Low Power Techniques and Architectures for Multicarrier Wireless Receivers

Mohd. Hasan

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
August 2003

Abstract

Power consumption is a critical issue in portable wireless communication. Multicarrier code division multiple access (MC-CDMA) has a significant potential to be included as a standard in the next generation of mobile communication. This thesis investigates new low power architectures for a MC-CDMA receiver. The FFT processor is one of the major power consuming blocks in multicarrier systems based on Orthogonal frequency division multiplexing (OFDM), like MC-CDMA, wireless LANs etc. Three low power schemes are presented for reducing the power consumption in FFT processors namely order based processing, coefficient memory reduction and simplified coefficient addressing.

The order based processing scheme is based on a novel concept of using either the normal or two's complement form for only the real part of the coefficients selectively to minimise the Hamming distance between successive coefficients fed to the multipliers. This significantly reduces the switching activity at the coefficient input of the multiplier and hence the power consumption. The coefficient memory reduction scheme exploits the relationship among the coefficient values to reduce the coefficient memory size from $N/2$ locations to $((N/8) + 1)$ locations for an N-point FFT, thereby saving both area and power for long FFTs. The proposed coefficient addressing scheme implements the complete coefficient addressing for all stages of a radix-2 FFT processor by using a simple multiplexer instead of a cascade of Barrel shifters. Low power single butterfly radix-2 FFT processor and radix-4 ordered pipelined FFT processor architectures based on the novel order based processing scheme are also proposed.

The ordered low power radix-4 FFT processor is combined with the combiner to realise a low power MC-CDMA receiver. The power consumption in a MC-CDMA receiver can be further reduced by introducing the concept of dynamically altering the complexity of the receiver in real time as per the changing channel parameters such as the delay spread, maximum Doppler frequency, transmission rate and signal to noise ratio instead of using a receiver designed for the worst case scenario. The FFT size in multicarrier systems like MC-CDMA varies from 16-point to 1024-points depending upon the channel parameters. This thesis has proposed a reconfigurable 256-point FFT processor architecture that can be configured in real time to act as a 64-point or 16-point FFT processor to prove the concept. The power reduction is significant in moving from a fixed 256-point FFT to a reconfigurable 256-point FFT provided that the FFT size is varying over a large range, which is indeed, the case for a MC-CDMA receiver. This power reduction is achieved by using an appropriate FFT size (shorter FFTs) by disabling the clocks of the higher stages in real time. A reconfigurable pipelined MC-CDMA receiver architecture is also proposed that can be configured in real time to process 256 or 64 sub-carriers on the basis of the channel parameters. The power saving is obtained by disabling the first stage and the last ordering stage of the FFT processor and also by disabling the unused equaliser memory in the Combiner by switching from 256 to 64 sub-carriers.

An FIR filter is also an important block in wireless receivers. A number of novel low power FIR filter cores based on different low power algorithms and their hybrid have also been presented.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself in the School of Engineering and Electronics at the University of Edinburgh.

Acknowledgements

I would like to thank Dr Tughrul Arslan and Dr John Thompson for their excellent support and guidance during the research period.

Grateful thanks to Dr Ahmet Erdogan for his help throughout the last three years. Thanks to Dr. Emad Al-susa, Mr. Robert Thompson, Dr. Nizamettin Aydin, Dr. Peter Hillman and all my lab colleagues for their help throughout my research work.

Special thanks to the Association of Commonwealth Universities and the British Council for funding my research work.

Heartfelt thanks to my wife for her patience, encouragement and support throughout my research work. Thanks to my son Imaad for making my life so wonderful.

A very special thanks to my parents who have guided and supported me throughout my life.

Contents

List of figures

List of tables

Acronyms and abbreviations

ACC	Accumulator
ADC	Analog to digital converter
ADD	Adder
ADM	Additional dual port random access memory
ASIC	Application specific integrated circuit
AU	Arithmetic unit
BP	Block processing
BPSK	Binary phase shift keying
CA	Coefficient address
CDMA	Code division multiple access
CLACC	Clearing accumulator logic
CMOS	Complementary metal oxide semiconductor technology
COMB	Combination of coefficient segmentation and Block processing
COMP	Inversion
CONV	Conventional
CSEG	Coefficient segmentation
CSA	Carry save array multiplier
DAC	Digital to analog converter
DF	Direct form
DFT	Discrete Fourier transform
DM	Dual port random access memory
DS-CDMA	Direct sequence code division multiple access
DSP	Digital signal processing
FDMA	Frequency division multiple access
FFT	Fast Fourier transform
FIFO	First in first out memory
FIR	Finite impulse response filter
FSM	Finite state machine
IFFT	Inverse fast Fourier transform

I/O	Input-output
ISI	Intersymbol interference
LAN	Local area network
LNS	Logarithmic number system
LSB	Least significant bit
LUT	Look-up-table
MC-CDMA	Multicarrier code division multiple access
MMSE	Minimum mean square error
MSB	Most significant bit
NBW	Non-Booth coded Wallace tree multiplier
OFDM	Orthogonal frequency division multiplexing
ORD	Ordering block in a reconfigurable MC-CDMA receiver
ORDER	Coefficient ordering
RAM	Random access memory
RAME	Even parity memory bank
RAMO	Odd parity memory bank
RF	Radio frequency
RISC	Reduced instruction set computer
ROM	Read only memory
RTL	Register transfer language
SAIF	Switching activity interchange format
SDF	Standard delay format
SNR	Signal to noise ratio
SOC	Silicon on chip
SR	Shift register
SUB	Subtractor
SUM	Summer
TCMP	Two's complementer module
TDF	Transpose direct form
TDMA	Time division multiple access
TM	Triple port random access memory
VLSI	Very large scale integration
WALL	Booth-coded Wallace tree multiplier

Nomenclature

clk	Clock signal
C_{load}	Load capacitance
f	Frequency of operation
f_d	Maximum Doppler frequency
λ	Number of active users dependent parameter
μ	Micron
N	FFT size in points
N_f	FIR filter order
Nf_d	Normalised maximum Doppler frequency
P	Processing gain
P_{sw}	Switching power consumption
R	Transmission rate
S_w	Switching activity
τ	Delay spread
$\sim$	Complement
$x(n)$	Input sampled sequence
$X(k)$	DFT of the input sequence x(n)
V_{dd}	Supply voltage

Chapter 1
Introduction

1.1 Motivation

The design of portable devices requires critical consideration of the time averaged power consumption which is directly proportional to the battery weight and volume required to operate circuits for a given amount of time. The battery life depends on both the power consumption of the system and the battery capacity. The battery technology has improved considerably with the advent of portable systems but it is not expected to offer significant advances in the near future [1, 2]. Most of the portable applications demand high speed computation, complex functionalities and often real-time processing capabilities with low power consumption. The explosive growth of portable wireless devices like cellular phones, pagers, wireless modems and laptops along with the limitation of the battery technology has elevated power consumption to be one of the most critical design requirement [3].

Moreover, there is a strong need to reduce the power consumption in high performance microprocessors to limit the cost of packaging and cooling requirements [4–6]. Also, high power systems are more prone to several silicon failure mechanisms. Every 10^0C rise in operating temperature roughly doubles a component's failure rate [7]. Hence, power consumption has now become an important design criteria just like speed and silicon area.

Multicarrier systems perform much better than single carrier systems in the hostile wireless environment [8]. MC-CDMA, a combination of OFDM and CDMA, has a lot of potential to be included as a standard in the future generations of mobile communications [9] and therefore power consumption is an important issue in multicarrier systems. This thesis is motivated by the desire to reduce the power consumption of a MC-CDMA receiver.

1.2 Contribution

This thesis investigates low power techniques and architectures for the important signal processing and Telecommunication blocks used in MC-CDMA and other multicarrier based receivers

such as the FFT processors, Combiner and FIR filters by reducing their switched capacitances. It also introduces a novel concept of dynamically altering the hardware complexity of a MC-CDMA receiver in real time as per the channel parameters instead of using a fixed receiver designed for the worst case channel conditions.

The FFT processor is one of the most critical power consuming blocks in all multicarrier based wireless communication systems [10]. This thesis proposes both single butterfly and pipelined low power FFT processor architectures. The novel low power architectures are based on a new order based processing scheme. The conventional order based processing scheme is based on ordering the coefficients so as to minimise the Hamming distance (Bit transition) between successive coefficients fed to the multiplier [11, 12]. This results in a reduction of switching activity at the coefficient input of the power consuming multiplier and hence its power consumption. It has been demonstrated that the conventional order based processing scheme does not lead to much power savings. This is due to only small switching activity reduction and hardware overhead required to realise the scheme in case of FFT processor.

The modified order based processing scheme is also based on Hamming distance minimisation between successive coefficients fed to the butterfly with the difference that either the real part of the coefficient or its two's complemented value is used for the minimisation of switching activity. It has been demonstrated that this procedure results in significant reduction in switching activity of the order of 53% compared to only 27% with conventional ordering approach in case of FFT processors. This reduction in switching activity leads to power savings in the range of 25% to 1% for 16-point to 512-point FFT processor cores respectively.

The order based processing scheme has been extended to the domain of radix-4 pipelined FFT processor architecture [13]. It has been possible to apply order based processing to the first stage of a 16-point or second stage of a 64-point radix-4 FFT processor. The results have shown around 14-37% power reduction as compared to the conventional architecture for the 16-point radix-4 FFT processor depending upon the multiplier type. This technique is limited only to the first stage of a 16-point FFT to contain the hardware overhead in the form of an additional memory required for restoring the data order after every stage. The power saving is around 4-30% for the 64-point FFT processor depending upon the type of multiplier and the FIFO implementation.

This thesis also proposes a coefficient memory reduction scheme. According to this scheme,

the size of ROM required to store N/2 coefficient in an N-point FFT can be reduced to N/8+1 rather than N/4 [14]. This reduction in memory leads to both power and area savings for long FFT's.

This thesis also presents a novel coefficient addressing scheme for the single butterfly radix-2 FFT processor. The coefficient addressing scheme is based on a single multiplexer rather than a cascade of Barrel shifters [15]. This saves both area and power.

This thesis also proposes a low power MC-CDMA receiver by combining a low power FFT processor architecture with a low power Combiner for binary phase shift keying (BPSK).

In MC-CDMA, the FFT size directly depends upon the channel parameters like the delay spread, maximum Doppler frequency and the transmission rate [16]. The FFT size varies from 16-point to 1024-point depending upon the channel parameters which in turn depends on the location of the user. Two reconfigurable 256 point radix-4 pipelined FFT processor architectures namely *RFFT-I* and *RFFT-II* are proposed which can also be dynamically configured to act as a 16-point or 64-point FFT processors depending upon the channel parameters to prove the concept. The difference between the two architectures lies in the extent of clock gating to disable the unused modules. This approach will significantly reduce the power consumption because only the optimum FFT size is used at all times instead of a large fixed FFT for the worst case channel conditions.

Two reconfigurable MC-CDMA receiver architectures namely *RECEIVER-I* and *RECEIVER-II* are also proposed by combining the respective 256-point reconfigurable pipelined FFT processor architectures with the reconfigurable Combiner. The receiver can be reconfigured in real time to operate either on 256 or 64 sub-carriers depending upon the channel parameters. This reconfigurabilty leads to power reduction in the receiver by switching from 256 to 64 sub-carriers as compared to a fixed 256 sub-carrier receiver. This reduction is achieved by shutting down the first stage and partly disabling the last ordering stage of the 256-point reconfigurable FFT processor, and also the unused RAM for storing the equaliser coefficients in the Combiner for 64 sub-carriers in *RECEIVER-I*. The shutdown of the unused modules is partial in *RECEIVER-II*.

The FIR filter is also an important block in wireless receivers and other Telecommunication systems. The power consumption in case of direct form FIR filters can be reduced by applying techniques such as coefficient ordering, coefficient segmentation and block processing etc.

Most of the work in the literature deals with the effect of these techniques on the power consuming multiplier block of the FIR filter [12, 17, 18]. Each of these techniques has hardware overhead and therefore it is very important to know the power reduction obtained by these techniques on the overall FIR filter. This work also proposes low power architectures for the direct form FIR filter based on the previously mentioned techniques and their combination. Each architecture has been investigated for the different types of multiplier. It has been found that the combination of block processing and coefficient segmentation algorithms yield best power reduction results at a slightly higher expense in area. Any low power architecture can be chosen depending upon the power-area budget.

1.3 Structure

The structure of this thesis is as follows:

- Chapter 2 presents review of the research work in the area of low power techniques and architectures for the multicarrier based wireless receiver.
- Chapter 3 presents an overview of the Discrete Fourier transform and the fast Fourier transform. It mainly covers radix-2 and radix-4 FFT algorithms.
- Chapter 4 presents new techniques for reducing the power consumption in an FFT processor. A novel form of order based processing, tailored for FFT processors, is proposed. A new technique to reduce the coefficient memory from N/4 locations to N/8+1 locations for an N-point FFT processor is also described. The chapter concludes with the presentation of a new circuit for coefficient address generation.
- Chapter 5 describes a low power architecture of a single butterfly FFT processor based on a novel order based processing algorithm. Power results are presented to demonstrate the effectiveness of the technique.
- Chapter 6 presents a low power pipelined radix-4 FFT processor based on a novel order based processing algorithm. The first stage of a 16-point radix-4 FFT processor is modified to incorporate the corresponding data ordering as per the new coefficient ordering. Power results are also given to demonstrate the effectiveness of the scheme.
- Chapter 7 presents a pipelined architecture for a MC-CDMA receiver. This architecture

is based on a low power 64-point radix-4 pipelined FFT processor. The combiner module architecture is also tailored for low power by disabling the unused hardware blocks.

- Chapter 8 describes reconfigurable architectures for both the pipelined FFT processor and the pipelined MC-CDMA receiver. In multicarrier wireless systems, FFT size varies from 16-point to 1024-point depending upon the channel parameters. Two reconfigurable 256-point FFT processor architectures that can be configured as a 64-point or a 16-point are proposed to prove the concept. The difference between the two architectures lies in the extent of applied clock gating. Two reconfigurable MC-CDMA receivers are also proposed which can be configured to act either as a 256 sub-carrier system or a 64 sub-carrier system depending upon the channel parameters in real time. The difference between the two receivers again lies in the extent of clock gating. Power saving results, in switching from 256 to 64 sub-carriers as compared to a fixed 256 sub-carrier receiver, for both the architectures are presented. Similarly, power saving results in switching from 256-point FFT processor to a 64-point or a 16-point FFT processor for both the FFT architectures with respect to a fixed 256-point FFT processor, are also given.

- Chapter 9 presents low power FIR filter architectures for the different low power algorithms namely coefficient ordering, coefficient segmentation, block processing and their combination. The post layout power and area results are presented for the whole FIR filter.

- Chapter 10 presents conclusions and a summary of the thesis and suggests topics for future research.

- Appendix A lists the publications arising from this thesis.

- Appendix B lists the C code for the order based processing scheme.

- Appendix C lists the MATLAB code for the MC-CDMA transceiver.

- Appendix D lists the Verilog code for the 64 sub-carrier MC-CDMA receiver.

1.4 Summary

Low power design is a crucial research area not only for portable systems but also for high performance systems like microprocessors. Multicarrier systems are very attractive for all portable

wireless applications. This thesis presents techniques and architectures to reduce the power consumption in the blocks of a MC-CDMA receiver like the FFT, Combiner and FIR filter as well as for the whole receiver. The next chapter describes the existing low power techniques and architectures for the multicarrier based receivers.

Chapter 2
Low power techniques and architectures for multicarrier wireless receivers

2.1 Introduction

This chapter starts with the identification of the sources of power consumption in CMOS technology. This is followed by the review of the general algorithmic and architectural level low power techniques for reducing the switched capacitance. Then the low power techniques and architectures targeted for the basic blocks of the multicarrier based wireless receiver namely the FFT and the FIR filter are described. After that, a low power technique for reducing the power consumption of the Combiner block and the whole MC-CDMA receiver is presented. In the end, the power characteristics of the three commonly used multiplier types for different low power algorithms for a 73-tap FIR filter are presented.

2.2 Sources of power consumption in CMOS technology

There are three significant sources of power consumption in CMOS circuits: switching power P_{sw}, short-circuit power and leakage power [5, 19]. P_{sw} is the power consumed in charging and discharging the load capacitance C_{load}. The switching power, given by the following expression, accounts for around 80% [20] of the total power consumption in CMOS.

$$P_{sw} = 1/2 S_w C_{load} V_{dd}^2 f$$

Where V_{dd} is the supply voltage, f is the clock frequency , C_{load} is the load capacitance of the gate and S_w is the switching activity factor which is defined as the average number of times that the gate makes a logic transition ($1 \rightarrow 0$ or $0 \rightarrow 1$) in each clock cycle. The product $S_w C_{load}$ is defined as the switched capacitance.

The short-circuit power is due to the conduction path established between the supply rails on account of finite signal rise/fall times. The leakage power [21] is due to the flow of subthreshold and reverse-biased diodes currents. This work deals with the reduction of the dominant switching power component of the total power consumption in CMOS.

The switching power is a strong function of the supply voltage and therefore any technique of scaling down the supply voltage significantly brings down the power consumption. The switching power can also be reduced by decreasing the switched capacitance. The switched capacitance can be reduced either by lowering the load capacitance C_{load} or/and the switching activity S_w. The switched capacitance based power reduction has been studied at various levels of the design abstraction starting from algorithmic level down to the technology level [5, 19]. Among these levels, power consumption can be significantly reduced at the algorithmic and architectural level [22] especially in memory and mathematically intensive digital signal processing algorithms by reducing the switched capacitance.

This thesis explores techniques at algorithmic and architectural levels for reducing the switched capacitance thereby saving power in multicarrier based wireless receivers.

2.3 General low power techniques for reducing the switched capacitance

The major power consumption in CMOS occurs only during switching and therefore the switching activity has to be reduced to the minimal level required to perform the computation. The following sections explore the system level approaches to minimise the switched capacitance at the algorithmic, architectural, logic and physical design levels.

2.3.1 Clock gating

Clock switching has a considerable impact on power. It can be as much as twice the logic power [19]. Clock gating is used to disable unused modules of the system. It saves power by both preventing unnecessary switching activity in the logic modules as well as by eliminating power dissipation in the clock distribution network [23, 24]. Clock gating must be carefully applied by taking into account the physical design aspect as well. The optimal solution points to a partial approach in which power saving due to gating is balanced by efficient wiring.

2.3.2 Operation minimisation

The switched capacitance can be effectively reduced by minimising the number of operations. This is accomplished by transforming the algorithm such that it can be realised by using less hardware in terms of multiplier, memory and other power consuming modules [25].

2.3.3 Operation substitution

The multiplication with constants is a commonly used operation in most of the signal processing algorithms. This multiplication can also be realised by using shift-add operation. The power consumed by the execution unit of an 11-tap FIR filter is reduced to one-eighth of the original power by replacing multiplication by shift and add operations [25]. The shifter and adder consume much less power as compared to the multiplier.

The multiplication can also be performed by a ROM, shift registers and adders using Distributed arithmetic [26, 27]. This concept is quite attractive for low throughput applications.

2.3.4 Input and constant coefficient ordering

The switched capacitance can also be reduced by reordering the inputs in a chain of operation such that the higher activity inputs should enter the chain at a later stage [25].

The multiplication with a constant coefficient is a common operation in signal processing algorithms. It is possible to save around 15% power by carrying out the multiplication after ordering the coefficients as per Hamming distance in single functional unit architectures [11].

2.3.5 Reducing glitching activity

Glitching refers to spurious transitions due to finite propagation delays from one logic block to the next [5]. It arises when paths with unbalanced propagation delays converge at the same point in the circuit. A node can undergo multiple power consuming transitions in a single clock cycle before settling to the correct logic level. An average of 15-20% of the total power is consumed in gliching [3]. It can be minimised by balancing all signal paths along with the reduction of logic depth [5]. A retiming approach for glitch reduction was proposed in [28, 29] by placing registers to compensate different path delays. Several other techniques like

restructuring multiplexer networks and clocking control signals have also been proposed [30].

2.3.6 Precomputation

It is based on selectively precomputing the output logic values of the circuit one clock cycle before they are required, and using the precomputed values to reduce the internal switching activity in the succeeding clock cycle [31, 32]. The precomputation logic determines the output values for a subset of input conditions. The original circuit can be turned off in the subsequent clock cycle resulting in reduced switching activity. The size of the precomputation logic determines the power dissipation reduction and area increase relative to the original circuit.

2.3.7 Data representation

The choice of data representation also influences the switching activity. In two's complement representation, the signal transition from positive to negative or vice-versa causes the MSB sign-bits to switch resulting in high switching activity. The switching activity increases significantly when the signals being processed switch frequently around zero and when they do not utilise the entire bit-width. On the other hand, in sign magnitude representation, the switching around zero results in slight increase in switching activity due to only sign bit toggling [5]. In [33], floating and logarithmic number systems are compared with fixed point in terms of accuracy and switched capacitance for speech coding application. For little or no increase in distortion, capacitance reduction factor of two to three are achieved by trading precision for dynamic range.

2.3.8 Bus encoding

Gray coding is normally employed when the data to be transmitted on the bus is sequential and highly correlated. Gray-coded instruction addressing results in around 30-50% reduction in switching activity in a RISC processor compared to normal binary coded addressing [34, 35].

Bus-invert coding reduces I/O bus activity leading to reduction in I/O peak power by 50% and average I/O power consumption by 25% [36]. It uses an extra control bit called invert. This coding scheme is realised by first computing the Hamming distance between the present bus value and next data value. If the Hamming distance is greater than half the number of bits, set

invert to high and make the next bus value equal to the inverted next data value. Otherwise, the invert bit is set to low and the next bus value equals the next data value . The receiver extracts the correct bus value by conditionally inverting the bus value according to the invert bit.

2.3.9 Memory partitioning

In many embedded applications, highly accessed locations can fit into a relatively small memory space. The memory is partitioned such that the highly accessed locations are mapped to a small memory [34, 37–39]. The average power in accessing the memory is decreased because a large fraction of accesses is concentrated on a small power efficient memory. Moreover, other memory banks that are not accessed in a given cycle are disabled through their chip select inputs.

2.3.10 State assignment

A state assignment procedure has been presented in [40] for reducing the switching activity of the state variables by minimising the number of bit changes during state transitions. The probability of state transitions in a FSM is calculated from the given input switching probability. This information is then used to find out an encoding that minimises the switching probability of the state variables. The authors in [41] have addressed both FSM and combinational logic synthesis problems to minimise the average number of transitions. The synthesis process comprises of two parts: state assignment, which determines the combinational logic function and multilevel optimization of the combinational logic, which tries to minimise area while at the same time trying to reduce the circuit activity at the internal nodes of the circuit.

2.3.11 Scheduling and resource binding

Scheduling and resource binding algorithms are proposed in [42] to minimise the number of transitions on the signals feeding the functional units (adders, multipliers etc.) and registers, which effectively minimises the switched capacitance. This is achieved by scheduling the candidate nodes in control steps as close as possible and binding them to the same resource. The candidate nodes are selected such that there is no change of values in the input operands between consecutive operations of the same functional unit.

2.3.12 Selection of appropriate gate level implementation

The switching activity also depends upon the gate level topology of different implementation for the same function. In [43], six different adders are constructed with inverters and two to four-input AND and OR gates. Extensive simulations are used to evaluate their switching characteristics and the results of those simulations are used to rank the adders on speed, size and the number of logic transitions. This approach is adopted for the different multipliers as well [44, 45]. Hence, it is important to choose the appropriate gate level implementation depending upon speed, power and area.

2.3.13 Technology decomposition and mapping

For the same gate-level implementation, there are many possible circuit level implementations and each of these implementations, has different switching activity. Therefore, technology decomposition and mapping techniques have been introduced in the literature for minimising switching activities of such networks [46, 47]. According to this concept, a given Boolean network is decomposed such that the switching activity of the network is minimised. Further power reductions are then achieved by hiding high activity nodes inside more complex CMOS gates. Complex gates tend to exhibit an overall lower capacitance since more signals are confined to internal nodes rather than to the more heavily loaded output nodes. This results in the mapping of signals with high switching activities to low capacitance internal nodes.

2.3.14 Wordlength reduction

The number of bits used strongly affect all key parameters of a design including speed, area and power [25, 48]. It is desirable to minimise the number of bits during power optimisation because fewer bits result in fewer switching events and therefore lower switched capacitance. Moreover, fewer bits not only reduce the number of transfer lines, but also decrease the average interconnect length and capacitance.

2.3.15 Physical capacitance reduction

The switching power depends linearly on the physical capacitance being switched. The physical capacitance at the output of a CMOS gate is the sum of the input capacitances of N driven

gates, parasitic output capacitance of the driver gate and the interconnect capacitance. The parasitic capacitances in CMOS technology are of either parallel plate type or voltage dependent diode junction type. These capacitances can be minimised by using least logic, smaller devices and shorter wires [3]. The area of the circuit can be reduced by logic minimisation and gate re-sizing [49]. The interconnect capacitance can be reduced by appropriate placement [50], partitioning [51] and wire sizing [52].

2.4 Low power techniques and architectures for FFT processor

Here is a brief description of the low power techniques and architectures for the FFT processor which is one of the most power consuming block of a multicarrier based wireless receiver.

2.4.1 Cache based architecture

The basic FFT has poor locality because each of the N outputs of an N-point FFT depends on each of the N inputs. The author in [53, 54] has proposed a low power cache based 1024-point FFT processor. The author has proposed an FFT algorithm that offers good locality over large portions of the computation. The cache based architecture is similar to the single memory architecture except that a small cache memory resides between the processor and the main memory. The cache memory architecture is energy efficient because small memories require lower energy per access.

2.4.2 Partial product ordering

The aim of reordering is the minimization of the sum of Hamming distances between successive pairs of partial products on the busses connecting the data and coefficient storage elements to the functional units. The authors in [55–57] have proposed a general technique to reorder the sequence of evaluation of partial products that constitute the basic computation. The Hamming distance between the coefficient part of the partial product can be directly evaluated since coefficients are known before realisation. The evaluation of the exact Hamming distance between two data samples is impossible since data are not known before run-time. An approximate average Hamming distance between the data terms of the partial product is determined by simulation of typical data as per the application. In [55, 57], the authors have shown that the coefficient

switching activity has reduced by 19-25% but the data switching activity varies slightly in both directions for the different FFTs after reordering. This indicates that data ordering is not useful.

2.4.3 Wordlength optimisation

The size of the pipelined FFT processor is dominated by the FIFOs. It is, therefore, important to minimize the sizes of the FIFOs. Since the FIFO size is large in the early stages, it is desirable to have as shorter wordlength as possible in the early stages. The authors in [58] have proposed a wordlength optimisation scheme for the pipelined FFT processor. The optimisation of the wordlength depends on the desired value of signal-to-noise ratio (SNR) for a given input and output wordlengths. The desired SNR value is achieved by altering the internal wordlengths. The authors have shown that increasing the wordlength progressively from the input wordlength of 12 bits to 16-bits towards the output before rounding it back to 12 bits for the output, gives much better performance in terms of SNR and memory size than a fixed wordlength system. It is possible to achieve 10dB increase in SNR with less than 5% increase in memory size by using progressive wordlength instead of fixed wordlength of 12 bits.

2.4.4 Operation substitution

The authors in [59, 60] have proposed an FFT algorithm in which non-trivial multiplication by $\sqrt{2}/2$ is performed by shifters and adders for reducing the power consumption. The algorithm is not very useful because it is not as regular as radix-2 or radix-4.

2.4.5 High radix architecture

High radix FFT algorithms have less multiplications and reduced number of stages than radix-2 but their butterfly is complex having large number of complex multipliers and adders [61]. Radix-4 is a judicious choice and a compromise between the number of stages, multiplications and the complexity of the butterfly for pipelined FFT [61, 62]. Radix-4 algorithm can be very efficiently realised in a pipelined architecture by using only one complex multiplier just like radix-2 [13, 63]. All the pipelined FFT processor architectures in this work are based on this low power radix-4 pipelined architecture.

2.4.6 Reduced precision redundancy

In reduced precision redundancy, a reduced precision replica operates in parallel with the main system in order to detect and correct errors. The authors in [64, 65] have proposed an algorithmic noise-tolerance technique referred to as reduced precision redundancy for compensating the degradation in the SNR at an FFT output due to voltage overscaling (Scaling the supply voltage beyond the critical voltage required for correct operation). The soft errors due to voltage overscaling in the main system appear first in the most significant bits as the arithmetic units are assumed to use least significant bit first computation. This results in large magnitude error thereby degrading SNR severely. This large error can be detected by finding out the difference between the main system and reduced precision replica outputs. If the difference is more than a threshold, the output of the reduced precision system is declared an actual output. It has been assumed that the reduced precision system does not suffer from soft errors. This assumption is valid provided the critical path delay of the reduced precision system is smaller than the clock period. This is quite true for array adder and multiplier architectures where the delay decreases linearly with precision. The proposed technique for the butterfly multipliers in radix-2 FFT processor requires hardware overhead of 40%. The power saving achieved through voltage overscaling more than compensate this overhead.

2.4.7 Asynchronous implementation

Asynchronous techniques are appealing because of the power saving obtained in the clock distribution network. The authors in [66, 67] proposed a low power asynchronous FFT processor architecture. The novelty of the architecture lies in its high localisation of components and pipelining with no need to share a global memory. High throughput is attained using large number of small, local components working in parallel.

2.4.8 Data representation

The complexity of the log-FFT depends on the size of the look-up table which is determined by the bit-width of the LNS. The authors in [68, 69] proposed a low power FFT based on LNS. In coded OFDM, simulation results have shown that there is no degradation in bit error rate performance when only two fractional bits are used for LNS. The look-up table can be easily implemented for such small bit-width. The power consumption in the butterfly module

is reduced by 60% as compared to the fixed point representation.

2.5 Low power techniques and architectures for the FIR filter

There are many techniques for reducing power consumption in FIR filters. Here is a brief description of the commonly used techniques employed to bring down the switched capacitance in FIR filter architectures.

2.5.1 Coefficient ordering

It is possible to reduce the power consumption in an FIR filter by reordering the filter coefficients so as to reduce the number of logic transitions between those filter coefficients used in successive multiplication operations by adopting criteria like minimum Hamming distance [11, 12]. This reduces the number of transitions at the coefficient input of the multiplier resulting in power saving. The choice of an ordered coefficient set in which successive coefficients are highly correlated is both computationally complex and NP-complete for practical size filters. The identification of such an order will require a heuristic search or Genetic algorithms.

2.5.2 Coefficient segmentation

This algorithm decomposes individual coefficients into two primitive sub-components [17]. The decomposition, performed using a heuristic approach, divides a given coefficient such that a part is produced which can be implemented using a single shift operation leaving another part with a reduced word length to be applied to the coefficient input of the hardware multiplier. This results in a significant reduction in the amount of switched capacitance and consequently power consumption. The algorithm has been used with a number of practical FIR filter examples achieving up to 63% power saving at the multiplier level.

2.5.3 Block processing

In the direct form realisation of the FIR filter, a new data sample and the corresponding coefficient are multiplied at each clock cycle followed by accumulation in a conventional multiply and accumulate unit. This leads to high switching activity because both the inputs of the multiplier receive new data at every clock cycle. Any technique, which leads to a reduction in

this switching activity, will directly reduce the power consumption. Another source of power consumption in DSP is the activity on data and address buses. Since each time a new data sample is to be multiplied with a new coefficient, both data and address buses experience high switching activity. This activity is responsible for large power consumption since bus capacitances are usually several orders of magnitude higher than those of internal gates of the circuit. Consequently, considerable amount of power can be saved by reducing the number of memory accesses. There is considerable improvement in the power consumption if the filter outputs are processed in blocks [18] rather than individually. It is possible to retain the coefficients and data at the input of the multipliers for more than one cycle leading to considerable power saving.

2.5.4 Approximate processing

Adaptive filtering algorithms have been generally used to dynamically change the values of the filter coefficients, while maintaining a fixed filter order. This algorithm involves the dynamic adjustment of the filter order in accordance with the stopband energy of the input signal [70]. This approach leads to filtering solutions in which the stopband energy in the filter output may be kept below a specified threshold while using as small a filter order as possible. Since power consumption is proportional to filter order, the approach achieves power reduction with respect to a fixed order filter whose output is similarly guaranteed to have stopband energy below the specified threshold. The overhead associated with the update process is much less as compared to the power saving achieved by using a lower order filter. The filter order is dynamically adjusted by observing the strength of the stopband component of the input signal. When the strength of the stopband component of the signal increases, it is desirable to increase the stopband attenuation of the filter. This can be accomplished by using a higher-order filter. Conversely, the filter order may be lowered when the energy of the stopband component decreases.

2.5.5 Multirate architectures

The multirate architectures enable computationally efficient implementations of FIR filters [71]. Multirate architectures involve implementing the FIR filter in terms of its decimated sub-filters. These filters are derived using Winograd's algorithms for reducing computational complexity of polynomial multiplications. For an N-tap filter, the direct form FIR structure requires N multiplications and $N-1$ additions per output whereas the multirate architecture requires $3N/4$

multiplications and $(3N+2)/4$ additions per output. This reduced computational complexity of the multirate architectures enables reducing the frequency and the supply voltage for the same throughput as the conventional direct form FIR filter, thus significantly reducing the power dissipation.

2.5.6 Coefficient scaling and optimisation

Scaling the coefficients preserves the filter characteristics in terms of passband ripple and stopband attenuation, but results in an overall magnitude gain equal to the scaling factor. The coefficients are scaled in the first stage such that the total Hamming distance between successive scaled coefficients is least. It is followed by slightly modifying the scaled coefficients in the second stage so as to reduce the total Hamming distance while still satisfying the filter characteristics. This modification is an iterative process and continues till no further reduction in total Hamming distance is possible [72].

2.5.7 Filter realisation through differential coefficient

Most realization of FIR filters use the coefficients directly to compute the convolution with the input data. This technique involves the use of various orders of differences between coefficients along with stored intermediate results for computing the convolution [73]. The memory requirement and the number of memory accesses are more as compared to the conventional implementation on account of storage of intermediate results, but the net computations necessary per convolution has gone down as compared to directly using the coefficients. This results in net power reduction at the multiplier due to small values of the differences as compared to the coefficient. This algorithm is useful only if the range of the differences is small compared to the coefficients. In the multiplication operation, a long multiplier is traded for a short one along with overheads in the form of extra memory requirement. If the power savings in multiplication is greater than the net cost due to overheads then a net saving in power is obtained.

2.5.8 Reduced two's complement data representation

A reduced representation for 2's complement numbers has been proposed to avoid sign-extension and the switching of the sign-extended bits [74]. The maximum magnitude of a 2's complement number is detected and its reduced representation is dynamically generated to represent

the signal. A constant error is introduced by the reduced representation and this error is also compensated. The proposed signal representation is more useful in filters with slowly varying coefficients having small magnitude.

