Imperial College
London

Department of Electrical and Electronic Engineering Imperial College London

Real-time digital signal processing

Report on "*Interrupt I/O*" (laboratory 3)

15 pages total

Report written by:

Quentin McGaw,	CID 00746622	Username QDM12
Alexandra Rouhana,	CID 00736752	Username AR4412

Exercise 1: Interrupt service routine

After having properly configured the configuration file and the bios as described in figure 1.1, the function *ISR_AIC* has been designed to be added to the C code file *intio.c* as shown in figure 1.2 below. The full *intio.c* content is shown in appendix 1.

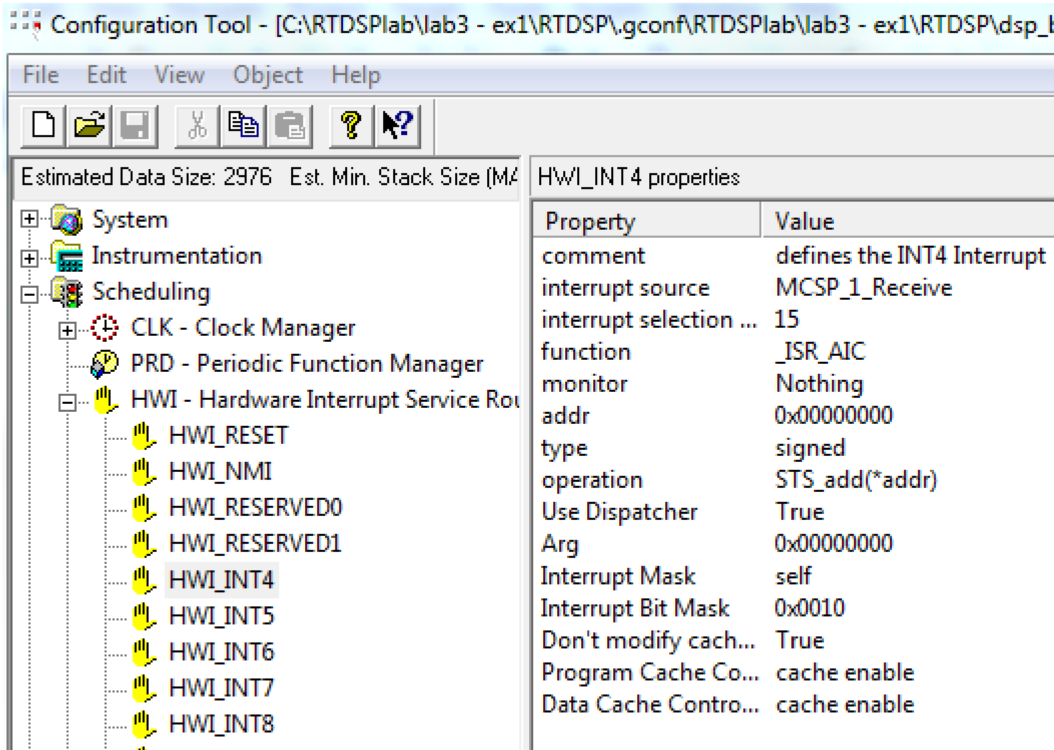


Figure 1.1 : BIOS configuration for interrupt HWI_INT4

```
/************************** INTERRUPT SERVICE ROUTINE *********************************/

void ISR_AIC(void) /*MODIFICATION_01 - Read input port, fully-rectify signal and
output*/
{
    rectified = mono_read_16Bit();
    if((fullrectify) && (rectified > 0)){
        rectified = -rectified;}
    mono_write_16Bit(rectified);
}
```

Figure 1.2 : ISR_AIC function definition

This function is also declared at the beginning of the *intio.c* code. It is called as soon as a sample is read from the input ports. It first stores the half of the sum of the Left and Right channels samples by using the *mono_read_16Bit()* function. The Boolean variable *fullrectify* is defined as a global variable in order to be able to turn on or off the full rectification of the input signal. The full rectification of the signal is done in the *IF* statement by reversing the sign of the variable *rectified*. Finally, this processed variable which represents the input sample rectified or not, is outputted with *mono_write_16Bit(rectified)*.

This exercise requires us to connect the two input ports (L and R) of the board to a signal generator. The configuration shown in figure 1.3 is used to connect and measure the signals.

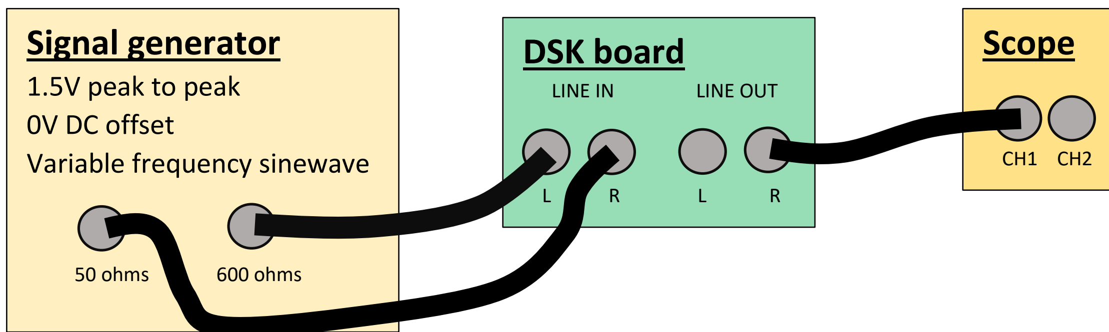


Figure 1.3 : Connection configuration for input and output ports of the DSK board

The signals obtained for an input sine wave at a frequency of 1KHz are shown in the next three figures below. They show the input signal, the output signal without rectification and the output signal with rectification.

