CHAPTER 3

IMPLEMENTATION OF FIR FILTERS WITH MATLAB AND DSP PROCESSORS
3.1 Introduction

The word MATLAB stands for matrix laboratory. It is a matrix based software and it can be used for computation and visualization of scientific and engineering applications. Compared to other high level languages like Fortran and C, MATLAB is an easy programming tool especially for complex numerical problems. New functions and commands can be easily created using MATLAB. MATLAB can be implemented on several platforms like DOS, Win95, Win98, UNIX and Apple Macintosh etc. The basic MATLAB program is further enhanced by the availability of many tool boxes. Each tool box is a collection of specific topic related functions. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

Typical uses of MATLAB include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

3.2 The MATLAB System

The MATLAB system [1-3] consists of five main parts:
The MATLAB language

This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

The MATLAB working environment

This is the set of tools and facilities that one works with as the MATLAB user or programmer. It includes facilities for managing the variables in the workspace and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files, MATLAB's applications.

Handle Graphics

This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow one to fully customize the appearance of graphics as well as to build complete Graphical User Interfaces (GUI) on the specific MATLAB applications.

The MATLAB mathematical function library

This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigenvalues, Bessel functions, and fast Fourier transforms.
The MATLAB Application Program Interface (API)

This is a library that allows one to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

In the present work, the advantages of MATLAB are taken in order to calculate the coefficients of FIR filters. Following is the description and program development done successfully with MATLAB.

3.3 Finding FIR filter Coefficients using MATLAB

Following is the specifications of an FIR filter.

Type of filtering is lowpass; sampling frequency is 7200Hz; passband is 0-2 kHz; stopband is 2.4-3.6 kHz and stopband attenuation is 25 dB.

It is clear from Table 1.2 that the Barlett window can provide attenuation of 25dB. Hence Barlett window can be selected for the given specifications.

The following MATLAB program calculates the FIR filter coefficients.

```matlab
clear all;
ws=2000;
wp=2400;
fs=7200;
tr_wd=(wp-ws)*2*pi/fs;
% determining the order of the filter
M=ceil(5.6*pi/tr_wd);
n=[0:1:(M-1)];
% Calculation of corner frequency
wc=((ws+wp)/2)*2*pi/fs;
% finding the ideal impulse response
hd=sin(wc*(n-25))./(pi*(n-25));
hd(26)=wc/pi
% finding the Barlett window coefficients
for i=0 : (M-1)/2,
```
hn(i+1)=2*i/(M-l)
end
for i=(M-l)/2: M-l,
    hn(i+1)=2-2*i/(M-l)
end

% Calculating filter Coefficients
h=hd.*hn;
% plotting ideal impulse response
subplot(1,1,1)
subplot(2,2,1);stem(n,hd);title('Ideal Impulse Response');
axis([0 M-l -0.1 wc/pi+0.1]);xlabel('n');
ylabel('hd(n)');
% plotting Barlett window response
subplot(2,2,2);stem(n,hn);title('Window Response');
axis([0 M-l 0 1.1]);xlabel('n');
ylabel('hn(n)');
% plotting actual filter coefficients
subplot(2,2,3);stem(n,h);title('Actual Response');
axis([0 M-l -0.1 wc/pi+0.1]);xlabel('n');
ylabel('h(n)');

The output plots of the above program are shown in Figure 3.1.

The filter coefficients generated from the above program will be in floating point format. When these coefficients are to be used with fixed point DSP or in VHDL code these fractional numbers will be stored in Q15 format.

3.4 Q15 Format

Q-format is a number representation commonly used when performing operations on non-integer numbers. In a Q-format, the Q-number (15 in Q15) denotes how many bits are located to the right of the binary point. [4]

For example a 16-bit Q number has a assumed binary point, a sign bit, i integer bits, 15-i fractional bits as shown below:
Figure 3.1: Output of the MATLAB program to calculate filter coefficients
The decimal number equal to binary given is 2.625. This particular number is said to be represented in a Q8 format (8 fractional bits). Its range is between -128 and 127.996. The fractional accuracy of a Q8 number is about 0.004. It is more common to work entirely with fractions represented in a Q15 format or integers in Q0 format. This is especially true for DSP algorithms where multiply and accumulate operations are dominant.

The following MATLAB program converts the floating point numbers to Q15 Hex format.

```
clear all
f1=fopen('output.dat','w');
f2=fopen('input.dat','r');
%read the coefficients from input file and store it in an array q1
q1=fscanf(f2,'%f');
%conversion of each coefficient to Q15 HEX format
for k=1:length(q1)
    q=q1(k);
    if q>=0
        y=round(q*65535/2);
    else
        y=2^16-round(-q*65535/2)+1;
    end
    %store it in the output file
    fprintf(f1,'%3X
',y);
end
fclose(f1);
close x,q;
end
fclose(f1);
```

The contents of the input.dat and output.dat files are shown in Table 3.1.
Table 3.1: The FIR filter coefficients in real format and Q15 format

<table>
<thead>
<tr>
<th>“Input.dat” Coefficients in floating format</th>
<th>“output.dat” Coefficients in Q15 Hex format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>0.0005</td>
<td>0010</td>
</tr>
<tr>
<td>0.0002</td>
<td>0007</td>
</tr>
<tr>
<td>-0.0017</td>
<td>FFC9</td>
</tr>
<tr>
<td>0.0012</td>
<td>0027</td>
</tr>
<tr>
<td>0.0020</td>
<td>0042</td>
</tr>
<tr>
<td>-0.0038</td>
<td>FF84</td>
</tr>
<tr>
<td>-0.0000</td>
<td>0000</td>
</tr>
<tr>
<td>0.0056</td>
<td>00B7</td>
</tr>
<tr>
<td>-0.0046</td>
<td>FF6A</td>
</tr>
<tr>
<td>-0.0042</td>
<td>FF77</td>
</tr>
<tr>
<td>0.0099</td>
<td>0144</td>
</tr>
<tr>
<td>-0.0020</td>
<td>FFBF</td>
</tr>
<tr>
<td>-0.0119</td>
<td>FE7B</td>
</tr>
<tr>
<td>0.0124</td>
<td>0196</td>
</tr>
<tr>
<td>0.0065</td>
<td>00D5</td>
</tr>
<tr>
<td>-0.0226</td>
<td>FD1C</td>
</tr>
<tr>
<td>0.0093</td>
<td>0131</td>
</tr>
<tr>
<td>0.0251</td>
<td>0336</td>
</tr>
<tr>
<td>-0.0349</td>
<td>FB89</td>
</tr>
<tr>
<td>-0.0088</td>
<td>FEE1</td>
</tr>
<tr>
<td>0.0658</td>
<td>086C</td>
</tr>
<tr>
<td>-0.0467</td>
<td>FA07</td>
</tr>
<tr>
<td>-0.0941</td>
<td>F3F6</td>
</tr>
<tr>
<td>0.2871</td>
<td>24C0</td>
</tr>
<tr>
<td>0.6111</td>
<td>4E38</td>
</tr>
</tbody>
</table>
3.5 DSP Processors

DSP chips are special microprocessor devices having hardware architectures and instruction sets designed for high speed digital signal processing applications. Texas Instruments is one of the world's leading suppliers of DSP chips and over many years they have had remarkable success with their TMS320 family of DSP processors. In the present work FIR filters are implemented and tested using the DSP processors: TMS320C50 and TMS320C6711.

3.6 The DSP chip TMS320C50 [5]

The TMS320C50 is a 16 bit fixed point digital signal processor that combines the flexibility of a high speed controller with the numerical capability of an array processor, thereby offering an inexpensive alternative to multichip bit-slice processors. The highly paralleled architecture and efficient instruction set, provide speed and flexibility capable of executing 10MIPS. The TMS320C50 optimizes speed by implementing functions in hardware while other processors implement through microcode or software.

The TMS320C50 is the third generation digital signal processor in the TMS320 family. Its powerful instruction set, inherent flexibility, high speed number crunching capabilities, and innovative architecture have made this high performance, cost effective processor to be the ideal solution to many telecommunications, computer, commercial, industrial and military applications.

3.6.1 Key Features of TMS320C50

The key features of the Digital Signal Processor TMS320C50 are:
- 35/50 ns single cycle fixed point instruction execution time (28.6/20 MIPS)
- Upward source code compatible with all C1X and C2X devices
- RAM based memory operation
- 9K X 16 bit single cycle on-chip program/data RAM
- 2K X 16 bit single cycle on chip boot ROM
- 1056 X 16 bit dual access on chip data RAM
- 224K X 16 bit maximum addressable external memory space (64K program, 64K data, 64K I/O and 32K global)
- 32 bit arithmetic logic unit (ALU), 32 bit accumulator (ACC), and 32 bit accumulator buffer (ACCB)
- 16bit Parallel logic unit.
- 16 X 16 bit parallel multiplier with a 32 bit product capability.
- Single cycle multiply/accumulate instructions
- Eight auxiliary registers with a dedicated auxiliary register arithmetic unit for indirect addressing.
- Eleven context switch registers for storing strategic CPU controlled registers during an interrupt service routine.
- Eight level hardware stack.
- 0 to 16 bit left and right data barrel shifters and a 64 bit incremental data shifter.
- Two indirectly addressed circular buffers for circular addressing.
- Single instruction repeat and block repeat operations for program code.
- Block memory move instructions for better program/data management.
- Full duplex synchronous serial port for direct communication between the C5X and another serial device.
- Time-division multiple access (TDM) serial port
- Interval timer with period, control and counter registers for software stop, start and reset.
- 64K parallel I/O ports, 16 of which are memory mapped.
- Sixteen software programmable wait state generators for program, data and I/O memory spaces.

3.6.2 Architecture

A detailed architectural block diagram of TMS320C50 is shown in the Figure 3.2. The TMS320C50 utilizes a modified Harvard architecture for speed and flexibility. In a strict architecture, program and data memory are in two separate spaces, permitting a full overlap of instruction fetch and execution. The TMS320 family's modification of the Harvard architecture allows transfer between program and data spaces, thereby increasing the flexibility of the device. This modification permits coefficients stored in program memory to be read into the data RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate instructions and subroutines based on computed values.
Figure 3.2: Architecture diagram of TMS320C50
32-Bit Accumulator

The TMS320C50 contains a 32-bit ALU and accumulator for support of double precision, two’s complement arithmetic. The ALU is a general purpose arithmetic unit that operates on 16 bit words taken from the data RAM or derived from immediate instructions. In addition to the usual arithmetic instructions, the ALU can perform Boolean operations, providing the bit manipulation ability required of a high speed controller. The accumulator stores the value from the ALU and is often an input to the ALU. Its word length is 32-bit. The accumulator is divided into a high order word (bit 31 to 16) and a low order word (bit 15 to 0). Instructions are provided for storing and loading the high and lower order accumulator words to memory.