2.5.9 Sharing multiplication

This technique is based on computation sharing multiplier, which targets the reduction of redundant computations in FIR filtering [75]. In vector scaling operations, a set of small bit sequences are selected so that the actual multiplication result can be obtained by only add and shift operations. These bit sequences are chosen such that they cover all the coefficients. In TDF filter implementation, a precomputer block contains a set of multipliers for multiplying the data input with the short length bit sequences. This precomputer block is shared by all the coefficients. All the coefficient multipliers can be replaced by just shifters and adders in the presence of this common precomputer block. This sharing of the precomputer block leads to power savings in FIR filters.

2.6 Low power architecture of a MC-CDMA receiver

The low power MC-CDMA receiver architecture is based on a low power algorithm which processes the received symbols in blocks rather than individually [76, 77]. This reduces power by holding one input to the multiplier circuit constant over a number of clock cycles resulting in a power reduction of 50% in the Combiner circuit. This algorithm is also extended to the FFT and results in a power reduction of 13% for the whole receiver.

The processing of symbols in block form is applicable provided that the channel fading is sufficiently slow so as to allow the use of the same channel equaliser coefficients for the entire block length of symbols. The assumption of the slow fading channel is satisfied at pedestrian speed but it may not be true at vehicular speeds. The problem with this architecture is that the memory size in FFT increases linearly with the block size. This means that the reduction in switching activity or power consumption, obtained by increasing the block size, is neutralised by the power consumed in the additional memory required in the FFT processor to support the bigger block size. Moreover, the length of the block size depends upon the channel and therefore will be different at pedestrian and vehicular speeds. Hence, there is a need of some generic techniques for reducing the power consumption of a MC-CDMA receiver. This thesis

Scheme	*csa*		*nbw*		*wall*	
	(mW)	%	(mW)	%	(mW)	%
CONV	0.682	-	0.566	-	0.687	-
ORDER	0.556	18	0.425	25	0.373	46
CSEG	0.331	51	0.325	43	0.597	13
BP	0.466	32	0.388	31	0.438	36
COMB	0.207	70	0.227	60	0.349	49

Table 2.1: *Power consumption analysis for different multipliers.*

explores novel techniques for reducing the power consumption in the individual blocks of a MC-CDMA receiver as well as for the whole receiver.

2.7 Power characteristics of commonly used multipliers

This thesis proposes power saving schemes in all the blocks of a MC-CDMA receiver by reducing their switched capacitances. Most of the low power schemes for the signal processing blocks in a MC-CDMA receiver save power by reducing the power consumed by the multiplier. This section investigates the power saving potential of the commonly used multiplier types after the application of low power schemes to a 73-tap FIR filter. Table 2.1 lists the power consumed by the three commonly used multiplier types namely Carry save array (*csa*), Non-Booth coded Wallace tree (*nbw*) and Booth coded Wallace tree (*wall*) after the application of the low power schemes like Coefficient ordering (*ORDER*) [11, 12], Coefficient segmentation (*CSEG*) [17], Block processing (*BP*) [18] and the combination of Block processing and Coefficient segmentation (*COMB*) to a 73-tap band pass FIR filter. The conventional FIR filter is referred to as *CONV*. The following conclusions can be drawn:

- The multiplier *nbw* consumes least power for the *CONV* architecture. Therefore, *nbw* multiplier is a logical choice for all low power conventional architectures which are not incorporating any low power scheme. The power profile of the multiplier after the application of low power schemes changes significantly. For instance, in Table 2.1, *nbw* consumes least power in the *CONV* architecture but after the application of ordering, the *wall* multiplier has the lowest power consumption. It is interesting to note that *csa* consumes least power consumption for the *COMB* architecture. The power consumption results for the various multipliers are filter dependent and therefore cannot be generalised.

- The *ORDER* scheme is most effective for the *wall* multiplier.
- The *CSEG* scheme is least effective for the *wall* multiplier as compared to other multipliers due to the reduction of switching activity only in the most significant bits of the filter coefficients after *CSEG* rather than in all the bits. The *wall* multiplier saves maximum power if the switching activity reduction takes place in all the bits. The *CSEG* scheme is most effective for the *csa* multiplier type for the filter considered.
- The *wall* multiplier also performs best for *BP* scheme because this scheme also tries to reduce the switching activity at the inputs of the multiplier like the *ORDER* scheme for all the bits.
- The power saving in *csa* multiplier is maximum for the combination of *CSEG* and *BP* because of its good power saving potential for both *CSEG* and *BP* as compared to other multipliers.

It can be further concluded from Table 2.1 that the power consumption in the multiplier can be significantly reduced to different levels by employing the above mentioned low power schemes. The power saving potential of most of the low power schemes are investigated only up to the multiplier level without taking into consideration hardware overhead associated with each of these algorithms. This thesis also investigates the power saving potential of each of these low power schemes on the overall FIR core rather than on only the multiplier for a fair comparison.

2.8 Summary

This chapter described general and block specific low power techniques for reducing the switched capacitance and therefore the power consumption of the multicarrier based wireless receiver. Most of the techniques exploit signal correlations, hardware size reduction through some transformations and shutting down unused portion of the architecture in real time. The FFT is one of the most power consuming block of the multicarrier receiver. This thesis first explores the low power architectures for the FFT processor. The next chapter deals with the overview of the discrete and the fast Fourier transform (FFT).

Chapter 3
The Discrete and the fast Fourier transform

This chapter begins with the review of the discrete Fourier Transform. The remainder of the chapter focuses on the introduction to a collection of algorithms used to efficiently compute the DFT. These algorithms are known as FFT algorithms. This chapter concentrates on only radix-2 and radix-4 FFT algorithms. Many books exist on the topic of FFT [61, 78, 79].

3.1 Overview of DFT

The DFT is one of the most widely used digital signal processing algorithms. DFT is always calculated with the help of FFT, which comprises of a collection of algorithms that efficiently calculate the DFT of a sequence. The DFT operates on an N-point sequence of numbers $x(n)$. This sequence is usually obtained by uniform sampling of a finite period of some continuous function. The DFT of $x(n)$ is also an N-point sequence $X(k)$, and is defined by equation 3.1.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N}, \qquad k = 0, 1, ..., N-1 \tag{3.1}$$

This can also be written as follows.

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \qquad k = 0, 1, ..., N-1, \tag{3.2}$$

Where W_N is defined as

$$W_N = e^{-j2\pi/N}. \tag{3.3}$$

It is easily seen that W_N^{nk} is periodic with period N. The inverse DFT (IDFT) which transforms $X(k)$ back into $x(n)$ is as follows.

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk}, \qquad n = 0, 1, ..., N-1 \tag{3.4}$$

Both $x(n)$ and $X(k)$ are defined only in the interval 0 to $N-1$. However, since $x(n)$ and $X(k)$ are periodic in N, they also exist for all n and k respectively. It is quite clear from equation 3.2 that for each value of k, direct computation of $X(k)$ involves N complex multiplications and $N-1$ complex additions. Consequently, to compute all N values of the DFT requires N^2 complex multiplications and $N(N-1)$ complex additions.

3.2 The Fast Fourier transform (FFT)

The fast Fourier transform is a class of efficient algorithms for computing the DFT. The term FFT was originally used by Cooley and Tukey in their landmark paper [80]. For simplicity, N is chosen to be a power of 2 ($N = 2^m$), where m is a positive integer. With this assumption, it is possible to break $x(n)$ of length N into two sequences of lengths $N/2$. The first sequence contains all even samples of $x(n)(x_{even}(m))$ and the second sequence $x_{odd}(m)$ contains all odd samples. Equation 3.2 can now be written as follows.

$$X(k) = \sum_{n_{even}=0}^{N-2} x(n)W_N^{nk} + \sum_{n_{odd}=1}^{N-1} x(n)W_N^{nk} \tag{3.5}$$

If $2m$ and $2m+1$ are substituted for n in the even and odd summations respectively, then equation 3.5 can be written as follows.

$$X(k) = \sum_{m=0}^{N/2-1} x(2m)(W_N^2)^{mk} + \sum_{m=0}^{N/2-1} x(2m+1)(W_N^2)^{mk})W_N^k \tag{3.6}$$

But $W_N^2 = W_{N/2}$ and hence,

$$X(k) = \sum_{m=0}^{N/2-1} x(2m)W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} x(2m+1)W_{N/2}^{mk} \tag{3.7}$$

Equation 3.7 can also be written in terms of even and odd DFTs as follows.

$$X(k) = F_{even}(k) + W_N^k F_{odd}(k) \tag{3.8}$$

The terms on the right hand side of equation 3.7 corresponds to $N/2$ point DFTs of even ($F_{even}(k)$) and odd ($F_{odd}(k)$) parts of $x(n)$. The direct computation of $F_{even}(k)$ requires $(N/2)^2$ complex multiplications. The same is true for $F_{odd}(k)$. Moreover, there are $N/2$ ad-

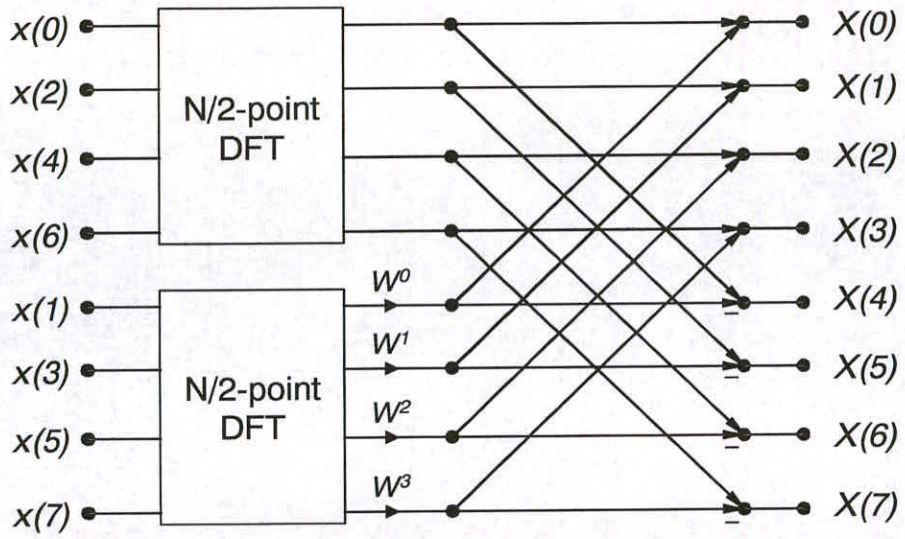


Figure 3.1: *Flow graph of an 8-point FFT calculated using two N/2-point DFTs.*

ditional complex multiplications required to compute $W_N^k F_{odd}(k)$. Hence the computation of X(k) requires $2(N/2)^2 + N/2$ complex multiplications. This first step has resulted in a reduction of the number of multiplications from $(N)^2$ to $(N)^2/2+N/2$, a reduction of 50% for large N. It is possible to reduce the N multiplications by W_N^k to half by exploiting the relationship $W_N^{k+N/2} = -W_N^k$. Figure 3.1 shows the data flow of this algorithm for $N = 8$ in a graphical format. The vertical axis represents memory locations. A total of N-memory locations are needed for storing the sequence $x(n)$. The horizontal axis represents stages of the computation. Data processing starts from the input sequence $x(n)$ on the left side and progresses to the right until the output $X(k)$ is obtained. The calculations after the W_N multiplications are 2-point DFTs.

The number of FFT points N is chosen to be power of 2 and if N is greater than 2 then $x_{even}(m)$ and $x_{odd}(m)$ also have even number of points. It means that they too can be decimated further into their even and odd sequences, and computed from N/4-point DFTs. This decimation can be applied recursively until an even and odd separation results in sequences with two members. This decimation procedure can be applied $log_2(N) - 1$ times, producing $log_2(N)$ stages. From equation 3.2, the final 2-point DFTs are very easy to calculate and requires no multiplication. Figure 3.2 shows the dataflow diagram of the 8-point FFT. It is clear from the dataflow diagram that the number of multiplications per stage equals $N/2$ and therefore the total number of multiplications for computing the N-point FFT reduces to $N/2log_2(N)$.

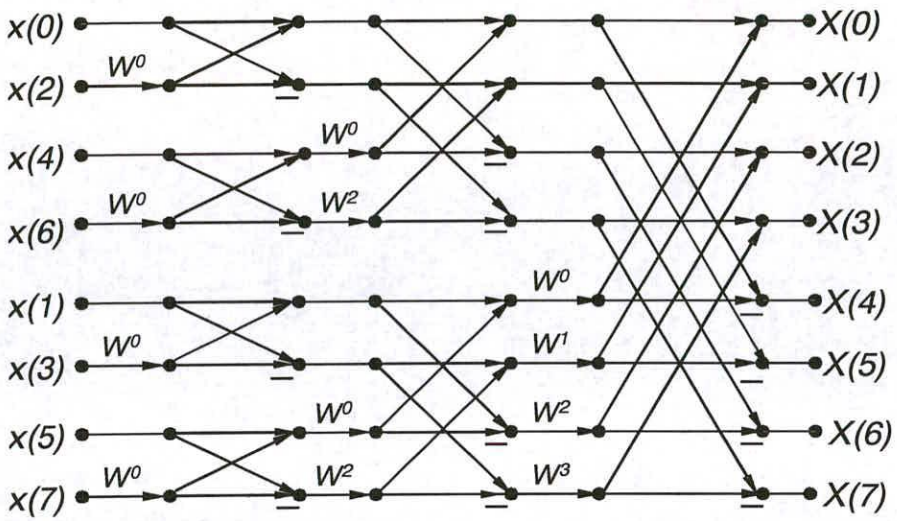


Figure 3.2: *Flow graph of an 8-point radix-2 decimation-in-time FFT.*

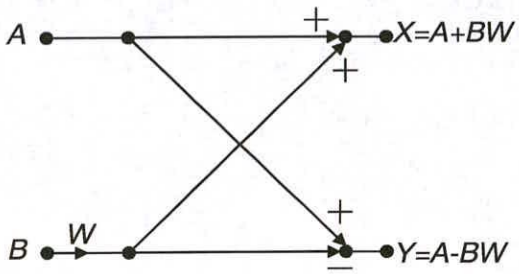


Figure 3.3: *Signal flow representation of a radix-2 decimation-in-time butterfly.*

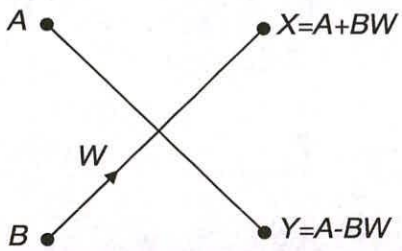


Figure 3.4: *Simplified representation of a radix-2 decimation-in-time butterfly.*

The FFT algorithm described here is called decimation-in-time algorithm since at each stage the input sequence (time sequence) is divided into smaller sequences. Since at each stage the DFT is broken into two smaller DFTs, this FFT is called radix-2 FFT.

The basic operation of the radix-2 decimation-in-time algorithm is the butterfly shown in Figure 3.3 and Figure 3.4 in both signal flow and simplified representations respectively. The butterfly accepts two inputs A and B and generates two outputs X and Y. The multiplication

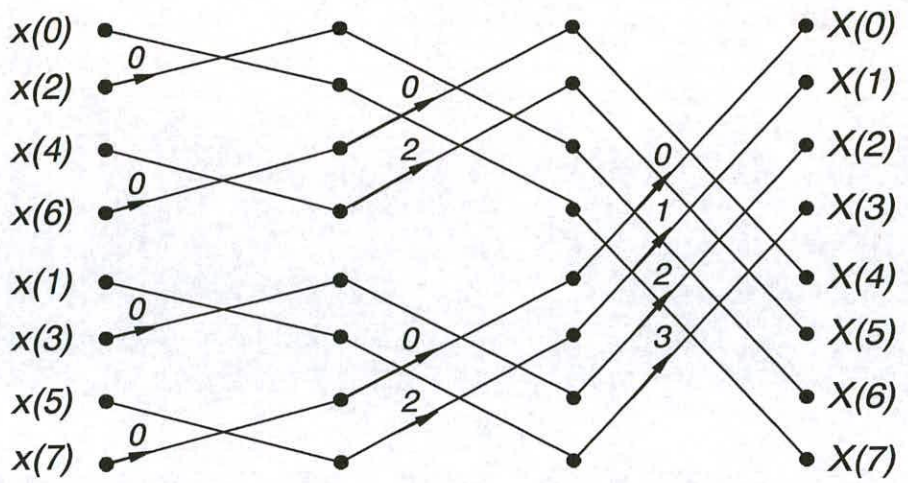


Figure 3.5: *Simplified flow graph of an 8-point radix-2 decimation-in-time FFT algorithm.*

factor W_N^k in the butterfly due to the combining operation is called twiddle factors. Twiddle factors are referred to as coefficient in this thesis. The simplified flow graph of an 8-point radix-2 decimation-in-time FFT algorithm is shown in Figure 3.5. The coefficients (W_N^k) in the simplified flowgraph are represented by their corresponding $'k'$ values for simplicity.

A second algorithm namely radix-2 decimation-in-frquency algorithm is obtained by decimating the frequency values X(k) during each stage. The details of this algorithm can be found in the references [61, 78, 79].

It is clear from the flow graph, shown in Figure 3.5, that the basic operations in the FFT are multiplication of the complex data inputs with the FFT coefficients at each stage of the flow graph followed by their addition or subtraction. The multiplication and addition/subtraction operation are performed by the butterfly module in the FFT processor. A RAM is required to store the data inputs and intermediate outputs after every stage of the signal flow graph. A ROM is needed to store the fixed coefficient. A FSM is also required to properly generate the coefficient and data addresses for providing appropriate data and coefficient to the butterfly module in successive stages of the FFT.

3.3 A unified approach to the FFT

All the different FFT algorithms can be derived from successively representing a one-dimensional array into a two dimensional array. Consider an N-point DFT such that it can be represented

l \ m	0	1	2		M-1
0	X(0)	X(1)	X(2)	···	X(M-1)
1	X(M)	X(M+1)	X(M+2)	···	X(2M-1)
2	X(2M)	X(2M+1)	X(2M+2)	···	X(3M-1)
	⋮	⋮	⋮	···	⋮
L-1	X(L-1)M	X((L-1)M+1)	X((L-1)M+2)	···	X(LM-1)

Figure 3.6: *Row wise arrangement of data array.*

as a product of two integers namely L and M,

$$N = LM$$

The sequence x(n), $0 \leq n \leq N-1$ is a one dimensional array index by 'n'. The same sequence can also be stored in a two dimensional array indexed by l and m, where $0 \leq l \leq L-1$ and $0 \leq m \leq M-1$. Thus the sequence x(n) can be stored in a rectangular array in a number of ways depending upon the mapping of index n to the row index l and column index m. Let us start with the row-wise mapping,

$$n = Ml + m.$$

This results in an arrangement in which the first row contains the first M elements of x(n), the second row contains the next M elements of x(n) and so on as shown in Figure 3.6. The column-wise mapping, shown in Figure 3.7, is given by,

$$n = l + mL.$$

This results in an arrangement in which the first L elements of x(n) are stored in the first column, the next L elements in the second column and so on. The DFT values are also mapped along similar lines. In case of DFT, the mapping is from a index k to a row index p and a column index q, where $0 \leq p \leq L-1$ and $0 \leq q \leq M-1$. The row-wise mapping is given by,

$$k = Mp + q,$$

l \ m	0	1	2		M-1
0	X(0)	X(L)	X(2L)	···	X(M-1)L
1	X(1)	X(L+1)	X(2L+1)	···	X((M-1)L+1)
2	X(2)	X(L+2)	X(2L+2)	···	X((M-1)L+2)
	. . .	. . .	. . .	...	. . .
L-1	X(L-1)	X(2L-1)	X(3L-1)	···	X(LM-1)

Figure 3.7: *Column wise arrangement of data array.*

and the column-wise mapping by,

$$k = qL + p.$$

In row-wise mapping, the first row consists of the first M elements of DFT X(k), the second row consists of the next M elements of X(k) and so on.

If x(n) is mapped column-wise to x(l,m) and X(k) is mapped row-wise to X(p,q), then the DFT can be expressed as a double sum over the elements of the rectangular array multiplied by the corresponding phase factors given by equation 3.9.

$$X(p,q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W_N^{(Mp+q)(mL+l)} \tag{3.9}$$

Now,

$$W_N^{(Mp+q)(mL+l)} = W_N^{LMpm} W_N^{Mlp} W_N^{Lmq} W_N^{lq}$$

But $W_N^{Npm} = W_{N/N}^{pm} = 1, W_N^{Mlp} = W_{N/M}^{lp} = W_L^{lp}$ and $W_N^{Lmq} = W_{N/L}^{mq} = W_M^{mq}$

Equation 3.9 can now be simplified as,

$$X(p,q) = \sum_{l=0}^{L-1} \{W_N^{lq} [\sum_{m=0}^{M-1} x(l,m) W_M^{mq}]\} W_L^{lp} \tag{3.10}$$

The arrangement of the input sequence as a two dimensional array involves the computation of DFT's of length M and L. Equation 3.9 can be understood by dividing the computation into three steps.

- Compute the M-point DFTs $G(l,q) = \sum_{m=0}^{M-1} x(l,m)W_M^{mq}, \quad 0 \leq q \leq M-1$ for each of the rows l = 0,1,...,L-1.
- The M-point DFTs $G(l,q)$ are transformed into a new array $H(l,q)$ by multiplying it with the coefficient W_N^{lq}.

$$H(l,q) = W_N^{lq} G(l,q), \qquad 0 \leq q \leq M-1,\ 0 \leq l \leq L-1$$

- Compute the L-point DFTs for each column $q = 0, 1, ..., M-1$ of the array $H(l,q)$

$$X(p,q) = \sum_{l=0}^{L-1} H(l,q)W_L^{lp}$$

The decomposition of larger DFT into smaller DFT always leads to a reduction in computational complexity as already discussed. When N is a highly composite number such that it can be factored into a product of prime numbers $N = r_1 r_2 ... r_v$ then the above decomposition can be carried out (v-1) more times leading to more computationally efficient algorithm. Further decomposition of the rectangular array involves segmentation of each row and column of the rectangular array into smaller rectangular arrays. The decomposition ends when N is factored into its prime factors.

The already discussed radix-2 decimation-in-time algorithm is obtained by selecting L=2 and M= N/2 in the unified approach for each of the $log_2 N$ stages. This leads to $n = 2m + l$ and $k = (N/2)p + q$.

3.3.1 Radix-4 decimation-in-time FFT algorithm

When the number of data points N in the DFT is a power of $4(N = 4^v)$, then it is more efficient computationally to employ a radix-4 algorithm instead of a radix-2 algorithm. A radix-4 decimation-in-time FFT algorithm is obtained by splitting the N-point input sequence $x(n)$ into four subsequences $x(4n)$, $x(4n+1)$, $x(4n+2)$ and $x(4n+3)$. The radix-4 decimation-in-time FFT algorithm is obtained by selecting L=4 and M=N/4 in the unified approach. This leads to $n = 4m + l$ and $k = (N/4)p + q$. The radix-4 algorithm is obtained by following the decomposition procedure outlined in the previous section v times recursively. The signal flow graph of a 16-point radix-4 decimation-in-time algorithm is shown in Figure 3.8. The details can be found in [79].

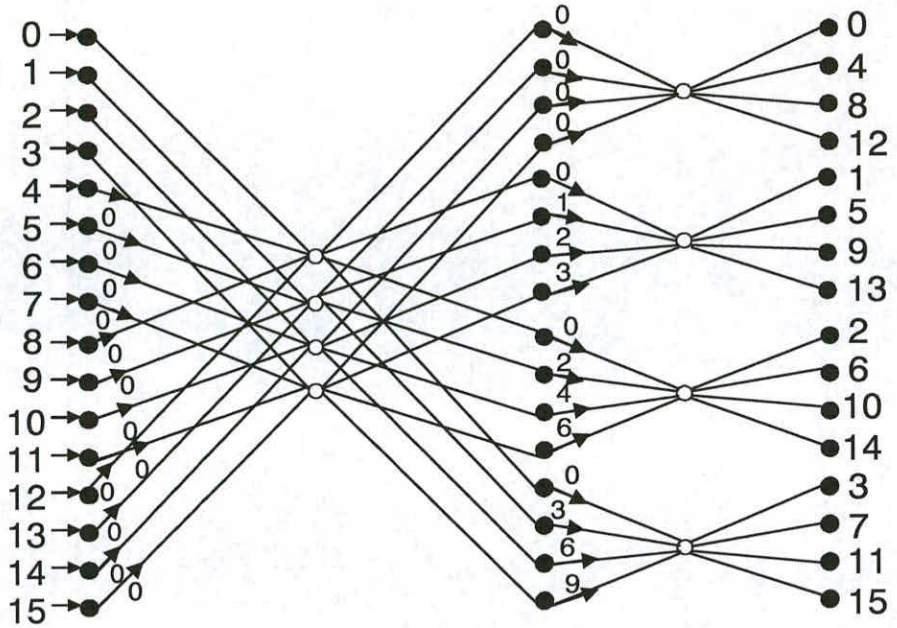


Figure 3.8: *Flow graph of a 16-point radix-4 decimation-in-time FFT algorithm.*

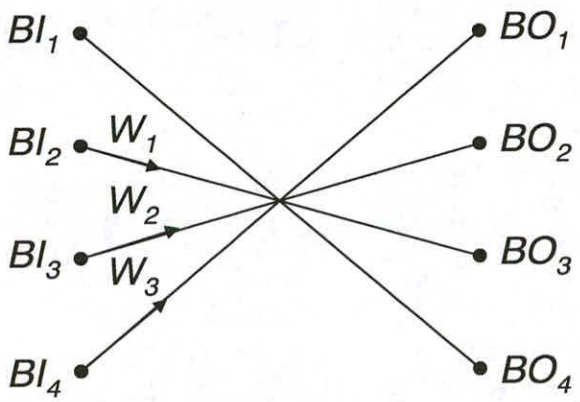


Figure 3.9: *Radix-4 decimation-in-time butterfly.*

The radix-4 butterfly, shown in Figure 3.9, is constructed by merging 4-point DFT with associated coefficients between DFT stages. The four outputs of the radix-4 butterfly namely BO_1, BO_2, BO_3 and BO_4 are expressed in terms of its inputs BI_1, BI_2, BI_3 and BI_4 as follows:

$$BO_1 = BI_1 + BI_2W_1 + BI_3W_2 + BI_4W_3 \tag{3.11}$$

$$BO_2 = BI_1 - iBI_2W_1 - BI_3W_2 + iBI_4W_3 \tag{3.12}$$

$$BO_3 = BI_1 - BI_2W_1 + BI_3W_2 - BI_4W_3 \tag{3.13}$$

$$BO_4 = BI_1 + iBI_2W_1 - BI_3W_2 - iBI_4W_3 \tag{3.14}$$

The radix-4 butterfly requires three complex multiplications. The multiplication with 'i' is accomplished by negation and swapping of the real and imaginary parts. Radix-4 has a computational advantage over radix-2 because radix-4 butterfly does the work of four radix-2 butterflies using three multipliers instead of four multipliers in four radix-2 butterflies [62]. On the negative side, a radix-4 butterfly is more complicated to implement than a radix-2 butterfly.

While radix-2 and radix-4 FFTs are certainly the most widely known algorithms, it is also possible to design FFTs with even higher radix butterflies. They are not often used because the control and dataflow of their butterflies are more complicated and the additional efficiency gained diminishes rapidly for radices greater than four [62].

3.4 Summary

This chapter introduced DFT and FFT algorithms for efficiently computing the DFT. The radix-2 decimation-in-time FFT algorithm is covered in detail whereas the overview of the radix-4 decimation-in-time algorithm is given. It has been shown that radix-4 algorithm is more computationally efficient than radix-2 alogrithm. The next chapter covers the proposed low power techniques for the FFT processor.

Chapter 4
Low power schemes for FFT processor

This chapter proposes three different schemes for reducing the power consumption in an FFT processor. The first scheme namely, *order based processing of coefficients*, modifies the traditional coefficient ordering for the FFT processor for achieving much higher reduction in switching activity and hence the power consumption. The second scheme namely, *Coefficient memory reduction scheme*, reduces the size of the coefficient memory from $N/2$ to $(N/8+1)$ locations in an N-point FFT processor thereby saving both area and power for long FFTs. The third scheme namely, *Coefficient addressing scheme*, simplifies the coefficient address generation in a single butterfly FFT processor thereby saving both area and power.

This chapter is organised into three sections. The first section describes the *Order based processing scheme* and the results to show its effectiveness in reducing the switching activity as compared to the traditional order based processing scheme. The architectures of the low power FFT processors based on the *Order based processing scheme* will be described in Chapter 5 and Chapter 6. The second section proposes a *Coefficient memory reduction scheme* and its hardware implementation. The third section introduces a *Coefficient addressing scheme* and its hardware implementation. It is important to note that 16-bit two's complement fixed point number representation is used in this thesis.

4.1 Order based coefficient processing scheme

The multiplier is one of the most power consuming blocks in FFT processor. Order based processing is a very effective way of reducing the switching activity between the fixed successive coefficient inputs to the multiplier. The basic idea is to order the multiplication operations such that the switching activity is minimum between successive coefficients. For an N-point FFT, the number of possible ways to arrange the coefficients equals N! but the number of distinct coefficient orders equal (N-1)!/2. Thus the choice of the highly correlated coefficient set is a computationally complex NP-complete problem requiring some heuristic search algorithm. In

the conventional order based processing approach, the nearest neighbour selection heuristic is used for ordering the coefficient set so as to minimise the Hamming distance (Bit transitions or switching activity) between successive coefficients fed to the multiplier blocks of the butterfly module. This work is reported in [81].

In an FFT processor, the positions of the real and imaginary parts of the coefficient set cannot be changed independently (Every real part has a corresponding imaginary part and vice versa). The reduction in switching activity of the coefficient set by the conventional order based processing approach is insignificant because the switching activities of the real and imaginary parts of the coefficient set are almost complementary. For instance, if one changes the order of only the imaginary parts of the coefficient set for minimum switching activity, the new order of the real parts will be such that the net activity of the overall coefficient set will not reduce substantially and vice-versa. The proposed scheme addresses the above mentioned problem by either using the real part of the coefficient or its two's complement on the basis of minimum Hamming distance with the preceding real part. The scheme can be best illustrated with the help of a flowchart shown in Figure 4.1. Let us assume that X_R and X_I be the $N/2$ element real and imaginary part arrays respectively for an N-point FFT. Also assume the presence of the following functions for understanding the flowchart:

Ham(i,j) - Hamming distance between array elements $X_I(i)$ and $X_I(j)$.

HamN(i) - Hamming distance between elements $X_R(i)$ and $X_R(i-1)$.

HamC(i) - Hamming distance between elements $-X_R(i)$ and $X_R(i-1)$.

A($XI(i)$) - Number of 1's in array element $X_I(i)$.

The flow chart is divided into four sections. Section I selects the first imaginary part of the ordered coefficient set on the basis of minimum number of 1's. Section II arranges the remaining imaginary parts of the coefficient set on the basis of minimum Hamming distance between successive imaginary coefficients. After sections I and II, the imaginary parts of the coefficient set are ordered and are stored in array X_I. Section III deals with the selection of the first real part corresponding to the already ordered first imaginary part of the coefficient set. Either $X_R(0)$ or its two's complemented value $-X_R(0)$ is selected on the basis of lesser number of 1's. A flag bit is asserted in case the complemented value is selected. Section IV chooses subsequent real parts or their two's complemented values corresponding to their imaginary parts on the basis of

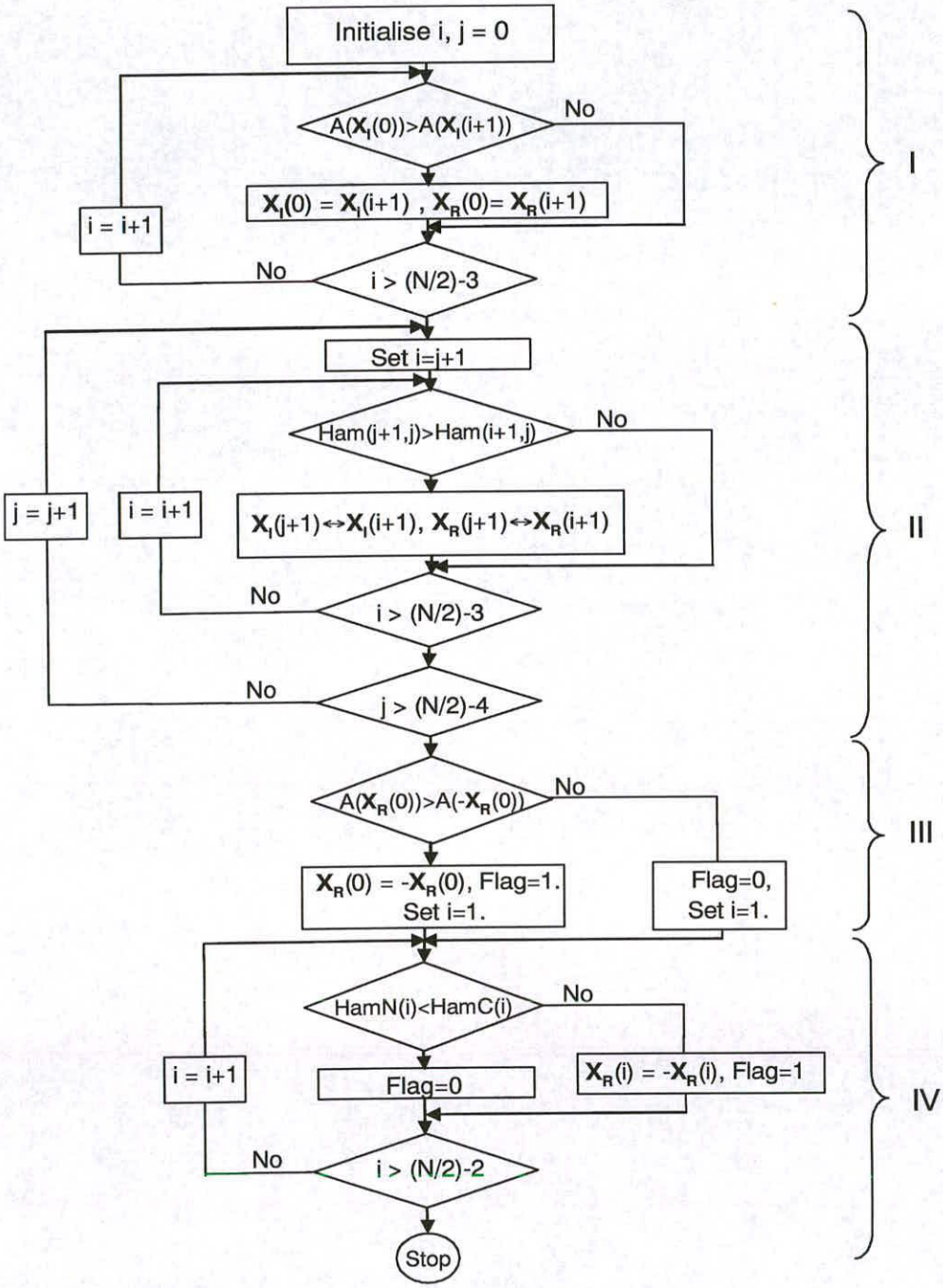


Figure 4.1: *Flow chart of the order based processing scheme.*

Address	*Coefficient set (Real, Imag)*	*16-bit quantised coefficient set (Real, Imag)*	*Others scheme (ordered set) (Real, Imag)*	*Our scheme (ordered set) (flag,Real, Imag)*
0000	1.0,0.0	7ffff,0000	0000,8000	1,8001,0000
0001	.98,-0.19	7d89,e706	7641,cf04	0,0000,8000
0010	.92,-0.38	7641,cf04	7d89,e706	0,7641,cf04
0011	.83,-.55	6a6d,b8e3	89be,cf04	1,7642,cf04
0100	.71,-.71	5a82,a57d	8276,e706	0,7d89,e706
0101	.55,-.83	471c,9592	e706,8276	1,7d8a,e706
0110	.38,-.92	30fb,89be	cf04,89be	1,e707,8276
0111	.19,-.98	18f9,8276	471c,9592	0,e706,8276
1000	0.0,-1.0	0000,8000	7fff,0000	1,cf05,89be
1001	-.19,-.98	e706,8276	18f9,8276	0,cf04,89be
1010	-.38,-.92	cf04,89be	30fb,89be	1,471d,9592
1011	-.55,-.83	b8e3,9592	b8e3,9592	0,471c,9592
1100	-.71,-.71	a57d,a57d	9592,b8e3	1,6a6e,b8e3
1101	-.83,-.55	9592,b8e3	6a6d,b8e3	0,6a6d,b8e3
1110	-.92,-.38	89be,cf04	a57d,a57d	0,a57d,a57d
1111	-.98,-.19	8276,e706	5a82,a57d	1,a57e,a57d

Table 4.1: *Listing of the ordered coefficient sets obtained by different order based processing schemes for a 32-point FFT processor as an example.*

minimum Hamming distance with the already ordered preceding real part. Table 4.1 depicts the coefficient sets obtained by application of our Order based processing scheme (*Our*) and the conventional order based processing schemes (*Others*), to a 32-point FFT processor as an example. The *Others* ordered set is obtained by using the minimum Hamming distance approach on successive coefficients. *Our* ordered set is obtained by following the procedure outlined in Figure 4.1. A flag bit is stored along with the ordered coefficients to indicate the form of the real part of the coefficient. *Our* scheme is most effective in the last stage of the signal flowgraph of the radix-2 decimation-in-time FFT algorithm where the coefficient switching activity is most intense (The coefficients are different for each butterfly unlike other FFT stages as shown in Figure 4.2).

