P-3449

# FIR FILTER SYNTHESIS ALGORITHMS FOR MINIMISING NUMBER OF ADDERS

By

ANUSREE P DEV

Reg No: 0920106001

*of*

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE - 641 049.

(An Autonomous Institution affiliated to Anna University of Technology, Coimbatore)

A PROJECT REPORT

*Submitted to the*

FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING

*In partial fulfillment of the requirements for the award of the degree*

*of*

MASTER OF ENGINEERING

IN

APPLIED ELECTRONICS

APRIL 2011

## BONAFIDE CERTIFICATE

Certified that this project report titled "**FIR FILTER SYNTHESIS ALGORITHMS FOR MINIMISING NUMBER OF ADDERS**" is the bonafide work of **Ms. ANUSREE P DEV (0920106001)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]

**Dr. RAJESWARI MARIAPPAN Ph.D.,**

**PROJECT GUIDE**

[illegible]

**Dr. RAJESWARI MARIAPPAN Ph.D.,**

**HEAD OF THE DEPARTMENT**

The candidate with **University Register No. 0920106001** was examined by us in Project Viva-Voce examination held on 21-04-2011

[illegible] 21/4/11

**INTERNAL EXAMINER**

[illegible]

**EXTERNAL EXAMINER**

## ACKNOWLEDGEMENT

I would like to express my thanks and appreciation to the many people who have contributed to the successful completion of this project. Firstly I express my profound gratitude to our Chairman **Padmabhusan Arutselvar Dr. N. Mahalingam B.Sc., F.I.E.,** and Co-Chairman **Dr. B. K. Krishnaraj Vanavarayar, B.Com., B.L.,** for giving this opportunity to pursue this course.

I would like to thank **Dr. J. Shanmugam, Ph.D.,** Director. for providing me an opportunity to carry out this project work.

I would like to thank **Dr. S. Ramachandran Ph.D.,** Principal. for providing me an opportunity to carry out this project work.

My heartfelt thanks to **Dr. Rajeswari Mariappan Ph.D.,** Head of the Department and my internal guide, Electronics and Communication Engineering, who gave her continual support and technical guidance for me throughout the course of this project.

I would like to thank **Ms. R. Latha M.E.,** Associate Professor and Project Coordinator. for her contribution and innovative ideas at various stages of the project and for her help to successful completion of this project work.

I express my sincere gratitude to my family members. friends and to all my staff members of Electronics and Communication Engineering department for their support throughout the course of my project

## ABSTRACT

Finite-impulse response (FIR) filters are critical to most DSP applications; an energy-aware filter design helps significantly in reducing the total power intake of the system. Existing conventional scheme generate several partial products which increases the number of adders thus in turn increasing the hardware complexity. Several algorithms are implemented in the architectures. Using CSE (Common Sub expression Elimination) algorithm the number of operations are reduced after decomposing multiplications into shifts and additions. Canonical Signed Digit (CSD) form is used for the coefficients to minimize the number of additions. When using LCCSE (Level Constrained CSE) algorithm the number of adder levels (ALs) required to compute each of the coefficient outputs is constrained. This is achieved by restricting the "important filter coefficients" to a less number of computation steps than the maximum allowed in a CSE-based filter implementation. The main objective of this paper is to design efficient FIR filter architectures with the help of these algorithms which favor minimum number of adders. Using the BCSE (Binary Common Sub expression Elimination) technique two reconfigurable architecture of low complexity FIR filters are proposed, namely constant shifts method and programmable shift method. The proposed architecture offer good area, power reductions and speed improvement.The architecture is designed using VHDL and functional verification is done by using Modelsim.

## TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

## LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| FIR | --- | Finite Impulse response |
| CSD | --- | Canonical signed digit |
| CSE | --- | Common sub expression Elimination |
| LCCSE | --- | Level constraint common sub expression Elimination |
| BCSE | --- | Binary Common Sub expression Elimination |
| HCS | --- | Horizontal Common Sub expression |
| VCS | --- | Vertical Common Sub expression |
| LUT | --- | Look Up Tables |
| FPGA | --- | Field Programmable Gate Array |
| AL | --- | Adder Levels |
| LOs | --- | Logical Operators |
| CSM | --- | Constant Shift Method |
| PSM | --- | Programmable Shift Method |
| MCM | --- | Multiple Constant Multiplications |

# CHAPTER 1
# INTRODUCTION

## 1.1 NEED FOR LOW POWER TECHNIQUE

With explosive growth in the demand of portable computing and wireless communication systems, power dissipation is becoming an increasing concern. Higher power consumption reduces the battery lifetime of portable devices, affects device reliability, and increases cost. Early works have focused on replacing multiplications by decomposing them into simple operations such as addition, subtraction and shifting. As the coefficients of an application specific filter are constant, the decomposition is more efficient than employing multipliers [1]. The FIR filter computations play an important role in the DSP systems. The present generation embedded systems have stringent requirements on performance and power consumption.

Many embedded systems employ DSP algorithms for communications, image processing, video processing etc, which are very compute intensive. Custom hardware implementation of these computation intensive DSP kernels is a good solution to meet the requirements for latency and power consumption. The problem of designing FIR filters has received a great attention during the last decade, as the filters are suffering from a large number of multiplications, leading to excessive area and power consumption even if implemented in full custom integrated circuits.

The modern digital communication and multimedia computation require high performance and low complexity digital signal processing systems. Therefore, low-power methods are necessary for the design of these DSP-based systems. Finite-impulse response (FIR) filters are unavoidable occurrences in most of real time applications[2]. So designing an efficient filter concept can bring about significant power reduction which can also bring about several advantages which includes reduced hardware complexity. Therefore, several methods have to be adopted to reduce computational complexity.

## 1.2 OVERVIEW

Finite-impulse response (FIR) digital filters are frequently used in digital signal processing by virtue of stability and easy implementation. It is known that the common sub expression elimination (CSE) methods based on canonical signed digit (CSD) coefficients and level-constrained common sub expression-elimination algorithm (LCCSE) produce low complexity FIR filter coefficient multiplier. Using CSE, multiple occurrences of identical bit patterns that are present in the CSD representation of coefficients are identified and these redundant multiplications are eliminated. Using LCCSE the number of adder levels (ALs) required to compute each of the coefficient outputs are constrained. Using this algorithm the adders get reduced and so power consumption is reduced .BCSE technique is more advantageous over CSE and LCCSE algorithms and BCSE is used in proposed architecture which offers better results in terms of speed, power and area requirements.

## 1.3 MOTIVATION OF THE WORK

FIR digital filters find extensive applications in every field of application. The DSP algorithms contain a large number of multiplications with constants. Decomposing these constant multiplications into shifts and additions leads to an efficient hardware implementation. Finding common sub expressions in the set of additions further reduces the complexity of the implementation. The common sub expressions correspond to the common partial products formed during the multiplication of the variable with the constants. The complexity of FIR filters in this case is dominated by the number of additions/subtractions used to implement the coefficient multiplications. To reduce the complexity, the coefficients can be restricted to powers-of-two or expressed in canonical signed-digit (CSD) or graph representation to minimize the number of additions/subtractions required in each coefficient multiplication. Taking into account of this technique a new architecture is proposed which can contribute much to real time signal processing.

## 1.4 OBJECTIVE OF THE WORK

- To develop an efficient filter architecture using the binary common sub expression elimination technique (BCSE) which can bring about significant power reduction and decreased hardware complexity.
- The architecture is verified for its functionality using the existing CSE and LCCSE algorithms where the Canonical Signed Digit (CSD) form is used for the coefficients to minimize the number of additions.

- ➢ The proposed CSM and PSM architecture is designed using VHDL code and simulated using Modelsim. Simulation results are used for performance comparison.

- ➢ Performance parameters taken for analysis are gate count, delay and power and these parameters are compared with the existing algorithms.

## 1.5 INTRODUCTION TO VHDL

VHDL is the main source that is to be needed to go through the project. In the mid-1980's the U.S. Department of Defence and the IEEE sponsored the development of this hardware description language with the goal to develop very high-speed integrated circuit. It has become now one of industry's standard languages used to describe digital systems. The other widely used hardware description language is Verilog. Both are powerful languages that allow describing and simulating complex digital systems. A third HDL language is ABEL (Advanced Boolean Equation Language) which was specifically designed for Programmable Logic Devices (PLD). ABEL is less powerful than the other two languages and is less popular in industry. This proposed work deals with VHDL, as described by the IEEE standard. A HDL program mimics the behaviour of a physical, usually digital, system. It also allows incorporation of timing specifications (gate delays) as well as to describe a system as an interconnection of different components. VHDL allows one to describe a digital system at the structural or the behavioural level. The behavioural level can be further divided into two kinds of styles: Data flow and Algorithmic.

### 1.5.1 Basic Structure of a VHDL File

A digital system in VHDL consists of a design entity that can contain other entities that are then considered components of the top-level entity. Each entity is modelled by an entity declaration and an architecture body. The entity declaration is considered as the interface to the outside world that defines the input and output signals, while the architecture body contains the description of the entity and is composed of interconnected entities, processes and components, all operating concurrently, as schematically shown in Figure 1.1 below. In a typical design there will be many such entities connected together to perform the desired function.

VHDL uses reserved keywords that cannot be used as signal names or identifiers. Keywords and user-defined identifiers are case insensitive. Lines with comments start with two adjacent hyphens (--) will be ignored by the compiler.

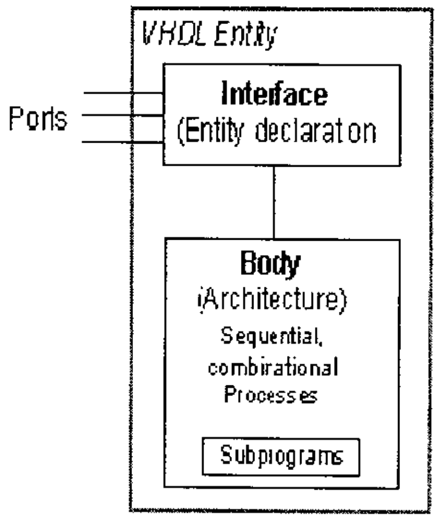


**Fig 1.1 A VHDL entity with entity and architecture description**

VHDL also ignores line breaks and extra spaces. VHDL is a strongly typed language which implies that one has always to declare the type of every object that can have a value, such as signals, constants and variables.

## 1.5.2 VHDL Coding Styles and Synthesis

The style of writing VHDL code is very important. Effective VHDL coding techniques can make all the difference between designs that meet tough synthesis targets and verification schedules versus those requiring re-spins, Vector independent code is important. Code reuse depends on the code being portable between different synthesis tools. Each synthesis tool vendor accepts a slightly different subset of VHDL. Hence the same code gives different results on different synthesis tools. After the design has been successfully simulated, the synthesis stage converts the text-based design into a gate level Net list. This Net list is a non-readable file that describes the actual circuit to be implemented at a very low level.

### 1.5.3 VHDL DESIGN FLOW

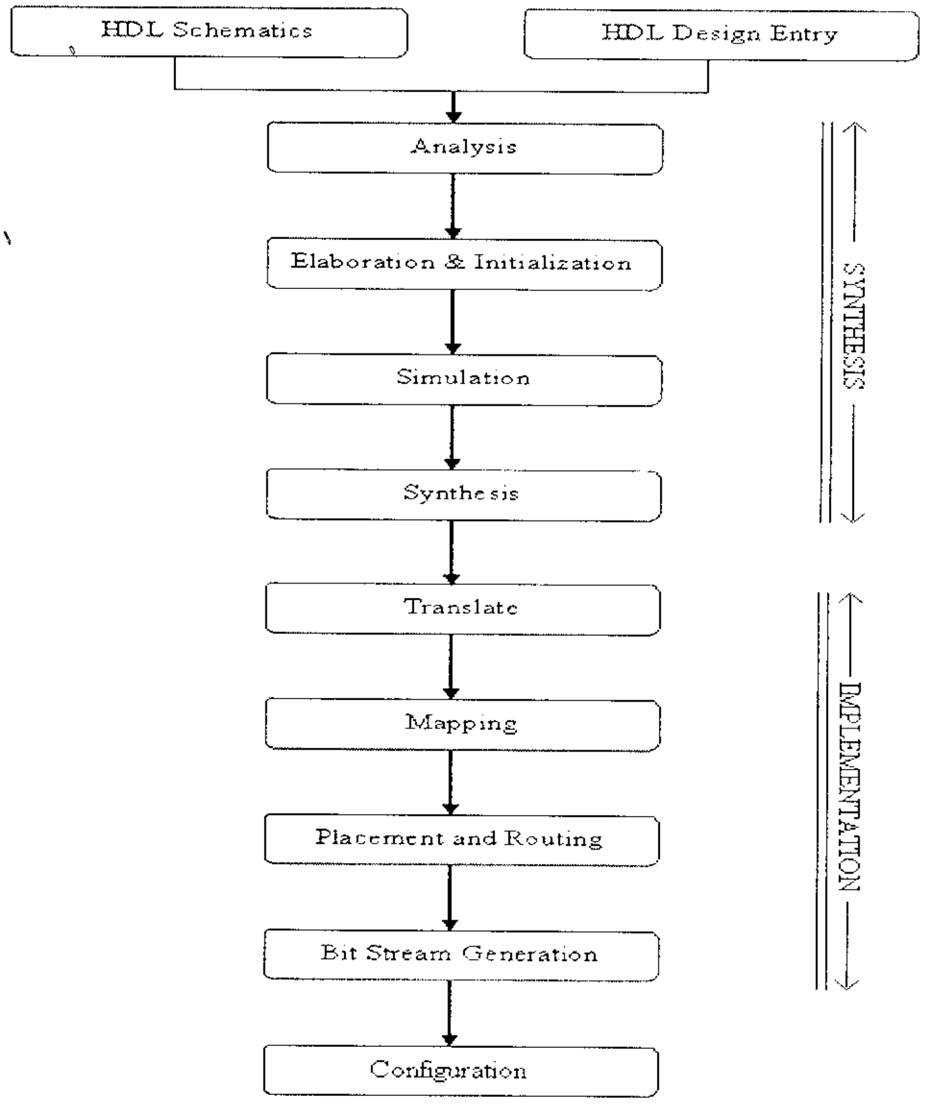


**Fig 1.2 VHDL Design Flow**

## 1.6 DESIGN FLOW DURING SYNTHESIS

The detailed process of synthesis can be described as a sequence of the following events:

- Analysis,
- Elaboration

• Initialization

• Simulation,

• Synthesis

### 1.6.1 Analysis

It consists of compiling the .vhd(dot vhd) file and storing it in a design library. During compilation, the compiler checks the syntax and semantics of the .vhd file and converts it into an intermediate form, which is stored in a specific design library. Finally, during analysis, the analyzer maps the equation based form of design to a standard cell library.

### 1.6.2 Elaboration

During the elaboration phase, flattening of the design hierarchy takes place. Elaboration involves the following tasks:

• Initialization of signals, processes and variables.

• Components are bound to architectures,

• Memory is allocated for storage of various objects.

The final result of the elaboration process is a flat collection of the signal nets and processes.

### 1.6.3 Initialization

Explicit signals of ports and signals of architectures are initialized, resolved and assigned values.

### 1.6.4 Simulation

Simulation follows Initialization. During simulation execution of the processes takes place.

• Signals declared implicitly by attributes are assigned values.