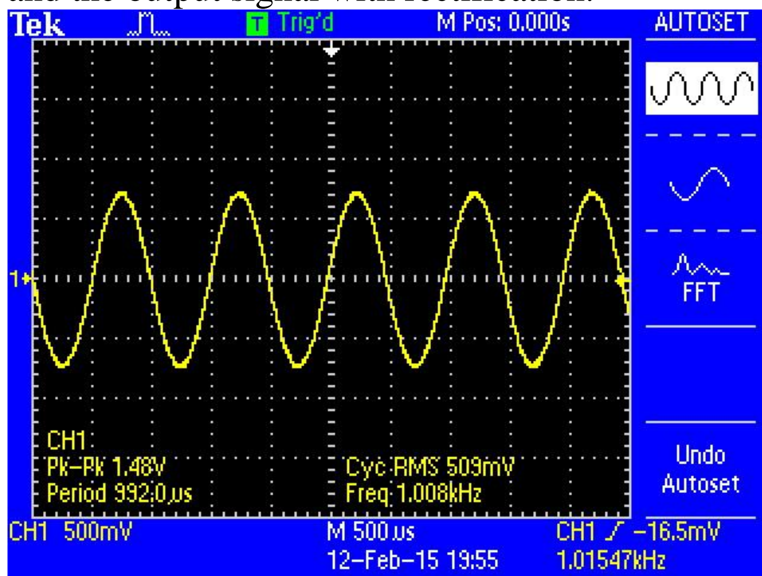


Figure 1.4 : Input signal to the DSK board (1.5V p-p, 1KHz sine wave)

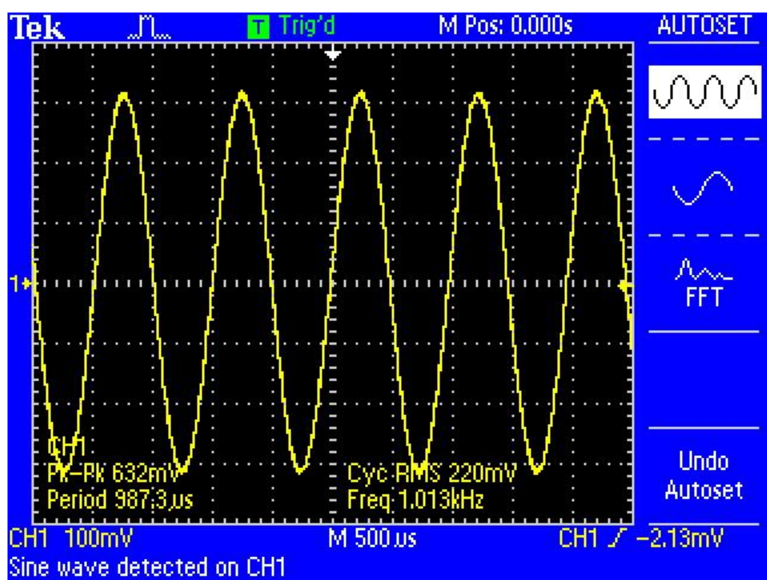


Figure 1.5 : Output signal from the DSK board without rectification

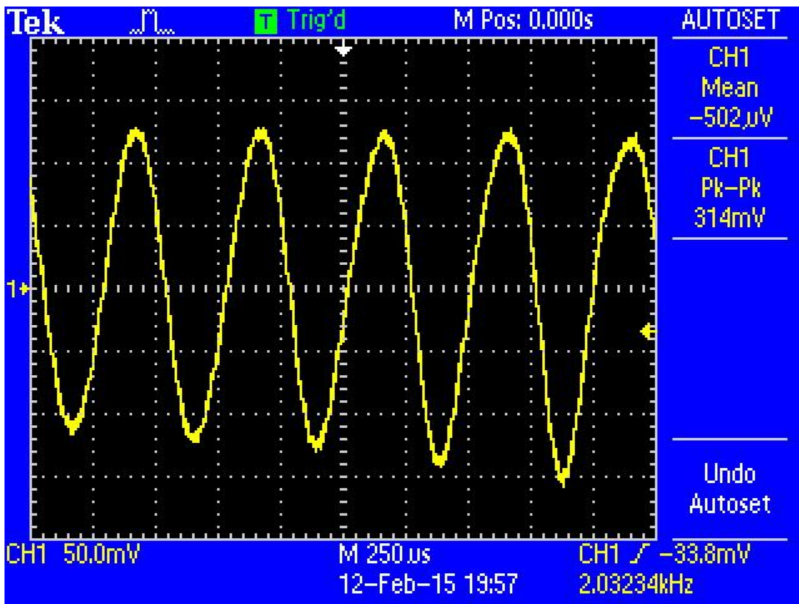


Figure 1.6 : Fully rectified output signal from the DSK board

As shown on figure 1.6, the frequency of output signal has doubled to 2.03234KHz hence the full rectification of signal works.

Why is the full rectified waveform centred around 0V and not always above 0V as you may have been expecting? (1 point)

