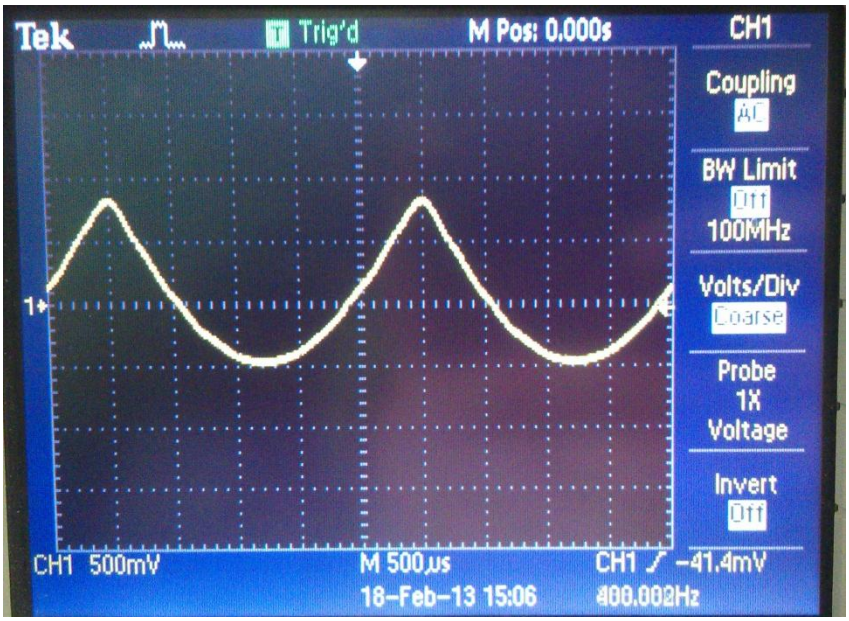
RTDSP Lab 3

Questions.

1. The full wave rectified waveform is centered around 0V and not above because the audio chip has high pass filters around it to remove DC gain as this is undesirable in an audio system.
2. The system only produces the correct output full-wave rectified version when the input signal is below the nyquist frequency. The resultant waveform with an input of 3.8kHz is as below:



This is because the sampling frequency of the output audio codec is too low to produce an output of 2*input as they are almost the same frequency. Since the output frequency is above that of the nyquist frequency, folding occurs about the nyquist frequency giving the result above.

Explanation:

Lab 3:

This lab takes an input signal from the dsp line in, full wave rectifies it and outputs the resulting signal through the line out port. The board hardware is initialised with the interrupt routine being mapped. Whenever an audio sample has been generated from the line in by the ADC, an interrupt is triggered which calls the ISR_AIC function. This reads a mono input from the ADC using the provided helper function, full-wave rectifies it, multiplies it by an optimum gain value (~2^16-1), and the sample is then written back to the audio codec, again using a provided helper function. The code also features a Boolean toggle as to whether the input signal should be full-wave rectified or passed straight through to the output.