4.1.1 Results

Table 4.2 lists the switching activity reductions obtained by following the *Others* and *Our* order based processing schemes for different lengths of the FFT processors. It is clear from the table that the switching activity reduction is close to 50% for *Our* scheme compared to only around

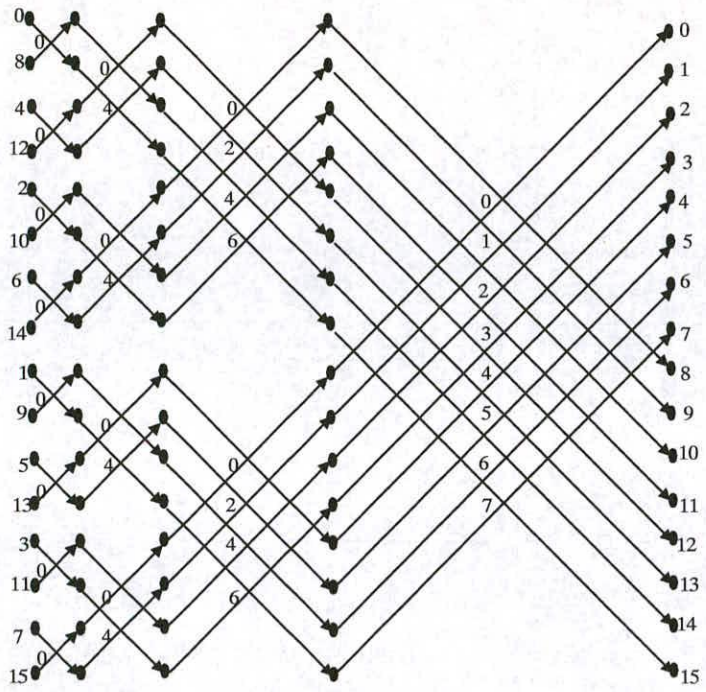


Figure 4.2: *Signal flow graph of a 16-point FFT processor.*

FFT size	*Total switching activity*	*Others scheme*		*Our scheme*	
		Switching activity	*[%] reduction*	*Switching activity*	*[%] reduction*
16	126	120	05	68	46
32	240	204	15	126	48
64	476	368	23	222	53
128	828	672	19	424	49
256	1520	1196	21	780	49
512	2874	2106	27	1492	48
1K	5250	3934	25	2786	47

Table 4.2: *Switching activity comparison of different schemes for different lengths of the radix-2 FFT processor.*

20% for the *Others* scheme for all FFT lengths. Table 4.3 lists the reduction in switching activity obtained by *Others* and *Our* schemes for different coefficient wordlengths for a 128-point FFT processor. It is clear that the switching activity reduction remains almost the same for different

Wordlength	*Total switching activity*	*Others scheme*		*Our scheme*	
		Switching activity	*[%] reduction*	*Switching activity*	*[%] reduction*
8	328	266	19	222	32
10	448	352	21	244	46
12	552	444	20	296	46
14	696	566	19	372	47
16	828	672	19	424	49
18	970	806	17	510	48
20	1084	884	18	560	48

Table 4.3: *Switching activity comparison of different schemes for different wordlengths for a 128-point radix-2 FFT processor. The coefficient set is obtained by rounding to the nearest integer.*

Wordlength	*Total switching activity*	*Others scheme*		*Our scheme*	
		Switching activity	*[%] reduction*	*Switching activity*	*[%] reduction*
8	334	258	23	146	56
10	450	352	22	220	51
12	598	446	25	244	59
14	722	568	21	316	56
16	868	670	23	366	58
18	942	788	17	448	52
20	1106	890	20	486	56

Table 4.4: *Switching activity comparison of different schemes for different wordlengths for a 128-point radix-2 FFT processor. The coefficient set is obtained by rounding up to the nearest integer.*

wordlengths. Table 4.4 lists the coefficient switching activity obtained by using the coefficient set obtained through rounding up to the nearest integer instead of traditional rounding to the nearest integer. The switching activity reduction is much better by the later method because the quantised representation of a negative number and its generation through two's complement becomes exactly identical. There always exists a difference of unity between the representation and its generation through two's complement in the former method leading to inferior switching activity reductions.

'g'	*Address*	*Coefficient set (Real, Imag)*	*Our scheme*	*Others scheme*
0	0000	1.0,0.0		
1	0001	.98,-.19	Block	
2	0010	.92,-.38	I	
3	0011	.83,-.55		Block
4	0100	.71,-.71		A
0	0101	.55,-.83	Block	
1	0110	.38,-.92	II	
2	0111	.19,-.98		
0	1000	0.0,-1.0		
1	1001	-.19,-.98	Block	
2	1010	-.38,-.92	III	
3	1011	-.55,-.83		Block
4	1100	-.71,-.71		B
0	1101	-.83,-.55	Block	
1	1110	-.92,-.38	IV	
2	1111	-.98,-.19		

Table 4.5: *Description of the various memory organisation schemes.*

4.2 Coefficient memory reduction scheme

The power consumption in an FFT processor can also be reduced by decreasing the coefficient memory size required for its implementation. The proposed scheme reduces the size of the coefficient memory from the existing $N/4$ locations [14, 82] to $((N/8) + 1)$ locations for an N-point FFT processor. The power and area saving is significant as compared to the existing scheme for long FFTs.

The radix-2 FFT coefficients are expressed as follows:

$$W_k = e^{(-j2\pi k/N)}$$

Where k varies from 0 to $N/2-1$ giving rise to $N/2$ coefficients for an N-point FFT where 'N' indicates the number of data points or the length of the FFT. The memory required to store these coefficients will thus require N/2 locations for *Conv* implementation. Each coefficient memory location stores the real and the imaginary parts of the coefficient. The values of the coefficients for a 32-point FFT in two's complement form are given in Table 4.5. The partitioning of coefficients into blocks (as shown in Table 4.5) is applicable to all FFT lengths. *Others* proposed to divide the memory into two identical blocks namely A and B [14, 82]. It is clear that the

coefficient values in block B can be generated from those in block A by interchanging the real and imaginary parts of the coefficients and by also complementing the real part before its assignment to the imaginary part corresponding to block B. Hence, only block A needs to be stored. In *Our* scheme, the memory is partitioned into four blocks (Block I to Block IV) rather than two as shown in Table 4.5. The memory size in *Our* scheme is reduced to $((N/8)+1)$ locations (Only Block I is needed) from the $N/4$ locations proposed by *Others* schemes (Only Block A is needed). Using *Our* scheme, there is a need to store only coefficient values in block I and the rest of the coefficient values in other blocks and their corresponding first block addresses can be generated by following the general procedure given below. This procedure can be explained with the help of a 32-point FFT example given in Table 4.5. Let the complex coefficient values in terms of the real and imaginary parts be represented as in equation 4.1.

$$Z_{bg} = R_{bg} + jI_{bg} \tag{4.1}$$

Where 'b' indicates the memory block number and 'g' is an index which points to the coefficient values within individual blocks. The first coefficient value in each block has an index 'g' equal to zero. The first block coefficient values are obtained from equation 4.1 by replacing 'b' with '1' as shown in equation 4.2.

$$Z_{1g} = R_{1g} + jI_{1g} \qquad 0 \leq g \leq N/8 \tag{4.2}$$

Let the coefficient memory address generated and the actual block address be represented by an n-bit array 'A_m' and an (n-1) bit array 'A_{bg}' respectively. The coefficient memory block address is always one bit less than the conventional memory address as the block size is limited to $((N/8)+1)$ instead of $N/2$. The corresponding addresses of the coefficient values in the first block are given by the following equation.

$$A_{1g}[n-2:0] = A_m[n-2:0] \quad Where \quad n = log_2(N/2),\ 0 \leq g \leq N/8,\ 0 \leq m \leq N/8$$

The second block coefficient values can be obtained in terms of the first block coefficient values by performing the following substitution in the right-hand side of equation 4.2: $R_1 \rightarrow\sim I_1$, $I_1 \rightarrow\sim R_1$ and $g \rightarrow (N/8-1-g)$. The symbol '$\sim$' here corresponds to a complement operation. This can also be verified from Table 4.5. The resulting equation is as follows:

$$Z_{2g} = [\sim I_{1(N/8-1-g)}] + j[\sim R_{1(N/8-1-g)}] \qquad 0 \leq g \leq ((N/8)-2)$$

When the coefficient memory address generator proceeds to generate the address in the second block, the corresponding address in the first block (Only Block I is stored) are obtained by taking the two's complement of the coefficient memory address as follows:

$$A_{2g} =\sim A_m[n-2:0]+1 \qquad 0 \leq g \leq ((N/8)-2),\ ((N/8)+1) \leq m \leq ((N/4)-1)$$

Similarly, the coefficient values in the third block are obtained from the first block coefficient values using equation 4.2 as follows:

$$Z_{3g} = [I_{1g}] + j[\sim R_{1g}] \qquad 0 \leq g \leq N/8$$

When the coefficient memory address generator proceeds to generate the addresses in the third block, the corresponding first block addresses are obtained as follows:

$$A_{3g} = A_m[n-2:0] \qquad 0 \leq g \leq N/8,\ N/4 \leq m \leq 3N/8$$

Similarly, the fourth block coefficient values are obtained in terms of the first block coefficient values again using equation 4.2 as follows:

$$Z_{4g} = [\sim R_{1(N/8-1-g)}] + j[I_{1(N/8-1-g)}] \qquad 0 \leq g \leq ((N/8)-2)$$

Similarly for the fourth block, the corresponding addresses in the first block are given by the following equation.

$$A_{4g} =\sim A_m[n-2:0]+1 \qquad 0 \leq g \leq ((N/8)-2),\ ((3N/8+1) \leq m \leq (N/2)-1)$$

Let us consider an example to understand the above scheme. In Table 4.5, the real and imaginary coefficient values for a 32-point FFT (n equals 4) corresponding to index '0' of the second block are .55 and $-.83$ respectively. The coefficient address $A_5[2:0]$ for these values is 101. These coefficient values can be generated by moving to index 3 of the first block and then by complementing and interchanging the real and imaginary values stored at this address. The appropriate first block address $A_{20}(011)$ corresponding to index 3 is generated by taking the two's complement of the coefficient memory address $A_5[2:0]$. The blocks can be easily identified with the help of the higher two bits along with the all zero combination of the remaining bits. Hence, only block I needs to be stored and the remaining blocks can be generated by designing

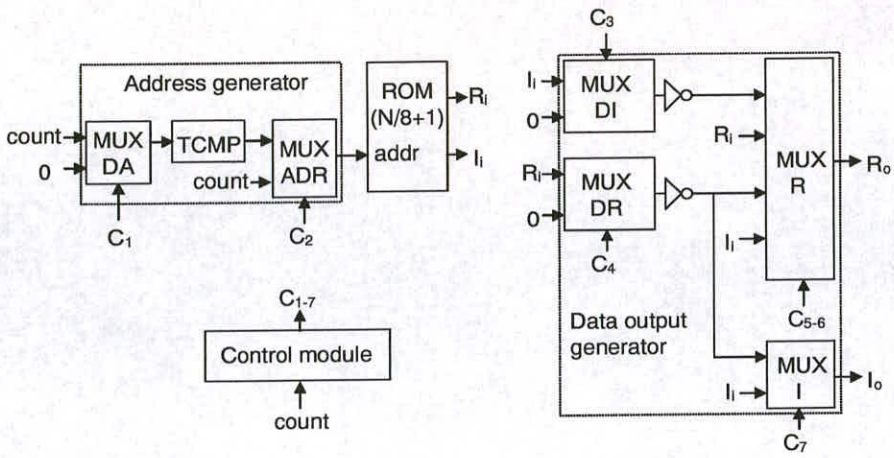


Figure 4.3: *Hardware implementation of* Our *coefficient memory reduction scheme.*

an additional hardware for implementing the described procedure. The delay introduced by the additional block is not of much consequence because the coefficient delay in a typical FFT implementation is half that of the data delay [83]. Moreover, the access time of memory in our case is much lower on account of its smaller size. This work is reported in [84, 85]

4.2.1 Memory implementation

The block diagram of the hardware required to implement the memory in *Our scheme* is shown in Figure 4.3. The major hardware modules are as follows:

- Coefficient ROM of size ((N/8)+1) locations.
- Address generator for generating the appropriate address corresponding to all the partitioned blocks of the original coefficient memory.
- Data output generator for modifying the output of the coefficient ROM as per the partitioned blocks of the original coefficient ROM.
- Control module provides select lines to the multiplexers as per the original coefficient ROM address generated by an external counter.

The address generator module comprises of a multiplexer (*DA*) to feed the count value to the two's complementor module (*TCMP*) only during the execution of blocks II and IV of Table 4.5. The module *DA* keeps all the inputs to *TCMP* block at '0' for blocks I and III thereby reducing

the switching activity. The mutiplexer (*ADR*) chooses the address or its two's complement depending upon the block in Table 4.5.

The data output generator contains two multiplexers namely (*DI*) and (*DR*) to feed the imaginary and real parts of the coefficient memory output for inversion only whenever required as per Table 4.5. This approach again reduces the switching activity at the input of the inverter. The multiplexers (*R*) and (*I*) are used to select the appropriate real and imaginary part of the output of the coefficient ROM depending upon the blocks of Table 4.5 to generate the final real (R_o) and imaginary (I_o) memory outputs.

The control module is responsible for generating the select lines for all the multiplexers as per the address count. The select lines for the address and data output generator depend upon the block in which the current address count is located. The select lines vary from one block to the other.

4.2.2 Design flow

The discussion of the design flow is important at this stage because it helps in understanding the results presented in the present and subsequent chapters of this thesis. The basic design flow is illustrated with the help of a flowchart shown in Figure 4.4. The conventional and low power architectures are defined at the register transfer level (RTL) using Verilog hardware description language. The functionally correct designs are then synthesised using SYNOPSYS *DesignCompiler* (dc-shell) or Cadence *BuildGates* to convert the RTL description into a gate level netlist. The synthesis tool also generates a delay file in standard delay format (SDF) for more accurate gate level timing simulation. The *DesignCompiler* also provides a timing constraint file in SDF format for the layout tool (Silicon Ensemble). The functionality and timing of the designs are again verified at the gate level using *Verilog-XL* simulator. In case of any problem, either the RTL code or/and the synthesis timing constraints have to be modified.

The power consumption for some designs is computed at the gate level. The power consumption is estimated with the help of SYNOPSYS *DesignPower*. It uses a gate level netlist along with the net switching activity obtained after gate level simulation to compute the dynamic power. The switching activity is computed by defining the whole design as the toggle region in the testbench. The toggling of each and every net is recorded in the switching activity file called SAIF (Switching activity interchange format). The accuracy of the results will improve if the

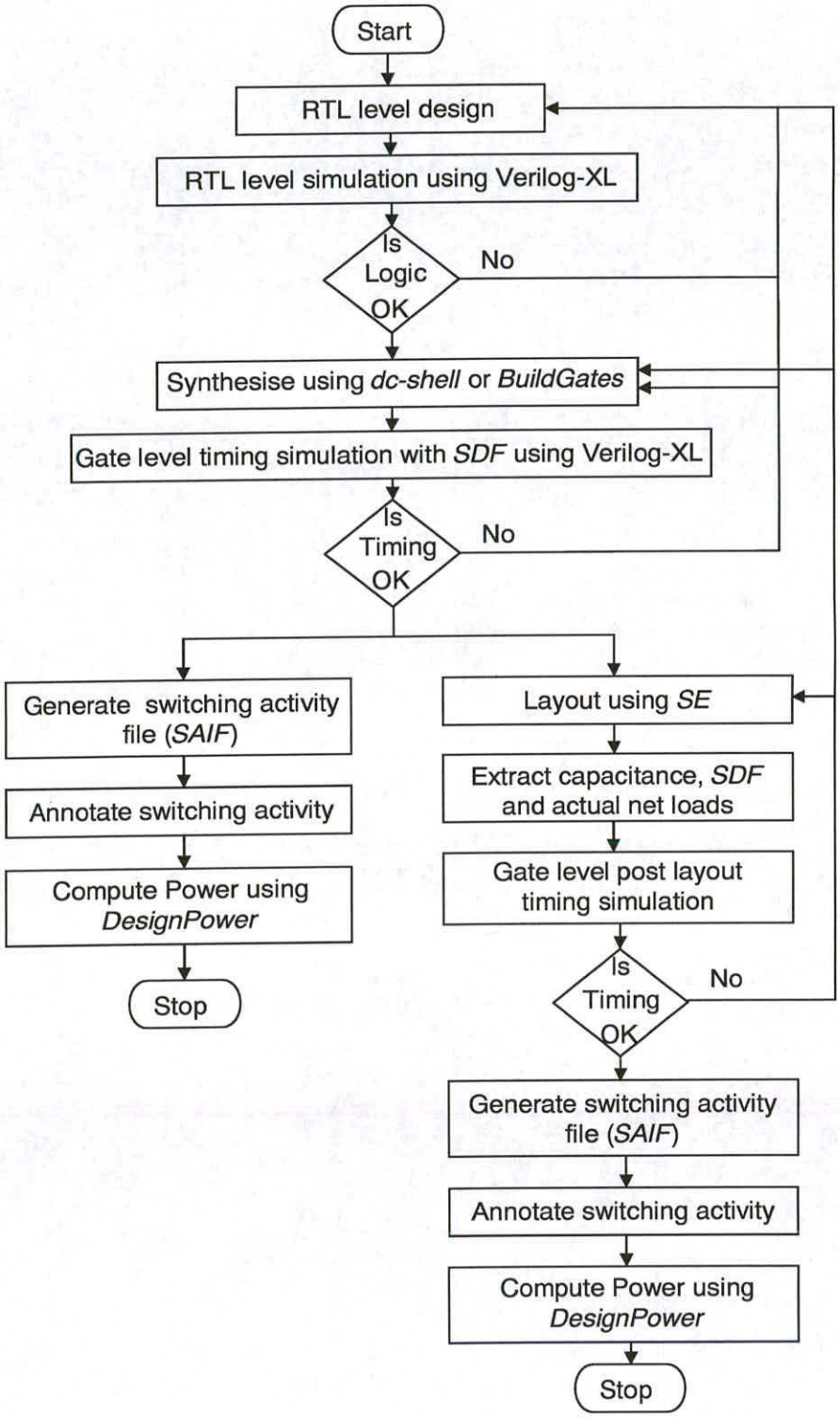


Figure 4.4: *Flow chart depicting the design flow.*

FFT size	*Conv scheme* μW	*Others scheme* μW	*Our scheme* μW	*% reduction*
64	183	228	283	-24
128	222	261	355	-36
256	286	322	406	-26
512	372	383	474	-24
1K	714	485	482	+.62
2K	867	934	623	+33
4K	3403	1121	797	+29
8K	5296	3331	1353	+59

Table 4.6: *Comparison of the various schemes in terms of power.*

power estimation is carried out after layout. Cadence *Silicon Ensemble* is used to convert the gate level netlist into layout with the help of the timing constraint file. The modified netlist, the post layout SDF and the file containing the extracted net capacitances are then used to verify the post layout functionality of the designs and also to estimate the power consumed by the designs using *DesignPower*. The dynamic power comprises of two components namely internal power and switching power. The internal power is the power consumed within the boundary of a cell. It is basically a combination of short circuit power and the power needed for charging and discharging the capacitances inside the cell. The switching power is the power consumed in charging and discharging the load capacitance at the output of the cell.

4.2.3 Results

The scheme has been implemented in register transfer level Verilog hardware description language for different FFT lengths and then synthesized using Cadence *BuildGates* with 0.35μ Alcatel MTC 45000 CMOS technology library. Power evaluation was carried out using *Synopsys DesignPower* for the circuit netlist. Gate level simulations were carried out for one million clock cycles using a supply voltage of 3.3V and a clock frequency of 10MHz. The same procedure was followed for the Conventional (*Conv*) and Ma's and Parhi's (*Others*) approaches. The comparative results in terms of power and area for the different FFT lengths are given for all the schemes in Table 4.6 and Table 4.7 respectively. It is clear from Table 4.6 and Table 4.7 that *Our* scheme proves to be beneficial for 1K points onwards both in terms of area and power. The savings in area ranges from 48% to 54% whereas the power saving varies from 0.62% to 59% for longer FFT's (1K points onwards). The saving in area goes on increasing with FFT

FFT size	*Conv scheme [n]*	*Others scheme [n]*	*Our scheme [n]*	*% reduction*
64	259	212	179	16
128	520	343	256	25
256	1012	605	383	37
512	2222	1070	652	39
1K	4452	2232	1168	48
2K	8729	4449	2272	49
4K	19986	8526	4391	49
8K	34890	18366	8378	54

Table 4.7: *Comparison of the various schemes in terms of area. The area is expressed in equivalent nand gates[n].*

Wordlength	*Conv scheme*		*Others scheme*		*Our scheme*		*[%] area saving /Others*	*[%] power saving /Others*
	area [n]	*power* μW	*area* [n]	*power* μW	*area* [n]	*power* μW		
8	4149	510	1538	378	912	262	41	31
12	6700	640	3501	847	1575	428	55	49
16	8729	867	4449	934	2272	623	49	33

Table 4.8: *Power and area saving of Our scheme as compared to Others scheme for different coefficient wordlengths for a 2K-point radix-2 FFT processor.*

length because the additional hardware required to implement *Our* scheme remains almost fixed. Table 4.8 lists the power and area consumed by the different schemes for three different wordlengths for a 2K-point FFT processor. It is clear from this table that maximum power saving of 49% is obtained for an intermediate wordlength of 12-bits. This indicates that this scheme is more useful for intermediate wordlengths of 12-bits rather than extreme wordlengths of 8 or 16-bits.

4.3 Coefficient addressing scheme

The coefficient memory addressing is important for enhancing the performance of FFT processors [14]. The power consumption is reduced by minimising the hardware required to carry out coefficient address generation in a radix-2 single butterfly FFT processor. The most popular and computationally efficient method of coefficient address generation was proposed by Cohen [15]. According to this method, address generation occurs through the application of variable shifts to address lines through a cascade of Barrel shifters. The work in [14] modified the data generation scheme proposed by Cohen while retaining the coefficient address generation method. This section presents a novel scheme for coefficient address generation in an FFT processor. The proposed addressing scheme involves manipulation of the address lines taking into consideration coefficient addresses required at various FFT stages. It has been demonstrated in this section that the scheme can be implemented more efficiently with much reduced hardware than the *Cohen's* scheme leading to more power and area efficient realisation of FFT processors.

4.3.1 Detailed design

The radix-2 FFT, shown in Figure 4.2 for N equals to 16, is an efficient way to compute an N-point DFT. It has been assumed that the data inputs are arranged in bit reverse order and the outputs are produced in normal order. The basic operations in FFT are multiplication of the complex data inputs by the FFT coefficients at each stage in the signal flowgraph followed by their summation or subtraction and associated data and coefficient address generation. The coefficient and data address generation logic are required to be fast, area and power efficient in order to realise fast, miniaturised and low power FFT processors which could be integrated into complex VLSI systems. The coefficient address generation in an FFT is accomplished by partitioning a p-bit counter into two sections as shown in Figure 4.5. The more significant counter section comprises of $(C_{p-1}, ..., C_b)$ bits whereas the lower section has $(C_{b-1}, ..., C_0)$ bits. Let the more and less significant counter section bits be represented by a $(p-b)$-bit array x and a b-bit array y respectively. The coefficient address (CA) is then expressed as follows:

$$CA = F(x, y) \tag{4.3}$$

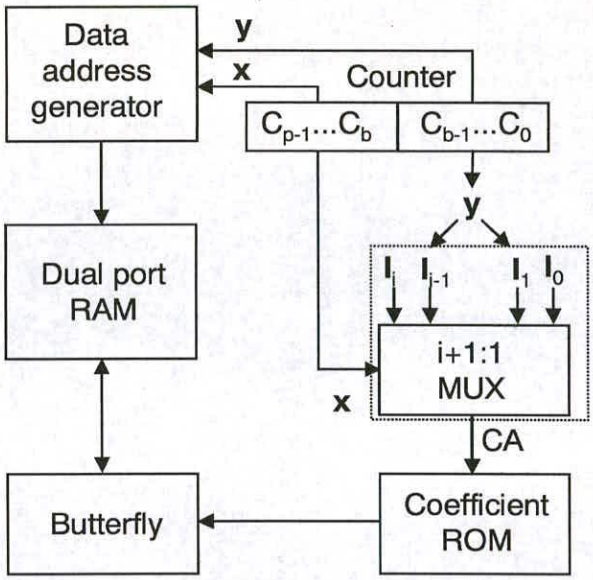


Figure 4.5: *Architecture of a single butterfly based radix-2 FFT processor. The coefficient address generator is enclosed by a dotted rectangle.*

Let us also assume that i = Decimal equivalent of x, which is always greater than or equal to 'b' depending upon the FFT size, then equation 4.3 can be written as follows:

$$CA = I_i \tag{4.4}$$

Where I_i is a b-bit array, which is expressed in terms of an array y according to the following set of equations:

$$I_0 = (0...0)$$
$$I_1 = (y[b-1]0...0)$$
$$I_2 = (y[b-1]y[b-2]0...0)$$
$$\cdots$$
$$I_{b-1} = (y[b-1]y[b-2]y[b-3]...y[1]0)$$
$$I_b = y$$
$$I_{b+1} = (X...X)$$
$$\cdots$$
$$I_i = (X...X) \quad \text{Where } N = \text{FFT size}, \; b = log_2(N/2)$$

X = Don't care bit and $p = b + (log_2(log_2(N)))$ (rounded up to the nearest integer)

The set of operations described by equation 4.4 could be realised in hardware by using a single multiplexer based coefficient address generation logic as shown in Figure 4.5. The input channels of the multiplexer are set according to I_i and are selected as per x. In order to illustrate this scheme, consider a 16-point radix-2 FFT signal flowgraph, shown in Figure 4.2 as an example. In the first stage, only the first coefficient with an address value $0(W^0)$, is required for all the eight butterflies. In the second stage, W^0 and W^4 coefficients are required. In the third stage, W^0, W^2, W^4 and W^6 coefficients are needed whereas in the last stage all the coefficients from W^0 to W^7 are required for the butterfly operations. In this example, the coefficient memory has eight locations for storing coefficients (W^0 to W^7) and hence the lower counter section should comprise of only three bits C_2, C_1 and C_0. The more significant counter section, which tracks the four stages of the 16-point FFT, will be having only two bits namely C_4 and C_3. It is clear from Figure 4.2 that in the first stage the coefficient address remains equal to '000'(only W^0) irrespective of the lower section counter value. The first stage is represented by the '00' combination of the higher section bits and hence the input channel of the multiplexer corresponding to this bit combination must always remains at '000'. In the second stage, the address '000'(W^0) is required for the first four computed butterflies and address '100'(W^4) is required for the next four butterflies. The second stage '01' of the FFT is dependent on the status of the C_2 bit of the counter. This means that the second channel input should be set to '$C_2$00'. The third stage '10' of the FFT requires the generation of four different addresses '000'(W^0), '010'(W^2), '100'(W^4) and '110'(W^6). It is clear that these addresses differ only in the C_2C_1 combination and C_0 remains equal to zero always. It means that the third channel of the multiplexer should be connected to '$C_2C_1$0' to accomplish this task. The last stage '11' needs all the coefficients and hence the last channel must be connected directly to '$C_2C_1C_0$'. The same technique can be very easily extended to any FFT size by following the formulations described earlier. This work is reported in [86, 87]

4.3.2 Results

The new coefficient addressing scheme (*Our*) has been synthesized for different FFT lengths from the RTL level Verilog description using *Cadence BuildGates* with 0.35μ Alcatel MTC 45000 CMOS technology library. The power based gate level simulations were carried out at a clock frequency of 100MHz and at a supply voltage of 3.3V for one million clock cycles using *Synopsys DesignPower*. The same procedure was followed for the *Cohen's* scheme and the comparative results in terms of power and area for the different FFT lengths are given in

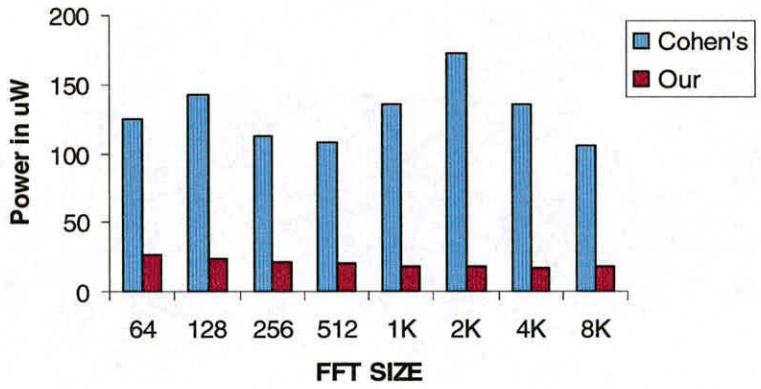


Figure 4.6: *Comparison of the two schemes in terms of power.*

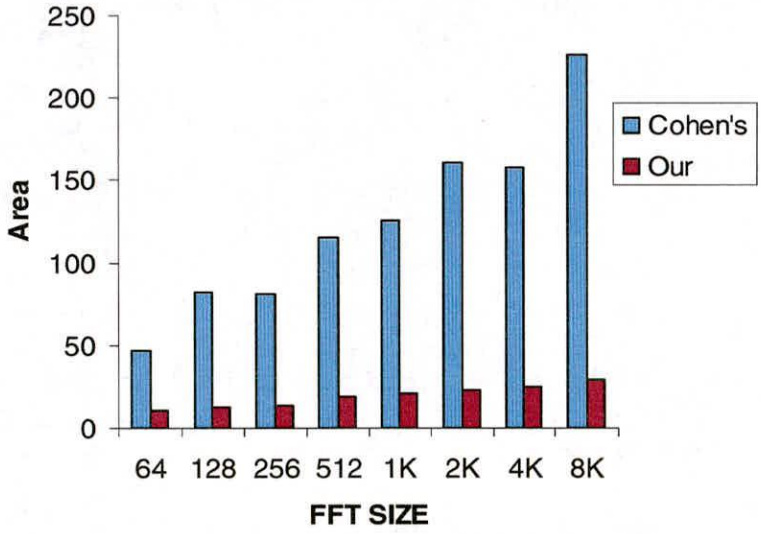


Figure 4.7: *Comparison of the two schemes in terms of area. The area is expressed in equivalent nand gates.*

Figure 4.6 and Figure 4.7 respectively. The power consumption, given in Figure 4.6, is the average power consumed by the respective circuit per clock cycle. It is evident from Figure 4.6 that the power consumption of the coefficient address generation logic depends on the switching activity of the inputs and outputs and not on the FFT size. In Figure 4.7, the area is slightly smaller for the coefficient address generation logic for a 2K-point FFT as compared to 1K-point in a *Cohen* scheme on account of the optimisation of the subtractor with one fixed input by the synthesis tool. It is clear from Figure 4.8 that *Our* scheme results in power and area savings in the range of 80% to 90% as compared to the *Cohen* scheme for almost all FFT lengths.

Chapter 5
Low power single butterfly FFT processor architecture

The basic radix-2 FFT processor can be realised with the help of a single butterfly architecture for low throughput chip area limited applications. This chapter presents a low power radix-2 single butterfly FFT processor architecture which is based on the proposed order based processing scheme described in the previous chapter.

This chapter is organised into two main sections. The first section starts with the introduction to a conventional radix-2 single butterfly FFT processor architecture proposed by Cohen [15]. This single butterfly architecture is chosen because its control logic for coefficient and data address generation is very simple [14, 83]. The second section describes the proposed low power radix-2 single butterfly FFT processor architecture. The conventional radix-2 single butterfly FFT processor architecture is modified to support the order based processing in the proposed low power radix-2 single butterfly FFT processor architecture.