• Processes are executed until they suspend.

### 1.6.5 Synthesis

After the design has been successfully simulated, the synthesis stage converts the text-based design into a gate level Net list. This Net list is a non-readable file that describes the actual circuit to be implemented at a very low level.

- Post Synthesis Design Flow

Once synthesis is complete the design is implemented on the chip by sequentially following the phases:

- Translation,
- Mapping,
- Placement and Routing, (PAR).

During the implementation phase the design Net list and normally a constraints file is used to recreate the design using the available resources within the FPGA.

- Translate

This stage prepares the synthesized design for use within the FPGA. It checks the design and ensures that the Net list is consistent with the chosen architecture. The result is stored in a file in a tool specific binary format that describes the logic design in terms of design primitives such as latches, Flip-Flops and Function Generators.

- Mapping

In this stage, the design is distributed to the resources in the FPGA. Thus, mapping assigns a design's logic elements to the specific physical elements such as CLBs and IOBs that actually implement logic functions in the device.

- Place and Route (Par)

During the placement phase of the PAR, the design blocks created during mapping are assigned specific locations in the FPGA. The Routing phase assigns the interconnect paths in the FPGA.

- Bit Stream Generation

The output obtained from the Place and Route phase is converted into a bit file, which is used to configure the FPGA.

- Configuration

During configuration, the bit file is downloaded from the programming file into the SRAM of the target device.

- Synthesis Tools

Synthesis tools play a vital role in digital electronic design automation. It accepts the conceptual hardware description language design definition as the input and provides the corresponding physical or logical representation for the targeted silicon device as an output. To produce highly optimized results with a fast compile time and quick turnaround time a state of the art synthesis engine is required. The synthesis engine must be integrated with the physical implementation tool and should have the ability to proactively meet the design timing requirements. Most of the modern synthesis tools are robust, reliable and produce high quality optimized circuits from the initial Register Transfer Level, (RTL), and VHDL designs.

All of the VHDL features such as Access Types and Files, which are abstractions that are too unconnected with the hardware, cannot be synthesized. Therefore, each synthesis tool builder defines a subset of the VHDL language for input to the product. The size of the VHDL subset forms the basic criteria for choosing the synthesis tool.

- Benefits Of Logic Synthesis

Logic synthesis increases designer productivity. A few lines of RTL code is equivalent to hundreds of logic gates. Therefore, designs can be created and verified more quickly and efficiently at the RTL level.

Different Design Options: High level constructs available in VHDL and friendly graphical interface tools help the designer to experiment with the design in order to obtain the best possible solution by using the various available options.

## 1.7 SOFTWARES USED

- ModelSim 6.4
- Xilinx ISE 9.2i

## 1.8 ORGANISATION OF THE REPORT

**Chapter 2** discusses the overview of existing methods of complexity reduction.

**Chapter 3** deals with the proposed architectures which use the BCSE method.

**Chapter 4** deals with simulation reports for CSM and PSM architecture implementations.

**Chapter 5** deals with conclusion and future work.

# CHAPTER 2

# EXISTING ALGORITHMS FOR FILTER DESIGN

## 2.1 GENERAL OVERVIEW

Reduction of hardware complexity directly relates to lower power consumption; therefore, several methods have been reported to reduce computational complexity. One popular approach for complexity mitigation is the replacement of multiplications of a FIR filter by addition and shift operations.

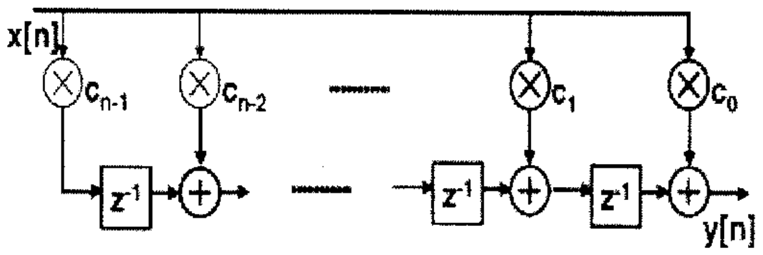


**Fig 2.1 N tap transposed FIR filter**

Further power optimization is obtained by minimizing the number of adders required to implement the multiplications. For this purpose, all coefficients of a transposed-form FIR filter are considered as a whole and replaced by a single multiplier block, as shown in Figure. 2.2. The redundancy across the coefficients in the multiplier block is then exploited to share computations and reduce the number of adders.

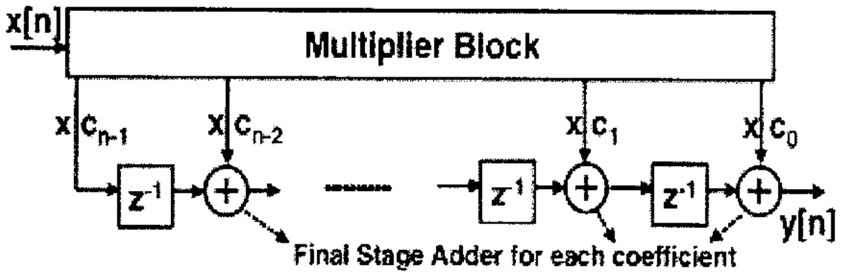


**Fig 2.2 Coefficient multiplications replaced by a Multiplier block**

P-3449

To effectively enable such sharing, a variety of methods have been developed.

- CSE method
- LCCSE method

To reduce complexity and critical path length, CSE-based filter designs are often used. A variety of CSE methods have been proposed. Most of these implementations take filter coefficients in canonical-sign-digit (CSD) format, where coefficients are represented with a minimum number of nonzero bits. The redundancy present in the CSD coefficients is then exploited to share some of the adders to further reduce the hardware complexity, thereby reducing power consumption.

## 2.2 CANONICAL SIGNED DIGIT FORM

An Encoding a binary number such that it contains the fewest number of non-zero bits is called Canonical Signed Digit .A CSD representation is a kind of sum of signed power of two representations. Unlike binary numbers, that is expressed using only 0 and1, but the CSD representation use 0, 1 and -1. It is signed digit number system that minimizes the number of non-zero digits. It can reduce the number of partial product additions in a hardware multiplier. They are successful in implementing multipliers with less complexity. Since the complexity of the multipliers is typically estimated through the number of non-zero elements, which can be reduced by using signed digit numbers.

For each tap in a CSD representation filter, the data tap value is shifted by the number of bit positions corresponding to the position of each non-zero bit in the coefficient for that tap. The resulting shifted data tap values are then added. This is done for every coefficient in the filter. A primary consequence of this is that a larger adder i e. one with more data inputs is required than would be needed for a conventional FIR filter implementation. However, this is still desirable since no multipliers are used. The high hardware cost of multipliers makes the CSD implementation especially attractive for very long digital filters. Such filters are often needed, for example, in communication particularly for the demodulation of digital data signals.

## 2.3 COMMON SUB EXPRESSION ELIMINATION

The objective of CSE algorithms is to find bit patterns that are common in more-than-one coefficients to reduce the total adder cost by sharing the common bit patterns among the coefficients.CSE has been utilized as a very powerful tool in FIR filter design to reduce the number of arithmetic units (adders and shifters). Sub-expression can be any bit patterns within the CSD coefficients.

The concept of CSE can be illustrated with the help of the following example. Consider two functions F1 and F2, where F1 = 13X and F2 = 29X. Both F1 and F2 can be represented in the following manner: F1 = X + 4X + 8X = X +X << 2 + X << 3 and F2 = X + 4X+ 8X + 16X = X +X << 2 + X << 3 + X << 4, where "<<" means bitwise left shift. Both expressions F1 and F2 have some common terms D = X + X << 2 + X << 3. Therefore, F1 and F2 can be rewritten as F1 = D and F2 = D + X << 4. Reusing D in both expressions reduces the computation overhead and the number of adders required to implement both expressions. Significant power savings can be achieved by reducing number of adders using CSE (only three adders in the CSE-based implementation compared with five adders in the unshared case)

When CSE is not used the process requires 5 adders

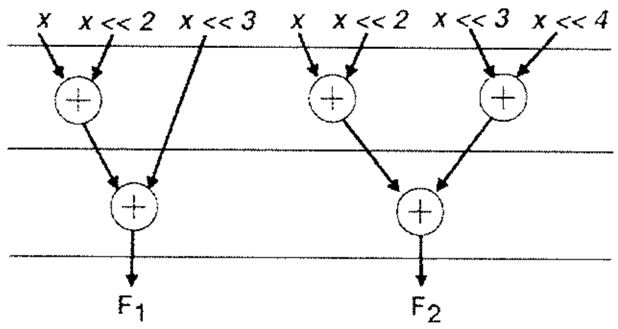


**Fig 2.3 Multiplication without CSE**

This diagram clearly shows the reduced number of adders when using CSE method.It is found that the adders get reduced from five to three when the same output is calculated by using CSE method.Reduction in number of adders will reduce hardware complexity and will bring about reduced power consumption.

The goal of CSE is to identify multiple occurrences of identical bit patterns that are present in the CSD representation of coefficients, and eliminate these redundant multiplications. A modification of the 2-bit CSE technique for identifying the proper patterns for elimination of redundant computations and to maximize the optimization impact was taken into consideration. The technique was modified to minimize the logic depth (LD) (LD is defined as the number of adder-steps in a maximal path of decomposed multiplications) and thus to improve the speed of operation.

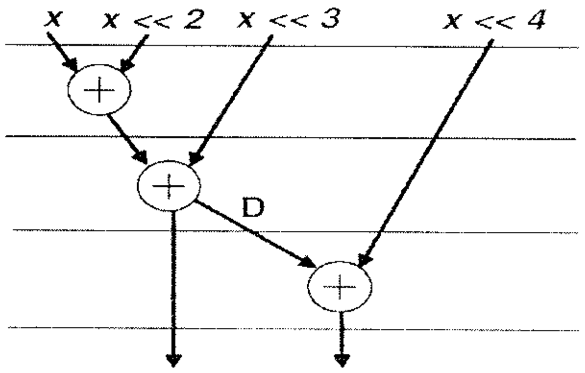


**Fig 2.4 Multiplication with CSE**

The diagrammatic representation clearly shows the reduced steps when conventional method is replaced by CSE method. A general example showing sub expression Elimination procedure is given as follows.

Y=1010101

Y= x +x<<2 + x<<4 + x<<6

Y = x + x<<2 +(x + x<<2) <<4

S = x + x<< 2

Y= S + S<<4

The CSE techniques utilize the CSs (common sub expressions) that occur within the CSD representation of the filter coefficients called HCSs (horizontal common sub expressions) and that occur among the adjacent coefficients called VCSs (vertical common sub expressions) to eliminate redundant computations. A brief review of the HCSE and VCSE techniques is given below.

### 2.3.1 Horizontal CSE (HCSE)

The optimization procedure targets the minimization of the multiplier block area. After expressing the coefficients in a canonical signed digit (CSD) format; in order to reduce the total number of nonzero bits (thus also the additions/subtractions necessary), an add shift expansion is performed. The idea of HCSE is to identify the bit patterns that are present in the coefficient set more than once. Since it is sufficient to implement the calculation of the multiple identical expressions only once, the resources necessary for these operations can be shared. The pattern is present twice, so an optimized structure can be implemented instead of the original one. The second occurrence of the pattern is removed, and only the result is used for the further calculation. In general, the goal of HCSE can be defined as follows.

1) Identify multiple patterns in the coefficient set.

2) Remove these patterns and calculate them only once.

HCSE utilizes CSs that occur within each coefficient to eliminate redundant computations. The coefficient $h_k$ = 0.10-1010101010010-1 is used as an example to illustrate the HCSE method. In direct implementation (i.e., implementation of the multiplier using shifts and adds without using CSE), the filter tap output is

$$y_k = 2^{-1}x_1 - 2^{-3}x_1 + 2^{-5}x_1 + 2^{-7}x_1 + 2^{-9}x_1 + 2^{-11}x_1 + 2^{-14}x_1 - 2^{-16}x_1 \qquad (1)$$

where $x_1$ is the input signal.

It requires seven LOs (adders and/or subtractors) to implement (1). The bit patterns [1 0 1] and [1 0 −1] are repeated twice in $h_k$, which can be expressed as CSs ($x_2 = x_1 + 2^{-2}x_1$ and $x_3 = x_1 - 2^{-2}x_1$, respectively). Using CSs, the output (1) can be expressed as

$$y_k = 2^{-1}x_3 + 2^{-5}x_2 + 2^{-9}x_2 + 2^{-14}x_3 \qquad (2)$$

Note that only five LOs [two LOs for CSs $x_2$ and $x_3$, and three LOs for (2)] are needed for HCSE implementation, which is a saving of two LOs when compared to direct implementation.

### 2.3.2 Vertical CSE (VCSE)

The VCSE utilizes CSs that exist across adjacent coefficients to eliminate redundant computations. The VCSE technique can be illustrated using the four-tap symmetrical filter coefficient set. The coefficient multipliers are realized by employing vertical CSs of [1 1] and [1 −1] .The VCS of [1 1] can be expressed as $x_4 = x_1 + x_1[-1]$, and the VCS of [1 −1] can be expressed as $x_5 = x_1 - x_1[-1]$, where $x_1[-1]$ represents the input $x_1$ delayed by one unit. By using VCSs, the output $y_0$ can be expressed as

$$y_0 = 2^{-1}x_4 + 2^{-3}x_5 - 2^{-6}x_4 + 2^{-8}x_1 \qquad (3)$$

The output $y_0$ requires three LOs (logical operators) for (3) and two LOs for $x_4$ and $x_5$, with a total of five LOs, whereas direct implementation would require six LOs. In general, VCSE offers reduction of LOs due to the occurrence of many vertical CSs in the coefficient set.

However, CSD-based VCSE fails to exploit the symmetry of coefficients in implementing the filter. For the symmetric part of (3), by using the same VCSs, the expression for the output of coefficients h2 and h3 is given by

$$y_2 = 2^{-1}x_4[-2] - 2^{-3}x_5[-2] - 2^{-6}x_4[-2] + 2^{-8}x_1[-3] \qquad (4)$$

There are two issues (constraints) related to symmetry exploitation in VCSE using CSD representation of filter coefficients, first, due to the differences in signs of the second terms in (3) and (4), and second, delay differences for the fourth terms in (3) and (4). Hence, (4) cannot be directly obtained from its symmetric part (3) by a simple delay operation; instead, extra LOs are needed to compensate the sign and delay differences. This requirement of extra LOs poses constraints in reducing the number of LOs in CSD-based VCSE method. In binary based approach, only the second problem of delay difference will occur.

## 2.4 COMMON SUBEXPRESSION ELIMINATION PROCEDURE

A conventional structure generally used when implemented, have more number of adders. This brings about increased hardware complexity of the system. A conventional structure is implemented and its output is verified. When implementing without CSE, most of the filter Coefficients are taken in canonical-sign-digit (CSD) format, where coefficients are represented with a minimum number of nonzero bits .When CSE is implemented numbers of operations are reduced. Experimental results show the superiority of CSE over conventional techniques for common sub expression elimination.

Conventional methods for finding common sub expressions rely on finding common digit patterns in the set of constants multiplied by a single variable. The common sub expressions correspond to the common partial products formed during the multiplication of the variable with the constants. Finding all possible common digit patterns can extract all possible common sub expressions when all the constant multiplications are with a single variable such as found in the transformed form of FIR digital filters.