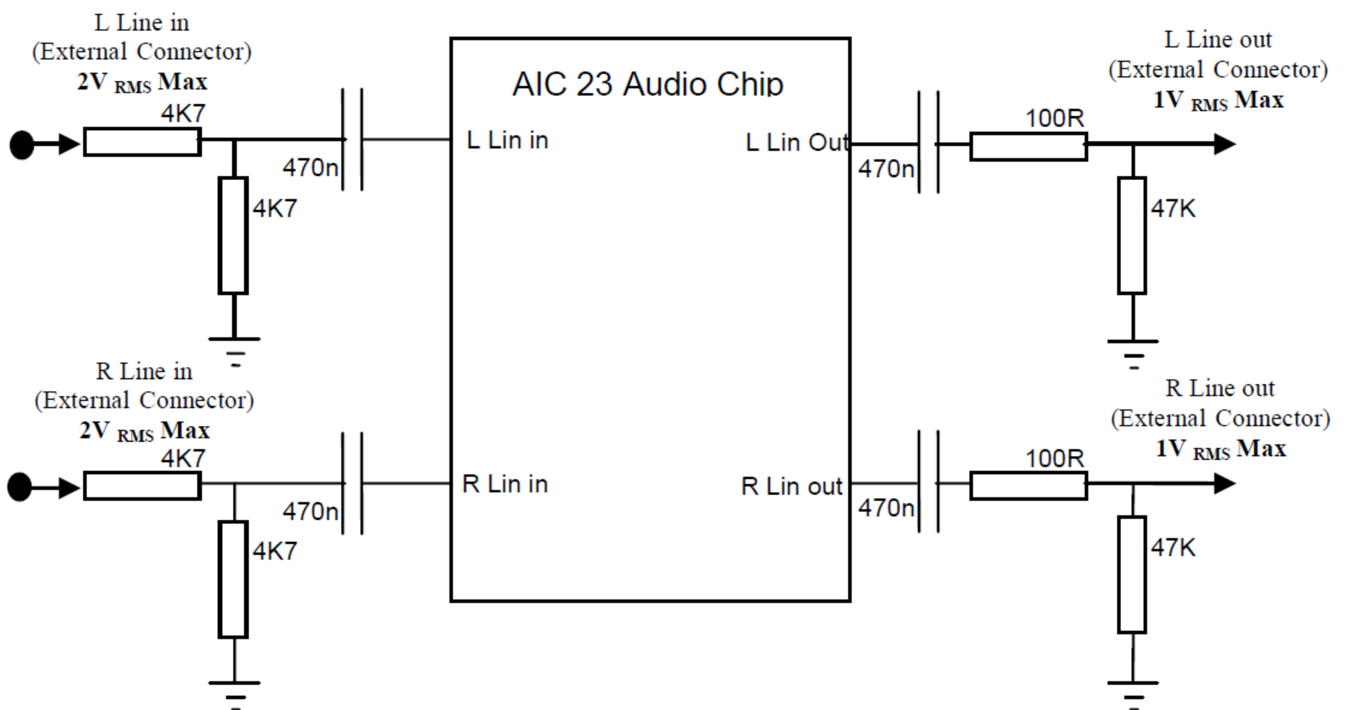


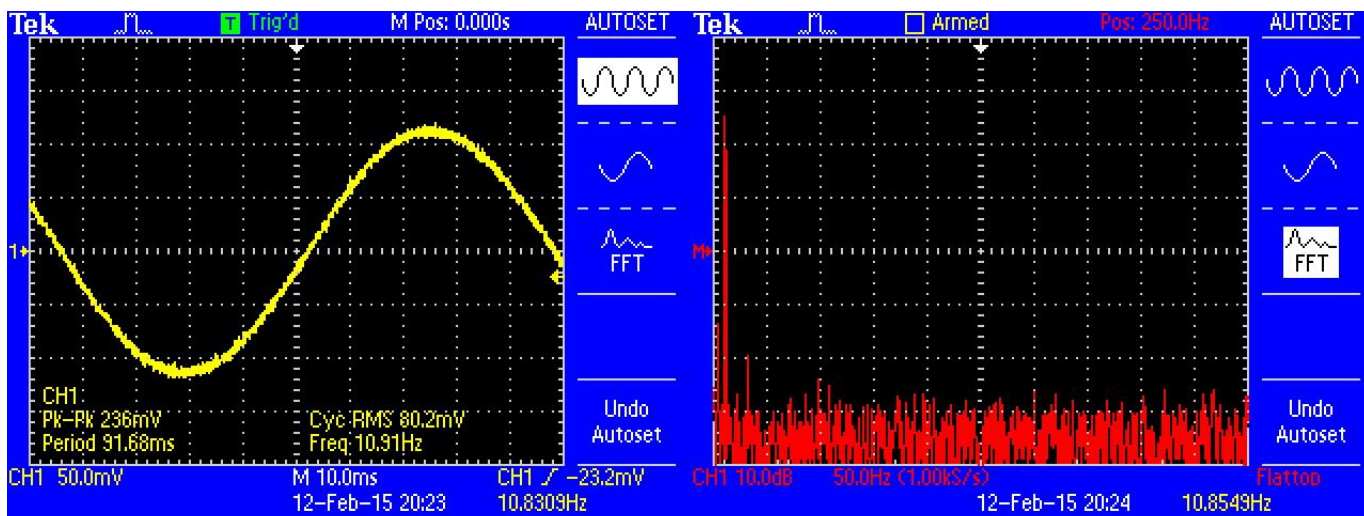
Figure 1.7 : AIC23 Audio chip external components (adapted from TMS320C6713 Technical ref (page A-14, 2003 revision A)

Left and Right circuits are the same for both the input and output sides. All of these have a capacitor placed such that the DC offset is removed from the input signal (or output from the Audio Chip). The DC offset of a sine wave is zero so the output is unchanged.

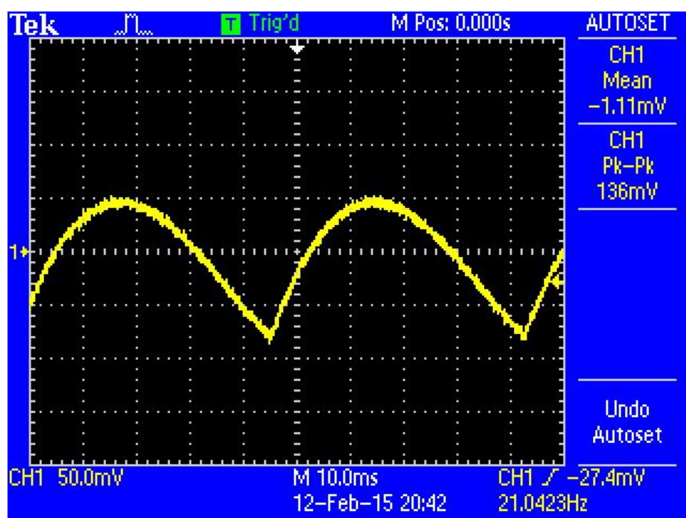
However, the DC offset of a fully rectified sine wave is the half of the amplitude of the non-rectified sine wave. This implies that the output from the DSK board for this kind of rectified signal will be lowered by its DC offset. This is why the rectified sine wave is centred at zero.

Note that the output waveform will only be a full-wave rectified version of the input if the input from the signal generator is below a certain frequency. Why is this? You may wish to explain your answer using frequency spectra diagrams.

The full rectification of the input signal works but is limited to a certain frequency range. First, the frequency of the input sine wave signal has to be greater than 10.5Hz approximately. The following signals are hence obtained.



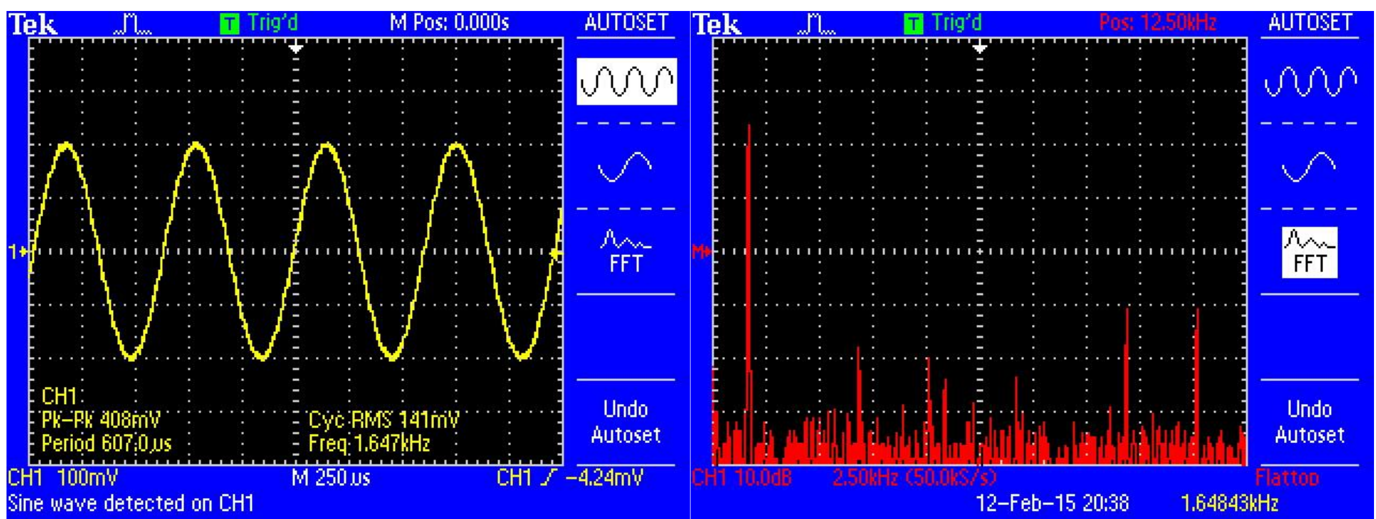
Figures 1.8 and 1.9 : Non-rectified output signal from the DSK board at a 10.83Hz frequency (Time domain on left, FFT on right)



