EEL319 Digital Signal Processing Lab

Experiment 1: Familiarization with Code Composer Studio (CCS)

Aim:

The overall aim is to get comfortable with the Code Composer Studio, learn to run and verify code given to you, and understand about the architecture of the processor.

At the end of the end of this experiment, you should be comfortable with the software, and be able to 1) start projects, and make relevant files for the same, 2) change memory location, and view in different Q formats, 3) make your own command file, 4) get online help on various instructions 5) run and single-step sample programs given to you, 6) set break-points, set watch, 7) view memory locations as a graph, 8) profile programs, 9) and appreciate the difference between assembly and C programming.

Procedure:

Part IA

We use a sample program to find the sum of the numbers in an array (placed in a table) to learn about CCS. . .

First create a new subdirectory for yourself in the user directory. Create a new project entitled expla in your directory using the "new" in the project menu (choose asm file).

Enter the following:

```
        .mmregs
        .def    x,y,init
x       .usect vars,10          ; declaring space for variables
y       .usect  vars,1
        .sect  table
init    .int    4000h,2000h,1000h,800h,400h,200h,100h,80h,40h,20h       ;declaring table, these
values need to be summed
        .text
        .def start
start                           ;label declaring start of code

        BCLR C54CM              ;setting into native c55 mode
        BCLR AR4LC          ;setting AR4 into linear mode
        BCLR AR6LC          ; setting AR6 into linear mode
        AMOV #x,XAR4        ;loading XAR4 with address of start of x ;
        AMOV #init,XAR6     ;loading XAR6 with start of table of values
        RPT #9                  ;copying10 size table into another array
        Mov *ar6+,*ar4+     ;both registers are incremented
        mov #0,ac0 || aMov #x,ar4 ; note the parallel instruction
        ; Previous 3 instructions are actually not necessary
        rpt #9              ; actual accumulation into ac0
        add *ar4+,ac0,ac0
        mov ac0,*(#y)
        idle
```

(In the above program, the indenting is important, and only labels should appear in the first column.

Also, you can mix UppER and LoWer case alphabets). Save the above in a file named expla.asm. Include in project by using "add files to project" in project menu.
Enter the following:

```
MEMORY
{
DARAM       : origin=000100h,length=8000h
SARAM       : origin=010000h, length=8000h
}
SECTIONS
{
        vars    :> DARAM
        table   :> SARAM
        .text   :> SARAM
}
```

and save in file exp1a.cmd. Add this file to project too. This file tells the simulator where the various quantities and variables in the program are to be stored. You must learn to make your own file, but can use the above as a template.

Go to the "build options" under the "project" menu, click on "linker" and enter "start" without. quotes under code entry point (this ensures that PC is set to "start" on entry).

Click on "options" then on "disassembly style", and select mnemonic. Click on "options" then on "customize", select "program load options", and select "load program on build". With this, the program will get loaded every time it is built, and will not require separate loading.

Build the project, load the program, and single step the code (see the menu to the left of the screen). You should see an arrow to the left of the main window indicating the PC value.

Keep a watch on the memory location (using view memory) as you single step. You can also view registers by licking on registers under the "view" menu. You can choose to view in any *format* you like (view the variable 'init' by clicking on view memory, and selecting init as the address in Q0 and Q15 formats – this you can do by right clicking and changing properties related to Q format). Also look at the various registers. You can change values in the array by clicking on them. It is also possible to attend fractional values directly when the right Q format is selected (there is no need to convert from fractional to hexadecimal values). Clearly, the processor only deals with second complement numbers (integers).

Note that sizes of all windows can be adjusted using the mouse. Also, you can place windows in convenient sizes by right clicking and choosing "float in main window".

You can choose to keep a watch on some special locations by clicking on "quick watch" under the "view" menu, and typing the variable name. You can reload program at any time.

Note that you can add breakpoints (see "debug" menu) or run to breakpoint or animate. You can simply right click on the line of code instead. Use this to run sections of code, to measure the number of clock cycles taken to execute that section, etc.

You can highlight any portion of the code, and press FI *for* on-line help (for example on instructions).

To find the number of clock cycles your program takes, set the PC to the desired address, you can use the "profiler" menu, set the clock to zero, enable clock, run the code and view it. You can either single step *or* run the program to the desired point.

You can also choose to view a graph of memory locations using the "graph" menu under the "view" menu. Carefully go through the options in the view graph menu, and note that you have to specify the start address ('init' for example) and the size of the array, as well as the precision used to represent the numbers (16 bit signed format in this case). You can adjust the way the graphs is plotted, and also plot the FFT of an array for example. This option is useful to quickly verify the performance of a code visually.

Single step through the program and write down changes execution of the instructions causes. Verify the answer in memory location "y". Note that the actual code that performs the sum of numbers in an array is only a few instructions long in assembly. The other instructions are added to illustrate the use of instructions, and are not all necessary.

CCS also allows use to save a portion of the memory in a .dat file that you can load before you run the program. For example, the array in "init" can instead be loaded before execution. Use the menu "Data" under "FILE" and choose to save or load.

You can tag a file to a memory location so that the output of a program can be placed in a file. Use the manuals provided to familiarize your self with the software (softcopies are available on all PCs, and hardcopies are available in the lab).

There are many other menus in the CCS. Do take some time to go through these.

Part IB

Start a new project by name exp I b. Write a C program for the same (summing the positive integers in an array) with the first two lines being:

```
#include <stdio.h>
# include <stdlib.h>
```

Save in a file exp1b.c. Add this file to the project. Add the same command file you used earlier to this project. Also include the rts55.lib file given to you. Compile and run the program, and verify. Check the number of clock cycles this program takes. What is the length of the code as compared to the code generated earlier?

In addition you can use a higher level “block level visual language” to “write” programs. Note that the size of the code generated will be larger, though the ‘code’ can be ‘written’ extremely fast (this software is also available in the lab).

Note: In future experiments, the above files can be used as templates to save time.
In case you do not complete the experiment, you are requested to download the software, and complete it after hours.