ECE4703 B Term 2006 Project 4

Signoff due by 1:50pm 29-Nov-2006. Report due by 4:00pm 30-Nov-2006.

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 and benchmarking capabilities of CCS.

1 Problem Statement

There are many 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 assembly language for critical functions. 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 a C-callable assembly language function to calculate the output of one second-order Direct Form II IIR filter section. You can then call this function (repeatedly) from your main C code to compute the overall filter output of a system realized in Direct Form II Second Order Sections.

1.1 Specific Requirements

You are required to write a C-callable assembly language function to calculate the output of a second-order Direct Form II IIR filter section. Do not use linear assembly or inline assembly language. Your function should be written entirely in standard TMS320C6x assembly language and you are permitted to use any valid commands and directives in the programming guide. Your code should accept the following inputs (passed via the registers as described in your textbook and lecture notes):

- $x[n]$ (current input, 32 bit float)
- *stage* (an integer indicating the current SOS)

You are permitted to store all filter coefficients and the DF-II intermediate buffers in global arrays. Your code will need to load the feedforward and feedback coefficients into appropriate registers. In the process of computing the output $y[n]$, your our code will modify/compute the intermediate values $u[n]$, $u[n-1]$, and $u[n-2]$ (32 bit floats) for the current stage. Your function should return the filter output $y[n]$ (32 bit float).

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. The team that can compute the DFII-SOS output in the least number of cycles (while also satisfying the I/O specifications above) will receive a bonus on the signoff.

In the signoff, you will be required to demonstrate your assembly language code realizing the following filter:

1. Filter type: IIR elliptic bandstop
2. Filter realization structure: DFII-SOS
3. Coefficient quantization: single precision floating point
4. Intermediate result quantization: single precision floating point
5. Sampling rate: 44100Hz
6. First passband: 0-2500Hz, 0dB nominal gain, $\pm$0.5dB maximum deviation
7. First transition band: 2500-3500Hz.
8. Stop band: 3500-10500Hz, 50dB minimum suppression
9. Second transition band: 10500-12500Hz
10. Second passband: 12500-22050Hz, 0dB nominal gain, $\pm$0.5dB maximum deviation
11. Filter order: Your filter should end up being 5 second order sections.

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). 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. 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.

4. 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. Make sure everyone on the team can explain your final design at the signoff.

5. Make sure your filter still works after optimization!

4 Laboratory Report and Grading

See Laboratory Assignment 2.

4.1 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) and discussion here. Your results should discuss the profiling gains you were able to make with respect to the C code you wrote in Laboratory Assignment 3.