## 2.5 CHOICE OF SUBEXPRESSION

The choice of which sub expressions to choose at each iteration is not dependent only on the number of times an expression occurs, but rather on the expected reduction in number of operators.

**2.5.1 Estimation of Number of Adders:** The number of adders that will be eliminated by a common sub expression occurring $N$ times is $N$ - 1. The only trick in making sure that the count is accurate to take care that overlapping sub expressions are not counted twice. In other words, if a given term occurs twice in sub expressions of the same type, (once as first member and once as second member) then it should not be counted twice. The number of times each sub expression occurs is counted by an exhaustive search. This requires a time complexity of O(n2) where n is the total number of nonzero terms.

This figure shows the general format of a linear system and then its decomposition by means of shift and add operations and then arriving at common sub expression procedure.

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 5 & 7 \\ 4 & 12 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

$$\begin{aligned} 5 &= 0101b \\ 4 &= 0100b \\ 7 &= 0111b \\ 12 &= 1100b \end{aligned}$$

(a) Example Linear System

$$\begin{aligned} Y_1 &= X_1 + X_1{<<}2 + X_2 + X_2{<<}1 + X_2{<<}2 \\ Y_2 &= X_1{<<}2 + X_2{<<}2 + X_2{<<}3 \end{aligned}$$

(b) Decomposing constant multiplications into shifts and additions

$$\begin{aligned} D_1 &= X_2 + X_2{<<}1 \\ Y_1 &= X_1 + X_1{<<}2 + D_1 + X_2{<<}2 \\ Y_2 &= X_1{<<}2 + D_1{<<}2 \end{aligned}$$

(c) Extracting common bit patterns among constants multiplying a single variable

$$\begin{aligned} D_2 &= X_1 + X_2 \\ D_1 &= D_2 + X_2{<<}1 \\ Y_1 &= D_1 + D_2{<<}2 \\ Y_2 &= D_1{<<}2 \end{aligned}$$

(d) Extending common subexpressions to include multiple variables

**Fig 2.5 example showing improvement obtained by using CSE algorithm**

Consider the linear system shown in Figure 2.5 a, which has two variables X1 and X2. Assume the binary representation for constants. The constants in column 1 (5 and 4) are multiplied with X1 and the constants in column 2 (7 and 12) are multiplied with variable X2. Decomposing the constant multiplications into shifts and additions we have an implementation with six shifts and six additions (Figure 2.5b). Finding common bit patterns in the constants multiplying the same variable, the pattern "11" is detected between the constants 7(0111) and 12 (1100), which multiply variable X2. Extracting that common sub expression, have an implementation with five shifts and five additions (Figure 2.5c). If the common sub expressions are extended to include multiple variables, then an implementation with just three additions and three shifts (Figure 2.5d) can be made. Therefore by extending common sub expressions to include multiple variables, the number of additions has reduced compared to methods which extract common sub expressions including only one variable.

## 2.6 CONCEPTUAL DIAGRAM SHOWING CSE

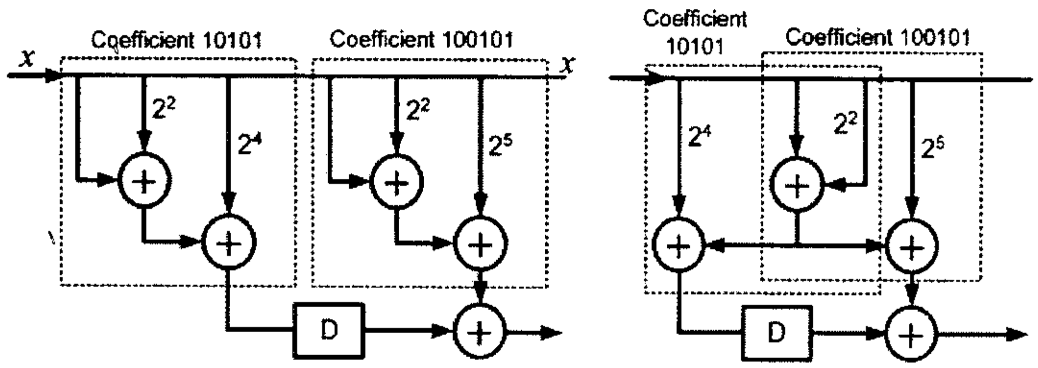


**Fig 2.6(a) each coefficient is independently implemented by shifters and adders**

**(b) Common sub expression extracted and implemented.**

One major problem in CSE techniques is the selection of representation of filter coefficients. When the multipliers in filters are implemented using shift and add operations, the number of adders is directly proportional to the number of nonzero digits present in the filter coefficients. Thus, number systems with fewer numbers of nonzero digits are widely employed for the representation of filter coefficients compared to a conventional binary system. Among such number representations, the CSD representation is one of the most popular ones. If an MSD representation was used, it always resulted in the least number of nonzero digits. A modified MSD representation was used to implement the FIR filters. However, the main drawback of the MSD representation is that it can have more than one representation format for the same decimal or binary representation. The flexibility of MSD representation does not have a significant impact in the optimization solutions. The CSE methods employing these alternate number systems such as CSD or MSD suffer from the drawback that the symmetry of FIR filter coefficients cannot be completely exploited when the bits in VCSs are of opposite sign. As a result, additional LOs are required to obtain the symmetric part of the coefficients when more than one VCSs with bits of opposite sign exist. Another problem with these alternative number systems such as CSD and MSD compared to binary representation is that they involve subtraction operation, which can be a constraint in realizing reconfigurable filters. In addition, if dynamic reconfigurability is required, on-the-fly conversion of filter coefficient binary values into CSD or MSD may not be possible in real time. In addition to this, there is an additional memory requirement when storing the signed bits in case of CSD or MSD representations for reconfigurable filters.

## 2.7 LEVEL CONSTRAINED CSE ALGORITHM

The LCCSE algorithm utilizes the CSE method. While most CSE algorithms are focused on minimizing the number of adders without considering the delay of the critical path, the LCCSE algorithm can constrain the number of adders in the critical path. Hence, this LCCSE method can perform trade-off between the critical path delay (ALs) and the hardware overhead (the number of adders) in a direct way, which is an essential property for design methodology. The same number of adders obtained by the standard CSE technique cannot be obtained in LCCSE implementation. While in the standard CSE implementation only the maximum levels or the maximum number of adders in the critical path would be specified, in LCCSE, coefficients are constrained to certain levels based on their sensitivities. However, filters designed with LCCSE are robust to process variations and allow for significant power savings.

## 2.8 LCCSE PROCEDURE

As mentioned earlier, the LCCSE algorithm utilizes the CSE method, which considers not only resource sharing among the filter coefficients but also the length of the critical path in the multiplier block. The LCCSE algorithm also takes into consideration the level constraints of each coefficient based on its sensitivity. The difference of LCCSE from the previous CSE method is the fact that different level constraints can be given for each coefficient so that tighter timing bounds can be asserted for more important coefficients. Only in one case that a single level constraint (FAL) is specified for all coefficients, LCCSE yields results identical to that of the conventional CSE. Before going into the details of the proposed LCCSE technique, some notations used to design the algorithm can be discussed.

**2.8.1 Decomposed Set (DS)**: the set of absolute values of CSD numbers that have been decomposed as a sum of other CSD numbers. For example, 101001 (25) can be decomposed as (101 << 3) + 1 or 100001 − (1 << 3) ($25 = 3 \times 2^3 + 1 = 33 - 1 \times 2^3$).

**2.8.2 Un decomposed Set (UDS):** the set of absolute values of CSD numbers waiting to be decomposed with other CSD numbers. Initially, {UDS} contains all the filter coefficients.

**2.8.3 All possible combination set (APCS):** the set of candidate CSD numbers which can be utilized to decompose CSD numbers in {UDS}. This set is constructed in the following manner. All the possible combinations of nonzero terms of a coefficient are extracted, and the extracted CSD numbers are continuously right-shifted until they become odd. The absolute values of these numbers (except the value of one) are added to {APCS}. For instance, from a coefficient "101001" (25), we can extract six CSD numbers {100001, 101000, 1001, 100000, 1000, 1}, which are right-shifted to become {100001, 101, 1001, 1, 1, 1}, respectively. Finally, their absolute values, except one, are added to {APCS} so that {APCS} has the following three terms {101(3), 100001(33), |1001| (7)}. This procedure is repeated for each filter coefficient. In the final step, any CSD numbers which belong to both {UDS} and {APCS} are erased from {APCS}. After {APCS} is constructed, all the coefficients can be expressed as combinations of some elements of {APCS} U {1}.

The basic approach of the LCCSE algorithm is as follows.

1) Throughout the procedure, {UDS} contains CSD numbers which must be decomposed with other CSD numbers to complete the filter implementation. Try to decompose elements of {UDS} with other CSD numbers in {DS} U {UDS} U {APCS} U {1}. The LCCSE routine ends when {UDS} becomes empty.

2) Once a number in {UDS} is decomposed with a pair of other CSD numbers (sub expressions), it is removed from {UDS} and added to {DS}. If elements of {APCS} are chosen as sub expressions for this decomposition, they are moved from {APCS} to {UDS}, waiting to be decomposed.

3) When sub expressions for decomposition are chosen, avoid using elements of {APCS} as much as possible to minimize hardware overhead. Since every decomposition requires one additional adder, using elements of {APCS} increases the number of adders required for the filter implementation.

4) Perform a timing check whenever one CSD number is decomposed. The procedure to check the level constraints is more complicated in LCCSE than in the conventional CSE [7] because in LCCSE, each coefficient is allowed to have a different level constraint. There are no sign problems as there are no negative bits in binary. This makes binary-based CSE method more efficient for VCSE compared to the CSD-based approach. Filters used in wireless communication receivers are mainly linear phase (symmetric coefficients) to avoid phase distortions. Hence, binary-based VCSE is more feasible and will result in better reduction of adders compared to CSD-VCSE for these filters.

# CHAPTER 3
# PROPOSED ARCHITECTURE USING BCSE TECHNIQUE

## 3.1 BASIC CONCEPT OF BINARY SUBEXPRESSION ELIMINATION

A method based on binary representation of the filter coefficients is used to design the architecture and the method is called BSE. The basic idea in this method is to search and eliminate the redundant horizontal, vertical, and sub expressions that exist in the binary representation of filter coefficients. The most of the CSE methods make use of the CSD representation of the filter coefficients. This is because the number of nonzero bits in the CSD representation is fewer than that in corresponding binary representation. It is shown that the number of nonzero digits is reduced for CSD representation compared to normal two's complement form. As the number of nonzero bits in CSD is few, only fewer adders are needed to realize the coefficient multiplier compared to binary representation. However, it should also be noted that, as the number of nonzero bits are minimum, the potential of the CSD-based CSE technique to reduce the number of adders by forming CSs is less than that of binary.

The cost of BCSE method mainly depends on three factors: the total number of nonzero bits in the coefficient set, the number of CSs that can be formed from the nonzero bits, and the number of unpaired bits (bits that do not form CSs).

## 3.2 REVIEW OF BCSE METHOD

A review of BCSE algorithm which deals with the elimination of redundant binary common sub expressions (BCSs) that occur within the coefficients is done here. The BCSE technique focuses on eliminating redundant computations in coefficient multipliers by reusing the most common binary bit patterns (BCSs) present in coefficients .An n-bit binary number can form $2n - (n + 1)$BCSs among themselves.

For example, a 3-bit binary representation can form four BCSs, which are [0 1 1], [1 0 1], [1 1 0], and [1 1 1]. These BCSs can be expressed as[0 1 1] = $x_2 = 2^{-1}x + 2^{-2}x$, [1 0 1] = $x_3 = x + 2^{-2}x$, [1 1 0] = $x_4 = x + 2^{-1}x$, and [1 1 1] = $x_5 = x + 2^{-1}x + 2^{-2}x$, where x is the input signal. Note that other BCSs such as [0 0 1], [0 1 0], and [1 0 0] do not require any adder for implementation as they have only one nonzero bit. A straightforward realization of above BCSs would require five adders.

However $x_2$ can be obtained from $x_4$ by a right shift operation (without using any extra adders): $x_2 = 2^{-1}x + 2^{-2}x = 2^{-1}(x + 2^{-1}x) = 2^{-1}x_4$. Also, $x_5$ can be obtained from $x_4$ using an adder: $x_5 = x + 2^{-1}x + 2^{-2}x = x_4 + 2^{-2}x$. Thus, only three adders are needed to realize the BCSs $x_2$ to $x_5$.

The number of adders required for all the possible n-bit binary sub expressions is $2^{n-1} - 1$.The number of adders needed to implement the coefficient multipliers using the binary representation-based BCSE is considerably less than the CSD-based CSE methods. The proposed FIR filter architecture is based on transposed direct form as shown in Fig. 3.1. In the transposed direct form, the coefficient multipliers (shown as dotted outline in Fig.3.1) share the same input and hence commonly known as multiplier block (MB). The MB reduces the complexity of the FIR filter implementations, by exploiting the redundancy in MCM. Thus, redundant computations (partial product additions in the multiplier) are eliminated using BCSE.

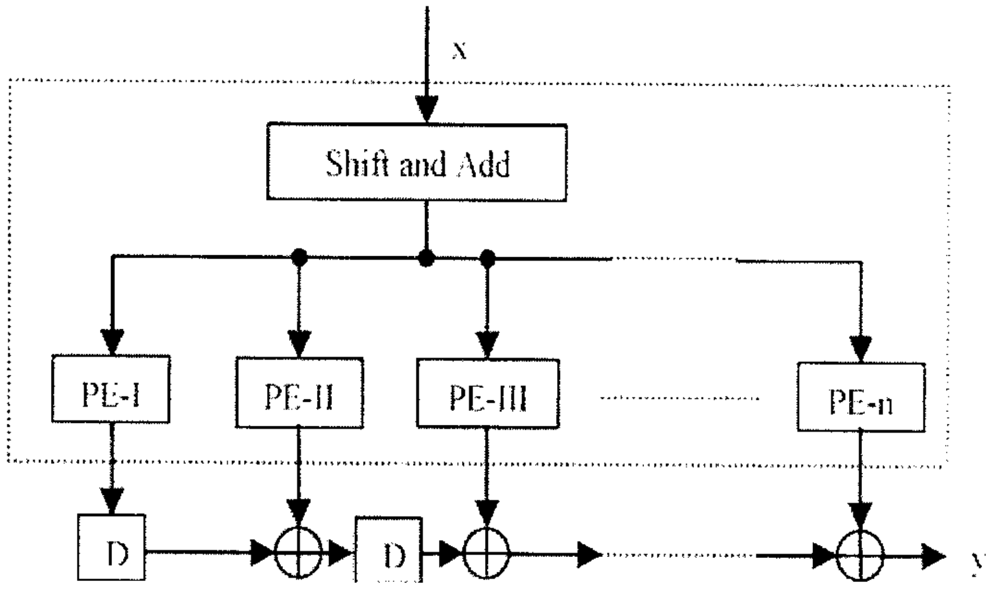


**Fig. 3.1 Transposed direct form of an FIR filter.**

The BCSE method was formulated as a low complexity solution to realize application specific filters where the coefficients are fixed. In the case of channel filters for SDR receivers, the coefficients need to be changed as the filter specification changes with the communication standard. Therefore, reconfigurability is a necessary requirement for SDR channel filters. In the next section, two architectures that incorporate reconfigurability into the BCSE-based low complexity filter design were proposed. Although BCSE is used to illustrate proposed reconfigurable filter architectures; it must be noted that the proposed architectures can satisfy any FIR filter designs with appropriate modifications.

