Lab 6 backup code

**Part A First Backup Code**

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

volatile *int16\_t* adc;

int main(void)

{

CLK\_32MHZ();

ADC();

//int16\_t adc; //8 bit adc. y=(adc/51)+.009804. .5 V= adc of

while(1) {

//while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);

adc=ADCA\_CH0\_RES;

//ADCA\_CH0\_INTFLAGS=0x01;

}

return 0;

}

void ADC(void) {

/\*

PORTA\_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CTRLA=0x01; //enable ADC

ADCA\_CTRLB= 0b00010100; //signed mode, free running, and 8 bit right adjusted

ADCA\_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5

ADCA\_PRESCALER=0b00000000; //adc prescaler of 512

ADCA\_CH0\_CTRL=0b00000011; //start channel 0 conversion, 1x gain, differential input signal with gain

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

\*/

/\*

PORTA\_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CTRLA=0x01; //enable ADC

ADCA\_CTRLB= 0b00011100; //signed mode, free running, and 8 bit right adjusted

ADCA\_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5

ADCA\_PRESCALER=0b00000111; //adc prescaler of 512

ADCA\_CH0\_CTRL=0b10000011; //start channel 0 conversion, 1x gain, differential input signal with gain

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

\*/

ADCA\_REFCTRL=ADC\_REFSEL\_AREFB\_gc; //adc reference as PORTB aref. start scanning on channel 0

ADCA\_PRESCALER=ADC\_PRESCALER\_DIV128\_gc; //512 prescaler or adc clock

ADCA\_CTRLB=ADC\_CONMODE\_bm | ADC\_RESOLUTION\_8BIT\_gc | ADC\_FREERUN\_bm; //signed mode, 12 bit resolution, free run

PORTA\_DIRCLR= 0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CH0\_CTRL=ADC\_CH\_GAIN\_1X\_gc | ADC\_CH\_INPUTMODE\_DIFFWGAIN\_gc;

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

ADCA\_CTRLA=ADC\_ENABLE\_bm|ADC\_CH0START\_bm;

}

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

}

**Part B**  
#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <float.h>

void CLK\_32MHZ(void);

void ADC(void);

void USART\_INIT(void);

void TIMER\_INIT(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

#define timer\_100 (32000000\*.1)/1024

*int16\_t* adc;

*uint8\_t* sign; //for +, -, or neither

float voltage;

float voltage2;

float voltage3;

int int1;

int int2;

int int3;

int int1\_send;

int int2\_send;

int int3\_send;

*uint8\_t* hex1\_send;

*uint8\_t* hex2\_send;

*uint8\_t* adc\_send;

int main(void)

{

CLK\_32MHZ();

ADC();

USART\_INIT();

TIMER\_INIT();

//8 bit adc. y=(adc/51)+(1/102). .5 V= adc of

//y=(1/819)x + (1/1638) for 12 bit

//while(1) { //uncomment for full part b code

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrup flag of 100 ms for TCC0

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

//while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01); //wait for adc conversion to be completed

adc=ADCA\_CH0\_RES; //take adc value

//ADCA\_CH0\_INTFLAGS=0x01; //clear adc interrupt flag

if (adc < 0) {

sign='-';

} else if (adc > 0) {

sign='+';

} else if (adc==0) {

sign=' ';

}

OUT\_CHAR(sign); //transmit positive or negative sign

voltage = ( (((float)adc)/51)+(1/102)); //get floating point voltage value

if (voltage<0) {

voltage=voltage\*(-1); //so voltage value will always be positive when i am doing math later

}

int1 = (int) voltage; //transmit the tenth place

int1\_send = int1+48; //from number to ascii according to the ascii table

OUT\_CHAR(int1\_send);