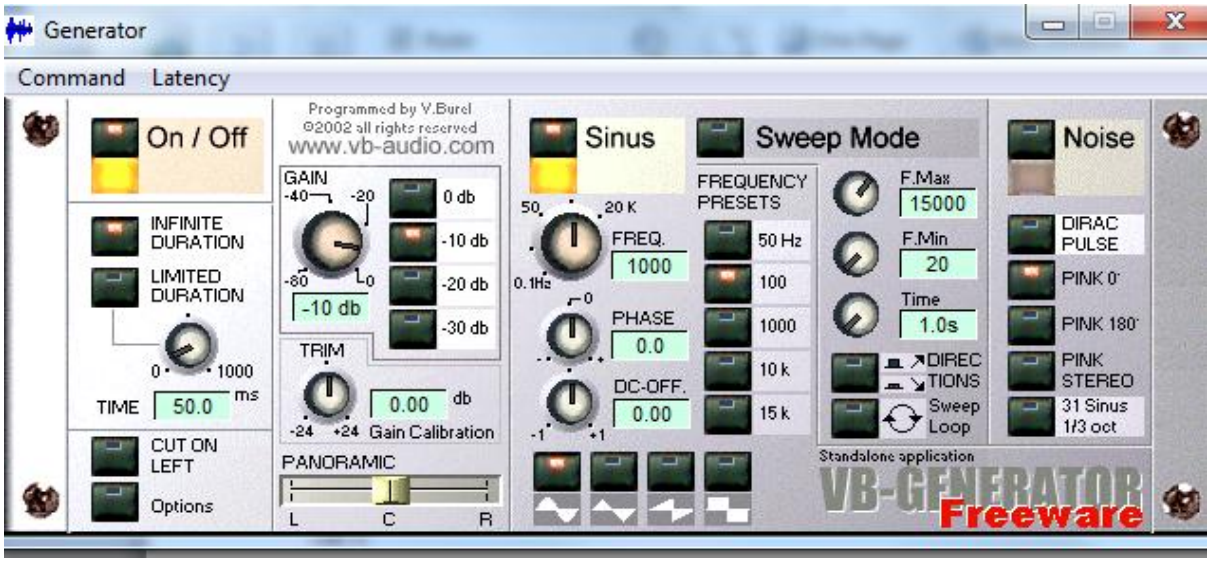
Lab3b:

This lab works in the same way as the previous example however it uses the sine table from lab 2 to generate an "input" signal sine wave instead of reading the sample in from the line in and the interrupt is triggered when the audio codec is ready to receive a sample, not when it is ready to provide one. As above, there is a Boolean toggle which only full-wave rectifies the sample generated if set true.

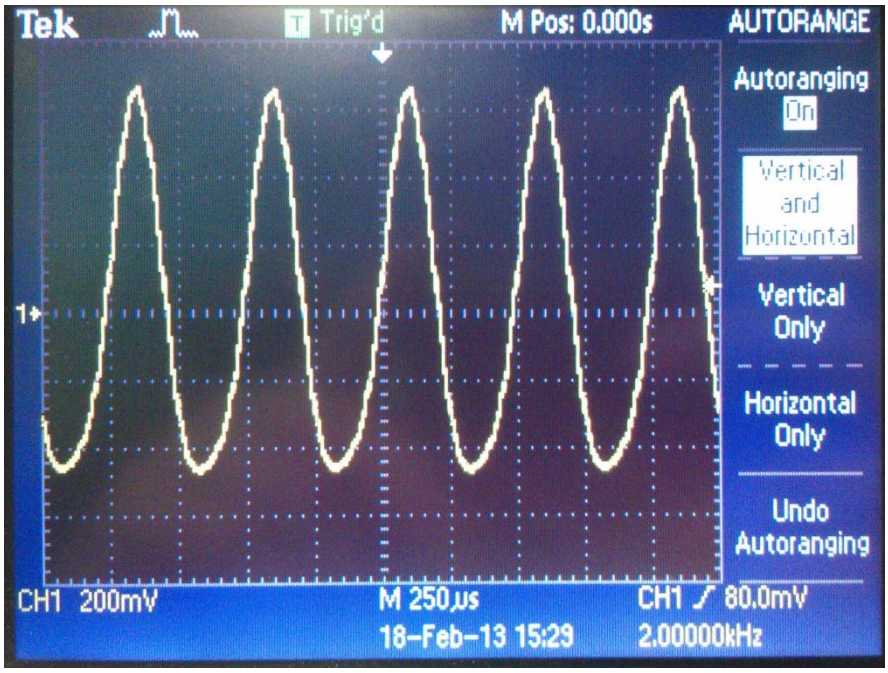
Code in operation:

Lab 3:

