/\* 8 bit DAC backup

Name: Pengzhao Zhu

Section#: 112D

TA Name: Chris Crary

Description: This program generates a 1760 Hz sine waveform using a look-up data of 256 data points.

It will then output the signal to the speaker on the analog backpack continuously.

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void DAC(void);

void ADC(void);

void TIMER\_INIT(void);

void USARTD0\_init(void);

*uint8\_t* IN\_CHAR(void);

void OUT\_CHAR(*uint8\_t* data);

#define BSELHIGH (((4)\*((32000000/(16\*57600))-1))>>8) //bscale of -2

#define BSEL ((4)\*((32000000/(16\*57600))-1)) //bscale of -2

//double timer = (32000000/(256\*1200));

//double decimal (1/901120);

//double timer=((32000000)\*decimal);

*uint16\_t* Wave[256];

volatile *uint8\_t* j=0;

*uint8\_t* input;

/\* 12 bit ADC

const uint16\_t Sine[]= {

2048,2098,2148,2198,2248,2298,2348,2398,

2447,2496,2545,2594,2642,2690,2737,2784,

2831,2877,2923,2968,3013,3057,3100,3143,

3185,3226,3267,3307,3346,3385,3423,3459,

3495,3530,3565,3598,3630,3662,3692,3722,

3750,3777,3804,3829,3853,3876,3898,3919,

3939,3958,3975,3992,4007,4021,4034,4045,

4056,4065,4073,4080,4085,4089,4093,4094,

4095,4094,4093,4089,4085,4080,4073,4065,

4056,4045,4034,4021,4007,3992,3975,3958,

3939,3919,3898,3876,3853,3829,3804,3777,

3750,3722,3692,3662,3630,3598,3565,3530,

3495,3459,3423,3385,3346,3307,3267,3226,

3185,3143,3100,3057,3013,2968,2923,2877,

2831,2784,2737,2690,2642,2594,2545,2496,

2447,2398,2348,2298,2248,2198,2148,2098,

2048,1997,1947,1897,1847,1797,1747,1697,

1648,1599,1550,1501,1453,1405,1358,1311,

1264,1218,1172,1127,1082,1038,995,952,

910,869,828,788,749,710,672,636,

600,565,530,497,465,433,403,373,

345,318,291,266,242,219,197,176,

156,137,120,103,88,74,61,50,

39,30,22,15,10,6,2,1,

0,1,2,6,10,15,22,30,

39,50,61,74,88,103,120,137,

156,176,197,219,242,266,291,318,

345,373,403,433,465,497,530,565,

600,636,672,710,749,788,828,869,

910,952,995,1038,1082,1127,1172,1218,

1264,1311,1358,1405,1453,1501,1550,1599,

1648,1697,1747,1797,1847,1897,1947,1997

};

\*/

const *uint8\_t* Sine[]={

128,131,134,137,140,143,146,149,

152,155,158,162,165,167,170,173,

176,179,182,185,188,190,193,196,

198,201,203,206,208,211,213,215,

218,220,222,224,226,228,230,232,

234,235,237,238,240,241,243,244,

245,246,248,249,250,250,251,252,

253,253,254,254,254,255,255,255,

255,255,255,255,254,254,254,253,

253,252,251,250,250,249,248,246,

245,244,243,241,240,238,237,235,

234,232,230,228,226,224,222,220,

218,215,213,211,208,206,203,201,

198,196,193,190,188,185,182,179,

176,173,170,167,165,162,158,155,

152,149,146,143,140,137,134,131,

128,124,121,118,115,112,109,106,

103,100,97,93,90,88,85,82,

79,76,73,70,67,65,62,59,

57,54,52,49,47,44,42,40,

37,35,33,31,29,27,25,23,

21,20,18,17,15,14,12,11,

10,9,7,6,5,5,4,3,

2,2,1,1,1,0,0,0,

0,0,0,0,1,1,1,2,

2,3,4,5,5,6,7,9,

10,11,12,14,15,17,18,20,

21,23,25,27,29,31,33,35,

37,40,42,44,47,49,52,54,

57,59,62,65,67,70,73,76,

79,82,85,88,90,93,97,100,

103,106,109,112,115,118,121,124

};