5.1 Coventional radix-2 single butterfly FFT processor architecture

The basic operations of a 16-point radix-2 FFT processor is given by its signal flowgraph shown in Figure 5.1. The complex inputs (X) to the FFT are in bit-reversed order but its complex outputs (*FFTO*) are in normal order. The signal flowgraph for a 16-point FFT is divided into four stages. The first stage is represented by i=0 and the last stage by i=3. The butterfly operation is the most important operation in the FFT. Let $< s,t >$ represents the butterfly operation, then the inputs and outputs of the butterfly are related as follows:

$$XO(s) = X(s) + W * X(t) \tag{5.1}$$

$$XO(t) = X(s) + W * X(t) \tag{5.2}$$

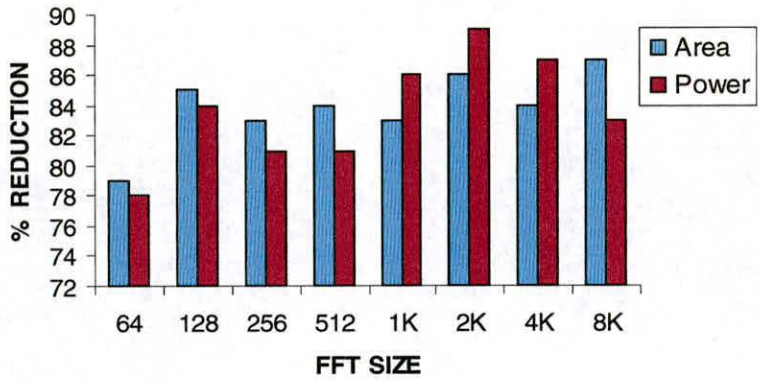


Figure 4.8: *Percentage reduction in power and area of Our scheme over Cohen scheme.*

4.4 Summary

This chapter described three schemes namely the order based processing scheme, the coefficient memory reduction scheme and the coefficient addressing scheme for reducing the power consumption in an FFT processor. The order based processing scheme reduced the switching activity by more than 50% as compared to only around 20% using conventional order based processing. This significant reduction in switching activity will lead to considerable power savings in both single butterfly and pipelined FFT processors described in chapters 5 and 6 respectively. The single butterfly and the pipelined FFT processors described in the subsequent chapters are based on this order based processing scheme.

The remaining two schemes are based on hardware size reduction of an FFT processor resulting in power saving. The coefficient memory reduction scheme required only $((N/8) + 1)$ memory locations instead of $N/4$ for storing the coefficient set of an N-point FFT processor. This scheme has resulted in both power and area savings for long length (more than 1K-points) FFT processors used mainly in OFDM applications. The coefficient addressing scheme implemented the coefficient address generation logic with the help of a single multiplexer instead of a cascade of Barrel shifters. This led to around 80% power and area savings with respect to the *Cohen* scheme in FFT processors of all lengths. The next chapter describes a low power architecture of a single butterfly radix-2 FFT processor to demonstrate the effectiveness of the order based processing scheme.

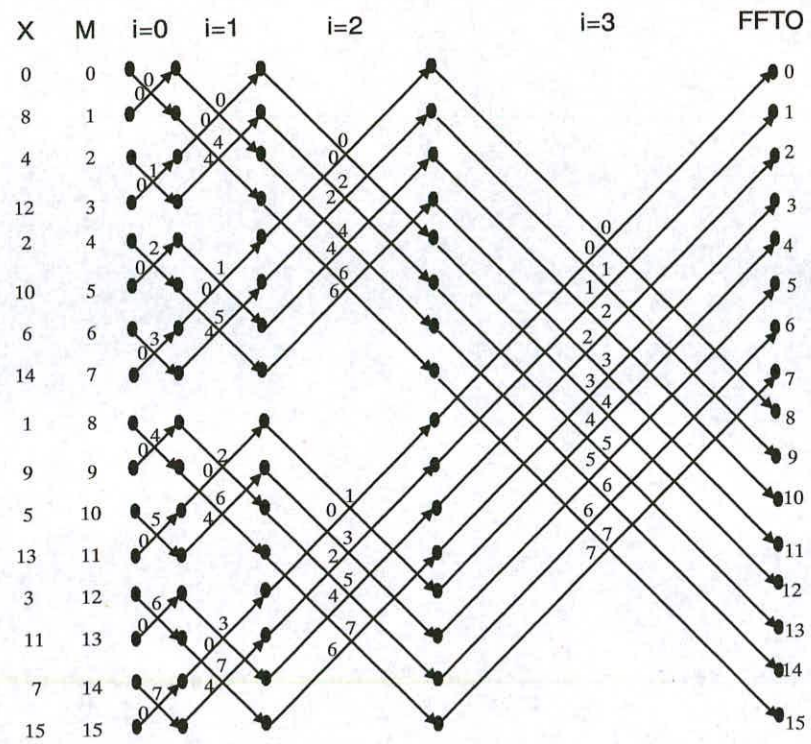


Figure 5.1: *Signal flow graph of a 16-point FFT. 'M' indicates the memory location.*

A butterfly accepts two complex inputs $X(s)$ and $X(t)$ and generates two complex outputs $XO(s)$ and $XO(t)$. The FFT coefficients are represented by W. The butterfly operation in the signal flowgraph can be identified by a number (j) above a line intersection in every stage of the signal flowgraph. There are eight butterflies in every stage and hence 'j' varies from 0 to 7. The number on the left hand side of the intersection indicates the coefficient value used for that butterfly (0 means W^0 and so on).

5.1.1 Memory organisation in a single butterfly FFT processor

A single butterfly 16-point FFT processor architecture comprises of a butterfly unit to realise equations 5.1 and 5.2, sixteen memory locations to store the inputs and intermediate outputs after every stage of a 16-point FFT, and associated address generation logic to feed the proper data and coefficient inputs to the butterfly. All the butterflies for the first stage must be computed first before the computation of the second stage butterflies. All the inputs in the memory are replaced by their corresponding intermediate outputs after the first stage. This procedure has to be repeated for the subsequent stages till the final output is obtained. The detailed sequence

of butterfly execution from the signal flowgraph shown in Figure 5.1 according to the butterfly numbering (j) is as follows:

Stage1 : $< 0,1 >< 2,3 >< 4,5 >< 6,7 >< 8,9 >< 10,11 >< 12,13 >< 14,15 >$

Stage2 : $< 0,2 >< 1,3 >< 4,6 >< 5,7 >< 8,10 >< 9,11 >< 12,14 >< 13,15 >$

Stage3 : $< 0,4 >< 1,5 >< 2,6 >< 3,7 >< 8,12 >< 9,13 >< 10,14 >< 11,15 >$

Stage4 : $< 0,8 >< 1,9 >< 2,10 >< 3,11 >< 4,12 >< 5,13 >< 6,14 >< 7,15 >$

The actual order followed for low power to reduce the switching activity of the coefficient during the execution of successive butterflies in stages 2 and 3 is as follows:

Stage1 : $< 0,1 >< 2,3 >< 4,5 >< 6,7 >< 8,9 >< 10,11 >< 12,13 >< 14,15 >$

Stage2 : $< 0,2 >< 4,6 >< 8,10 >< 12,14 >< 1,3 >< 5,7 >< 9,11 >< 13,15 >$

Stage3 : $< 0,4 >< 8,12 >< 1,5 >< 9,13 >< 2,6 >< 10,14 >< 3,7 >< 11,15 >$

Stage4 : $< 0,8 >< 1,9 >< 2,10 >< 3,11 >< 4,12 >< 5,13 >< 6,14 >< 7,15 >$

One more reason for the modified order of butterfly execution is that the j^{th} butterfly in the i^{th} iteration is $< s,t >$ where:

$$s = ROTATE_n(2j, i) \qquad n = log_2 N \tag{5.3}$$

$$t = ROTATE_n(2j+1, i) \qquad i = 0,1,...,(n-1), j = 0,1,2,...,(N/2-1) \tag{5.4}$$

Where $ROTATE_n(X, m)$ is the value of X rotated left by 'm' bits within 'n' bits, for instance $ROTATE_4(13,3) = 14$.

The author in [88] demonstrated that for every butterfly the two indices (s and t) differ in their parity. The parity of X is defined as zero if the number of 1's in its binary representation is even and one otherwise. This observation can be exploited in organising the N-point memory needed into two banks according to the parity of the addresses. During any clock cycle, the two points $X(s)$ and $X(t)$ are accessed in parallel from the two memory banks because these points are always stored in different banks and also because 's' and 't' differ in parity. The parallel memory block organisation is shown in Figure 5.2. The memory is partitioned into two

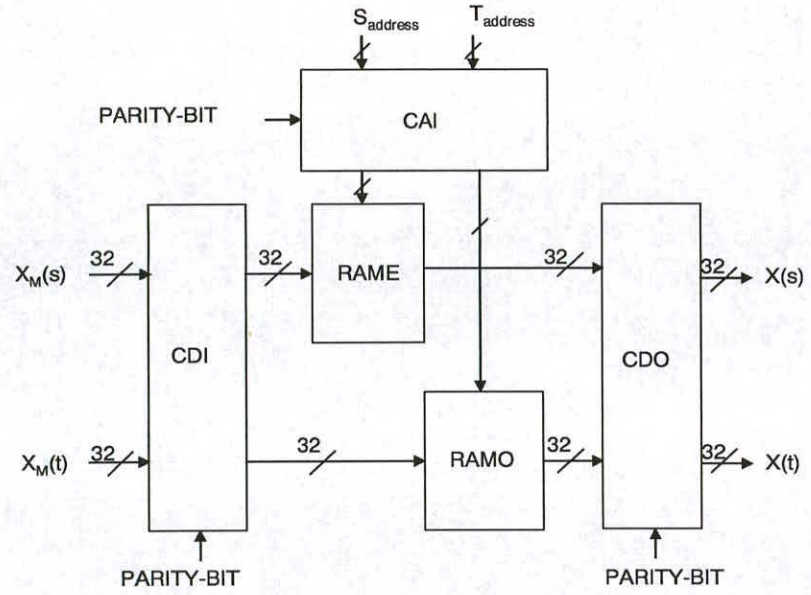


Figure 5.2: *Parallel memory block organisation on the basis of parity bit.*

banks namely *RAME* and *RAMO*. Dual port RAMs are used in the design to read the data for each butterfly operation and to write back the butterfly outputs to the same memory location in the same clock cycle. The data interchange blocks *CDI* and *CDO* and the address interchange block *CDA* are controlled by the *PARITY-BIT*. These blocks direct data and the address to the appropriate memory bank depending upon the parity. The (*PARITY-BIT*) is generated by the *PARITY* block. The *CDI* block receives data from the the input multiplexer block *MUXIN*.

5.1.2 Architecture of the FFT processor

The architecture of the conventional single butterfly radix-2 N-point FFT processor is shown in Figure 5.3. The in-place strategy (Inputs and intermediate outputs are stored in the same memory location after every stage) is employed to reduce the memory size for an N-point transform to only 'N' locations. The conventional radix-2 FFT processor consists of a parallel memory block having N memory locations which are organised in the form of two memory banks for parallel access, a butterfly (*BUTTERFLY*) block to implement basic FFT operations, an input multiplexer *MUXIN* to control the loading of memory, a *FSM* for proper sequencing of operation, a ROM (*CROM*) for storing the coefficients and the address generation block for generating the addresses of coefficient and data.

The data address generation logic is based on two rotation blocks namely *ROT0* and *ROT1* to

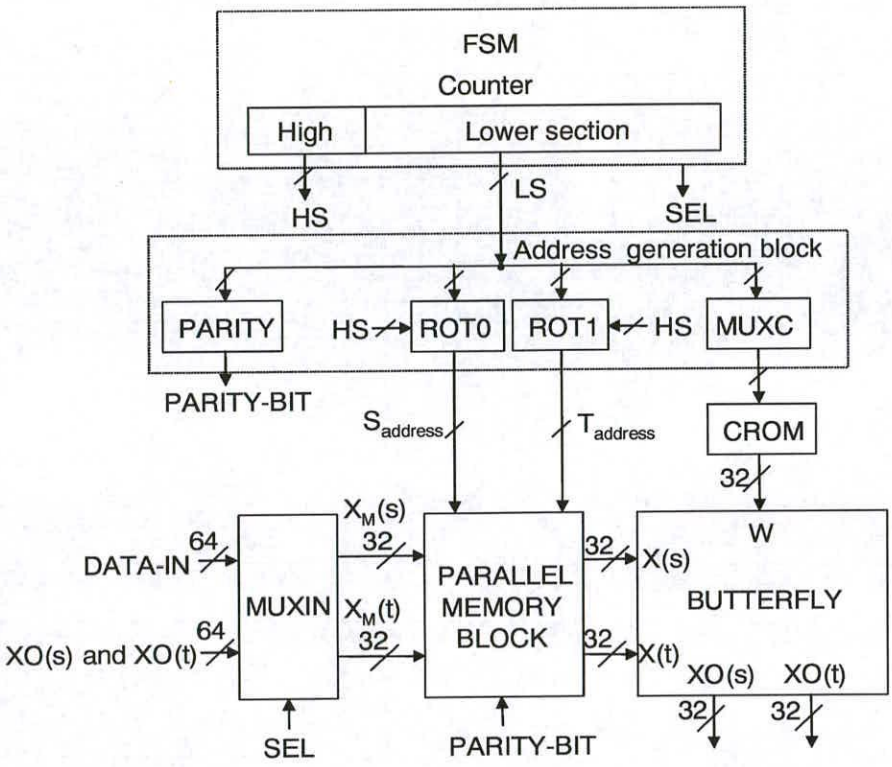


Figure 5.3: *Conventional single butterfly radix-2 FFT processor architecture.*

realise equations 5.3 and 5.4. A least significant bit having value '0' is inserted to the input of ROT0 module and a '1' is inserted to the input of ROT1 to satisfy the shifting and bit insertion requirements of equations 5.3 and 5.4 respectively prior to rotation operation.

The parity generation block (*PARITY*) makes its decision on the basis of the inputs to the rotation block *ROT0* rather than on its output. This is made possible because the parity remains the same after the rotate operation. This approach avoids the unnecessary delay in parity generation after rotation. The coefficient address generation logic is based on the proposed single multiplexer (*MUXC*) implementation for all the FFT stages as described in the previous chapter. The fixed FFT coefficients are stored in the ROM (*CROM*).

The overall operation of the FFT is controlled by the finite state machine (*FSM*). The *FSM* comprises of a counter which is partitioned into two sections. The higher counter section *HS*, denoted by 'i', keeps track of the various FFT stages whereas the lower counter section *LS*, denoted by 'j', takes care of the number of butterflies within every stage. The *FSM* operates in two phases. In the first or input loading phase, the *FSM* loads bit-reversed input data *DATA-IN* into the memory banks as well as output data values corresponding to the previous input.

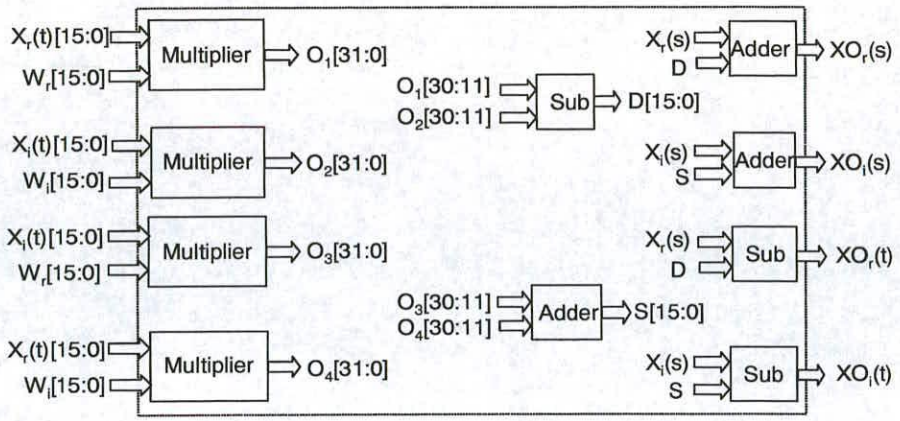


Figure 5.4: *Conventional butterfly architecture.*

Two data values are stored in the memory (one in each memory bank) in every clock cycle. A multiplexer (*MUXIN*) is needed to select the loading of external input (*DATA-IN*) or butterfly output (*XO(s), XO(t)*) to the memory. The select line of *MUXIN* selects *DATA-IN* in the input loading phase. The select line *SEL* of *MUXIN* is again controlled by the *FSM*. In the second or FFT execution phase, the *FSM* clears the counter and then initiates the FFT computation by incrementing it. *MUXIN* selects the butterfly output in this phase. These intermediate butterfly outputs are loaded into the memory banks. The FFT execution phase ends after the execution of all the butterflies in all the FFT stages. For instance, the execution phase in a 16-point FFT processor ends after 32 butterfly operations which corresponds to 32 clock cycles. The overall counter value has to be monitored in the two phases to keep track of their completion. The counter should be cleared after the execution of the FFT execution phase. The two phases are to be executed in a cyclic fashion till all the FFT blocks are processed.

The block diagram of the conventional butterfly is shown in Figure 5.4. A butterfly of a radix-2 decimation-in-time FFT has two complex inputs namely $X(s)$ $(X_r(s) + jX_i(s))$ and $X(t)$ $(X_r(t) + jX_i(t))$ and two complex outputs $XO(s)$ $(XO_r(s) + jXO_i(s))$ and $XO(t)$ $(XO_r(t) + jXO_i(t))$. The complex FFT coefficients are represented by W $(W_r + \mathrm{j}W_i)$. All the inputs are of 16-bits. The butterfly outputs and inputs are related by the following equations:

$$XO_r(s) = X_r(s) + (X_r(t)W_r - X_i(t)W_i)$$

$$XO_i(s) = X_i(s) + (X_i(t)W_r + X_r(t)W_i)$$

$$XO_r(t) = X_r(s) - (X_r(t)W_r - X_i(t)W_i)$$

$$XO_i(t) = X_i(s) - (X_i(t)W_r + X_r(t)W_i)$$

It is important to note that only 20-bits of the multiplier output are applied to the second stage adder and subtractor instead of 32-bits to limit the hardware size. Since the outputs are of 16-bits, it has been found that 20-bits of intermediate accuracy is a judicious choice between the noise tolerated and the hardware size [64].

5.2 Low power radix-2 FFT processor architecture

The conventional FFT processor architecture, proposed by Cohen [15], has been modified to support order based processing scheme. The order based processing scheme has been discussed in detail in the previous chapter. The FFT processor based on the order based processing scheme is called ordered radix-2 FFT processor. An ordered radix-2 FFT processor core, shown in Figure 5.5, comprises of the following components:

- Ordered butterfly module.
- Two dual port memories for holding data in each FFT stage.
- ROM for holding the fixed coefficients.
- Address generation logic for generating address for both the coefficient and data.
- Control logic in the form of a finite state machine (FSM).
- Ordering logic in the form of Look up table (LUT) and a multiplexer to support order based processing.

An ordered butterfly module accepts complex data at every clock cycle in order to produce complex outputs during the same cycle. Each memory location is 32-bit wide for storing both the 16-bit real and imaginary parts of data. The memory is organised into two banks based on parity of the address bits in order to generate the addresses of data required at successive FFT stages. The inputs are read from the RAMs (Both *RAME* and *RAMO*) into the butterfly in the same cycle as the outputs are written back to the RAMs for minimising the number of clock cycles [15].

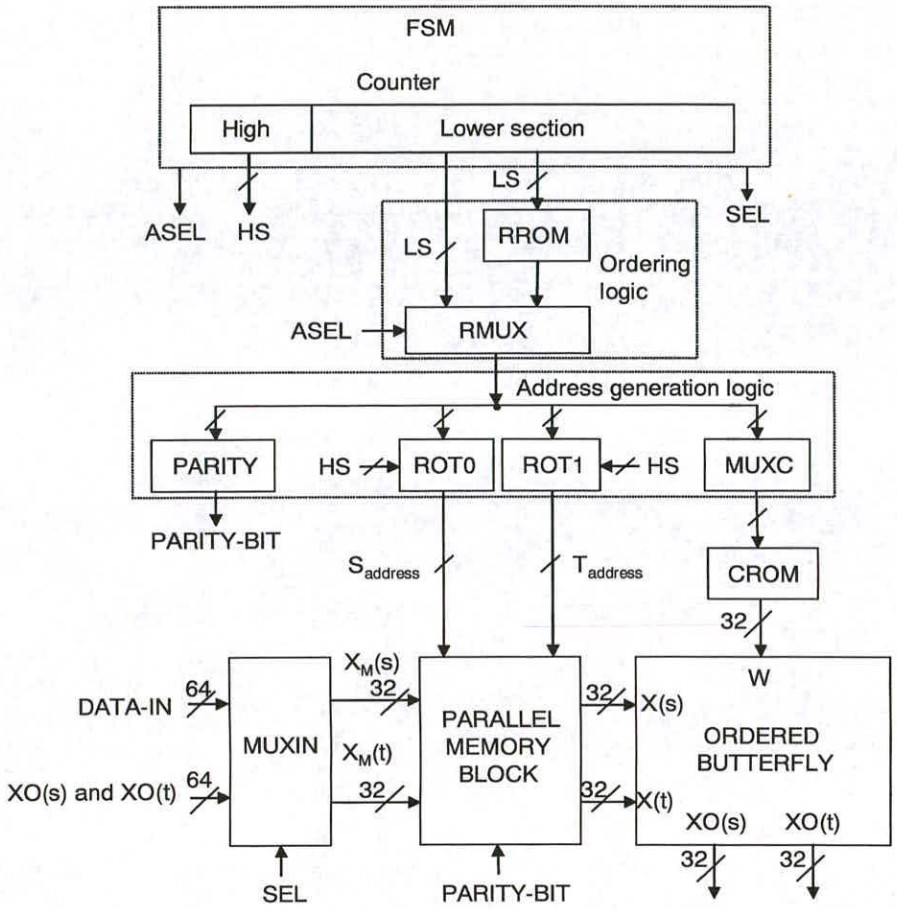


Figure 5.5: *Low power single butterfly ordered radix-2 FFT processor architecture.*

The order based processing block (*Ordering logic*) consists of an LUT (*RROM*) and a multiplexer (*RMUX*). *RROM* stores the addresses of the ordered coefficient set. These addresses will be active only in the last stage of the FFT when order based processing has to be incorporated. *RMUX* is used to select between the ordered addresses for the coefficient and data and the conventional addresses. The ordered addresses are used only in the last stage whereas the conventional addresses are used in all the previous FFT stages. The select signal (*ASEL*) for *RMUX* can be generated by the *FSM* using the bits of the more significant counter section. The order based processing scheme is most effective in the last stage of the signal flowgraph of the radix-2 decimation-in-time FFT algorithm where the coefficient switching activity is most intense (The coefficients are different for each butterfly unlike other FFT stages as shown in Figure 5.1). The investigations revealed that the power saving is maximum if the order based processing scheme is limited only to the last FFT stage. The hardware overhead to support order based processing in the penultimate stages outweigh the power saving obtained in these

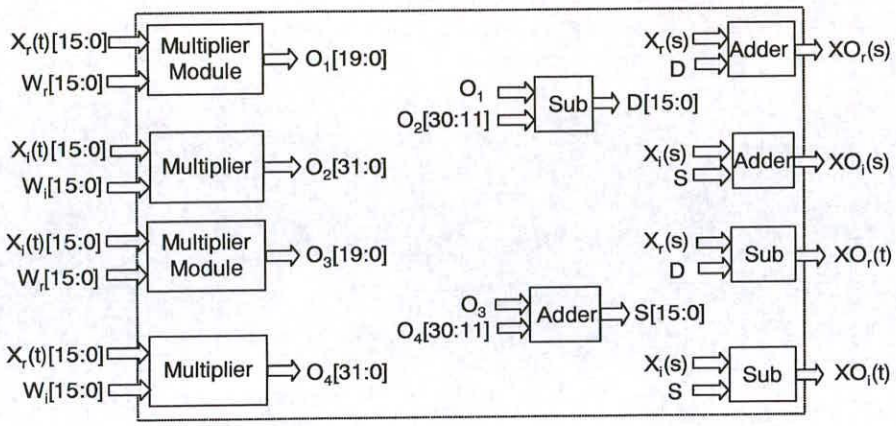


Figure 5.6: *Low power butterfly ordered architecture.*

stages. This work is reported in [81].

The hardware overhead required to support the order based processing scheme is in the form of an additional ROM (*RROM*) having N/2 locations with word width equal to $log_2(N/2)$, a 2:1 channel multiplexer (*RMUX*) and an array of 20 Ex-OR gates in the modified multiplier module of the ordered butterfly for the real coefficients only. It is clear that only the ROM size and the word width to a lesser extent increase with the FFT size. The ROM block consumes much less power and hence introduces only little overhead as compared to the conventional approach. The ROM and the multiplexer blocks are also required in the *Others* ordering approaches and hence the hardware overhead of *Our* approach with respect to the *Others* are only the insignificant Ex-OR gates within the multiplier module. The ordered butterfly module for the low power radix-2 FFT processor architecture is discussed in the next section.

5.2.1 Ordered butterfly module

The block diagram of the ordered butterfly hardware for the low power architecture is shown in Figure 5.6. The ordered butterfly needs a multiplication module instead of a two's complement multiplier for the real part of the coefficient to implement the order based processing scheme. The low power order based processing scheme requires selective complementation. The correct output of the multiplier is obtained by selectively complementing the multiplier output corresponding to the real coefficients. A dedicated multiplication module has to be used for the real part of the coefficient (W_r) instead of a two's complement multiplier to support *Our* order based processing scheme. This will be discussed in the next section.

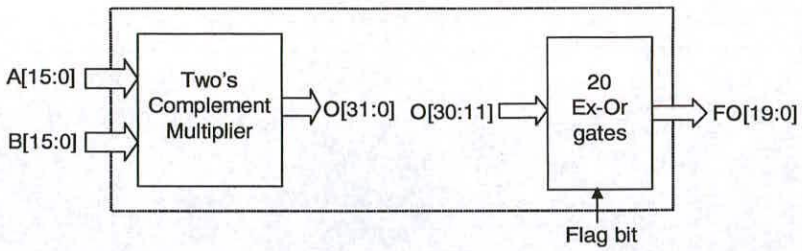


Figure 5.7: *Architecture of the multiplication module for the low power butterfly.*

5.2.2 Multiplication module

A simple two's complement multiplier cannot be used for the real coefficients because the real coefficients are used in two's complement form as well. A flag bit is also stored along with each real coefficient to indicate the form of the real coefficient. The multiplication module corresponding to the real coefficients is obtained by adding 20 Control inverters (Ex-Ors) to the output of a two's complement multiplier shown in Figure 5.7. The flag bit controls the Ex-Ors and therefore the final output of the multiplication module (*FO*). *FO* is the complement of the 16-bit multiplier output in case the flag bit is '1' otherwise it will be the same as the multiplier output. The multiplier output is limited to only 20-bits instead of 32-bits. There is a difference of unity at the LSB position between *FO* and the actual multiplier's 20-bit output in the conventional approach. This difference arises only in rare cases where all the lower 11 bits of the actual multiplier's 32-bit output are zero and when the real coefficient is represented in its two's complement form. This is because only the upper 20-bit output of the multiplier is complemented instead of two's complementing the whole 32-bit output. This is performed to avoid the use of a 32-bit adder to generate the two's complement of the multiplier output, thereby, saving power. Moreover, it does not lead to any error because the 16-bit output of the butterfly at each FFT stage is halved to avoid overflow. This approach has been verified extensively by comparing the outputs of different length FFT's with random data.

5.3 Results

A number of FFT cores of varying sizes have been designed at the register transfer level (RTL) using Verilog hardware description language. The cores were synthesised using Synopsys *DesignCompiler* with 0.35μ Alcatel MTC45000 CMOS technology library. In order to evaluate the performance of synthesised cores, gate level netlist simulations were performed using Ca-

FFT size	*Conv FFT mW*	*Others FFT mW*	*Our FFT mW*	*% reduction /Others*
16	67.60	68.44	51.49	25
32	84.55	86.82	75.68	13
64	104.93	105.88	95.72	10
128	213.28	210.34	198.89	7
256	355.27	354.74	343.80	3
512	864.35	863.80	855.21	1

Table 5.1: *Power consumption comparison of FFT cores.*

dence *Verilog-XL* simulator for 1000 FFT blocks of uniformly distributed random input data samples. The switching activity information obtained from the gate level netlist simulations was then fed into Synopsys *DesignPower* for power analysis. The power analysis was performed at a clock frequency of 10MHz and a supply voltage of 3.3V. The same procedure is adopted for the *Conv* and *Others* cores for a fair comparison. Table 5.1 depicts the power consumption comparison for the different FFT cores obtained by following the conventional *Conv*, *Others* and *Our* approaches for the Non-Booth-coded Wallace tree type multiplier. It is evident from the table that the power saving of *Our* scheme ranges from 1% to 25% for 512-point to 16-point FFT processors respectively over the *Others* approaches. The percentage power saving continues to reduce for longer length FFT's because *Our* scheme is directed at reducing power by lowering the switching activity at the coefficient inputs of the multipliers in the butterfly structure. The butterfly complexity in an FFT remains fixed with FFT length whereas the RAM size goes on increasing. This means that the power consumed in the butterfly increases slightly with FFT length as compared to the power consumed in the RAM blocks. This results in lowering of the percentage savings in power for longer FFT's. The *Others* scheme leads to no power savings in most cases because the switching activity reduction is much less as compared to the hardware overhead required to support the scheme. This is not true for *Our* scheme due to the significant reduction in the switching activity of the ordered coefficient set. Table 5.2 lists the power consumed by the various blocks of the 64-point FFT processor core. It can be concluded from the table that the butterfly and the RAM contributes most to the power consumption. The ROM and other modules consume less power. *Our* scheme leads to power savings both in the internal cells as well as on the nets. It has been found that the relative performance of *Our* approach remains the same for different data sets.

The area overhead of *Our* 64-point single butterfly FFT is 0.3% and 0.01% with respect to *Conv*

FFT core cells	*Conv FFT mW*	*Others FFT mW*	*Our FFT mW*
BUTTERFLY	22.84	22.82	18.25
RAME	7.77	7.82	7.84
RAMO	7.77	7.80	7.82
MUXIN	1.22	1.25	1.20
CDI	0.40	0.41	0.41
CDO	0.38	0.38	0.38
FSM	0.05	0.05	0.05
RROM	-	0.05	0.05
CROM	0.03	0.04	0.03
ROT1	0.01	0.01	0.01
ROT0	0.01	0.01	0.01
CAI	0.02	0.02	0.02
MUXC	0.002	0.002	0.002
RMUX	-	0.007	0.007
PARITY	0.008	0.009	0.009
Internal cell power(IP)	40.51	40.68	36.08
Net switching power(NP)	64.42	65.20	59.64
Total FFT power(TP)=NP+IP	104.93	105.88	95.72

Table 5.2: *Power consumption of the different cells along with net switching power of a 64-point FFT core.*

and *Others* FFTs respectively. This means that the area overhead is negligible as compared to the power saving.

5.4 Summary

This chapter described a low power radix-2 FFT processor architecture. The FFT processor is based on the order based processing scheme. The power saving is higher for short FFTs because the order based processing scheme is directed at saving power in the butterfly and not in the memory. The memory size goes on increasing with the FFT size and therefore the power share of the butterfly module goes on reducing with FFT size. The order based processing scheme can also be successfully applied to a pipelined FFT processor. The next chapter introduces a low power radix-4 pipelined FFT processor architecture which is also based on the order based processing scheme.

Chapter 6
Low power radix-4 pipelined FFT processor architecture

The pipelined FFT processors are commonly used in all multicarrier applications requiring real time processing. The multicarrier receiver needs a low power pipelined FFT processor. A low power radix-4 pipelined FFT processor architecture is chosen as a conventional pipelined FFT processor [13]. This radix-4 pipelined architecture is considered because its butterfly can be implemented by using only one complex multiplier just like radix-2 FFT processors with all the other attributes of the radix-4 FFT processor for sequential input processing. A novel low power radix-4 ordered pipelined FFT processor architecture is proposed by incorporating order based processing scheme into the conventional radix-4 pipelined FFT processor.

This chapter is organised into four sections. The first section describes the need of pipelined FFT processor. The second section explains the algorithm and the architecture of the conventional radix-4 pipelined FFT processor. The third section proposes the ordered radix-4 pipelined FFT processor architecture by altering the penultimate stage of the conventional radix-4 FFT processor to support the order based processing scheme. The chapter concludes with a section on the comparison of the ordered and conventional radix-4 FFT processor architectures in terms of power and area.

6.1 Need of pipelined FFT processor architectures

The single butterfly architectures suffer from speed limitations for long FFTs. The N-point single butterfly FFT processor needs 'N' complex words of memory for in-place algorithm. If this memory is not partitioned unlike the previous chapter, the number of read/write accesses to perform the FFT creates a bottleneck: an N-point FFT requires $(N/r)log_rN$ butterfly computations. This implies $2Nlog_rN$ read/write accesses. It means that the read/write access to the internal RAM every 4.7ns to perform an 8K-point FFT in 1ms using a radix-2 approach. This

value is difficult to achieve. The high frequency of operation considerably increases the power consumption as well.

The said problem can be solved by either using a higher radix to reduce the number of butterflies or to partition the memory into r banks as in the previous chapter. The higher radix approach leads to complex butterflies and the memory partitioning requires complex addressing and higher area. Moreover, in all single butterfly architectures, a clock having frequency much higher than the input data sampling rate is required for all multicarrier applications. The presence of multiple clocks complicates the multicarrier receiver design.

The real advantage of the pipelined architecture is that all the hardware blocks operate at the input data sampling rate. The pipelined hardware is obtained by perpendicularly projecting the FFT signal flow graph to the data flow. The hardware consists of several butterfly units (one per FFT stage) with associated complex multipliers, separated by delay commutator. The speed of the pipelined architecture can be very easily enhanced by increasing the level of pipelining in the arithmetic units. The only disadvantage is that the butterfly unit has to be replicated $log_r N$ times compared to the single butterfly implementation.

6.2 Conventional radix-4 pipelined FFT processor architecture

The pipelined FFT processor architectures are most commonly employed in multi-carrier receivers because these architectures can be directly interfaced to input data operating at the sampling rate for real time applications [89–91]. The radix-4 architectures are preferred over radix-2 for low power applications because of the reduced number of multiplications in radix-4 as compared to radix-2 [62]. The conventional radix-4 butterfly comprises of three complex multipliers and requires the availability of all the four butterfly inputs at the same time as explained in chapter 3 [61, 92]. In general, the data from the A/D converter always arrives in word sequential format with one input at a time. This data/processor mismatch can be bridged by providing input buffers which consumes a lot of power and by operating the pipeline at $(1/4)^{th}$ of the input rate. Bi and Jones proposed a radix-4 pipelined architecture in which only one complex multiplier is used in its butterfly [13]. Moreover, this architecture can be directly interfaced to word sequential data input without the need of input buffering. The hardware comparison of Bi and Jones architecture with respect to the traditional Gold and Bially [92] and a digit serial Hui architectures [93] in terms of multipliers, adders and memory is listed in

	Gold and Bially	Bi and Jones	Hui
Memory	3.25N	2.75N	2.5N
Multiplier	$3log_4N$	log_4N	$3log_4N^*$
Adder	$8log_4N$	$3log_4N$	$12log_4N^*$

Table 6.1: *Comparison of the different pipelined FFT architectures. * indicates digit serial.*

Table 6.1. The memory requirement of Hui architecture is slightly less than Bi and Jones but the power consumed in its memory is more because of deep memory fragmentation [91]. Deep memory fragmentation prevents the realisation of FIFOs using dual port RAMs. The traditional shift register based realisation of the fragmented memory increases the switching activity and hence the power consumption of Hui's architecture. The Bi and Jones pipelined architecture is implemented by Bidet as a first single chip 8K point FFT for multi-carrier orthogonal frequency division multiplexing applications [63]. The Bi and Jones architecture is selected because it has the lowest value of FFTs per energy among all the pipelined architectures [53]. Moreover, it is also a very popular architecture for implementing multi-carrier receivers [63, 89, 91]. The conventional radix-4 architecture is based on Bi and Jones architecture. The conventional radix-4 FFT processor architecture is now derived from the Bi and Jones algorithm.

