.C"

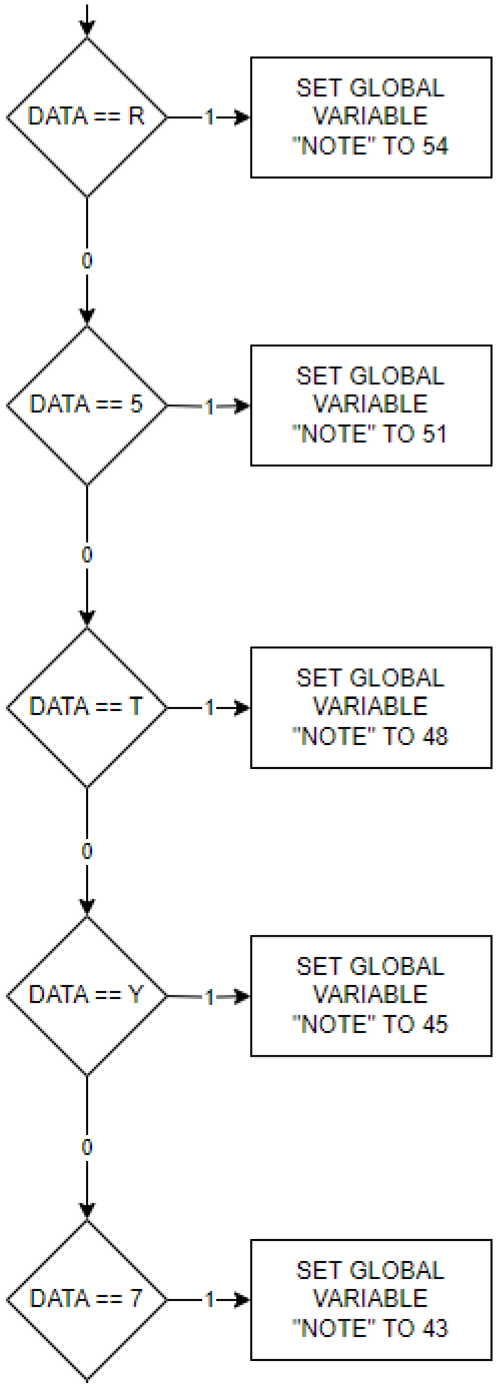

Figure 10: Flowchart for USARTD0 ISR for "lab8_5.C"

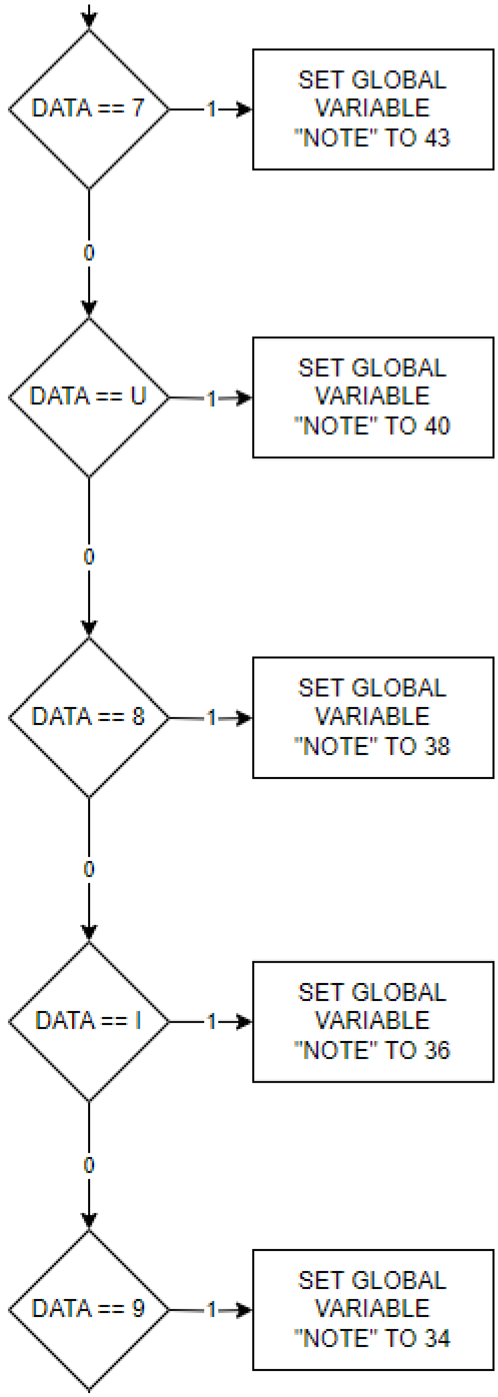

Figure 11: Flowchart for USARTD0 ISR for "lab8_5.C"

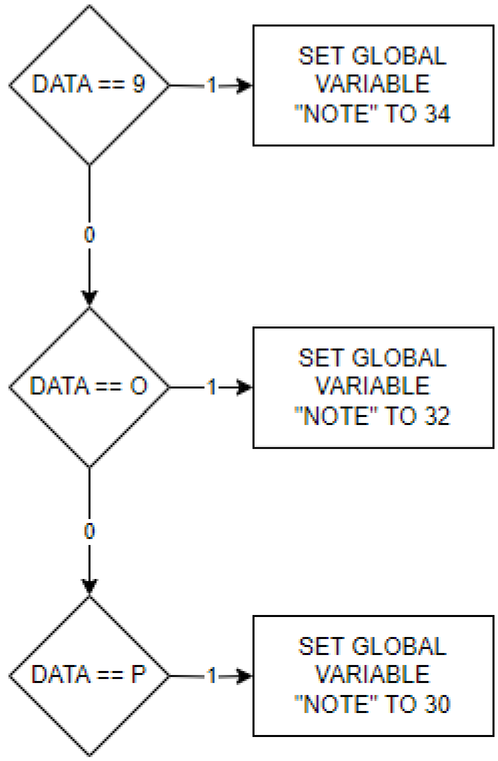

Figure 12: Flowchart for USARTD0 ISR for "lab8_5.C"

