FIR Filtering: Exercise for TI TMS320C55x*

Thomas Shen
David Jun

Based on FIR Filtering: Exercise for TI TMS320C54x (ECE 320 specific)† by Mark Butala

This work is produced by The Connexions Project and licensed under the Creative Commons Attribution License ‡

Abstract

You will implement band-pass finite impulse-response (FIR) filters with time-domain processing.

1 Introduction

In this exercise, you will program in the DSP’s assembly language to create FIR filters. Begin by studying the assembly code for the basic FIR filter filtercode.asm¹. For help with circular addressing, view Addressing Modes for TI TMS320C55x².

¹Version 1.8: Feb 7, 2010 2:47 pm US/Central
²http://cnx.org/content/m14262/latest/
¹http://cnx.org/content/m11020/2.6/
²http://creativecommons.org/licenses/by/2.0/
¹See the file at <http://cnx.org/content/m13791/latest/filtercode.asm>
²"Addressing Modes for TI TMS320C55x" <http://cnx.org/content/m14262/latest/>
filtercode.asm

.ARMS_off ; enable assembler for ARMS=0
.CPL_on ; enable assembler for CPL=1
.mmregs ; enable mem mapped register names

.global _filter
.global _inPtr
.global _outPtr

.copy "macro.asm" ; Copy in macro declaration

.sect ".data"

FIR_len1 .set 8 ; This is a 8-tap filter

.align 32 ; Align to a multiple of 16
coeff1 ; assign label "coeff1"
.copy "coef.asm" ; Copy in coefficients

.align 32
inputBuffer .space 16*FIR_len1 ; Allocate 8 words of storage for filter state

new_sample_index ; Allocate storage to save index in inputBuffer
.word 0

.copy "testvect.asm"

.sect ".text2"

_filter

ENTER_ASM ; Call macro. Prepares registers for assembly

MOV #0, AC0 ; Clears AC0 and XAR3
MOV AC0, XAR3 ; XAR3 needs to be cleared due to a bug

MOV dbl (*(#_inPtr)), XAR6 ; XAR6 contains address to input
MOV dbl (*(#_outPtr)), XAR7 ; AR7 contains address to output

BSET AR2LC ; sets circular addressing for AR2

MOV #inputBuffer, AR2 ; State pointer is in AR2
MOV mmap(AR2), BSA23 ; BSA23 contains address of inputBuffer
MOV #new_sample_index, AR4 ; State index pointer is in AR4
MOV *AR4, AR2 ; AR2 contains the index of oldest state

MOV #coeff1, AR1 ; initialize coefficient pointer
MOV #FIR_len1, BK03 ; initialize circular buffer length for register 0-3

MOV *AR6+ ≪ #16, AC0 ; Receive ch1 into AC0 accumulator
MOV AC0, AC1 ; Transfer AC0 into AC1 for safekeeping

MOV HI(AC0), *AR2+ ; store current input into state buffer
MOV #0, AC0 ; Clear AC0
filtercode.asm applies an FIR filter to the signal from input channel 1 and sends the resulting output to output channel 1. It also sends the original signal to output channel 2.

First, create a work directory on your network drive for the files in this exercise, and copy the filter folder from v:\ece420\55x\ccs4\filter to your work directory. Then, use MATLAB to generate two 20-tap FIR filters. The first filter should pass signals from 4 kHz to 8 kHz; the second filter should pass from 8 kHz to 12 kHz. For both filters, allow a 1 kHz transition band on each edge of the filter passband. To create these filters, first convert these band edges to digital frequencies based on the 48 kHz sample rate of the system, then use the MATLAB command firpm to generate this filter; you can type help firpm for more information. Use the save_coef command to save each of these filters into different files. (Make sure you reverse the vectors of filter coefficients before you save them.) Also save your filters as a MATLAB matrix, since you will need them later to generate test vectors. This can be done using the MATLAB save command. Once this is done, use the freqz command to plot the frequency response of each filter.