OUT\_CHAR('.');

voltage2=10\*(((float)voltage)-int1); //transmit the first decimal place

int2= (int) voltage2;

int2\_send= int2+48; //from number to ascii according to the ascii table

OUT\_CHAR(int2\_send);

voltage3=10\*(((float)voltage2)-int2); //transmit the second decimal place

int3= (int) voltage3;

int3\_send=int3+48; //from number to ascii according to the ascii table

OUT\_CHAR(int3\_send);

OUT\_CHAR(' ');

OUT\_CHAR('V');

OUT\_CHAR(' ');

OUT\_CHAR('(');

OUT\_CHAR('0');

OUT\_CHAR('x');

adc\_send= adc>>4; //take the upper byte of the 8 bit

adc\_send=adc\_send & 0x0F;

if ( adc\_send >= 10) { //if it is a character, add 55 (ascii table)

hex1\_send=adc\_send+55;

} else if (adc\_send < 10) { //if it is a number, add 48 (ascii table)

hex1\_send=adc\_send +48;

}

OUT\_CHAR(hex1\_send);

adc\_send= adc; //take the lower byte of the 8 bit

adc\_send=adc\_send & 0x0F;

if ( adc\_send >= 10) { //if it is a character, add 55 (ascii table)

hex2\_send=adc\_send+55;

} else if (adc\_send < 10) { //if it is a number, add 48 (ascii table)

hex2\_send=adc\_send +48;

}

OUT\_CHAR(hex2\_send);

OUT\_CHAR(')');

OUT\_CHAR(' ');

OUT\_CHAR(' ');

OUT\_CHAR(' ');

//} //uncomment for full part B code

return 0;

}

void ADC(void) {

/\*

PORTA\_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CTRLA=0x01; //enable ADC

ADCA\_CTRLB= 0b00011100; //signed mode, free running, and 8 bit right adjusted

ADCA\_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5

ADCA\_PRESCALER=0b00000111; //adc prescaler of 512

ADCA\_CH0\_CTRL=0b10000011; //start channel 0 conversion, 1x gain, differential input signal with gain

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

\*/

ADCA\_REFCTRL=ADC\_REFSEL\_AREFB\_gc; //adc reference as PORTB aref. start scanning on channel 0

ADCA\_PRESCALER=ADC\_PRESCALER\_DIV512\_gc; //512 prescaler or adc clock

ADCA\_CTRLB=ADC\_CONMODE\_bm | ADC\_RESOLUTION\_8BIT\_gc | ADC\_FREERUN\_bm; //signed mode, 12 bit resolution, free run

PORTA\_DIRCLR= 0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CH0\_CTRL=ADC\_CH\_GAIN\_1X\_gc | ADC\_CH\_INPUTMODE\_DIFFWGAIN\_gc;

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

ADCA\_CTRLA=ADC\_ENABLE\_bm|ADC\_CH0START\_bm;

}

void USART\_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

}

void TIMER\_INIT(void) {

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

TCC0\_PER=(*uint16\_t*) timer\_100; //timer per value to 100 ms

TCC0\_CTRLA=0b00000111; //timer prescaler of 1024

}

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

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

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

}

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

}

Part A Real

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

volatile *int16\_t* adc;

int main(void)

{

CLK\_32MHZ();

ADC();

//int16\_t adc; //8 bit adc. y=(adc/51)+.009804. .5 V= adc of

while(1) {

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);

adc=ADCA\_CH0\_RES;

ADCA\_CH0\_INTFLAGS=0x01;

}

return 0;

}

void ADC(void) {

/\*

PORTA\_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CTRLA=0x01; //enable ADC

ADCA\_CTRLB= 0b00010100; //signed mode, free running, and 8 bit right adjusted

ADCA\_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5

ADCA\_PRESCALER=0b00000000; //adc prescaler of 512

ADCA\_CH0\_CTRL=0b00000011; //start channel 0 conversion, 1x gain, differential input signal with gain

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

\*/

/\*

PORTA\_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CTRLA=0x01; //enable ADC

