**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?**

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

**PSEUDOCODE/FLOWCHARTS**

**SECTION 1**

**A diagram of a computer system

Description automatically generated  
Figure 1: Flowchart for “lab8\_1.C”**

**SECTION 2**

**A diagram of a process

Description automatically generated  
  
Figure 2: Flowchart for “lab8\_2.C”**

**PROGRAM CODE**

**SECTION 1**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//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;

}

**APPENDIX**

A screen shot of a computer

Description automatically generated **Figure x:Screenshot of section 1**

**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