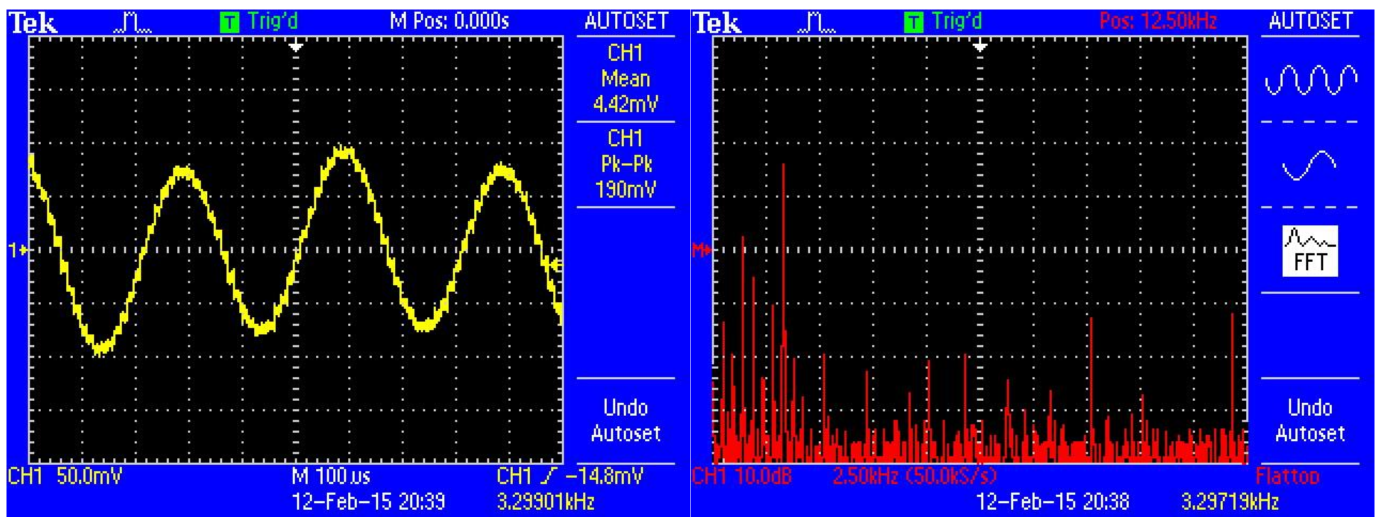
Figures 1.10 : Fully rectified output signal from the DSK board at a 10.83Hz frequency

Lowering the frequency below 10.83Hz will produce a zero DC signal.

The maximum input frequency to obtain the right frequency for the fully rectified output signal is 1.65 KHz. The signals obtained are shown in the figures below.



Figures 1.11 and 1.12 : Non-rectified output signal from the DSK board at a 1.65KHz frequency
(Time domain on left, FFT on right)

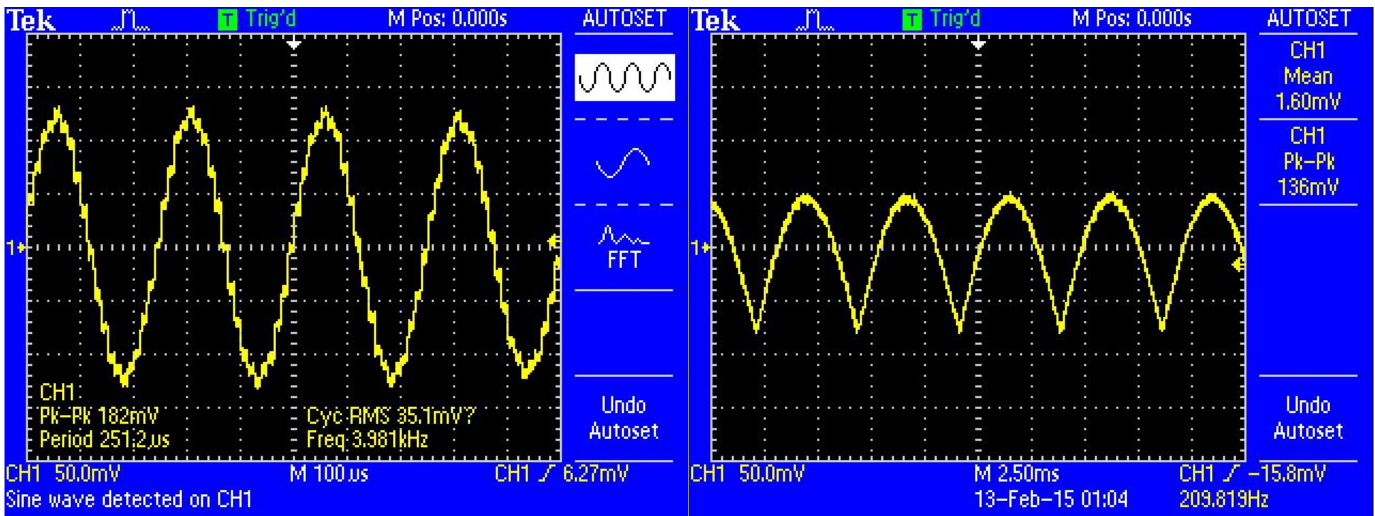


Figures 1.13 and 1.14 : Fully rectified output signal from the DSK board at a 1.65KHz frequency (input)
(Time domain on left, FFT on right)

