Lab 7  
Pengzhao Zhu   
Section: 112D

B) Prelab Questions

1. How would you complete this lab without using the Event System? What is the major drawback of doing it without Events?

**I can complete this lab using interrupts instead of using the event system. I can set up interrupts to trigger when the timer overflows and when ADC channel completes a conversion. However, the major drawback of doing it without events (and DMA) is that I would need CPU intervention and reducing the efficiency of the system (i.e interrupt code would need to run instead of an event-triggered DMA transfer that requires minimal CPU intervention).**

1. What is the range of digital and analog values you should be able to read with your ADC the way it is configured in this lab?

**Range of digital value is 0 to 255 because it is a 8 bit unsigned system. Range of analog value is 0 to 2.5 V because we used an analog voltage reference of 2.5 V.**

1. How many DMA channels are available on the XMEGA?

**There are 4 DMA channels on the XMEGA.**

1. How many different base value trigger options are available within the XMEGA’s DMA system?

**27 base value options (not including the offsets)**

C) Problems Encountered

The first problem I encountered in this lab was getting ADC to work without using free running mode. Without using free running mode, the ADC conversion system only runs once with my code. It took me a long time to figure out that I had to enable ADC and ADC channel conversion every time I want to do a conversion (Part A).

The second problem I encountered in this lab was that I had a hard time setting up the DMA system. I had to think hard about how to set up the system so it would always transfer from the same source address to the same destination address.

D) Future Application

By completing this lab, I gained more knowledge of a microprocessor’s event and DMA system. By learning how to use event and DMA, I will be able to write more efficient code that reduces CPU usage and decreases processing time. This is potentially very important when I work with embedded systems that requires large computation/data transfer in my future job and senior design.

E) Schematics

N/A

F) Pseudocode/Flowcharts

**Part A**

Call function to initialize system frequency to 32Mhz

Call function to initialize ADC

While(1) {

Enable ADC and ADC Channel 0

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);  
Read from ADCA\_CH0\_RES;  
Clear interrupt flag

}

\*Function to initialize ADC to be single ended, 8 bit unsigned mode with PORTB as reference

\*Function to set up 32MHZ clock

**Part B**

Call function to initialize system frequency to 32Mhz

Call function to initialize ADC

Call function to initialize timer

While(1) {

Enable ADC and ADC Channel 0

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);  
Read from ADCA\_CH0\_RES;  
Clear interrupt flag

}

\*Function to initialize ADC to be single ended, 8 bit unsigned mode with PORTB as reference

\*Function to set up 32MHZ clock

\*Function to initialize timer to overflow at 20 KHZ. Also set TCC0 OVF as the source for event 0

**Part C**

Call function to initialize system frequency to 32Mhz

Call function to initialize ADC

Call function to initialize timer

Call function to initialize USARTD0

While(1) {

Enable ADC and ADC Channel 0

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);  
Read from ADCA\_CH0\_RES;  
Clear interrupt flag

}

\*Function to initialize ADC to be single ended, 8 bit unsigned mode with PORTB as reference

\*Function to set up 32MHZ clock

\*Function to initialize timer to overflow at 20 KHZ. Also set TCC0 OVF as the source for event 0

\*Function to initialize USARTD0 for a baud rate of 115200

**Part D**

Call function to initialize system frequency to 32Mhz

Call function to initialize ADC

Call function to initialize timer

Call function to initialize USARTD0

Call function to initialize DMA

While(1) {

Enable ADC and ADC Channel 0

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);  
Read from ADCA\_CH0\_RES;  
Clear interrupt flag

}

\*Function to initialize ADC to be single ended, 8 bit unsigned mode with PORTB as reference.

Set up ADCA complete as the source for Event 1

\*Function to set up 32MHZ clock

\*Function to initialize timer to overflow at 20 KHZ. Also set TCC0 OVF as the source for event 0.

Also set up ADCA complete as the source for event 1

\*Function to initialize USARTD0 for a baud rate of 115200

\*Function to initialize DMA channel 0 for infinite repeat mode. Trigger source of DMA to be event 1.

