**RTDSP Lab 2**

Questions:

1. The following is a trace table for the running of the sinegen function

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0,707100000000000 | 0,999980820000000 | 0,707072875644000 | -3,83592642554387e-05 | -0,707127123315510 | -0,999980818528539 | -0,707045750247549 | 7,67185284543670e-05 | 0,707154245590490 |

As can be seen from the table, a complete wave consists of 8 generated samples.

1. The output of the sinewave is fixed at 1kHz because the sample is only generated once per loop of the main function. The progress of this function is throttled because of

// send to LEFT channel (poll until ready)

**while** (!DSK6713\_AIC23\_write(H\_Codec, ((Int32)(sample \* L\_Gain))))

{};

// send same sample to RIGHT channel (poll until ready)

**while** (!DSK6713\_AIC23\_write(H\_Codec, ((Int32)(sample \* R\_Gain))))

{};

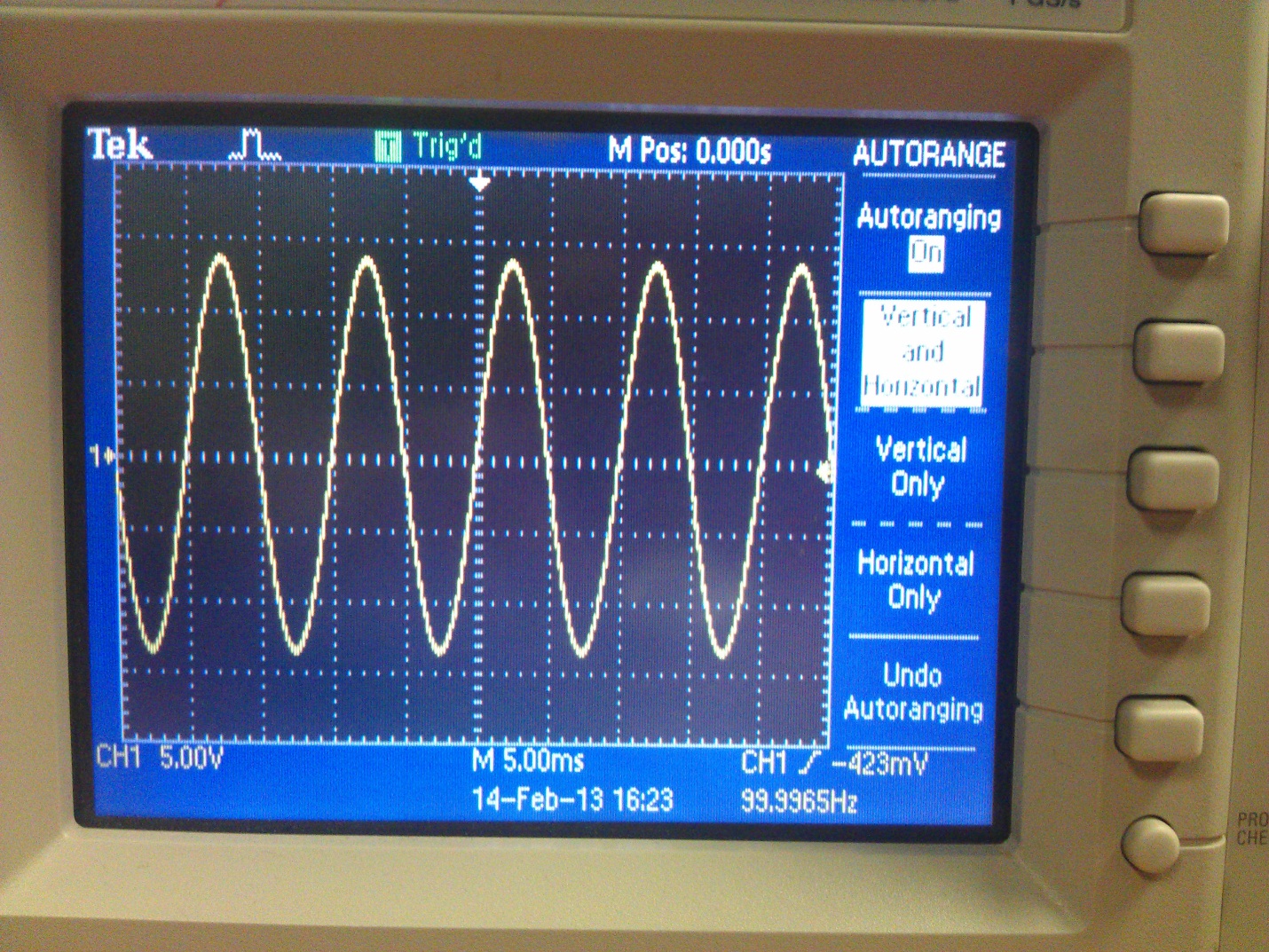
This means that the function will stall until the audio interface is ready to receive a sample which occurs at the frequency specified in sampling\_freq.