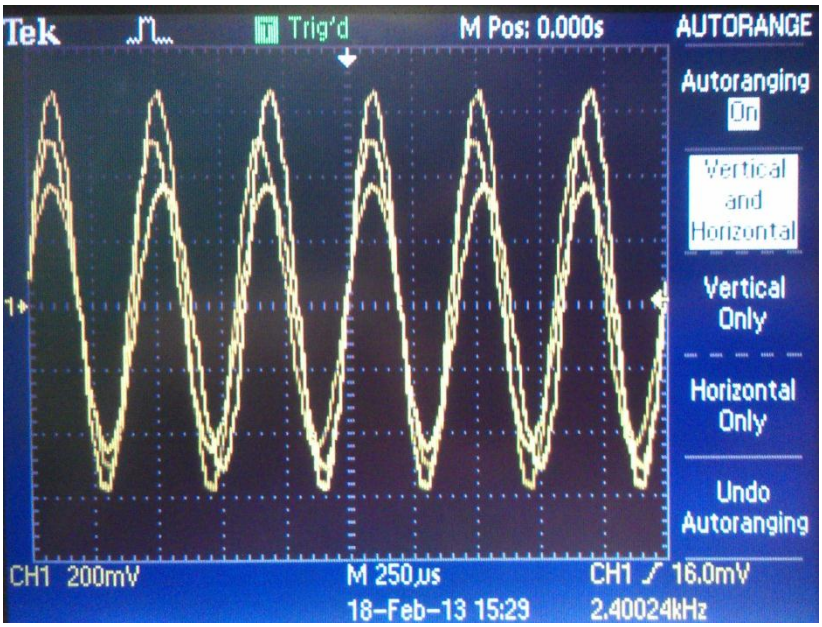
The limit for Lab 3 was an input of 1kHz.



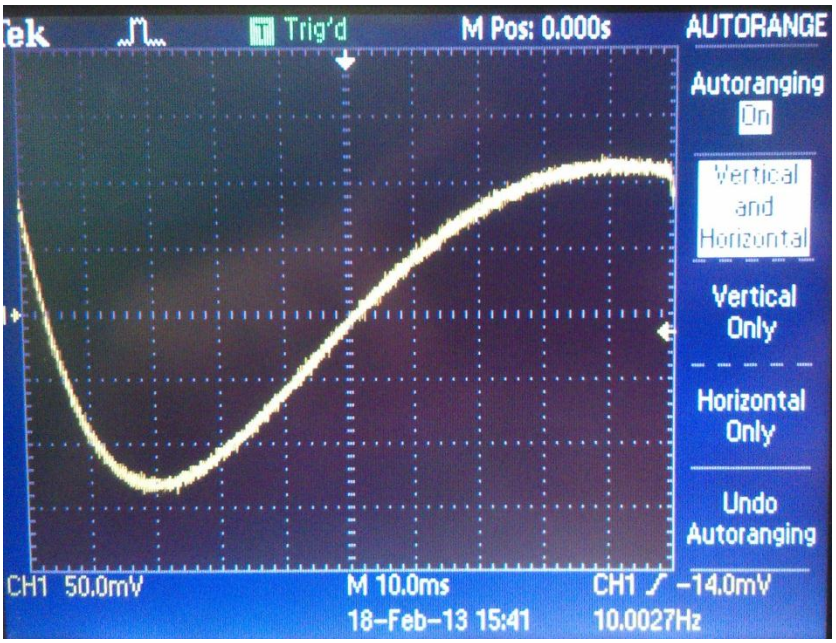
This gives the output as below:



An input signal of any higher frequency gives an output like below:

