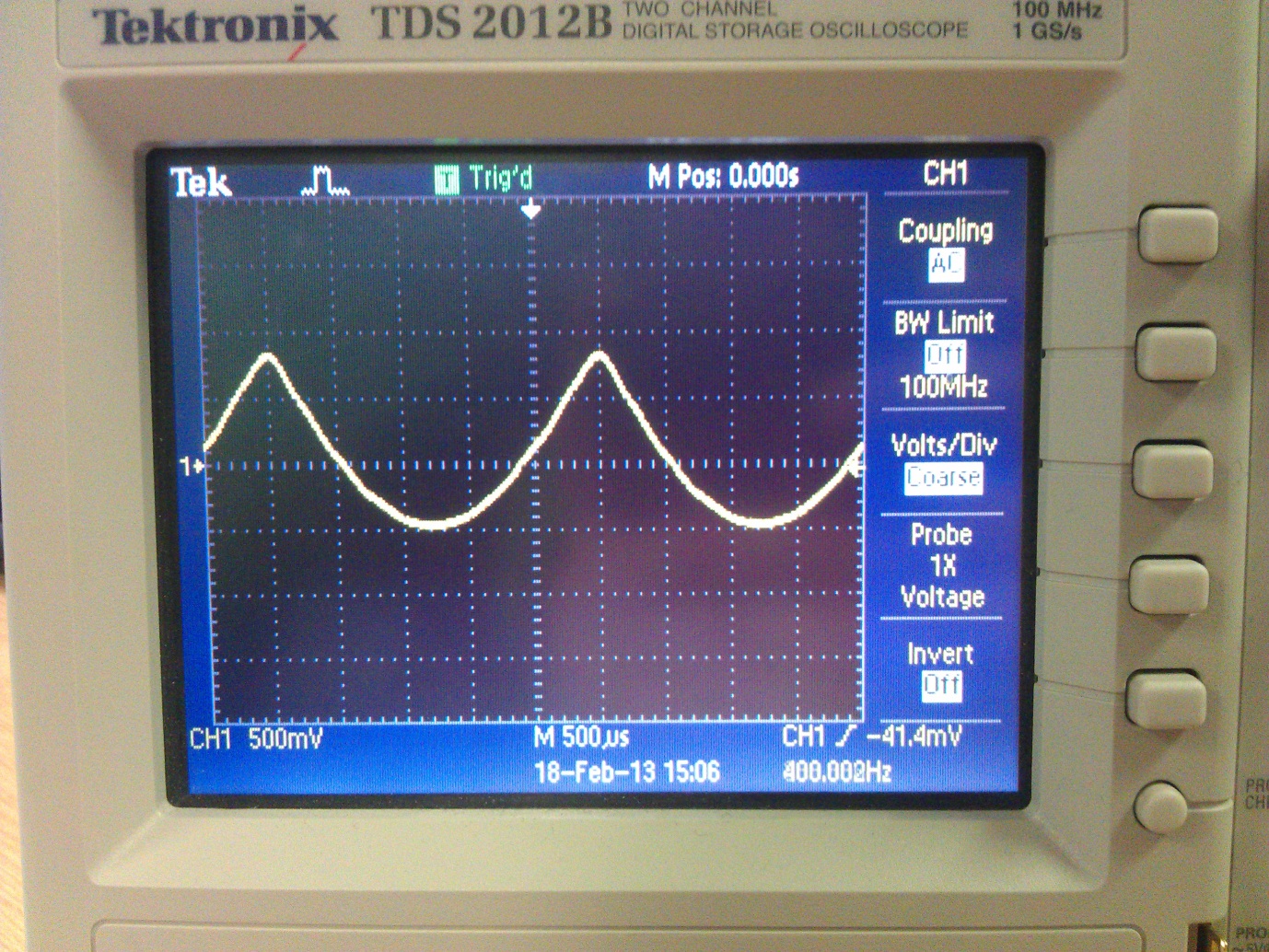
**RTDSP Lab 3**

**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.

