EFFICIENT IMPLEMENTATION OF TAP DELAY LINE FILTER USING HIGH SPEED DIGITAL SIGNAL PROCESSOR

Muhammad Imran Akram and Asrar U.H. Sheikh

King Fahd University of Petroleum and Minerals
Department of Electrical Engineering
Dhahran, Saudi Arabia 31261
Email: miakram, asrarhaq@kfupm.edu.sa

ABSTRACT
An efficient implementation of the linear Finite Impulse Response (FIR) Filter has been performed over the Texas Instrument (TI) TMS320C6416 fixed point Digital Signal Processor (DSP) platform. The implementation fully exploits the pipelined architecture of the processor along with the circular buffering to gain the speed factor of 7 times than the reference approach hence making this more suitable for high speed real-time signal processing applications involving tap delay line (TDL) model.

1. INTRODUCTION
Filtering is a basic building block which is used quite often in many digital signal processing applications. The applications include audio, video processing, computer vision, telecommunications etc. Mathematically, the output of the system (filter) is the convolution of input signal with the impulse response of the system. The filter block is modeled as a Tap Delay Line (TDL).

With an increase in signal bandwidth, the data rate is increased and the processing time per sample is reduced. Hence a faster processor is required to process the required data in real time. With the computation power requirement goes beyond the capabilities of general microprocessors, specialized digital signal processors that provide a set of instructions and registers to perform the signal processing tasks in efficient manner were introduced so that the real time application requirements are met. These also provide portability, low cost solution and low power consumption and are suitable for portable devices like mobile phones and PDAs.

Unlike general purpose microprocessors, DSP (digital signal processor) provides various assembly instructions and registers designed specially for real time signal processing. When using such these resources optimally, DSP can guarantee higher performance than other processors that have the same clock speed in signal processing computation. DSP also allows high speed convolution computation using addressing mode registers (AMR).

TI DSP TMS320C6416 provides an efficient and high speed implementation platform for TDL filter as it offers a circular addressing mode as part of its hardware features. The basic implementation of TDL filter is very simple and is given in [1]. A lot of work has been done on optimization of the filter and several algorithms exist [2]-[4],[6],[7]. Most of the optimization work has been done for real taps filter. Floating point TI DSP TMS320C67X, having similar architecture as C64X DSP, has been used for the optimization of complex taps TDL filter in [5].

This paper uses TMS320C6416 DSP for the high speed implementation of TDL filter, which can be used in high speed real time DSP applications for wideband signals in communication systems. The implementation differs from the conventional one as the tap spacing is non uniform and the tap interval includes multiple number of samples. It uses the circular addressing support by the TI processor to implement the TDL filter optimization. Although the code mentioned in [5] is efficient as compared to the conventional C programming, but it does not utilize fully the DSP capabilities in optimal way. The pipelined architecture along with the circular addressing mode in DSP has been used for the implementation which is efficient in terms of number of CPU cycles than the one mentioned in [5]. The output and computational efficiencies of floating point MATLAB code, fixed point pipelined assembly code have also been compared.

The remaining part of the paper is organized as follows. Section 2 explains the application of the concept of the circular buffering in tapped delay line filtering. Section 3 explains the brief overview of TMS320C6416 DSP architecture. In Section 4, implementation of TDL filter is described along with its performance analysis has been shown. Section 5 shows comparison of the proposed implementation with the existing one and also compares the output of fixed point code with MATLAB output. Finally, section 6 makes some concluding remarks.

2. CIRCULAR CONVOLUTION IN TDL FILTERING
Tapped Delay Line Filter is the basic block of many digital signal processing applications. It is based on the following
Functional Units (FU) which include .M1, .M2) and filter coefficient respectively at of two register files registers of length 32 bit and equally divided into groups instructions per cycle. It consits of 64 general purpose time signal processing. The CPU can execute eight

To implement an algorithm over a DSP, its architecture must be known. TMS320C6416 has the Very long Instruction Word VLIW architecture designed specially for real time signal processing. The CPU can execute eight 32 bit instructions per cycle. It consists of 64 general purpose registers of length 32 bit and equally divided into groups of two register files Ak:B. In addition to that, there are 8 Functional Units (FU) which include 2 multipliers (.M1, .M2) and 6 Arithmetic and Logic Units ALU (.L1, .L2, .S1, .S2, .D1, .D2). They all can runs independently in a single cycle.