6.2.1 Bi and Jones algorithm for DFT decomposition

The N-point DFT of a finite duration sequence x(n) is defined again here by equation 6.1.

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \qquad k = 0, 1, ..., N-1 \tag{6.1}$$

Where W_N is defined as,

$$W_N = e^{-j2\pi/N} \tag{6.2}$$

Let N be a composite number of v integers so that $N = r_1 r_2 ... r_v$, and define,

$$N_t = N/r_1 r_2 ... r_t \qquad 1 \leq t \leq v-1 \tag{6.3}$$

Where 't' is the stage number of the decomposed DFT and r_t its radix. Using the recursive property of equation 6.3 for radix r_1, equation 6.1 becomes,

$$X(k) = \sum_{p=0}^{r_1-1} x(N_1 p) W_N^{N_1 pk} + \sum_{p=0}^{r_1-1} x(N_1 p + 1) W_N^{(N_1 p+1)k} + \tag{6.4}$$

$$\ldots + \sum_{p=0}^{r_1-1} x(N_1 p + N_1 - 1) W_N^{(N_1 p+N_1-1)k}$$

Equation 6.4 can be modified by using the relationship $W_{N_i N_j}^{N_j k} = W_{N_i}^{k}$ as follows:

$$X(k) = \sum_{q_1=0}^{N_1-1} W_N^{q_1 k} \sum_{p=0}^{r_1-1} x(N_1 p + q_1) W_{r_1}^{pk} \tag{6.5}$$

Now defining indexes k_1 and m_1 by $k = r_1 k_1 + m_1$ where $0 \leq k_1 \leq N_1 - 1$ and $0 \leq m_1 \leq r_1 - 1$, equation 6.5 becomes,

$$X(r_1 k_1 + m_1) = \sum_{q_1=0}^{N_1-1} x_1(q_1, m_1) W_{N_1}^{q_1 k_1} \tag{6.6}$$

Where,

$$x_1(q_1, m_1) = W_N^{q_1 m_1} \sum_{p=0}^{r_1-1} x(N_1 p + q_1) W_{r_1}^{p m_1} \tag{6.7}$$

Equation 6.7 defines the computation for the first stage. Continuing the decomposition process of equation 6.6 for radix numbers other than r_1, the complete N-point DFT can be decomposed into v-1 further stages of computation. The final stage is defined by equation 6.8 as follows [13]:

$$X(r_1 r_2 \ldots r_{v-1} m_v + \ldots + r_1 m_2 + m_1) = \sum_{q_{v-1}=0}^{r_v-1} x_{v-1}(q_{v-1}, m_{v-1}) W_{r_v}^{q_{v-1} m_v} \tag{6.8}$$

Whereas the intermediate stages 't' are given by the following equation.

$$x_t(q_t, m_t) = W_{N_{t-1}}^{q_t m_t} \sum_{p=0}^{r_t-1} x_{t-1}(N_t p + q_t m_{t-1}) W_{r_t}^{p m_t} \tag{6.9}$$

Where $2 \leq t \leq v - 1, \quad 0 \leq m_i \leq r_i - 1, \quad 0 \leq q_i \leq N_i - 1$ and $\quad 2 \leq i \leq v$

Each summation in these equations represents an r_t point DFT. The coefficient $W_{N_{t-1}}^{q_t m_t}$ is outside the summation, the decomposition thus corresponds to a decimation-in-frequency computation.

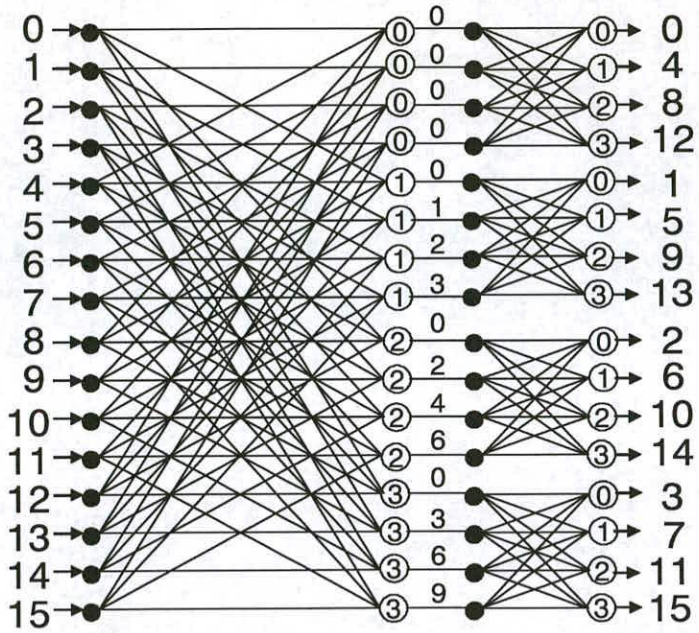


Figure 6.1: *Signal flow graph of a radix-4 16-point FFT.*

Since radix numbers can be any positive integer, equations 6.7, 6.8 and 6.9 can be used for either mixed radix computation or uniform radix computation. For $r_1 = 4$, the flowgraph of a 16-point FFT based on the above formulation is shown in Figure 6.1. The corresponding equations are,

$$X(4m_2 + m_1) = \sum_{q_1=0}^{3} x_1(q_1, m_1) W_4^{q_1 m_2} \tag{6.10}$$

Where,

$$x_1(q_1, m_1) = W_{16}^{q_1 m_1} \sum_{p=0}^{3} x(4p + q_1) W_4^{p m_1} \quad 0 \leq m_1, m_2 \leq 3 \tag{6.11}$$

In Figure 6.1, each open circle represents the summation while the dots define the stage boundaries. The number inside the open circle is the value of m_1 (for stage 1) or m_2 (for stage 2). The number outside the open circle is the FFT coefficient applied.

6.2.2 Hardware implementation

A pipelined N-point radix-4 FFT processor based on the previously described algorithm, shown in Figure 6.2, has $log_4 N$ stages. Each stage produces one output within each word cycle. Each stage contains a commutator, a butterfly element (for summation) and a complex multiplier. The sequential outputs at each stage must be ordered in accordance with the value of m_t. For

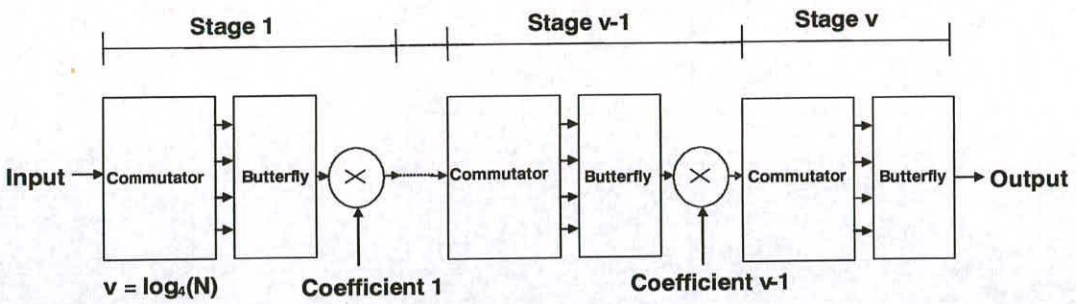


Figure 6.2: *Conventional N-point radix-4 pipelined FFT processor architecture.*

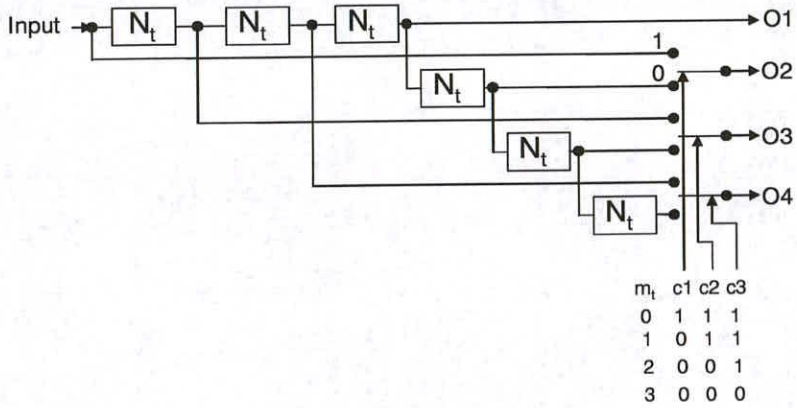


Figure 6.3: *Commutator architecture for the conventional radix-4 pipelined FFT processor architecture.*

instance, from Figure 6.1 at stage 1, the outputs associated with $m_1 = 0$ are produced in the first four word cycles, then those associated with $m_1 = 1$ in the next four cycles and so on. It is clear from equation 6.9 that the input data for each summation at stage 't' are separated in time by N_t words. The required commutator comprises of six shift registers each providing N_t word delay along with three multiplexers and is shown in Figure 6.3. The control signals C_1, C_2 and C_3 select the appropriate data using 2:1 multiplexers according to the value of m_t. The timing diagram of the commutator for stage 1 of a 16-point FFT is shown in Figure 6.4. In Figure 6.4, t' is the instant when the first input word arrives. Each input word occupies a word slot of duration 'T' and is numbered according to its appearance in time. The four complex outputs from the commutator are connected to its associated butterfly. The commutator supplies the same set of data for N_t word cycles.

The FIFO block of the commutator can also be realised by either using a dual port RAM (*DM*) or a Triple port RAM (*TM*) for low power implementation. A *DM* can be converted into a FIFO by maintaining a difference of unity between its read pointer (r) and write pointer (w). Let us assume that the read and the write pointers are incremented after every clock cycle of

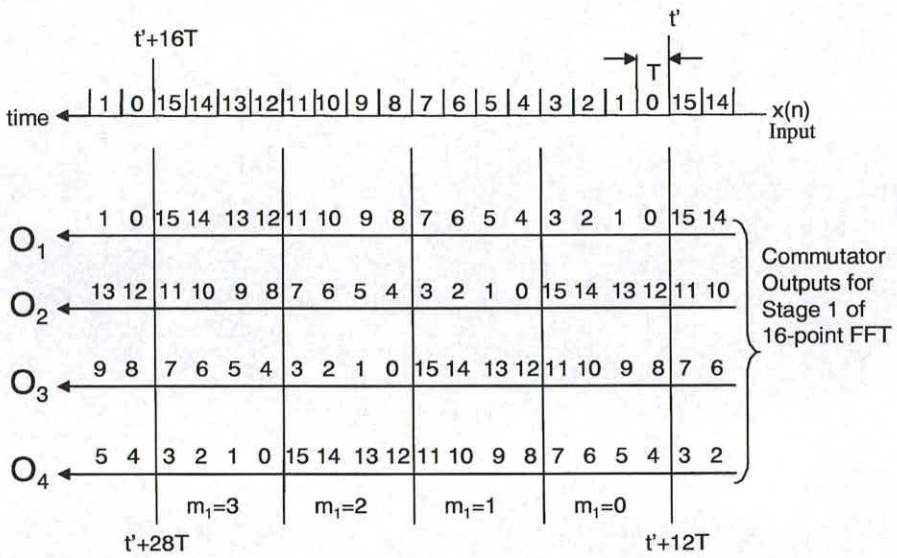


Figure 6.4: *Timing diagram of commutator outputs for the first stage of a 16-point radix-4 pipelined FFT processor.*

duration 'T'. The read and the write pointers in a FIFO are arranged in such a way that the write pointer always writes to the memory location read during the previous clock cycle. Figure 6.5 illustrates the principle of operation of a *DM* based FIFO having length equal to four with synchronous write and aysnchronous read capabilities. The FIFO block receives an input after every clock cycle. It is clear from Figure 6.5 that the output (O) of *DM* is exactly identical to the output obtained from a traditional shift register based FIFO of length equal to four. The advantage of the *DM* based FIFO is that there is no need to move all the stored data values after every new data entry into the FIFO contrary to a shift register based FIFO. This translates into lower switching activity and therefore low power consumption. The block diagram of the commutator with a *DM* based FIFO is shown in Figure 6.6.

The *TM* used here has two asynchronous read ports and one synchronous write port and therefore it is equivalent to a *DM* with one additional read port. This additional read pointer in *TM* based FIFO can be independently positioned to read any stored FIFO values thereby making it ideal for handling complex data ordering. A novel architecture of the commutator based on *TM* based FIFO will be proposed in the low power ordered pipelined radix-4 FFT processor section of this chapter. The design of the butterfly and the complex multiplier are illustrated in the next sections.

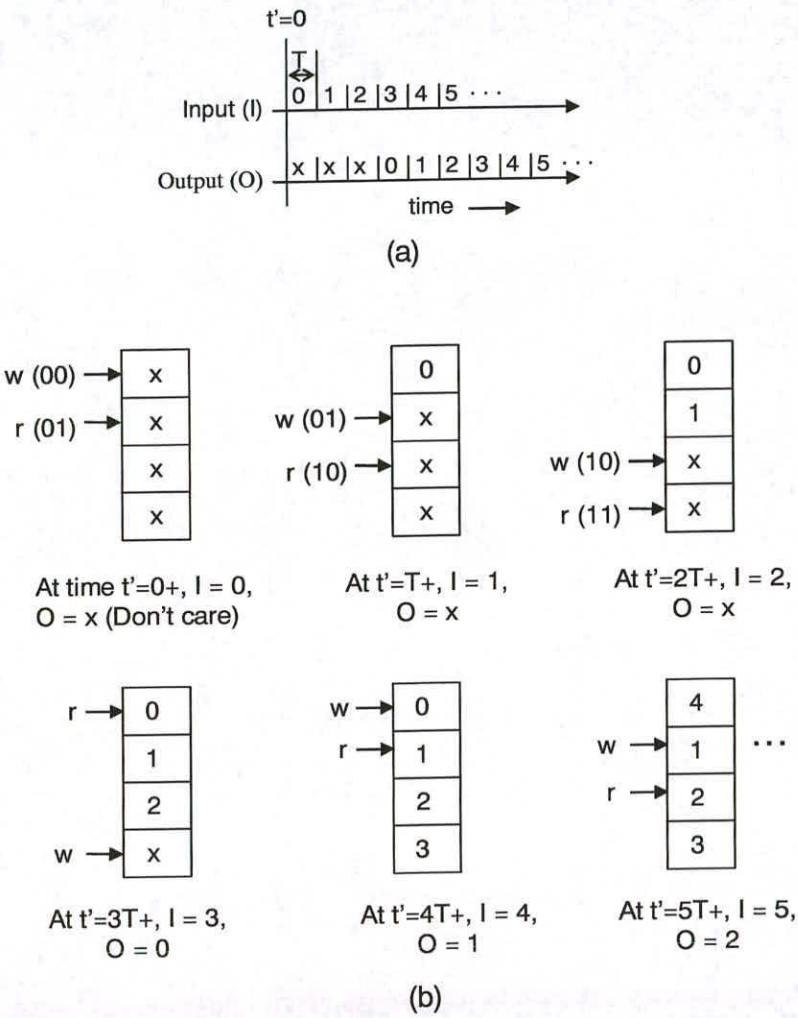


Figure 6.5: *(a) Timing diagram showing the input and output values of a FIFO of length equal to four, (b) The contents of the four location dual port RAM based FIFO, its Input and Output after every clock cycle for six consecutive clock cycles(+ means just after).*

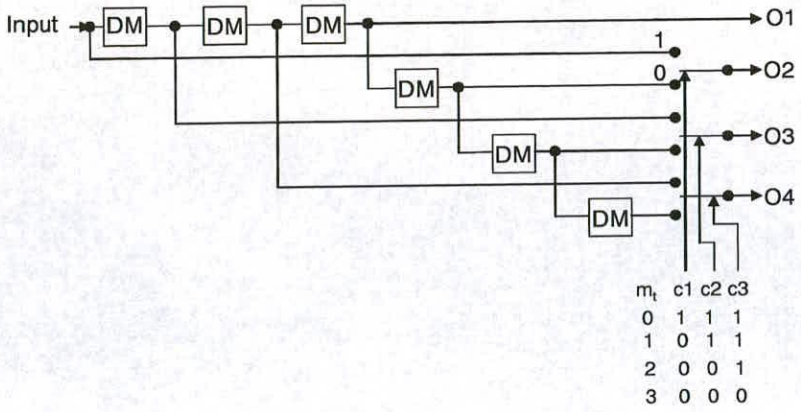


Figure 6.6: *Commutator architecture for the* DM *based FIFO.*

m_t	C_4	C_5	C_6	C_7
0	0	0	0	0
1	1	0	1	1
2	0	1	1	0
3	1	1	0	1

Table 6.2: *Control signals for the different values of* m_t *(0=addition, 1=subtraction).*

6.2.2.1 Butterfly

The radix-4 butterfly unit implements the summations of equations 6.8 and 6.9 and is shown in Figure 6.7. The complex multiplication with $\pm j$ in the summations can be replaced by the combination of addition, subtraction and swapping between the real and imaginary parts for the radix-4 FFT processor. The butterfly architecture is based on three complex adder/subtractors instead of eight complex adders. The control signals C_4, C_5 and C_6 select the appropriate data and the function (addition or subtraction) in accordance with the value of m_t. The values of the control signals for the different values of m_t are listed in Table 6.2.

6.2.2.2 Conventional complex multiplier

A conventional complex multiplier accepts two complex inputs namely data $(X_r + jX_i)$ and coefficient $(W_r + jW_i)$ and produces a complex output $(XO_r + jXO_i)$. It is constructed by using four real multipliers along with an adder and a subtractor. The outputs and inputs of the

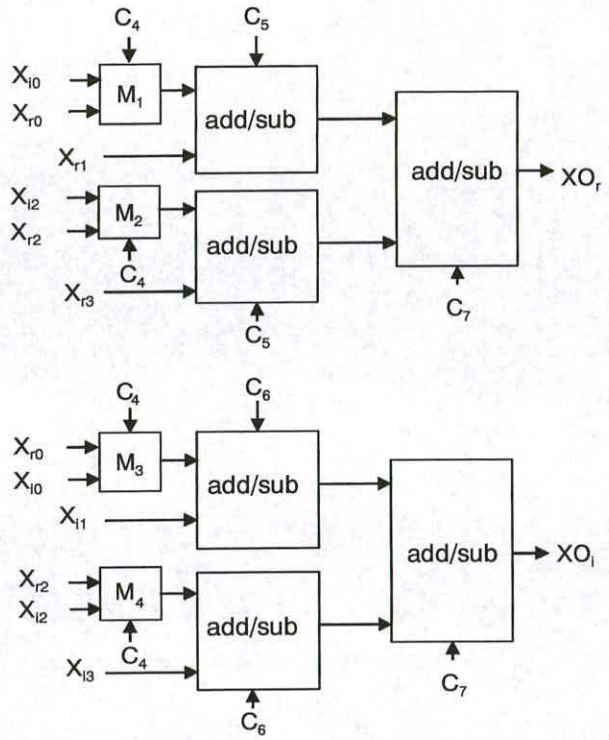


Figure 6.7: *Conventional butterfly architecture.*

complex multiplier are related as follows:

$$XO_r = (X_r W_r - X_i W_i)$$

$$XO_i = (X_i W_r + X_r W_i)$$

The complex multiplier is shown in Figure 6.8. Only 20-bits of the multiplier outputs are used by the adder and subtractor to reduce the hardware without introducing significant errors.

6.3 Low power ordered pipelined radix-4 FFT processor architecture

This section proposes a low power ordered radix-4 FFT processor architecture by modifying the operation sequence of the conventional low power radix-4 pipelined FFT processor architecture. The complex multiplier within the butterfly processing unit is one of the most power consuming block in a pipelined FFT processor. The switching activity between successive coefficients fed to the complex multiplier can be significantly reduced by order based processing

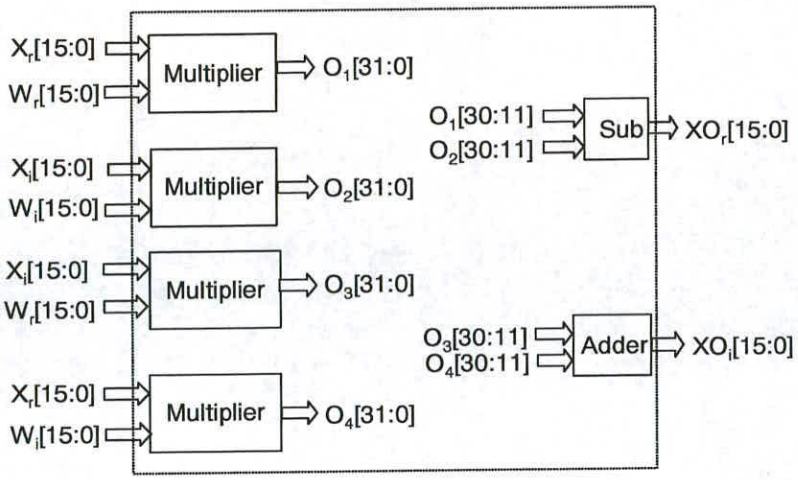


Figure 6.8: *Conventional complex multiplier.*

and hence its power consumption. The order based processing of fixed coefficients requires corresponding data sequencing as per new coefficient ordering in a pipelined FFT processor. Data sequencing is performed by a commutator in a pipelined FFT processors. Hence, a novel commutator architecture is proposed to handle the new data sequencing for stage 1 of a 16-point FFT processor. The data sequencing for stage 2 is restored by using a dual port RAM (*DM*) along with a ROM for its address generation. This ordering technique is suitable only for stage 1 of a 16-point radix-4 FFT processor due to the need of restoring data ordering for the following stage. This in turn requires only small hardware overhead in the form of a six word additional *DM* (*ADM*) following stage 1 of a 16-point FFT contrary to a much larger *ADM* following stage 1 of a 64-point FFT. The large size *ADM* is required because stage 1 of a 64-point FFT handles 64 data samples compared to only 16 for stage 1 of a 16-point FFT. Thus our order based processing scheme is limited only to stage 1 of a 16-point FFT processor or stage 2 of a 64-point FFT processor and so on. Moreover, the commutator design for incorporating order based processing becomes very difficult for the bigger stage 1 of a 64-point FFT processor.

6.3.1 Order based processing of coefficients in a radix-4 pipelined FFT processor

This section proposes an altered operation sequencing for stage 1 of a 16-point radix-4 pipelined FFT processor based on its signal flowgraph shown in Figure 6.9. Normally, the fixed coefficients are fed to the complex multiplier in an order given in Figure 6.2 starting from $m_1 = 0$ and ending with $m_1 = 3$ for stage 1 of a 16-point FFT processor. Our approach involves ordering the coefficient sequence so as to minimise switching activity between successive co-

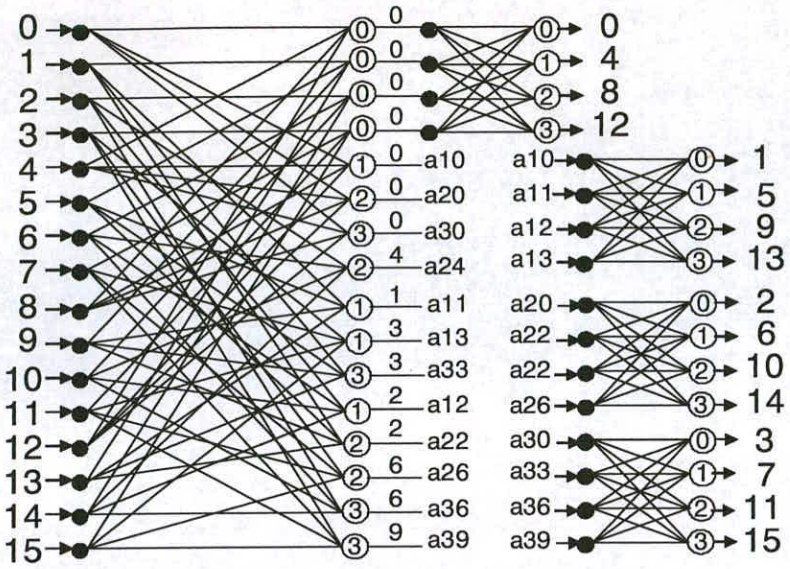


Figure 6.9: *Signal flow graph of the ordered 16-point radix-4 pipelined FFT.*

efficients fed to the multiplier for stage 1 of a 16-point FFT or stage 2 of a 64-point FFT as listed in Table 6.3 and also shown in Figure 6.9. The coefficients are ordered so as to minimise the switching activity between successive coefficients by minimising the Hamming distance between them. The ordered coefficient set is obtained by first arranging only the imaginary part of the coefficient set on the basis of Hamming distance. It is followed by picking up the corresponding real part of the coefficient or its two's complement depending upon the Hamming distance with respect to the previously arranged real part. A flag bit is asserted to indicate the presence of real part in two's complement form. This flag bit is also used to selectively complement the multiplier output as discussed in chapter 4. The switching activity decreases from 192 to just 78, a reduction of 59% by following our ordering approach. The order based processing of coefficients requires corresponding data ordering. The data ordering is performed by a novel design of the commutator for stage 1 of a 16-point radix-4 FFT processor. The ordered data sequence at the output of the complex multiplier for stage 1 of the 16-point FFT processor has to be converted back into a normal data sequence for its stage 2. This data sequence conversion is accomplished by the combination of *ADM* along with a ROM (*ROM0*) for its addressing. The new architecture of the 16-point ordered pipelined FFT processor is shown in Figure 6.10. The input and output sequences of *ADM* namely *DI* and *DO* respectively are shown in Figure 6.11. It is clear that *DO* is in normal order to be directly fed to the stage 2 commutator. The stage 2 commutator will be the same as in the conventional architecture. The stage 1 commutator design, to support the ordering scheme, is discussed in detail now.

Coefficient sequence after stage 1 of a 16-point FFT	*16-bit quantised coefficient sequence*	*Ordered coefficient sequence*	*Ordered quantised coefficient sequence flag, real, imag*
W_0	7fff,0000	W_0	1,8001,0000
W_0	7fff,0000	W_0	1,8001,0000
W_0	7fff,0000	W_0	1,8001,0000
W_0	7fff,0000	W_0	1,8001,0000
W_0	7fff,0000	W_0	1,8001,0000
W_1	7641,cf04	W_0	1,8001,0000
W_2	5a82,a57d	W_0	1,8001,0000
W_3	30fb,89be	W_4	0,0000,8000
W_0	7fff,0000	W_1	0,7641,cf04
W_2	5a82,a57d	W_3	1,cf05,89be
W_4	0000,8000	W_3	1,cf05,89be
W_6	a57d,a57d	W_2	0,5a82,a57d
W_0	7fff,0000	W_2	0,5a82,a57d
W_3	30fb,89be	W_6	1,5a83,a57d
W_6	a57d,a57d	W_6	1,5a83,a57d
W_9	89be,30fb	W_9	1,7642,30fb
Normal switching activity = 198		Switching activity after ordering = 78	

Table 6.3: *Ordered and conventional coefficient sequences for a 16-point radix-4 FFT.*

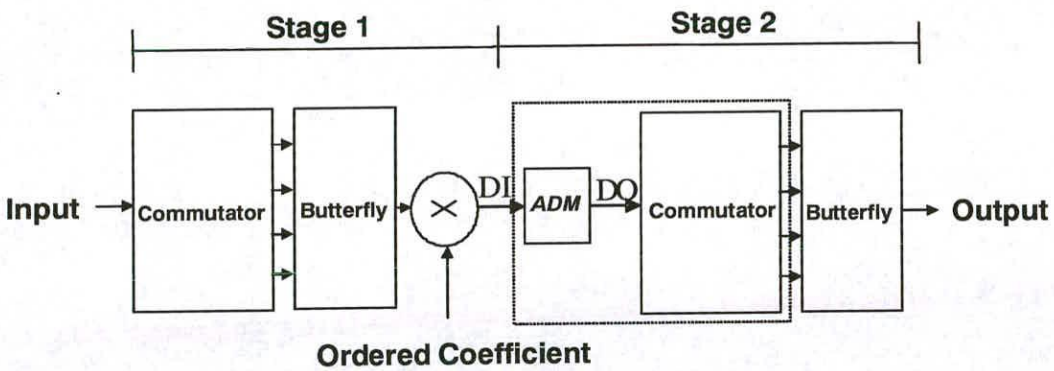


Figure 6.10: *Ordered 16-point radix-4 pipelined FFT processor architecture.*

6.3.2 Stage 1 Commutator design for a 16-point FFT Processor

As seen in equation 6.9 and Figure 6.1, the input data for each summation at stage 1 of a 16-point FFT are separated in time by four words. The timing of the ordered data sequence corresponding to the ordered coefficient sequence and the normal sequence as a function of

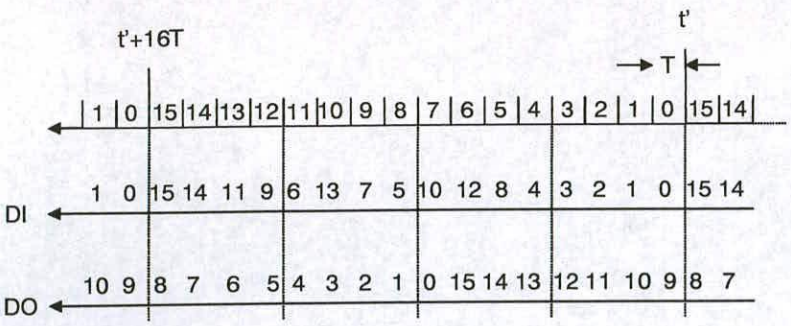


Figure 6.11: *Input and output data sequence of* ADM.

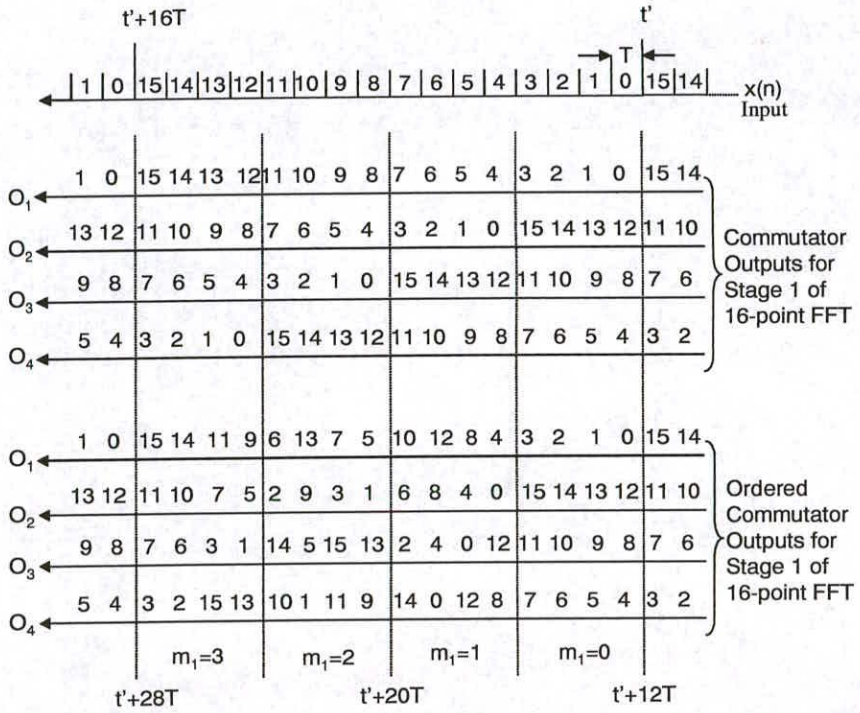


Figure 6.12: *Timing diagram of normal and ordered commutator outputs for the first stage of a 16-point radix-4 pipelined FFT processor.*

time is shown in Figure 6.12, t' is the instant when the first input word arrives. Each input word occupies a word slot of duration T and is numbered according to its appearance in time. This ordered data sequence can be generated with the help of a commutator. It is difficult to generate the ordered data sequence with the help of a conventional FIFO based on shift registers (*SRs*) or *DMs*. In order to achieve flexibility, the commutator is constructed by using double size (eight words) three triple port RAM (*TM*) based FIFOs rather than six four word *DM* based FIFOs. The additional read port in *TM* greatly helps in generating the ordered sequence. The

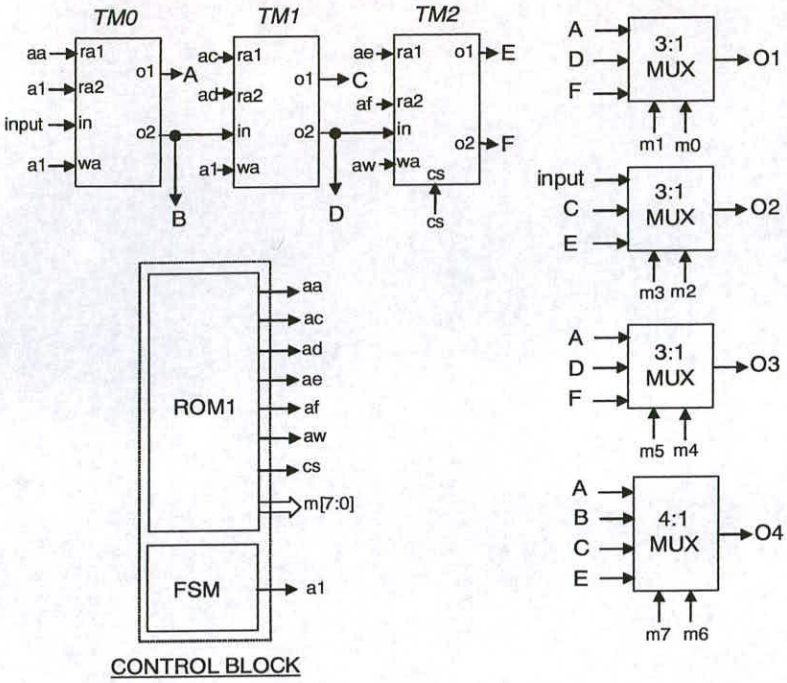


Figure 6.13: *First stage commutator architecture for the ordered 16-point radix-4 FFT processor.*

commutator comprises of three *TMs* (Two read ports and one write port), a finite state machine (*FSM*), a ROM (*ROM1*) and four multiplexers of variable size as shown in Figure 6.13. A *TM* acts as a FIFO with two possible outputs for flexibility. The three *TMs* have six possible outputs but only four outputs are chosen at any time by the four multiplexers depending upon the desired ordered sequence. Each *TM* has a depth of eight words for stage 1 of a 16-point FFT processor. The three bit sequential address *a1* is generated by the *FSM* of the CONTROL BLOCK for the write ports of *TM0* and *TM1* and the B port of *TM0* respectively. The *ROM1* of the CONTROL BLOCK provides the addresses of all the other read and write ports along with multiplexer controls to generate the ordered data sequence. It also helps to keep the unused outputs of *TMs* to their previous values. The *ROM1* contents are listed in Table 6.4. The lower byte of the 31-bit wide *ROM1* controls the four multiplexers (m[7:0]). The next more significant 18-bits generate the write address (*aw*) and the read addresses (*adrf,...,adra*) of *TMs*. The still higher 3-bits control the stage 2 butterfly (c[2:0]) and the most significant bit controls the chip select of *TM2* (*cs*). *TM2* is selectively disabled for writing by the logic high on its chip select input. This is done to avoid unnecessary writing of *TM2* thereby reducing its power consumption. It is clear from the highlighted nibbles of Table 6.4 that the lower three bits (*adra*) of these nibbles remain fixed in the two blocks. It means that *adra* remains fixed for all these locations. This results in

Address	*Contents* {*cs,c[2:0],aw,adrf,adre,adrd,adrc,adra,m[7:0]*}
0000	2e880**4**, 29
0001	1b418**4**, e4
0010	2fac86, 29
0011	16a34d, 89
0100	17a7df, 89
0101	38534**d**, e5
0110	34659**5**, 9a
0111	2c1a6**d**, a9
1000	2c3ef**d**, a9
1001	38659**5**, e5
1010	3877**dd**, e5
1011	000000, 41
1100	008041, 41
1101	010082, 41
1110	0180c3, 41
1111	16010**4**, 41

Table 6.4: *Contents of ROM1 for generating addressing and control signals for the commutator.*

no switching activity on port A for almost half of the time duration. This addressing approach reduces the switching activity on the unused ports and therefore the power consumption. This sort of addressing is also employed for ports C, E and F. This work is reported in [94, 95].