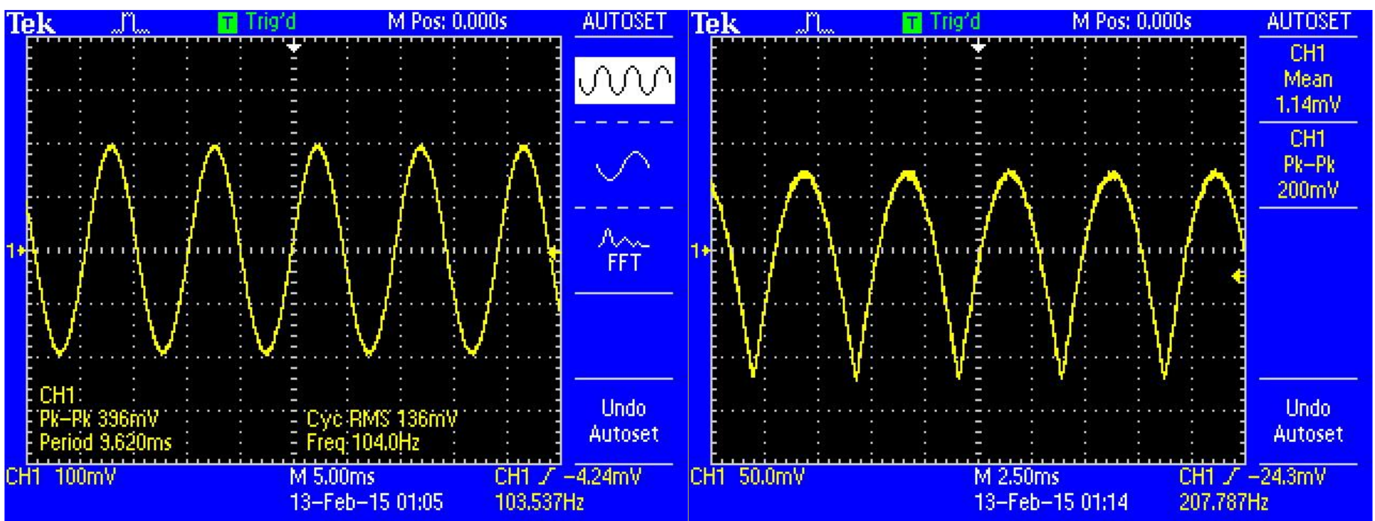
Increasing further the frequency will make the fully rectified signal's frequency be less than twice the frequency of the input non-rectified sine wave signal.

What kind of output do you see when you put in a sine wave at around 3.8 kHz? Can you explain what is going on? (10 points)

When inputting a sine wave with a frequency of 3.981 KHz, the result is a fully rectified signal corresponding to a 100 Hz sine wave. The following signals are obtained.



Figures 1.15 and 1.16 : Non-rectified (left) and rectified (right) output signals from the DSK board, at 3.981KHz



Figures 1.17 and 1.18 : Non-rectified (left) and rectified (right) output signals from the DSK board, at 100 Hz

As this is shown above, the same waveform is obtained for input sine wave frequencies of 3.9KHz and 100Hz. This is also happening for the two set of frequencies 0 Hz – 2000 Hz and 2000 Hz – 4000 Hz which are symmetric, where DC is mapped to 4000 Hz. This happens because of the symmetry in the frequency domain. To further proof this, a MatLab script has been designed to generate a full rectified sine wave and to plot its spectrum. The script is described below in figure 1.19 and the resulting plots in figure 1.20.

```
%% Time specifications:

Fs = 8000; % samples per second
dt = 1/Fs; % seconds per sample
StopTime = 1; % seconds
t = (0:dt:StopTime-dt)';
N = size(t,1);

%% Sine wave:
Fc = 3900; % hertz
x = abs(sin(2*pi*Fc*t));

%% Fourier Transform:
X = fftshift(fft(x));

%% Frequency specifications:
```

```
dF = Fs/N; % hertz
f = -Fs/2:dF:Fs/2-dF; % hertz

%% Plot the spectrum:
figure;
subplot(1,2,1);
plot(t(1:100),x(1:100));
xlabel('Time (in seconds)');
subplot(1,2,2);
plot(f,abs(X)/N);
xlabel('Frequency (in hertz)');
title('Magnitude Response');
```

Figure 1.19 : Matlab script to plot spectrum of rectified sinewave (original frequency of 3900Hz)