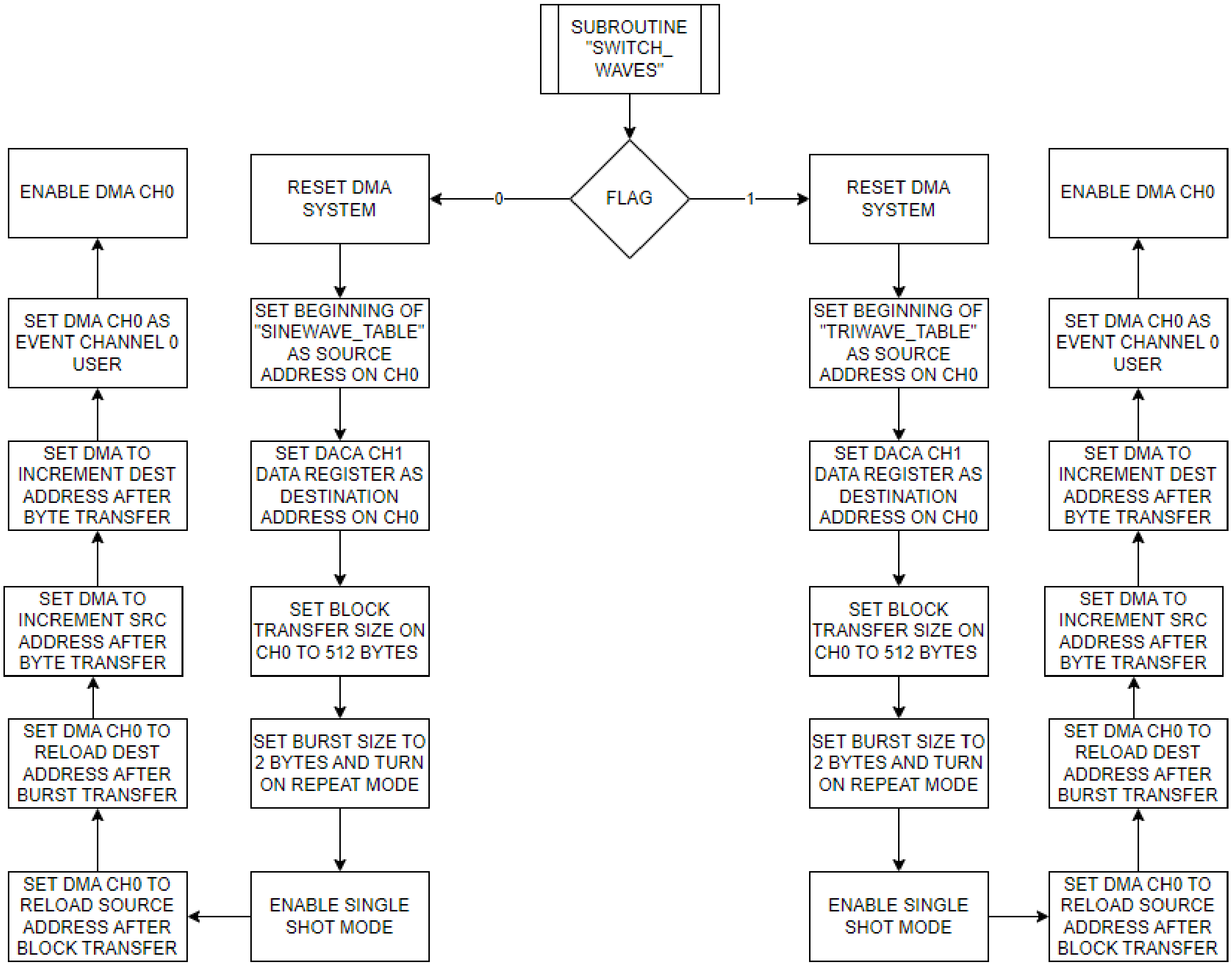

Figure 13: Flowchart for “switch_waves” subroutine in “lab8_5.C”

PROGRAM CODE

SECTION 1

```c
//*******************************************
//Lab 8, Section 1
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: outputs constant 1.7 volts using DAC
//*******************************************
#include <avr/io.h>
#define onepointsevenvolts 0xAE1
#define maxvoltage 4095 //2.5v converted to decimal
#define slope (2.5/4095)
extern void clock_init(void);

int main(void)
{
	//set system clock to 32 mhz
	clock_init();
	//initialize DAC
	dac_init();

    while (1)
    {
		//wait until channel 0 is empty (conversion is finished)
		while(!(DACA.STATUS & DAC_CH0DRE_bm))
		{
			//do nothing
		}
		DACA.CH0DATA = onepointsevenvolts;
    }
}

void dac_init(void)
{
	//use only channel 0
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc;
	//use arefb
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;
	//enable channel 0
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
}
```

SECTION 2a

```
//*******************************************
//Lab 8, Section 2a
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: outputs sinewave at 988 hz
//*******************************************
#include <avr/io.h>
#include <avr/interrupt.h>
#define maxvoltage 4095 //2.5v converted to decimal
#define slope (2.5/4095)
extern void clock_init(void);
extern uint16_t sine_wave[256];
volatile uint8_t index = 255;
int main(void)
{
	//set system clock to 32 mhz
	clock_init();
	//initialize DAC
	dac_init();
	//initialize timer counter
	tcc0_init();
	//start tcc0 timer
	TCC0.CTRLA = TC_CLKSEL_DIV2_gc;
	//enable global interrupts
	PMIC.CTRL = PMIC_MEDLVLEN_bm;
	sei();
	//dummy loop
	while (1)
	{
	}
}

void dac_init(void)
{
	//use only channel 0
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc;
	//use arefb
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;
	//enable channel 0
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
}
void tcc0_init(void)
{
	uint16_t period =  63;
	int8_t offset = -21;
	//INITIALIZE COUNT REGISTER
	TCC0.CNT = 0;
	//initialize tcc0 period register
	TCC0.PER = period + offset;
	TCC0.CNT = 0;
	//enable tcc0 overflow interrupts as medium priority
	TCC0.INTCTRLA = TC_OVFINTLVL_MED_gc;
};
```

```
ISR(TCC0_OVF_vect)
{
	//reset counter
	TCC0.CNT = 0;
	//begin conversion
	DACA.CH0DATA = sine_wave[index];
	if(index == 0)
	{
		index = 255;
	}
	else
	{
		index = index-1;
	}


}
```