The conventional butterfly structure is also modified for low power by replacing programmable adder/subtractors by a combination of control inverters and a summer and is explained in the next section.

6.3.3 Low power butterfly

The low power butterfly, shown in Figure 6.14, is obtained by using six control inverters (CI_1 to CI_6) to generate the normal or the complement form as per the control signals C_5, C_6 and C_7. The C_4 signal controls the select lines of the four multiplexers (M_1 to M_4) for directing appropriate data to the inputs of the summer. Two four input summers (SUM_1 and SUM_0) are needed to generate both the real (XO_r) and the imaginary component (XO_i) of the output. Two summer inputs arrive directly from the preceding commutator whereas the remaining two inputs come from the multiplexers. The four complex inputs to the butterfly are $X_{r0} + jX_{i0}$ to $X_{r3} + jX_{i3}$. This architecture consumes less power due to the realisation of subtraction

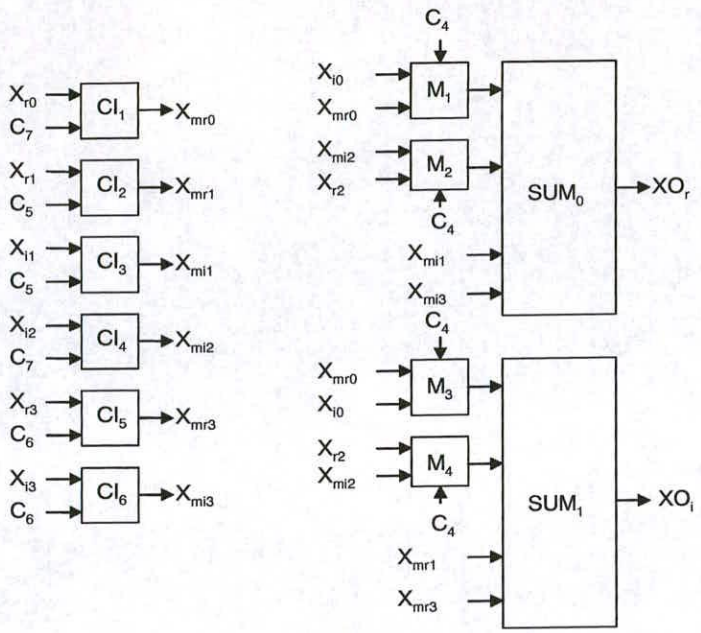


Figure 6.14: *Low power butterfly architecture.*

through control inversion. This approximation of realising two's complement through inversion introduces error in the butterfly operation. This error is insignificant for short FFTs.

The complex multiplier for the ordered pipelined FFT processor architecture is obtained by modifying the conventional complex multiplier to support the order based processing scheme for the first stage of a 16-point pipelined FFT processor. The ordered complex multiplier is discussed in the next section.

6.3.4 Ordered complex multiplier

The ordered complex multiplier is obtained from the conventional complex multiplier by replacing the two's complement multiplier with the multiplier module for only the real part of the coefficient as discussed in chapter 4. The multiplier module comprises of a two's complement multiplier along with a bank of 20 Ex-OR gates for control inversion depending upon the flag bit. The ordered complex multiplier is shown in Figure 6.15.

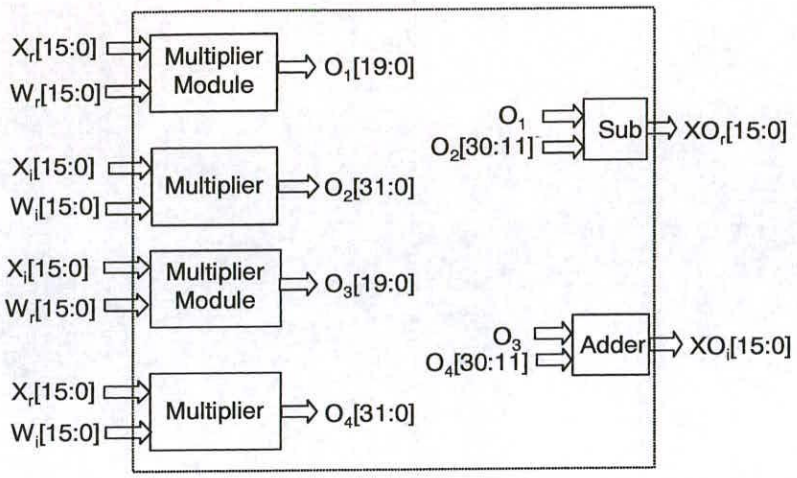


Figure 6.15: *Ordered complex multiplier.*

6.4 Results

The conventional architecture of a 64-point FFT has been implemented in three different ways depending upon the FIFO implementation style. The conventional *SR* and *DM* architectures are obtained by either using a shift register or *DM* based FIFOs in both stage 1 and stage 2 respectively. The *DM-SR* architecture is obtained by using *DM* based FIFO for the bigger stage 1 and *SR* based FIFO for the smaller stage 2. Our *Ordered* low power architecture is based on *TM* based FIFOs with order based processing scheme incorporated into its penultimate stage. All the cores have been designed for three different multiplier types namely, the carry save array type (*csa*), the Non-Booth coded Wallace tree type (*nbw*) and the Booth-coded Wallace tree type (*wall*).

The conventional and *Ordered* pipelined FFT processor architectures have been designed at the register transfer level using Verilog hardware description language. The architectures are then synthesized using 0.18μ CMOS technology library. Power evaluation was then carried out on the gate level netlist with SDF using a supply voltage of 1.8V and a clock frequency of 100MHz for 1000 FFT blocks of uniformly distributed random data samples. The switching activity decreases from 192 to 78, a reduction of 59% as per Table 6.3. The comparative results in terms of power for different FFT lengths, FIFO implementation styles and three common multiplier types are given in Table 6.5. It is clear from Table 6.5 that our *Ordered* architecture gives power savings for the three multiplier types and different FIFO architectures for the 16-point and 64-point FFT processors. The percentage power saving is maximum for the *wall* multiplier. The percentage power saving of our *Ordered* approach is less for *nbw* multiplier

FFT size	*Multiplier type*	*SR based in mW*	*DM based in mW*	*DM-SR based in mW*	*Ordered in mW*	*% saving /SR*	*% saving /DM*	*% saving /DM-SR*
16-point	csa	125.32	135.42	-	96.48	23	29	-
	nbw	93.77	114.28	-	80.19	14	30	-
	wall	106.83	128.67	-	81.43	24	37	-
64-point	csa	351.60	289.14	276.73	252.73	28	13	09
	nbw	296.18	238.06	225.18	217.18	27	09	04
	wall	318.21	262.11	247.96	224.46	30	14	10

Table 6.5: *Comparative power consumption for the Ordered and conventional FFT processors.*

type in most cases but the *nbw* multiplier based architecture consumes less power than the ones based on *csa* and *wall* multipliers. Moreover, *DM* based approach is better for stage 1 of a 64-point FFT whereas *SR* based approach out-performs *DM* for the smaller FIFO required in stage 1 of a 16-point FFT processor. Hence, *DM-SR* architecture for the 64-point FFT is more effective than the *SR-SR* and *DM-DM* architectures. Our *Ordered* approach gives power savings of the order of 23% and 29% with respect to *SR* and *DM* respectively for the 16-point FFT processor using *csa* multiplier. The power saving of our *Ordered* approach is 28%, 13% and 9% with respect to *SR*, *DM* and *DM-SR* respectively for the 64-point FFT processor using *csa* multiplier. The percentage power saving with respect to the overall power consumption will go down further for longer FFTs because the *Ordering* approach is restricted only to stage 1 of a 16-point FFT or stage 2 of a 64-point FFT processor. This restriction is imposed in view of the large *ADM* requirement and commutator design complexities for the initial stages of longer FFTs. This large *ADM* will more than offset any power saving due to ordering in the complex multipliers. Table 6.6 lists the power consumed by the major cells of the pipelined FFT processor for different FIFO implementations. It is clear from Table 6.6 that the *SR* based FIFO architecture is inferior to the other architectures for the 64-point FFT due to the high power consumption in the large FIFO blocks of stage 1 commutator. This high power consumption is attributed to the shifting (switching) of all data samples after every clock cycle in the traditional FIFO based on *SR*. It is also evident from Table 6.6 that the power saving in the *Ordered* approach is taking place not only in the multiplier but also in the novel stage 2 commutator. The novel stage 2 commutator architecture comprises of three double size FIFOs based on *TMs* rather than the traditional six *DM* or *SR* based FIFOs. The new commutator architecture consumes much less power due to less data movement and therefore less switching activity among the three FIFOs as compared to the traditional six FIFOs. The power consumption in

S.No.	*FFT Cells*	*SR based in mW*	*DM based in mW*	*DM-SR based in mW*	*Our Ordered in mW*
1.	Stage 1 Commutator	91.01	53.87	53.84	53.77
2.	Stage 2 Commutator	23.27	33.35	22.77	15.95
3.	Stage 3 Commutator	9.64	9.29	9.37	15.17
4.	Stage 1 Butterfly	1.82	1.63	1.63	1.78
5.	Stage 2 Butterfly	2.01	1.81	1.59	3.04
6.	Stage 3 Butterfly	6.15	6.10	6.10	4.40
7.	Stage 1 multiplier	26.32	26.50	26.35	23.89
8.	Stage 2 multiplier	25.27	25.81	25.81	18.31

Table 6.6: *Comparative major cells power consumption for the Ordered and conventional FFT processor for the 64-point FFT processor with csa multiplier.*

the additional read ports of *TM* is reduced by keeping the outputs of the unused read ports to their previous values by addressing these ports through *ROM1*. The stage 3 commutator consumes more power in the *Ordered* approach as compared to the other approaches because its power consumption also includes the power consumed by *ADM*. The stage 2 butterfly in the *Ordered* approach consumes more power than the other approaches mainly due to the different stage 2 commutator architecture. The stage 3 butterfly in our *Ordered* approach consumes much less power than the other approaches because it has been designed using XOR gates (control inverters) and a summer rather than the traditional programmable adders/subtractors. The stage 1 and stage 2 butterflies in our *Ordered* approach consume more power than the other approaches due to the different input/output conditions in the form of a new stage 2 commutator architecture based on larger size *TMs*. The stage 1 multiplier consumes less power in our *Ordered* approach due to different input/output conditions in the form of a different stage 1 butterfly and stage 2 commutator architectures. The stage 2 multiplier consumes substantially less power in our *Ordered* approach because of the significant reduction in the switching activity at its coefficient input by ordering.

The area overhead of Our 64-point *Ordered* FFT processor architecture is 3%, 9% and 28% with respect to *DM*,*DM-SR* and *SR* FFT processor architectures respectively for the *wall* multiplier type.

6.5 Summary

This chapter has presented a low power ordered radix-4 pipelined FFT processor architecture by incorporating order based processing scheme into the first stage of an existing low power 16-point pipelined FFT processor architecture. The order based processing scheme significantly reduces the power consumption of the complex multiplier. Moreover, power saving also occurs in the *TM* based novel commutator architecture for stage 1 of a 16-point FFT. This results in a reduction of the overall power consumption from 37% to 14% for 16-point FFT and 30% to 4% for 64-point FFT using different types of multipliers and FIFOs. The order based processing scheme is restricted only to the first stage of a 16-point FFT or second stage of a 64-point FFT due to the hardware overhead in the form of a dual port RAM for restoring data order for the subsequent stage. This approach is very attractive for orthogonal frequency division multiplexing (OFDM) based wireless LAN (IEEE 802.11) requiring short FFTs but it can also be applied to the penultimate stage of long FFT. The next chapter presents a low power architecture of a MC-CDMA receiver which is obtained by combining a low power 64-point radix-4 pipelined FFT processor architecture discussed here with a low power Combiner architecture.

Chapter 7
Low power MC-CDMA receiver architecture

MC-CDMA [96–105] is a spread spectrum technology which combines the advantages of OFDM (Orthogonal frequency division multiplexing) [106] and CDMA (Code division multiple access) to produce a spectrally efficient multi-user access system. This access system may be utilised in future mobile wireless systems, and therefore power consumption is important. A frequency domain processing MC-CDMA receiver contains two main system blocks, namely an FFT block to demodulate the OFDM signals and a Combiner block which equalises the signal and separates out the coded users. The Combiner in Our architecture is based on MMSE (minimum mean squared error) detection. This chapter deals with the low power architecture of a MC-CDMA receiver which is obtained by reducing the switching activity and also by shutting down blocks through clock gating. The FFT processor of the receiver is based on a novel coefficient ordered architecture for low power which has been explained in detail in the previous chapter. The blocks within the Combiner module are also clock gated to bring down their power consumption.

This chapter is divided into five sections. The first two sections provide the overview of CDMA and MC-CDMA respectively. The third section describes the modeling of a MC-CDMA transmitter and receiver in Matlab. The fourth section describes the MC-CDMA receiver architectures. The chapter concludes with a section on the comparison of power and area between the conventional and the low power MC-CDMA receiver architectures.

7.1 Overview of CDMA

CDMA is based on spread spectrum communications [8, 107–109]. Spread spectrum is a means of transmission in which the signal occupies a bandwidth in excess of the minimum necessary to send the information. The spreading is performed by a code which is independent of the data at the transmitter. The same code is used at the receiver for despreading and subsequent

data recovery. The receiver has to be synchronised with the transmitter. Despreading is done at the receiver by correlating the received signal with the sychronised replica of the code. The processing gain of the spread spectrum system is expressed as the ratio of the bandwidth of the spread spectrum waveform to that of the data.

Spread spectrum provides multiple user random access communications with selective addressing capability. Multiple access refers to the sharing of a fixed communication channel, such as a wireless channel, by a group of users. The objective of the multiple access scheme is to allow users to share a channel without creating unmanageable interference with each other. The traditional techniques are frequency division multiple access (FDMA) and time division multiple access (TDMA). In FDMA, all users transmit simultaneously but use non-overlapping frequency bands. In TDMA, all users occupy the same bandwidth but transmit sequentially in time. CDMA provides the capability of separating signals transmitted simultaneously in time and which occupy the same bandwidth as well. This is accomplished by assigning each user a unique spreading code that is orthogonal with the spreading codes of all the other users. CDMA is often implemented using direct sequence spread-spectrum techniques (DS-CDMA). According to this technique, the data signal $b(t)$ is modulated by a unique high frequency spreading CDMA code $c(t)$ which comprises of +1/-1 chips as shown in Figure 7.1. CDMA code sequences have statistical properties similar to sampled white noise and are generated by a linear feedback shift register. The processing gain (P) in DS-CDMA is defined as the ratio of CDMA code frequency $(1/Tc)$ and the data frequency $(1/T)$. The processing gain in Figure 7.1 is assumed to be eight for the sake of illustration. The spread signal $s(t)$ has no resemblance with the data signal $b(t)$ and therefore it is impossible to recover the data signal without the knowledge of the used CDMA code sequence $c(t)$. It may be noted that the bandwidth of the spread spectrum signal $s(t)$ is P times the bandwidth of the data signal $b(t)$ and hence the name spread spectrum. The processing gain represents the amount of interference protection provided by the CDMA code because it is a measure of the amount of spreading of data power over a bandwidth. The spread signal $s(t)$ is modulated by a high frequency carrier (f_{car}) to obtain the spread spectrum transmitted signal $s_{tr}(t)$. The DS-CDMA transmitter and the power spectrum of the wideband transmitted signal $s_{tr}(t)$ are shown in Figure 7.2.

The data signal is recovered at the receiver by correlating the received signal with the synchronised replica of the same CDMA chips. Since each user has been assigned a unique orthogonal spreading sequence (low cross correlation with the codes of the other users), and therefore the

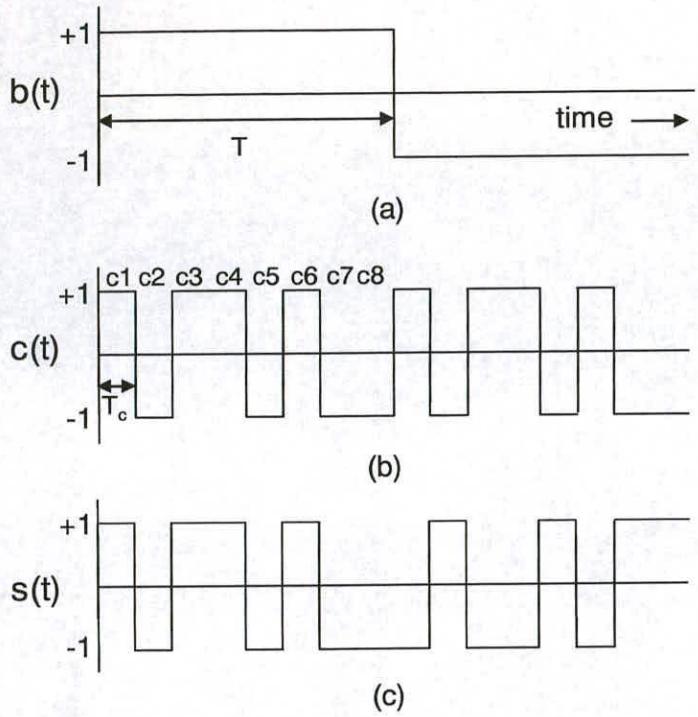


Figure 7.1: *Illustration of the spreading principle with the help of (a) Data signal $b(t)$, (b) CDMA code sequence $c(t)$ and (c) Spread signal $s(t)$, the processing gain is assumed as eight.*

signals corresponding to other users are suppressed at the time of correlation. Unfortunately, each additional user increases the overall noise level thus degrading the quality for all users. The reception of DS-CDMA signal becomes problematic due to the presence of multipath fading channel. The composite received signal is the combination of several signals with different delays in the time domain. The ability of the receiver to recover the transmitted signal is determined by the auto correlation characteristic of the spreading codes.

7.2 Overview of MC-CDMA

Channel dispersion results in the reception of multiple resolvable paths in case of direct sequence CDMA (DS-CDMA). The RAKE receiver used for DS-CDMA has multiple correlators, each synchronised to a different resolvable path. The problem here is that no code exists to ensure that the partial correlation of a slightly delayed path is orthogonal to the dominant line-of-sight path. In DS-CDMA each data symbol requires the entire available spectrum as shown in Figure 7.2. Due to the bursty nature of the channel, several adjacent symbols might

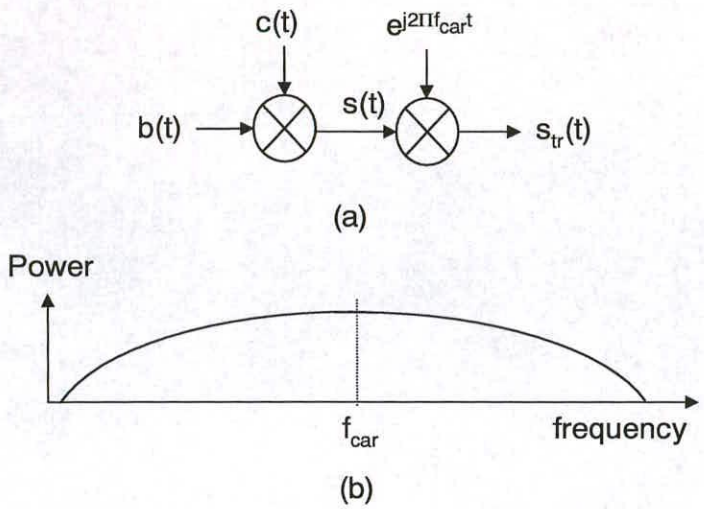


Figure 7.2: *(a) CDMA transmitter and (b) Power spectrum of the transmitted wideband CDMA signal.*

be destroyed during a deep fade.

The main advantage of MC-CDMA is its capacity to spread the signal bandwidth without increasing the adverse effect of delay spread. The chip duration can be equal to the symbol duration in MC-CDMA contrary to the wideband waveforms used in case of DS-CDMA. In MC-CDMA, the user signal is not multiplied by a high speed orthogonal code sequence, but the same bit is transmitted on multiple sub-carriers as shown in Figure 7.3. The power spectrum of the transmitted signal $S_{tr}(t)$ is shown in Figure 7.4. The sub-carrier spacing is equal to the reciprocal of the data symbol duration $(1/T)$. The CDMA chip duration is assumed to be equal to the data symbol duration. The processing gain P here is assumed to be eight for the sake of illustration in Figure 7.3. The processing gain depends upon the length of the CDMA code. The number of sub-carriers is chosen to be more than or equal to the processing gain so as to prevent frequency selective fading. A cyclic prefix having duration longer than the channel delay spread is appended to remove ISI between successive bits. For each user, the sub-carriers are shifted with a 0 or π phase offset. The set of sub-carrier phase offsets follows a signature code sequence to distinguish different users. The resulting signal has a coded structure in the frequency domain and multiple access is possible using the orthogonality of the different codes. If the number of sub-carriers is appropriately chosen, then it is highly unlikely that all the sub-carriers will be located in deep fade and therefore frequency diversity is achieved. If the processing gain is equal to the number of sub-carriers then this system modulates all the sub-carriers with the same data bit, but with a phase shift on each sub-carrier determined by

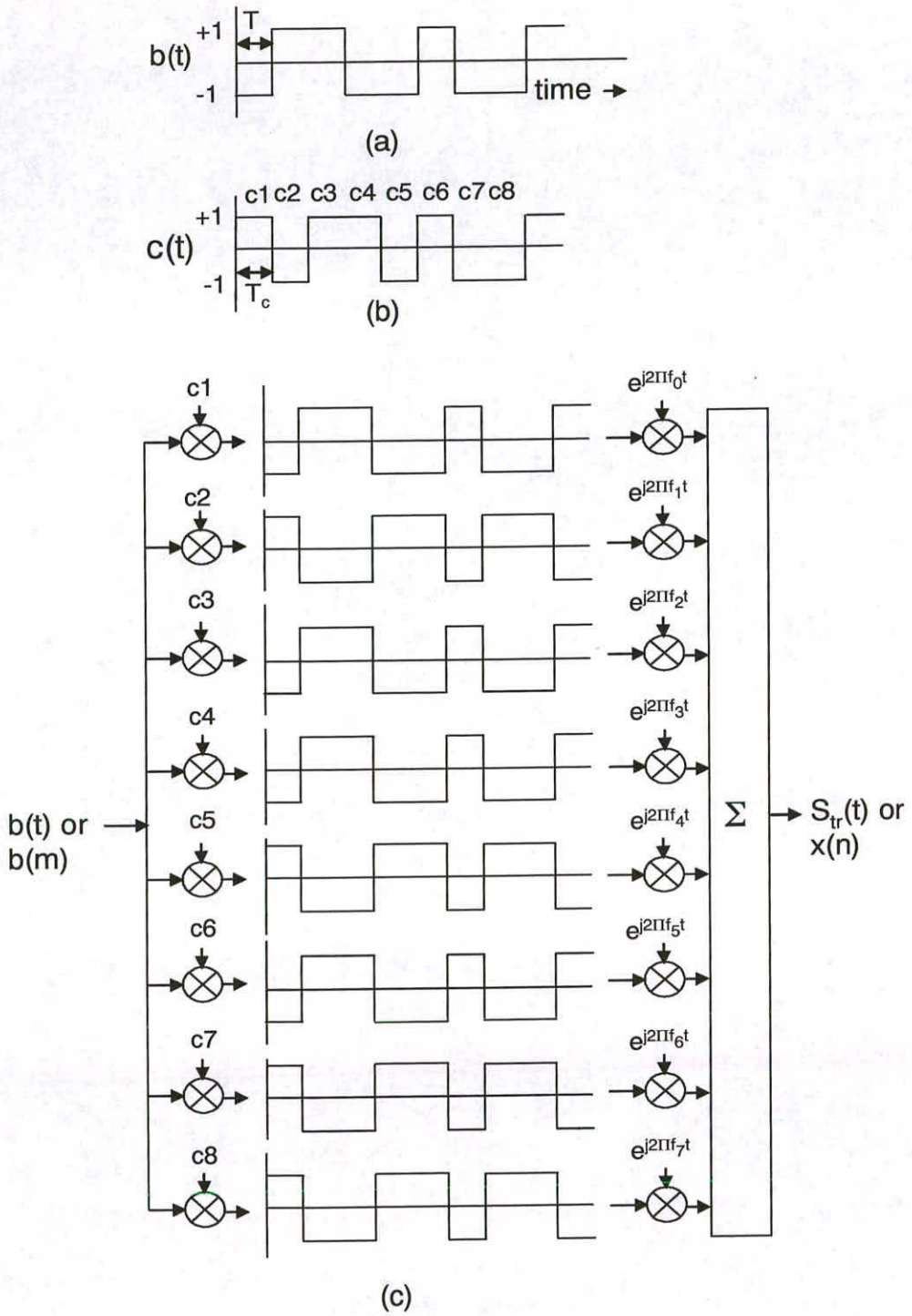


Figure 7.3: *(a) Data signal, (b) CDMA code sequence, (c) Illustration of the one chip/carrier MC-CDMA scheme.*

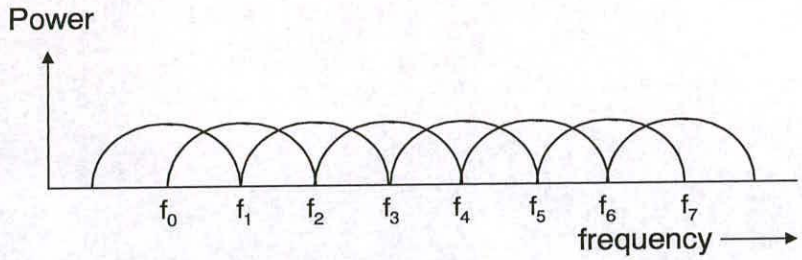


Figure 7.4: *Power spectrum of the MC-CDMA signal depicting eight sub-carriers with their overlapping spectra.*

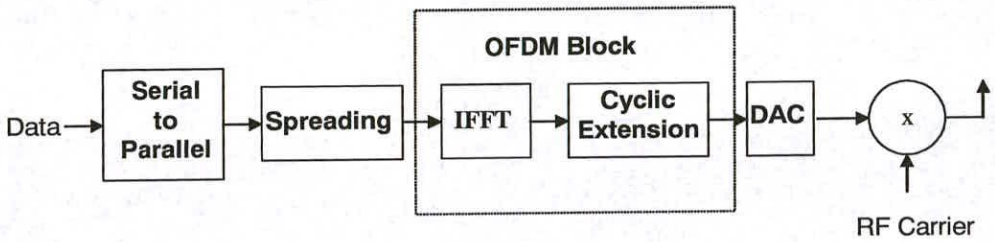


Figure 7.5: *MC-CDMA transmitter.*

the spreading code. MC-CDMA spreads the signal in the frequency domain. This multi-carrier modulation can also be implemented using an inverse FFT.

If the k^{th} chip of the spreading code for user u is defined as $c(k,u) \in$-1,+1 then the transmitted baseband signal for the m^{th} data symbol $b(m)$ is:

$$x(n) = \sum_{n=0}^{N-1} e^{j2\pi kn/N} c(k,u)b(m) \tag{7.1}$$

The baseband signal is then cyclically extended by more than the channel delay spread to remove ISI. The resulting symbol is then passed through DAC prior to upconversion to the high frequency RF carrier. The block diagram of a MC-CDMA transmitter is shown in Figure 7.5. It is very important to have frequency non-selective fading over each sub-carrier to limit the amount of dispersion in MC-CDMA. As described above, MC-CDMA will convert a high data rate signal to low data rate with the help of a serial to parallel Converter before spreading over the frequency domain in order to prevent frequency selective fading.

By using a guard interval, the receiver selects the portion of the signal that is free from ISI. This is then processed by the FFT block to demodulate the sub-carriers. The channel effect of a multipath channel $h(n)$ at the output of the FFT is narrowband for each sub-carrier, $H(k)$, and therefore equalisation and de-spreading can be incorporated into a single combining operation

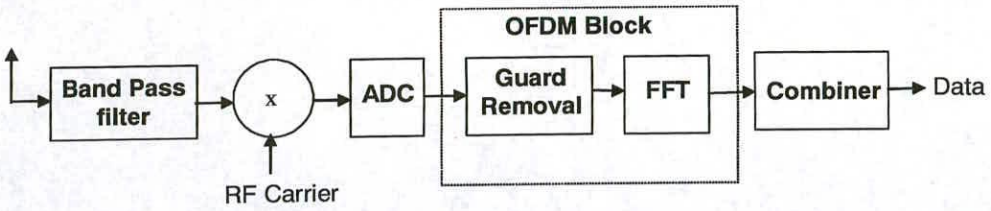


Figure 7.6: *MC-CDMA receiver.*

to estimate the transmitted data bit. If the output of the FFT block at frequency bin k is defined as $Y(k)$ then the combining operation can be represented by

$$x_{rec}(n) = \text{sign}(\sum_{k=0}^{N-1} \Re(c(k,u)A(k)Y(k))) \tag{7.2}$$

The entire receiver structure is shown in Figure 7.6. The Combiner block can be implemented by setting $A(k)$ by equation 7.3 for the minimum mean square error (MMSE) solution, where λ is a parameter dependent upon the signal to noise level and the number of users. The sign bit of the value obtained after accumulation of the real part of the product of the equaliser coefficient $A(k)$, the FFT output $Y(k)$ and the corresponding CDMA chip $c(k,u)$ over the code length of 64, is a measure of the received data estimate.

$$A(k) = H^*(k)/(|H(k)|^2 + \lambda) \tag{7.3}$$

7.3 Modeling of MC-CDMA transmitter and receiver in Matlab

The MC-CDMA transmission frame assumed here is shown in Figure 7.7. Each frame comprises of 32 symbols. The first symbol within each frame acts as a pilot or training symbol. The known pilot symbol is used by the receiver to compute the impulse response of the channel. This channel transfer function is assumed to remain fixed between pilot symbols. This means that a slow fading channel is assumed here. The channel transfer function is then used to estimate the data symbols at the receiver. The processing gain here is assumed to be 64 which is also equal to the number of sub-carriers. Hence, the same data bit is transmitted on all the 64 sub-carriers. The Walsh-Hadamard code having zero cross-correlation is used for spreading the data in the synchronised downlink here. The known pilot bit is fed to the inverse fast Fourier Transform. The IFFT output is a 64-valued complex symbol to which cyclic extension is applied by appending its last 'x' values to the front to avoid inter-symbol interference. The data

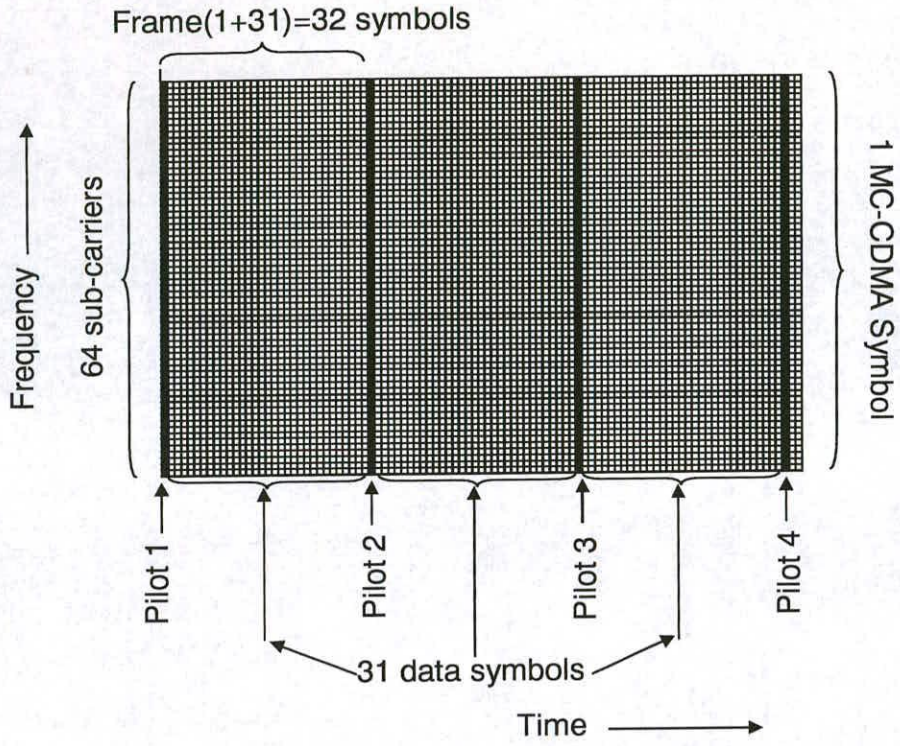


Figure 7.7: *MC-CDMA transmission frame assumed for 64 sub-carriers.*

bits for 'N' users are then spread by multiplying them by their corresponding Walsh-Hadamard code matrix corresponding to 'N' users before summing them. The resultant values are then modulated with the help of inverse fast Fourier transformation. The cyclic prefix is also appended to the output of the inverse fast Fourier transform. The same transmission procedure has to be repeated for all the 31 data bits. The next frame again commences with a known pilot symbol. The number of users is assumed to be 32. The Matlab code for modelling of a MC-CDMA transceiver is listed in Appendix B.