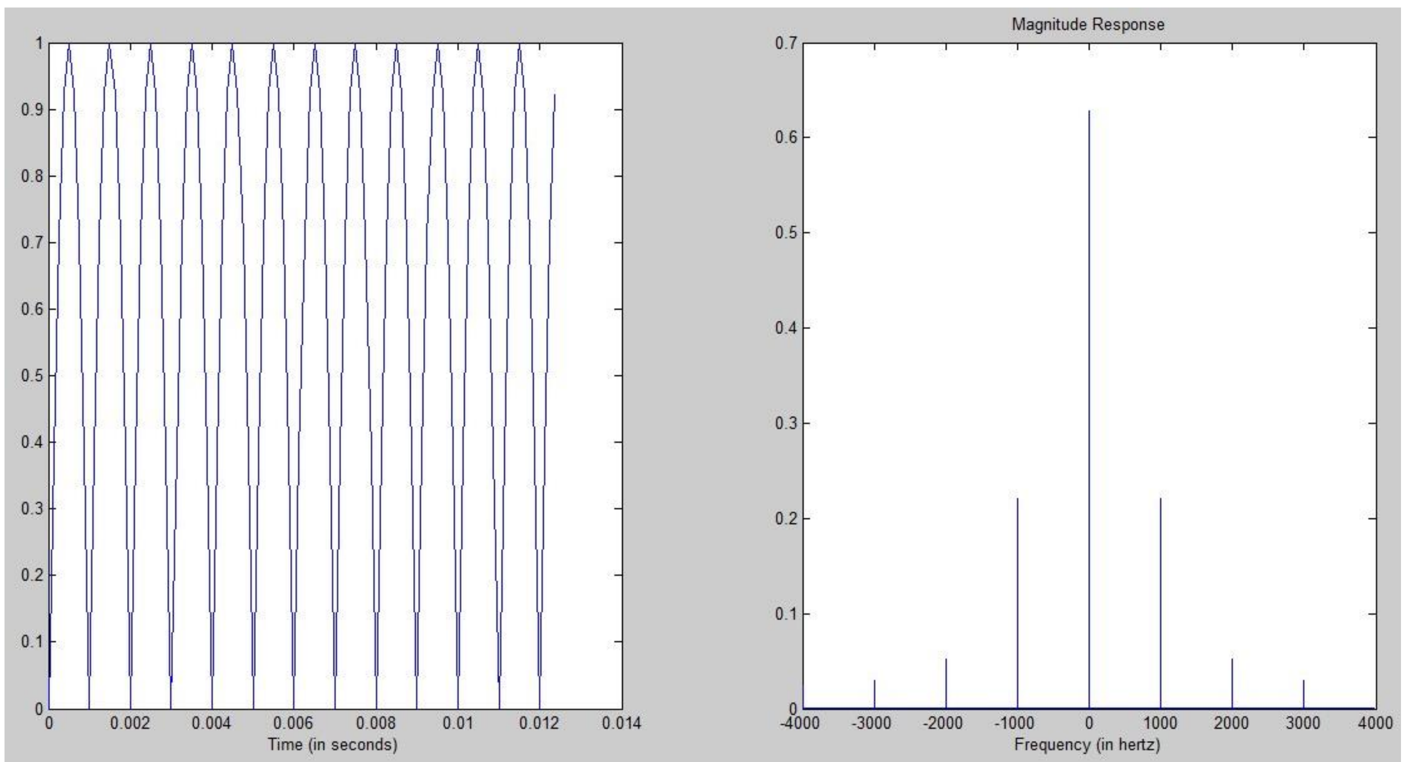


Figure 1.20 : Matlab plots, time domain (left) and spectrum (right) obtained for both 100Hz and 3900Hz frequencies

The two plots shown in figure 1.20 were both obtained for sine waves at frequencies of 100 Hz and 3900 Hz. This again is true for all frequencies in the ranges 0 to 2000 Hz and 4000 Hz to 2000 Hz (symmetric).

Exercise 2: Interrupt-driven sine wave

As for exercise 1, the bios configuration file has to be modified, as described in figure 2.1. The full *intio.c* file for this second exercise is shown in appendix 2.

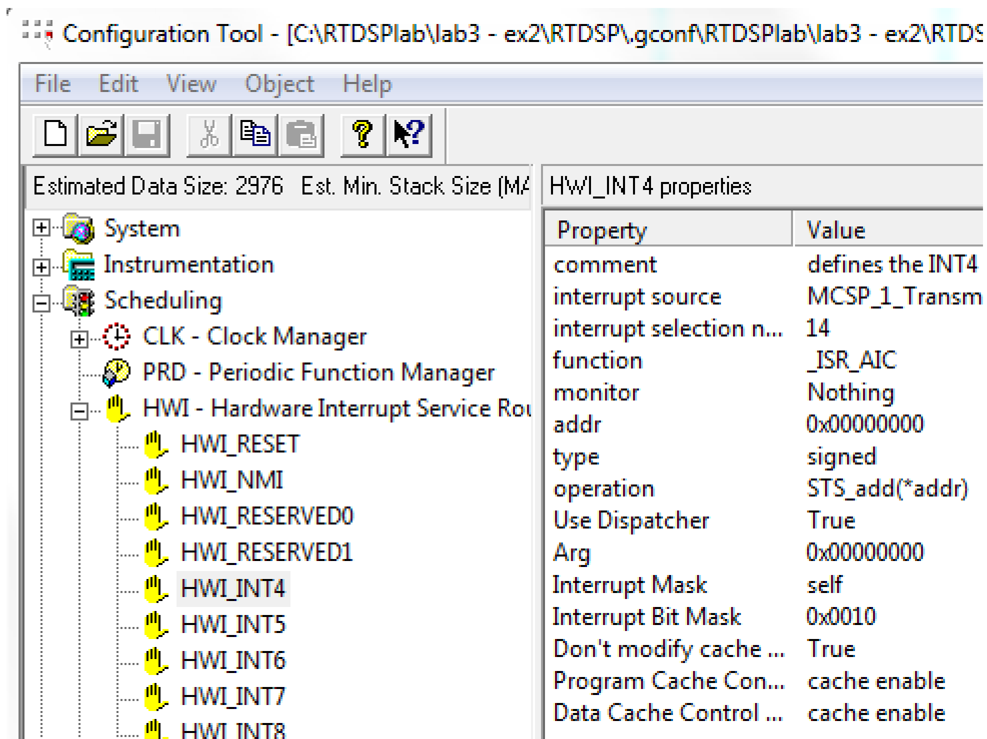


Figure 2.1 : BIOS configuration for interrupt HWI_INT4

The *ISR_AIC* function is changed as described in figure 2.2, and the functions *sine_init()* and *sinegen()* are added to the C code in order to achieve the generation of the sine wave from a look-up table.

```
/************************** INTERRUPT SERVICE ROUTINE ********************************/
void ISR_AIC(void) /*MODIFICATION_01 - Generates sample, full-rectify and output it*/
{
    sinegen(); //this changes the variable sample and sampleIndex
    if((fullrectify) && (sample > 0)){
        sample = -sample;}
    /*MODIFICATION_02 - Write the sample to the output*/
    while (!DSK6713_AIC23_write(H_Codec, ((Int32)(sample * Output_Gain)))) {};
}
```

Figure 2.2 : ISR_AIC function generating the sample to output

The global variable *sample* is an integer initialized to zero. The constant *Output_Gain* is equal to 2100000000. The Boolean *fullrectify* is set to true in order to rectify the samples generated. In this interrupt service routine, the variable *sample* is set by the function *sinegen()*. On the other hand, the function *sine_init()* is essential and must be executed at the start of the program (in the *main*).

```
/*********************************** sine_init()****************************************/
void sine_init(void) /*MODIFICATION_02*/
```

```
{
    int i = 0;
    while(i < SINE_TABLE_SIZE)
    {
        table[i] = sin((2 * PI * i) / SINE_TABLE_SIZE);
        i++;
    }
}
```

Figure 2.3 : sine_init() function definition

This function initiates a table of 256 values describing a full sine wave cycle. The resulting global array of floats *table* will then be used by *sinegen()* to generate the appropriate sample according to the desired sine wave frequency set (1000Hz by default).

```
/********************************* sinegen() ****************************************/
float sinegen(void) /*MODIFICATION_02 - Change the output sample index*/
{
    sample = table[sampleIndex];
    sampleIndex += SINE_TABLE_SIZE * (sine_freq/sampling_freq);
    if(sampleIndex >= SINE_TABLE_SIZE){
        sampleIndex -= SINE_TABLE_SIZE;}
}
```

Figure 2.4 : sinegen() function definition

The IF statement in this function is used to prevent setting an off-limit index of the *table* array corresponding to the sample to output. So this function mainly changes *sample* according to the *sampleIndex* set from the previous loop. This *sampleIndex* is its previous value added to 256 times the desired sine wave frequency and divided by the sampling frequency.

These modifications made the program work as desired. The working frequency range is now from 31.25Hz to 2KHz. The following signals are obtained for frequencies of 31.25Hz, 1000Hz and 1990Hz.