SECTION 2b

```
//*******************************************
//Lab 8, Section 2b
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: outputs sinewave at 1567.98 Hz
//*******************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#define maxvoltage 4095 //2.5v converted to decimal
#define slope (2.5/4095)
extern void clock_init(void);
extern uint16_t sine_wave[256];
volatile uint8_t index = 255;
int main(void)
{
	//set system clock to 32 mhz
	clock_init();
	//initialize DAC
	dac_init();
	//initialize timer counter
	tcc0_init();
	//start tcc0 timer
	TCC0.CTRLA = TC_CLKSEL_DIV2_gc;
	while (1)
	{
		while(!(DACA.STATUS & DAC_CH0DRE_bm))
		{
			//do nothing
		}
		DACA.CH0DATA = sine_wave[index];
		if(index == 0)
		{
			index = 255;
		}
		else
		{
			index = index-1;
		}

	}
}

void dac_init(void)
{
	//use only channel 0
	DACA.CTRLB = DAC_CHSEL_SINGLE_gc| DAC_CH0TRIG_bm;
	//use arefb
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;
	//enable channel 0
	DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
	//enable conversion trigger on channel event 0
	DACA.EVCTRL = DAC_EVSEL_0_gc;
}
```

```
void tcc0_init(void)
{
	uint16_t period = 40;
	int8_t offset = 0;
	//INITIALIZE COUNT REGISTER
	TCC0.CNT = 0;
	//initialize tcc0 period register
	TCC0.PER = period + offset;
	//set tcc0 as event channel 0 source
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
};
```

SECTION 3

```
//*********************************************
//Lab 8, Section 3
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: outputs sinewave at ~1567.98 Hz using DMA
//*********************************************
#include <avr/io.h>
#include <avr/interrupt.h>
#define maxvoltage 4095 //2.5v converted to decimal
#define slope (2.5/4095) //0.000610500611 volts/decimalvalue
extern void clock_init(void);
extern uint16_t sine_wave[256];
volatile uint16_t sine_wave_address = (&sine_wave);
volatile uint16_t daca_ch0_data_address = (&DACA_CH0DATA);
int main(void)
{
       //set system clock to 32 mhz
       clock_init();
       //initialize DAC
       dac_init();
       //initialize timer counter
       tcc0_init();
       //initialize DMA system
       DMA_CH0_INIT();
       //start tcc0 timer
       TCC0.CTRLA = TC_CLKSEL_DIV2_gc;
       //dummy loop
       while (1)
       {
              //DO NOTHING
       }
}

void dac_init(void)
{
       //use only channel 0
       DACA.CTRLB = DAC_CHSEL_SINGLE_gc;
       //use arefb
       DACA.CTRLC = DAC_REFSEL_AREFB_gc;
       //enable channel 0
       DACA.CTRLA = DAC_CH0EN_bm | DAC_ENABLE_bm;
}
void tcc0_init(void)
{
       uint16_t period = 40;
       int8_t offset = 0;
       //INITIALIZE COUNT REGISTER
       TCC0.CNT = 0;
       //initialize tcc0 period register
       TCC0.PER = period + offset;
       //set tcc0 as event channel 0 source
       EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
};
```

```c
void DMA_CH0_INIT(void)
{
	//set beginning of sinewave table as source address on ch0
	DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)sine_wave);
	DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)sine_wave)>>8);
	DMA.CH0.SRCADDR2 = (uint8_t)((uint32_t)(((uintptr_t)sine_wave)>>16));
	//set daca ch0 register as destination address
	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));
	//set block transfer size on ch0 to 512 bytes
	DMA.CH0.TRFCNT = 512;
	//set burst size to 2 bytes and turn on repeat mode
	DMA.CH0.CTRLA |= (DMA_CH_BURSTLEN_2BYTE_gc |DMA_CH_REPEAT_bm);
	//enable single shot mode
	DMA.CH0.CTRLA |= DMA_CH_SINGLE_bm;
	//set dma ch0 to reload source address after block transfer and
	//destination address after burst transfer
	DMA.CH0.ADDRCTRL |= (DMA_CH_SRCRELOAD_BLOCK_gc |  DMA_CH_DESTRELOAD_BURST_gc);
	//set dma to increment source and destination address after byte transfer
	DMA.CH0.ADDRCTRL |= (DMA_CH_SRCDIR_INC_gc|DMA_CH_DESTDIR_INC_gc);
	//set dma ch0 as event channel 0 user
	DMA.CH0.TRIGSRC |= DMA_CH_TRIGSRC_EVSYS_CH0_gc;
	//enable dma ch0
	DMA.CTRL |= DMA_ENABLE_bm;
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;
}
```