ADCA\_CTRLB= 0b00011100; //signed mode, free running, and 8 bit right adjusted

ADCA\_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5

ADCA\_PRESCALER=0b00000111; //adc prescaler of 512

ADCA\_CH0\_CTRL=0b10000011; //start channel 0 conversion, 1x gain, differential input signal with gain

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

\*/

ADCA\_REFCTRL=ADC\_REFSEL\_AREFB\_gc; //adc reference as PORTB aref. start scanning on channel 0

ADCA\_PRESCALER=ADC\_PRESCALER\_DIV128\_gc; //512 prescaler or adc clock

ADCA\_CTRLB=ADC\_CONMODE\_bm | ADC\_RESOLUTION\_8BIT\_gc | ADC\_FREERUN\_bm; //signed mode, 12 bit resolution, free run

PORTA\_DIRCLR= 0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CH0\_CTRL=ADC\_CH\_GAIN\_1X\_gc | ADC\_CH\_INPUTMODE\_DIFFWGAIN\_gc;

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

ADCA\_CTRLA=ADC\_ENABLE\_bm|ADC\_CH0START\_bm;

}

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

}

Part B Real

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

void USART\_INIT(void);

void TIMER\_INIT(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

#define timer\_100 (32000000\*.1)/1024

*int16\_t* adc;

*uint8\_t* sign; //for +, -, or neither

float voltage;

float voltage2;

float voltage3;

int int1;

int int2;

int int3;

int int1\_send;

int int2\_send;

int int3\_send;

*uint8\_t* hex1\_send;

*uint8\_t* hex2\_send;

*uint8\_t* adc\_send;

int main(void)

{

CLK\_32MHZ();

ADC();

USART\_INIT();

TIMER\_INIT();

//8 bit adc. y=(adc/51)+(1/102). .5 V= adc of

//y=(1/819)x + (1/1638) for 12 bit

//while(1) { //uncomment for full part b code

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrup flag of 100 ms for TCC0

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01); //wait for adc conversion to be completed

adc=ADCA\_CH0\_RES; //take adc value

ADCA\_CH0\_INTFLAGS=0x01; //clear adc interrupt flag

if (adc < 0) {

sign='-';

} else if (adc > 0) {

sign='+';

} else if (adc==0) {

sign=' ';

}

OUT\_CHAR(sign); //transmit positive or negative sign

voltage = ( (((float)adc)/51)+(1/102)); //get floating point voltage value

if (voltage<0) {

voltage=voltage\*(-1); //so voltage value will always be positive when i am doing math later

}

int1 = (int) voltage; //transmit the tenth place

int1\_send = int1+48; //from number to ascii according to the ascii table

OUT\_CHAR(int1\_send);

OUT\_CHAR('.');

voltage2=10\*(((float)voltage)-int1); //transmit the first decimal place

int2= (int) voltage2;

int2\_send= int2+48; //from number to ascii according to the ascii table

OUT\_CHAR(int2\_send);

voltage3=10\*(((float)voltage2)-int2); //transmit the second decimal place

int3= (int) voltage3;

int3\_send=int3+48; //from number to ascii according to the ascii table

OUT\_CHAR(int3\_send);

OUT\_CHAR(' ');

OUT\_CHAR('V');

OUT\_CHAR(' ');

OUT\_CHAR('(');

OUT\_CHAR('0');

OUT\_CHAR('x');

adc\_send= adc>>4; //take the upper byte of the 8 bit

adc\_send=adc\_send & 0x0F;

if ( adc\_send >= 10) { //if it is a character, add 55 (ascii table)

hex1\_send=adc\_send+55;

} else if (adc\_send < 10) { //if it is a number, add 48 (ascii table)

hex1\_send=adc\_send +48;

}

OUT\_CHAR(hex1\_send);

adc\_send= adc; //take the lower byte of the 8 bit

adc\_send=adc\_send & 0x0F;

if ( adc\_send >= 10) { //if it is a character, add 55 (ascii table)

hex2\_send=adc\_send+55;