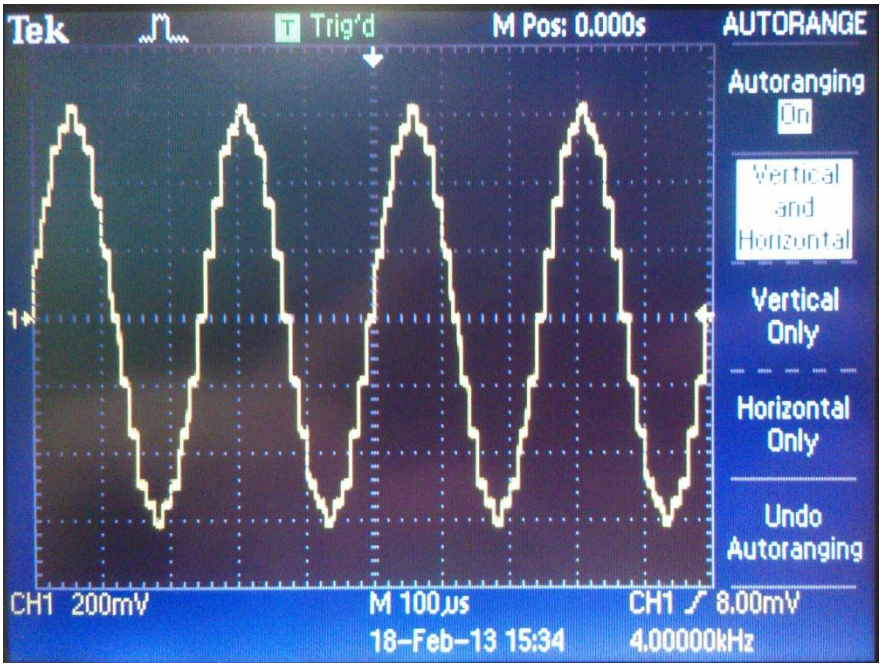


The code begins working correctly at a frequency of around 5Hz as shown below although gives a misshapen output waveform as below:

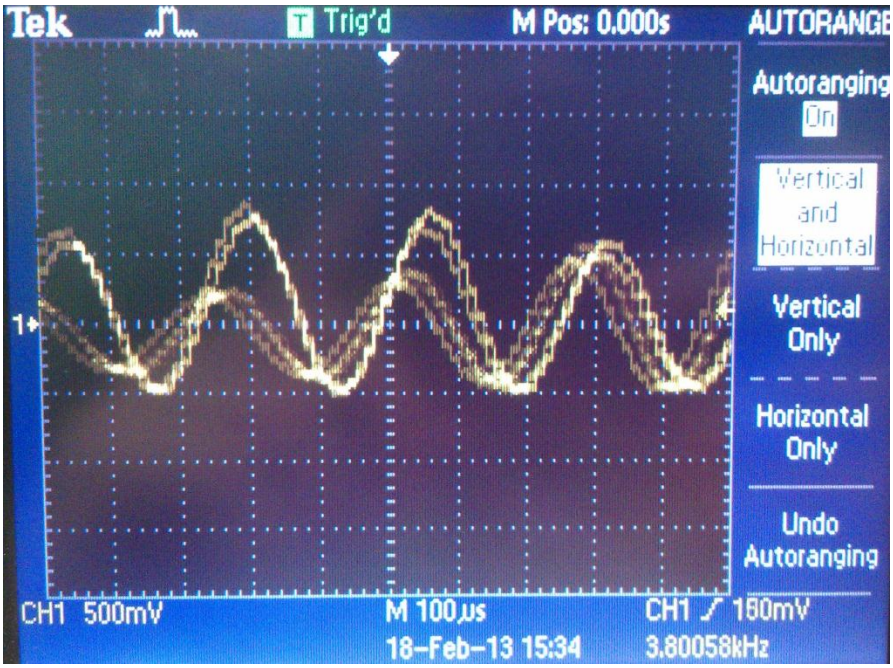


Lab 3b:

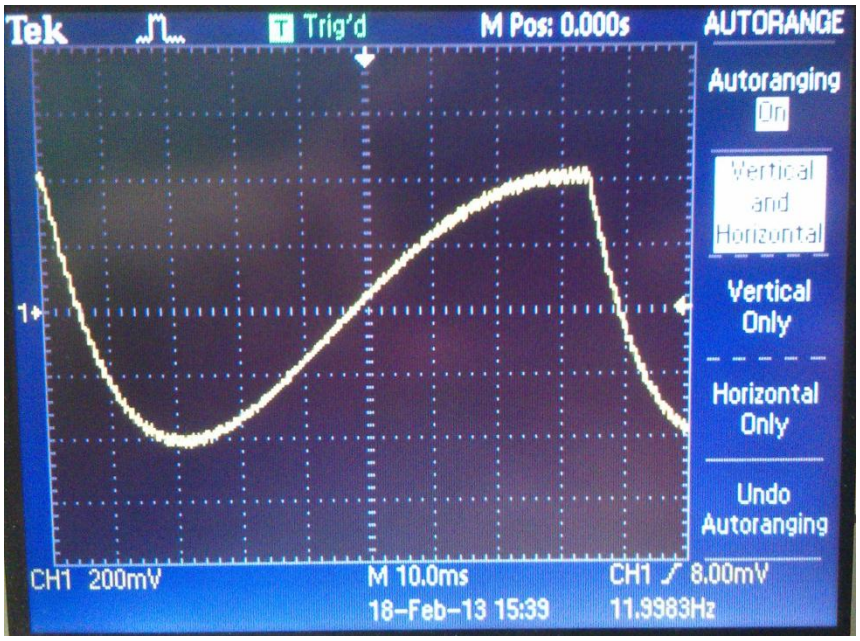
Lab 3b works up to an input frequency of 2kHz as shown below:



However, any higher input frequency gives an output as below:



The start frequency is around 6Hz which gives the correct frequency but a somewhat misshaped output waveform as below:



Commented Code:

Lab 3:

```
#include <stdlib.h>
// Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"

/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include "dsk6713.h"
#include "dsk6713_aic23.h"

// math library (trig functions)
#include <math.h>

// Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>

#define GAIN 10

int rectify = 0;
float sample =0;
float wave =0;

/******************************  Global declarations ********************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
                                /************************************************************************/
                                /*   REGISTER                FUNCTION                          SETTINGS     */
                                /************************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB             */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB             */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB             */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB             */\
    0x0011,  /* 4 ANAPATH    Analog audio path control     DAC on, Mic boost 20dB*/\
    0x0000,  /* 5 DIGPATH    Digital audio path control    All Filters off    */\
    0x0000,  /* 6 DPOWERDOWN Power down control          All Hardware on    */\
    0x0043,  /* 7 DIGIF      Digital audio interface format 16 bit           */\
    0x008d,  /* 8 SAMPLERATE Sample rate control         8 KHZ              */\
    0x0001   /* 9 DIGACT     Digital interface activation  On                */\
```

```
                                /************************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;

/*********************************  Function prototypes *********************************/
void init_hardware(void);
void init_HWI(void);
void ISR_AIC(void);
/*********************************  Main routine ***********************************/
void main(){


        // initialize board and the audio port
 init_hardware();

 /* initialize hardware interrupts */
 init_HWI();

 /* loop indefinitely, waiting for interrupts */
 while(1)
 {};

}

/*********************************  init_hardware() ********************************/
void init_hardware()
{
   // Initialize the board support library, must be called first
   DSK6713_init();

   // Start the AIC23 codec using the settings defined above in config
   H_Codec = DSK6713_AIC23_openCodec(0, &Config);

        /* Function below sets the number of bits in word used by MSBSP (serial port) for
        receives from AIC23 (audio port). We are using a 32 bit packet containing two
        16 bit numbers hence 32BIT is set for  receive */
        MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);

        /* Configures interrupt to activate on each consecutive available 32 bits
        from Audio port hence an interrupt is generated for each L & R sample pair */
        MCBSP_FSETS(SPCR1, RINTM, FRM);

        /* These commands do the same thing as above but applied to data transfers to
        the audio port */
        MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
        MCBSP_FSETS(SPCR1, XINTM, FRM);

}

/*********************************  init_HWI() ************************************/
void init_HWI(void)
{
        IRQ_globalDisable();                    // Globally disables interrupts
        IRQ_nmiEnable();                                // Enables the NMI interrupt (used by the debugger)
        IRQ_map(IRQ_EVT_RINT1,4);               // Maps an event to a physical interrupt
        IRQ_enable(IRQ_EVT_RINT1);              // Enables the event
        IRQ_globalEnable();                             // Globally enables interrupts

}

/*******************  WRITE YOUR INTERRUPT SERVICE ROUTINE HERE**********************/

void ISR_AIC(void)
{
        wave = mono_read_16Bit();
        if (rectify)
                sample = abs(wave) * GAIN;
        else
```

```
                    sample = wave * GAIN;
            mono_write_16Bit(sample);
}
```