## 3.3 PROPOSED FILTER ARCHITECTURES

Two architectures that integrate reconfigurability and low complexity to realize FIR filters are proposed. The FIR filter architectures proposed are called constant shifts method (CSM) and programmable shifts method (PSM). In this section, the architecture of the proposed FIR filter is presented. The dotted portion in Fig. 3.1 represents the MB. In Fig.3.1, PE-i represents the processing element corresponding to the i th coefficient. PE performs the coefficient multiplication operation with the help of a shift and add unit which will be explained in the latter part of this section. The architecture of PE is different for proposed CSM and PSM. In the CSM, the filter coefficients are partitioned into fixed groups and hence the PE architecture involves constant shifters. But in the PSM, the PE consists of programmable shifters (PS). The FIR filter architecture can be realized in a serial way in which the same PE is used for generation of all partial products by convolving the coefficients with the input signal (h * x[n]) or in a parallel way, where parallel PE architectures are employed. The first option is used when power consumption and area are of prime concern.

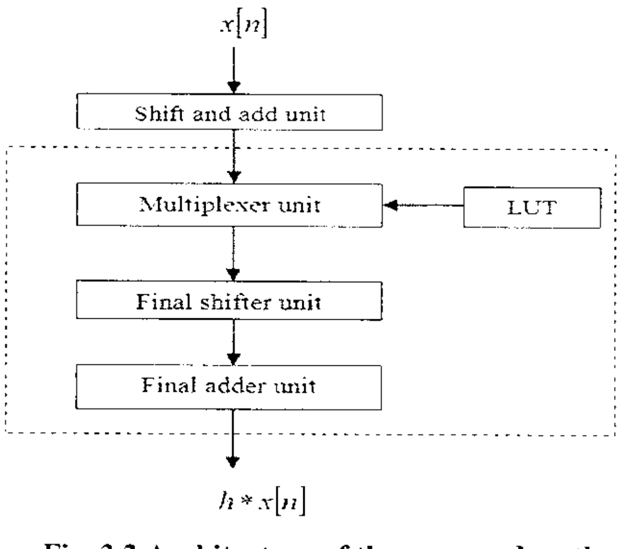


**Fig. 3.2 Architecture of the proposed method.**

The basic architecture of the PE (dotted portion) is shown in Fig.3.2. The functions of different blocks of the PE are explained below.

**3.3.1 Shift and Add Unit**: It is well known that one of the efficient ways to reduce the complexity of multiplication operation is to realize it using shift and add operations. In contrast to conventional shift and add units the BCSs-based shift and add unit is used in proposed CSM and PSM architectures. The architecture of shift and add unit is shown in Fig. 3.3. The shift and add unit is used to realize all the 3-bit BCSs of the input signal ranging from [0 0 0] to [1 1 1]. In Fig. 3.3, "x>>k" represents the input x shifted right by k units. All the 3-bit BCSs [0 1 1], [1 0 1], [1 1 0], and [1 1 1] of a 3-bitnumber are generated using only three adders, whereas a conventional shift and add unit would require five adders. Since the shifts to obtain the BCSs are known beforehand, PS is not required. All these eight BCSs (including [000]) are then fed to the multiplexer unit. In both the architectures (CSM and PSM) the same shift and add unit is used. Thus, the use of 3-bit BCSs reduces the number of adders needed to implement the shift and add unit compared to conventional shift and add units.

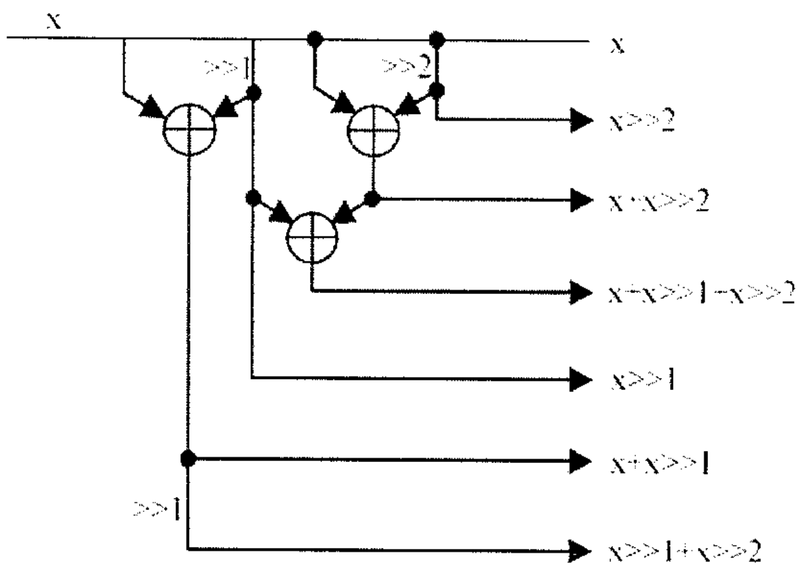


**Fig. 3.3 Architecture of shift and add unit**

**3.3.2 Multiplexer Unit**: The multiplexer units are used to select the appropriate output from the shift and add unit. All the multiplexers will share the outputs of the shift and add unit. The inputs to the multiplexers are the 8/4 inputs from the shift and add unit and hence 8:1/4:1 multiplexer units are employed in the architecture. The select signals of the multiplexers are the filter coefficients which are previously stored in a look up table (LUT). The CSM and PSM architectures basically differ in the way filter coefficients are stored in the LUT. In the CSM, the coefficients are directly stored in LUTs without any modification whereas in PSM, the coefficients are stored in a coded format. The number of multiplexers will also be different for PSM and CSM. In CSM, the number of multiplexers will be dependent on the number of groups after the partitioning of the filter coefficient into fixed groups. The number of multiplexers in the PSM is dependent on the number of non-zero operands in the coefficient for the worst case after the application of BCSE algorithm.

**3.3.3 Final Shifter Unit:** The final shifter unit will perform the shifting operation after all the intermediate additions (i.e., intra-coefficient additions) are done. This can be illustrated using the output expression

$$y = 2^{-4}x + 2^{-6}x + 2^{-15}x + 2^{-16}x \qquad (1)$$

By coefficient-partitioning,

$$y = 2^{-4}(x + 2^{-2}x) + 2^{-15}(x + 2^{-1}x) \qquad (2)$$

After obtaining the intermediate sums $(x + 2^{-2}x)$ and $(x + 2^{-1}x)$ from the shift and add units with the help of multiplexer unit, the final shifter unit will perform the shift operations $2^{\ 4}$and $2^{-15}$ in (2). The PSM and CSM architectures also differ in the nature of final shifters. In the CSM, the final shifts are constants and hence no PS is required. In the PSM, we have used PS.

**3.3.4 Final Adder Unit:** This unit will compute the sum of all the intermediate additions $2^{-4}(x + 2^{-2}x)$ and $2^{-15}(x + 2^{-1}x)$ as in (2). As the filter specifications of different communication standards are different, the coefficients change with the standards. In conventional reconfigurable filters, the new coefficient set corresponding to the filter specification of the new communication standard is loaded in the LUT. Subsequently, the shift and add unit performs a bitwise addition after appropriate shifts.

On the contrary, the proposed CSM and PSM architectures perform a binary common sub expression (BCS)-wise addition (instead bitwise addition). Thus, the same hardware architecture can be used for different filter specifications to achieve the necessary reconfigurability. Moreover, the proposed BCS-based shift and add unit reduces addition operations and hence offers hardware complexity reduction. Also the two CSD based methods are employed in CSM and PSM to compare the complexities of the CSD and binary based CSE techniques. The proposed architectures consider coefficients as constants (as they are stored in LUTs) and input signal as variable. The coefficient multiplication in such a case is known as multiple constant multiplications (MCM), i.e., multiplication of one variable (input signal) with multiple constants (filter coefficients) . The MCM is then optimized for eliminating redundancy using our recently proposed BCSE algorithm to minimize the filter complexity. The proposed CSM focuses on the implementing FIR filters by partitioning the filter coefficients into fixed groups. The PSM has a pre-analysis part which eliminates the redundancy in filter coefficients using the BCSE algorithm.

## 3.4 ARCHITECTURE OF CSM

In the CSM architecture, the coefficients are stored directly in the LUT. These coefficients are partitioned into groups of 3-bits and are used as the select signal for the multiplexers. The number of multiplexer units required is [n/3], where n are the word length of the filter coefficients. The CSM can be explained with the help of an 8-bit coefficient h= "0.11111111." This coefficient h is the worst-case 8-bit coefficient since all the bits are nonzero and hence needs a maximum number of additions and shifts. In this case, n = 8, and therefore the number of multiplexers required is 3. The output y = h * x is expressed as follows.

$$y = 2^{-1}x+2^{-2}x+2^{-3}x+2^{-4}x+2^{-5}x+2^{-6}x+2^{-7}x+2^{-8}x \tag{3}$$

By partitioning into groups of three bits from most significant bit (MSB) (3).

$$h = 2^{-1}(x+2^{-1}x+2^{-2}x+2^{-3}x+2^{-4}x+2^{-5}x+2^{-6}x+2^{-7}x) \tag{4}$$

$$h = 2^{-1}(x+2^{-1}x+2^{-2}x+2^{-3}(x+2^{-1}x+2^{-2}x) +2^{-6}(x+2^{-1}x)) \tag{5}$$

Note that the terms $x + 2^{-1}x + 2^{-2}x$ and $x + 2^{-1}x$ can be obtained from the shift and add unit. Then by using the three multiplexers (mux), two 8:1 mux for the first two 3-bit groups and one 4:1 mux for the last two bits of the filter coefficients, the intermediate sums shown inside the brackets of (5) can be obtained. The final shifter unit will perform the shift operations $2^{-1}$,

$2^{-3}$, and $2^{-6}$. Since these shifts are always constant irrespective of the coefficients, programmable shifters are not required and these shifts can be hardwired. The final adder unit will compute the sum of all the intermediate sums to obtain h $*$ x[n]. The architecture of PE for CSM is shown in Fig. 3.4.

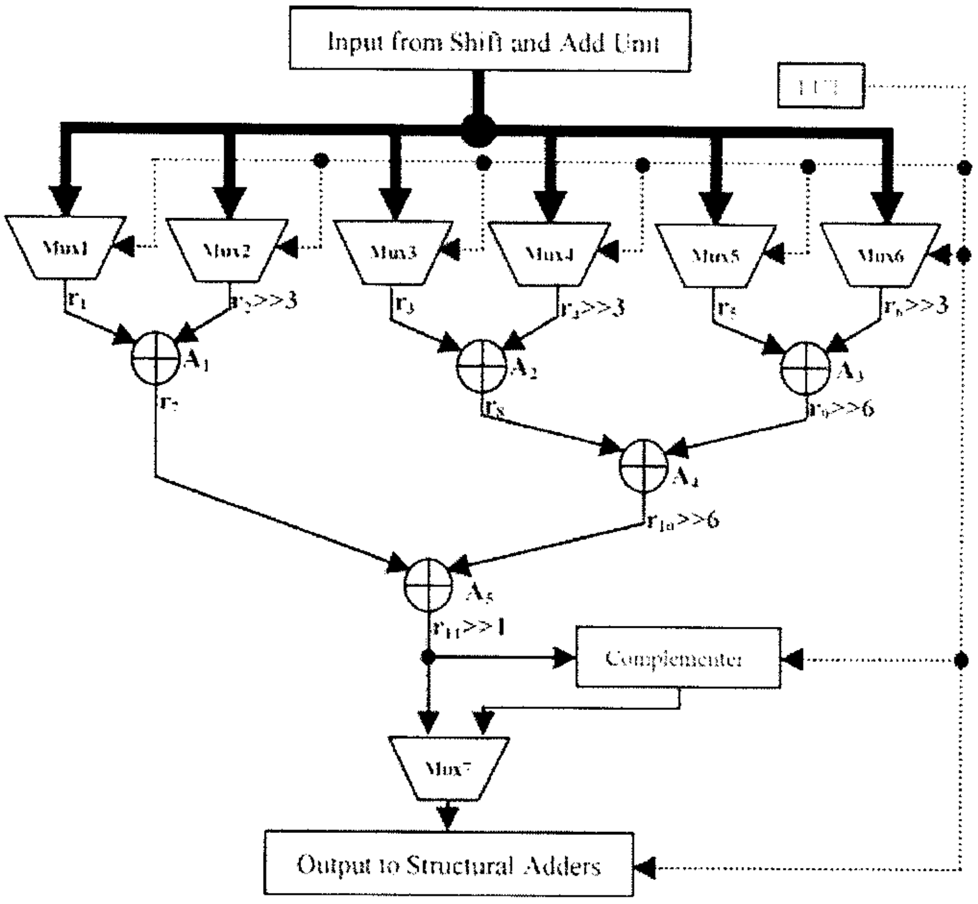


**Fig. 3.4 Architecture of PE for CSM.**

The coefficient word length is considered as 16 bits. The filter coefficients are stored in the LUT in sign-magnitude form with the MSB reserved for the sign bit. The first bit after the sign bit is used to represent the integer part of the coefficient and the remaining 16 bits are used to represent the fractional part of the coefficient. Thus, each 16-bit coefficient is stored as an 18-bit value in LUTs. Each row in LUT corresponds to one coefficient. Note that only half the number of coefficients needs to be stored as FIR filter coefficients are symmetric. The coefficient values corresponding to $2^0$ to $2^{-14}$ are partitioned into groups of three bits and are used as select signals to multiplexers Mux1 to Mux5. i.e., the set ($2^0$, $2^{-1}$, $2^{-2}$) forms the select signal to Mux1 and so on. Since there are 3-bits, eight combinations are possible and hence Mux1 to Mux5 are 8:1 multiplexers. The value corresponding to $2^{-15}$ forms the select to a 2:1 multiplexer, Mux6. The output from the ith multiplexer is denoted as $r_i$. Note that even though the coefficient with values up to a precision of 16 bits is taken, the shifting of $2^{-1}$ is done finally as shown in (4) and (5) and hence the maximum shift will be $2^{-15}$. Mux7 determines whether the output needs to be complemented based on the sign bit of the filter coefficient and hence it is a 2:1 multiplexer. In FIR filters, coefficient values are always less than one. Hence, the integer bit is not employed. However if an integer digit is required, the proposed architectures do not impose any restrictions to accommodate it.

In Fig.3. 4, the shifts are obtained as follows. Let $r_1$ to $r_6$ denotes the outputs of Mux1 to Mux6, respectively. Then

$$y = 2^{-1}r_1 + 2^{-4}r_2 + 2^{-7}r_3 + 2^{-10}r_4 + 2^{-13}r_5 + 2^{-16}r_6 \qquad (6)$$

The shifts are obtained by partitioning the 16-bit coefficient into groups of 3-bits.

