

REAL TIME DIGITAL SIGNAL PROCESSING

LAB 3 REPORT



February 4, 2015

Imperial College London

Jiabo Zhou (JZ2611), Hao Ding (HD1812)

**CONTENT**

**1. Quiz Answer**

1.1 Centralised Rectified Waveform

1.21 Frequency limitation for full-wave rectification

1.22 Rectification for 3.8 kHz sine wave

**2. Code Explanation**

2.1 Interrupt Service Routine

2.2 Interrupt-driven Sine Wave

**3. Results and Discussions**

3.1 Scope Trace

3.2 Observation and Explanation

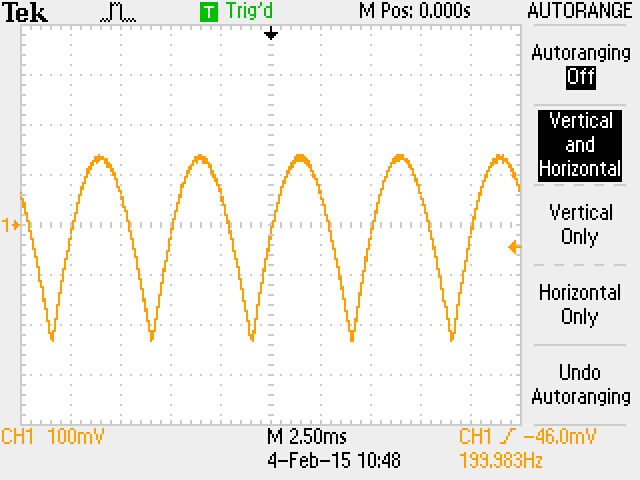
**4. Appendix**

**5. Reference**

**Quiz Answer**

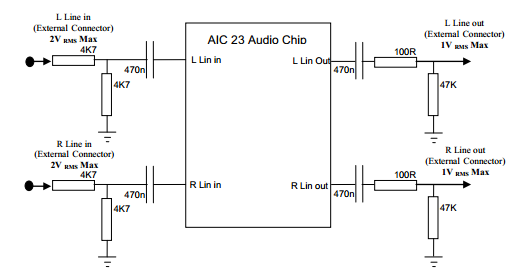
* 1. **Centralised Rectified Waveform**

The full rectified waveform is observed to be centred at around 0V, instead of always being above 0V as expected (Figure 1). This is because the DC offset is removed by the high pass filter at output port.



**Figure 1.** Rectified Waveform centred around 0 V

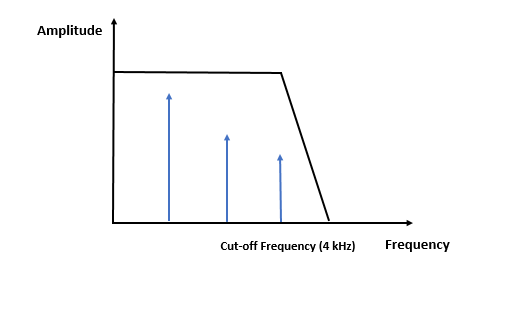
In Figure 2, it can be seen that high-pass filters are attached to left and right output ports. Rectification outputs a frequency spectrum of even multiples of input frequency, including 0 Hz, which is removed by high-pass filter. The cut-off frequency of high-pass filter is 7.1 Hz, using the formula:



**Figure 2 [1].** High pass filter at output port

**1.21 Frequency limitation for full-wave rectification**

The output waveform is a full-wave rectified only when input signal is below a certain frequency. This happens due to the non-linearity of rectification process, which changes the spectrum of input signal. As input is a sine wave, full-wave rectification yields even harmonics of input wave, i.e. 0f, 2f 4f, etc. Our DSK board is over sampled. As a result, processed signal is passed through a low-pass filter with cut-off frequency at 4 kHz before producing 8 kHz samples.