Lab 3b:

```
// Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"

/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include "dsk6713.h"
#include "dsk6713_aic23.h"

// math library (trig functions)
#include <math.h>

// Some functions to help with configuring hardware
#include <helper_functions_ISR.h>

// PI defined here for use in your code
#define PI 3.141592653589793

// Define variable for size of look up table
#define SINE_TABLE_SIZE 256

/*****************************  Global declarations ****************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = {\
                                /************************************************************************/
                                /*   REGISTER                FUNCTION                                SETTINGS       */
                                /************************************************************************/\
   0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB              */\
   0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB              */\
   0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB              */\
   0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB              */\
   0x0011,  /* 4 ANAPATH    Analog audio path control      DAC on, Mic boost 20dB*/\
   0x0000,  /* 5 DIGPATH    Digital audio path control     All Filters off     */\
   0x0000,  /* 6 DPOWERDOWN Power down control           All Hardware on     */\
   0x0043,  /* 7 DIGIF      Digital audio interface format 16 bit              */\
   0x008d,  /* 8 SAMPLERATE Sample rate control          8 KHZ               */\
   0x0001   /* 9 DIGACT     Digital interface activation   On                  */\
                                /************************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;

/* Sampling frequency in HZ. Must only be set to 8000, 16000, 24000
32000, 44100 (CD standard), 48000 or 96000  */
int sampling_freq = 8000;

// Keep track of which sample we're on
float sample_number = 0;

// Holds the value of the current sample
float sample;
float wave;
int rectify = 0;

// Define Output Gain
#define GAIN 32000

/* Use this variable in your code to set the frequency of your sine wave
   be carefull that you do not set it above the current nyquist frequency! */
float sine_freq = 1000.0;
```

```
//Create table to store sine values
float table[SINE_TABLE_SIZE];

/*******************************  Function prototypes *******************************/
void init_hardware(void);
void init_HWI(void);
void init_sine(void);
void ISR_SINEOUT(void);
//float sinegen(void);
/*********************************  Main routine *********************************/
void main()
{

        // initialize board and the audio port
        init_hardware();

        init_HWI();

        // initialize the table of sine values
        init_sine();

  // Loop endlessley generating a sine wave
  while(1){};

}

/*******************************  init_sine() *************************************/
void init_sine()
{
        /* Function to populate the values in the sine table */
        int i;
        for(i=0; i<=SINE_TABLE_SIZE; i++){
                table[i] = sin((2 * PI * i)/SINE_TABLE_SIZE);
        };
}


/*******************************  init_hardware() *********************************/
void init_hardware()
{
  // Initialize the board support library, must be called first
  DSK6713_init();

  // Start the codec using the settings defined above in config
  H_Codec = DSK6713_AIC23_openCodec(0, &Config);

        /* Defines number of bits in word used by MSBSP for communications with AIC23
         NOTE: this must match the bit resolution set in in the AIC23 */
        MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
        MCBSP_FSETS(SPCR1, XINTM, FRM);

}

/*********************************  init_HWI() ************************************/
void init_HWI(void)
{
        IRQ_globalDisable();                    // Globally disables interrupts
        IRQ_nmiEnable();                                // Enables the NMI interrupt (used by the debugger)
        IRQ_map(IRQ_EVT_XINT1,4);               // Maps an event to a physical interrupt
        IRQ_enable(IRQ_EVT_XINT1);              // Enables the event
        IRQ_globalEnable();                             // Globally enables interrupts

}

void ISR_SINEOUT(void)
{
        // Calculate number of samples per complete sine wave
        float sample_count = sampling_freq/sine_freq;

        // Calculate the next look up table element to be returned
        sample_number = ((sample_number + (SINE_TABLE_SIZE/sample_count)));
```

```
        if (sample_number > SINE_TABLE_SIZE) sample_number -= SINE_TABLE_SIZE;

        // Return sine table element corresponding to this sample
        wave = table[(int)sample_number] * GAIN;

        if (rectify)
                {sample = abs(wave);}
        else
                {sample = wave;}

        mono_write_16Bit(sample);

}
```