16 x 16 bit parallel Multiplier

The multiplier performs a 16 X 16-bit two’s complement multiplication with a 32-bit result in a single instruction cycle. The multiplier consists of three units: the T-register, P-register and multiplier array. The 16-bit T-register temporarily stores the multiplicand and the P-register stores the 32-bit product. Multiplier values either come from the data memory or derived immediately from the MPY instruction word. The fast on-chip multiplier allows the device to perform fundamental operations such as convolution, correlation and filtering. Two multiply accumulate instructions in the instruction set fully utilize the computational bandwidth of the multiplier, allowing both operands to be processed simultaneously.
Shifters

A 16-bit scaling shifter is available at the accumulator input. This shifter produces a left shift of 0 to 16 bits on the input data to accumulator.

TMS320C50 also contains a shifter at the accumulator output. This shifter provide a left shift of 0 to 7 on the data from either the ACCH or ACCL register.

In addition one shifter at the output of P-register, can shift the product by 1 or 4 bits left or 6 bits right before transferring the product to accumulator.

Data and Program Memory

Since the TMS320C50 uses Harvard architecture, data and program memory reside in two separate spaces. Additionally TMS320C50 has one more memory space called I/O memory space. The total memory capacity of TMS320C50 is 64KW each of program, data and I/O. The 64KW of data memory is divided into 512 pages with each page containing 128 words. Only one page can be active at a time. One data page selection is done by setting data page pointer. TMS320C50 has 1056 words of dual access on-chip data RAM and 9K words of single access Data/Program RAM. The 1056 words of on chip data memory is divided as three blocks B0, B1 and B2 of which B0 can be configured as program or data RAM.

Out of the 64KW of total program memory, TMS320C50 has 2K words of on chip program ROM.

The TMS320C50 offers two modes of operation defined by the state of the MC/MP pin, The microcomputer mode (MC/MP=1) or the microprocessors mode (MC/MP=0). In the microcomputer mode, on chip ROM is mapped into the memory
space with up to 2K words of memory available. In the microprocessor mode all 64K words of program memory are external.

**Interrupts and Subroutines**

The TMS320C50 has three external maskable user interrupts available for external devices that interrupts the processor.

The TMS320C50 contains an eight level hardware stack for saving the contents of the program counter during interrupts and subroutine calls. Instructions are available for saving the device's complete context. PUSH and POP instructions permit a level of nesting restricted only by the amount of available RAM.

**Serial Port**

A full duplex on chip serial port provides direct communication with serial devices such as codecs, serial A/D converters and other serial systems. The interface signals are compatible with codecs and many others serial devices with a minimum of external hardware.

**Input and Output**

The 16 bit parallel data bus can be utilized to perform I/O functions in two cycles. The I/O ports are addressed by the four LSBs on the address lines, allowing 16 input and 16 output ports. In addition, a polling input for bit test and jump operations (BIO) and three interrupt pins (INT0 to INT2) have incorporated for multitasking.
3.7 FIR Filter Implementation with TMS320C50

The experimental setup to implement FIR filters using TMS320C50 is shown in Figure 3.3. A typical DSP application requires at least an analog input and an analog output. The analog signal is then converted into a digital signal by an Analog to Digital Converter. This digital data is given to the processor and after calculations again this digital data is sent to a Digital to Analog converter. AD1674 is used for Analog to Digital Conversion and DAC 1220 is used to Digital to Analog conversion. Both are used in 12 bit resolution mode. ‘VI-Microsystems Micro-50LC’ DSP kit is used for the filter implementation. The input signal is fed from a signal generator with peak-to-peak amplitude of 1V. The resulting output waveform was observed on the CRO and the output peak-to-peak voltage is noted for various frequencies of input signal.

Specifications of a Lowpass filter

The experiment was conducted for a Lowpass filter with sampling frequency of 10 kHz, and the order of the filter was selected as 8 and 16.

3.7.1 Assembly language Program to implement the FIR filters using TMS320C50

```
.mmregs
.text
B START
TABLE:
    .word 096AH
    .word 0104CH
    .word 015F9H
    .word 0192EH
    .word 0192EH
    , word015F9H
    . word0104CH
    . word 096AH
```
Figure 3.3: The experimental setup to implement FIR filters using TMS320C50
* Moving the filter coefficients from program memory to data memory

START: MAR *, AR0
  LAR AR0, #0200H
  RPT #07H
  BLKP TABLE, *+
  SETC CNF

* read ADC data

ISR:
  LDP #0AH
  LACC #0
  SAACL 0
  OUT 0,05H
  IN 0,06H
  LAR AR7, #0
  MAR *, AR7

BACK: BANZ BACK, *-
  IN 0,04
  NOP
  NOP
  NOP
  MAR *, AR1
  LAR AR1, #0300H
  LACC 0
  AND #0FFFH
  SUB #800H
  SAACL *
  LAR AR1, #333H
  MPY #0
  ZAC
  RPT #07H
  MACD 0FF00H, *-
  APAC
  LAR AR1, #0300H
  SACCH *

END

; send an output signal to determine sampling frequency
; start ADC conversion

; send the result to DAC1
; send output pulse to find sampling frequency
The response obtained for 8 tap and 16 taps with the above program are shown in Table 3.2 and Table 3.3, respectively.

3.8 The DSP Chip TMS320C6711 [6]

The TMS320C67x DSPs (including the TMS320C6711, TMS320C6711B, TMS320C6711C devices) compose the floating-point DSP family in the TMS320C6000 DSP platform. The C6711, C6711B, and C6711C devices are based on the high-performance, very-long-instruction-word (VLIW) architecture developed by Texas Instruments (TI), making these DSPs an excellent choice for multichannel and multifunction applications.

With performance of up to 900 million floating-point operations per second (MFLOPS) at a clock rate of 150 MHz, the C6711/C6711B device offers cost-effective solutions to high-performance DSP programming challenges. The C6711/C6711B DSP possesses the operational flexibility of high-speed controllers and the numerical capability of array processors. This processor has 32 general-purpose registers of 32-bit word length and eight highly independent functional units. The eight functional units provide four floating-/fixed-point ALUs, two fixed-point ALUs, and two floating-/fixed-point multipliers. The C6711/C6711B can produce two MACs per cycle for a total of 300 MMACS.
### Table 3.2: Response of an 8-order FIR filter with TMS320C50

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Output (v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 Hz - 2 kHz</td>
<td>1.2</td>
</tr>
<tr>
<td>2.5 kHz</td>
<td>1</td>
</tr>
<tr>
<td>3.5 kHz</td>
<td>0.8</td>
</tr>
<tr>
<td>4.5 kHz</td>
<td>0.6</td>
</tr>
<tr>
<td>5.0 kHz</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 3.3: Response of an 16-order FIR filter with TMS320C50

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Output (v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upto 1 kHz</td>
<td>2.4</td>
</tr>
<tr>
<td>2.5 kHz</td>
<td>2.2</td>
</tr>
<tr>
<td>3.5 kHz</td>
<td>1.2</td>
</tr>
<tr>
<td>4.5 kHz</td>
<td>0.8</td>
</tr>
</tbody>
</table>
With performance of up to 1200 million floating-point operations per second (MFLOPS) at a clock rate of 200 MHz, the C6711C device also offers cost-effective solutions to high-performance DSP programming challenges. The C6711C DSP also possesses the operational flexibility of high-speed controllers and the numerical capability of array processors. This processor has 32 general-purpose registers of 32-bit word length and eight highly independent functional units. The eight functional units provide four floating-/fixed-point ALUs, two fixed-point ALUs, and two floating-/fixed-point multipliers. The C6711C can produce two MACs per cycle for a total of 400 MMACS.

The C6711/C6711B/C6711C DSPs also have application-specific hardware logic, on-chip memory, and additional on-chip peripherals. The C6711 / C6711B / C6711C uses a two-level cache-based architecture and has a powerful and diverse set of peripherals. The Level 1 program cache (L1P) is a 32-Kbit direct mapped cache and the Level 1 data cache (L1D) is a 32-Kbit 2-way set-associative cache. The Level 2 memory/cache (L2) consists of a 512-Kbit memory space that is shared between program and data space. L2 memory can be configured as mapped memory, cache, or combinations of the two. The peripheral set includes two multichannel buffered serial ports (McBSPs), two general-purpose timers, a host-port interface (HPI), and a glueless external memory interface (EMIF) capable of interfacing to SDRAM, SBSRAM and asynchronous peripherals.

The C6711/C6711B/C6711C has a complete set of development tools which includes: a new C compiler, an assembly optimizer to simplify programming and
scheduling, and a Windows debugger interface for visibility into source code execution.

**CPU (DSP core) description**

The CPU fetches advanced very-long instruction words (VLIW) (256 bits wide) to supply up to eight 32-bit instructions to the eight functional units during every clock cycle. VLIW architecture features controls by which all eight units do not have to be supplied with instructions if they are not ready to execute. The first bit of every 32-bit instruction determines if the next instruction belongs to the same execute packet as the previous instruction, or whether it should be executed in the following clock as a part of the next execute packet. Fetch packets are always 256 bits wide; however, the execute packets can vary in size. The variable-length execute packets are a key memory-saving feature, distinguishing the C67x CPU from other VLIW architectures.

The CPU features two sets of functional units. Each set contains four units and a register file. One set contains functional units .L1, .S1, .M1, and .D1; the other set contains units .D2, .M2, .S2, and .L2. The two register files each contain 16 32-bit registers for a total of 32 general-purpose registers. The two sets of functional units, along with two register files, compose sides A and B of the CPU (as shown in the functional block and CPU diagram Figure 3.4 and 3.5). The four functional units on each side of the CPU can freely share the 16 registers belonging to that side. Additionally, each side features a single data bus connected to all the registers on the other side, by which the two sets of functional units can access data from the register files on the opposite side. While register access by functional units on the same side
Figure 3.4: Functional block and CPU diagram
Figure 3.5: CPU data paths
of the CPU as the register file can service all the units in a single clock cycle, register access using the register file across the CPU supports one read and one write per cycle.

The C67x CPU executes all C62x instructions. In addition to C62x fixed-point instructions, the six out of eight functional units (L1, S1, M1, M2, S2, and L2) also execute floating-point instructions. The remaining two functional units (D1 and D2) also execute the new LDDW instruction which loads 64 bits per CPU side for a total of 128 bits per cycle. Another key feature of the C67x CPU is the load/store architecture, where all instructions operate on registers (as opposed to data in memory). Two sets of data-addressing units (D1 and D2) are responsible for all data transfers between the register files and the memory. The data address driven by the .D units allows data addresses generated from one register file to be used to load or store data to or from the other register file. The C67x CPU supports a variety of indirect addressing modes using either linear- or circular-addressing modes with 5- or 15-bit offsets. All instructions are conditional, and most can access any one of the 32 registers. Some registers, however, are singled out to support specific addressing or to hold the condition for conditional instructions (if the condition is not automatically "true"). The two .M functional units are dedicated for multipliers.