****

**6f**

**4f**

**2f**

**Figure 3.** Full-rectified signal at low-pass filter

In order to show the property of full-rectified signal, at least three components are required, that is to say, the 2f, 4f, 6f harmonics are necessary. Therefore, the maximum frequency for full-wave rectification to be observed is:

When input frequency is becoming closer but lower than 666.7 Hz, imperfect wave can still be observed due to the missing frequency components are significant enough to cause distortion.

**1.22 Rectification for 3.8 kHz sine wave**

**Figure 4.** Wave for in time and frequency domain for 3.8 kHz input

|  |  |
| --- | --- |
| Time Domain | Frequency Domain |
| D:\TEK0004.JPG | D:\TEK0026.JPG  Remaining peaks are result of oscilloscope sampling  400 Hz |

When input signal is 3.8 kHz, a stable waveform with frequency exactly 400 Hz is shown on oscilloscope (Figure 4).

**7.6 kHz**

**Amplitude**

**15.6 kHz**

**Frequency**

**Fs = 8 kHz**

**400 Hz**

**Figure 5.** Spectrum rectifies 3.8 KHz signal after sampling

A 3.8 kHz sine wave yields a combined harmonics of 0 Hz, 7.6 kHz, 15.2 kHz, etc. After the high-pass filter with cut-off frequency at 7.1 Hz, only 0 Hz wave, that is the DC offset, is removed. After sampling at 8 kHz, periodic components are shown around multiples of 8 kHz, since multiplying of rectified waveform and delta function train is equivalent to convolution in frequency domain. The 7.6 kHz peak around 8 kHz sampling frequency, therefore, corresponds to a 400 Hz peak at baseband due to aliasing (Figure 5).

**Code Explanation**

**2.1 Interrupt Service Routine**

In exercise 1, we are required to write an interrupt function which read, write and rectify samples. The basic principle of exercise 1 is that, after signal is generated from software generator, it is read by the function *mono\_read\_16Bit ()* and causes an interrupt routine to rectify signal. Eventually signal is output to AIC audio port through the function *mono\_write\_16Bit ().*

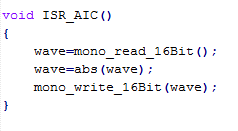
Interrupt event is set to *MCBSP\_1\_receive* which calls interrupt function when signal is received. It is then mapped to the physical event *IRQ\_EVT\_RINT1* in *IRQ\_Map ().* The interrupt priority number is 4, which matches the interrupt number.

Sampling frequency is set to 8 kHz by the following command:

We use a handle to set up configuration:

## 

In the interrupt function below, signal is first read into a variable called *wave*, which is rectified using Abs function in *math.h* library. *Wave* is declared as type *short*, which is 16 bit integer (suitable format for mono\_write\_16Bit function). After that, wave is outputted to oscilloscope.

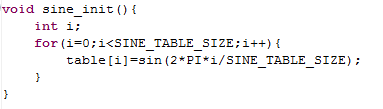


The programme *Signal Generator* provided on Blackboard is used in exercise.

**2.2 Interrupt-driven Sine Wave**

In exercise 2, we are required to write an interrupt function to rectify signal generated from a lookup-table. The main principle is similar to exercise 1. However, instead of trigger an interrupt routine when signal is received, interrupt is called when transmitting, since no signal is passed to input port in this exercise.

Look-up table initialised to contain a whole sine wave using a For-loop:



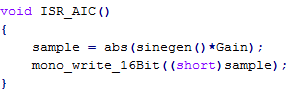
The variable *table* is defined as an array containing 256 elements, all of them are in *float* type to ensure accuracy.

However, value stored in look-up table are all 16-bit floating point, which are essentially in the range of -1 to 1, which is tiny and hardly distinguished by processor.

Therefore, we define a constant *Gain*, with the value of 215-1 = 32767, which is the maximum number represented by 16bit signed number.



The ISR is associated with the *MCSP\_1\_Transmit*, which is then mapped to the physical event *IRQ\_EVT\_XINT1* to trigger interrupt when transmitting.