2 Part 1: Single-Channel FIR Filter

For now, you will implement only the filter with a 4 kHz to 8 kHz passband. Edit filtercode.asm to use the coefficients for this filter by making several changes.

First, the length of the FIR filter for this exercise is 20, not 8. Therefore, you need to change FIR_len1 to 20. FIR_len1 is set using the .set directive, which assigns a number to a symbolic name. You will need to change this to FIR_len1 .set 20.

Second, you will need to ensure that the .copy directive brings in the correct coefficients. Change the filename to point to the file that contains the coefficients for your first filter.

Third, you will need to modify the .align and .space directives appropriately. The TI TMS320C55x DSP requires that circular buffers, which are used for the FIR filter coefficient and state buffers, be aligned so that they begin at an address that is a multiple of a power of two greater than the length of the buffer. Since you are using a 20-tap filter (which uses 20-element state and coefficient buffers), the next greater power of two is 32. Therefore, you will need to align both the state and coefficient buffers to an address that is a multiple of 32. (16-element buffers would also require alignment to a multiple of 32.) This is done with the .align command. In addition, memory must be reserved for the state buffer. This is done using the .space directive, which takes as its input the number of bits of space to allocate. Therefore, to allocate 20 words of storage, use the directive .space 16*20 as shown below:

1 .align 32 % Align to a multiple of 32
2 coef1 .copy "coef1.asm" % Copy FIR filter coefficients
3
4 .align 32 % Align to a multiple of 32
5 firState1 .space 16*20 % Allocate 20 words of data space

Assemble your code, load the output file, and run. Ensure that it is has the correct frequency response. After you have verified that this code works properly, proceed to the next step.

3 Part 2: Dual-Channel FIR Filters

First, make a copy of your modified filtercode.asm file from Part 1 (Section 2: Part 1: Single-Channel FIR Filter). Work from this copy; do not modify your working filter from the previous part. You will use that code again later.

Next, modify your code so that in addition to sending the output of your first filter (with a 4 kHz to 8 kHz passband) to output channel 1 and the unfiltered input to output channel 2, it sends the output of your second filter (with a 8 kHz to 12 kHz passband) to output channel 3. To do this, you will need to use the .align and .copy directives to load the second set of coefficients into data memory. You will also need to
add instructions to initialize a pointer to the second set of coefficients and to perform the calculations for the second filter.

**Exercise 1**

**Extra Credit Problem**

One extra credit point will be awarded to you and your partner if you can implement the dual-channel system without using the auxiliary registers AR0, AR3 and AR5. Why is this more difficult?

Renaming the registers using the .asg directive does not count!

Using the techniques introduced in DSP Development Environment: Introductory Exercise for TI TMS320C55x, generate an appropriate test vector and expected outputs in MATLAB. Then, using the test-vector core file also introduced in DSP Development Environment: Introductory Exercise for TI TMS320C55x, find the system’s output given this test vector. In MATLAB, plot the expected and actual outputs of the both filters and the difference between the expected and actual outputs. Why is the output from the DSP system not exactly the same as the output from MATLAB?

4 Part 3: Alternative Single-Channel FIR Implementation

An alternative method of implementing symmetric FIR filters uses the firsadd instruction. Modify your code from Part 1 (Section 2: Part 1: Single-Channel FIR Filter) to implement the filter with a 4 kHz to 8 kHz passband using the firsadd.

Two differences in implementation between your code from Part 1 (Section 2: Part 1: Single-Channel FIR Filter) and the code you will write for this part are that firsadd requires the states to be broken up into two separate circular buffers. Refer to the firsadd instruction on page 5-152 in the Mnemonic Instruction Set manual.

```
1 mov *AR1, *AR2- ; write x(-N/2) over x(-N)
2 mov HI(AC0), *AR1 ; write x(0) over x(-N/2)
3 add *AR1-, *AR2-, AC0 ; add x(0) and x(-(N-1))
4 ; (prepare for first multiply)
5 rpt #(FIR_len1/2-1)
6 firsadd *AR1-, *AR2-, *CDP+, AC0, AC1
7 round AC1
8 amar ????????????????? ; Fill in these two instructions
9 amar ??? ? ; They modify AR1 and AR2
10
11 ; AC1 accumulator
```