SECTION 4

```
//*********************************************
//Lab 8, Section 4
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: outputs sinewave at ~1567.98 Hz using DMA on ADCA channel 1
//*********************************************
#include <avr/io.h>
#include <avr/interrupt.h>
#define maxvoltage 4095 //2.5v converted to decimal
#define slope (2.5/4095) //0.000610500611 volts/decimalvalue
extern void clock_init(void);
extern uint16_t sine_wave[256];
volatile uint16_t sine_wave_address = (&sine_wave);
volatile uint8_t poweramp_on = (0x01<<7);
int main(void)
{
	//set system clock to 32 mhz
	clock_init();
	//initialize DAC
	dac_init();
	//initialize power amplifier
	poweramp_init();
	//initialize timer counter
	tcc0_init();
	//initialize DMA system
	DMA_CH0_INIT();
	//start tcc0 timer
	TCC0.CTRLA = TC_CLKSEL_DIV2_gc;
	//dummy loop
	while (1)
	{
		//DO NOTHING
	}
}

void dac_init(void)
{
	//use only channel 1
	DACA.CTRLB = DAC_CHSEL_SINGLE1_gc;
	//use arefb
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;
	//enable channel 1
	DACA.CTRLA = DAC_CH1EN_bm | DAC_ENABLE_bm;
}
void tcc0_init(void)
{
	uint16_t period = 40;
	int8_t offset = 0;
	//INITIALIZE COUNT REGISTER
	TCC0.CNT = 0;
	//initialize tcc0 period register
	TCC0.PER = period + offset;
	//set tcc0 as event channel 0 source
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
};
```

```
void DMA_CH0_INIT(void)
{
	//set beginning of sinewave table as source address on ch0
	DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)sine_wave);
	DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)sine_wave)>>8);
	DMA.CH0.SRCADDR2 = (uint8_t)((uint32_t)(((uintptr_t)sine_wave)>>16));
	//set daca ch1 register as destination address
	DMA.CH0.DESTADDR0 = (uint8_t)((uintptr_t)&DACA.CH1DATA);
	DMA.CH0.DESTADDR1 = (uint8_t)(((uintptr_t)&DACA.CH1DATA)>>8);
	DMA.CH0.DESTADDR2 = (uint8_t)((uint32_t)(((uintptr_t)&DACA.CH1DATA)>>16));
	//set block transfer size on ch0 to 512 bytes
	DMA.CH0.TRFCNT = 512;
	//set burst size to 2 bytes and turn on repeat mode
	DMA.CH0.CTRLA |= (DMA_CH_BURSTLEN_2BYTE_gc |DMA_CH_REPEAT_bm);
	//enable single shot mode
	DMA.CH0.CTRLA |= DMA_CH_SINGLE_bm;
	//set dma ch0 to reload source address after block transfer and
	//destination address after burst transfer
	DMA.CH0.ADDRCTRL |= (DMA_CH_SRCRELOAD_BLOCK_gc |  DMA_CH_DESTRELOAD_BURST_gc);
	//set dma to increment source and destination address after byte transfer
	DMA.CH0.ADDRCTRL |= (DMA_CH_SRCDIR_INC_gc|DMA_CH_DESTDIR_INC_gc);
	//set dma ch0 as event channel 0 user
	DMA.CH0.TRIGSRC |= DMA_CH_TRIGSRC_EVSYS_CH0_gc;
	//enable dma ch0
	DMA.CTRL |= DMA_ENABLE_bm;
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;
}

void poweramp_init(void)
{
	//TURN ON THE POWER AMPLIFIER
	PORTC.OUTSET = poweramp_on;
	PORTC.DIRSET = poweramp_on;

}
```

