Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine

EE3T Study Project: Real-Time Digital Signal Processing with TMS320C6000

Laboratory 3 – Learning C and Sinewave Generation

Objectives

- Learn how to create and build a project.
- Learn how to use functions in C.
- Learn how to generate a sinewave using table lookup.

Map the Network Drive

1. Map `\\reserver\sp_data` to `m:`
2. Copy the files from `m:\C6x\labs\lab3` to an appropraite subdirectory in your workspace on `h:` for use. This should be the same directory in which you create the project below.

Create the project file

The first step in creating a new DSP program is to create a project file and to add appropriate configuration files. These configuration files automatically handle memory mapping and allow you to set various global parameters (in particular, we will use the configuration file in later labs to assign interrupts).

1. Create a new project called `lab3.mak`

   `Project:New`

   and save it in an appropriate subdirectory in your workspace on `h:`
2. Create a new DSP/BIOS configuration file called `lab3.cdb`

   `File:New:DSP/BIOS Configuration`

   using the `dsk6711.cdb` template. Save this file in the same subdirectory. Take care to use the correct extension, for some reason the default extension is a PowerPoint file!
3. Add the configuration file to the project.

   `Project:Add Files to Project`

   This also automatically adds the file `lab3cfg.s62`.
4. Also add the linker command file `lab3cfg.cmd`.

The sinewave generation program

You are provided with a simple program `sine.c` whose listing is given in Appendix A. This program does the following:

- Initializes I/O ports with calls to `CSL_Init`, `BSL_init`, and `codec_init`. This is done using predefined functions, we will look at these functions in more detail in a later lab.
- A while loop runs forever, writing out data calculated in the subroutine `sinegen()`. Currently this subroutine simply outputs a sine wave of fixed frequency. In the reminder of the lab you will write a routine that outputs a sine wave of a user-defined frequency.

1. Add the C program `sine.c` to the project.

Building the project

Building the project consists of three steps: compiler, assembler, linker. Each of these steps has various options which determine, for example, how efficient is the DSP code that is generated. Two `.txt` files are provided that contain appropriate compiler and linker options.

1. Set the compiler options

   `Project:Options`

   by copying the contents of `compiler.txt` to the Compiler window.

2. Set the assembler options to `-gls`

3. Set the linker options by copying the contents of `linker.txt` to the Linker window.

4. Now build the project

   `Project:Rebuild All`

   This can also be done using the appropriate icon on the left hand side. Once it has successfully compiled and linked, it will create a file `sine.out` that is the DSP executable file.

Load and run the program

Having built the program it can be downloaded to the DSP and run.

1. Load the program to the DSP

   `File:Load Program`

2. Connect an oscilloscope to the DSK output, and run the program

   `Debug:Run`

   or click the appropriate icon on the left. Ensure that the output is a sinewave.

3. Halt the program

   `Debug:Halt`

   or click the appropriate icon on the left.

Later when the program is modified and rebuilt it can be reloaded to the DSP by using `File:Reload Program`. Note that you should always halt the DSP before loading or reloading a program.

Generating a sine wave

Currently, the function `sinegen` generates a sinewave with fixed frequency of around 1 kHz. In the remainder of the lab you will rewrite this function so that the frequency that is output is determined by the global constant `SINE_FREQ`.

The simplest way of generating a sinewave signal is by using a sine lookup table. We will define a lookup table in memory, and then modify `sinegen` so that it uses this table to output a sinewave of the desired frequency.

1. Define a global constant `SIZE_TABLE` that determines how many values are in the table, and assign it a value of 256. (See the definition of `SAMPLE_FREQ` for how to declare global constants.)

2. Define a global variable `x` that is an array of floats containing `SIZE_TABLE` elements. This is done by inserting the following line in the section of the program designated as being for Global Variables:

   ```
   float x[SIZE_TABLE];
   ```

   Notice that the definition for `y` defines an array of 3 floats, and assigns some initial values. The above definition for `x` does not assign initial values.

   In C, arrays are indexed starting from 0, so an array of 10 values would be indexed 0 through 9. Individual elements of an array are accessed by using the variable name followed by the index in square brackets. For example, to assign the zeroth element of `x` to 0.2, you would use

   ```
   x[0] = 0.2;
   ```

3. Having defined how big the table is and where it will be stored in memory, we now need to create a function that will fill it with the appropriate sine values. Create a function `sine_init` that will do this. It will have no parameters and return no value, its purpose is simply to initialize the global variable `x`. This will involve defining the function prototype (look at the definition of `codec_init` at the top of `sine.c` for how to do this). Now you will need to write the implementation of this function. It should consist of a `for` loop (see C programming guide for syntax) that fills `x` with the appropriate number of points within one period of a sine wave. In the `math.h` library file (which is included in the shell program), there is a function `sin()` that returns the sine of its argument (in radians). A global constant `PI` is assigned a value of 3.1415926 (in the included file `c6211dsk.h`) for you to use.