By partitioning (6)

$$y = 2^{-1}[(r_1 + 2^{-3}r_2) + 2^{-6}[(r_3 + 2^{-3}r_4) + 2^{-6}(r_5 + 2^{-3}r_6)]] \qquad (7)$$

Substituting $(r_1 + 2^{-3}r_2)$, $(r_3 + 2^{-3}r_4)$, and $(r_5 + 2^{-3}r_6)$ by $r_7$, $r_8$, and $r_9$, respectively, we get

$$y = 2^{-1}[r_7 + 2^{-6}(r_8 + 2^{-6}r_9)] \qquad (8)$$

By substituting $(r_8 + 2^{-6}r_9)$ by $r_{10}$

$$y = 2^{-1}(r_7 + 2^{-6}r_{10}) \qquad (9)$$

By substituting $(r_7 + 2^{-6}r_{10})$ by $r_{11}$

$$y = 2^{-1}(r_{11}) \qquad (10)$$

The expressions from (6)–(10) are represented in Fig.3. 4. The main advantage of the CSM architecture is that all the shifts are constants irrespective of the coefficients and hence can be hardwired resulting in high speed operation of the filter. The shift and add unit which can generate all the 3-bit BCSs using only three adders is employed. The impact of using higher order BCSs (4-bit, 5-bit BCSs, etc.) has also been investigated. The choices of the best shift and add unit will depend on the complexities of: 1) shift and add unit; 2) multiplexer unit; and 3) final adder unit. The number of adders needed to implement n-bit CSs is $2n-1-1$.Thus, shift and add units capable of generating 4-bit, 5-bit, and 6-bit BCSs would require 7, 15, and 31 adders, respectively. The LD is two adder-steps for both the 3-bit and the 4-bit

BCSs based shift and add units, and hence they have the same speed. The LD of 5-bit and 6-bit BCSs-based shift and add units are same, i.e., three adder-steps, which is one adder-step more than that of the 3-bit and 4-bit BCSs. Thus, the 3-bit BCSs based shift and add unit results in fewer number of adders than the 4-bit BCSs-based shift and adder unit (reduction of four adders) with the same LD. The requirement of additional four adders would increase the complexity of the 4-bit BCSs based shift and add unit. Note that the cost of shift and add unit is independent of the number of coefficients (filter length) as the same shift and add unit is shared by all the coefficients.

In the proposed CSM architecture, [W/3] number of 8:1 multiplexers ([W/3] 8:1 multiplexers and remaining 2:1 or 4:1 multiplexers in some cases) of bit-width (x + 2) are required, where W is the coefficient word length and $x$ is the input data word length. For example, if W = 16 (16-bit coefficient), the proposed 3-bit BCSs-based approach requires five 8:1 multiplexers and one 2:1 multiplexer. On the other hand, if 4-bit BCSs were used instead of 3-bit BCSs, four 16:1 multiplexers are required. Assuming an 8:1 multiplexer is equivalent to four 2:1 multiplexers and a 16:1 multiplexer is equivalent to eight 2:1 multiplexers, then the 3-bit BCSs n based PE requires 21 2:1 multiplexers and 4-bit BCSs-based PE requires thirty two 2:1 multiplexers, respectively. Thus, the multiplexer complexity would increase when 4-bit BCSs are used. To be more precise, for each PE with 16-bit filter coefficients, the multiplexer complexity of 4-bit BCSs-based PE is increased by eleven 2:1 multiplexers when compared to 3-bit BCSs based shift and add unit. But it can be noted that the total number of adders required for 3-bit BCS-based filter with n coefficients is 3 + 5n (three adders for shift and add unit and five adders for each PE) and that for 4-bit BCS-based PE is 7 + 3n (seven adders for shift and unit and three adders for each PE). Hence, two adders are saved for 4-bit BCSs based filter for each PE.

From the above discussion, it can be concluded that if 4-bit BCSs were used instead of 3-bit BCSs, the complexity of shift and add unit and multiplexer unit of PE would have increased, whereas complexity of final adder unit would decrease. To provide a quantitative comparison, consider a 16-bit (W = 16) coefficient with an 8-bit quantized (x = 8) input signal. The proposed 3-bit BCSs-based CSM architecture requires twenty one 2:1 multiplexers of word length x + 2 = 10 bits and 4-bit BCSs-based CSM architecture requires thirty two 2:1 multiplexers of word length x + 3 = 11 bits. The 3-bit BCSs based shift and add unit requires three adders with adder-length [number of full adders (FAs)] of 10- bits each. Thus, roughly 3-bit BCSs-based shift and add unit requires 30 FAs (assuming ripple carry addition). For the final adder unit, the proposed 3-bit BCSs-based PE requires five adders (as shown in Fig.3. 4.) Now considering the total complexity of PE (Note that complexity of PE is directly proportional to the number of filter coefficients and total complexity = complexity of multiplexers + complexity of final adder unit).Thus, it is very evident that the 3-bit BCSs-based shift and add unit results in low complexity implementation when compared to the 4-bit BCSs based implementation.

Nevertheless, it must be noted that the CSM architecture can be easily modified to incorporate 4-bit or 5-bit shift and add unit based CSM architectures, if there is such a requirement. In the CSM approach, the coefficients are directly stored in the LUT and hence complete redundancy in coefficient multiplication is not avoided. Also in case of the outputs of any of the multiplexers becoming zero, the adder corresponding to that mux will be used, which is not required if the output is zero. But it can be seen that the adders at the output of the multiplexers can be combined in many ways and hence the best power solution saving can be utilized. Also carry save adders can be employed if much faster operation is required. The drawbacks in CSM are resolved by employing the BCSE algorithm. This forms the PSM architecture which is explained in the next section.

## 3.5 ARCHITECTURE OF PSM

The PSM is based on the BCSE algorithm. The PSM architecture presented in this section incorporates re configurability into BCSE. The PSM has a pre-analysis part in which the filter coefficients are analyzed using the BCSE algorithm. Thus, the redundant computations (additions) are eliminated using the BCSs and the resulting coefficients in a coded format are stored in the LUT. The coding format is explained in the latter part of this section. The shift and add unit is identical for both PSM and CSM. The number of multiplexer units required can be obtained from the filter coefficients after the application of BCSE. The number of multiplexers is selected after considering the number of non-zero operands (BCSs and unpaired bits) in each of the coefficients after the application of the BCSE algorithm. The number of multiplexers will be corresponding to the number of non-zero operands for the worst-case coefficient (worst-case coefficient being defined as coefficient that has the maximum number of non-zero operands). The architecture of PE for PSM is shown in Fig.3. 5.

The coefficient word length is fixed as 16 bits. The statistical analysis for various filters with coefficient precision of 16 bits and different filter lengths (20, 50, 80, 120, 200, 400, and 800 taps) and it was found that the maximum number of non-zero operands is 5 for any coefficient.

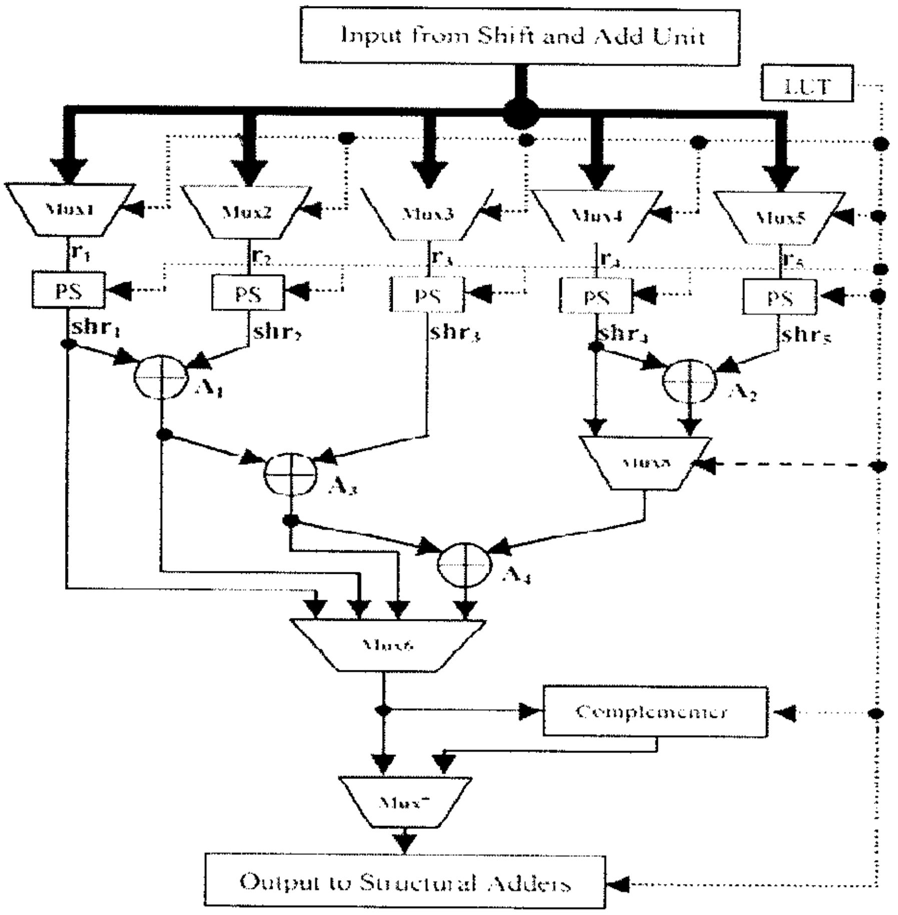


**Fig. 3.5 Architecture of PE for PSM**

The LUT consists of two rows of 18 bits for each coefficient of the form SDDDDXXDDDDXXMMMML and DDDDXXDDDDXXDDDDXX, where "S" represents the sign bit, "DDDD" represents the shift values from $2^0$ to $2^{-15}$ and "XX" represents the input "x" or the BCSs obtained from the shift and add unit. In the coded format, XX = "01" represents "x," "10" represents x+2 $^{1}$x, "11" represents x + 2 $^{2}$x, and "00" represents x + 2 $^{1}$x + $2^{-2}$x, respectively. Thus, the two rows can store up to five operands which is the worst case number of operands for a 16-bit coefficient. In most of the practical coefficients, the number of operands is less than the worst case number of operands, 5. In that case "MMMML" can be used to avoid unnecessary additions. The values "MMMM" will be given as select signal to the Mux6 and "L" to Mux8. "MMMML" indicates the presence of five operands. A "1" in each position indicates the presence of each operand. Thus, for all operands to be present will be indicated by "MMMML" = "11111." This means the Mux6 will select the output from the output of adder, A4 and Mux8 will select the output of adder, A2. If only first operand is present, "MMMML" = "10 000." This means the Mux8 will select the output of PS, shr4 and Mux6 will select the output of PS, shr1. As a result of this none of the adders shr1 to shr4 will be loaded saving significant amount of dynamic power. The coding can be explained as given below. Consider the positive coefficient h,

h = [1010011001010011] (11)

By using the BCSE substituting 2 = [1 1], 3 = [1 0 1],

(11) becomes

h = [3000020003000020] (12)

Then (12) will be stored in the LUT as 000001101011011110 and 100111111010000000. It must be noted that as (12) has only four operands, the fifth operand values "DDDDXX" are substituted as 000000 and "MMMML" as "11110." The XX values are given as select signals for Mux1 to Mux5. The values of DDDD are fed to corresponding PS.

The multiplexer Mux6 and Mux8 will select the appropriate output in case the number of operands after BCSE is less than 5. The use of Mux6 and Mux8 reduces the number of adders utilized by selecting the output from the appropriate adder as all the adders in the PE are not always needed. For example, in (12), as only four operands occur, output can be taken from the output of PS, shr4 without using adder, shr2. Mux8 will do this and hence the adder shr2 is not loaded and consumes zero current and power. The select signals of Mux6 and Mux8 have five bits and hence 25 different control signals are possible which adds lots of flexibility to the architecture which can be employed in future if required. Mux7 is used to complement the output in case of a negative coefficient and its select signal is the sign bit "S" of the coefficient. The PSM architecture has two advantages; first, it guarantees a reduced number of additions compared to CSM, and second it offers the flexibility of changing the word length of coefficients. The same PSM architecture designed for 16-bit coefficients is capable of operating for any coefficient word length less than 16 bits. This means, if the word length is reduced, the format of the LUT can be changed if required. The main advantage of reducing the precision is that some of the adders in the PSM architecture will be unloaded resulting in zero dynamic power.

To the best of our knowledge, the PSM architecture is the first approach toward programmable coefficient word length FIR filter architecture. This means that the coefficient word length of the proposed PSM architecture can be changed dynamically without any change in hardware.

The advantage of CSM is that it produces high-speed filters at the cost of a slight increase in area and power consumption. On the contrary, the PSM produces filters with low area and power consumption at the cost of a slight increase in delay. Another advantage of PSM is that the word length of the filter coefficients can be dynamically changed without any modification in the hardware.

## 3.6 COMPARISON BETWEEN CSM AND PSM

The idea of CSM is to split the filter coefficients into groups of three bits and use these groups as selectors to multiplexer unit and obtain the product h $*$ x[n]. This doesn't guarantee the minimum number of additions to be performed. In PSM, since the BCSE algorithm is employed, the number of additions to be performed will always be reduced compared to CSM. This can be illustrated as follows. Consider the coefficient h = [010100001010]. If CSM is employed, always four multiplexers are needed and this means that the shift and add unit in Fig. 3.3 needs to be used four times. Thus, always three additions are required for CSM. But if PSM is used, first we apply BCSE, then $h_1$ = [020000002000]. The output computation requires only two additions, one for $h_1$ and one for obtaining 2 = [101]. This reduction is significant for higher order filters.

The filters used in SDR channelizers must have a large number of taps to meet the stringent adjacent channel attenuation specifications. Therefore, the proposed PSM architecture is best suited for the channel filters in SDRs. In the case of PSM, the final shifting is done based on the values from LUT using programmable shifters whereas in the case of CSM, the shifts are constants as always splitting or partitioning the filter coefficients into groups of 3-bits is done.

Thus, the CSM architecture results in faster coefficient multiplication operation at the cost of few extra adders compared to PSM architecture whereas the PSM architecture results in fewer number of additions and thus less area and power consumption compared to the CSM architecture.

Another advantage of PSM is that it is independent of the word length of the filter coefficients. For PSM architecture, the number of multiplexers is fixed based on the number of BCSs present in a given coefficient set (worst case-coefficient of the set). Thus, even if the word length changes, it hardly affects the architecture of PSM. It was pointed out that for many filter taps; the highest coefficient precision is not required. Valuable hardware resources will be wasted if all taps are implemented with the highest precision. The proposed PSM can be implemented for dynamically varying coefficient precision as it is word length independent. One of the limitations of the PSM architecture is that it requires pre-analysis of filter coefficients and hence on-the-fly reconfigurability is not always feasible. But this restriction does not impose constraints on popular reconfigurable filter applications like wireless communications. This is because in such applications, there is a distinct filter for each communication standard and the coefficients of the filter are fixed for a specific standard. In other words, when the communication system is operating on a particular wireless standard, the filter coefficients do not change, i.e., the filter is not required to be an adaptive filter. When the system changes its mode of operation to a different wireless communication standard (as in the case of a multi-standard transceiver), the coefficient set corresponding to the specification of the new standard is loaded (replacing the current filter coefficients). Note that the coefficients of the new standard are known beforehand (pre-stored) and therefore the pre-analysis can be done offline and the problem with reconfigurability can be solved. The comparison of CSM and PSM architectures employing BCSE technique can be viewed clearly from the analysis reports at the end. Each architecture is advantageous in their own respective ways.