SECTION 5

```
//********************************************
//Lab 8, Section 5
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: synthesizer!
//********************************************

#include <avr/io.h>
#include <avr/interrupt.h>
#define maxvoltage 4095 //2.5v converted to decimal
#define slope (2.5/4095) //0.000610500611 volts/decimalvalue
extern void clock_init(void);
extern uint16_t sine_wave[256];
extern uint16_t triangle_wave[256];
//global variables
volatile uint8_t poweramp_on = (0x01<<7);
volatile uint8_t wave_type = 0; //0 = sine/ 1 = triangle
volatile int8_t bsel = 4;
volatile int8_t bscale = 1;
volatile uint8_t DMA_CH_DISABLE_bm = (0x1<<7);
volatile uint8_t switch_wave_flag = 0;
volatile uint8_t switch_frequency_flag = 0;
volatile uint8_t key_pressed =0;
volatile uint8_t note = 0;
/////////////////////
int main(void)
{
	//set system clock to 32 mhz
	clock_init();
	//initialize DAC
	dac_init();
	//initialize power amplifier
	poweramp_init();
	//initialize timer counter
	tcc0_init();
	//initialize DMA system
	DMA_CH0_INIT();
	//initialize usart set baud rate to 200000
	usartd0_init();
	//enable global interrupts
	PMIC.CTRL = PMIC_MEDLVLEN_bm;
	sei();
```

```
	while (1)
	{
			//check if we need to switch waveforms
			if(switch_wave_flag == 1)
			{
				switch_wave_flag = 0;
				//switch to triangle wave
				if(wave_type == 1)
				{
					switch_waves(1);
				}
				//switch to sine wave
				else if (wave_type == 0)
				{
					switch_waves(0);
				}
			}
			else if(key_pressed)
			{
				//if key pressed, turn on timer
				TCC0.CTRLA = TC_CLKSEL_DIV2_gc;
				TCC0.PER = note;
				//keep timer on for certain amount of time
				for(volatile uint16_t i =0;i < 32767; i++)
				{
					while(!(TCC0.INTFLAGS &TC0_OVFIF_bm))
					{
						//do nothing
					}
				}
				//turn off timer
				TCC0.CTRLA = 0;
				key_pressed = 0;
			}

	}
}

void dac_init(void)
{
	//use only channel 1
	DACA.CTRLB = DAC_CHSEL_SINGLE1_gc;
	//use arefb
	DACA.CTRLC = DAC_REFSEL_AREFB_gc;
	//enable channel 1
	DACA.CTRLA = DAC_CH1EN_bm | DAC_ENABLE_bm;
}
void tcc0_init(void)
{
	uint16_t period = 63;
	int8_t offset = 0;
	//INITIALIZE COUNT REGISTER
	TCC0.CNT = 0;
	//initialize tcc0 period register
	TCC0.PER = period + offset;
	//set tcc0 as event channel 0 source
	EVSYS.CH0MUX = EVSYS_CHMUX_TCC0_OVF_gc;
};
```

```
void DMA_CH0_INIT(void)
{
	//RESET DMA SYSTEM
	DMA.CTRL = DMA_CH_DISABLE_bm;
	DMA.CTRL = DMA_RESET_bm;
	//set beginning of sinewave table as source address on ch0
	DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)sine_wave);
	DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)sine_wave)>>8);
	DMA.CH0.SRCADDR2 = (uint8_t)((uint32_t)(((uintptr_t)sine_wave)>>16));
	//set daca ch1 register as destination address
	DMA.CH0.DESTADDR0 = (uint8_t)((uintptr_t)&DACA.CH1DATA);
	DMA.CH0.DESTADDR1 = (uint8_t)(((uintptr_t)&DACA.CH1DATA)>>8);
	DMA.CH0.DESTADDR2 = (uint8_t)((uint32_t)(((uintptr_t)&DACA.CH1DATA)>>16));
	//set block transfer size on ch0 to 512 bytes
	DMA.CH0.TRFCNT = 512;
	//set burst size to 2 bytes and turn on repeat mode
	DMA.CH0.CTRLA = (DMA_CH_BURSTLEN_2BYTE_gc |DMA_CH_REPEAT_bm);
	//enable single shot mode
	DMA.CH0.CTRLA |= DMA_CH_SINGLE_bm;
	//set dma ch0 to reload source address after block transfer and
	//destination address after burst transfer
	DMA.CH0.ADDRCTRL = (DMA_CH_SRCRELOAD_BLOCK_gc |  DMA_CH_DESTRELOAD_BURST_gc);
	//set dma to increment source and destination address after byte transfer
	DMA.CH0.ADDRCTRL |= (DMA_CH_SRCDIR_INC_gc|DMA_CH_DESTDIR_INC_gc);
	//set dma ch0 as event channel 0 user
	DMA.CH0.TRIGSRC = DMA_CH_TRIGSRC_EVSYS_CH0_gc;
	//enable dma ch0
	DMA.CTRL = DMA_ENABLE_bm;
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;

}


void poweramp_init(void)
{
	//TURN ON THE POWER AMPLIFIER
	PORTC.OUTSET = poweramp_on;
	PORTC.DIRSET = poweramp_on;

}
```

```
void usartd0_init(void)
{
	//initialize transmitter and reciever pins
	PORTD.OUTSET = PIN3_bm;
	PORTD.DIRSET = PIN3_bm;
	PORTD.DIRCLR = PIN2_bm;

	//set baud rate
	USARTD0.BAUDCTRLA = (uint8_t)bsel;
	USARTD0.BAUDCTRLB = (uint8_t)((bscale << 4)|(bsel >> 8));

	//set to 8 bit odd parity with 1 stop bit
	USARTD0.CTRLC = 	(USART_CMODE_ASYNCHRONOUS_gc |USART_PMODE_ODD_gc|
USART_CHSIZE_8BIT_gc)&(~USART_SBMODE_bm);

	//ENABLE TRANSMITTER AND RECIEVER
	USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

	//enable reciever interrupts
	USARTD0.CTRLA = USART_RXCINTLVL_MED_gc;
}
```

```
ISR(USARTD0_RXC_vect)
{
	char data;
	data = USARTD0.DATA;
	if(data == 's' || data == 'S')
	{
		if(wave_type == 1)
		{
			//SWITCH TO SINE WAVE
			wave_type = 0;
			switch_wave_flag = 1;
		}
		else if(wave_type == 0)
		{
			//SWITCH TO   TRIANGLE WAVE
			wave_type = 1;
			switch_wave_flag = 1;
		}
	}
	//check for notes
	/*
	E = 61
	4 = 57
	R = 54
	5 = 51
	T = 48
	Y = 45
	7 = 43
	U = 40
	8 = 38
	I = 36
	9 = 34
	O = 32
	P = 30