Source address to be ADCA\_CH0\_RES

Destination address to be USARTD0\_DATA

Same source and destination address after every transaction

**Part E (Same Code and Pseudocode cause Part E is connecting the data visualizer)**

Call function to initialize system frequency to 32Mhz

Call function to initialize ADC

Call function to initialize timer

Call function to initialize USARTD0

Call function to initialize DMA

While(1) {

Enable ADC and ADC Channel 0

while((ADCA\_CH0\_INTFLAGS & 0x01)!= 0x01);  
Read from ADCA\_CH0\_RES;  
Clear interrupt flag

}

\*Function to initialize ADC to be single ended, 8 bit unsigned mode with PORTB as reference.

Set up ADCA complete as the source for Event 1

\*Function to set up 32MHZ clock

\*Function to initialize timer to overflow at 20 KHZ. Also set TCC0 OVF as the source for event 0.

Also set up ADCA complete as the source for event 1

\*Function to initialize USARTD0 for a baud rate of 115200

\*Function to initialize DMA channel 0 for infinite repeat mode. Trigger source of DMA to be event 1.

Source address to be ADCA\_CH0\_RES

Destination address to be USARTD0\_DATA

Same source and destination address after every transaction

G) Program Code

**Part A**

/\* Lab 7 Part A

Name: Pengzhao Zhu

Section#: 112D

TA Name: Chris Crary

Description: This program initializes the ADC system to be 8-bit unsigned, not free run mode, AREFB (2.5 V) as reference,

and a ADC prescaler that will generate at most 2 mega samples per second

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

*uint16\_t* adc;

int main(void)

{

CLK\_32MHZ(); //call 32MHZ clock

ADC(); //initialize ADC system

//8 bit unsigned adc with 2.5 V as reference. v=(1/102)adc

while(1) {

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

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

adc=ADCA\_CH0\_RES;

ADCA\_CH0\_INTFLAGS=0x01;

}

return 0;

}

void ADC(void) {

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\_RESOLUTION\_8BIT\_gc ; //unsigned mode, 8 bit resolution, no free run

PORTA\_DIRCLR= PIN0\_bm; //PA0 as input

ADCA\_CH0\_CTRL=ADC\_CH\_INPUTMODE\_SINGLEENDED\_gc; //single ended mode

ADCA\_CH0\_MUXCTRL=ADC\_CH\_MUXPOS\_PIN0\_gc; //mux control

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**

/\* Lab 7 Part B

Name: Pengzhao Zhu

Section#: 112D

TA Name: Chris Crary

Description: This program add to the Part A of lab 7.

This program configures the timer to overflow at a rate of 2 KHZ. It also configures timer overflow

as the source for Event Channel 0

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

void TIMER\_INIT(void);

*uint16\_t* adc;

double event\_timer = ((32000000\*(1/20000))/64);

int main(void)

{

CLK\_32MHZ(); //call 32MHZ clock

ADC(); //initialize ADC system

TIMER\_INIT(); //initialize timer system

//8 bit unsigned adc with 2.5 V as reference. v=(1/102)adc

while(1) {

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

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

adc=ADCA\_CH0\_RES;

ADCA\_CH0\_INTFLAGS=0x01;

}

return 0;

}

void ADC(void) {

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\_RESOLUTION\_8BIT\_gc ; //unsigned mode, 8 bit resolution, no free run

PORTA\_DIRCLR= PIN0\_bm; //PA0 as input

ADCA\_CH0\_CTRL=ADC\_CH\_INPUTMODE\_SINGLEENDED\_gc; //single ended mode

ADCA\_CH0\_MUXCTRL=ADC\_CH\_MUXPOS\_PIN0\_gc; //mux control

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

ADCA\_EVCTRL=ADC\_SWEEP\_0\_gc | ADC\_EVSEL\_0123\_gc | ADC\_EVACT\_CH0\_gc; //only sweep channel 0, 0123 event as selected inputs,

// then furtuhur reduced down to use EVENT0 to

// trigger ADC CHANNEL0

}

void TIMER\_INIT(void) {

TCC0\_CNT=0x00; //set CNT to zero

TCC0\_PER=25; //timer per value to output 1760 Hz sine wave

TCC0\_CTRLA=TC\_CLKSEL\_DIV64\_gc; //

EVSYS\_CH0MUX=EVSYS\_CHMUX\_TCC0\_OVF\_gc; //set TCC0 OVF as the source for CH0 event

}

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**

/\* Lab 7 Part C

Name: Pengzhao Zhu

Section#: 112D

TA Name: Chris Crary

Description: This program add to the Part B of lab 7.