Sample is also declared as type *float*, which is assigned to the absolute value of sine wave in lookup table after amplified by *Gain*. Sample is then converted to *short* integer and passed to audio port by *mono\_write\_16Bit ().*

|  |  |
| --- | --- |
| **f=8 Hz** | **f=20 Hz** |
| D:\TEK0025.JPG | D:\TEK0011.JPG |
| **f=100 Hz** | **f=500 Hz** |
| D:\TEK0013.JPG | D:\TEK0014.JPG |
| **f=1.3 kHz** | **f=2 kHz** |
| D:\TEK0016.JPG | D:\TEK0017.JPG |

**Results and Discussion**

***Table 1.*** *Scope traces for 0 Hz < f <2 kHz*

**3.1 Scope Traces**

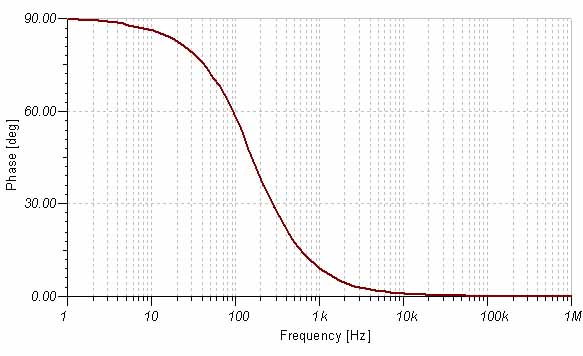
***Table 2.*** *Scope traces for 2 kHz < f <4 kHz*

|  |  |
| --- | --- |
| **f=2.7 kHz** | **f=3.5 kHz** |
| D:\TEK0019.JPG | D:\TEK0020.JPG |
| **f=3.9 kHz** | **f=3.98 kHz** |
| D:\TEK0021.JPG | D:\TEK0022.JPG |
| **f=3992 Hz** | **f>4 kHz** |
| D:\TEK0024.JPG | **A copy of low frequency spectrum is observed, explained in 3.2** |

**3.2 Observation and Explanation**

**Below 10 Hz**

From table 1, distorted waveform is observed at very low frequency. This is caused by the phase lag of high-pass filter (Figure 6).



**Figure 6 [2]**. Phase response for a common high-pass filter

The frequency spectrum of full-rectified signal at very low frequency consists of many peaks ranging from low to high. However, the phase response for different frequency components varies. Lags are introduced to high frequency components while low frequency parts has no phase lag. Such effect contribute to the distorted signal shape at very low frequencies.

**Between 10 Hz and 100 Hz**

For input signals within this region, a clear full-rectified waveform is shown on screen. Effect of phase lag gradually disappear because low-frequency components shift to right as frequency increases so that all harmonics have the same phase lag.

**Between 100 Hz and 1000 Hz**

According to Table 1, an imperfect full-rectified signal with smooth edge is generated. Our calculation to 1.1 has provided evidence that the missing components are significant enough to cause signal distortion for input within and above this frequency region.

For instance, frequency spectrum of 5 kHz signal is shown below:

**Fs=4 kHz**

**Amplitude**

**Frequency (Hz)**

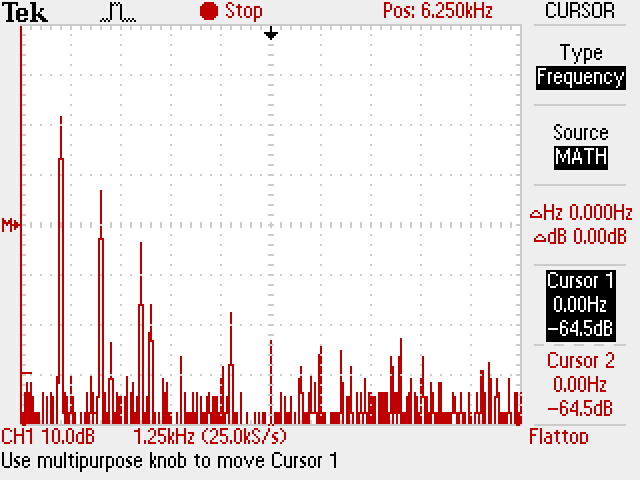
**2k**

**3k**

**4k**

**1k**

**Figure 7.** Spectrum of 500 Hz signal with low-pass filter

Shown in Figure 7, there are three components remaining after low-pass filter and one peak at filter transition band.