The instruction sets are divided among these units each instruction can be performed by specific units. Shift operations can only be performed by .S units, for load and store operations .D units are used, for multiplications .M units and additions can be performed by .L, .S, .D units.

The CPU also has internal memory SRAM of 1 MB for the storage of program and data a part of this memory is used as Cache memory to enhance the program efficiency. In addition to that, the CPU also includes 3 Timer units [14], Control registers for setting CPU modes (like Address mode register AMR, Interrupt register IHR etc.), Enhanced Direct Memory Access (EDMA) unit for efficient data I/O operations [13], 3 Multiple Channel Buffered Serial Ports (MCBSP) [15], Multiple Channel Audio Serial Port (MCASP) [16]. Reference [9] gives the detail of the CPU instruction set and architecture.

4. TDL FILTERING IMPLEMENTATION

TDL Filter can be implemented in several ways depending upon the application. Here, the filter is modelled as a frequency selective channel, where the taps are assumed as multipath fingers located at multiple of sampling time \( T_s \). The \( N = M + 1 \) tap time intervals are assumed as \( \tau_i \) where \( i \) varies from 0 to \( M \) and \( \tau_0 = 0 \). The channel model is shown in Fig. 2. The tap index \( N_i \) is related with \( \tau_i \) as

\[
N_i = \left\lfloor \frac{\tau_i}{T_s} + 0.5 \right\rfloor
\]

where \( \lfloor . \rfloor \) indicates the truncation operation. For the buffer size \( L \), the maximum excess delay that last finger can be computed as

\[
\tau_{max} = LT_s
\]

3. OVERVIEW OF TMS320C6416 DSP ARCHITECTURE

To implement an algorithm over a DSP, its architecture must be known. TMS320C6416 has the Very long Instruction Word VLIW architecture designed specially for real time signal processing. The CPU can execute eight 32 bit instructions per cycle. It consists of 64 general purpose registers of length 32 bit and equally divided into groups of two register files Ak:B. In addition to that, there are 8 Functional Units (FU) which include 2 multipliers (.M1, .M2) and 6 Arithmetic and Logic Units ALU (.L1, .L2, .S1, .S2, .D1, .D2). They all can runs independently in a single cycle.

The instruction sets are divided among these units each instruction can be performed by specific units. Shift operations can only be performed by .S units, for load and store operations .D units are used, for multiplications .M units and additions can be performed by .L, .S, .D units.

The CPU also has internal memory SRAM of 1 MB for the storage of program and data a part of this memory is used as Cache memory to enhance the program efficiency. In addition to that, the CPU also includes 3 Timer units [14], Control registers for setting CPU modes (like Address mode register AMR, Interrupt register IHR etc.), Enhanced Direct Memory Access (EDMA) unit for efficient data I/O operations [13], 3 Multiple Channel Buffered Serial Ports (MCBSP) [15], Multiple Channel Audio Serial Port (MCASP) [16]. Reference [9] gives the detail of the CPU instruction set and architecture.

4. TDL FILTERING IMPLEMENTATION

TDL Filter can be implemented in several ways depending upon the application. Here, the filter is modelled as a frequency selective channel, where the taps are assumed as multipath fingers located at multiple of sampling time \( T_s \). The \( N = M + 1 \) tap time intervals are assumed as \( \tau_i \) where \( i \) varies from 0 to \( M \) and \( \tau_0 = 0 \). The channel model is shown in Fig. 2. The tap index \( N_i \) is related with \( \tau_i \) as

\[
N_i = \left\lfloor \frac{\tau_i}{T_s} + 0.5 \right\rfloor
\]

where \( \lfloor . \rfloor \) indicates the truncation operation. For the buffer size \( L \), the maximum excess delay that last finger can be computed as

\[
\tau_{max} = LT_s
\]
and $q$ is the number of bits representing mantissa. The signed integer part will have range $(-2^{q-1} - 1$ to $2^{q-1} - 1$)