This program configures USARTD0 to have a baud rate of 115200 with one stop bit, one start bit,

and no parity bit.

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

void TIMER\_INIT(void);

void USARTD0\_init(void);

*uint16\_t* adc;

double BSELHIGH = (((4)\*((32000000/(16\*115200))-1))>>8); //bscale of -2

double BSEL= ((4)\*((32000000/(16\*115200))-1)); //bscale of -2

//double event\_timer = ((32000000\*(1/20000))/64); //PER value to trigger event0, which then trigger ADC channel 0 conversion

int main(void)

{

CLK\_32MHZ(); //call 32MHZ clock

ADC(); //initialize ADC system

TIMER\_INIT(); //initialize timer system

//8 bit unsigned adc with 2.5 V as reference. v=(1/102)adc

USARTD0\_init();

while(1) {

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

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

adc=ADCA\_CH0\_RES;

ADCA\_CH0\_INTFLAGS=0x01;

}

return 0;

}

void ADC(void) {

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\_RESOLUTION\_8BIT\_gc ; //unsigned mode, 8 bit resolution, no free run

PORTA\_DIRCLR= PIN0\_bm; //PA0 as input

ADCA\_CH0\_CTRL=ADC\_CH\_INPUTMODE\_SINGLEENDED\_gc; //single ended mode

ADCA\_CH0\_MUXCTRL=ADC\_CH\_MUXPOS\_PIN0\_gc; //mux control

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

ADCA\_EVCTRL=ADC\_SWEEP\_0\_gc | ADC\_EVSEL\_0123\_gc | ADC\_EVACT\_CH0\_gc; //only sweep channel 0, 0123 event as selected inputs,

// then furtuhur reduced down to use EVENT0 to

// trigger ADC CHANNEL0

}

void TIMER\_INIT(void) {

TCC0\_CNT=0x00; //set CNT to zero

TCC0\_PER=25; //timer per value to output 1760 Hz sine wave

TCC0\_CTRLA=TC\_CLKSEL\_DIV64\_gc; //

EVSYS\_CH0MUX=EVSYS\_CHMUX\_TCC0\_OVF\_gc; //set TCC0 OVF as the source for CH0 event

}

void USARTD0\_init(void)

{

PORTD\_DIRSET=PIN3\_bm; //set transmitter as output

PORTD\_DIRCLR=PIN2\_bm; //set receiver as input

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

USARTD0\_CTRLC= USART\_CHSIZE\_8BIT\_gc | USART\_CMODE\_ASYNCHRONOUS\_gc | USART\_PMODE\_DISABLED\_gc; //USART asynchronous, 8 data bit, no 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= PIN3\_bm; //set transit pin idle

}

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**

/\* Lab 7 Part D

Name: Pengzhao Zhu

Section#: 112D

TA Name: Chris Crary

Description: This program add to the Part C of lab 7.

This program configures ADCA as the source for DMA. It also configures the DMA to transfer recorded ADC data

to the USARTD0 system everytime ADCA conversion is complete.

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

void TIMER\_INIT(void);

void USARTD0\_init(void);

void DMA\_INIT(void);

*uint16\_t* adc;

double BSELHIGH = (((4)\*((32000000/(16\*115200))-1))>>8); //bscale of -2

double BSEL= ((4)\*((32000000/(16\*115200))-1)); //bscale of -2

//double event\_timer = ((32000000\*(1/20000))/64); //PER value to trigger event0, which then trigger ADC channel 0 conversion

int main(void)

{

CLK\_32MHZ(); //call 32MHZ clock

ADC(); //initialize ADC system

TIMER\_INIT(); //initialize timer system

//8 bit unsigned adc with 2.5 V as reference. v=(1/102)adc

USARTD0\_init();

DMA\_INIT();

while(1) {

/\*

while((TCC0\_INTFLAGS & 0x01)!= 0x01);

TCC0\_CNT=0x00;

TCC0\_INTFLAGS=0x01;

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

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

ADCA\_CH0\_INTFLAGS=0x01;

TCC0\_CTRLA=TC\_CLKSEL\_DIV64\_gc;