**Figure 8**. Spectrum of 500 Hz signal on oscilloscope

It is observed that the fourth peak at 4 kHz is unstable. The first three frequency component is still not enough to produce an accurate graph as 500 Hz is quite close to the frequency boundary, 666.7 Hz.

**Between 1000 Hz and 2000 Hz**

At 1.3 kHz, amplitude modulation effect of Oscilloscope step in.

**Amplitude**

**5.4**

**Fs=8 k**

**Frequency (KHz)**

**5.2**

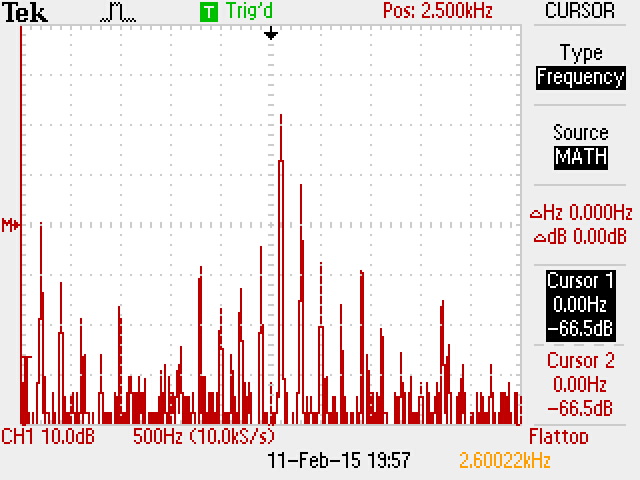
**FCut-off**

**2.8**

**2.6**

**Figure 9**. Spectrum of rectified 1.3 KHz sine wave

See Figure 9, by using a low-pass filter, it is expected to keep peaks at 2.6 kHz only. However, a higher harmonic, 2.8 kHz, centred at sampling frequency fall in the pass band. This peak has significant effect on the scope trace.



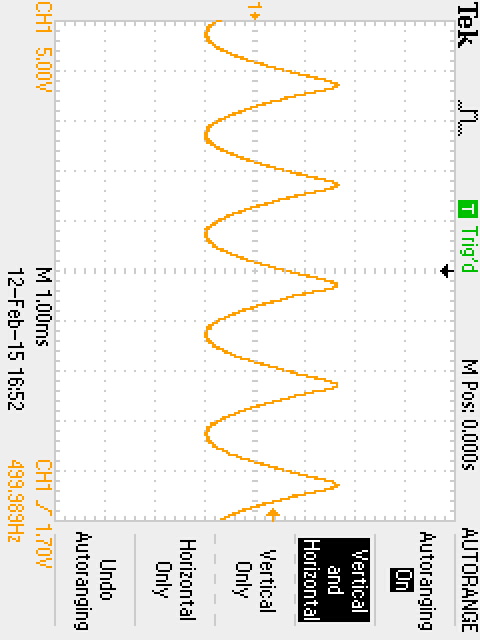
**Figure 10**. FFT of rectified 1.3 KHz sine wave

Figure 10 shows two high peaks at 2.6 kHz and 2.8 kHz, which proves our theory. As a result, it can be concluded that sine wave suffer from amplitude modulation effect in this frequency region.

**Between 2000 Hz and 4000 Hz**

From table 2, it is noticed that scope traces are similar to these between 0 to 2000 Hz. This is mainly because the rectification process doubles the most important harmonic. With the Nyquist frequency at 4 kHz, rectification of wave above 2000 Hz shows obvious aliasing effect. Essentially, the observed frequency and trace is same as f = fNyquist - fin. For example, trace of 3.8 kHz has the same waveform as input is: 4000 – 3800 = 200 Hz.

**Above 4 kHz**

****In theory, wave shape for frequency above 4 kHz is essentially a copy of the fundamental system (0 ~ 4 kHz). An example is shown in Figure 11.