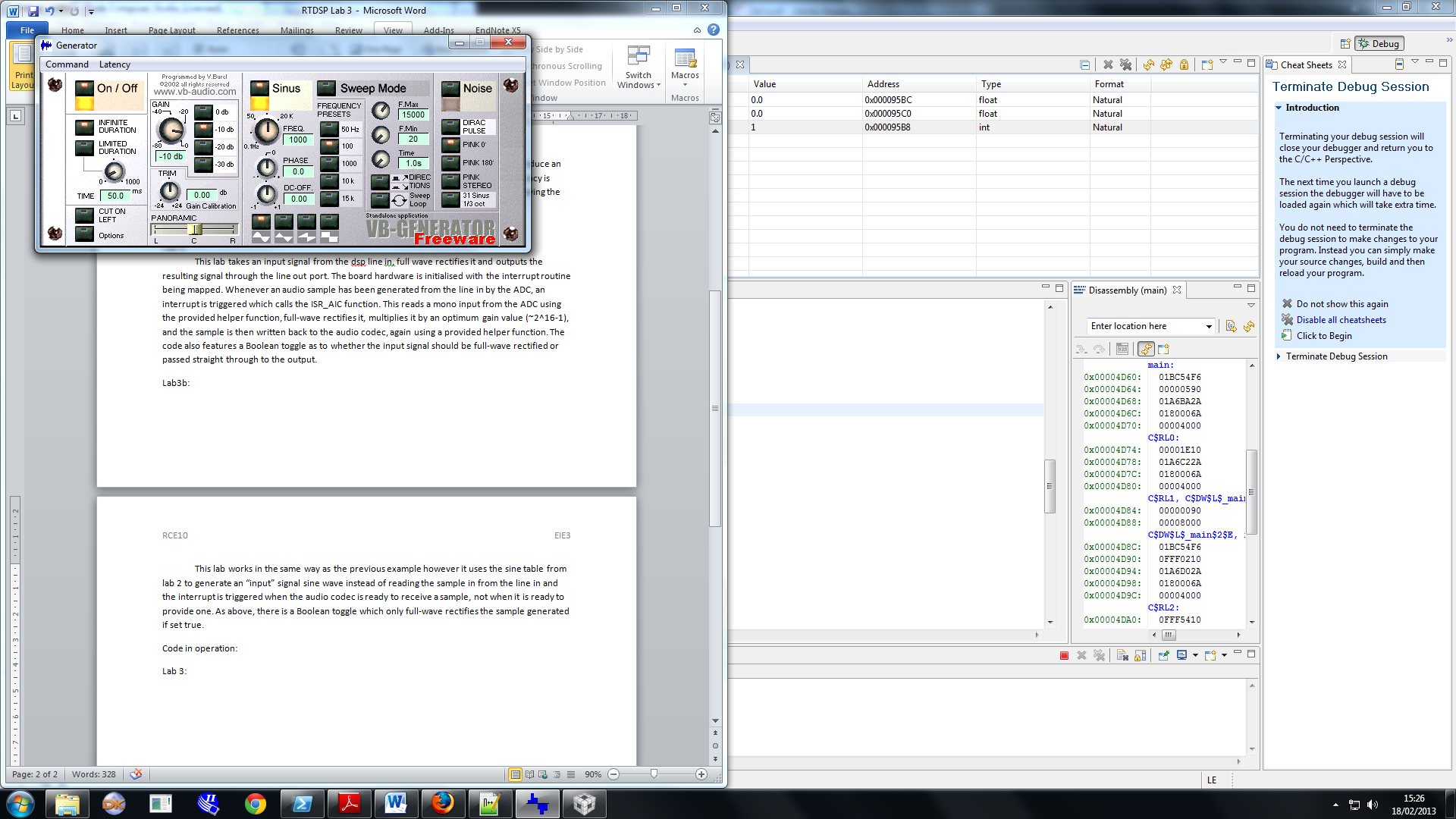
Lab3b:

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.