# CHAPTER 4

# SIMULATION RESULTS

The tools used to obtain the simulated output for constant shift method (CSM) and programmable shift method (PSM) architecture are as follows. They are

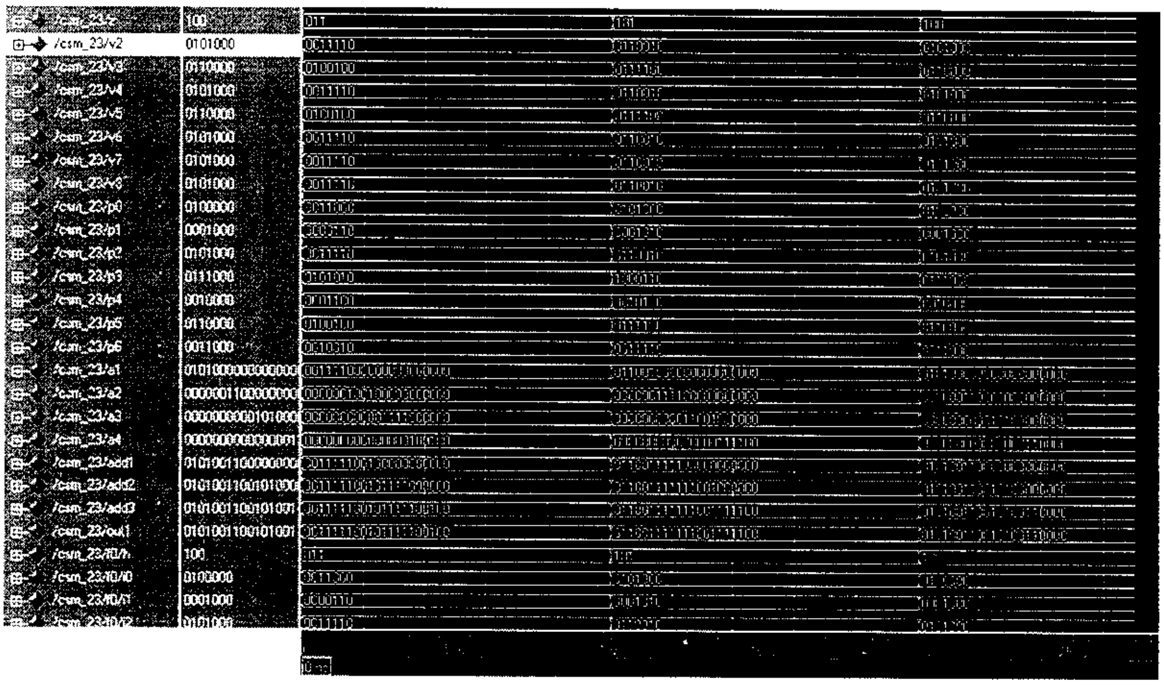
1) Modelsim 6.4

2) Xilinx ISE 9.2i

The simulated waveforms are obtained by assigning the input values at various levels of extraction and the corresponding outputs are obtained from the assigned inputs. The outputs obtained are complementary with respect to the corresponding complementary inputs. The simulated waveforms are shown here.

## 4.1 SIMULATED OUTPUT OF CSM WHEN USING CSE

Figure 4.1 shows the simulated output of proposed architecture of CSM using CSE algorithm, done by using ModelSim software

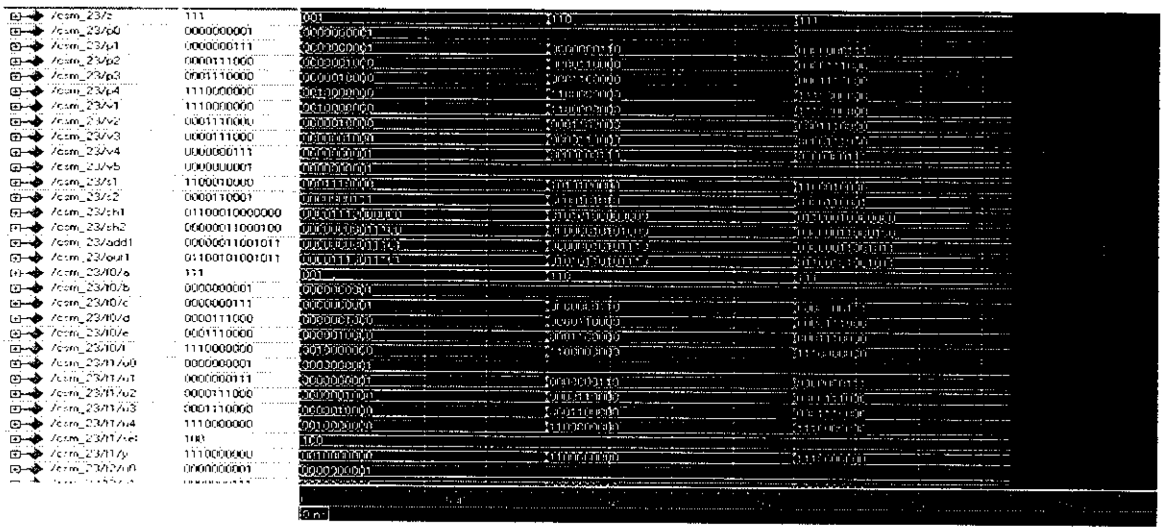


```
/csm_23/z        100
/csm_23/v2       0101000
/csm_23/v3       0110000
/csm_23/v4       0101000
/csm_23/v5       0110000
/csm_23/v6       0101000
/csm_23/v7       0101000
/csm_23/v3       0101000
/csm_23/p0       0100000
/csm_23/p1       0001000
/csm_23/p2       0101000
/csm_23/p3       0111000
/csm_23/p4       0010000
/csm_23/p5       0110000
/csm_23/p6       0011000
/csm_23/a1       0101000000000000
/csm_23/a2       0000001100000000
/csm_23/a3       0000000000101000
/csm_23/a4       0000000000000001
/csm_23/add1     0101001100000000
/csm_23/add2     0101001100101000
/csm_23/add3     0101001100101001
/csm_23/out1     0101001100101001
/csm_23/f0/h     100
/csm_23/f0/f0    0100000
/csm_23/f0/f1    0001000
/csm_23/f0/f2    0101000
```

Fig.4.1. Simulation output when using CSE in CSM

## 4.2 SIMULATED OUTPUT OF CSM WHEN USING LCCSE

Figure 4.2 shows the simulated output of proposed architecture of CSM using LCCSE algorithm, done by using ModelSim software

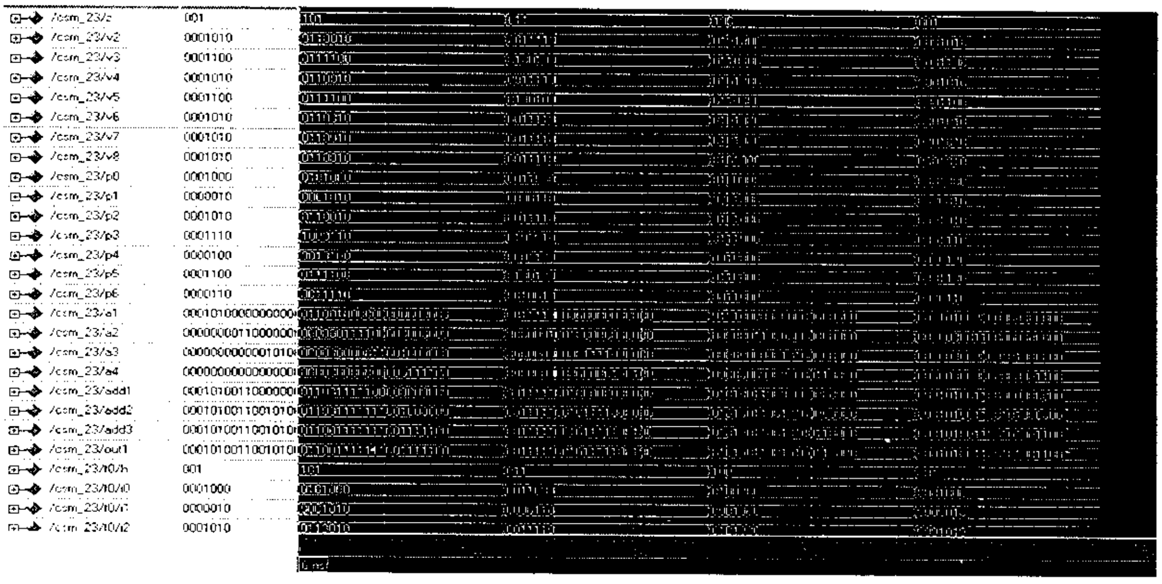


```
/csm_23/z          111
/csm_23/p0         0000000001
/csm_23/p1         0000000111
/csm_23/p2         0000111000
/csm_23/p3         0001110000
/csm_23/p4         1110000000
/csm_23/v1         1110000000
/csm_23/v2         0001110000
/csm_23/v3         0000111000
/csm_23/v4         0000000111
/csm_23/v5         0000000001
/csm_23/s1         1100010000
/csm_23/s2         0000110001
/csm_23/sh1        0110001000000
/csm_23/sh2        0000001100100
/csm_23/add1       0000001100101l
/csm_23/out1       0110010100101l
/csm_23/t0/a       111
/csm_23/t0/b       0000000001
/csm_23/t0/c       0000000111
/csm_23/t0/d       0000111000
/csm_23/t0/e       0001110000
/csm_23/t0/f       1110000000
/csm_23/t1/u0      0000000001
/csm_23/t1/u1      0000000111
/csm_23/t1/u2      0000111000
/csm_23/t1/u3      0001110000
/csm_23/t1/u4      1110000000
/csm_23/t1/sel     100
/csm_23/t1/y       1110000000
/csm_23/t2/u0      0000000001
```

Fig.4.2. Simulation output when using LCCSE in CSM

## 4.3 SIMULATED OUTPUT OF CSM WHEN USING BCSE

Figure 4.3 shows the simulated output of proposed architecture of CSM using BCSE algorithm, done by using ModelSim software

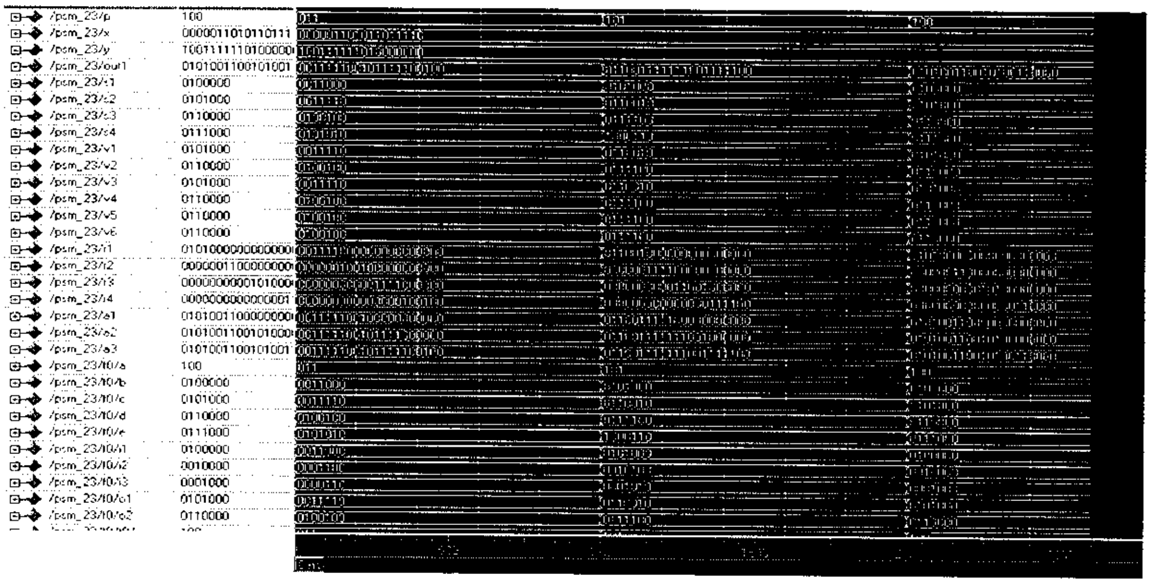


```
/csm_23/z          001
/csm_23/v2         0001010
/csm_23/v3         0001100
/csm_23/v4         0001010
/csm_23/v5         0001100
/csm_23/v6         0001010
/csm_23/v7         0001010
/csm_23/v8         0001010
/csm_23/p0         0001000
/csm_23/p1         0000010
/csm_23/p2         0001010
/csm_23/p3         0001110
/csm_23/p4         0000100
/csm_23/p5         0001100
/csm_23/p6         0000110
/csm_23/a1         0001010000000000
/csm_23/a2         0000000011000000
/csm_23/a3         0000000000001010
/csm_23/a4         0000000000000000
/csm_23/add1       0001010011000000
/csm_23/add2       0001010011001010
/csm_23/add3       0001010011001010
/csm_23/out1       0001010011001010
/csm_23/t0/h       001
/csm_23/t0/i0      0001000
/csm_23/t0/i1      0000010
/csm_23/t0/i2      0001010
```

Fig.4.3. Simulation output when using BCSE in CSM

## 4.4 SIMULATED OUTPUT OF PSM WHEN USING CSE

Figure 4.4 shows the simulated output of proposed architecture of PSM using CSE algorithm, done by using ModelSim software

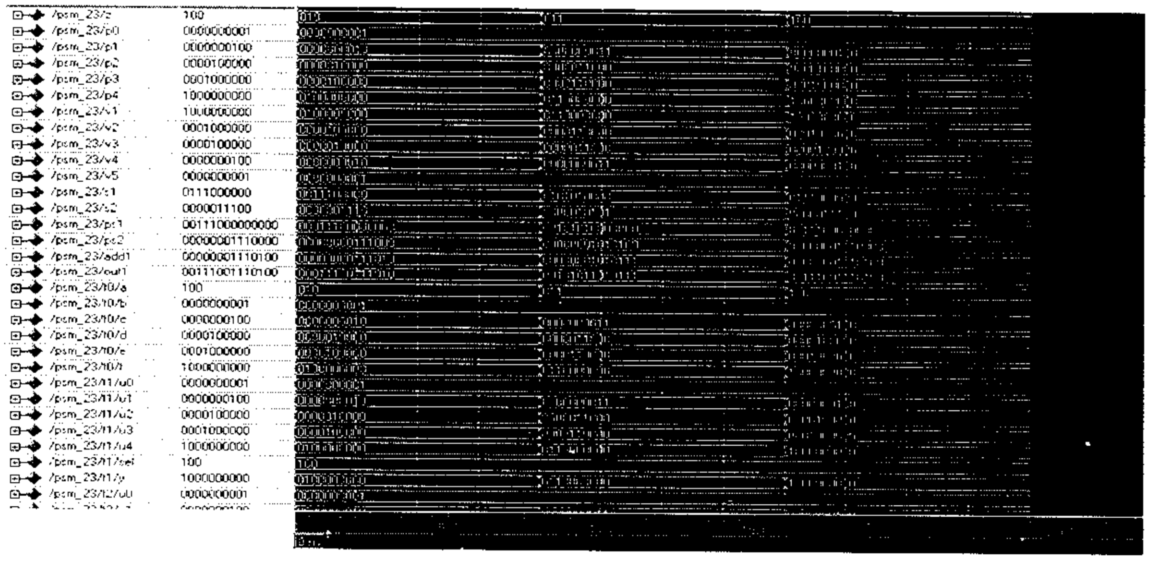


```
/psm_23/p        100
/psm_23/x        0000011010110111
/psm_23/y        1001111110100000
/psm_23/out1     0101001100101001
/psm_23/s1       0100000
/psm_23/s2       0101000
/psm_23/s3       0110000
/psm_23/s4       0111000
/psm_23/v1       0101000
/psm_23/v2       0110000
/psm_23/v3       0101000
/psm_23/v4       0110000
/psm_23/v5       0110000
/psm_23/v6       0110000
/psm_23/r1       0101000000000000
/psm_23/r2       0000001100000000
/psm_23/r3       0000000000101000
/psm_23/r4       0000000000000001
/psm_23/a1       0101001100000000
/psm_23/a2       0101001100101000
/psm_23/a3       0101001100101001
/psm_23/t0/a     100
/psm_23/t0/b     0100000
/psm_23/t0/c     0101000
/psm_23/t0/d     0110000
/psm_23/t0/e     0111000
/psm_23/t0/i1    0100000
/psm_23/t0/i2    0010000
/psm_23/t0/i3    0001000
/psm_23/t0/o1    0101000
/psm_23/t0/o2    0110000
```