} else if (adc\_send < 10) { //if it is a number, add 48 (ascii table)

hex2\_send=adc\_send +48;

}

OUT\_CHAR(hex2\_send);

OUT\_CHAR(')');

OUT\_CHAR(' ');

OUT\_CHAR(' ');

OUT\_CHAR(' ');

//} //uncomment for full part B code

return 0;

}

void ADC(void) {

/\*

PORTA\_DIRCLR=0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CTRLA=0x01; //enable ADC

ADCA\_CTRLB= 0b00011100; //signed mode, free running, and 8 bit right adjusted

ADCA\_REFCTRL=0b00110000; //arefb are the voltage reference of 2.5

ADCA\_PRESCALER=0b00000111; //adc prescaler of 512

ADCA\_CH0\_CTRL=0b10000011; //start channel 0 conversion, 1x gain, differential input signal with gain

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

\*/

ADCA\_REFCTRL=ADC\_REFSEL\_AREFB\_gc; //adc reference as PORTB aref. start scanning on channel 0

ADCA\_PRESCALER=ADC\_PRESCALER\_DIV512\_gc; //512 prescaler or adc clock

ADCA\_CTRLB=ADC\_CONMODE\_bm | ADC\_RESOLUTION\_8BIT\_gc | ADC\_FREERUN\_bm; //signed mode, 12 bit resolution, free run

PORTA\_DIRCLR= 0b01000010; //PA1 as positive input, PA6 as negative input. used later for cds cell

ADCA\_CH0\_CTRL=ADC\_CH\_GAIN\_1X\_gc | ADC\_CH\_INPUTMODE\_DIFFWGAIN\_gc;

ADCA\_CH0\_MUXCTRL=0b00001010; //muxcontrol for PA1 as positive, PA6 as negative

ADCA\_CTRLA=ADC\_ENABLE\_bm|ADC\_CH0START\_bm;

}

void USART\_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

}

void TIMER\_INIT(void) {

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

TCC0\_PER=(*uint16\_t*) timer\_100; //timer per value to 100 ms

TCC0\_CTRLA=0b00000111; //timer prescaler of 1024

}

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

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

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

}

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

}

Part C

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void DAC(void);

int main(void) {

CLK\_32MHZ();

DAC(); //initialize DAC

//VDAC=(CHDATA/0xFFF) x VREF

PORTA\_DIRSET=0x04; //set PA2 as DAC0 output

DACA\_CH0DATA=1638; //DAC output value according to the formula

while(1);

return 0;

}

void DAC(void) {

//DACA\_CTRLA= DAC\_ENABLE\_bm | DAC\_CH0EN\_bm ; //enable DAC, enable channel 0 output

//DACA\_CTRLB=DAC\_CHSEL\_SINGLE\_gc; //single-channel operation on channel 0

//DACA\_CTRLB=DAC\_REFSEL\_AREFB\_gc; //AREF on PORTB as reference

//i tried used the group configuration. but they are not right.

DACA\_CTRLA= 0b00000101; //enable DAC, enable channel 0 output

DACA\_CTRLB= 0x00; //single-channel operation on channel 0

DACA\_CTRLC= 0b00011000; //AREF on PORTB as reference

}

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

}

Part D Code

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void DAC(void);

void ADC(void);

void TIMER\_INIT(void);

//#define timer\_freq ((32000000)\*(1/901120))

//#define timer\_freq ((32000000)\*.1)/1024

//double decimal (1/901120);