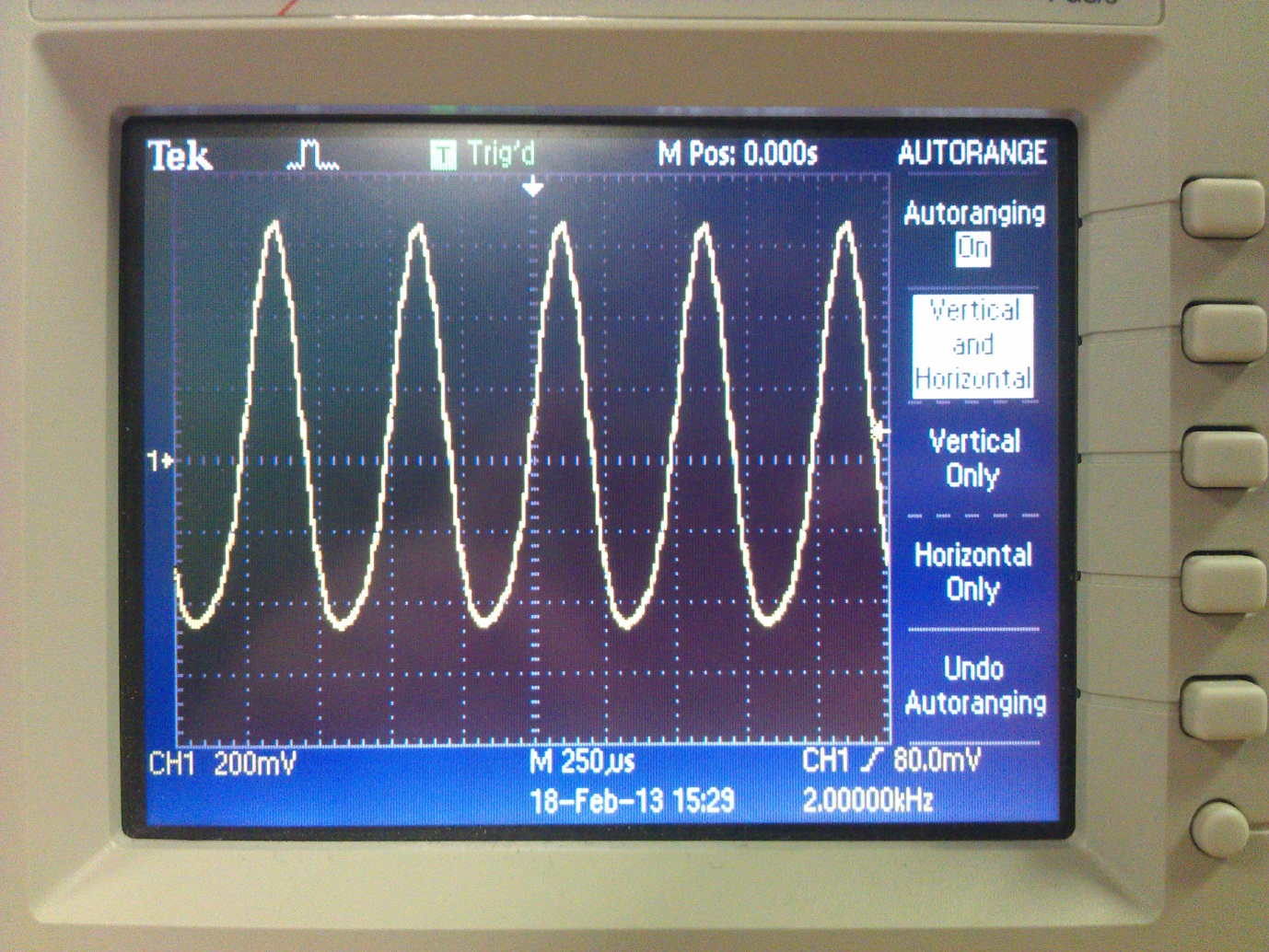
**Code in operation**:

Lab 3:

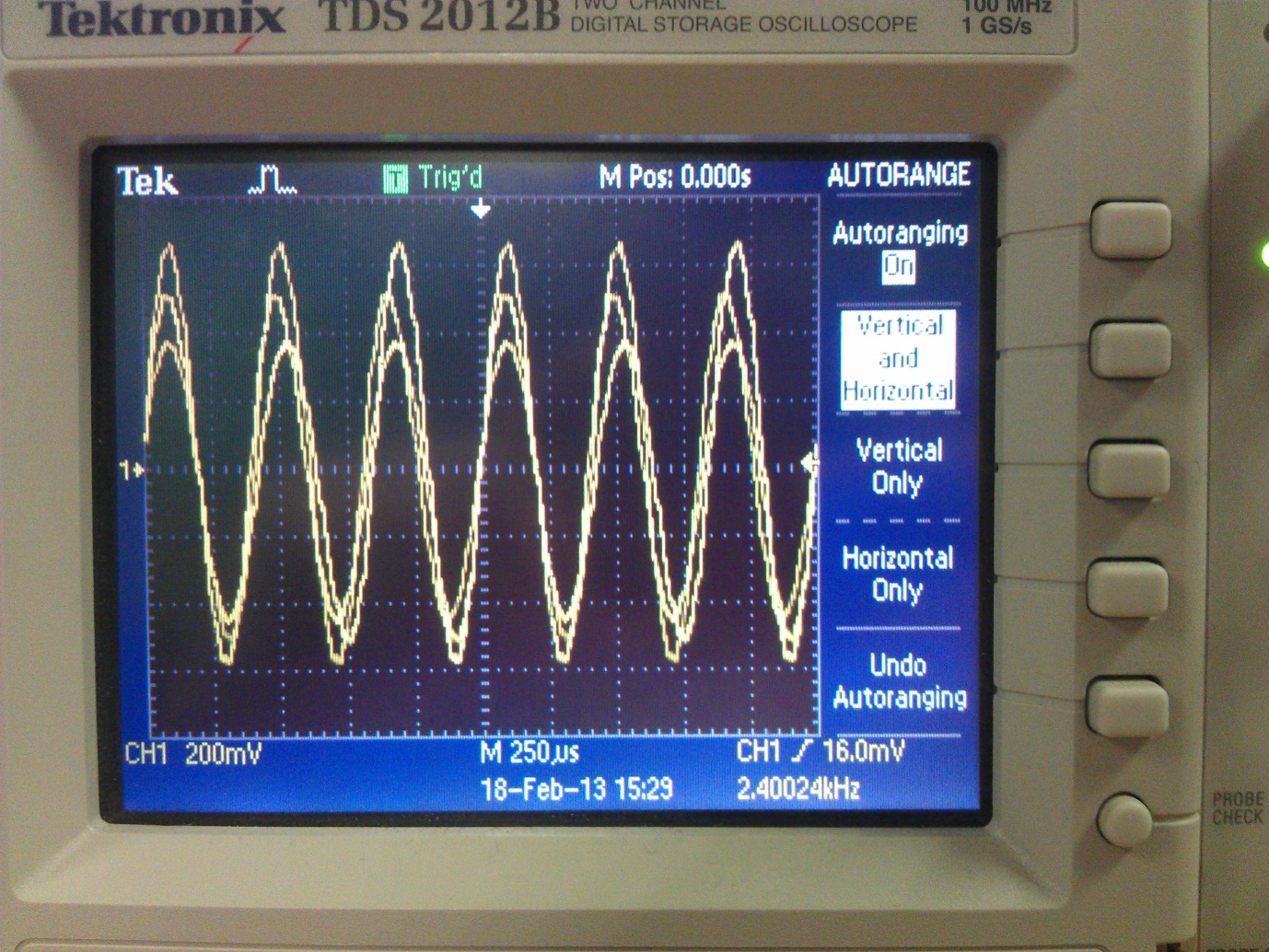
The limit for Lab 3 was an input of 1kHz.