The two .S and .L functional units perform a general set of arithmetic, logical, and branch functions with results available every clock cycle. The processing flow begins when a 256-bit-wide instruction fetch packet is fetched from a program memory. The 32-bit instructions destined for the individual functional units are "linked" together by the logic "1" bits in the least significant bit (LSB) position of the
instructions. The instructions that are “chained” together for simultaneous execution (up to eight in total) compose an execute packet. A “0” in the LSB of an instruction breaks the chain, effectively placing the instructions that follow it in the next execute packet. If an execute packet crosses the fetch-packet boundary (256 bits wide), the assembler places it in the next fetch packet, while the remainder of the current fetch packet is padded with NOP instructions. The number of execute packets within a fetch packet can vary from one to eight. Execute packets are dispatched to their respective functional units at the rate of one per clock cycle and the next 256-bit fetch packet is not fetched until all the execute packets from the current fetch packet have been dispatched. After decoding, the instructions simultaneously drive all active functional units for a maximum execution rate of eight instructions every clock cycle. While most results are stored in 32-bit registers, they can be subsequently moved to memory as bytes or half-words as well. All load and store instructions are byte-, half-word, or word-addressable.

The program for implementation of FIR filter using TMS320C6711 DSP processor, written in C language is given in the following pages and the response obtained is given in Table 3.3. Figure 3.6 shows the TMS320C50 DSP processor module used in the present work. Figure 3.7 shows the complete experimental assembly.

Figure 3.8 shows the TMS320C6711 DSP processor module and Figure 3.9 shows the experimental assembly. The output obtained from the implementation for sinusoidal input is shown in Figure 3.10.
Figure 3.6: TMS320C50 DSP trainer kit
Figure 3.7: Experimental setup with TMS320C50
Figure 3.8: TMS320C6711 DSP processor module

Figure 3.9: Experimental setup with TMS320C6711
Figure 3.10: Filter output
FIR filter program executed with TMS320C6711

#include <stdio.h>
#include <c6x.h>
#include "c6211dsk.h"
#include "codec_poll.h"
#include "co.h"

int main()
{
    /* dsp and peripheral initialization */
    CSR=0x100;
    /* disable all interrupts */
    IER=1; /* disable all interrupts except NMI */
    ICR=0xffffffff; /* clear all pending interrupts */
    *(unsigned volatile int *)EMIF_GCR = 0x3300; /* EMIF global control */
    *(unsigned volatile int *)EMIF_CE0 = 0x30; /* EMIF C0 control */
    *(unsigned volatile int *)EMIF_CE1 = 0xffffffff03; /* EMIF C1 control, 8bit async */
    *(unsigned volatile int *)EMIF_SDCTRL = 0x07117000; /* EMIF SDRAM control */
    *(unsigned volatile int *)EMIF_SDRP = 0x61a; /* EMIF SDRAM refresh period */
    *(unsigned volatile int *)EMIF_SDEXT = 0x54519; /* EMIF SDRAM extension */

    mcbsp0_init();
    codec_playback();
    return(0);
}

void mcbsp0_init()
{
    /* set up McBSP0 */
    *(unsigned volatile int *)McBSP0_SPCR = 0; /* reset serial port */
    *(unsigned volatile int *)McBSP0_PCR = 0; /* set pin control reg. */
    *(unsigned volatile int *)McBSP0_RCR = 0x10040; /* set rx control reg. one 16 bit */
    *(unsigned volatile int *)McBSP0_XCR = 0x10040; /* set tx control reg. one 16 bit */
    *(unsigned volatile int *)McBSP0_DXR = 0;
    *(unsigned volatile int *)McBSP0_SPCR = 0x12001; /* setup SP control reg. */
}
void mcbspO_write(int out_data)
{
    int temp;
    temp = *(unsigned volatile int *)McBSPO_SPCR & 0x20000;
    while (temp == 0)
    {
        temp = *(unsigned volatile int *)McBSPO_SPCR & 0x20000;
    }
    *(unsigned volatile int *)McBSPO_DX = out_data;
}

int mcbspO_read()
{
    int temp;
    temp = *(unsigned volatile int *)McBSPO_SPCR & 0x2;
    while (temp == 0)
    {
        temp = *(unsigned volatile int *)McBSPO_SPCR & 0x2;
    }
    temp = *(unsigned volatile int *)McBSPO_DRR;
    return temp;
}

void codec_playback()
{
    int temp, i, z;
    double y, x[31];
    // set up control register 3 for S/W reset
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(1);
    mcbsp0_read();
    mcbsp0_write(0x0386);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    // set up control register 3 for mic input
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0306);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x2330);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
if((temp & 0xff) != 0x06)
{
#if PRINT
    printf("Error in setting up register 3.\n");
    exit(0);
#endif
}
// set up control register 4
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0400);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x2430);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xff) != 0x00) {
    #if PRINT
    printf("Error in setting up register 4.\n");
    exit(0);
    #endif
}
// set up control register 5
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0502);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0x2530);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xfe) != 0x2) {
    #if PRINT
    printf("Error in setting up register 5.\n");
    exit(0);
    #endif
}
while(1) /* play back about 5 minutes */ {
    x[0] = mcbsp0_read();
y = 0;
    for(i=0; i<11; i++)
        y = y + x[i] * h[i];
z = y;
z = z & 0xffff;
mcbsp0_write(z);
    for(i=9; i>0; i--)

{ 
    x[i+1] = x[i];
}

Table 3.4. Response of a 32-order FIR filter with TMS320C6711

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Output (v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upto 1.5 kHz</td>
<td>2.8</td>
</tr>
<tr>
<td>2.5</td>
<td>2.6</td>
</tr>
<tr>
<td>3.5</td>
<td>2</td>
</tr>
<tr>
<td>4.5</td>
<td>1</td>
</tr>
</tbody>
</table>
References

1. Matlab reference manual – Mathworks Ltd.


4.1 Introduction

This chapter is divided into two parts. The first part deals with the description of various types of FPGAs, generalized FPGA architecture and the architecture features of the FPGAs considered in the present work. The second part deals with the details and the decided advantages in using FPGA for DSP applications, comparison of FPGA based DSP applications with Programmable DSP processors. This chapter also encompasses the newly developed FIR filter structures and the VHDL / Verilog codes.

PART - A

4.2 Introduction to Programmable Logic Devices

The FPGA [1,2] is a type of programmable device. Programmable devices are a class of general-purpose chips that can be configured for a wide variety of applications. The first programmable device that achieved a widespread use was the PROM (Programmable Read-Only Memory). PROMs, a one-time programmable device comes in two basic versions: 1) The Mask-Programmable Chip, programmed only by the manufacturer, 2) The Field-Programmable Chip, which can be programmed by the end-user. The Field Programmable PROMs are divided developed into two types, the Erasable Programmable Read-Only Memory (EPROM) and the Electrically Erasable Programmable Read-Only Memory (EEPROM). The EEPROM has the advantage of being erasable and reprogramable many a times.

Another advancement that took place in this field, has led to the development of the Programmable Logic Device (PLD). These devices were constructed to
implement logic circuits. The internal architecture of a PLD include an array of AND gates connected to an array of OR gates. The PAL (Programmable Array Logic) is a commonly used PLD consisting of a programmable AND-plane followed by a fixed OR-plane. PALs come in both mask and field versions. The PAL was designed to be used in small logic circuits.

The Mask-Programmable Gate Array (MPGA) was developed to handle larger logic circuits. A common MPGA consists of rows of transistors that can be interconnected to implement desired logic circuits. User specified connects are available both within the rows and between the rows. This enabled implementation of basic logic gates and the ability to interconnect the gates. As the metal layers are defined at the manufacturer, significant time and cost are incurred in producing the run. In 1985, Xilinx Inc. introduced the FPGA (Field Programmable Gate Array). The salient feature of FPGA is that the interconnects between all the elements were designed to be user programmable.

4.2.1 Architecture of FPGA

An FPGA device features an architecture similar to that of a gate-array, with a matrix of logic cells surrounded by a periphery of I/O cells. A network of metal interconnects can be linked in an arbitrary manner by programmable switches to form the desired connections between logic or I/O cells. Each cell is highly programmable, and provides the flexibility that is needed to implement a wide range of circuit designs. FPGA memory is volatile (SRAM-Based). The FPGA loses its configuration when power is turned off and must be reprogrammed when power is reapplied. Xilinx offers several families of PROM devices as well as other FPGA programming
methods [3-5], through which automatic reloading takes place after the power position is retained.

There are four main categories of FPGAs currently commercially available: i) Symmetrical array, ii) Row-based, iii) Hierarchical PLD, and iv) Sea-of-gates. In all these FPGAs the interconnections and the modes of programming varies. Currently there are four technologies in use. They are: i) Static RAM cells, ii) Anti-fuse, iii) EPROM transistors, and iv) EEPROM transistors. Depending upon the application, one FPGA technology may have features desirable for that application.

i) Static RAM Technology

In the Static RAM FPGA, programmable connections are made using pass-transistors, transmission gates, or multiplexers that are controlled by SRAM cells. The advantage of this technology is that it allows fast in-circuit reconfiguration. The major disadvantage is that the chip size required by the RAM technology, is more.

ii) Anti-Fuse Technology

An anti-fuse resides in a high-impedance state, and can be programmed into low impedance or "fused" state. A less expensive than the RAM technology, this device is a one time programmable (OTP) device.

iii) EPROM / EEPROM Technology

This method is same as the one used in the EPROM memories. One advantage of this technology is that it can be reprogrammed without external storage of configuration; though the EPROM transistors cannot be re-programmed in-circuit.
4.2.2 The SRAM based FPGA

The Field-Programmable Gate Arrays (FPGAs) provide the benefits of custom CMOS VLSI, and avoids the initial cost, time delay, and inherent risk of a conventional masked gate array. The FPGAs are customized by loading configuration data into the internal memory cells. The FPGA can either actively read its configuration data out of external serial or byte-parallel PROM (master mode), or the configuration data can be written into the FPGA (slave and peripheral mode). The FPGA can be programmed number of times without any limitation.

The FPGA has three major configurable elements as shown in Figure 4.1. They are: Configurable Logic Blocks (CLB), Input/Output Blocks (IOB), and interconnects. The CLBs provide the functional elements for constructing user's logic. The IOBs provide the interface between the package pins and internal signal lines. The programmable interconnect resources provide routing paths to connect the inputs and outputs of the CLBs and IOBs onto the appropriate networks. Customized configuration is established by programming internal static memory cells that determine the logic functions and internal connections implemented in the FPGA.

Figure 4.2 depicts a FPGA with a two-dimensional array of logic blocks that can be interconnected by interconnect wires. All internal connections are composed of metal segments with programmable switching points to implement the desired routing. An abundance of different routing resources is provided to achieve efficient automated routing. There are four main types of interconnect. These are distinguished by the relative length of their segments: single-length lines, double-length lines and
Figure 4.1: FPGA block diagram
Figure 4.2: Array of CLBs and interconnect wires
Longlines. In addition there will be global buffers which drive the signals fast with low skew nets. On these lines are transmitted: clocks and control signals.