The input data $x[n]$ is assumed complex having both Inphase $I$ and Quadrature $Q$ components in Q1.15 signed format. The channel coefficients $h[n]$ are complex with both components in Q3.13 signed format. The resulting complex output data $y[n]$ having both (I and Q) are in Q3.13 format. The input and output buffers are 32 bits (16 bit I and 16 bit Q) linear buffers having size $L$ where $\log_2 L$ is an integer. The C code is shown in Table 1. The $I$ and $Q$ components are placed near to each other in memory such that right from the starting location of buffer every 32 bit memory location consists of the lower addressed 16 bits $I$ and upper addressed 16 bits $Q$. The same holds true for the channel coefficients $h[n]$ as well.

### 4.2. Optimization of the Code for TMS320C6416

The C code was then converted into Assembly language code to optimize computation. The pipelined architecture of TMS320C6416 processor mentioned in [9] has been fully utilized for this purpose. For two complex number $X$ and $H$, the product $Y$ is obtained as,

\[ Y_I = X_I H_I - X_Q H_Q \]
\[ Y_Q = X_I H_Q + X_Q H_I \]

This implies that to obtain complex output, each complex multiplication consists of 4 real multiplications, one addition and one subtraction. Ideally, if $x[n]$ and $h[n]$ are already loaded in the registers and both multiplier units operate in parallel the results of these complex multiplications must be obtained in 5 cycles as shown in Table 2. For $N$ complex taps, the number of real multiplication required will be $5N$ and the results will be obtained in $5N$ cycles. Table 2 shows the complex multiplication and accumulation.

### Using the pipelining approach mentioned in [10] the code has been optimized for $N = 8$ taps. The inner loop was completely unrolled to reduce the loop overhead, the dependency graph was created and the instructions were pipelined to reduce the number of cycles. The optimized code consists of 3 parts. The **prolog**, the **mainloop** and **epilog**.

The prolog consists of initialization of local variables, pushing registers over stack for usage inside the function, loading taps coefficients $h[n]$ from memory into registers and defining input buffer as circular. Defining Input as Circular buffer removes the overhead of line 15 of Table 1. The use of circular buffer prevents the constant test of wrapping. The tap locations are loaded first and on the basis of that, filter coefficients are loaded. The tap locations and coefficients will remain constant for the complete $L$ sample buffer processing. The prolog is to be executed once for $L$ size input buffer. It takes 45 cycles to execute this code.

The main loop is also known as kernel of the program which is executed most of the time. It is optimized and instructions are scheduled to maximize the utilization of the CPU resources. For $N$ taps it is executed $2N$ times per input sample. For 4 taps, the resources allocations are shown in table 3. It is also shown that loading of data from memory and its storage into memory are done at the same time using .D1 and .D2 functional units. whereas branching instruction, re-initialization, output storing and counter increments have also been scheduled. $STH$ instructions have been used to store the sample output back into the internal memory, $ZERO$ to re-initialize the output registers back to zero for the computation of next sample output and $SHR$ to bring the output in the required Q3.13 format.

The epilog consists of the remaining part of the function. This include remaining loop portion, popping data back to the registers and branch out of the function. This part takes 42 cycles to execute.

<table>
<thead>
<tr>
<th><strong>Table 1. C Code of Tapped Delay Line Filter</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Table 2. Complex Multiplication and Accumulation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FU</strong></td>
</tr>
<tr>
<td>.M1</td>
</tr>
<tr>
<td>.M2</td>
</tr>
<tr>
<td>.L1</td>
</tr>
<tr>
<td>.L2</td>
</tr>
</tbody>
</table>

Using the pipelining approach mentioned in [10] the code has been optimized for $N = 8$ taps. The inner loop was completely unrolled to reduce the loop overhead, the dependency graph was created and the instructions were pipelined to reduce the number of cycles. The optimized code consists of 3 parts. The **prolog**, the **mainloop** and **epilog**.
Table 3: 4 Taps TDL Resources Allocation for main Loop