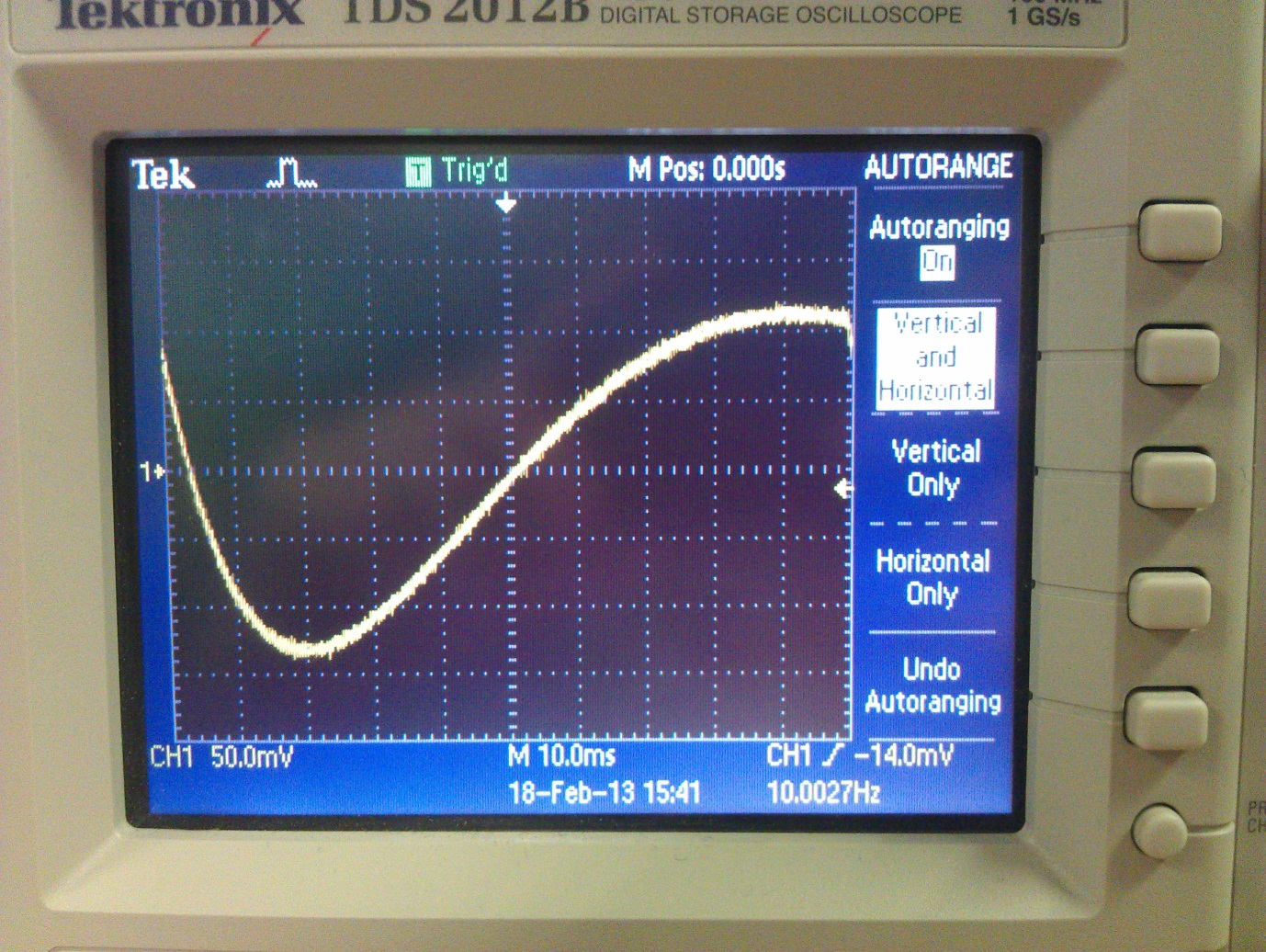
This gives the output as below:



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

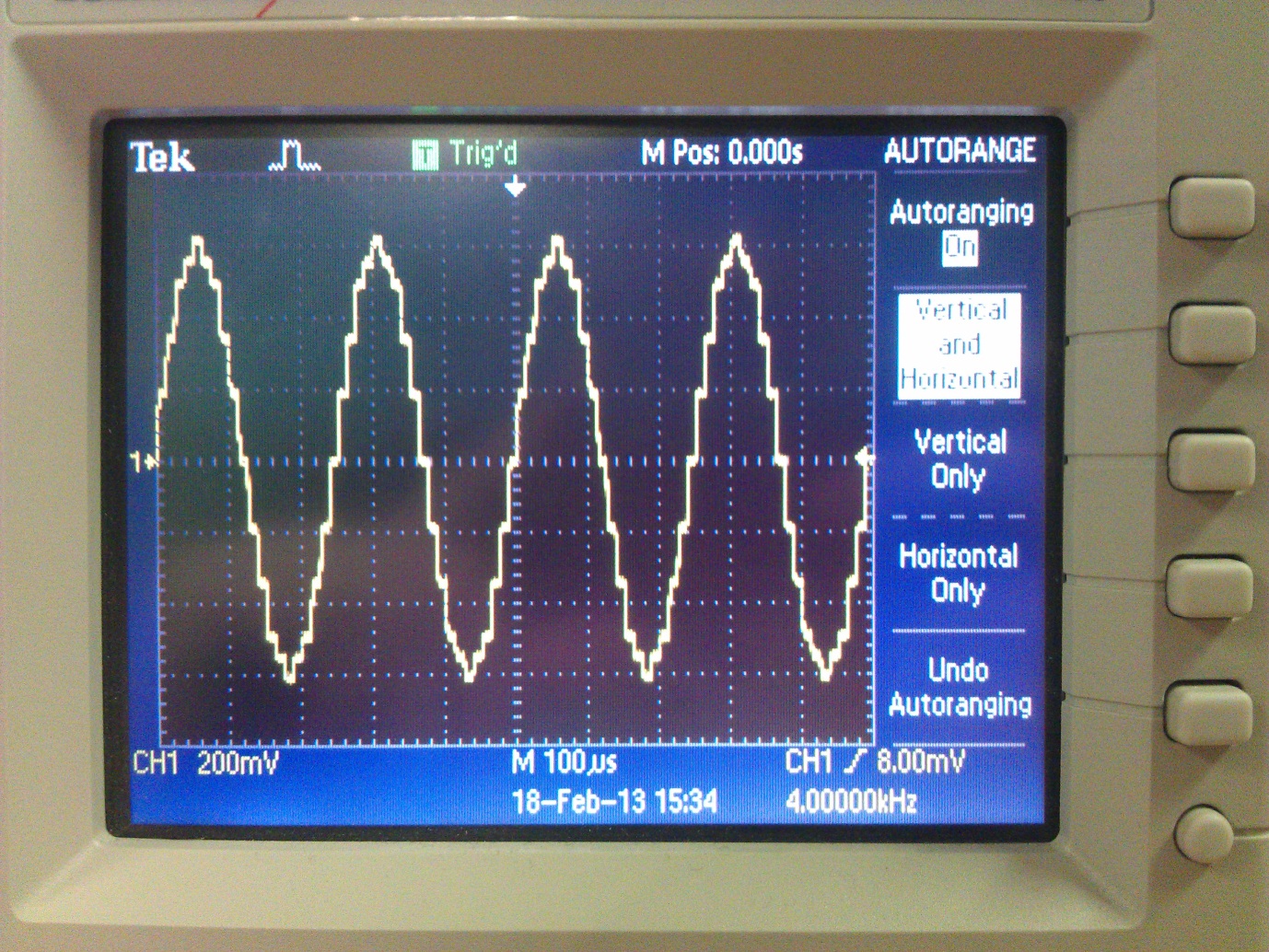


The code begins working correctly at a frequency of around 5Hz as shown below although gives a misshapen output waveform as below:

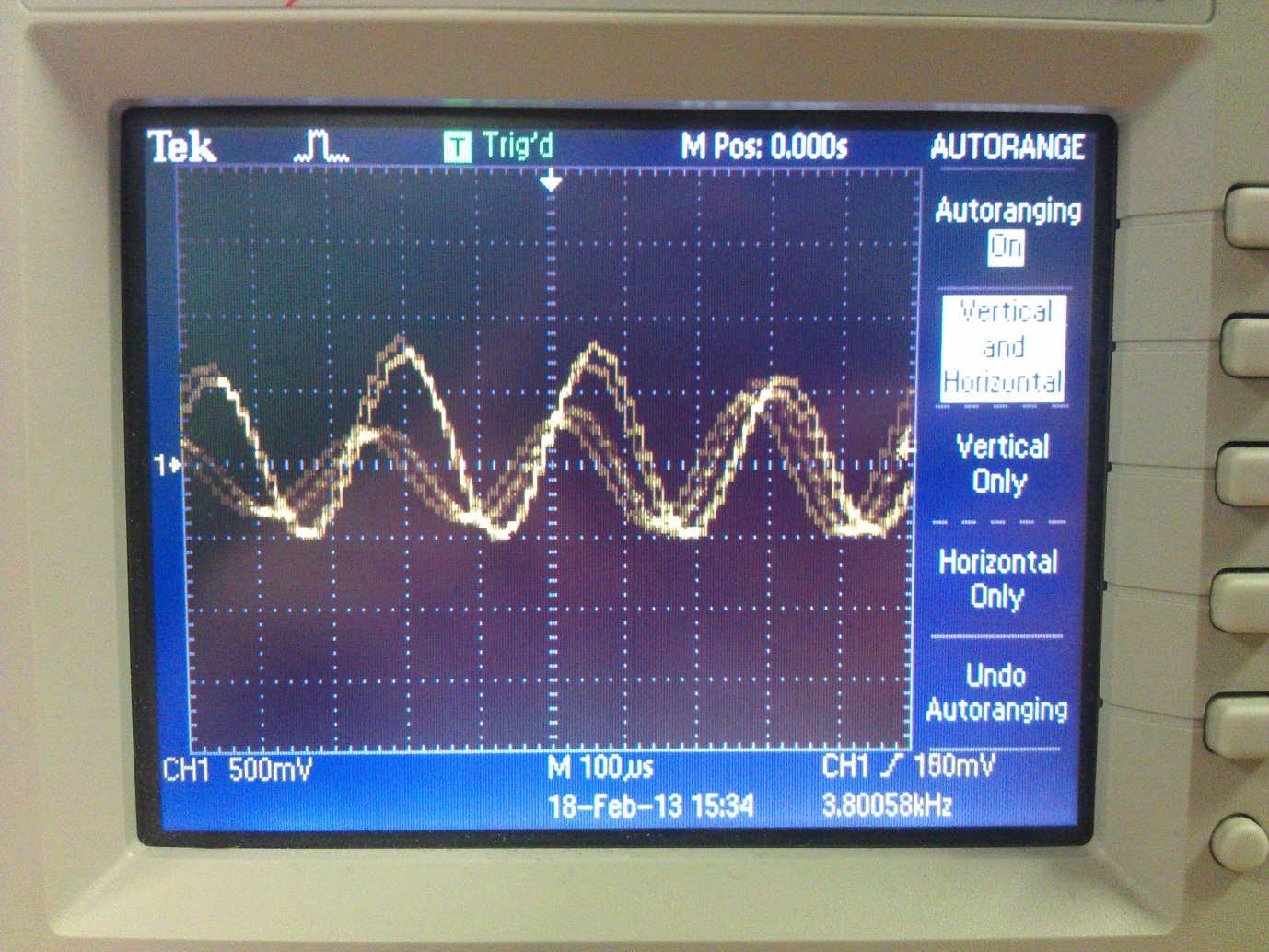


Lab 3b:

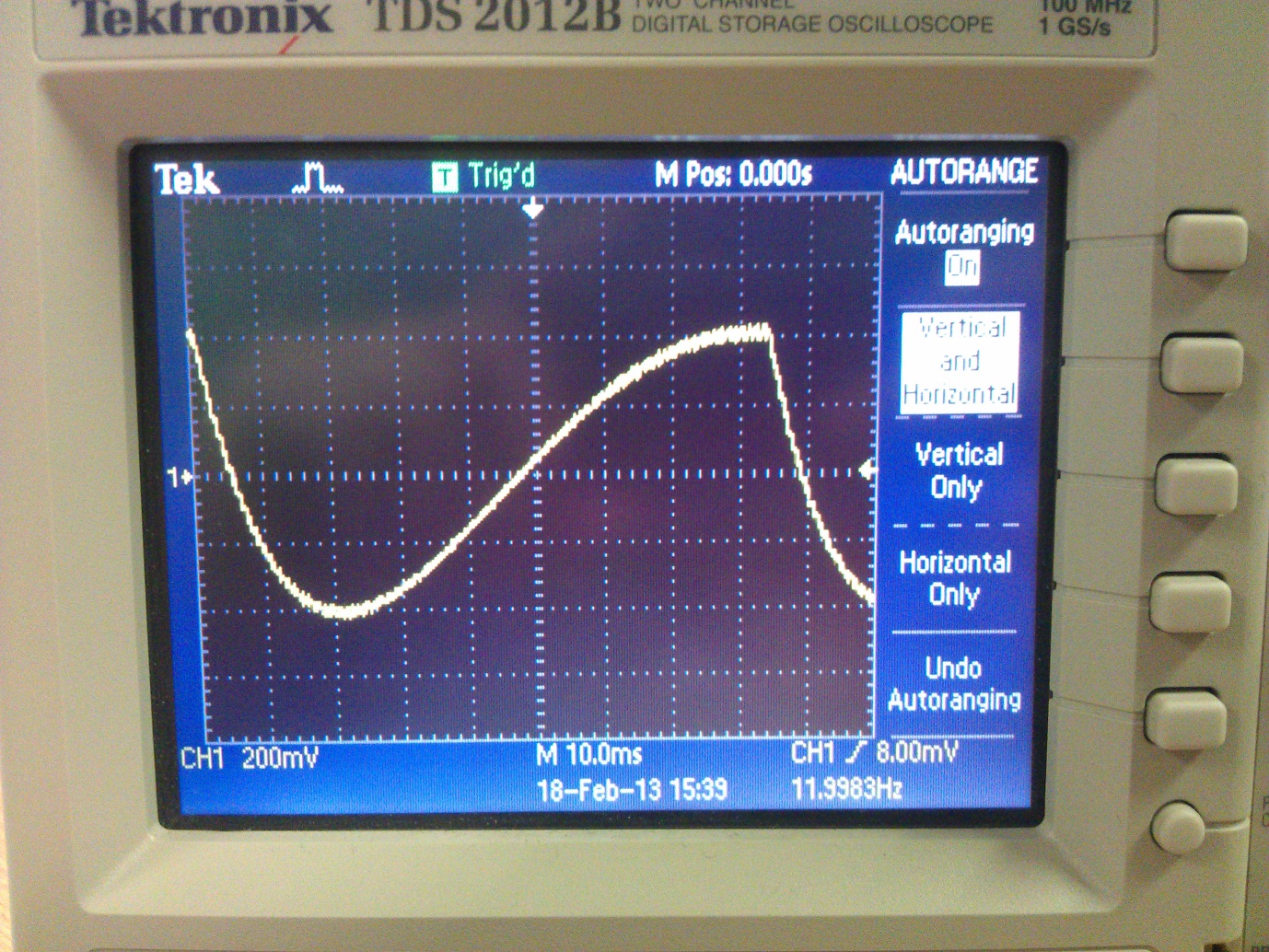
Lab 3b works up to an input frequency of 2kHz as shown below:



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);

}