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

const *uint16\_t* Table[]= {

2048,2073,2098,2123,2148,2173,2198,2223,

2248,2273,2298,2323,2348,2373,2398,2422,

2447,2472,2496,2521,2545,2569,2594,2618,

2642,2666,2690,2714,2737,2761,2784,2808,

2831,2854,2877,2900,2923,2946,2968,2990,

3013,3035,3057,3078,3100,3122,3143,3164,

3185,3206,3226,3247,3267,3287,3307,3327,

3346,3366,3385,3404,3423,3441,3459,3477,

3495,3513,3530,3548,3565,3581,3598,3614,

3630,3646,3662,3677,3692,3707,3722,3736,

3750,3764,3777,3791,3804,3816,3829,3841,

3853,3865,3876,3888,3898,3909,3919,3929,

3939,3949,3958,3967,3975,3984,3992,3999,

4007,4014,4021,4027,4034,4040,4045,4051,

4056,4060,4065,4069,4073,4076,4080,4083,

4085,4087,4089,4091,4093,4094,4094,4095,

4095,4095,4094,4094,4093,4091,4089,4087,

4085,4083,4080,4076,4073,4069,4065,4060,

4056,4051,4045,4040,4034,4027,4021,4014,

4007,3999,3992,3984,3975,3967,3958,3949,

3939,3929,3919,3909,3898,3888,3876,3865,

3853,3841,3829,3816,3804,3791,3777,3764,

3750,3736,3722,3707,3692,3677,3662,3646,

3630,3614,3598,3581,3565,3548,3530,3513,

3495,3477,3459,3441,3423,3404,3385,3366,

3346,3327,3307,3287,3267,3247,3226,3206,

3185,3164,3143,3122,3100,3078,3057,3035,

3013,2990,2968,2946,2923,2900,2877,2854,

2831,2808,2784,2761,2737,2714,2690,2666,

2642,2618,2594,2569,2545,2521,2496,2472,

2447,2422,2398,2373,2348,2323,2298,2273,

2248,2223,2198,2173,2148,2123,2098,2073,

2048,2022,1997,1972,1947,1922,1897,1872,

1847,1822,1797,1772,1747,1722,1697,1673,

1648,1623,1599,1574,1550,1526,1501,1477,

1453,1429,1405,1381,1358,1334,1311,1287,

1264,1241,1218,1195,1172,1149,1127,1105,

1082,1060,1038,1017,995,973,952,931,

910,889,869,848,828,808,788,768,

749,729,710,691,672,654,636,618,

600,582,565,547,530,514,497,481,

465,449,433,418,403,388,373,359,

345,331,318,304,291,279,266,254,

242,230,219,207,197,186,176,166,

156,146,137,128,120,111,103,96,

88,81,74,68,61,55,50,44,

39,35,30,26,22,19,15,12,

10,8,6,4,2,1,1,0,

0,0,1,1,2,4,6,8,

10,12,15,19,22,26,30,35,

39,44,50,55,61,68,74,81,

88,96,103,111,120,128,137,146,

156,166,176,186,197,207,219,230,

242,254,266,279,291,304,318,331,

345,359,373,388,403,418,433,449,

465,481,497,514,530,547,565,582,

600,618,636,654,672,691,710,729,

749,768,788,808,828,848,869,889,

910,931,952,973,995,1017,1038,1060,

1082,1105,1127,1149,1172,1195,1218,1241,

1264,1287,1311,1334,1358,1381,1405,1429,

1453,1477,1501,1526,1550,1574,1599,1623,

1648,1673,1697,1722,1747,1772,1797,1822,

1847,1872,1897,1922,1947,1972,1997,2022

};

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();

// int arr[100]={1,2,3,4,5};

//int size = sizeof(arr)/sizeof(arr[0]);

// to find number of elements in an array

PORTA\_DIRSET=0x04; //set PA2 as DAC0 output

while(1) {

for (int i=0; i< 512;i++) { //go through the 1024 samples

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

DACA\_CH0DATA=Table[i]; //DAC output value according to the formula

TCC0\_CNT=0x00; //reset TCC0\_CNT to 0

}

}

return 0;

