ECE4703 B Term 2008 Laboratory Assignment 4

Project Code and Report Due by 3:00pm 04-Dec-2008

The goals of this laboratory assignment are:

- to familiarize you with assembly language programming and code optimization on the TMS320C6713,
- to allow you to experimentally try various coding strategies to best use the pipeline and functional units available on the TMS320C6713,
- to reinforce your understanding of the profiling capabilities of CCS.

1 Problem Statement

There are often cases where C code is unable to achieve the required performance for a timing-critical application. In these cases, you have a couple options. You could use the optimization capabilities of the CCS C compiler to improve your code performance or you could optimize your code by writing critical functions in *hand-optimized assembly language*. While writing your own assembly code can be difficult for complicated algorithms, it is a reasonable approach for simple algorithms. Writing assembly language code also allows you to directly control the operation of the DSP and to get a much better feeling for the sensitivity of its performance to various factors including pipeline efficiency and functional unit efficiency.

In this lab assignment, you will modify your IIR filter code written for Laboratory Assignment 3 to improve its efficiency. Specifically, you will write two C-callable assembly language functions, `df2sos_sp` and `df2sos_dp`, to calculate the output of one second-order Direct Form II IIR filter section. The first function (`df2sos_sp`) will do this for the case when all math is performed in 32-bit single-precision floating point i.e. the inputs, intermediate values, filter coefficients, gains, and outputs are all `float` datatypes. The second function (`df2sos_dp`) will do this for the case when all math is performed in 64-bit double-precision floating point, i.e. the inputs, intermediate values, filter coefficients, gains, and outputs are all `double` datatypes. There is no fixed-point component to this assignment. There is also no Matlab simulation component to this assignment.

To get a head start on this assignment, it is recommended that you read Chapters 3 and 7 of your textbook.

1.1 Specific Requirements

You are required to write two C-callable assembly language functions that perform the same task: calculating the output of a second-order Direct Form II IIR filter section and updating the intermediate variable memory buffer. One function uses strictly single-precision floating point math and the other function uses strictly double-precision floating point math. Do not use linear assembly or inline assembly language. Your functions should be written entirely in standard TMS320C6x assembly language and you are permitted to use any valid commands and directives in the programming guide.

To evaluate your functions, it is recommended that you realize the same IIR filter as in lab assignment 2. This is an eighth order filter, hence your DF-II SOS function (either `df2sos_sp` or `df2sos_dp`) will be called from your C code four times. You can just create one project for this entire assignment and have a `#define` at the beginning of your C code to select single-precision or double-precision math. Make sure you provide enough comments at the top of your C code so that the grader knows how to test both the single-precision and the double-precision versions of your SOS code.

Your code should accept the following inputs (passed via the registers as described in your textbook and lecture notes):

- $x[n]$ (current input, cast as either `float` or `double`)
- *stage* (an `int` indicating the index of the current SOS)

As in lab assignment 3, you should store all filter coefficients and the DF-II intermediate buffers in global arrays (these will be declared as `float` or `double`). Your function should load the feedforward and feedback coefficients (also declared as `float` or `double`) into appropriate registers. In the process of computing the output $y[n]$, your our code should also modify/compute the intermediate values $u[n]$, $u[n-1]$, and $u[n-2]$ (either `float` or `double`) for the current stage. Your function should return the filter output $y[n]$ (either `float` or `double`).

As part of your assembly language programming, you are allowed to specify which instructions are to be grouped into one *execution packet* via the parallel bars || (see Section 3.8 of your textbook). You are also allowed to specify which functional unit should execute each command (recall the 8 functional units available in the TMS320C6713). You are encouraged to try various approaches to this problem to minimize the number of execution packets and the number of clock cycles required to execute the DFII-SOS function.

Two 20-point bonuses will be given in this assignment. The first bonus will go to the team that can compute the DFII-SOS output using all `float` variables in the least number of cycles while satisfying the I/O specifications and also accurately realizing the correct frequency response. The second bonus will go the team that can compute the DFII-SOS output using all `double` variables in the least number of cycles while satisfying the I/O specifications and also accurately realizing the correct frequency response. Both bonuses will be based on profiling results on the DFII-SOS function as measured by the grader. The DFII-SOS functions have to work correctly to be eligible for bonus points.

2 In Lab

You will work with the same lab partner as in the prior laboratory assignments. Please contact the instructor if your lab partner has dropped the course or if you have concerns about your lab partner's performance on the prior assignment.

3 Suggested Procedure for Software Design

1. Begin by reading up on assembly language programming for the TMS320C6x. There are many new instructions but you will probably want to familiarize yourself with instructions like MPYSP (multiply single precision floats) and MPYDP (multiply double precision floats). Your textbook is a good place to start but you will probably need to refer to the TMS320C6000 Programmer's Guide for the full details on certain instructions.

2. You may want to look at the assembly language produced by CCS for your DFII-SOS code from Laboratory Assignment 3. This may give you some ideas on the types of instructions you will need to realize your DFII-SOS function in assembly.

3. Get the `float` version of your filter working first. This one is easier to understand because all of the variables will fit into single registers.

4. Don't worry about parallelizing/optimizing your code in the beginning. Just get your function working correctly. You can set breakpoints in your assembly code and also view the contents of registers (via the "View" menu) to facilitate troubleshooting.

5. Once you have your assembly language DFII-SOS function working and you've fully tested it, think carefully about how to put instructions in parallel to maximize parallel processing (decrease the number of execution packets per fetch packet). Can you reorder instructions to avoid resource conflicts as well as avoid data and branch hazards?

   To help with optimization, it is highly recommended that you draw some flowcharts and dependency diagrams in this step. These should be included in your report.

6. Make sure your filter still works after optimization!

3.1 Specific Items to Discuss in Your Report

Your report should focus primarily on your approach to developing efficient assembly language code for DFII-SOS IIR filtering. There is a lot of room for analysis (pipeline usage, functional unit usage, data hazards, ...) and discussion here. You should discuss the profiling gains you were able to make with respect to the double-precision floating point C code you wrote in Laboratory Assignment 3. You should also discuss any differences in your profiling results between the `float` and `double` versions of your C-callable assembly language functions. Is there any difference in the frequency response between the `float` and `double` versions of your code?