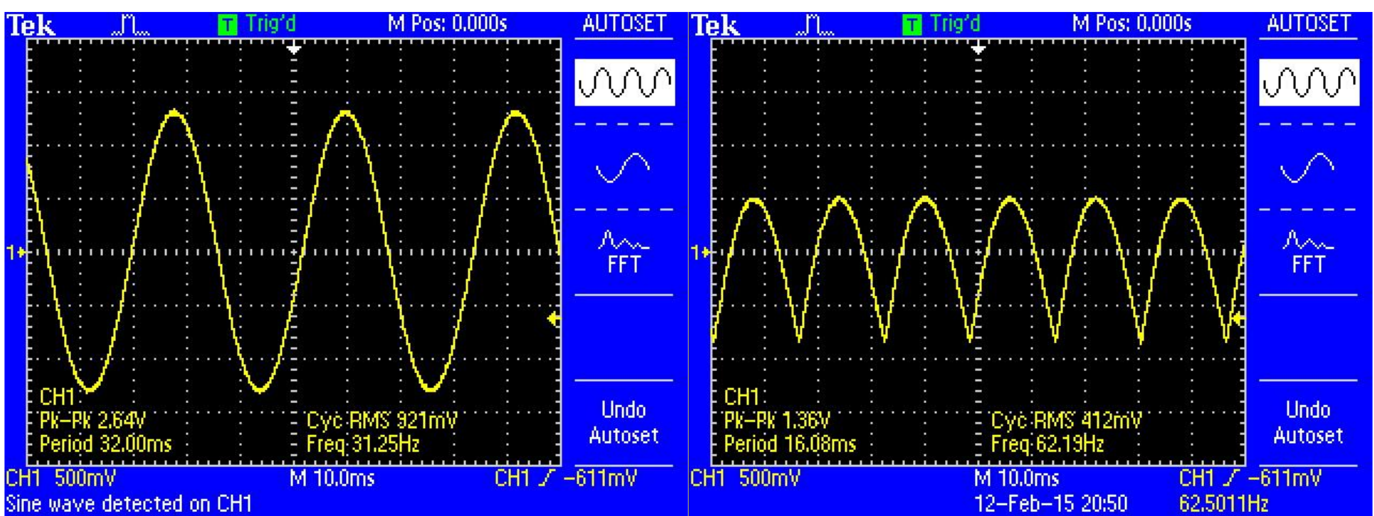


Figure 2.5 : Non-rectified (left) and rectified (right) signals at 31.25Hz input signal frequency

Lowering the frequency below that minimum limit will again give a zero DC voltage.

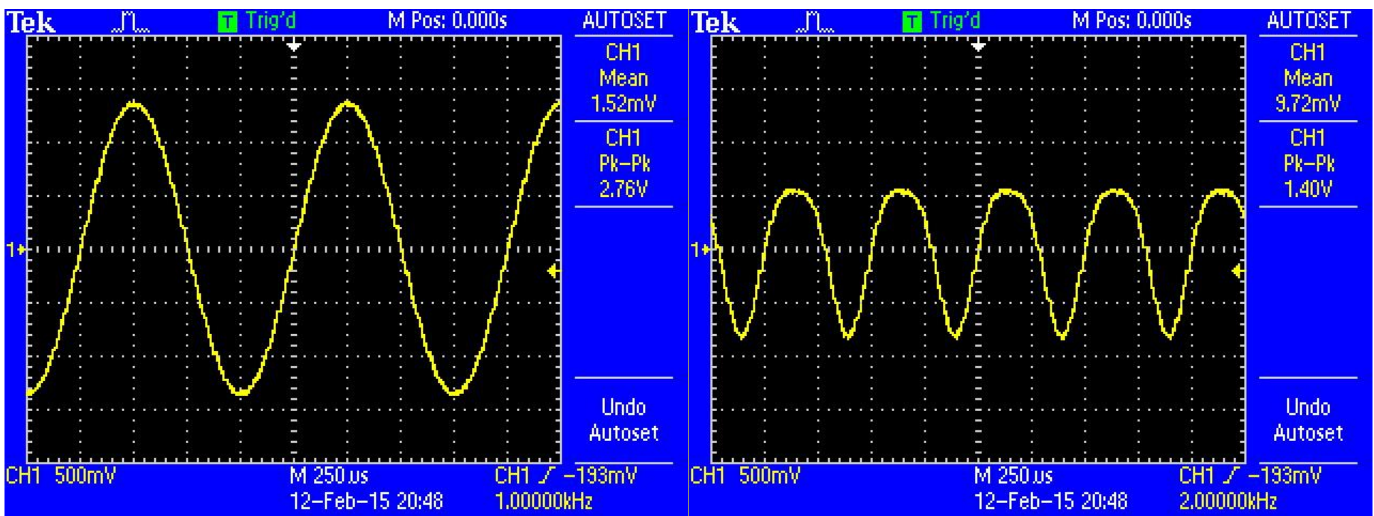


Figure 2.6 : Non-rectified (left) and rectified (right) signals at 1KHz input signal frequency

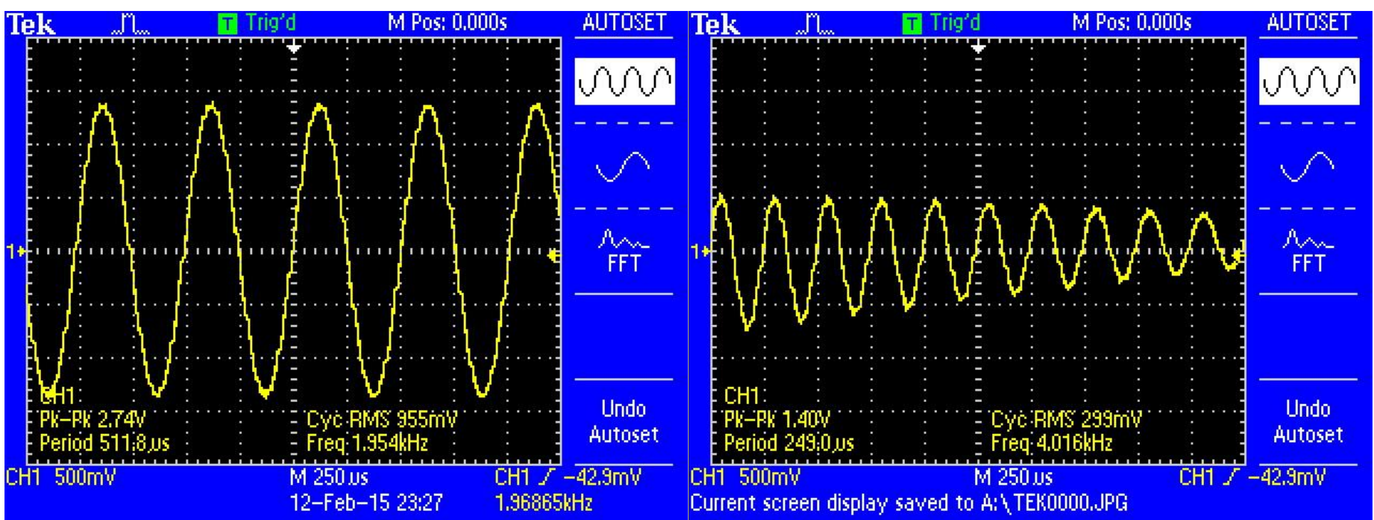


Figure 2.7 : Non-rectified (left) and rectified (right) signals at 1.990KHz input signal frequency