4. Add a call to `sine_init` in the main program, just before the while loop. Rebuild the project and download it to the DSP. Set a breakpoint just after the call to `sine_init`, and run the program. To check that your function is successful, look at the memory starting at `x` (`View:Memory`) and confirm that 256 points within a single period of a sine wave are stored there. You could also do this by using the graph function (`View:Graph:Time/Frequency`). Ensure that `sine_init` is working correctly before proceeding.

5. Now rewrite the function `sinegen` so that it reads the appropriate value from the sine table and returns this value to the main routine. You will need to use the global constants `SIZE_TABLE` and `SAMPLE_FREQ` which have been pre-defined for you. You shouldn't need to modify the main program further at this stage, it is only necessary to rewrite the implementation of `sinegen`.

6. Since the frequency of the sine wave is determined by the global variable `SINE_FREQ`, you can easily change the output frequency by changing the value of this variable. Open a watch window (`View:Watch Window`) and insert `SINE_FREQ`. By double-clicking on `SINE_FREQ` within the watch window you will be able to update the frequency of the sine wave without requiring you to recompile the code.

Comments

Within the existing function `sinegen`, the variable `wave` is a *local* variable. Any variable that is defined within a function is local, and it is undefined outside that particular function. Moreover, when the function is exited the local variable loses its value. If it is necessary for a variable to keep its value even after a local function is exited, or from one function call to the next, it should be defined as a *global* variable, by inserting its definition before `main` (as you just did when defining `x`).

You will probably need to define one more variable as a global variable, and use it to keep track of where you are within the sine table from one call of `sinegen` to the next.

Appendix A

```
/*

SINE.C Sinewave generation lab

DBW 24/4/01

*/

/***************************************
* Function prototypes
***************************************/
void codec_init(void);
float sinegen(void);

/***************************************
* Include files
***************************************/
#include <c6x.h>/* C6000 compiler definitions header */
#include <c6211dsk.h>        /* C6000 DSK definitions header */
#include <csl.h>/* CSL headers */
#include <bsl.h>/* BSL headers */
#include <bsl_ad535.h>
#include <bsl_extra.h>
#include <mcbsp.h>              /* Multichannel serial port header */
#include <math.h>

/***************************************
* Definitions
***************************************/
#define SAMPLE_FREQ 8000.0

/***************************************
* Global variables
***************************************/
AD535_Handle hAD535;
AD535_Config my_AD535_Config = { AD535_LOOPBACK_DISABLE,
                          AD535_MICGAIN_OFF,
                          AD535_GAIN_0DB,
                          AD535_GAIN_0DB
                        };

float SINE_FREQ = 3000.0;

/* THE FOLLOWING DEFINITION IS ONLY NEEDED
   FOR THE SHELL PROGRAM, IT CAN BE REMOVED LATER */
float y[3] = {0,0.7071,0};

/***************************************
* Main routine
***************************************/
void main(){
```

```
  /* Local variables */
  float sample;

  /* Initialize the chip support library, required */
  CSL_Init();
  /* Initialize the board support library, required */
  BSL_init();
  /* Open and configure the local codec */
  codec_init();

  /* loop forever writing sinewave */
  while(1) {
  /* scale the output to use the full range
     and output it using the pre-defined function */

    sample = sinegen();
    AD535_write(hAD535,(int)(32000*sample));
  }
}

/***********************************
* Codec functions
***********************************/
void codec_init()
{
  /* Use BSL routines to open , reset, and configure the codec */
  hAD535 = AD535_open(AD535_localId);
  AD535_reset(hAD535);
  AD535_config(hAD535, &my_AD535_Config);
  MCBSP_setfree(0);
}

/***************************************
* Function implementations
***************************************/
float sinegen(void)
{

/*  YOU NEED TO REWRITE THIS FUNCTION */

  float wave;
  float A = 1.4142;

  y[0] = y[1] * A - y[2];
  y[2] = y[1];
  y[1] = y[0];

  wave = y[0];

  return(wave);
}
```