*/
```

```
	else
	{
		if(data == 'e' || data == 'E')
		{
			key_pressed = 1;
			note = 61;
		}
		else if(data == '4' || data == '4')
		{
			key_pressed = 1;
			note = 57;
		}
		else if(data == 'r' || data == 'R')
		{
			key_pressed = 1;
			note = 54;
		}
		else if(data == '5' || data == '5')
		{
			key_pressed = 1;
			note = 51;
		}
		else if(data == 't' || data == 'T')
		{
			key_pressed = 1;
			note = 48;
		}
		else if(data == 'y' || data == 'Y')
		{
			key_pressed = 1;
			note = 45;
		}
		else if(data == '7' || data == '7')
		{
			key_pressed = 1;
			note = 43;
		}
		else if(data == 'u' || data == 'U')
		{
			key_pressed = 1;
			note = 40;
		}
		else if(data == '8' || data == '8')
		{
			key_pressed = 1;
			note = 38;
		}
		else if(data == 'i' || data == 'I')
		{
			key_pressed = 1;
			note = 36;
		}
		else if(data == '9' || data == '9')
		{
			key_pressed = 1;
			note = 34;
		}
```

```
            else if(data == 'o' || data == 'O')
            {
                  key_pressed = 1;
                  note = 32;
            }
            else if(data == 'p' || data == 'P')
            {
                  key_pressed = 1;
                  note = 30;
            }



      }

}

void switch_waves(uint8_t flag)
{
      //switch to sine
      if(flag == 0)
      {
            //initialize DMA system
            DMA.CTRL = DMA_CH_DISABLE_bm;
            DMA.CTRL = DMA_RESET_bm;
            //set beginning of sinewave table as source address on ch0
            DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)sine_wave);
            DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)sine_wave)>>8);
            DMA.CH0.SRCADDR2 = (uint8_t)((uint32_t)(((uintptr_t)sine_wave)>>16));
            //set daca ch1 register as destination address
            DMA.CH0.DESTADDR0 = (uint8_t)((uintptr_t)&DACA.CH1DATA);
            DMA.CH0.DESTADDR1 = (uint8_t)(((uintptr_t)&DACA.CH1DATA)>>8);
            DMA.CH0.DESTADDR2 = (uint8_t)((uint32_t)(((uintptr_t)&DACA.CH1DATA)>>16));
            //set block transfer size on ch0 to 512 bytes
            DMA.CH0.TRFCNT = 512;
            //set burst size to 2 bytes and turn on repeat mode
            DMA.CH0.CTRLA |= (DMA_CH_BURSTLEN_2BYTE_gc |DMA_CH_REPEAT_bm);
            //enable single shot mode
            DMA.CH0.CTRLA |= DMA_CH_SINGLE_bm;
            //set dma ch0 to reload source address after block transfer and
            //destination address after burst transfer
            DMA.CH0.ADDRCTRL |= (DMA_CH_SRCRELOAD_BLOCK_gc |  DMA_CH_DESTRELOAD_BURST_gc);
            //set dma to increment source and destination address after byte transfer
            DMA.CH0.ADDRCTRL |= (DMA_CH_SRCDIR_INC_gc|DMA_CH_DESTDIR_INC_gc);
            //set dma ch0 as event channel 0 user
            DMA.CH0.TRIGSRC |= DMA_CH_TRIGSRC_EVSYS_CH0_gc;
            //enable dma ch0
            DMA.CTRL |= DMA_ENABLE_bm;
            DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;
            wave_type = 0;
      }
```

```
        //switch to triangle
        else if(flag ==1)
        {
                //initialize DMA system
                DMA.CTRL = DMA_CH_DISABLE_bm;
                DMA.CTRL = DMA_RESET_bm;
                //set beginning of triangle wave table as source address on ch0
                DMA.CH0.SRCADDR0 = (uint8_t)((uintptr_t)triangle_wave);
                DMA.CH0.SRCADDR1 = (uint8_t)(((uintptr_t)triangle_wave)>>8);
                DMA.CH0.SRCADDR2 = (uint8_t)((uint32_t)(((uintptr_t)triangle_wave)>>16));
                //set daca ch1 register as destination address
                DMA.CH0.DESTADDR0 = (uint8_t)((uintptr_t)&DACA.CH1DATA);
                DMA.CH0.DESTADDR1 = (uint8_t)(((uintptr_t)&DACA.CH1DATA)>>8);
                DMA.CH0.DESTADDR2 = (uint8_t)((uint32_t)(((uintptr_t)&DACA.CH1DATA)>>16));
                //set block transfer size on ch0 to 512 bytes
                DMA.CH0.TRFCNT = 512;
                //set burst size to 2 bytes and turn on repeat mode
                DMA.CH0.CTRLA |= (DMA_CH_BURSTLEN_2BYTE_gc |DMA_CH_REPEAT_bm);
                //enable single shot mode
                DMA.CH0.CTRLA |= DMA_CH_SINGLE_bm;
                //set dma ch0 to reload source address after block transfer and
                //destination address after burst transfer
                DMA.CH0.ADDRCTRL |= (DMA_CH_SRCRELOAD_BLOCK_gc |  DMA_CH_DESTRELOAD_BURST_gc);
                //set dma to increment source and destination address after byte transfer
                DMA.CH0.ADDRCTRL |= (DMA_CH_SRCDIR_INC_gc|DMA_CH_DESTDIR_INC_gc);
                //set dma ch0 as event channel 0 user
                DMA.CH0.TRIGSRC |= DMA_CH_TRIGSRC_EVSYS_CH0_gc;
                //enable dma ch0
                DMA.CTRL |= DMA_ENABLE_bm;
                DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm;
                wave_type =1;
        }

}
```