The output goes back to a DC zero voltage if the frequency is set to 2000 Hz through *sine_freq* global variable. Going over 2000 Hz will produce symmetric results. 4000 Hz will be associated to DC, and 3500 Hz will for example produce the same signal as an input sine wave with a 500 Hz frequency.

Please refer to the appendices for the two full codes with their associated comments.

Appendices

Appendix 1: C code file *intio.c* for exercise 1 (signal generator connected)

intioExercise1.c

```
/*************************************************************************************
 
                    DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                                   IMPERIAL COLLEGE LONDON 

                         EE 3.19: Real Time Digital Signal Processing
                              Dr Paul Mitcheson and Daniel Harvey

                                     LAB 3: Interrupt I/O

                               ********* I N T I O. C **********

  Demonstrates inputing and outputing data from the DSK's audio port using interrupts. 

 ************************************************************************************
                Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
                Updated for CCS V4 Sept 10
 ***********************************************************************************/
/*
 *  You should modify the code so that interrupts are used to service the 
 *  audio port.
 */
/************************** Pre-processor statements ******************************/

#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>
typedef int bool; /*MODIFICATION_01 - Defines boolean type for debugging*/
#define true 1
#define false 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;
bool fullrectify = true; /*MODIFICATION_01 - True to rectify the output signal */
int short rectified = 0; /*MODIFICATION_01 - Variable used to store the sample to
output */

 /******************************** Function prototypes ********************************/
void init_hardware(void);
void init_HWI(void);
void ISR_AIC(void); /*MODIFICATION_01 - Interrupt function declaration*/

/*********************************** Main routine *************************************/
void main()
{
    init_hardware(); //initialize board and the audio port
    init_HWI(); //initialize hardware interrupts
    while(1){}; //loop indefinitely, waiting for interrupts
}

/*********************************** 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
}

/************************* INTERRUPT SERVICE ROUTINE ********************************/
void ISR_AIC(void) /*MODIFICATION_01 - Read input port, full-rectify signal and
output*/
{
    rectified = mono_read_16Bit();
    if((fullrectify) && (rectified > 0)){
        rectified = -rectified;}
    mono_write_16Bit(rectified);
}
```

Appendix 2: C code file *intio.c* for exercise 2 (look up table)

intioExercise2.c

```
/*************************************************************************************
                   DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                                  IMPERIAL COLLEGE LONDON

                      EE 3.19: Real Time Digital Signal Processing
                          Dr Paul Mitcheson and Daniel Harvey

                                 LAB 3: Interrupt I/O

                            ********* I N T I O. C **********

  Demonstrates inputing and outputing data from the DSK's audio port using interrupts.

 *************************************************************************************
                 Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
                 Updated for CCS V4 Sept 10
 ************************************************************************************/
/*
 *  You should modify the code so that interrupts are used to service the
 *  audio port.
 */
/**************************** Pre-processor statements ******************************/

#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>

typedef int bool; /*MODIFICATION_01 - Defines boolean type for debugging*/
#define true 1
#define false 0
#define PI 3.141592653589793 /*MODIFICATION_02*/
#define SINE_TABLE_SIZE 256 /*MODIFICATION_02*/

/******************************* 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;
bool fullrectify = true; /*MODIFICATION_01 - True to rectify the output signal */
int sampling_freq = 8000; /*MODIFICATION_02 - Sampling frequency*/
float sine_freq = 1000.0; /*MODIFICATION_02 - sine wave frequency*/
float table[SINE_TABLE_SIZE]; /*MODIFICATION_02 - Table of sine wave samples*/
Int32 Output_Gain = 2100000000; /*MODIFICATION_02 - Gain for output (mono)*/
int sampleIndex = 0; /*MODIFICATION_02 - Sample Index corresponding to the next sample
to ouput*/
float sample = 0.0; //used in sinegen

 /******************************** Function prototypes *********************************/
void init_hardware(void);
void init_HWI(void);
void ISR_AIC(void); /*MODIFICATION_01 - Interrupt function*/
void sine_init(void); /*MODIFICATION_02 - Fills table with sine wave sample values*/
float sinegen(void); /*MODIFICATION_02 - Generates output sample of sine wave*/

/*********************************** Main routine ************************************/
void main()
{
    init_hardware(); // initialize board and the audio port
    init_HWI(); //initialize hardware interrupts
    sine_init(); /*MODIFICATION_02*/
    while(1) {}; //loop indefinitely, waiting for interrupts
}

/********************************* 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_XINT1,4);       // Maps an event to a physical interrupt
/*MODIFICATION_02*/
    IRQ_enable(IRQ_EVT_XINT1);      // Enables the event /*MODIFICATION_02*/
```

```c
    IRQ_globalEnable();            // Globally enables interrupts
}

/************************** INTERRUPT SERVICE ROUTINE ********************************/
void ISR_AIC(void) /*MODIFICATION_01 - Generates sample, full-rectify and output it*/
{
    sinegen(); //this changes the variable sample and sampleIndex
    if((fullrectify) && (sample > 0)){
        sample = -sample;}
    /*MODIFICATION_02 - Write the sample to the output*/
    while (!DSK6713_AIC23_write(H_Codec, ((Int32)(sample * Output_Gain)))) {};
}

/********************************** sine_init()
****************************************/
void sine_init(void) /*MODIFICATION_02*/
{
    int i = 0;
    while(i < SINE_TABLE_SIZE)
    {
        table[i] = sin((2 * PI * i) / SINE_TABLE_SIZE);
        i++;
    }
}

/********************************** sinegen() ***************************************/
float sinegen(void) /*MODIFICATION_02 - Change the output sample index*/
{
    sample = table[sampleIndex];
    sampleIndex += SINE_TABLE_SIZE * (sine_freq/sampling_freq);
    if(sampleIndex >= SINE_TABLE_SIZE){
        sampleIndex -= SINE_TABLE_SIZE;}
}
```