Fig.4.4. Simulation output when using CSE in PSM

## 4.5 SIMULATED OUTPUT OF PSM WHEN USING LCCSE

Figure 4.5 shows the simulated output of proposed architecture of PSM using LCCSE algorithm, done by using ModelSim software

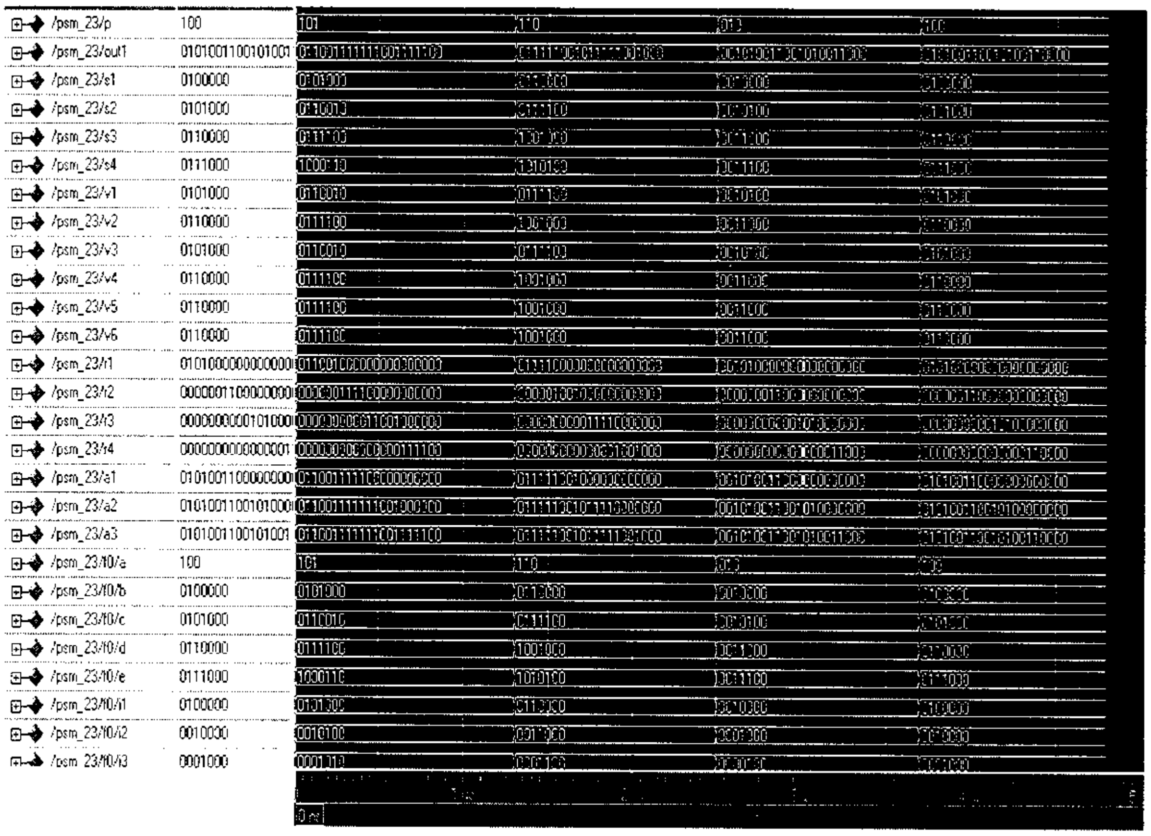


```
/psm_23/z        100
/psm_23/p0       0000000001
/psm_23/p1       0000000100
/psm_23/p2       0000100000
/psm_23/p3       0001000000
/psm_23/p4       1000000000
/psm_23/v1       1000000000
/psm_23/v2       0001000000
/psm_23/v3       0000100000
/psm_23/v4       0000000100
/psm_23/v5       0000000001
/psm_23/s1       0111000000
/psm_23/s2       0000011100
/psm_23/ps1      0011100000000
/psm_23/ps2      0000001110000
/psm_23/add1     0000001110100
/psm_23/out1     0011100111010 0
/psm_23/t0/a     100
/psm_23/t0/b     0000000001
/psm_23/t0/c     0000000100
/psm_23/t0/d     0000100000
/psm_23/t0/e     0001000000
/psm_23/t0/f     1000000000
/psm_23/t1/u0    0000000001
/psm_23/t1/u1    0000000100
/psm_23/t1/u2    0000100000
/psm_23/t1/u3    0001000000
/psm_23/t1/u4    1000000000
/psm_23/t1/sel   100
/psm_23/t1/y     1000000000
/psm_23/t2/u0    0000000001
```

Fig.4.5. Simulation output when using LCCSE in PSM

## 4.6 SIMULATED OUTPUT OF PSM WHEN USING BCSE

Figure 4.6 shows the simulated output of proposed architecture of PSM using BCSE algorithm, done by using ModelSim software



```
/psm_23/p        100
/psm_23/out1     0101001100101001
/psm_23/s1       0100000
/psm_23/s2       0101000
/psm_23/s3       0110000
/psm_23/s4       0111000
/psm_23/v1       0101000
/psm_23/v2       0110000
/psm_23/v3       0101000
/psm_23/v4       0110000
/psm_23/v5       0110000
/psm_23/v6       0110000
/psm_23/r1       0101000000000000
/psm_23/r2       0000001100000000
/psm_23/r3       0000000000101000
/psm_23/r4       0000000000000001
/psm_23/a1       0101001100000000
/psm_23/a2       0101001100101000
/psm_23/a3       0101001100101001
/psm_23/f0/a     100
/psm_23/f0/b     0100000
/psm_23/f0/c     0101000
/psm_23/f0/d     0110000
/psm_23/f0/e     0111000
/psm_23/f0/i1    0100000
/psm_23/f0/i2    0010000
/psm_23/f0/i3    0001000
```

**Fig.4.6. Simulation output when using BCSE in PSM**

## 4.7 SYNTHESIS REPORT FOR CSM USING CSE METHOD

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

| | | |
|---|---|---|
| Number of Slices: | 59 out of 6912 | 1% |
| Number of 4 input LUTs: | 106 out of 13,824 | 1% |
| Number of bonded IOBs: | 281 out of 510 | 55% |

Total equivalent gate count for design = 888

## 4.8 SYNTHESIS REPORT FOR CSM USING LCCSE METHOD

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

| | | |
|---|---|---|
| Number of Slices: | 58 out of 6912 | 1% |
| Number of 4 input LUTs: | 109 out of 13,824 | 1% |
| Number of bonded IOBs: | 194 out of 510 | 38% |

Total equivalent gate count for design = 1275

## 4.9 SYNTHESIS REPORT FOR PROPOSED CSM USING BCSE TECHNIQUE

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

| | | |
|---|---|---|
| Number of Slices: | 18 out of 6912 | 1% |
| Number of 4 input LUTs: | 35 out of 13,824 | 1% |
| Number of bonded IOBs: | 269 out of 510 | 52% |

Total equivalent gate count for design = 288

Timing Summary:

Speed Grade: -7

Minimum input arrival time before clock: 8.026ns

Maximum output required time after clock: 4.264ns

## 4.10 SYNTHESIS REPORT FOR PSM USING CSE METHOD

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

| | | | | |
|---|---|---|---|---|
| Number of Slices: | 118 | out of | 6912 | 1% |
| Number of 4 input LUTs: | 228 | out of | 13,824 | 1% |
| Number of bonded IOBs: | 48 | out of | 510 | 9% |

Total equivalent gate count for design = 2158

## 4.11 SYNTHESIS REPORT FOR PSM USING LCCSE METHOD

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

| | | | | |
|---|---|---|---|---|
| Number of Slices: | 79 | out of | 6912 | 1% |
| Number of 4 input LUTs: | 150 | out of | 13,824 | 1% |
| Number of bonded IOBs: | 202 | out of | 510 | 9% |

Total equivalent gate count for design = 1814

## 4.12 SYNTHESIS REPORT FOR PROPOSED PSM USING BCSE TECHNIQUE

Design summary:

Selected Device: Xilinx FPGA- 2s300eft256-6

| | | | | |
|---|---|---|---|---|
| Number of Slices: | 18 | out of | 6912 | 1% |
| Number of 4 input LUTs: | 34 | out of | 13,824 | 1% |
| Number of bonded IOBs: | 24 | out of | 510 | 9% |

Total equivalent gate count for design = 276

Timing Summary:

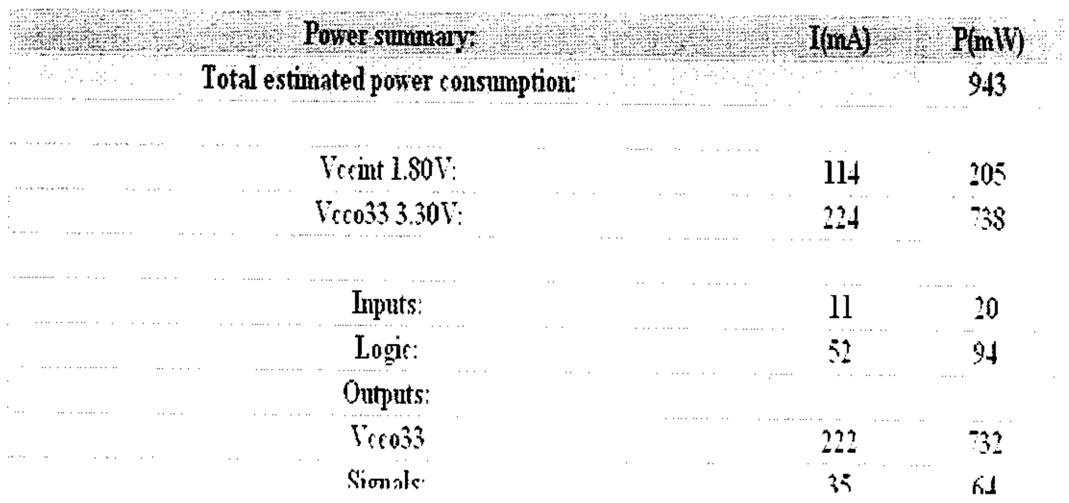
Speed Grade: -7

Minimum input arrival time before clock: 10.014ns

Maximum output required time after clock: 6.514ns

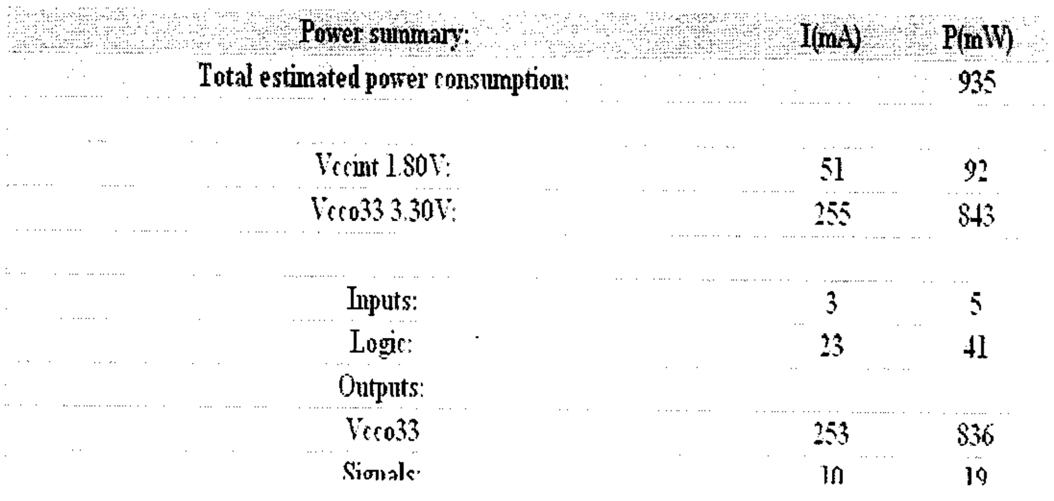
## 4.13 POWER CALCULATION

The power calculation of proposed CSM and PSM architecture with CSE and LCCSE algorithm and BCSE technique are shown below and comparison of CSM and PSM technique can be viewed from the power report.



| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 943 |
| Vccint 1.80V: | 114 | 205 |
| Vcco33 3.30V: | 224 | 738 |
| Inputs: | 11 | 20 |
| Logic: | 52 | 94 |
| Outputs: | | |
| Vcco33 | 222 | 732 |
| Signals: | 35 | 64 |

Table 4.1 Power Calculation in CSM with CSE



| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 935 |
| Vccint 1.80V: | 51 | 92 |
| Vcco33 3.30V: | 255 | 843 |
| Inputs: | 3 | 5 |
| Logic: | 23 | 41 |
| Outputs: | | |
| Vcco33 | 253 | 836 |
| Signals: | 10 | 19 |

Table 4.2 Power Calculation in CSM with LCCSE

| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 318 |
| | | |
| Vccint 1.80V: | 173 | 312 |
| Vcco33 3.30V: | 2 | 7 |
| | | |
| Inputs: | 24 | 44 |
| Logic: | 103 | 186 |
| Outputs: | | |
| Vcco33 | 0 | 0 |
| Signals: | 31 | 55 |
| | | |
| Quiescent Vccint 1.80V: | 15 | 27 |
| Quiescent Vcco33 3.30V: | 2 | 7 |

Table 4.3 Power Calculation in PSM with CSE

| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 280 |
| | | |
| Vccint 1.80V: | 36 | 64 |
| Vcco33 3.30V: | 65 | 216 |
| | | |
| Clocks: | 1 | 2 |
| Inputs: | 2 | 4 |
| Logic: | 10 | 18 |
| Outputs: | | |
| Vcco33 | 63 | 209 |
| Signals: | 7 | 13 |
| | | |
| Quiescent Vccint 1.80V: | 15 | 27 |
| Quiescent Vcco33 3.30V: | 2 | 7 |

Table 4.4 Power Calculation in PSM with LCCSE

| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 627 |
| | | |
| Vccint 1.80V: | 25 | 45 |
| Vcco33 3.30V: | 176 | 582 |
| | | |
| Inputs: | 0 | 1 |
| Logic: | 7 | 12 |
| Outputs: | | |
| Vcco33 | 174 | 575 |
| Signals: | 3 | 6 |
| | | |
| Quiescent Vccint 1.80V: | 15 | 27 |
| Quiescent Vcco33 3.30V: | 2 | 7 |

**Table 4.5 Power Calculation of proposed CSM architecture with BCSE technique**

| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 106 |
| | | |
| Vccint 1.80V: | 26 | 47 |
| Vcco33 3.30V: | 18 | 59 |
| | | |
| Inputs: | 1 | 1 |
| Logic: | 8 | 15 |
| Outputs: | | |
| Vcco33 | 16 | 52 |
| Signals: | 2 | 4 |
| | | |
| Quiescent Vccint 1.80V: | 15 | 27 |
| Quiescent Vcco33 3.30V: | 2 | 7 |

**Table 4.6 Power Calculation of proposed PSM architecture with BCSE technique**

# CHAPTER 5
# CONCLUSION AND FUTURE SCOPE

## 5.1 CONCLUSION

Thus a novel synthesis technique for generating FIR filter designs which simultaneously cater to low-energy requirements while maintaining a reasonably accurate filter response is implemented. The Common sub expression elimination (CSE) algorithm and Level Constrained Common Sub expression (LCCSE) algorithm helps in reducing the number of adders thus bringing about reduction in hardware complexity.