\*/

}

return 0;

}

void DMA\_INIT(void) {

DMA\_CTRL=DMA\_ENABLE\_bm | DMA\_DBUFMODE\_DISABLED\_gc; //enable DMA and disable duffer buffer mode

DMA\_CH0\_REPCNT=0x00; //repeat count of 0, which is unlimited repeat

DMA\_CH0\_ADDRCTRL=0b10001000; //source address and destination reloaded with initial value at end of each burst

//source and destination does not increment

DMA\_CH0\_TRIGSRC= DMA\_CH\_TRIGSRC\_ADCA\_CH0\_gc; //trigger source for DMA as event channel 1

DMA\_CH0\_SRCADDR0= (*uint8\_t*)&ADCA\_CH0\_RES; //source address is ADCA\_CH0RES

DMA\_CH0\_SRCADDR1= ((*uint16\_t*)&ADCA\_CH0\_RES) >> 8;

DMA\_CH0\_SRCADDR2= ((*uint32\_t*)&ADCA\_CH0\_RES) >> 16;

DMA\_CH0\_DESTADDR0=(*uint8\_t*)&USARTD0\_DATA; //destination address is USARTD0\_DATA

DMA\_CH0\_DESTADDR1=((*uint16\_t*)&USARTD0\_DATA) >> 8;

DMA\_CH0\_DESTADDR2=((*uint32\_t*)&USARTD0\_DATA) >> 16;

DMA\_CH0\_CTRLA=DMA\_CH\_ENABLE\_bm | DMA\_CH\_REPEAT\_bm | DMA\_CH\_SINGLE\_bm | DMA\_CH\_BURSTLEN\_1BYTE\_gc;

//repeat mode, single shot data transfer

//burst mode defaults to 00=1 byte

}

void ADC(void) {

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\_RESOLUTION\_8BIT\_gc ; //unsigned mode, 8 bit resolution, no free run

PORTA\_DIRCLR= PIN0\_bm; //PA0 as input

ADCA\_CH0\_CTRL=ADC\_CH\_INPUTMODE\_SINGLEENDED\_gc; //single ended mode

ADCA\_CH0\_MUXCTRL=ADC\_CH\_MUXPOS\_PIN0\_gc; //mux control

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

ADCA\_EVCTRL=ADC\_SWEEP\_0\_gc | ADC\_EVSEL\_0123\_gc | ADC\_EVACT\_CH0\_gc; //only sweep channel 0, 0123 event as selected inputs,

// then furtuhur reduced down to use EVENT0 to

// trigger ADC CHANNEL0

}

void TIMER\_INIT(void) {

TCC0\_CNT=0x00; //set CNT to zero

TCC0\_PER=25; //timer per value to output 1760 Hz sine wave

TCC0\_CTRLA=TC\_CLKSEL\_DIV64\_gc; //

EVSYS\_CH0MUX=EVSYS\_CHMUX\_TCC0\_OVF\_gc; //set TCC0 OVF as the source for CH0 event

EVSYS\_CH1MUX=EVSYS\_CHMUX\_ADCA\_CH0\_gc; //set ADCA CH0 conversion complete as source for CH1 event

}

void USARTD0\_init(void)

{

PORTD\_DIRSET=PIN3\_bm; //set transmitter as output

PORTD\_DIRCLR=PIN2\_bm; //set receiver as input

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

USARTD0\_CTRLC= USART\_CHSIZE\_8BIT\_gc | USART\_CMODE\_ASYNCHRONOUS\_gc | USART\_PMODE\_DISABLED\_gc; //USART asynchronous, 8 data bit, no 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= PIN3\_bm; //set transit pin idle

}

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**

/\* Lab 7 Part E

Name: Pengzhao Zhu

Section#: 112D

TA Name: Chris Crary

Description: This program add to the Part C of lab 7.