APPENDIX

Figure 13: Screenshot of section 1

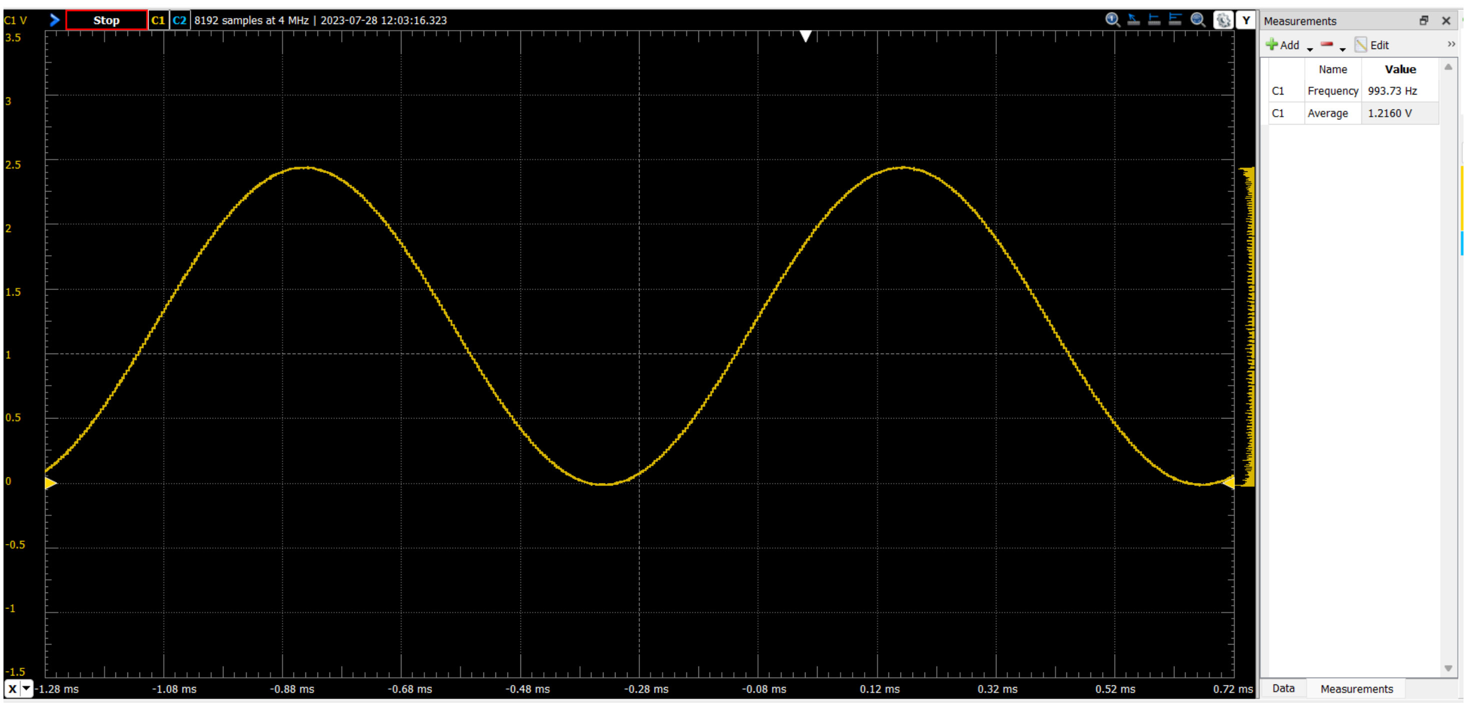

Figure 14: Screenshot of section 2a

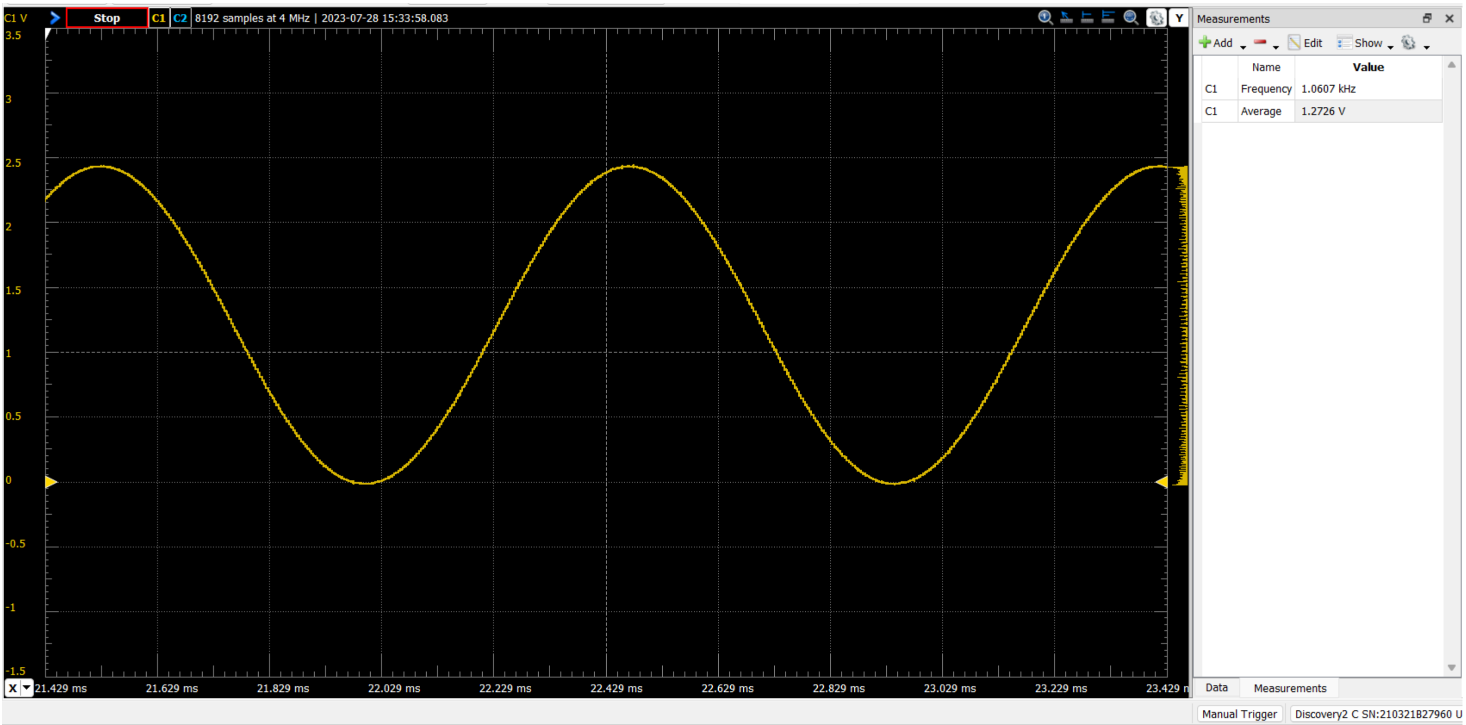

Figure 15: Screenshot of section 2a. The maximum frequency that can be obtained it 1.06khz

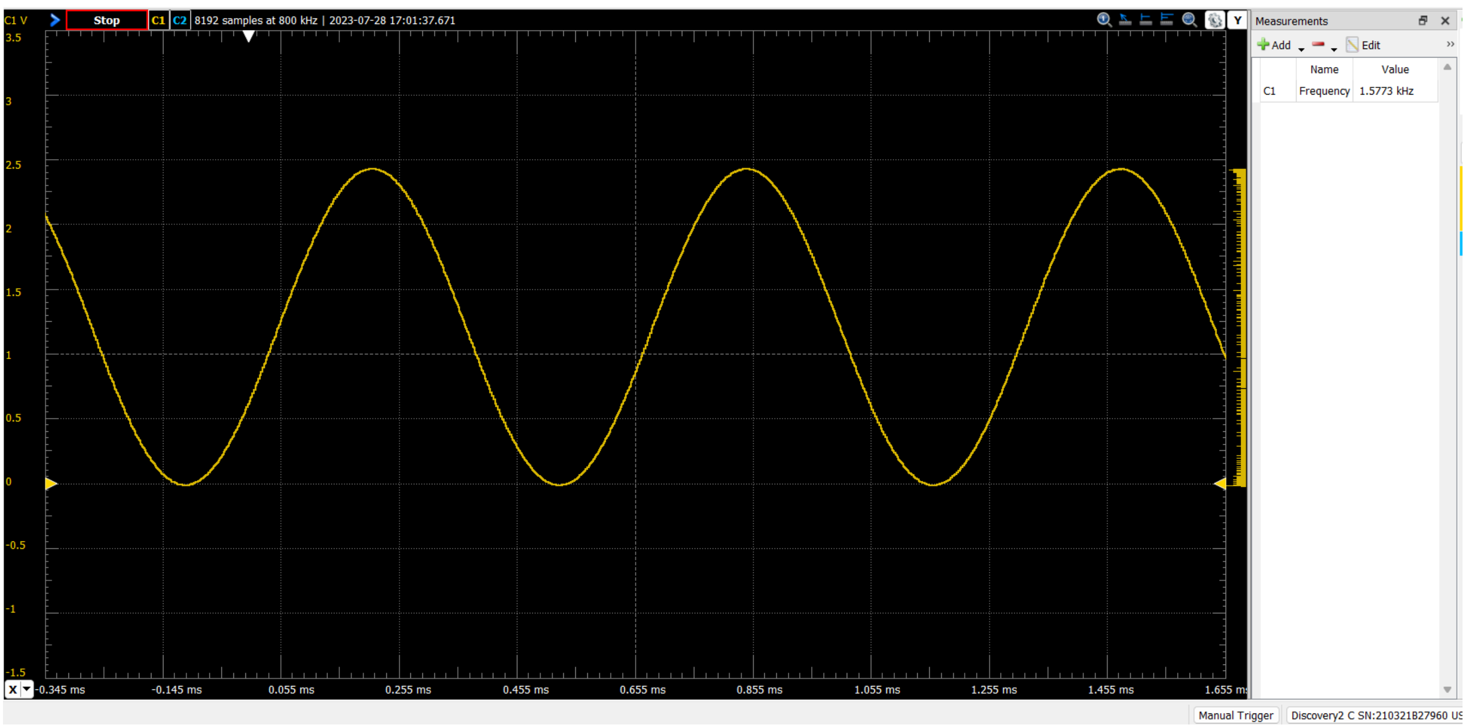

Figure 16: Screenshot of section 2b.