const *uint8\_t* Saw[]= {

1,2,3,4,5,6,7,8,

9,10,11,12,13,14,15,16,

17,18,19,20,21,22,23,24,

25,26,27,28,29,30,31,32,

33,34,35,36,37,38,39,40,

41,42,43,44,45,46,47,48,

49,50,51,52,53,54,55,56,

57,58,59,60,61,62,63,64,

65,66,67,68,69,70,71,72,

73,74,75,76,77,78,79,80,

81,82,83,84,85,86,87,88,

89,90,91,92,93,94,95,96,

97,98,99,100,101,102,103,104,

105,106,107,108,109,110,111,112,

113,114,115,116,117,118,119,120,

121,122,123,124,125,126,127,128,

128,129,130,131,132,133,134,135,

136,137,138,139,140,141,142,143,

144,145,146,147,148,149,150,151,

152,153,154,155,156,157,158,159,

160,161,162,163,164,165,166,167,

168,169,170,171,172,173,174,175,

176,177,178,179,180,181,182,183,

184,185,186,187,188,189,190,191,

192,193,194,195,196,197,198,199,

200,201,202,203,204,205,206,207,

208,209,210,211,212,213,214,215,

216,217,218,219,220,221,222,223,

224,225,226,227,228,229,230,231,

232,233,234,235,236,237,238,239,

240,241,242,243,244,245,246,247,

248,249,250,251,252,253,254,255

};

/\* 12 bit ADC

const uint16\_t Saw[]= {

16,32,48,64,80,96,112,128,

144,160,176,192,208,224,240,256,

272,288,304,320,336,352,368,384,

400,416,432,448,464,480,496,512,

528,544,560,576,592,608,624,640,

656,672,688,704,720,736,752,768,

784,800,816,832,848,864,880,896,

912,928,944,960,976,992,1008,1024,

1040,1056,1072,1088,1104,1120,1136,1152,

1168,1184,1200,1216,1232,1248,1264,1280,

1296,1312,1328,1344,1360,1376,1392,1408,

1424,1440,1456,1472,1488,1504,1520,1536,

1552,1568,1584,1600,1616,1632,1648,1664,

1680,1696,1712,1728,1744,1760,1776,1792,

1808,1824,1840,1856,1872,1888,1904,1920,

1936,1952,1968,1984,2000,2016,2032,2048,

2063,2079,2095,2111,2127,2143,2159,2175,

2191,2207,2223,2239,2255,2271,2287,2303,

2319,2335,2351,2367,2383,2399,2415,2431,

2447,2463,2479,2495,2511,2527,2543,2559,

2575,2591,2607,2623,2639,2655,2671,2687,

2703,2719,2735,2751,2767,2783,2799,2815,

2831,2847,2863,2879,2895,2911,2927,2943,

2959,2975,2991,3007,3023,3039,3055,3071,

3087,3103,3119,3135,3151,3167,3183,3199,

3215,3231,3247,3263,3279,3295,3311,3327,

3343,3359,3375,3391,3407,3423,3439,3455,

3471,3487,3503,3519,3535,3551,3567,3583,

3599,3615,3631,3647,3663,3679,3695,3711,

3727,3743,3759,3775,3791,3807,3823,3839,

3855,3871,3887,3903,3919,3935,3951,3967,

3983,3999,4015,4031,4047,4063,4079,4095

};

\*/

int main(void) {

//output frequency=sample rate(Hz)/ size of table

//how fast you need to sample 512 to get (1/1760) when you finished the whole table

//(1/1760)=512(1/x). x is the number in Hz

//sample rate(Hz)=output frequency x No. samples

CLK\_32MHZ();

TIMER\_INIT();