<table>
<thead>
<tr>
<th>Cy/FU</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
<td>M2</td>
<td>L1</td>
<td>L2</td>
<td>S1</td>
<td>S2</td>
<td>D1</td>
<td>D2</td>
</tr>
<tr>
<td>FU</td>
<td>MPY I4</td>
<td>MPY H Q4</td>
<td>SUB I3</td>
<td>ADD Q3</td>
<td>ADD</td>
<td>SUB</td>
<td>LDW</td>
<td>SUB</td>
</tr>
<tr>
<td></td>
<td>MPY I1</td>
<td>MPY H Q1</td>
<td>ADD I3</td>
<td>ADD Q4</td>
<td>B</td>
<td>ZERO</td>
<td>LDW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPY I2</td>
<td>MPY H Q2</td>
<td>SUB I4</td>
<td>ADD Q4</td>
<td>Q1</td>
<td>SHR</td>
<td>LDW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPY I3</td>
<td>MPY H Q3</td>
<td>ADD I4</td>
<td>ADD Q1</td>
<td></td>
<td></td>
<td>LDW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPY I4</td>
<td>MPY H Q4</td>
<td>ADD I2</td>
<td>ADD Q2</td>
<td></td>
<td></td>
<td>STH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ADD I1</td>
<td>ADD Q2</td>
<td></td>
<td></td>
<td>SUB</td>
<td></td>
</tr>
</tbody>
</table>

5. RESULTS AND COMPARISON

The TDL filter has been modeled here as a frequency selective multipath channel having $N$ taps. Each tap is complex having defined location. The channel is assumed static for one buffer period of time. Both C and assembly functions were executed for $L = 1024$ I/O buffer size, data sampling frequency $F_s = 12.5 MHz$ and $N = 8$ taps filter coefficients having values $h = \{1.8362 - 0.5073i, -0.2169 + 0.0915i, -0.1448 + 0.1585i, -0.2169 + 0.0915i, 0.0, 0.0, 0.0, 0.0\}$ with corresponding tap locations at $\{0, 0.5, 1.6, 1.8, 1.9, 2.0, 2.1\}$ usec. The magnitude response $|h|$ of the channel is shown in Fig. 3.

The sinusoidal input shown in Fig. 4 is applied on both I & Q channels. The output of both were found exactly match with each other. For fixed point C code, it takes 1460 cycles per sample to generate output whereas the optimized code gives output in 16 cycles per sample. From (3), the maximum excess delay $\tau_{max}$ the system can have is found to be 81.92 usec.

The comparison has also been made with an implementation given in [5]. The reference [5] uses C6713 DSP. The comparison is fair in terms of number of CPU cycles as both DSPs have same functional units. The only difference is in terms of number of general purpose registers. It takes 1.95 $\mu$sec to generate per sample output which on $225 MHz$ CPU takes 439 cycles whereas using the proposed optimization method, it takes only 62 cycles which is 7 times faster than the reference approach.

The comparison has also been made for a given complex input between the outputs of the MATLAB complex FIR filter code with the fixed point assembly code and are shown in Fig. 5 (Magnitude plot) and Fig. 6 (Phase plot). The number of taps assumed are $N = 8$ with buffer size $L = 1024$. The slight difference in both output is due to the truncation errors caused due to fixed point arithmetics. For C6416 DSP operating at $1 GHz$, the cycle time becomes $1 ns$ hence proposed algorithm will take around $16 ns$ per sample which means that data with around 60 MHz sampling frequency can be processed.

6. CONCLUSION

In this paper, an efficient implementation approach for Tapped Delay Line filter has been discussed. The fixed point C and assembly code for TNS320C6416 DSP has been developed and the results and comparison have been made with the existing implementation approach. The pipelined architecture of the processor and the circular buffer have been efficiently utilized. It has been found that the proposed approach performs 7 times better than the reference one. This makes the proposed implementation useful for real time high speed data applications in communication systems like wideband channel modeling and simulation.
Fig. 5. TDL Filter Output Magnitude plot

Fig. 6. TDL Filter Output Phase plot

7. REFERENCES