The internal CLB elements are shown in Figure 4.3. Each CLB contains a pair of flip-flops and two independent 4-input function generators. These function generators have a good deal of flexibility, as most combinatorial logic functions need less than four inputs. It is the Configurable Logic Blocks which implement most of the logic in an FPGA. The flexibility and symmetry of the CLB architecture facilitates the placement and routing of a given application.

The above described is the general architectural features of FPGA. The architectural features of the FPGAs used in the present work is described in the following sections.

4.3 Xilinx FPGAs:

4.3.1 Architecture of Xilinx FPGA – Virtex-II [6]

The Virtex-II family is a platform FPGA suitable for high performance from low density to high density designs that are based on Intellectual Property (IP) cores and customized modules. Virtex-II can be used in applications and be design areas of telecommunication, wireless, networking, video and DSP. The Virtex-II architecture is optimized for high speed with low power consumption. Combining a wide variety of flexible features and a large range of densities up to 10 million system gates, the Virtex-II family enhances programmable logic design capabilities and is a powerful alternative to mask-programmed gates arrays. The architecture of Virtex-II shown in Figure 4.4.
Figure 4.3: Internal diagram of a Configurable Logic Block
Programmable I/Os
DCM DCM
CLB Block SelectRAM Multiplier
Figure 4.4: Virtex II Architecture Diagram
The internal configurable logic includes four major elements organized in a regular array.

- Configurable Logic Blocks (CLBs) provide functional elements for combinatorial and synchronous logic, including basic storage elements. BUFTs (3-state buffers) associated with each CLB element drive dedicated segmentable horizontal routing resources.

- Block SelectRAM memory modules provide large 18 Kbit storage elements of dual-port RAM.

- 18-bit x 18-bit dedicated multiplier blocks

- DCM (Digital Clock Manager) blocks that provide self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division and coarse- and fine-grained clock phase shifting.

The general routing matrix (GRM) is an array of routing switches. Each programmable element is tied to a switch matrix, allowing multiple connections to the general routing matrix. The overall programmable interconnection is hierarchical and designed to support high-speed designs. All programmable elements, including the routing resources, are controlled by values stored in static memory cells. These values are loaded in the memory cells during configuration and can be reloaded to change the functions of the programmable elements.
Virtex-II Features

Input/Output Blocks (IOBs)

IOBs are programmable and can be categorized as follows:

- Input block with an optional single-data-rate or double-data-rate (DDR) register
- Output block with an optional single-data-rate or DDR register, and an optional 3-state buffer, to be driven directly or through a single or DDR register
- Bidirectional block (any combination of input and output configurations)

These registers are either edge-triggered D-type flip-flops or level-sensitive latches. IOBs support the following single-ended I/O standards:

- LVTTL, LVCMOS (3.3V, 2.5V, 1.8V, and 1.5V)
- PCI-X compatible (133 MHz and 66 MHz) at 3.3V
- PCI compliant (66 MHz and 33 MHz) at 3.3V
- CardBus compliant (33 MHz) at 3.3V

The digitally controlled impedance (DCI) I/O feature automatically provides on-chip termination for each I/O element. The IOB elements also support the following differential signaling I/O standards:

- LVDS
- BLVDS (Bus LVDS)
- ULVDS
- LDT
- LVPECL
Two adjacent pads are used for each differential pair. Two or four IOB blocks connect to one switch matrix to access the routing resources.

**Configurable Logic Blocks (CLBs)**

CLB resources include four slices and two 3-state buffers.

Each slice is equivalent and contains:

- Two function generators (F & G)
- Two storage elements
- Arithmetic logic gates
- Large multiplexers
- Wide function capability
- Fast carry look-ahead chain
- Horizontal cascade chain (OR gate)

The function generators F & G are configurable as 4-input look-up tables (LUTs), as 16-bit shift registers, or as 16-bit distributed SelectRAM memory. In addition, the two storage elements are either edge-triggered D-type flip-flops or level-sensitive latches. Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources

**Block SelectRAM Memory**

The block SelectRAM memory resources are 18 Kb of dual-port RAM, programmable from 16K x 1 bit to 512 x 36 bits, in various depth and width configurations. Each port is totally synchronous and independent, offering three
"read-during-write" modes. Block SelectRAM memory is cascadable to implement large embedded storage blocks. Supported memory configurations for dual-port and single- A multiplier block is associated with each SelectRAM memory block. The multiplier block is a dedicated 18 x 18-bit multiplier and is optimized for operations based on the block Select RAM content on one port. The 18 x 18 multiplier can be used independently of the block SelectRAM resource. Read/multiply/accumulate operations and DSP filter structures are extremely efficient. Both the SelectRAM memory and the multiplier resource are connected to four switch matrices to access the general routing resources.

Global Clocking

The DCM and global clock multiplexer buffers provide a complete solution for designing high-speed clocking schemes. Up to 12 DCM blocks are available. To generate de-skewed internal or external clocks, each DCM can be used to eliminate clock distribution delay. The DCM also provides $90^\circ, 180^\circ$, and $270^\circ$ phase-shifted versions of its output clocks.

Virtex-II devices have 16 global clock MUX buffers, with up to eight clock nets per quadrant. Each global clock MUX buffer can select one of the two clock inputs and switch glitch-free from one clock to the other. Each DCM block is able to drive up to four of the 16 global clock MUX buffers.

Routing Resources

The IOB, CLB, block SelectRAM, multiplier, and DCM elements all use the same interconnect scheme and the same access to the global routing matrix. Timing
models are shared, greatly improving the predictability of the performance of high-speed designs. There are a total of 16 global clock lines, with eight available per quadrant. In addition, 24 vertical and horizontal long lines per row or column as well as massive secondary and local routing resources provide fast interconnect.

4.3.2 Spartan-3 FPGA [7]

The 1.2V Spartan-3 family FPGAs is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. This family offers densities ranging from 50,000 to five million system gates.

The Spartan-3 is an improved version of Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os and the overall level of performance as well as by improving clock management functions. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment.

Features

- Very low cost, high-performance logic solution for high-volume, consumer-oriented applications
  - densities as high as 74K logic cells
  - 325 MHz system clock rate
  - Three separate power supplies for the core (1.2V), I/Os(1.2V to 3.3V), and special functions (2.5V)
- Select IO signaling
  - upto 784 I/O pins.
  - 622 Mbps data transfer rate per I/O
  - seventeen single-ended signal standards
  - six differential signal standards including LVDS
  - termination by digitally controlled impedance
  - signal swing ranging from 1.14V to 3.45V
  - Double Data Rate (DDR) support

- Logic Resources
  - Abundant, flexible logic cells with registers
  - wide multiplexers
  - fast look-ahead carry logic
  - dedicated 18 X 18 multipliers
  - JTAG logic compatible with IEEE 1149.1 / 1532 standards

- SelectRAM hierarchical memory
  - up to 1,872 Kb of total block RAM
  - up to 520 Kb of total distributed RAM

- Digital Clock Manager (up to four DCMs)
  - clock skew elimination
- Frequency synthesis
- High-resolution phase shifting

- Eight global clock lines and abundant routing
- Fully supported by Xilinx ISE development system
  - Synthesis, mapping, placement and routing.

Architectural Overview

The block diagram of the Spartan-3 is shown in Figure 4.5. The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.

- Input/Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus three-state operation. Twenty-three different signal standards, including six high-performance differential standards, are available. Double data Rate registers are included. The digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.

- Block RAM provides data storage in the form of 18Kb dual-port blocks.

- Multiplier blocks accept two 18 bit binary numbers as inputs and calculate the product.
Figure 4.5: Spartan-3 Family Architecture
Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing and phase shifting clock signals.

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

4.3.3 Spartan-2 FPGA [8]

The Spartan-II family of FPGAs is implemented with a regular, flexible, programmable architecture of CLBs, surrounded by a perimeter of programmable IOBs, interconnected by powerful hierarchy of versatile routing resources. The architecture also provides advanced functions such as Block RAM and clock control blocks. The architecture diagram of Spartan-II is shown in Figure 4.6.

Input / Output Block

The Spartan-II IOB features inputs and outputs that support 16 I/O signaling standards, including LVCOMS, HSTL, SSTL and GTL. These high speed inputs and outputs are capable of supporting various state-of-art memory and bus interfaces. The three IOB registers function either as edge triggered D-type flip-flops or as level sensitive latches. Each IOB has a clock signal (CLK) shared by the three registers and independent clock enable (CE) signals for each register. In addition to the CLK and CE control signals, the three registers share a Set/Reset (SR) signal. For each register
Figure 4.6: Basic Spartan-II Family FPGA Block Diagram
this signal can be independently configured as a synchronous Set, a synchronous Reset, an asynchronous Reset or an asynchronous Clear.

Logic Cells

The basic building block of the Spartan-II CLB is the logic cell (LC). An LC includes a four input function generator, carry logic and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each Spartan-II CLB contains four LCs, organized in two similar slices. In addition to the four basic LCs, the Spartan-II CLB contains logic that combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5LCs.

Spartan-II function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 X1 bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16 X 2 bit or 32 X 1 bit synchronous RAM, or a 16 X 1 dual-port synchronous RAM. The Spartan-II LUT can also provide a 16 bit shift register that is ideal for capturing high-speed or burst-mode data. The storage elements in Spartan-II slice can be configured either as edge triggered D-type flip-flops or as level sensitive latches.

Configuration

Configuration is the process by which the FPGA is programmed with the configuration file generated by the Xilinx development system. Spartan-II devices
support both serial configuration, using the master/slave serial and JTAG modes, as well as byte-wide configuration employing the slave parallel mode.

4.4 ALTERA FPGAs

4.4.1 Stratix FPGA [9]

The StratixTM family of FPGAs is based on a 1.5-V, 0.13-μm, all-layer copper SRAM process, with densities up to 114,140 logic elements (LEs) and up to 10 Mbits of RAM. Stratix devices offer up to 28 digital signal processing (DSP) blocks with up to 224 (9-bit × 9-bit) embedded multipliers, optimized for DSP applications that enable efficient implementation of high-performance filters and multipliers. Stratix devices support various I/O standards and also offer a complete clock management solution with its hierarchical clock structure with up to 420-MHz performance and up to 12 phase-locked loops (PLLs).

Features

- 10,570 to 114,140 LEs;
- Up to 10,118,016 RAM bits (1,264,752 bytes) available without reducing logic resources.
- TriMatrixTM memory consisting of three RAM block sizes to implement true dual-port memory and first-in first-out (FIFO) buffers.
- High-speed DSP blocks provide dedicated implementation of multipliers (at up to 250 MHz), multiply-accumulate functions.
- Up to 16 global clocks with 22 clocking resources per device region.
• Up to 12 PLLs (four enhanced PLLs and eight fast PLLs) per device provide spread spectrum, programmable bandwidth, clock switchover, real-time PLL reconfiguration, and advanced multiplication and phase shifting.