}

void DAC(void) {

DACA\_CTRLA= DAC\_ENABLE\_bm | DAC\_CH0EN\_bm ; //enable DAC, enable channel 0 output

DACA\_CTRLB=DAC\_CHSEL\_SINGLE\_gc; //single-channel operation on channel 0

DACA\_CTRLC=DAC\_REFSEL\_AREFB\_gc; //AREF on PORTB as reference

}

void TIMER\_INIT(void) {

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

TCC0\_PER=19; //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;

}

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

}

Part E

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void DAC(void);

void ADC(void);

void TIMER\_INIT(void);

#define timer\_freq ((32000000)\*(1/450560))

//#define timer\_freq ((32000000)\*.1)/1024

//double decimal (1/901120);

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

const *uint16\_t* Table[]= {

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

};

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();

// int arr[100]={1,2,3,4,5};

//int size = sizeof(arr)/sizeof(arr[0]);

// to find number of elements in an array

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) {

for (int i=0; i< 256;i++) { //go through the 512 samples

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

DACA\_CH1DATA=Table[i]; //DAC output value according to the formula

TCC0\_CNT=0x00; //reset TCC0\_CNT to 0

}

}

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; //AREF on PORTB as reference

}

void TIMER\_INIT(void) {

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

TCC0\_PER=54; //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;

}

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

}

PART F

#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

#define timer\_freq ((32000000)\*(1/450560))

//#define timer\_freq ((32000000)\*.1)/1024

//double decimal (1/901120);

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

*uint8\_t* input;

int change=2;

const *uint16\_t* Table[]= {

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,

};

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);

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') {

change=change \*(-1); //2 means sine, -2 means sawtooth

goto CHECK;

}

if ((input=='W') && (change==2)) {

TCC0\_PER=103;

} else if ((input=='W') && (change==-2)) {

TCC0\_PER=112;

}

if ((input=='3') && (change==2)) {

TCC0\_PER=95;

} else if ((input=='3') && (change==-2)) {

TCC0\_PER=103;

}

if ((input=='E') && (change==2)) {

TCC0\_PER=91;

} else if ((input=='E') && (change==-2)) {

TCC0\_PER=97;

}

if ((input=='4') && (change==2)) {

TCC0\_PER=85;

} else if ((input=='4') && (change==-2)) {

TCC0\_PER=90;

}

if ((input=='R') && (change==2)) {

TCC0\_PER=77;

} else if ((input=='R') && (change==-2)) {

TCC0\_PER=85;

}

if ((input=='T') && (change==2)) {

TCC0\_PER=72;

} else if ((input=='T') && (change==-2)) {

TCC0\_PER=79;

}

if ((input=='6') && (change==2)) {

TCC0\_PER=69;

} else if ((input=='6') && (change==-2)) {

TCC0\_PER=75;

}

if ((input=='Y') && (change==2)) {

TCC0\_PER=61;

} else if ((input=='Y') && (change==-2)) {

TCC0\_PER=71;

}

if ((input=='7') && (change==2)) {

TCC0\_PER=57;

} else if ((input=='7') && (change==-2)) {

TCC0\_PER=66;

}

if ((input=='U') && (change==2)) {

TCC0\_PER=54;

} else if ((input=='U') && (change==-2)) {

TCC0\_PER=62;

}

if ((input=='8') && (change==2)) {

TCC0\_PER=50;

} else if ((input=='8') && (change==-2)) {

TCC0\_PER=58;

}

if ((input=='I') && (change==2)) {

TCC0\_PER=46;

} else if ((input=='I') && (change==-2)) {

TCC0\_PER=54;

}

TCC0\_CNT=0x00;

if (change==2) {

for(int i=0; i< 100;i++){

for (int i=0; i< 256;i++) { //go through the 512 samples

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

DACA\_CH1DATA=Table[i]; //DAC output value according to the formula

TCC0\_CNT=0x00; //reset TCC0\_CNT to 0

}

i++;

