EEL319 Digital Signal Processing

Experiment 4: FIR Filter Implementation

(3 turns)

Aim:
The aim of this experiment is to familiarize you with the various possible instructions that can be used for FIR filtering and the C code that can be used.

Goals:
At the end of the experiment, you should be able to utilize the C5510 architecture efficiently for FIR filtering (assembly and C language programming).

Procedure:

1. An assembly language program to perform FIR filtering with non-symmetric impulse response using only one MAC is listed at the end of the experiment sheet. Assume all numbers are in Q15 format. Profile the above program. How many cycles does the program take to generate one output (initialization steps excluded)?
2. Repeat for the case when the impulse response is symmetric (assembly code). Use FIRSADD.
3. Implement the FIR assembly code as a subroutine, with each call generating one sample of the output.
4. Implement the general FIR filter (non-symmetric) in assembly optimally exploiting the dual MACs as described below. Profile the program.
5. Rewrite the general FIR filter program of part 1 in C language. Read up relevant material from the lab reference book, and the special notes on FIR filter implementation mailed to you.

Note: Your program for all parts should be general, and not restricted to any particular order (assuming odd or even length is okay in some cases).
All code that you write should be properly profiled – number of clock cycles taken to complete one loop should be indicated in the report. In the demo (on the day of submission of the report), I will be primarily interested in the number of clock cycles taken by your program to compute one output on the average assuming filter length is large.

Later, you will run this filter program in real-time using DSP hardware.

Notes on FIR Implementation:
For the first part, you will find MACM and MAC useful, among others.
For part 2, you should use the dual MACs efficiently by exploiting the structure in the convolution. In the case of symmetric impulse response, you can use instructions like:

FIRSADD, FIRSSUB (for asymmetric), MAC :: MAC etc

In the case when the impulse response is not symmetric (part 4), it is still possible to exploit the structure in the convolution itself. This is explained briefly below through an example.
Consider an LTI with only four non-zero impulse response coefficients. The idea can be generalized however. Consider the output of such a filter at various instants of time:

:

:

$$\begin{array}{llll}
y[n-2] = h_0x[n-2] & + h_1x[n-3] & + h_2x[n-4] & + h_3x[n-5] \\
y[n-1] = h_0x[n-1] & + h_1x[n-2] & + h_2x[n-3] & + h_3x[n-4] \\
y[n] \quad = h_0x[n] & + h_1x[n-1] & + h_2x[n-2] & + h_3x[n-3] \\
y[n+1] = h_0x[n+1] & + h_1x[n] & + h_2x[n-1] & + h_3x[n-2] \\
y[n+2] = h_0x[n+2] & + h_1x[n+1] & + h_2x[n] & + h_3x[n-1] \\
y[n+3] = h_0x[n+3] & + h_1x[n+2] & + h_2x[n+1] & + h_3x[n]
\end{array}$$

:

Consider the signal at two instants of time, say n and n+1. Notice that both these outputs require x[n], but multiply by two different coefficients (x[n] should be accessed through CDP?). The same is also true of x[n-2]. Suppose the partial sum of terms involving these two points is calculated and stored. That is, the terms $h_0x[n] + h_2x[n-2]$ and $h_1x[n] + h_3x[n-2]$ are calculated
Notice that in the previous instant of time, you would have calculated $h_1x[n-1] + h_3x[n-3]$ along with $h_0x[n-1] + h_2x[n-3]$ in a similar fashion.. You can then sum the partial sums to get the output at any instant of time. This algorithm is very well suited for C5510 processor.

You will find the following instructions useful:

MAC::MAC, MACMZ, MACM

Read up “Efficient implementation of FIR filters” (article number SPRA655 in the TI site). A sample FIR program (for part 1) to help you get started:

```
; This is  sample program but not efficient (takes N+2 cycles) notice that the dual MAC is NOT used; a simple FIR filtering program (offline computation)
; assume that filter coeff. and input is stored in Q15 format.
;before running, store coefficients and input at specified location
N               .set 8      ; no. of filter coefficients
M               .set 20 ;size of input
                .mmregs
                .def start
start
                BCLR C54CM
                BCLR AR0LC ; linear mode
                AMOV #010000h,XAR0 ;starting address for input
                BCLR AR1LC ; linear mode
                AMOV #010020h,XAR1; starting address for output
                BSET CDPLC; circular addressing
                BSET SATD; to saturate accumulator
                BSET FRCT; to shift acc. left by one bit
                AMOV #000100h,XCDP;starting address for coefficients
                MOV #0h,BSAC; offset for circular addressing od CDP
                MOV #N,BKC;size of block for circular addressing
                MOV #0h,AC0
                MOV #M,BRC0 ; initialization of local block repeat
                MOV #N,T0;offset to move pointer of AR0 back
                || RPTBLOCAL LOOP
                MPYM *AR0-,*CDP+,AC0  ; Why is the first one done separately? Read the MPYM carefully
                || RPT #N-3           ; and notice order of acc./mult. – helps to single-step program.
                MACM *AR0-,*CDP+,AC0
                MACM *(AR0+T0),*CDP+,AC0        ;Why is this done separately?
LOOP    MOV HI(AC0),*AR1+   ; storing output
                IDLE
```