The fading channel is modelled by a five tap FIR filter. The filter tap gains are modelled by complex random numbers which are exponentially decreasing in average power in powers from 0 to -4 in step of unity. This results in a random exponentially decreasing impulse response for the channel. The noise is added to the above channel representation as per the required signal to noise ratio [76].

The combining operation in the receiver can be divided into channel estimation and demodulation phases. In the channel estimation phase, the Combiner extracts the channel information from the training symbol whereas in the demodulation phase, it uses the estimated phase information to recover the data. The receiver first extracts the cyclic prefix from the received

symbols before the start of these phases. The symbols after cyclic prefix removal are fed to the FFT block to restore the signal back into the frequency domain. The data symbols are then despread by multiplying the received signal by the synchronised replica of the Walsh-Hadamard sequence before the combining operation.

The objective of the channel estimation phase is the computation of the equaliser coefficient $A(k)$ given by equation 7.3. If the known input pilot bit is assumed to be unity then each FFT output value corresponding to the pilot symbol (Total 64 FFT output values) is a measure of the channel transfer function at that sub-carrier frequency. The reciprocal of the channel transfer function is then obtained by taking its complex conjugate followed by division with the square of its magnitude. The reciprocal of the channel transfer function is also called the equaliser coefficients for $\lambda = 0$ as per equation 7.3. The channel estimation phase ends after the generation and storage of the equaliser coefficients. These coefficients remains fixed in each frame and are to be computed at the beginning of each frame. A slow fading channel is assumed here. The number of pilot symbols will considerably go up for fast fading channel and therefore the number of data symbols between the two pilot symbols will reduce for fast fading channel.

The channel demodulation phase commences by the multiplication of the FFT outputs corresponding to data symbols by their respective equaliser coefficients followed by summation over all the 64 sub-carriers as per equation 7.2. The sign bit of the accumulated output corresponds to the data bit received. The same procedure has to be repeated for all the 31 data symbols before the start of a fresh channel estimation phase. The estimation and demodulation phases are illustrated with the help of a flowchart shown in Figure 7.8 [76].

7.4 MC-CDMA receiver architecture

A MC-CDMA receiver consists of an FFT processor to extract the frequency contents of the received signal and a Combiner for despreading and equalisation. This section describes a conventional (*Conv*) and our low power (*Our*) MC-CDMA receiver architectures. A conventional 64 sub-carrier MC-CDMA receiver, shown in Figure 7.9, consists of a conventional 64-point radix-4 pipelined FFT processor and a conventional Combiner. The conventional 64-point radix-4 pipelined FFT processor has already been discussed in the previous chapter. The only difference between the conventional and the low power Combiner lies in the disabling of the

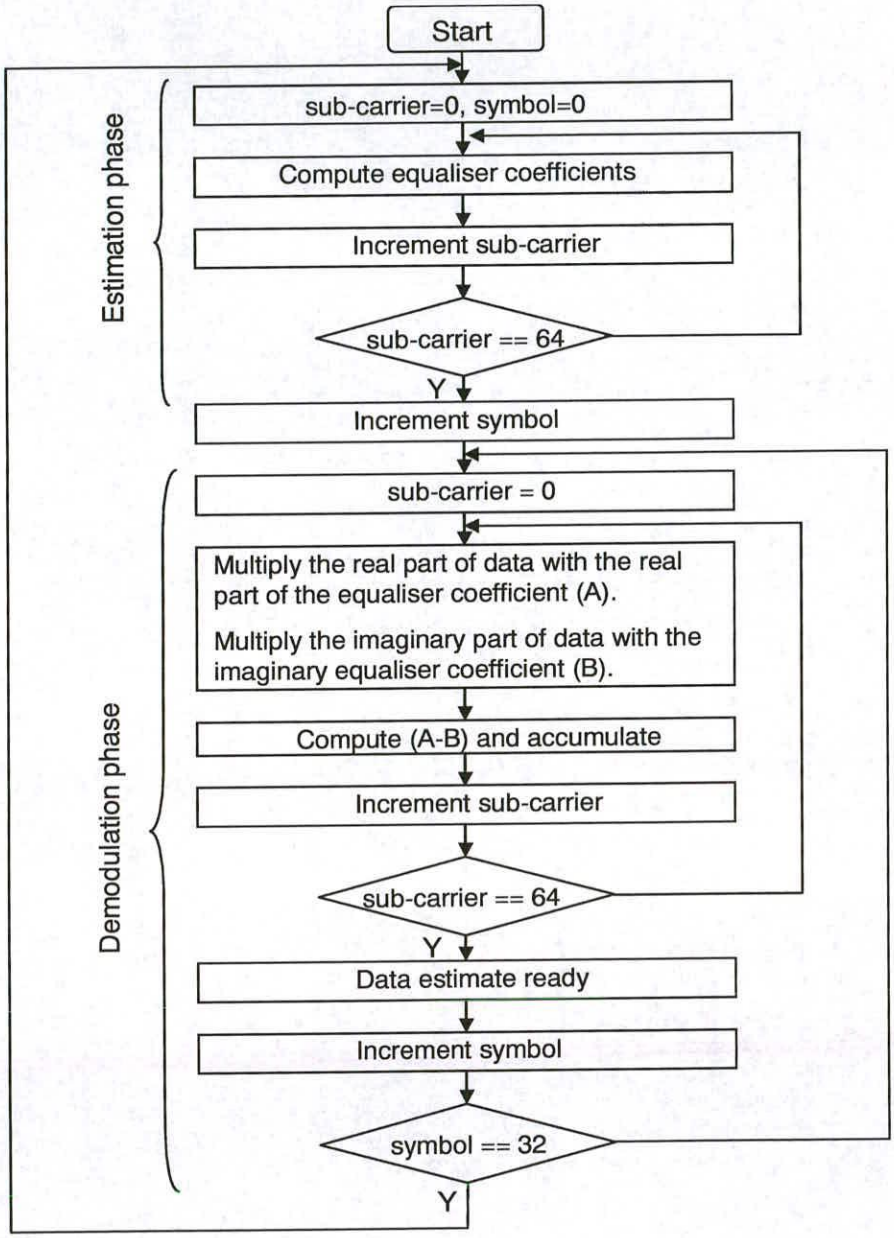


Figure 7.8: *Estimation and demodulation phases in a MC-CDMA receiver.*

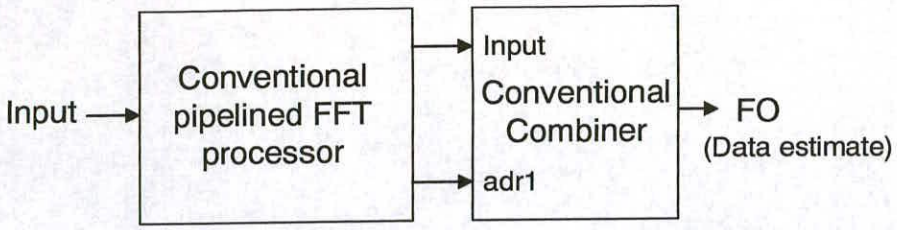


Figure 7.9: *Block diagram of the conventional MC-CDMA receiver.*

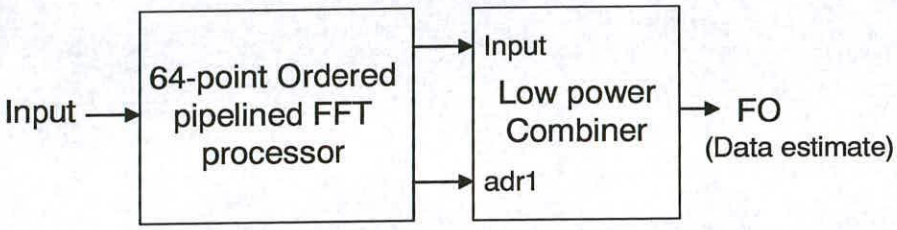


Figure 7.10: *Block diagram of our low power MC-CDMA receiver.*

FIFO and the divider modules in the data demodulation phase through clock gating. Hence, only a low power Combiner block is described in this section. A low power MC-CDMA receiver, shown in Figure 7.10, consists of a low power ordered 64-point radix-4 pipelined FFT processor and a low power Combiner. The ordered pipelined FFT processor architecture is based on altering the ordering of the coefficients fed to the multiplier of the second stage of a 64-point FFT processor so as to minimize the switching activity between successive coefficients resulting in power reduction as discussed in the previous chapter. The low power Combiner is also pipelined resulting in a fully pipelined MC-CDMA receiver architecture. Some of the modules of the low power Combiner are clock gated to reduce its power consumption. This work will be reported in [110].

Since the radix-4 FFT processor architectures have already been covered in detail in the previous chapter, and there is little difference between the conventional and the low power Combiner, therefore, this section explains only our low power Combiner architecture.

7.4.1 Low power Combiner architecture

A Combiner block performs de-spreading, channel estimation and data demodulation to recover the transmitted bits. It first estimates the channel transfer function with the help of some known symbols called pilot symbols. The transmitted data is then recovered by dividing the real part of the received signal by the estimated channel transfer function. A MC-CDMA data frame comprises of pilot and data symbols. The pilot symbol helps in estimating the channel transfer

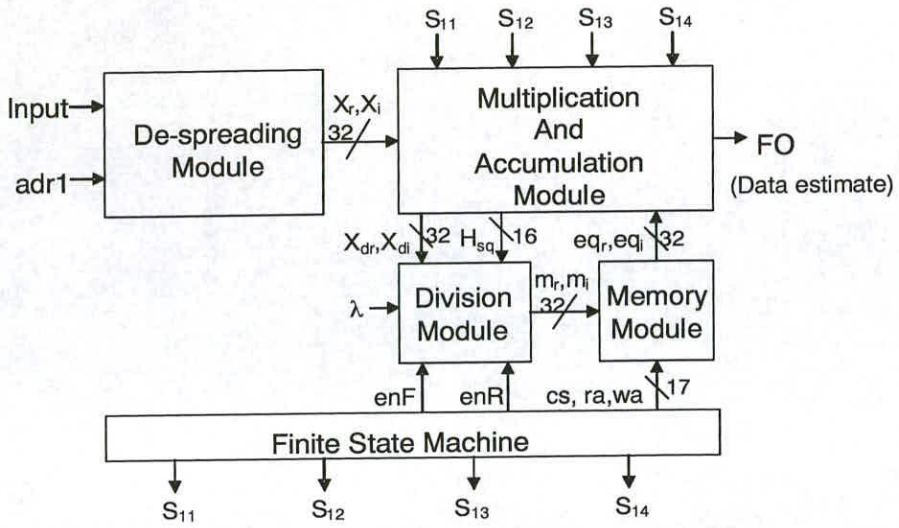


Figure 7.11: *Combiner architecture.*

function. This transfer function is assumed to remain fixed between the two pilot symbols. A 64 sub-carrier system is assumed and the data is divided into blocks of 32 symbols, with the first symbol in each block being used as a pilot for channel estimation. The equaliser coefficients corresponding to all the sub-carriers are computed in the estimation phase. These coefficients are stored in the memory and are used in the demodulation phase. The equaliser coefficients are assumed to be fixed during the demodulation phase. The Combiner architecture is shown in Figure 7.11. It comprises of the following modules:

- De-spreading module.
- Multiplication and accumulation module.
- Division module.
- Memory module.
- Finite state machine.

7.4.1.1 De-spreading Module

The main purpose of this module is to multiply the input (FFT outputs) by either +1 or -1 depending upon the corresponding chip value. This is accomplished by selectively complementing the input for chip value of -1 and letting it through for +1. This task can be performed

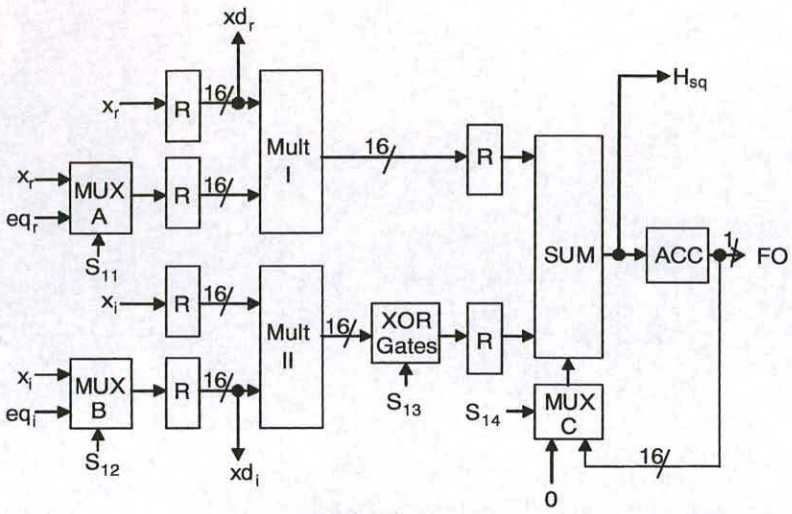


Figure 7.12: *Multiplication and accumulation module.*

easily by a set of XOR gates with one of their input connected to the chip value. The chip values are stored in a 64-bit ROM.

7.4.1.2 Multiplication and accumulation module

This module comprises of two multipliers which are used both in the channel estimation phase as well as in the demodulation phase for reducing the power consumption as shown in Figure 7.12. In the channel estimation phase, the multipliers Mult I and Mult II are fed with the same inputs x_r and x_i respectively to compute the square of the channel transfer function $H(k)^2$. This is accomplished by controlling the select inputs S_{11} and S_{12} of MUX A and MUX B with the help of a FSM. The control input S_{13} is set to '0' in the estimation phase so that the summation of the two multiplier outputs is performed by the summer (*SUM*). The summer based accumulator is used instead of a programmable adder/subtractor to reduce the power consumption. The select input S_{14} in the estimation phase always selects '0' for summing the two multiplier outputs because no accumulation is needed during the channel estimation phase. The output H_{sq} (the square of the channel transfer function) from this module is used by the division module for computation of the equaliser coefficients $A(k)$ given by equation 7.3. The registers are used at the input of the multipliers to reduce glitching at its inputs and therefore the power consumption. Moreover, the registers between the *SUM* block and the multiplier are used for reducing the length of the critical path resulting in low power consumption on account of less buffering. The demodulation phase requires the calculation of the real part of the product of data and the equaliser coefficient $(x_r.eq_r - x_i.eq_i)$ for each sub-carrier followed by their

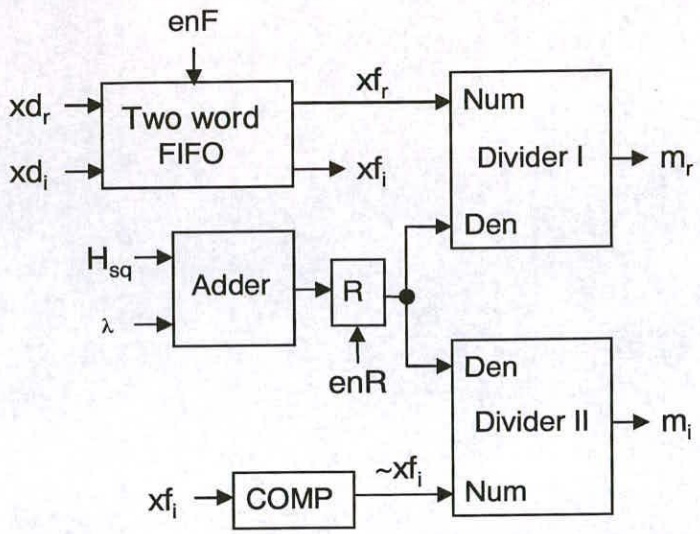


Figure 7.13: *Division module.*

summation over all the sub-carriers as required by equation 7.2. The multiplier Mult I is fed with x_r and eq_r and Mult II with x_i and eq_i. This is again controlled by the select lines S_{11} and S_{12} in Figure 7.12. The XOR control input S_{13} is set to '1' in the demodulation phase to compute the difference between the two multiplier outputs by the *SUM* block. The select input S_{14} in the demodulation phase is set such that it selects a zero input corresponding to the first data sub-carrier and the accumulator output (*ACC*) for the rest of the data sub-carriers. The MSB or sign bit of the accumulated output (FO) over 64 sub-carriers is a measure of the transmitted bit. The outputs xd_r and xd_i are fed to the division module to reduce the size of FIFO by one register level.

7.4.1.3 Division Module

The division module, shown in Figure 7.13, is used to compute the real and imaginary parts of the equaliser coefficients m_r and m_i respectively by using the input pilot symbol and the H_{sq} output from the accumulation module as required by equation 7.3. The real part of the equaliser coefficient m_r is obtained by dividing the real part of the pilot symbol with the summation of H_{sq} and the factor λ. The imaginary part of the equaliser coefficient is obtained by dividing the complement of the imaginary part of the pilot symbol with the previous divisor. The negation of the imaginary part is done to obtain the equaliser coefficient tending towards the reciprocal of the channel transfer function for $\lambda = 0$ as per equation 7.3. This is required to remove the channel effects later by multiplying the incoming data block with the equaliser coefficients.

The two word FIFO block and the register *R* are enabled only during the channel estimation phase by the *enF* and *enR* signals. This helps in reducing the power consumption by reducing the switching activity at the divider inputs and inside the FIFO block. The FIFO is needed to synchronise the numerator and the denominator of the divisor. A 1's complementer block (*COMP*) is also needed to generate the complement of the imaginary part of the FIFO output. The division module does not consume much power because it is active only during the short channel estimation phases. The division module is based on two dividers namely *Divider I* and *Divider* II with 'Num' indicating numerator and 'Den' for denominator.

The difference between the low power and the conventional Combiner lies in the provision of gated control signals *enF* and *enR*. These gated signals are absent from the conventional Combiner architecture because it does not support clock gating. This means that the dividers and the *FIFO* consume power even during the data demodulation phase in the conventional Combiner.

7.4.1.4 Memory Module

The memory module is used to store the equaliser coefficients. The memory size required in a 64 sub-carrier system is 64-words. A dual port RAM is used because of the need of simultaneous read and write operations due to slight overlap of the channel estimation and demodulation phases in the pipelined receiver. This phase overlap requires reading and writing the memory at the same time.

7.4.1.5 Finite State Machine Module

The Finite state machine in Figure 7.11 generates 23 control signals. The control signals S_{11}, S_{12}, S_{13} and S_{14} control the blocks in the Multiplication and Accumulation module depending upon the receiver's phases whereas enable signals *enF* and *enR* control the selective enabling of blocks in the Division module. For instance, during the channel estimation phase, S_{11} and S_{12} select the FFT output x_r and x_i whereas during the data demodulation phase, these signals select the equaliser coefficients eq_r and eq_i respectively. This module also generates the read address (*ra*) and the write address (*wa*) for the dual port RAM.

Multiplier type	*Conv* (mW)	*Our* (mW)	% Power saving
csa	148.23	129.55	13
nbw	140.73	126.74	10
wall	143.48	126.65	12

Table 7.1: *Power consumption comparison of the MC-CDMA receivers for different multiplier types.*

Major receiver blocks	*Conv* (mW)	*Our* (mW)	% Block power saving
FFT	100.35	92.76	9.3
Combiner	39.80	28.86	27

Table 7.2: *Power consumption comparison of the major blocks of the MC-CDMA receiver for* csa *multiplier.*

7.4.2 Results

The *Conv* and *Our* MC-CDMA receiver cores have been designed at the register transfer level (RTL) using the Verilog hardware description language. The cores were then synthesized using SYNOPSYS *DesignCompiler* with the UMC 0.18μ standard cell CMOS library. Layouts of the cores were generated using Envisa *Silicon Ensemble* place-and-route software. This was followed by extracting RC information and then performing RC back-annotated post layout gate-level netlist simulations for 4000 received data samples using *Verilog-XL* simulator. The resulting data including switching activity of the circuit nets and capacitive load information extracted from the layout was then used by the Synopsys *DesignPower* to compute the power consumption of the receiver cores. All the simulations were carried out at a clock frequency of 50MHz. The input data for the FFT is obtained by modelling the transmitter and receiver in Matlab for 32 users with a signal-to-noise ratio equal to 40. The receiver hardware has been verified for other values of the number of users and signal to noise ratios. The power consumption results are almost independent of the number users and the signal to noise ratio. The simulation results are listed in Tables 7.1, 7.2 and 7.3. It is clear from Table 7.1 that the total power saving of *Our* is maximum for *csa* multiplier and minimum for *nbw*. Table 7.2 lists the power consumed by the two major blocks of the receiver namely the FFT and the Combiner. The power saving in the FFT module is only 9.3% whereas the power saving in the Combiner module is 27% for the *csa* multiplier type. The power saving in the Combiner block will decrease for

Major Combiner blocks	*Conv* (mW)	*Our* (mW)
memory	20.45	19.65
FSM	0.234	0.245
mult	4.79	4.54
sum	0.30	0.28
acc	0.29	0.32
divider	10.30	0.595
fifo	1.17	0.741

Table 7.3: *Power consumption in the various blocks of the combiner for* csa *multiplier.*

Receiver architecture	Area(mm^2)	% Area overhead
Conv	2.22	-
Our	2.27	2.25

Table 7.4: *Area comparison of the MC-CDMA receiver for* csa *multiplier.*

fast fading channels because the required frequency of channel estimation goes up for these channels. It is clear from Table 7.1 that the total power consumed is higher than the sum of the individual block power. This power difference is due to the switching power consumed in the input/output nets of the top level modules of the receiver. Table 7.3 lists the power consumed in the various blocks of the Combiner for both *Conv* and *Our* receiver architectures. The memory used to store the equaliser coefficients consumes most of the power in both the architectures. The divider modules consume much more power in the *Conv* architecture as compared to *Our* because the inputs to the divider modules are kept fixed during the demodulation phase in *Our* architecture. The divider module is needed only in the channel estimation phase and this means that it can be disabled by holding its inputs fixed during the demodulation phase. This fixation of divider inputs (Zero switching activity) is achieved by clock gating both the FIFO and the register. The FIFO holds the dividend whereas the divisor is stored in a register *R*. This technique reduces the divisor power significantly because the demodulation phase is much longer than the channel estimation phase. The area overhead of *Our* architecture is 2.25% as per Table 7.4. The layouts for the *Conv* and *Our* architectures for *csa* multiplier type are shown in Figures 7.14 and 7.15 respectively.

7.5 Summary

This chapter has presented a low power MC-CDMA receiver architecture for a 64 sub-carrier system. The same architecture could be extended to any number of sub-carriers. The low power receiver architecture is based on the 64-point radix-4 ordered FFT processor. The Combiner architecture also employs extensive clock gating to reduce its power consumption. Power comparison results with respect to the Conventional receiver architecture are presented to demonstrate the effectiveness of the techniques used. The next chapter proposes a reconfigurable MC-CDMA receiver architecture which is based on a reconfigurable radix-4 FFT processor and a reconfigurable Combiner. The reconfigurability is exploited to design an adaptive MC-CDMA receiver such that its architecture varies with the channel parameters and is not designed for the worst case channel conditions. This reconfigurability saves power in switching from long FFTs to shorter FFTs etc.

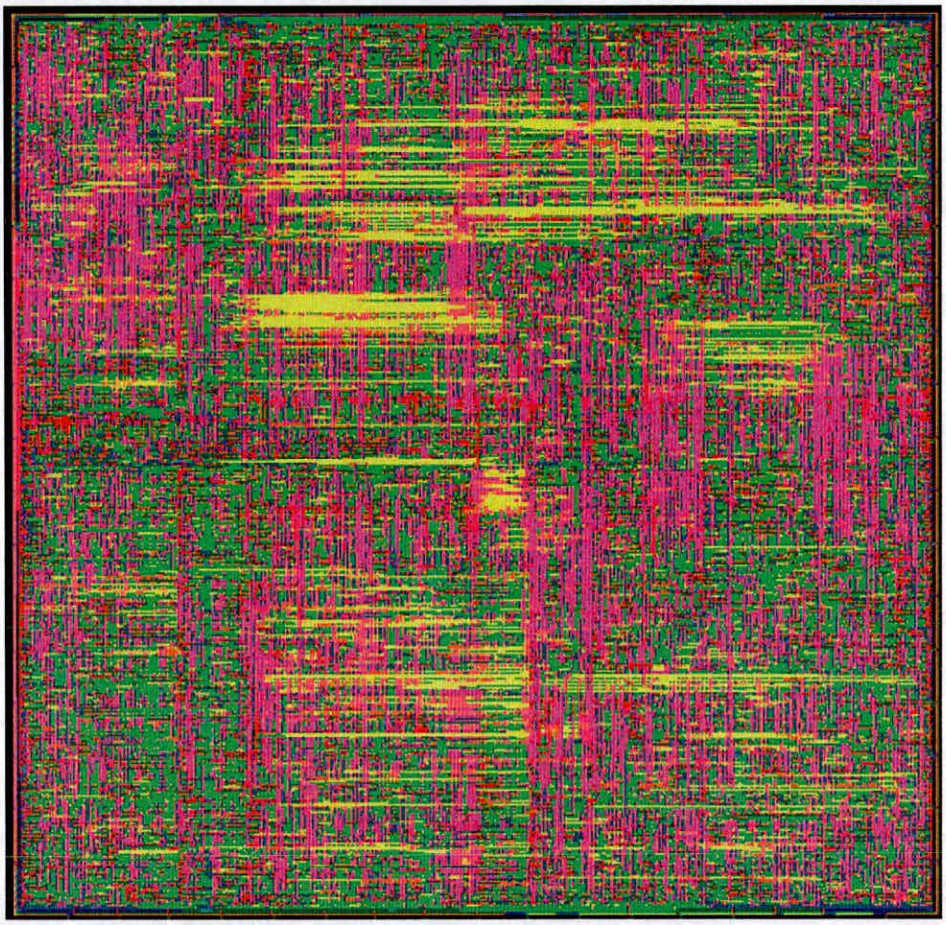


Figure 7.14: *Layout of the Conventional MC-CDMA receiver core, Power = 148.23mW, Area=2.22mm^2.*

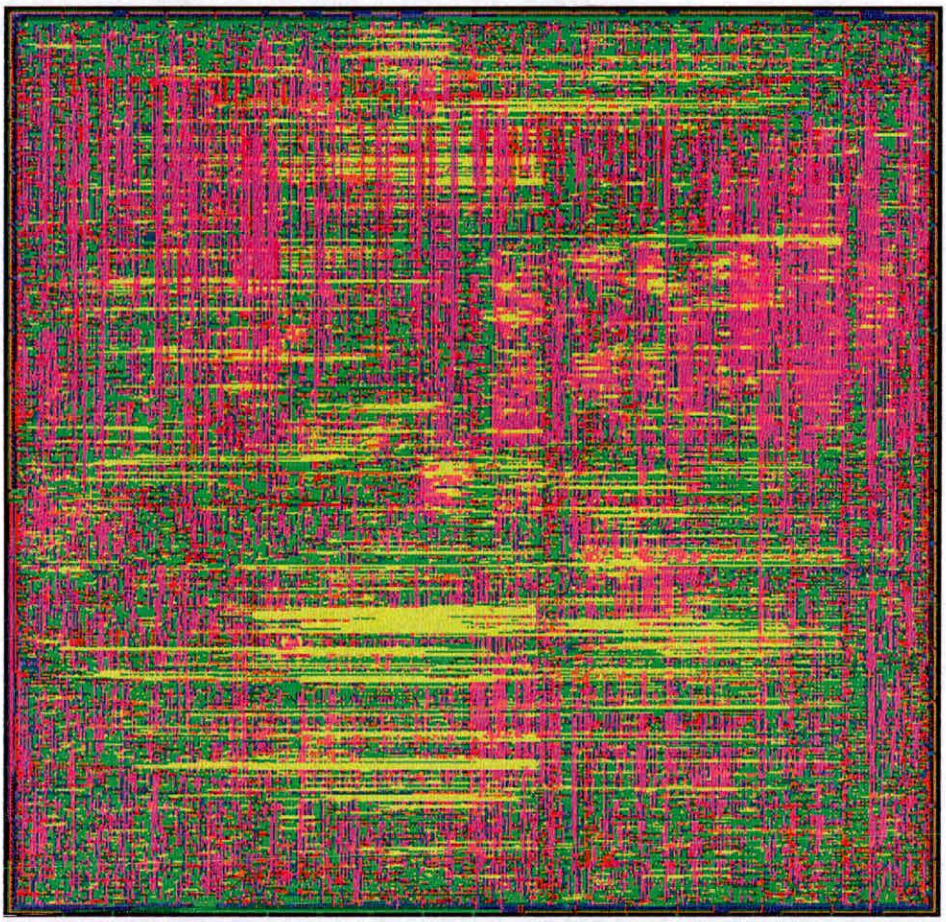


Figure 7.15: *Layout of Our low power MC-CDMA receiver core, Power = 129.55mW, Area=*$2.27mm^2$.

Chapter 8
Low power reconfigurable MC-CDMA receiver architecture

This chapter proposes a novel concept of adjusting the receiver hardware size in real time as per the channel parameters in multicarrier wireless receivers. The FFT is one of the most power consuming block in multicarrier receivers. The FFT size in a OFDM/MC-CDMA based wireless receiver varies from 1024-point to 16-point depending upon the channel parameters. A low power reconfigurable radix-4 256-point FFT processor architecture is proposed here that can also be configured as a 64-point or 16-point as per the channel parameters to prove the concept. By tailoring the clock of the higher FFT stages for longer FFTs, significant power saving is achieved by switching to shorter FFTs from longer FFTs. This channel parameter driven approach is also applied to the Combiner module of the receiver. This approach can also be applied to other blocks of the receiver like the Viterbi decoder.

This chapter is organised into five sections. The chapter starts with a section on the motivation behind this approach. The second section establishes the relationship between the FFT size and the channel parameters. The third section describes two 256-point reconfigurable FFT processor architectures that can also be reconfigured as 64-point or 16-point. The fourth section proposes two 256 sub-carrier MC-CDMA receiver architectures that can also be configured for 64 sub-carriers. The chapter concludes with the power saving results for both the reconfigurable FFTs and the reconfigurable MC-CDMA receivers.

8.1 Motivation

The critical design issue for future wireless receivers is the combined requirements of high-performance, low power and flexibility. Wireless systems have diverse application requirements in the form of changing data rate and bit error rate along with changing bandwidth and other channel parameters like the delay spread. It is desirable for wireless receivers to adapt their operation instead of being designed for the worst case scenario.

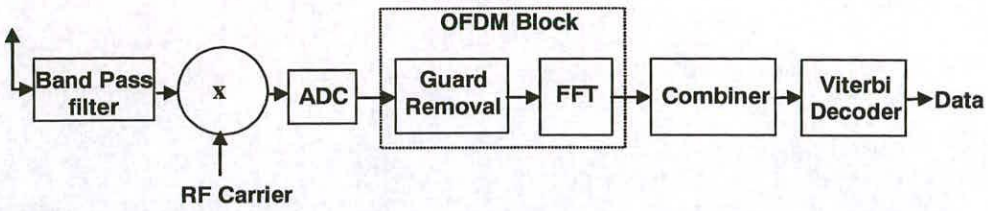


Figure 8.1: *MC-CDMA receiver.*

In multi-carrier systems like multi-carrier code division multiple access (MC-CDMA) or Orthogonal frequency division multiplexing (OFDM), the two most power consuming blocks in the receiver are the FFT and the Viterbi decoder [90] as shown in Figure 8.1. Researchers have already investigated low power architectures for these two important blocks [53, 94, 111]. The way forward to reduce the power consumption further is to dynamically reduce the complexity of the receiver architecture in real time as per the changing channel requirements like the delay spread, signal to noise ratio (SNR), bandwidth and bit error rate etc. In [112], the researchers have shown the potential of saving power in a Viterbi decoder by dynamically varying its architecture according to real-time changes in system characteristics.

8.2 Dependence of FFT size on channel parameters

In a basic OFDM system, a guard interval popularly known as a cyclic prefix is inserted in every symbol to overcome the effect of ISI. The guard interval needs to be longer than the delay spread of the channel. The OFDM symbol duration is chosen to be about five times longer than the guard interval in the interests of transmission efficiency [90]. The number of sub-carriers (FFT size) is determined by the following:

(OFDM symbol duration-guard interval)* bandwidth.

In MC-CDMA, the number of sub-carriers depends upon the delay spread (τ), maximum Doppler frequency (f_d) and the transmission rate (R) [16]. Figure 8.2 shows the variation of the optimum number of sub-carriers as a function of the normalised value of delay spread ($\tau/(R*P)$, where P is the processing gain) for different values of the normalised maximum Doppler frequency Nf_d ($f_d/(R*P)$). It is clear from Figure 8.2, that for the intermediate range of delay spread, f_d and R, the optimum number of sub-carriers vary from 16 to 1024. Only in extreme cases, the optimum number of sub-carriers go beyond this range. The bottom plot corresponds to $Nf_{d1} = 10^{-3}$, whereas the top plot corresponds to $Nf_{d6} = 10^{-8}$.

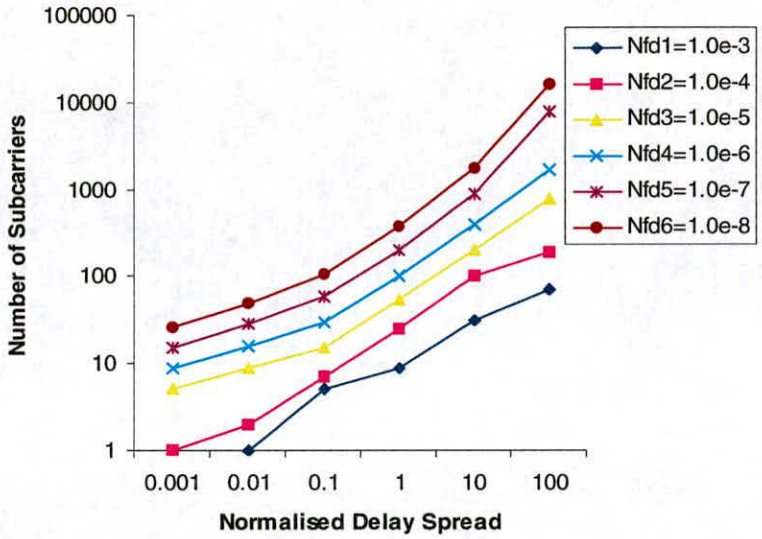


Figure 8.2: *Optimum number of sub-carriers as a function of normalised delay spread for different maximum Doppler frequencies.*

It has been clearly established that in both MC-CDMA and basic OFDM, the number of sub-carriers (FFT size) is a strong function of the delay spread. Since the indoor delay spread is measured in the range from 30ns to 370ns depending on the building size [90] and the outdoor delay spread is much longer, it is desirable to design a reconfigurable FFT processor whose size can be tailored as per the channel parameters like the delay spread, the maximum Doppler frequency and the transmission rate in real time. The basic idea is to design the receiver for the maximum number of sub-carriers (FFT size) and then clock gate unused blocks for the smaller sizes depending upon the delay spread. The Combiner architecture can also be made reconfigurable by clock downing the unused segmented memory used for storing the equaliser coefficients for smaller number of sub-carriers. This approach of hardware size adjustment on the basis of changes in the channel parameters in real time can be extended to other blocks of the receiver like the Viterbi decoder for saving even more power. This work is reported in [113].