• Support for numerous single-ended and differential I/O standards

• High-speed differential I/O support on up to 116 channels with up to 80 channels optimized for 840 megabits per second (Mbps).

• Terminator™ technology provides on-chip termination for differential and single-ended I/O pins with impedance matching.

• Support for high-speed external memory, including zero bus turnaround (ZBT) SRAM, quad data rate (QDR and QDRII) SRAM, double data rate (DDR) SDRAM, DDR fast cycle RAM (FCRAM), and single data rate (SDR) SDRAM.

• Support for multiple intellectual property megafunctions from Altera MegaCore® functions and Altera Megafuntion Partners Program (AMPPSM) megafunctions.

Stratix devices are available in space-saving FineLine BGATM and ball-grid array (BGA) packages. All Stratix devices support vertical migration within the same package (e.g., the designer can migrate between the EP1S10, EP1S20, and EP1S25 devices in the 672-pin BGA package). Vertical migration means that designers can migrate to devices whose dedicated pins, configuration pins, and power pins are the same for a given package across device densities. For I/O pin migration across densities, the designer must cross reference the available I/O pins using the device
pin-outs for all planned densities of a given package type to identify which I/O pins are migratable.

Functional Description

The Stratix block diagram is shown in Figure 4.7. Stratix devices contain a two-dimensional row- and column-based architecture to implement custom logic. A series of column and row interconnects of varying length and speed provides signal interconnects between logic array blocks (LABs), memory block structures, and DSP blocks.

The logic array consists of LABs, with 10 logic elements (LEs) in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LABs are grouped into rows and columns across the device.

M512 RAM blocks are simple dual-port memory blocks with 512 bits plus parity (576 bits). These blocks provide dedicated simple dual-port or single-port memory up to 18-bits wide at up to 318 MHz. M512 blocks are grouped into columns across the device in between certain LABs. M4K RAM blocks are true dual-port memory blocks with 4K bits plus parity (4,608 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 36-bits wide at up to 291 MHz. These blocks are grouped into columns across the device in between certain LABs.

M-RAM blocks are true dual-port memory blocks with 512K bits plus parity (589,824 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 144-bits wide at up to 269 MHz. Several M-RAM blocks
Figure 4.7: Stratix Block Diagram
are located individually or in pairs within the device's logic array. DSP blocks can implement up to either eight full-precision $9 \times 9$-bit multipliers, four full-precision $18 \times 18$-bit multipliers, or one full-precision $36 \times 36$-bit multiplier with add or subtract features. These blocks also contain 18-bit input shift registers for digital signal processing applications. Each Stratix device I/O pin is fed by an I/O element (IOE) located at the end of LAB rows and columns around the periphery of the device. I/O pins support numerous single-ended and differential I/O standards. Each IOE contains 7a bidirectional I/O buffer and six registers for registering input, output, and output-enable signals.

4.4.2 Cyclone FPGA [10]

The Cyclone™ field programmable gate array family is based on a 1.5-V, 0.13-μm, all-layer copper SRAM process, with densities up to 20,060 logic elements (LEs) and up to 288 Kbits of RAM. With features like phaselocked loops (PLLs) for clocking and a dedicated double data rate (DDR) interface to meet DDR SDRAM and fast cycle RAM (FCRAM) memory requirements, Cyclone devices are a cost-effective solution for data-path applications. Cyclone devices support various I/O standards, including LVDS at data rates up to 311 megabits per second (Mbps) and 66-MHz, 32-bit peripheral component interconnect (PCI), for interfacing with and supporting ASSP and ASIC devices. Altera also offers new low-cost serial configuration devices to configure Cyclone devices.
Features

- 2,910 to 20,060 LEs.
- Up to 294,912 RAM bits (36,864 bytes).
- Supports configuration through low-cost serial configuration device.
- Support for LVTTL, LVCMOS, SSTL-2, and SSTL-3 I/O standards.
- Support for 66-MHz, 32-bit PCI standard.
- Low speed (311 Mbps) LVDS I/O support.
- Up to two PLLs per device provide clock multiplication and phase shifting.
- Up to eight global clock lines with six clock resources available per logic array block (LAB) row.
- Support for external memory, including DDR SDRAM (133 MHz), FCRAM, and single data rate (SDR) SDRAM
- Support for multiple intellectual property (IP) cores, including Altera. MegaCore functions and Altera Megafuntions Partners Program (AMPPSM) megafunctions.

Functional Description

Cyclone devices contain a two-dimensional row- and column-based architecture to implement custom logic. Column and row interconnects of varying speeds provide signal interconnects between LABs and embedded memory blocks.
The logic array consists of LABs, with 10 LEs in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LABs are grouped into rows and columns across the device. Cyclone devices range between 2,910 to 20,060 LEs.

M4K RAM blocks are true dual-port memory blocks with 4K bits of memory plus parity (4,608 bits). These blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 36-bits wide at up to 200 MHz. These blocks are grouped into columns across the device in between certain LABs. Cyclone devices offer between 60 to 288 Kbits of embedded RAM.

Each Cyclone device I/O pin is fed by an I/O element (IOE) located at the ends of LAB rows and columns around the periphery of the device. I/O pins support various single-ended and differential I/O standards, such as the 66-MHz, 32-bit PCI standard and the LVDS I/O standard at up to 311 Mbps. Each IOE contains a bidirectional I/O buffer and three registers for registering input, output, and output-enable signals. Dual-purpose DQS, DQ, and DM pins along with delay chains (used to phase-align DDR signals) provide interface support with external memory devices such as DDR SDRAM, and FCRAM devices at up to 133 MHz (266 Mbps).

Cyclone devices provide a global clock network and up to two PLLs. The global clock network consists of eight global clock lines that drive throughout the entire device. The global clock network can provide clocks for all resources within the device, such as IOEs, LEs, and memory blocks.

The global clock lines can also be used for control signals. Cyclone PLLs provide general-purpose clocking with clock multiplication and phase shifting as well
as external outputs for high-speed differential I/O support. Figure 4.8 shows a diagram of the Cyclone EPIC12 device.

4.4.3 Mercury FPGA [11]

Introduction

Mercury devices are the first PLDs optimized for core performance. These LUT-based, enhanced memory devices use a network of fast routing resources to achieve optimal performance. These resources are ideal for data-path, register-intensive, mathematical, digital signal processing (DSP), or communications designs. The Mercury FPGA block diagram is given in Figure 4.9.

Features

- High-performance programmable logic device family.
  - Integrated high-speed transceivers with support for clock data recovery (CDR) at up to 1.25 gigabits per second (Gbps)
  - Look-up table based architecture optimized for high speed
  - Advanced interconnect structure for fast routing of critical paths
  - Enhanced I/O structure for versatile standards and interface support
  - Up to 14,400 LEs

- System-level features
  - Up to four general-purpose phase-locked loops (PLLs) with programmable multiplication and delay shifting
Local Interconnect: Connects LES within the Same or Adjacent LABs

Row and Priority Row Interconnect: Connects LABs within a Row

Column and Priority Column Interconnect: Connects LABs within Different Rows (Top to Bottom)

Leap Lines: Connects Adjacent LABs in Same Column

RapidLAB Interconnect: Connects Any 10 Consecutive LABs within a Row from a Central LAB

Figure 4.9: Mercury Architecture Block Diagram
- Up to 12 PLL output ports
- Dedicated multiplier circuitry for high-speed implementation of signed or unsigned multiplication up to $16 \times 16$
- Embedded system blocks (ESBs) used to implement memory functions including quad-port RAM, true dual-port RAM, first-in-first-out (FIFO) buffers, and content-addressable memory (CAM)
- Each ESB contains 4,096 bits and can be split and used as two 2,048-bit unidirectional dual-port RAM blocks

- Advanced high-speed I/O features
  - Robust I/O standard support, including LVTTL, PCI up to 66 MHz, 3.3-V AGP in $1 \times$ and $2 \times$ modes, 3.3-V SSTL-3 and 2.5-V SSTL-2, GTL+, HSTL, CTT, LVDS, LVPECL, and 3.3-V PCML.
  - High-speed differential interface (HSDI) with dedicated circuitry for CDR at up to 1.25 Gbps for LVDS, LVPECL, and 3.3-V PCML.
  - Support for source-synchronous True-LVDSTM circuitry up to 840 megabits per second (Mbps) for LVDS, LVPECL, and 3.3-V PCML.
  - Up to 18 input and 18 output dedicated differential channels of high-speed LVDS, LVPECL, or 3.3-V PCML.
  - Built-in 100- termination resistor on HSDI data and clock differential pairs.
Flexible-LVDSTTM circuitry provides 624-Mbps support on up to 100 channels with the EP1M350 device.

Versatile three-register I/O element (IOE) supporting double data rate I/O (DDRIO), double data-rate (DDR) SDRAM, zero bus turnaround (ZBT) SRAM, and quad data rate (QDR) SRAM.

- Designed for low-power operation
  - 1.8-V internal supply voltage (VCCINT)
  - MultiVoltTM I/O interface voltage levels (VCCIO) compatible with 1.5-V, 1.8-V, 2.5-V, and 3.3-V devices
  - 5.0-V tolerant with external resistor

- Advanced interconnect structure
  - Multi-level FastTrack® Interconnect structure providing fast, predictable interconnect delays.
  - Optimized high-speed Priority FastTrack Interconnect for routing critical paths in a design.
  - Dedicated carry chain that implements arithmetic functions such as fast adders, counters, and comparators (automatically used by software tools and megafunctions)
  - Leap lines allowing a single LAB to directly drive LEs in adjacent rows
The RapidLAB interconnect providing a high-speed connection to a 10-LAB-wide region.

Dedicated clock and control signal resources, including four dedicated clocks, six dedicated fast global signals, and additional row-global signals.

4.4.4 Flex 10KE FPGA [12]

Altera FLEX 10KE devices are enhanced versions of FLEX 10K devices. Based on reconfigurable CMOS SRAM elements. Up to 200,000 typical gates, FLEX 10KE devices provide the density, speed, and features to integrate entire systems, including multiple 32-bit buses, into a single device. The ability to reconfigure FLEX 10KE devices enables 100% testing prior to shipment and allows the designer to focus on simulation and design verification. FLEX 10KE reconfigurability eliminates inventory management for gate array designs and generation of test vectors for fault coverage. The block diagram of Flex 10 KE device is shown in Figure 4.10.

Features

- Embedded programmable logic devices (PLDs), providing system-on-a-programmable-chip (SOPC) integration in a single device
  
  - Enhanced embedded array for implementing megafunctions such as efficient memory and specialized logic functions.
  