1. ((Int32)(sample \* R\_Gain) suggests that the result of sample \* gain is being explicitly cast to an 32 bit integer.

Code Operation:

In the code, the hardware is initialised to default settings. The function init\_sine() then runs which populates an array of size SINE\_TABLE\_SIZE with values of the sine function as calculated by the math sine() function. The main infinite loop then starts which begins by retrieving the current sine table index value. The hardware then waits until both the left and right AIC channels are ready to receive samples and then sends the sample to the codec. The current sample is calculated in the sinegen() function which calculates the offset to the next value relative to the previous, wrapping around if the bounds of the sine table array are exceeded and returns the sample value.

Code running:



Limitations:

When the frequency is set very low, results are increasingly erratic until when very low the signal cannot be recognised as a sine wave. This is due to the high pass filter associated with the codec which strips DC gain as this is undesirable in an audio system. As the frequency is increased, the results also become skewed as you approach the nyquist frequency. This is because the sampling rate is no longer sufficient to correctly reconstruct the desired signal.

Commented code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

IMPERIAL COLLEGE LONDON

EE 3.19: Real Time Digital Signal Processing

Dr Paul Mitcheson and Daniel Harvey

LAB 2: Learning C and Sinewave Generation

\*\*\*\*\*\*\*\*\* S I N E . C \*\*\*\*\*\*\*\*\*\*

Demonstrates outputing data from the DSK's audio port.

Used for extending knowledge of C and using look up tables.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Updated for use on 6713 DSK by Danny Harvey: May-Aug 06/Dec 07/Oct 09

CCS V4 updates Sept 10

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Initialy this example uses the AIC23 codec module of the 6713 DSK Board Support

\* Library to generate a 1KHz sine wave using a simple digital filter.

\* You should modify the code to generate a sine of variable frequency.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Pre-processor statements \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 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\_polling.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 \*/\

0x004f, /\* 7 DIGIF Digital audio interface format 32 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;

// Array of data used by sinegen to generate sine. These are the initial values.

**float** y[3] = {0,0,0};

**float** x[1] = {1}; // impulse to start filter

**float** a0 = 1.4142; // coefficients for difference equation

**float** b0 = 0.707;

// Holds the value of the current sample

**float** sample;

/\* Left and right audio channel gain values, calculated to be less than signed 32 bit

maximum value. \*/

Int32 L\_Gain = 2100000000;

Int32 R\_Gain = 2100000000;

/\* 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 = 100.0;

//Create table to store sine values

**float** table[SINE\_TABLE\_SIZE];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function prototypes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** init\_hardware(**void**);

**void** init\_sine(**void**);

**float** sinegen(**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main routine \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** main()

{

// initialize board and the audio port

init\_hardware();

// initialize the table of sine values

init\_sine();

// Loop endlessley generating a sine wave

**while**(1)

{

// Calculate next sample

sample = sinegen();

/\* Send a sample to the audio port if it is ready to transmit.

Note: DSK6713\_AIC23\_write() returns false if the port if is not ready \*/

// send to LEFT channel (poll until ready)

**while** (!DSK6713\_AIC23\_write(H\_Codec, ((Int32)(sample \* L\_Gain))))

{};

// send same sample to RIGHT channel (poll until ready)

**while** (!DSK6713\_AIC23\_write(H\_Codec, ((Int32)(sample \* R\_Gain))))

{};

// 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\_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);

/\* Set the sampling frequency of the audio port. Must only be set to a supported

frequency (8000/16000/24000/32000/44100/48000/96000) \*/

DSK6713\_AIC23\_setFreq(H\_Codec, get\_sampling\_handle(&sampling\_freq));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* sinegen() \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**float** sinegen(**void**)

{

// temporary variable used to output values from function

**float** wave;

// 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];

**return**(wave);

}