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).

2. 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.

3. How many DMA channels are available on the XMEGA?
   There are 4 DMA channels on the XMEGA.

4. 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