DAC();

USARTD0\_init();

PORTA\_DIRSET=PIN3\_bm; //set PA3 as DAC1 output

PORTC\_DIRSET=PIN7\_bm; //set POWER DOWN pin as output

PORTC\_OUTSET=PIN7\_bm; //set POWER DOWN pin always high to prevent shutdown

while(1) {

CHECK:;

input=IN\_CHAR();

OUT\_CHAR(input);

//TCC0\_CTRLA=TC\_CLKSEL\_OFF\_gc;

if ((input != 'S') && (input != 'W')) {

goto CHECK;

}

/\*

if ((input != 'S') && (input != 'W') && (input != '3') && (input != 'E') && (input != '4') && (input != 'R')

&& (input != 'T') && (input !='6') && (input !='Y') && (input != '7') && (input != 'U') && (input != '8') && (input != 'I')) {

goto CHECK;

}

\*/

if (input=='S') {

for (int i=0; i< 256; i++) {

Wave[i]=Sine[i];

}

}

if (input=='W') {

for (int i=0; i< 256; i++) {

Wave[i]=Saw[i];

}

}

}

//TCC0\_CTRLA=TC\_CLKSEL\_DIV1\_gc;

return 0;

}

void DAC(void) {

DACA\_CTRLA= DAC\_ENABLE\_bm | DAC\_CH1EN\_bm ; //enable DAC, enable channel 1 output

DACA\_CTRLB=DAC\_CHSEL\_SINGLE1\_gc; //single-channel operation on channel 1

DACA\_CTRLC=DAC\_REFSEL\_AREFB\_gc | DAC\_LEFTADJ\_bm; //AREF on PORTB as reference and left adjusted

}

void TIMER\_INIT(void) {

TCC0\_CNT=0x0000; //set CNT to zero

TCC0\_PER=208; //timer per value to output 1760 Hz sine wave

TCC0\_CTRLA=TC\_CLKSEL\_DIV1\_gc; //timer prescaler of 1

//TCC0\_CTRLA=TC\_CLKSEL\_DIV1024\_gc;

TCC0\_INTCTRLA=0x01;

PMIC\_CTRL=0x01;

sei();

}

void CLK\_32MHZ(void)

{

OSC\_CTRL=0x02; //select the 32Mhz osciliator

while ( ((OSC\_STATUS) & 0x02) != 0x02 ); //check if 32Mhz oscillator is stable

//if not stable. keep looping

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_CTRL= 0x01; //select the 32Mhz oscillator

CPU\_CCP= 0xD8; //write IOREG to CPU\_CCP to enable change

CLK\_PSCTRL= 0x00; //0x00 for the prescaler

}

void USARTD0\_init(void)

{

PORTD\_DIRSET=0x08; //set transmitter as output

PORTD\_DIRCLR=0X04; //set receiver as input

USARTD0\_CTRLB=0x18; //enable receiver and transmitter

USARTD0\_CTRLC= 0X33; //USART asynchronous, 8 data bit, odd parity, 1 stop bit

USARTD0\_BAUDCTRLA= (*uint8\_t*) BSEL; //load lowest 8 bits of BSEL

USARTD0\_BAUDCTRLB= (((*uint8\_t*) BSELHIGH) | 0xE0); //load BSCALE and upper 4 bits of BSEL. bitwise OR them

PORTD\_OUTSET= 0x08; //set transit pin idle

}

*uint8\_t* IN\_CHAR(void) {

while( (USARTD0\_STATUS & 0x80) != 0x80); //keep looping if DREIF flag is not set

return USARTD0\_DATA;

}

void OUT\_CHAR(*uint8\_t* data) {

while( ((USARTD0\_STATUS) & 0x20) != 0x20); //keep looping if DREIF flag is not set

USARTD0\_DATA= (*uint8\_t*) data;

}

ISR(TCC0\_OVF\_vect) {

DACA\_CH1DATAH=Wave[j];

j++;

}