Because states and coefficients are now treated differently than in your previous FIR implementation, you will need to modify the pointer initializations to

```
1 bset AR1LC ; sets circular addressing for AR1
2 bset AR2LC ; sets circular addressing for AR2
3
4
5 mov #firState1, AR1
6 mov #firState1Index, AR4
```

\[\text{DSP Development Environment: Introductory Exercise for TI TMS320C55x}^3\] <http://cnx.org/content/m13811/latest/>

\[\text{DSP Development Environment: Introductory Exercise for TI TMS320C55x}^4\] <http://cnx.org/content/m13811/latest/>

\[\text{http://focus.ti.com/lit/ug/spnu374g/spnu374g.pdf}\]

http://cnx.org/content/m13791/1.8/
7    mov    mmap(AR1), BSA01
8    mov    *AR4, AR1      ; get pointer to oldest delayBuf in AR1
9    mov    #firState2, AR2
10   mov    #firState2Index, AR5
11   mov    mmap(AR2), BSA23
12   mov    *AR5, AR2
13
14   mov    #(FIR_len1/2), BKC
15   mov    #(FIR_len1/2), BK03     ; initialize circular buffer length for register 0-3
16   mov    #coef1, CDP          ; CDP contains address of coefficients
17   mov    *AR6 ≪ #16, AC0     ; copy input into AC0

There are also a couple other changes that need to be made before the code will compile successfully. Read
the comments carefully and understand how the \texttt{firsadd} instruction works to make the necessary changes.
Hint: Make sure accumulator usage (AC0, AC1, AC2) and what is sent to output is correct.

Use the test-vector core file to find the output of this system given the same test vector you used to test
the two-filter system. Compare the output of this code against the output of the same filter implemented
using the \texttt{mac} instruction. Are the results the same? Why or why not? Ensure that the filtered output is
sent to output channel 1, and that the unmodified output is still sent to output channel 2.

\textbf{WARNING:} You will lose credit if the unmodified output is not present or if the channels are reversed!

\section*{5 Quiz Information}

The quiz for Lab 1 is broken down as follows:

- 1 point: Prelab (must be ready to show the TA the week before the quiz)
- 4 points: Working code: you must demonstrate that your code works using input from function
generator and that it works using input from appropriate test vectors. Have an \texttt{.asm} file \textbf{ready} to
demonstrate each. Of the 4 points, you get 0.5 points for a single 20-tap filter, 2 points for the two-filter
system, and 1.5 points for the system using the \texttt{firs} opcode.
- 5 points: Oral quiz score.
- 1 extra credit point: As described above (p. 3).

The oral quiz may cover signal processing material relating to FIR filters, including, but not limited to,
the delay through FIR filters, generalized linear phase, and the differences between ideal FIR filters and
realizable FIR filters. You may also be asked questions about digital sampling theory, including, but not
limited to, the Nyquist sampling theorem and the relationship between the analog frequency spectrum and
the digital frequency spectrum of a continuous-time signal that has been sampled.

The oral quiz \textbf{will} cover the code that you have written during the lab. You are expected to understand,
in detail, all of the code in the files you have worked on, even if your partner or a TA wrote it. (You are not
expected to understand the core file in detail). The TA will ask you to explain various lines of code as part
of the quiz. The TAs may also ask questions about 2's complement fractional arithmetic, circular buffers,
alignment, and the mechanics of either of the two FIR filter implementations. You could be ready to trace
through any of the code on paper and explain what each line of code does.

Use the TI documentation, specifically the \textit{Mnemonic Instruction Set}\textsuperscript{6} manual. Also, feel free to ask the
TAs to help explain the code that you have been given.

\textsuperscript{6}http://focus.ti.com/lit/ug/spru374g/spru374g.pdf

http://cnx.org/content/m13791/1.8/