When using Binary Common Sub expression Elimination (BCSE) algorithm in the proposed two new approaches namely, CSM(constant shift method) and PSM(programmable shift method), there is reduction in the number of adders and low power consumption and increase in speed rather than using CSE and LCCSE algorithms; thus bringing low complexity. The CSM architecture results in high speed filters and PSM architecture results in low area and thus low power filter implementations. The PSM also provides the flexibility of changing the filter coefficient word lengths dynamically. This architecture results in high speed, low area and low power hence effectively used in the area requiring high throughput such as a real-time digital signal processing.

## 5.2 FUTURE SCOPE

Recently, with the advent of software defined radio (SDR) technology, finite impulse response (FIR) filter research has been focused on reconfigurable realizations. Wideband receivers in SDR must be realized to meet the stringent specifications of low power consumption and high speed. Reconfigurability of the receiver to work with different wireless communication standards is another key requirement in an SDR. In the future work, the focus will be towards modifying the proposed reconfigurable architectures to be easily employed in any method which provides a general approach for low complexity reconfigurable channel filters.

# REFERENCES

[1] Jung Hwan Choi, Nilangan Bannerjee and Koushik Roy, "Variation aware low power synthesis methodology for FIR filters," IEEE Transaction on Computers, vol. 28.No.1 January 2009 ,pp: 87 – 97.

[2] S. Vijay *et al.*, "A greedy common sub expression elimination algorithm for implementing FIR filters," in *Proc. ISCAS* ,May 2008, pp. 3451–3454

[3] R. Mahesh and A. P. Vinod, "A new common subexpression elimination algorithm for realizing low complexity higher order digital filters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 2, pp. 217–219, Feb. 2008.

[4] A. P. Vinod and E. Lai, "Low power and high-speed implementation of FIR filters for software defined radio receivers," *IEEE Trans. Wireless Commun.*, vol. 5, no. 7, pp. 1669–1675, Jul. 2007.

[5] Shahnam Mirzoei, Anup Hosangadi and Ryan Kastner, "FPGA implementation of High speed FIR filters using Add and Shift Method," *IEEE Trans. Circuits Syst. I, Reg.Papers*, vol. 49, no. 10, pp. 221–235, Nov. 2007.

[6] C. Yao *et al.*, "A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters," *IEEE Trans. Circuits Syst. I, Reg.Papers*, vol. 51, no. 11, pp. 2211–2215, Nov. 2006.

[7] K. H. Chen and T. D. Chiueh, "A low-power digit-based reconfigurable FIR filter," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 8, pp. 617–621, Aug. 2005.

[8] A. P. Vinod and E. M.-K. Lai, "On the implementation of efficient channel filters for wideband receivers by optimizing common subexpression elimination methods," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 2, pp. 295–304, Feb. 2005.

[9] T. Solla and O. Vainio, "Comparison of programmable FIR filter architectures for low power," in *Proc.* 28th *Eur. Solid-State Circuits Conf.*, Firenze, Italy, Sep. 2004, pp. 759–762.

[10] T. Zhangwen, J. Zhang, and H. Min, "A high-speed, programmable, CSD coefficient FIR filter," *IEEE Trans. Consumer Electron.*, vol. 48, no. 4, pp. 834–837, Nov. 2002.

[11] S. S. Demirsoy, I. Kale, and A. G. Dempster, "Efficient implementation of digital filters using novel reconfigurable multiplier blocks," in *Proc. 38th Asilomar Conf. Signals Syst. Comput.*, vol. 1. Nov. 2002, pp. 461– 464.

[12] H.-J. Kang and I.-C. Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders, " *IEEE Trans. Circuits Syst. II, Analog Digit .Signal Proc* vol. 48, no. 8, pp. 770–777, Aug. 2001

[13] R. M. Hewlitt and E. S. Swartzlander, "Canonical signed digit representation for FIR digital filters," in *Proc. IEEE WorkShop SiPS*, 2000, pp. 416–426.

[14] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 9, pp. 569–577, Sep. 2000.

[15] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Comput. Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 58–68, Jan. 1999.

[16] R. I. Hartley, "Sub expression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, no. 10, pp. 677–688,Oct. 1996

# APPENDIX

XILINX®

## Spartan-IIE 1.8V FPGA Family: Introduction and Ordering Information

DS077-1 (v1.0) November 15, 2001

**Preliminary Product Specification**

## Introduction

The Spartan™-IIE 1.8V Field-Programmable Gate Array family gives users high performance, abundant logic resources, and a rich feature set, all at an exceptionally low price. The five-member family offers densities ranging from 50,000 to 300,000 system gates, as shown in Table 1. System performance is supported beyond 200 MHz.

Spartan-IIE devices deliver more gates, I/Os, and features per dollar than other FPGAs by combining advanced process technology with a streamlined architecture based on the proven Virtex™-E platform. Features include block RAM (to 64K bits), distributed RAM (to 98,304 bits), 19 selectable I/O standards, and four DLLs (Delay-Locked Loops). Fast, predictable interconnect means that successive design iterations continue to meet timing requirements.

The Spartan-IIE family is a superior alternative to mask-programmed ASICs. The FPGA avoids the initial cost, lengthy development cycles, and inherent risk of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary (impossible with ASICs).

## Features

- Second generation ASIC replacement technology
  - Densities as high as 6,912 logic cells with up to 300,000 system gates
  - Streamlined features based on Virtex-E architecture
  - Unlimited in-system reprogrammability
  - Very low cost
- System level features
  - SelectRAM+™ hierarchical memory:
    - 16 bits/LUT distributed RAM
    - Configurable 4K-bit true dual-port block RAM
    - Fast interfaces to external RAM
  - Fully 3.3V PCI compliant to 64 bits at 66 MHz and CardBus compliant
  - Low-power segmented routing architecture
  - Full readback ability for verification/observability
  - Dedicated carry logic for high-speed arithmetic
  - Efficient multiplier support
  - Cascade chain for wide-input functions
  - Abundant registers/latches with enable, set, reset
  - Four dedicated DLLs for advanced clock control
  - Four primary low-skew global clock distribution nets
  - IEEE 1149.1 compatible boundary scan logic
- Versatile I/O and packaging
  - Low cost packages available in all densities
  - Family footprint compatibility in common packages
  - 19 high-performance interface standards, including LVDS and LVPECL
  - Up to 120 differential I/O pairs that can be input, output, or bidirectional
  - Zero hold time simplifies system timing
- Fully supported by powerful Xilinx ISE development system
  - Fully automatic mapping, placement, and routing
  - Integrated with design entry and verification tools

*Table 1:* **Spartan-IIE FPGA Family Members**

| Device | Logic Cells | Typical System Gate Range (Logic and RAM) | CLB Array (R x C) | Total CLBs | Maximum Available User I/O | Maximum Differential I/O Pairs | Distributed RAM Bits | Block RAM Bits |
|---|---|---|---|---|---|---|---|---|
| XC2S50E | 1,728 | 23,000 - 50,000 | 16 x 24 | 384 | 182 | 84 | 24,576 | 32K |
| XC2S100E | 2,700 | 37,000 - 100,000 | 20 x 30 | 600 | 202 | 86 | 38,400 | 40K |
| XC2S150E | 3,888 | 52,000 - 150,000 | 24 x 36 | 864 | 263 | 114 | 55,296 | 48K |
| XC2S200E | 5,292 | 71,000 - 200,000 | 28 x 42 | 1,176 | 289 | 120 | 75,264 | 56K |
| XC2S300E | 6,912 | 93,000 - 300,000 | 32 x 48 | 1,536 | 329 | 120 | 98,304 | 64K |

## General Overview

The Spartan-IIE family of FPGAs have a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs). There are four Delay-Locked Loops (DLLs), one at each corner of the die. Two columns of block RAM lie on opposite sides of the die, between the CLBs and the IOB columns. These functional elements are interconnected by a powerful hierarchy of versatile routing channels (see Figure 1).

Spartan-IIE FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or Boundary Scan modes. The Xilinx XC17S00A PROM family is recommended for serial configuration of Spartan-IIE FPGAs. The XC18V00 reprogrammable PROM family is recommended for parallel or serial configuration.

Spartan-IIE FPGAs are typically used in high-volume applications where the versatility of a fast programmable solution adds benefits. Spartan-IIE FPGAs are ideal for shortening product development cycles while offering a cost-effective solution for high volume production.

Spartan-IIE FPGAs achieve high-performance, low-cost operation through advanced architecture and semiconductor technology. Spartan-IIE devices provide system clock rates beyond 200 MHz. Spartan-IIE FPGAs offer the most cost-effective solution while maintaining leading edge performance. In addition to the conventional benefits of high-volume programmable logic solutions, Spartan-IIE FPGAs also offer on-chip synchronous single-port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features.

### Spartan-IIE Family Compared to Spartan-II Family

- Higher density and more I/O
- Higher performance
- Unique pinouts in cost-effective packages
- Differential signaling
  - LVDS, Bus LVDS, LVPECL
- $V_{CCINT}$ = 1.8V
  - Lower power
  - 5V tolerance with 100Ω external resistor
  - 3V tolerance directly
- PCI, LVTTL, and LVCMOS2 input buffers powered by $V_{CCO}$ instead of $V_{CCINT}$
- Unique larger bitstream

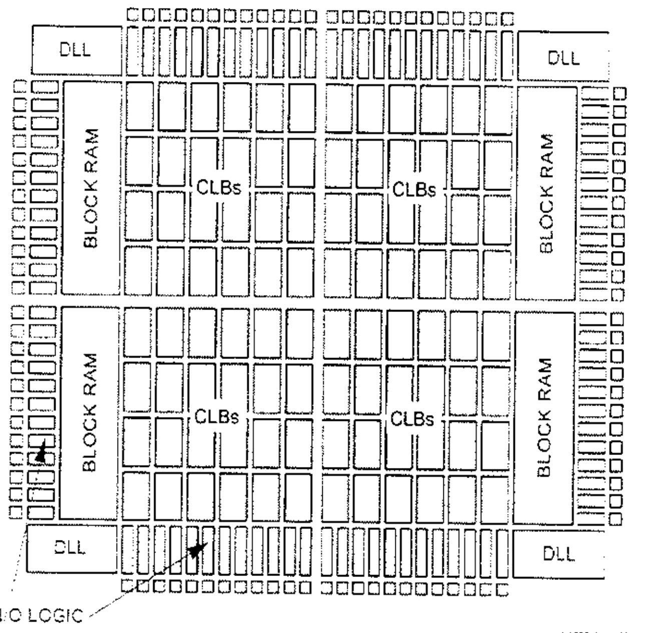


*Figure 1:* **Basic Spartan-IIE Family FPGA Block Diagram**

P - 3449

## Spartan-IIE Product Availability

Table 2 shows the package and speed grades available for Spartan-IIE family devices. Table 3 shows the maximum user I/Os available on the device and the number of user I/Os available for each device/package combination.

*Table 2:* **Spartan-IIE Package and Speed Grade Availability**

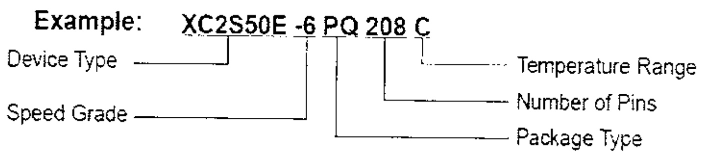
| | Pins | 144 | 208 | 256 | 456 |
|---|---|---|---|---|---|
| | Type | Plastic TQFP | Plastic PQFP | Fine Pitch BGA | Fine Pitch BGA |
| Device | Code | TQ144 | PQ208 | FT256 | FG456 |
| XC2S50E | -6 | C, I | C, I | C, I | - |
| | -7 | (C) | (C) | (C) | - |
| XC2S100E | -6 | C, I | C, I | C, I | C, I |
| | -7 | (C) | (C) | (C) | (C) |
| XC2S150E | -6 | - | (C, I) | (C, I) | (C, I) |
| | -7 | - | (C) | (C) | (C) |
| XC2S200E | -6 | - | C, I | C, I | C, I |
| | -7 | - | (C) | (C) | (C) |
| XC2S300E | -6 | - | C, I | C, I | C, I |
| | -7 | - | (C) | (C) | (C) |

Notes:
1. C = Commercial, $T_J = 0^\circ$ to $+85^\circ C$; I = Industrial, $T_J = -40^\circ C$ to $+100^\circ C$
2. Parentheses indicate product not yet released. Contact sales for availability.

*Table 3:* **Spartan-IIE User I/O Chart**

| Device | Maximum User I/O | Available User I/O According to Package Type | | | |
|---|---|---|---|---|---|
| | | TQ144 | PQ208 | FT256 | FG456 |
| XC2S50E | 182 | 102 | 146 | 182 | - |
| XC2S100E | 202 | 102 | 146 | 182 | 202 |
| XC2S150E | 263 | - | 146 | 182 | 263 |
| XC2S200E | 289 | - | 146 | 182 | 289 |
| XC2S300E | 329 | - | 146 | 182 | 329 |

## Ordering Information



### Device Ordering Options

| Device |
|---|
| XC2S50E |
| XC2S100E |
| XC2S150E |
| XC2S200E |
| XC2S300E |

| Speed Grade | |
|---|---|
| -6 | Standard Performance |
| -7 | Higher Performance |

| Package Type / Number of Pins | |
|---|---|
| TQ144 | 144-pin Plastic Thin QFP |
| PQ208 | 208-pin Plastic QFP |
| FT256 | 256-ball Fine Pitch BGA |
| FG456 | 456-ball Fine Pitch BGA |

| Temperature Range ($T_J$) | |
|---|---|
| C = Commercial | 0°C to +85°C |
| I = Industrial | –40°C to +100°C |

## Revision History

| Version No. | Date | Description |
|---|---|---|
| 1.0 | 11/15/01 | Initial Xilinx release. |

## The Spartan-IIE Family Data Sheet

DS077-1, *Spartan-IIE 1.8V FPGA Family: Introduction and Ordering Information* (Module 1)

DS077-2, *Spartan-IIE 1.8V FPGA Family: Functional Description* (Module 2)

DS077-3, *Spartan-IIE 1.8V FPGA Family: DC and Switching Characteristics* (Module 3)

DS077-4, *Spartan-IIE 1.8V FPGA Family: Pinout Tables* (Module 4)