  - Dual-port capability with up to 16-bit width per embedded array block (EAB)
  
  - Logic array for general logic functions
Figure 4.10: FLEX 10KE Device Block Diagram
• High density
  o 30,000 to 200,000 typical gates
  o Up to 98,304 RAM bits (4,096 bits per EAB), all of which can be used without reducing logic capacity

• System-level features
  o MultiVolt TM I/O pins can drive or be driven by 2.5-V, 3.3-V, or 5.0-V devices
  o Low power consumption.
  o Bidirectional I/O performance.
  o Fully compliant with the PCI Special Interest Group (PCI SIG) PCI Local Bus Specification, Revision 2.2 for 3.3-V operation at 33 MHz or 66 MHz
  o -1 speed grade devices are compliant with PCI Local Bus Specification, Revision 2.2, for 5.0-V operation.
  o Built-in Joint Test Action Group (JTAG) boundary-scan test (BST) circuitry compliant with IEEE Std. 1149.1-1990, available without consuming additional device logic.
  o Fabricated on an advanced process and operate with a 2.5-V internal supply voltage.
  o In-circuit reconfigurability (ICR) via external configuration devices, intelligent controller, or JTAG port.
ClockLockTM and ClockBoostTM options for reduced clock delay/skew and clock multiplication.

Built-in low-skew clock distribution trees.

100% functional testing of all devices; test vectors or scan chains are not required.

Pull-up on I/O pins before and during configuration.

Flexible interconnect

FastTrack® Interconnect continuous routing structure for fast, predictable interconnect delays

Dedicated carry chain that implements arithmetic functions such as fast adders, counters, and comparators (automatically used by software tools and megafunctions)

Dedicated cascade chain that implements high-speed, high-fan-in logic functions (automatically used by software tools and megafunctions)

Tri-state emulation that implements internal tri-state buses

Up to six global clock signals and four global clear signals

Powerful I/O pins

Individual tri-state output enable control for each pin.

Open-drain option on each I/O pin.

Programmable output slew-rate control to reduce switching noise.
o Clamp to VCCIO user-selectable on a pin-by-pin basis.

o Supports hot-socketing.

The above description on the various architectures of FPGAs clearly indicate that, they differ from one another with regard to the number of CLBs, rows and columns used in each FPGA.

The following section describes the applications of FPGAs in Digital Signal Processing.
PART - B

As mentioned earlier, this part deals with the implementation of FIR filters using FPAGAs.

4.5 FPGAs for DSP Applications

There are a large number of choices available in designing and implementing a digital signal processing system. Certain of them are:

- Using Microprocessors
- Using DSP processors
- Using Application Specific Integrated Circuits (ASICs)
- Using FPGAs.

All brief description and comparison of these methods are given below.

Programmable DSPs

The most common vehicle for implementation of a DSP design is the programmable DSP or pDSP. The pDSP is an off-the-shelf part that is essentially a microprocessor tuned to DSP applications. pDSPs are highly flexible because one can program them again and again using a familiar high level language like C. They allow fast design iterations and reduce time to market.

Typically a pDSP contains several functional units to process the signal stream. The designer encodes the algorithm into a program which is executed by the pDSP and is limited to a theoretical maximum data rate based on the speed and the number of multiplier/accumulators in the device[13]. Applications which require
several computations must be broken up into a sequential stream of computations. For example, an 8-tap FIR filter requires 8 multiplications and one 8 way addition per data sample. The implementation of this FIR filter might require 8 or more cycles on a pDSP. At each data sample, all eight taps require multiplication by their coefficients. If the number of taps for this filter were increased, then the number of cycles would also be increased, thereby reducing the data rate. Programmable DSP chips are intrinsically limited in performance. The more we want to do to a data sample, the more cycles the system need and the slower will be the data processing rate. One way to overcome this limitation is to employ more pDSP parts to implement the algorithm. Another method is to use FPGAs to implement the algorithm in hardware [14,15].

In traditional DSPs, the FIR filters are implemented in dedicated hardware without any parallelism, thus limiting the sample rate. The FPGAs have abundant hardware resources to facilitate full parallelism (each TAP has a dedicated multiplier and adder). For multiplier performance improvement, the features of the filter have to be carefully studied. The efficiency of the multiplier determines the overall performance of the filter. Hence, the multiplier must be implemented for the best possible performance.

The only drawback of FPGAs is, for very high production volumes, FPGAs do not exhibit as high of a performance/cost ratio as full custom Application Specific Integrated Circuit (ASIC) designs. The lower performance/cost of FPGAs comes in part because FPGAs must sacrifice some silicon area in order to be highly flexible. However, this should not be a concern to the designer expecting to ramp up to high volume.
Other advantages of FPGAs include:

- Parts may be reprogrammed over and over. If we want to upgrade the design, we do not need to replace FPGAs, just reprogram them.

- FPGAs are pre-tested. Traditional Gate Array design methodology requires that you also develop costly manufacturing test suites. This task is not required with FPGAs.

- FPGAs are a commodity part. Xilinx and Altera sells millions of FPGAs annually. This high production volume results in a lower per part cost and those savings are passed on to the customer.

- FPGAs can be dynamically reconfigured within the system. Sophisticated designers can build systems which adapt to changing conditions by altering the circuit configured within the FPGA. This re-configurable design approach is becoming more and more popular since many systems need to perform several different functions, but never all of them at the same time.

4.6 FPGAs Implementation of DSP systems

An FPGA design starts with the input circuit schematic or high level design description. Automatic synthesis, place, and route tools can be used to translate the designer's original circuit into an FPGA specific configuration.

FPGAs are generic commodity parts and are customized by downloading a user defined configuration in the form of a binary bitstream, much the same as you would load a pDSP with its program with a process that typically takes only a few
milliseconds. Most DSP designs are part of a complex system. It is very common for complex systems to go through several design iterations before product completion.

4.6.1 Structures for FIR Filters

Direct Form Structure

Digital filter algorithms are primarily composed of multipliers, adders, and registers. The basic structure of a Finite Impulse Response (FIR) filter is shown in Figure 4.11. The multipliers and adders form the heart of a FIR filter. The input data passes to the multiplier and then to the adder with interleaving delay elements. The critical path or minimum time required for processing a new sample is limited by one multiply and N-1 addition times, i.e. if $T_M$ is the time taken for multiplication and $T_A$ is the time needed for addition and N is the number of taps, then the sample period is given by

$$T_{sample} \geq T_M + (N - 1)T_A$$ \hspace{1cm} 4.1$$

Therefore the sampling frequency $f_{sample}$ or the throughput is given by

$$f_{sample} = \frac{1}{T_M + (N - 1)T_A}$$ \hspace{1cm} 4.2$$

The direct form structure can be used when the equation 4.1 is satisfied. But in some real time application demands a faster input rate, then this structure can't be used.

Transposed Form Structure

The critical path of the direct form structure can be reduced by this structure. The Transposition Theorem states that "Reversing the direction of all the edges in a
**Figure 4.11:** Basic direct form structure of FIR filter

**Figure 4.12:** Linear Phase Implementation of direct form FIR filter
given Signal Flow Graph (SFG) and interchanging the input and output ports preserves the functionality of the system" [16].

The Transposed Form FIR filter is shown in Figure 4.21. This structure is also called as the Broadcast Structure as the data is applied to all the multipliers simultaneously. This structure requires same resources as that of direct form structure. The products are applied to a cascaded chain of registered adders, combining the effect of accumulators and registers. The order of tap coefficients must be reversed with the first tap closest to the output. This structure allows expansion of the number of taps required in a filter, since each "tap module" is identical. Since the structure is uniform, a single component can be designed and instantiated as many times as required by the number of taps. The critical path of this structure is $T_m + T_A$.

**Exploiting Filter Symmetry**

The impulse response for many filters possesses significant symmetry. This symmetry can be exploited to minimize arithmetic requirements and produce area efficient filter realizations.

Instead of implementing this filter using the architecture shown in Figure 4.11, the more efficient signal flow-graph in Figure 4.12 can be used. In general the former approach requires $N$ multiplications and $(N-1)$ additions. In contrast, the architecture in Figure 4.12 requires only $N/2$ multiplications and approximately $N$ additions. This significant reduction in the computation workload can be exploited to generate efficient filter hardware implementations.
Pipelining of FIR Digital Filters

Pipelining means that a function such as multiplication is divided into smaller steps. Smaller steps mean less circuitry between clock steps, which in turn means that the clock can run faster and overall performance increases. Pipelining comes at an expense - results are delayed some small finite number of clock cycles before they are complete, but this latency is of negligible effect on DSP designs. For most DSP applications the data rate is much more important than any latency.

The critical path of pipelined implementation of 8 tap FIR filter structure is reduced to $T_M + T_A$.

**VHDL / Verilog code for implementing 8 Tap FIR Filter**

For explanation purpose, only the equations part of the VHDL [17-19] code is described below. Similar results are obtained using Verilog [20,21]. Hence only VHDL codes are described in the following section.

**Direct Form realization of FIR filters**

For all the below programs one cycle is required to latch the input and 8 cycles were required to shift the first input to the last tap. All the equations are written under clock’event so one cycle is required to latch the result in the left hand side variable.

The code to implement the shifting of registers is given below.

\[ x(0) \leq xi; \quad \text{--latching the input.} \]

for i in 1 to 7 loop

\[ x(i) \leq x(i+1); \]
In this code \(x(0), x(1), \ldots, x(7)\) are the 8 bit registers which will act as a tap and ‘\(x_i\)’ is the new input digital data. With this code one cycle is required to receive \(x_i\) input and 8 more cycles are required to shift the new data to last tap.

1) Filtering equation for latency of 9 cycles: The hardware implementation diagram is shown in Figure 4.13 and VHDL code is given in Program 4.1.

\[
z \leq x(0) \cdot h(0) + x(1) \cdot h(1) + x(2) \cdot h(2) + x(3) \cdot h(3) + x(4) \cdot h(4) + x(5) \cdot h(5) + x(6) \cdot h(6) + x(7) \cdot h(7);
\]

With this code all the multiplications and additions were performed in one clock cycle. The timing diagram of the signals is given in Figure 4.14.

2) For latency of 10 cycles and implementing linear phase method of realisation:

for \(i\) in 0 to 3 loop. The hardware implementation diagram is shown in Figure 4.15 and VHDL code is given in Program 4.2.

\[
s(i) \leq x(i) + x(7-i);
\]

end loop;

\[
z \leq s(0) \cdot h(0) + s(1) \cdot h(1) + s(2) \cdot h(2) + s(3) \cdot h(3);
\]

With this code one latency is increased compared with the previous case because of adders which are used to calculate \(s(0), s(1), s(2)\) and \(s(3)\). The timing diagram of the signals is given in Figure 4.16.

3) For latency of 11 cycles and implementing linear phase method of realization:
for i in 0 to 3 loop. The hardware implementation diagram is shown in Figure 4.17 and VHDL code is given in Program 4.3.

\[
s(i) \leq x(i) + x(7-i);
\]
end loop;

for i in 0 to 3 loop

\[
y(i) \leq s(i) \cdot h(i);
\]
end loop;

\[
z \leq y(0) + y(1) + y(2) + y(3);
\]

With this code one more latency is increased compared with the previous case because of multipliers which are used to calculate \(y(0), y(1), y(2)\) and \(y(3)\). The timing diagram of the signals is given in Figure 4.18.