This program configures ADCA as the source for DMA. It also configures the DMA to transfer recorded ADC data

to the USARTD0 system everytime ADCA conversion is complete. Connect to the data visualizer in Atmel

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

void CLK\_32MHZ(void);

void ADC(void);

void TIMER\_INIT(void);

void USARTD0\_init(void);

void DMA\_INIT(void);

*uint16\_t* adc;

double BSELHIGH = (((4)\*((32000000/(16\*115200))-1))>>8); //bscale of -2

double BSEL= ((4)\*((32000000/(16\*115200))-1)); //bscale of -2

//double event\_timer = ((32000000\*(1/20000))/64); //PER value to trigger event0, which then trigger ADC channel 0 conversion

int main(void)

{

CLK\_32MHZ(); //call 32MHZ clock

ADC(); //initialize ADC system

TIMER\_INIT(); //initialize timer system

//8 bit unsigned adc with 2.5 V as reference. v=(1/102)adc

USARTD0\_init();

DMA\_INIT();

while(1) {

}

return 0;

}

void DMA\_INIT(void) {

DMA\_CTRL=DMA\_ENABLE\_bm | DMA\_DBUFMODE\_DISABLED\_gc; //enable DMA and disable duffer buffer mode

DMA\_CH0\_REPCNT=0x00; //repeat count of 0, which is unlimited repeat

DMA\_CH0\_ADDRCTRL=0b10001000; //source address and destination reloaded with initial value at end of each burst

//source and destination does not increment

DMA\_CH0\_TRIGSRC= DMA\_CH\_TRIGSRC\_ADCA\_CH0\_gc; //trigger source for DMA as event channel 1

DMA\_CH0\_SRCADDR0= (*uint8\_t*)&ADCA\_CH0\_RES; //source address is ADCA\_CH0RES

DMA\_CH0\_SRCADDR1= ((*uint16\_t*)&ADCA\_CH0\_RES) >> 8;

DMA\_CH0\_SRCADDR2= ((*uint32\_t*)&ADCA\_CH0\_RES) >> 16;

DMA\_CH0\_DESTADDR0=(*uint8\_t*)&USARTD0\_DATA; //destination address is USARTD0\_DATA

DMA\_CH0\_DESTADDR1=((*uint16\_t*)&USARTD0\_DATA) >> 8;

DMA\_CH0\_DESTADDR2=((*uint32\_t*)&USARTD0\_DATA) >> 16;

DMA\_CH0\_CTRLA=DMA\_CH\_ENABLE\_bm | DMA\_CH\_REPEAT\_bm | DMA\_CH\_SINGLE\_bm | DMA\_CH\_BURSTLEN\_1BYTE\_gc;

//repeat mode, single shot data transfer

//burst mode defaults to 00=1 byte

}

void ADC(void) {

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\_RESOLUTION\_8BIT\_gc ; //unsigned mode, 8 bit resolution, no free run

PORTA\_DIRCLR= PIN0\_bm; //PA0 as input

ADCA\_CH0\_CTRL=ADC\_CH\_INPUTMODE\_SINGLEENDED\_gc; //single ended mode

ADCA\_CH0\_MUXCTRL=ADC\_CH\_MUXPOS\_PIN0\_gc; //mux control

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

ADCA\_EVCTRL=ADC\_SWEEP\_0\_gc | ADC\_EVSEL\_0123\_gc | ADC\_EVACT\_CH0\_gc; //only sweep channel 0, 0123 event as selected inputs,

// then furtuhur reduced down to use EVENT0 to

// trigger ADC CHANNEL0

}

void TIMER\_INIT(void) {

TCC0\_CNT=0x00; //set CNT to zero

TCC0\_PER=25; //timer per value to output 1760 Hz sine wave

TCC0\_CTRLA=TC\_CLKSEL\_DIV64\_gc; //

EVSYS\_CH0MUX=EVSYS\_CHMUX\_TCC0\_OVF\_gc; //set TCC0 OVF as the source for CH0 event

EVSYS\_CH1MUX=EVSYS\_CHMUX\_ADCA\_CH0\_gc; //set ADCA CH0 conversion complete as source for CH1 event

}

void USARTD0\_init(void)

{

PORTD\_DIRSET=PIN3\_bm; //set transmitter as output

PORTD\_DIRCLR=PIN2\_bm; //set receiver as input

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

USARTD0\_CTRLC= USART\_CHSIZE\_8BIT\_gc | USART\_CMODE\_ASYNCHRONOUS\_gc | USART\_PMODE\_DISABLED\_gc; //USART asynchronous, 8 data bit, no 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= PIN3\_bm; //set transit pin idle

}

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

}

# H) Appendix

N/A