**Figure 11**. Rectified wave at 4.25 KHz

Figure 11 is an example of rectifying sinewave at 4.25 kHz, resultant wave is at approximately 500 Hz. Theory is explained in the following:

**8.5 kHz**

**Frequency**

**500 Hz**

**-500 Hz**

**8 kHz**

**-8 kHz**

**Figure 12.** Frequency Spectrum of rectified wave at 4.25 KHz

In Figure 12, after rectifying sinewave at 4.25 kHz, harmonics appears at 8.5 kHz, 17 kHz, etc. After sampling and passing through a low-pass filter, a 500 Hz peak is observed due to the spectrum which should be centred at -8 kHz. Therefore, when frequency is above 4 kHz, observed frequency is:

(N is integer which make observed frequency within filter pass band)

**Summary of frequency behaviour**

**Good Shape**

**Good Shape**

**Amplitude Modulation**

**Amplitude Modulation**

7990 Hz

6 kHz

10 Hz

2 kHz

0 Hz

8 kHz

7 kHz

4 kHz

1 kHz

**Phase Lag**

**Imperfect Wave**

**Imperfect Wave**

**Phase Lag**

**0 Hz**

**Figure 13.** Observations of rectification on different frequency regions

**0 Hz**

**4000 Hz**

**4. Appendix**

**Exercise 1**

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  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>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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;  **short** wave;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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()  {  wave=mono\_read\_16Bit();  **if**(wave>0){  wave=-wave;  }  mono\_write\_16Bit(wave);  } |

**Exercise 2**

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  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"  // Some functions to help with configuring hardware  **#include** "helper\_functions\_polling.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** PI 3.141592653589793  **#define** Gain 32767  **#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 \*/\  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  };  //Global variable definition  **float** sine\_freq = 1000.0;  **float** sample;  //Table to hold sine wave  **float** table[SINE\_TABLE\_SIZE];  /\*Left and right audio channel gain values, calculated to be less than signed 32 bit  maximum value. \*/  **int** sampling\_freq = 8000;  **int** i=0;  // 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**);  **float** sinegen(**void**);  **void** sine\_init();  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main routine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **void** main(){    // initialize board and the audio port  init\_hardware();    /\* initialize hardware interrupts \*/  init\_HWI();    // Initialise look up table  sine\_init();    /\* loop indefinitely, waiting for interrupts \*/  **while**(1)  {  // Set the sampling frequency. This function updates the frequency only if it  // has changed. Frequency set must be one of the supported sampling freq.  set\_samp\_freq(&sampling\_freq, Config, &H\_Codec);  };    }    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 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  IRQ\_enable(IRQ\_EVT\_XINT1); // Enables the event  IRQ\_globalEnable(); // Globally enables interrupts  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WRITE YOUR INTERRUPT SERVICE ROUTINE HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **void** ISR\_AIC()  {  sample = abs(sinegen()\*Gain);  mono\_write\_16Bit((**short**)sample);  }  **float** sinegen(**void**)  {  /\* This code produces a fixed sine of 1KHZ (if the sampling frequency is 8KHZ)  using a digital filter.  You will need to re-write this function to produce a sine of variable frequency  using a look up table instead of a filter.\*/    // temporary variable used to output values from function  **float** wave;  **float** inteval;    //take sample from look up table  wave=table[(**int**)i];    //interval between each sample  inteval=SINE\_TABLE\_SIZE\*sine\_freq/sampling\_freq;    //increment sample number  i=i+inteval;    //keep i below table size and generate data continuously  **if**(i>=SINE\_TABLE\_SIZE){  i=i-SINE\_TABLE\_SIZE;  }    **return**(wave);    }  //Function to initialise table  **void** sine\_init(){  **int** i;  **for**(i=0;i<SINE\_TABLE\_SIZE;i++){  table[i]=sin(2\*PI\*i/SINE\_TABLE\_SIZE);  }  } |

**5. Reference**

[1] RTDSP\_Lab 3, page 9, Figure 4: AIC23 Audio chip external components.

[2] http://www.ee.surrey.ac.uk/Projects/CAL/frequency-response/passhp.html