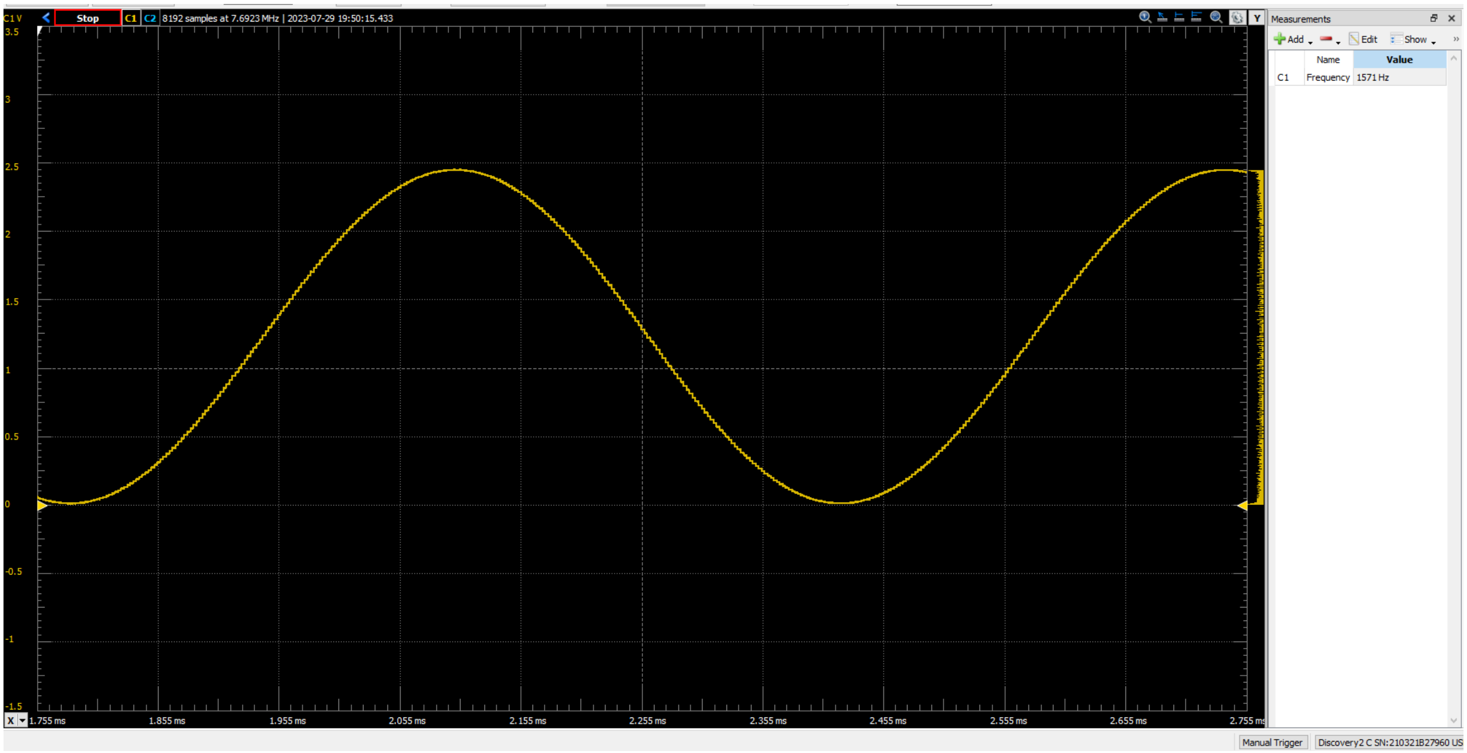

Figure 17: Screenshot of section 3

Clock.s

```
/* clock.s
 *
 * Last Updated: 21 June 2023
 * Authors: Wes, Dr. Schwartz
 */

 #include <avr/io.h>

 .section .text

 .global clock_init
 clock_init:
       push r24

       ldi r24, OSC_RC32MEN_bm
       sts OSC_CTRL, R24           ;Enables the 32MHz internal oscillator

check32MHzStatus:
       lds    r24, OSC_STATUS
       ;Ensure that the 32MHz clock is ready before proceeding
       sbrs r24, OSC_RC32MRDY_bp
       rjmp check32MHzStatus

       ;Writing to CCP disables interrupts for a certain number of cycles
       ;to give the clock time to switch sources. It also enables writes to certain registers.
       ldi r24, 0xD8
       sts CPU_CCP, r24

       ;Finally, select the now-ready 32MHz oscillator as the new clock source.
       ldi    r24, 0x01
       sts CLK_CTRL, r24

skip32MHZ_enable:
       ;CPU CLK prescaler settings
       ;Use values that are powers of 2 from 1 to 512 (1, 2, 4, 8, 16, ..., 512) for A. See Table 7-2 in
the manual.
       ;You can also change B/C. See Table 7-3 in the manual.

       ldi r24, 0xD8
       sts CPU_CCP, r24

       ldi r24, ((0x00 << 2) | (0x00 << 0))      ;32MHz
       ;ldi r24, ((0x05<<2) | (0x00<<0)) ;4MHz

       sts CLK_PSCTRL, r24

       pop r24

       ret
```

sinewave_table.C

```
//*******************************************
//sinewave_table.c
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: array that contains data points to make a sinewave from 0 to 4095
//*******************************************
#include <avr/io.h>
volatile uint16_t sine_wave[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
											};
```

triwave_table.C

```
//*******************************************
//triwave_table.c
//Name: Steven Miller
//Class #: 11318
//PI Name: Anthony Stross
//Description: array that contains data points to make a triangle wave from 0 to 4095
//*******************************************
#include <avr/io.h>
volatile uint16_t triangle_wave[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
									};
```