4) For latency of 12 cycles and implementing linear phase method of realization:

for i in 0 to 3 loop. The hardware implementation diagram is shown in Figure 4.19 and VHDL code is given in Program 4.4.

\[
s(i) \leq x(i) + x(7-i);
\]
end loop;

for i in 0 to 3 loop

\[
y(i) \leq s(i) \cdot h(i);
\]
end loop;

\[
y(4) \leq y(0) + y(1);
\]
With this code one more latency cycle is increased compared with the previous case, during this time the $y(4)$ and $y(5)$ are evaluated. The timing diagram of the signals is given in Figure 4.20.

It is quite evident from the above study that an increase in the latency leads to an increase in the maximum operational frequency.

**Transposed form of FIR filter Realization**

The hardware implementation diagram is shown in Figure 4.21 and VHDL code is given in Program 4.5. The following part of the code implements transpose form of FIR filter for 8 taps. All the equations are written in a process / always statement.

```
s0<=x*h(0);
s1<=x*h(1);
s2<=x*h(2);
s3<=x*h(3);
s4<=x*h(4);
s5<=x*h(5);
s6<=x*h(6);
s7<=x*h(7);
```
For all the multipliers the same incoming data x is applied. After multiplication with the coefficient all the terms are added to get the filtered output as shown in transposed FIR filter realization diagram. The latency of this code is nine cycles. The timing diagram of the signals is given in Figure 4.22.

This particular realization program has registered a maximum frequency with Altera Mercury FPGA.

**VHDL / Verilog code for implementing 16 Tap FIR Filter**

Proceeding in the similar way as described earlier the following code is written to implement 16 tap FIR filter.

The code to implement the shifting of registers is given below:

\[ x(0) = x_i; \]

\[ \text{for } i \text{ in } 1 \text{ to } 15 \text{ loop} \]
x(i) <= x(i+1);
end loop;

1) Filtering equation for latency of 17 cycles: The hardware implementation diagram is shown in Figure 4.23 and VHDL code is given in Program 4.6.

\[ y \leq x(0)h(0) + x(1)h(1) + x(2)h(2) + x(3)h(3) + x(4)h(4) + x(5)h(5) + \]
\[ x(6)h(6) + x(7)h(7) + x(8)h(8) + x(9)h(9) + x(10)h(10) + x(11)h(11) + \]
\[ x(12)h(12) + x(13)h(13) + x(14)h(14) + x(15)h(15); \]

The timing diagram of the signals is given in Figure 4.24.

2) For latency of 18 cycles and implementing linear phase method of realisation:

The hardware implementation diagram is shown in Figure 4.25 and VHDL code is given in Program 4.7.

\[ \text{for } i \text{ in } 0 \text{ to } 7 \text{ loop} \]
\[ s(i) \leq x(i) + x(15-i); \]
\[ \text{end loop;} \]
\[ z \leq s(0)h(0) + s(1)h(1) + s(2)h(2) + s(3)h(3) + s(4)h(4) + s(5)h(5) + \]
\[ s(6)h(6) + s(7)h(7); \]

With this code one latency is increased compared with the previous case because of adders which are used to calculate s(0),...s(7). The timing diagram of the signals is given in Figure 4.26.
3) For latency of 19 cycles and implementing efficient method of realization: The hardware implementation diagram is shown in Figure 4.27 and VHDL code is given in Program 4.8.

```vhdl
for i in 0 to 7 loop
    s(i) <= x(i) + x(15-i);
end loop;

for i in 0 to 7 loop
    y(i) <= s(i) * h(i);
end loop;

z <= y(0) + y(1) + y(2) + y(3) + y(4) + y(5) + y(6) + y(7);
```

With this code one more latency is increased compared with the previous case because of multipliers which are used to calculate y(0).....y(7). The timing diagram of the signals is given in Figure 4.28.

4) For latency of 21 cycles and implementing linear phase method of realization: The hardware implementation diagram is shown in Figure 4.29 and VHDL code is given in Program 4.9.

```vhdl
for i in 0 to 7 loop
    s(i) <= x(i) + x(15-i);
end loop;

for i in 0 to 7 loop
    y(i) <= s(i) * h(i);
end loop;

z <= y(0) + y(1) + y(2) + y(3) + y(4) + y(5) + y(6) + y(7);
```
y(i) <= s(i)*h(i);

end loop;

y(8)<= y(0)+y(1)+y(2)+y(3);

y(9)<=y(4)+y(5)+y(6)+y(7);

z <= y(8)+y(9);

With this code one more latency cycle is increased compared with the previous case, during this time the y(8) and y(9) are evaluated. The timing diagram of the signals is given in Figure 4.30.

5) For latency of 22 cycles and implementing efficient method of realization: The hardware implementation diagram is shown in Figure 4.31 and VHDL code is given in Program 4.10.

for i in 0 to 7 loop
s(i) <= x(i) +x(15-i);
end loop;

for i in 0 to 7 loop
y(i) <= s(i)*h(i);
end loop;

y(8)<= y(0)+y(1);

y(9)<=y(2)+y(3);

y(10)<=y(4)+y(5);
\( y(11) \leq y(6) + y(7); \)
\( y(12) \leq y(8) + y(9); \)
\( y(13) \leq y(10) + y(11); \)
\( z \leq y(12) + y(13); \)

With this code one more latency cycle is increased compared with the previous case, during this time the \( y_{12}, y_{13} \) are evaluated. The timing diagram of the signals is given in Figure 4.32.

The hardware implementation diagram for 16 tap transposed implementation is shown in Figure 4.33 and VHDL code is given in Program 4.11. The timing diagram of the signals is given in Figure 4.34.

In all the timing diagrams the end of the latency is indicated with a black line. After this time instant only the filters will give correct results.
Figure 4.13: 8 tap direct form realization
Output

Input data $x_i$

Reg.

Reg.

Reg.

Reg.

Reg.

Reg.

ADDER

Output
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity filters is
port
  (elk:in std_logic;
   xl:in std_logic_vector(7 downto 0);
   z:out std_logic_vector(15 downto 0) );
end filter8;

architecture filter_arch of filters is

type sigtype1 is array (0 to 15) of std_logic_vector(7 downto 0);
signal x:sigtype1;
type sigtype2 is array (0 to 7) of std_logic_vector(7 downto 0);
constant h:sigtype2 := (
  "01111111","01111111","01111111",
  "01111111","01111111","01111111",
  "01111111","01111111" );

begin
  process (clk)
  begin
    if clk='1' and clk'event then
      x(0)<=xl;
      for i in 1 to 7 loop
        x(i)<=x(i-1);
      end loop;
      z<= h(0)*x(0)+h(1)*x(1)+h(2)*x(2)+h(3)*x(3)+h(4)*x(4)+h(5)*x(5)+h(6)*x(6)+x(7)*h(7);
    end if;
  end process;
end filter_arch;

Program 4.1: VHDL Program for hardware shown in Figure 4.13
Figure 4.14: Waveforms obtained in implementing the hardware shown in Figure 4.13

Latency = 9 cycles
Figure 4.15: 8 tap Linear phase realization

Input data

Adder

Output

h_0

h_1

h_2

h_3

Reg_1

Reg_2

Reg_3

Reg_4

Reg_5

Reg_6

Reg_7

Reg_8

x_1

x_2

x_3

x_4

x_5

x_6

x_7

x_8

x_0

x_9

x_{10}

x_{11}

x_{12}

x_{13}

x_{14}

x_{15}
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity filter8 is
port
 (elk:in std_logic;
  xl:in std_logic_vector(7 downto 0);
  z:out std_logic_vector(15 downto 0) );
end filter8;

architecture filter_arch of filter8 is

  type sigtypel is array (0 to 7) of std_logic_vector (7 downto 0);
signal x:sigtypel;

type sigtype3 is array (0 to 4) of std_logic_vector (7 downto 0);
signal s:sigtype3;

  type sigtype2 is array (0 to 3) of std_logic_vector (7 downto 0);
constant h:sigtype2 :=("01111111","01111111","01111111" ,"01111111");

begin
  process(clk)
  begin
      if clk='1' and clk'event then
          x(0)<=xl;
          for i in 1 to 7 loop
              x(i)<=x(i-1);
          end loop;
          for i in 0 to 3 loop
              s(i)<= x(i)+x(7-i);
          end loop;
          z<= s(0)*h(0)+s(1)*h(1)+s(2)*h(2)+s(3)*h(3);
      end if;
  end process;
end filter_arch;

Program 4.2: VHDL Program for hardware shown in Figure 4.15
Figure 4.16: Waveforms obtained in implementing the hardware shown in Figure 4.15. Latency = 10 cycles.
Figure 4.17: 8 tap Linear phase realization with pipeline registers used after the multipliers
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

description filter8 is
port
(clk:in std_logic;
 xl:in std_logic_vector(7 downto 0);
z:out std_logic_vector(15 downto 0) );
end filter8;

architecture filter_arch of filter8 is
type sigtype4 is array (0 to 3) of std_logic_vector(15 downto 0);
signal y:sigtype4;
type sigtype1 is array (0 to 7) of std_logic_vector(7 downto 0);
signal x:sigtype1;
type sigtype3 is array (0 to 4) of std_logic_vector(7 downto 0);
signal s:sigtype3;
type sigtype2 is array (0 to 3) of std_logic_vector(7 downto 0);
constant h:sigtype2 :=("01111111","00111111","01111111 ", "01111111");
begin
process(clk)
begin
if clk='1' and clk'event then
 x(0)<=xl;
 for i in 1 to 7 loop
 x(i)<=x(i-1);
 end loop;
 for i in 0 to 3 loop
 s(i)<= x(i)+x(7-i);
 end loop;
 for i in 0 to 3 loop
 y(i)]<= s(i)*h(i);
 end loop;
z<= y(0)+y(1)+y(2)+y(3);
end if;
end process;
end filter_arch;

Program 4.3: VHDL Program for hardware shown in Figure 4.17
Figure 4.18: Waveforms obtained in implementing the hardware shown in Figure 4.17.
Figure 4.19: 8-tap Linear phase realization with pipelined adder
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity filter8 is
port
  (clk:in std_logic;
   xl :in std_logic_vector(7 downto 0) ;
   z:out std_logic_vector(15 downto 0) );
end filter8;

architecture filter_arch of filter8 is

  type sigtype4 is array (0 to 5) of std_logic_vector (15 downto 0);
signal y:sigtype4;

  type sigtypel is array (0 to 7) of std_logic_vector (7 downto 0);
signal x:sigtypel;

  type sigtype3 is array (0 to 4) of std_logic_vector (7 downto 0);
signal s:sigtype3;

  type sigtype2 is array (0 to 3) of std_logic_vector (7 downto 0);
constant h:sigtype2 :=
  ("01111111", "01111111", "01111111","01111111");
begin
  process (clk)
  begin
    if clk='1' and clk'event then
      x(0)<=xl;
      for i in 1 to 7 loop
        x(i)<=x(i-1);
      end loop;
      for i in 0 to 3 loop
        s(i)<= x(i)+x(7-i);
      end loop;
      for i in 0 to 3 loop
        y(i)<= s(i)*h(i);
      end loop;
      y(4)<= y(0)+y(1);
      y(5)<= y(2)+y(3);
      z<= y(4)+y(5);
    end if;
  end process;
end filter_arch;