}

}

if(change==-2) {

for(int i=0; i< 100;i++){

for (int i=0; i< 256;i++) { //go through the 512 samples

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

float sawtooth=i\*(273/17);

DACA\_CH1DATA=(int) sawtooth; //DAC output value according to the formula

TCC0\_CNT=0x00; //reset TCC0\_CNT to 0

}

i++;

}

}

}

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; //AREF on PORTB as reference

}

void TIMER\_INIT(void) {

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

TCC0\_PER=0; //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;

}

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;

}

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

}

BACKUP AGAIN

#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

#define timer\_freq ((32000000)\*(1/450560))

//#define timer\_freq ((32000000)\*.1)/1024

//double decimal (1/901120);

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

*uint8\_t* input;

int change=2;

const *uint16\_t* Table[]= {

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,

};

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);

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') {

change=change \*(-1); //2 means sine, -2 means sawtooth

goto CHECK;

}

if ((input=='W') && (change==2)) {

TCC0\_PER=103;

} else if ((input=='W') && (change==-2)) {

TCC0\_PER=112;

}

if ((input=='3') && (change==2)) {

TCC0\_PER=95;

} else if ((input=='3') && (change==-2)) {

TCC0\_PER=103;

}

if ((input=='E') && (change==2)) {

TCC0\_PER=91;

} else if ((input=='E') && (change==-2)) {

TCC0\_PER=97;

}

if ((input=='4') && (change==2)) {

TCC0\_PER=85;

} else if ((input=='4') && (change==-2)) {

TCC0\_PER=90;

}

if ((input=='R') && (change==2)) {

TCC0\_PER=77;

} else if ((input=='R') && (change==-2)) {

TCC0\_PER=85;

}

if ((input=='T') && (change==2)) {

TCC0\_PER=72;

} else if ((input=='T') && (change==-2)) {

TCC0\_PER=79;

}

if ((input=='6') && (change==2)) {

TCC0\_PER=69;

} else if ((input=='6') && (change==-2)) {

TCC0\_PER=75;

}

if ((input=='Y') && (change==2)) {

TCC0\_PER=61;

} else if ((input=='Y') && (change==-2)) {

TCC0\_PER=71;

}

if ((input=='7') && (change==2)) {

TCC0\_PER=57;

} else if ((input=='7') && (change==-2)) {

TCC0\_PER=66;

}

if ((input=='U') && (change==2)) {

TCC0\_PER=54;

} else if ((input=='U') && (change==-2)) {

TCC0\_PER=62;

}

if ((input=='8') && (change==2)) {

TCC0\_PER=50;

} else if ((input=='8') && (change==-2)) {

TCC0\_PER=58;

}

if ((input=='I') && (change==2)) {

TCC0\_PER=46;

} else if ((input=='I') && (change==-2)) {

TCC0\_PER=54;

}

TCC0\_CNT=0x00;

if (change==2) {

for(int i=0; i< 200;i++){

for (int i=0; i< 256;i++) { //go through the 512 samples

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

DACA\_CH1DATA=Table[i]; //DAC output value according to the formula

TCC0\_CNT=0x00; //reset TCC0\_CNT to 0

}

i++;

}

}

if(change==-2) {

for(int i=0; i< 200;i++){

for (int i=0; i< 256;i++) { //go through the 512 samples

while((TCC0\_INTFLAGS & 0x01) != 0x01); //wait for interrupt flag of sample rate to be set

TCC0\_INTFLAGS=0x01; //clears the interrupt flag

float sawtooth=i\*(273/17);

DACA\_CH1DATA=(int) sawtooth; //DAC output value according to the formula

TCC0\_CNT=0x00; //reset TCC0\_CNT to 0

}

i++;

}

}

}

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; //AREF on PORTB as reference

}

void TIMER\_INIT(void) {

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

TCC0\_PER=0; //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;

}

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;

}

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

}