Program 4.4: VHDL Program for hardware shown in Figure 4.19
Figure 4.20: Waveforms obtained in implementing the hardware shown in Figure 4.19. Latency = 12 cycles.
Figure 4.21: 8 Tap Transposed form of FIR filter
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity firt8 is
port ( clk:in std_logic;
       x:in std_logic_vector(7 downto 0);
       y:out std_logic_vector(15 downto 0)
     );
end firt8;

architecture fir_arch of firt8 is
signal s0,s1,s2,s3,s4,s5,s6,s7:std_logic_vector(15 downto 0);
signal y1,y2,y3,y4,y5,y6,y7,y8:std_logic_vector(15 downto 0);
type sigtype is array (0 to 7) of std_logic_vector(7 downto 0);
constant h:sigtype :=("01111111","01111111","01111111","01111111"
,"01111111","01111111","01111111","01111111");
begin
process(clk,x)
begin
if clk='1' and clk'event then
s0<=x*h(0);
s1<=x*h(1);
s2<=x*h(2);
s3<=x*h(3);
s4<=x*h(4);
s5<=x*h(5);
s6<=x*h(6);
s7<=x*h(7);
y7<=s7;
y6<=y7+s6;
y5<=y6+s5;
y4<=y5+s4;
y3<=y4+s3;
y2<=y3+s2;
y1<=y2+s1;
y<=y1+s0;
end if;
end process;
end fir_arch;

Program 4.5: VHDL Program for hardware shown in Figure 4.21
Latency = 9 cycles

Figure 4.22: Waveforms obtained in implementing the hardware shown in Figure 4.21
Figure 4.23: 16 tap direct form realization
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity filterl6 is
port
  (elk:in std_logic;
   xi:in std_logic_vector(7 downto 0);
   z:out std_logic_vector(15 downto 0) );
end filterl6;

architecture filter_arch of filterl6 is
  type sigtypel is array (0 to 15) of std_logic_vector(7 downto 0);
  signal x:sigtypel;
  type sigtype is array (0 to 15) of std_logic_vector (7 downto 0);
  constant h:sigtype :=("01111111","01111111","01111111",
                        "01111111","01111111","01111111",
                        "01111111","01111111","01111111",
                        "01111111","01111111","01111111",
                        "01111111","01111111","01111111",
                        "01111111","01111111","01111111");
begin
  process(elk)
  begin
    if clk=11' and elk'event then

      x(0)<=xi;

      for i in 1 to 15 loop
        x(i)<=x(i-1);
      end loop;

      z<=(h(0)*x(0)+h(1)*x(1)+h(2)*x(2)+h(3)*x(3)+x(4)*h(4) +x(5)*h(5)+x(6)*h(6)+x(7)*h(7)+h(8)*x(8)+h(9)*x(9)+h(10)*x(10)+h(11)*x(11)+x(12)*h(12)+x(13)*h(13)+x(14)*h(14)+x(15)*h(15));

    end if;
  end process;
end filter_arch;

Program 4.6: VHDL Program for hardware shown in Figure 4.23
Figure 4.24: Waveforms obtained in implementing the hardware shown in Figure 4.23  
Latency = 17 cycles
Figure 4.25: 16 tap Linear phase realization
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  

entity filter16 is  
port  
(clk:in std_logic;  
 xi:in std_logic_vector(7 downto 0);  
 z:out std_logic_vector(15 downto 0) );  
end filter16;  

architecture filter_arch of filter16 is  
type sigtypel is array (0 to 15) of std_logic_vector(7 downto 0);  
signal x:sigtypel;  
type sigtype2 is array (0 to 7) of std_logic_vector(7 downto 0);  
signal x:sigtype2;  
type sigtype is array (0 to 7) of std__logic_vector (7 downto 0);  
constant h:sigtype :={"01111111","01111111","01111111","01111111","01111111","01111111","01111111"};  
begn  
begin  
process(clk)  
begin  
if clk='1' and clk'event then  
x(0)<=xi;  
for i in 1 to 15 loop  
x(i)<=x(i-1);  
end loop;  
for i in 0 to 7 loop  
s(i)<=x(i)+x(15-i);  
end loop;  
z<= h(0)*s(0)+h(1)*s(1)+h(2)*s(2)+h(3)*s(3)+s(4)*h(4 )+s(5)*h(5)+s(6)*h(6)+s(7)*h(7);  
end if;  
end process;  
end filter_arch;  

Program 4.7: VHDL Program for hardware shown in Figure 4.25
Figure 4.26: Waveforms obtained in implementing the hardware shown in Figure 4.25

Latency = 18 cycles
Figure 4.27: 16 tap Linear phase realization with pipeline registers after multipliers
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity filter16 is
port
 (clk:in std_logic;
  xi:in std_logic_vector(7 downto 0);
  z:out std_logic_vector(15 downto 0) );
end filter16;

architecture filter_arch of filter16 is

type sigtype3 is array (0 to 7) of std_logic_vector(7 downto 0);
signal s:sigtype3;
type sigtype2 is array (0 to 7) of std_logic_vector(15 downto 0);
signal y:sigtype2;
type sigtype1 is array (0 to 15) of std_logic_vector(7 downto 0);
signal x:sigtype1;
type sigtype is array (0 to 7) of std_logic_vector(7 downto 0);
constant h:sigtype :="01111111","01111111","01111111",
  "01111111","01111111","01111111","01111111",
begin
process(clk)
begin
if clk='1' and clk'event then
 x(0)<=xi;
 for i in 1 to 15 loop
  x(i)<=x(i-1);
 end loop;
 for i in 0 to 7 loop
  s(i)<=x(i)+x(15-i);
 end loop;
 for i in 0 to 7 loop
  y(i)<=h(i)*s(i);
 end loop;
 z<= y(0)+y(1)+y(2)+y(3)+y(4)+y(5)+y(6)+y(7);
end if;
end process;
end filter_arch;

Program 4.8: VHDL Program for hardware shown in Figure 4.27
Figure 4.28: Waveforms obtained in implementing the hardware shown in Figure 4.27
Latency = 19 cycles
Figure 4.29: 16 tap Linear phase realization with two stage pipeline adders
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity filter16 is
port
  (clk:in std_logic;
   xi:in std_logic_vector(7 downto 0);
   z :out std_logic_vector (15 downto 0) );
end filter16;

architecture filter_arch of filter16 is
  type sigtype3 is array (0 to 7) of std_logic_vector(7 downto 0);
signal s:sigtype3;
type sigtype2 is array (0 to 9) of std_logic_vector(15 downto 0);
signal y:sigtype2;
type sigtype1 is array (0 to 15) of std_logic_vector(7 downto 0);
signal x:sigtype1;
type sigtype is array (0 to 7) of std_logic_vector(7 downto 0);
constant h:sigtype :=("01111111","01111111","01111111"
                      ,"01111111","01111111","01111111"
                      ,"01111111","01111111","");
begin
  process (clk)
  begin
    if clk='1' and clk'event then
      x(0)<=xi;
      for i in 1 to 15 loop
        x(i)<=x(i-1);
      end loop;
      for i in 0 to 7 loop
        s(i)<=x(i)+x(15-i);
      end loop;
      for i in 0 to 7 loop
        y(i)<=k(i)*s(i);
      end loop;
      y(8)<= y(0)+y(1)+y(2)+y(3);
      y(9)<= y(4)+y(5)+y(6)+y(7);
      z<= y(8)+y(9);
    end if;
  end process;
end filter_arch;

Program 4.9: VHDL Program for hardware shown in Figure 4.29
Figure 4.30: Waveforms obtained in implementing the hardware shown in Figure 4.29
Latency = 20 cycles
Figure 4.31: 16 tap Linear phase realization with three stage pipeline adders
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity filter16 is
port
(clk:in std_logic;
   xi:in std_logic_vector(7 downto 0);
   z:out std_logic_vector(15 downto 0));
end filter16;
architecture filter_arch of filter16 is
begin
process(clk)
begin
if clk='1' and clk'event then
   x(0)<=xi;
   for i in 1 to 15 loop
      x(i)<=x(i-1);
   end loop;
   for i in 0 to 7 loop
      s(i)<=x(i)+x(15-i);
   end loop;
   for i in 0 to 7 loop
      y(i)<=h(i)*s(i);
   end loop;
   y(8)<= y(0)+y(1);
   y(9)<= y(2)+y(3);
   y(10)<= y(4)+y(5);
   y(11)<= y(6)+y(7);
   y(12)<= y(8)+y(9);
   y(13)<= y(10)+y(11);
   z<=y(12)+y(13);
end if;
end process;
end filter_arch;

Program 4.10: VHDL Program for hardware shown in Figure 4.31
Figure 4.32: Waveforms for hardware shown in Figure 4.31
Latency = 21 cycles
Figure 4.3: 16 Tap Transposed form of FIR filter
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fir16 is
port ( clk:in std_logic;
       x:in std_logic_vector(7 downto 0);
       y:out std_logic_vector(15 downto 0)
);
end fir16;

architecture fir_arch of fir16 is
signal s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15:std_logic_vector(15 downto 0);
signal y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,y13,y14,y15:std_logic_vector(15 downto 0);
type sigtype is array (0 to 15) of std_logic_vector (7 downto 0);
constant hrsigtype := ("01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111","01111111");
begin
process(clk,x)
begin
if clk='1' and clk'event then
s0<=x*h(0);
s1<=x*h(1);
s2<=x*h(2);
s3<=x*h(3);
s4<=x*h(4);
s5<=x*h(5);
s6<=x*h(6);
s7<=x*h(7);
s8<=x*h(8);
s9<=x*h(9);
s10<=x*h(10);
s11<=x*h(11);
s12<=x*h(12);
s13<=x*h(13);
s14<=x*h(14);
s15<=x*h(15);
y15<=s15;
y14<=y15+s14;
y13<=y14+s13;
y12<=y13+s12;
y11<=y12+s11;
end if;
end process;
end fir_arch;

Program 4.11: VHDL Program for hardware shown in Figure 4.33
y10<=y11+s10;
y9<=y10+s9;
y8<=y9+s8;
y7<=y8+s7;
y6<=y7+s6;
y5<=y6+s5;
y4<=y5+s4;
y3<=y4+s3;
y2<=y3+s2;
y1<=y2+s1;
y<=y1+s0;
end if;
end process;
end fir_arch;

Program 4.11: VHDL Program for hardware shown in Figure 4.33
References