The switching to the appropriate FFT size, Combiner and Viterbi decoder architectures will be done automatically by the receiver after reading the channel parameters like the delay spread, SNR and the bit error rate in real time. This reading operation has to be carried out at a much lower frequency than other operations and therefore the power overhead is minimal.

The FFT stages can be constrained at the synthesis stage in such a way that the smaller stages can be able to operate at a higher frequency to support the higher bit rate demand for shorter FFTs as compared to the higher stages.

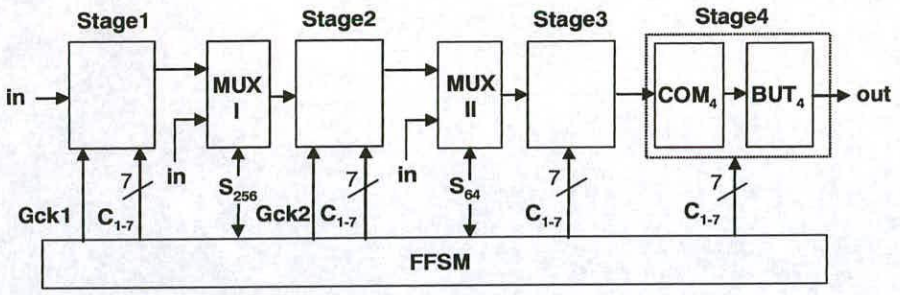


Figure 8.3: *Architecture of the radix-4 256-point reconfigurable FFT processor.*

8.3 Reconfigurable FFT processor architecture

This section proposes to adjust the FFT size in real time as per the channel parameters instead of using a fixed large FFT based receiver designed for the worst case channel parameters like the delay spread, transmission rate and the maximum Doppler frequency. The FFT size in MC-CDMA varies from 16-point FFT to 1024-point FFT [16] depending upon the channel parameters. Significant power saving is achieved by using the most appropriate FFT size instead of a fixed large FFT size for worst case channel conditions. This is achieved by monitoring the channel parameters in real time. In this work, a shorter 256-point reconfigurable radix-4 FFT processor architecture has been proposed that can also be configured as a 64-point or 16-point by tailoring the clocks of the higher stages in real time to prove the concept. The hardware overhead in the form of logic for monitoring the delay spread and other channel parameters is minimal because this logic will operate at a much lower frequency.

The reconfigurable FFT processors are based on the Bi and Jones radix-4 pipelined architecture [13]. It is better than other pipelined architectures in terms of computational efficiency and hardware savings in complex multipliers, adders and data stores. It consumes less power due to less hardware requirement as compared to the other radix-4 pipelined architectures [89]. A reconfigurable 256-point radix-4 pipelined architecture is proposed here for reducing the power consumption as compared to a radix-2 architecture.

A reconfigurable 256-point radix-4 pipelined FFT processor comprises of four radix-4 stages as shown in Figure 8.3. Reconfigurability is achieved by inserting two multiplexers namely MUX I and MUX II between the higher stages for directly routing the input data to stage 2 or stage 3 depending upon the required FFT size. The reconfigurable FFT processor can also act as a 64-point processor by feeding the input data directly into stage 2 and clocking down the first stage. This is accomplished by selecting the input data rather than the output of stage 1 by the

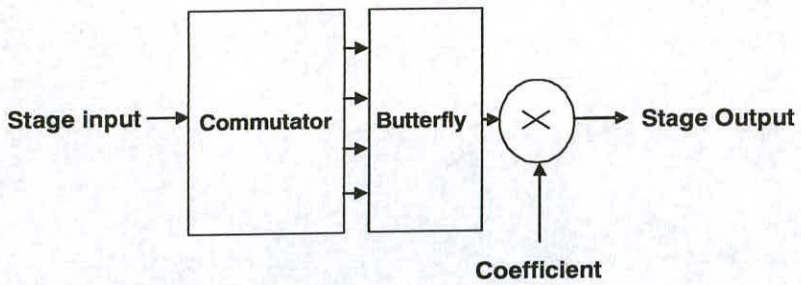


Figure 8.4: *Architecture of a basic stage of a radix-4 256-point reconfigurable FFT processor.*

external select input S_{256} of MUX I. Moreover, the gated clock input (G-ck1) to stage 1 is also disabled by the FFT Finite state machine (FFSM). Similarly, it can act as a 16-point processor by controlling the select input S_{64} of MUX II to feed the input data directly into stage 3 and disabling stage 1 and stage 2 with the help of gated clocks G-ck1 and G-ck2.

The basic stage of the FFT processor comprises of a commutator, a butterfly and a complex multiplier as shown in Figure 8.4. The last stage contains just the commutator (COM_4) and the butterfly (BUT_4) as per the architecture described in Chapter 6. The architecture of the commutator, butterfly and the complex multiplier have been discussed in detail in Chapter 6. The commutator contains six FIFOs which are realised using dual port RAMs for low power consumption. The FFT Finite state machine (FFSM) is responsible for generating all the control signals for all the FFT stages. It is a combination of four different finite state machines (FSM) i.e. one FSM per stage. Each FSM generates seven control signals C_{1-7} for its stage. The select lines of MUX I and MUX II are activated by the external inputs S_{256} and S_{64} as per the FFT size.

Two different reconfigurable FFT processor architectures are proposed depending upon the extent of clock gating. In the first architecture *RFFT-I*, clock gating is applied to all the blocks of the commutator including the dual port RAM based FIFOs for stages 1 and 2 of the FFT processor. The second FFT processor architecture namely *RFFT-II* is obtained by limiting clock gating to the registers and FSMs of stage 1 and stage 2 only. This selective clock gating approach reduces the power overhead of the reconfigurable FFT over the fixed FFT of the same size. On the other hand, it reduces the power saving in switching from longer to shorter FFTs. This reduction in power saving is due to the power consumed in the dual port RAMs for stage 1 and stage 2 commutators on account of the clock signal activity in these stages even for shorter FFTs.

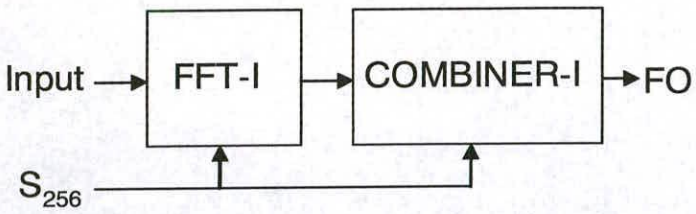


Figure 8.5: *Block diagram of* RECEIVER-I.

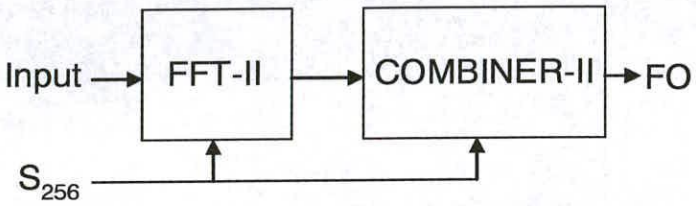


Figure 8.6: *Block diagram of* RECEIVER-II.

8.4 Reconfigurable MC-CDMA receiver architecture

This thesis proposes two different reconfigurable architectures for the 256 sub-carrier MC-CDMA receiver namely *RECEIVER-I* and *RECEIVER-II*. The difference between *RECEIVER-I* and *RECEIVER-II* architectures lies in the extent of clock gating within their FFT and Combiner blocks. Clock gating is applied to most of the unused blocks including the dual port RAMs of the FFT and the Combiner in *RECEIVER-I* whereas it is selectively applied to only the registers and FSMs of *RECEIVER-II*. The block diagrams of *RECEIVER-I* and *RECEIVER-II* are shown in Figure 8.5 and Figure 8.6 respectively.

RECEIVER-I comprises of a 256-point reconfigurable FFT processor (*FFT-I*) tailored for the MC-CDMA receiver and a reconfigurable Combiner block *COMBINER-I* with a partitioned equaliser memory that can be partly disabled for a 64 sub-carrier receiver. The reconfigurable *FFT-I* processor consists of an ordering block *ORD* for restoring the digit reversed FFT output into normal order for onward processing. The clock gating in *FFT-I* processor is applied to all the blocks of stage 1. It is important to note that the external select input S_{256} is used for disabling the blocks of both *RECEIVER-I* and *RECEIVER-II*.

RECEIVER-II consists of a 256-point reconfigurable FFT processor (*FFT-II*) tailored for the MC-CDMA receiver and a reconfigurable Combiner block, *COMBINER-II* with clock gating limited to only its FSM for the 256 sub-carriers. The reconfigurable *FFT-II* processor also consists of an *ORD* block like *FFT-I*. The clock gating in *FFT-II* processor is limited to the registers and FSM of its stage 1.

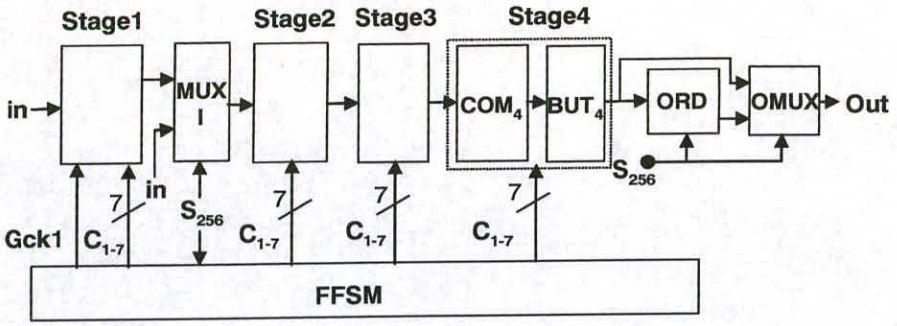


Figure 8.7: *Architecture of the 256-point reconfigurable FFT processor used in the reconfigurable 256 sub-carrier MC-CDMA receiver.*

The selective application of gated clock reduces the power overhead of the 256 sub-carrier reconfigurable receiver as compared to a fixed 256 sub-carrier receiver but it also reduces the power saving in going from 256 sub-carriers to 64 sub-carriers receiver. The next subsection is about the proposed reconfigurable FFT processor architectures namely *FFT-I* and *FFT-II* for the MC-CDMA receivers.

8.4.1 Architecture of the 256-point reconfigurable FFT processor used in the reconfigurable receiver

The architectures of the reconfigurable FFT processors namely, *FFT-I* and *FFT-II* are tailored to the requirements of the reconfigurable MC-CDMA receiver. Since the reconfigurable MC-CDMA receiver assumed here supports only 256 and 64 sub-carriers and therefore only 256-point and 64-point FFT sizes are required. Moreover, the Combiner of the 256 sub-carrier MC-CDMA receiver needs the FFT outputs in normal order. It means that the reconfigurable FFT processors *FFT-I* and *FFT-II* must have an additional ordering stage (*ORD*) to convert the digit reversed FFT outputs order into normal order. The architecture of the reconfigurable FFT processor employed in a MC-CDMA receiver is shown in Figure 8.7. Both *FFT-I* and *FFT-II* have the same basic architecture. The difference again lies in the extent of clock gating. The architecture of *FFT-I* is similar to *RFFT-I* in which the clock gating is applied to all the unused modules of the stage 1 commutator including its dual port RAM. The architecture of *FFT-II* is similar to *RFFT-II* with clock gating confined to the registers and FSM of stage 1. An ordering block (*ORD*), shown in Figure 8.8, is also needed in the reconfigurable pipelined FFT processor to restore the FFT output into normal order for carrying out the combining operation for a 256 sub-carrier receiver. The ordering block comprises of two RAMs namely, RAM0 and RAM1 each having 256 locations to store the 256 words of the FFT. The two RAMs are needed for

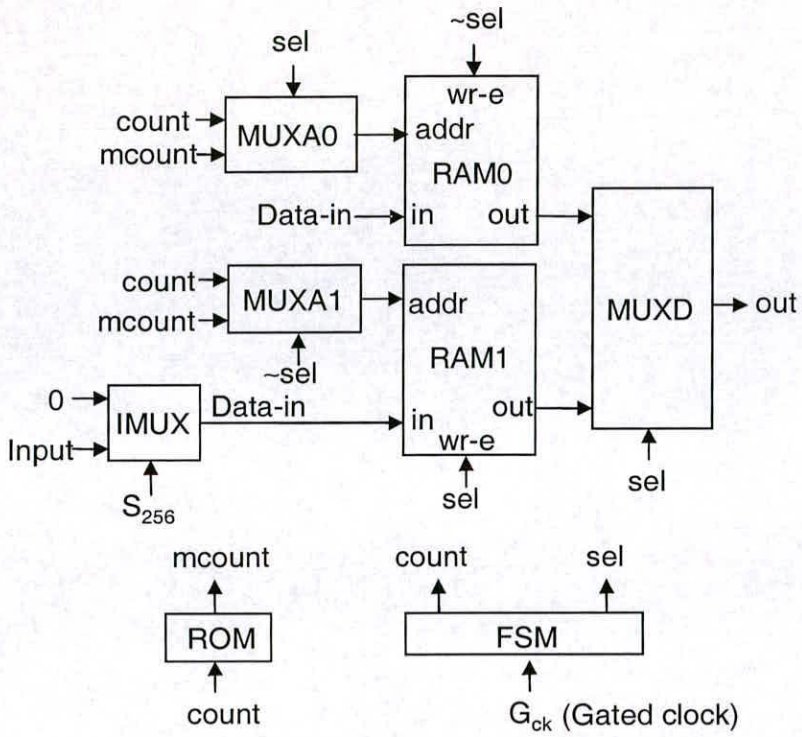


Figure 8.8: *Architecture of the reordering stage in a pipelined FFT processor.*

uninterrupted operation of the pipelined receiver. The first RAM stores the digit reversed output of the FFT whereas the second RAM outputs the previous block of the FFT output in normal order to the Combiner for further processing. The roles of the two RAMs will be reversed after each 256 clock cycles for real time processing. The address pointer to the RAM, which stores the FFT output, is generated by a ROM based on the actual location of the digit reversed data whereas the address pointer to the RAM outputting data to the Combiner is generated by a counter. Two multiplexers namely MUXA0 and MUXA1 are used to select the appropriate address for RAM0 and RAM1 respectively. A multiplexer (MUXD) is used to select the RAM outputs. A FSM controls the ordering operation by flipping between the RAMs for reading and writing operations. It controls the select lines of the multiplexers and also has a counter for generating the address.

This ordering block is partially disabled in case of a 64 sub-carrier receiver by keeping its input fixed with the help of a multiplexer *IMUX* and also by disabling the clock of its FSM in both *FFT-I* and *FFT-II* architectures. This is possible because the order of the 64-outputs of the FFT is immaterial for the Combiner in a 64 sub-carrier receiver assumed over here. The output multiplexer *OMUX* selects either the output of the *ORD* block or the digit reversed FFT output

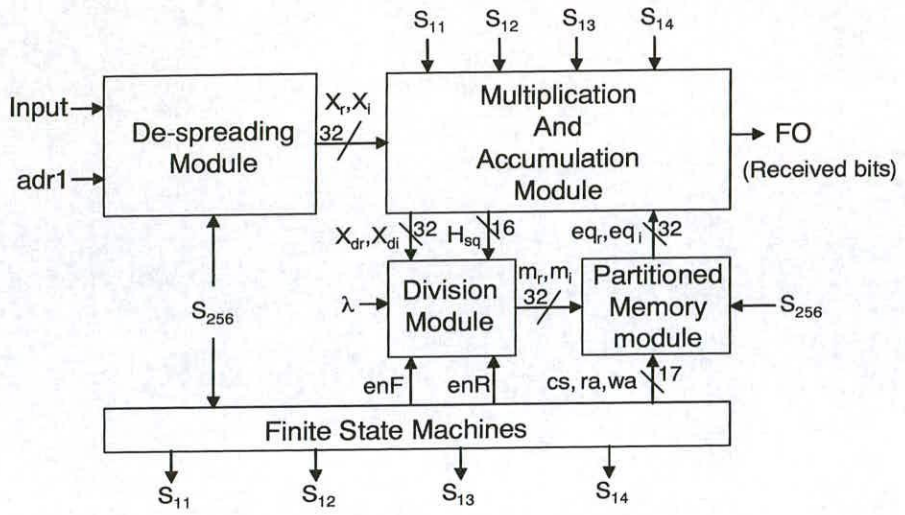


Figure 8.9: *Architecture of the reconfigurable Combiner.*

depending upon the number of sub-carriers. The *ORD* block is bypassed for the 64 sub-carrier system using *OMUX*.

8.4.2 Reconfigurable Combiner architecture

The designed reconfigurable Combiner can handle both 256 and 64 sub-carriers. The basic architecture of the reconfigurable Combiner is quite similar to the dedicated Combiner architecture discussed in the previous chapter and is shown in Figure 8.9. It consists of a despread module, a multiplication and an accumulation module, a partitioned memory module to store the equaliser coefficients, a divider module used in the channel estimation phase and finite state machines module for controlling the reconfigurable Combiner. The memory module and the finite state machines module are different from the 64 sub-carrier Combiner discussed in the previous chapter. The frame for the 256 sub-carrier system is shown in Figure 8.10. It is assumed to comprise of 32 symbols. The code length for this system is also assumed to be 64 and hence four bits are accommodated in one symbol. The first symbol is reserved for the pilot. The remaining 31 symbols represent data bits. The 256 sub-carrier Combiner also has two phases namely the channel estimation phase and the data demodulation phase. In the channel estimation phase, all the 256 equaliser coefficients corresponding to every sub-carrier are computed and stored in a 256 word RAM. The data demodulation phase estimates the transmitted bit by extracting the real part of the product of received data symbols and the equaliser coefficients followed by its accumulation over the code length of 64 corresponding to each transmitted bit. The accumulator register has to be cleared after 64 accumulations to initiate a fresh demodula-

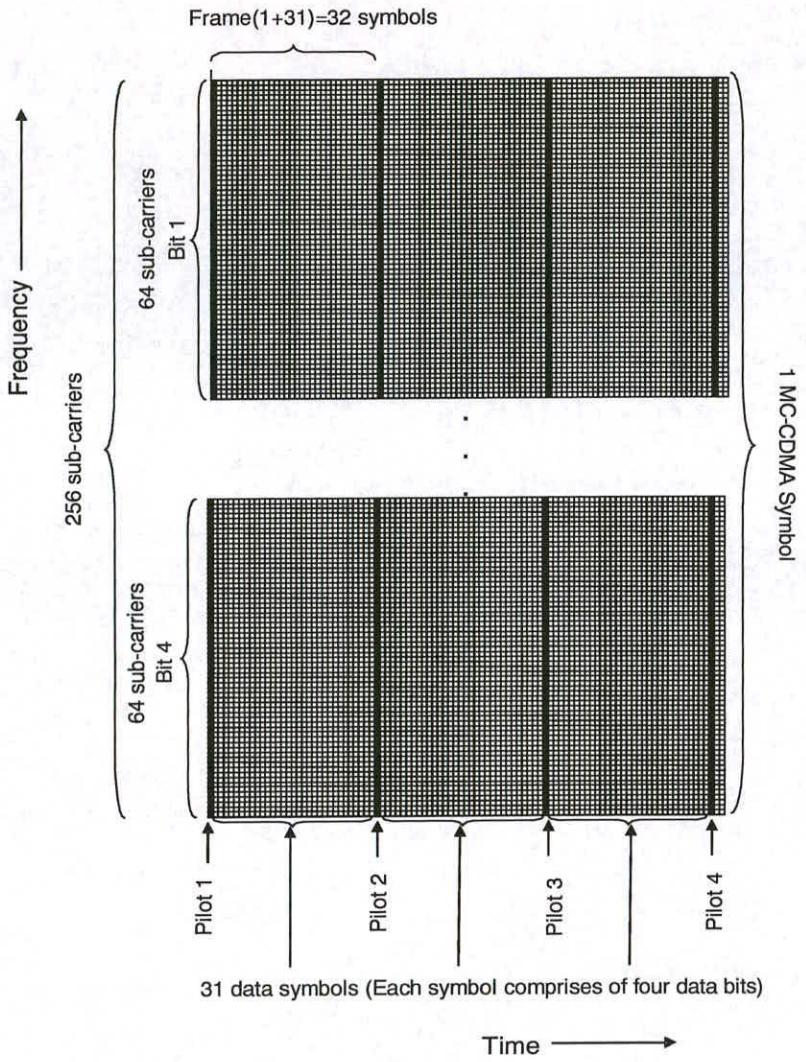


Figure 8.10: *MC-CDMA frame assumed for 256/64 sub-carriers.*

tion phase for the next data bit. This demodulation phase continues till all the 124 data bits in the 31 symbols are recovered by the Combiner. The next frame then commences again with a fresh channel estimation phase.

The finite state machines module of the reconfigurable Combiner is shown in Figure 8.11. It consists of two FSMs and a multiplexer. The FSMs namely FSM_{256} and FSM_{64} are needed to realise a reconfigurable Combiner architecture that can handle both 256 and 64 sub-carriers respectively. A multiplexer (MUX23) is used to select the appropriate FSM depending upon the number of sub-carriers as per the S_{256} control signal.

Two reconfigurable Combiner architectures namely *COMBINER-I* and *COMBINER-II* are proposed for *RECEIVER-I* and *RECEIVER-II* respectively. The reconfigurable Combiner memory

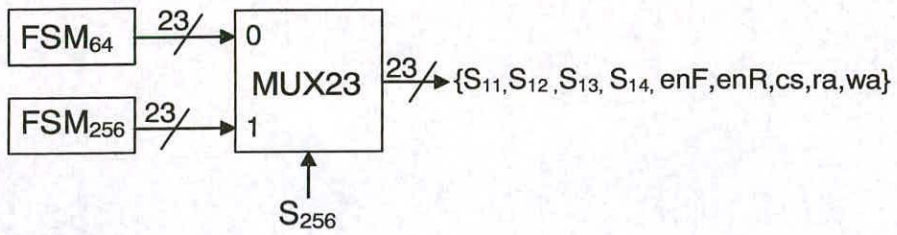


Figure 8.11: *Finite state machines module for the reconfigurable receiver.*

module has a partitioned memory such that it can operate both as a 64 and 256 words RAM. Only 64 words corresponding to 64 equaliser coefficients are needed for a 64 sub-carrier system contrary to 256 in a 256 sub-carrier system. The 192 words of RAM and FSM_{256} are disabled by clock gating for the 64 sub-carrier system in *COMBINER-I* whereas only FSM_{256} is disabled through clock gating in *COMBINER-II*. It means that the *COMBINER-II* employs limited clock gating unlike *COMBINER-I*.

It is clear that the power saving occurs both in the FFT and the Combiner in switching from 256 to 64 sub-carriers. The first FFT stage can be disabled partially or fully to significantly reduce its power consumption in switching from 256 to 64 sub-carriers receiver because this stage is not required in a 64 sub-carrier receiver. Similarly, the ordering stage can be partially disabled by keeping its inputs fixed for 64 sub-carriers. Moreover, the 192 words RAM and/or FSM_{256} can also be disabled in the Combiner of a 64 sub-carrier receiver thereby saving considerable power over a fixed 256 sub-carrier receiver.

8.5 Results

The reconfigurable 256 sub-carrier MC-CDMA receiver cores, namely *RECEIVER-I* and *RECEIVER-II*, the fixed 256 sub-carrier MC-CDMA receiver core, the 256-point reconfigurable FFT processor cores, namely *RFFT-I* and *RFFT-II* and the fixed 256-point FFT processor core have been designed at the register transfer level (RTL) using Verilog hardware description language. The cores were synthesized using SYNOPSYS *DesignCompiler* with UMC 0.18μ standard cell CMOS library. Layouts of the cores were generated using Envisa Silicon Ensemble place-and-route software. This was followed by extracting RC information and then performing RC back-annotated post layout gate-level netlist simulations for 4000 uniformly distributed random input data samples. The resulting switching activity of the circuit nets was then used by the SYNOPSYS *DesignPower* to compute the power consumption for different cores. All the

FFT type	FFT size in points	Power (mW)	% Power saving
Fixed	256	123.01	-
Reconfigurable RFFT-I	256	154.62	-26
	64	55.03	55
	16	18.69	85
Reconfigurable RFFT-II	256	124.31	-1
	64	91.48	26
	16	70.44	43

Table 8.1: *Power comparison between the fixed and the reconfigurable FFT processor architectures.*

simulations were carried out at a supply voltage of 1.8V and at a clock frequency of 25MHz. The input data was obtained by modelling the transmitter, receiver and the channel in Matlab for 32 users and signal-to-noise ratio equal to 40. The receiver hardware has been verified for other values of the number of users and signal to noise ratios with insignificant change in the power results. The input data wordlength is assumed to be 16-bits which is typically used in wireless LAN applications. The results are listed in Tables 8.1 to 8.3. Table 8.1 lists the power saving of reconfigurable FFTs *RFFT-I* and *RFFT-II* as compared to a fixed 256-point FFT. The power overhead is 26% for the 256-point reconfigurable architecture *RFFT-I* as compared to a fixed 256-point FFT processor. This power overhead is primarily due to the clock gating of the dual port RAMs of the commutators in both stage 1 and stage 2 of the reconfigurable FFT processor. This extensive clock gating in *RFFT-I* gives power savings of 55% and 85% in switching to 64-point or 16-point respectively as compared to a fixed 256-point FFT processor. The *RFFT-I* architecture is ideally suited to channel conditions where the required FFT size is mostly less than the largest 256-point.

The second reconfigurable architecture *RFFT-II* introduces a power overhead of just 1% as compared to the fixed 256-point FFT processor due to selective clock gating. The clock gating is applied only to the registers and FSM of stage 1 and stage 2 but not to the dual port RAM based FIFOs. This means that the power overhead in the clock gates used for gating the large memory is absent. The power savings of 26% and 43% are obtained in switching to 64-point and 16-point FFT respectively as compared to a fixed 256-point FFT. The power saving in going down to shorter FFT is much less for *RFFT-II* in comparison with *RFFT-I* due to selective clock gating. Hence, *RFFT-II* architecture is more suitable for channel conditions where the probability of the maximum FFT size (256) is high.

Data set	Time division	RFFT-I Aggregate Energy contribution (mJ)	RFFT-II Aggregate Energy contribution (mJ)	% Energy saving RFFT-I	% Energy saving RFFT-II
1	80% as 256	0.8*154.62=123.7	0.8*124.31=99.45		
	10% as 64	0.1*55.03=5.50	0.1*91.48=9.15		
	10% as 16	0.1*18.69=1.87	0.1*70.44=7.04		
	Total Energy	131.07	115.64	-6.6	+6
2	10% as 256	0.1*154.62=1.55	0.1*124.31=12.43		
	10% as 64	0.1*55.03=5.50	0.1*91.48=9.15		
	80% as 16	0.8*18.69=14.95	0.8*70.44=56.35		
	Total Energy	22	77.93	+82	+37
3	50% as 256	0.5*154.62=77.31	0.5*124.31=62.16		
	30% as 64	0.3*55.03=16.51	0.3*91.48=27.44		
	20% as 16	0.2*18.69=3.74	0.2*70.44=14.09		
	Total Energy	97.56	103.69	+21	+16

Table 8.2: *Aggregate Energy saving of RFFT-I and RFFT-II architectures with respect to a fixed 256-point FFT processor for three data sets corresponding to three different FFT size requirements over three different time durations.*

Table 8.2 lists the aggregate Energy saving of *RFFT-I* and *RFFT-II* architectures with respect to a fixed 256-point FFT processor for three different FFT size requirements over three different time durations. An overall time slot of unity is assumed and the percentage refers to the fractional time out of unity over which that FFT size is active. It is clear from Table 8.2 that for Data set 1, *RFFT-II* architecture is better than *RFFT-I* architecture. This is primarily due to the longer allocated fractional time slot of 80% and large power overhead of 256-point *RFFT-I* architecture with respect to the fixed 256-point FFT processor. It means that *RFFT-II* is better than *RFFT-I* in situations where the maximum FFT size (256-points) is required most of the time. On the other hand for Data set 2, *RFFT-I* architecture saves 82% energy as compared to only 35% for *RFFT-II*. This is because for Data set 2, the minimum FFT size (16-points) is required most of the time. Even for a relatively longer fractional time slot of 50% for the maximum FFT size corresponding to Data set 3, the energy saving corresponding to *RFFT-I* is more than that for *RFFT-II*. This indicates that for most cases where the largest FFT size is required for 50% or less time, *RFFT-I* architecture gives more energy saving than *RFFT-II*. Any of the two architectures can be chosen depending upon the application's requirements.

It is evident from Table 8.3 that stage 1 of the FFT processor contributes most to the power consumption followed by stage 2, stage 3 and then the last stage. The power consumed by stage 1 commutator is maximum for *RFFT-I* architecture because the clock gating is applied

FFT stage	Major blocks in each stage	Fixed FFT Power (mW)	RFFT-I Power (mW)	RFFT-II Power (mW)
Stage 1	COM_1	62.90	93.38	63.45
	BUT_1	0.94	0.92	0.94
	$MULT_1$	6.11	6.11	6.11
Stage 2	COM_2	18.79	28.05	19.01
	BUT_2	0.91	0.89	0.94
	$MULT_2$	6.15	6.14	6.13
Stage 3	COM_3	5.71	5.78	5.79
	BUT_3	0.94	0.98	0.97
	$MULT_3$	6.19	6.17	6.17
Stage 4	COM_4	1.31	1.30	1.31
	BUT_4	1.50	1.44	1.51
MUX-I	-	-	0.1	0.12
MUX-II	-	-	0.05	0.05

Table 8.3: *Power consumed by the major blocks of the fixed and reconfigurable 256-point FFT processors.*

to all the modules of its commutator. The stage 1 commutator power overhead is negligible for *RFFT-II* as compared to *RFFT-I* with respect to the fixed FFT because clock gating in *RFFT-II* architecture is limited to the FSM of its commutator and is not applied to its dual port RAMs. The same power consumption trend continues for stage 2 as well. Stages 3 and 4 consume almost identical power for all the architectures due to the absence of clock gating in these stages. The additional logic for reconfiguration in the form of two multiplexers namely MUX-I and MUX-II does not consume much power. The real strength of the reconfigurable FFT architecture lies in using the optimum FFT length in real time thereby saving power.

Table 8.4 lists the power consumed by the fixed and the reconfigurable receiver architectures. *RECEIVER-I*, in the 256 sub-carriers mode, consumes maximum power just like *RFFT-I* as compared to both *RECEIVER-II* and the fixed receiver. This is because of extensive clock gating in both the FFT processor and the Combiner. The power overhead of *RECEIVER-I* is 16% as compared to only 0.5% for *RECEIVER-II*. The power overhead of *RECEIVER-II* is much less as compared to *RECEIVER-I* because the clock gating is limited to only the registers and FSMs of the FFT and the Combiner. The power saving in going down to 64 sub-carrier is 47% in *RECEIVER-I* because the power saving occurs in the whole of commutator, equaliser RAM and the ordering block. The power saving in switching to 64 sub-carriers is only 19% for *RECEIVER-II* due to the limited power reduction in the commutator, ordering block and small

Receiver type	Receiver size (sub-carriers)	Power (mW)	% Power saving
fixed (256)	-	243.66	
RECEIVER-I	256	285.85	-16
	64	129.45	+47
RECEIVER-II	256	244.79	-0.5
	64	196.60	+19

Table 8.4: *Power comparison between the fixed and the reconfigurable receiver architectures.*

Major blocks of receiver	Fixed receiver (256)in mW	RECEIVER-I		RECEIVER-II	
		(256) in mW	(64) in mW	(256) in mW	(64) in mW
FFT	173.78	198.75	92.61	175.04	128.21
Combiner	38.43	53.15	16.95	38.56	36.31

Table 8.5: *Power consumed by the major blocks of the fixed and the reconfigurable receivers.*

power saving in the Combiner by disabling only FSM_{256}.

Table 8.5 lists the power consumed by the major blocks of the receiver. It is clear that the FFT block consumes most of the power in a MC-CDMA receiver. The power consumed by the FFT in the receiver has gone up considerably after the inclusion of the ordering block *ORD*.

8.6 Summary

This chapter has presented a novel concept of real time adjustment of the MC-CDMA receiver hardware as per the channel requirements. Two 256 sub-carrier reconfigurable MC-CDMA receiver architectures have been presented which can also be configured for 64 sub-carriers in real time. These architectures can be very easily modified to support any other combination of sub-carriers. The appropriate reconfigurable architecture is chosen depending upon the channel parameters. The power saving in going down to smaller number of sub-carriers has been clearly established. This concept of hardware size adjustment on the basis of changing channel parameters can also be extended to other receiver blocks like the Viterbi decoder. Two reconfigurable pipelined FFT processor architecture are also presented that can be configured as a 64-point or 16-point in real time as per the channel parameters.

This thesis investigated low power techniques and architectures for the FFT and the Combiner

blocks of the MC-CDMA receiver as well as techniques for saving power in the whole receiver by tuning its hardware as per the channel parameters. The FIR filter is also an important block of a wireless receiver. The next chapter presents low power architectures for the direct form FIR filter.

Chapter 9
Low power FIR filter architectures

The Finite impulse response filter is one of the most commonly used block in signal processing and telecommunication systems. It is also used in multicarrier receiver and therefore power consumption is important. The low power FIR filter architectures presented in this chapter are based on the existing low power algorithms. The power saving potential of most of the low power algorithms considered in this chapter has so far been evaluated only up to the multiplier level [11, 12, 17, 18]. It is very important to investigate the power saving potential of all these algorithms on the overall FIR filter in the presence of hardware overhead associated with each of these algorithms. All these power saving algorithms are mostly applicable to single multiply accumulate unit based FIR filter architectures. Therefore, this chapter presents the power saving potential of each of these algorithms and their hybrid on the single multiply accumulate unit based direct form FIR filter architecture.

This chapter is organised into seven sections. The first section gives an overview of the direct form FIR filter. The second section describes a conventional direct form FIR filter architecture. In the third section, the conventional filter architecture is modified to support low power coefficient ordering scheme. The fourth section proposes the low power FIR filter architecture for the coefficient segmentation algorithm. The fifth section introduces a block processing algorithm based FIR filter architecture. The sixth section describes a low power hybrid FIR filter architecture obtained by the combination of coefficient segmentation and block processing algorithms. The chapter concludes with a section on the comparison of power and area for the different low power FIR cores with respect to the conventional FIR core for the three commonly used multiplier types.

9.1 Overview of the direct form FIR filter

A filter is a system that selectively alters the characteristics of a signal in a specified way. The filters are commonly used to remove or minimise noise from a signal. It is also used to

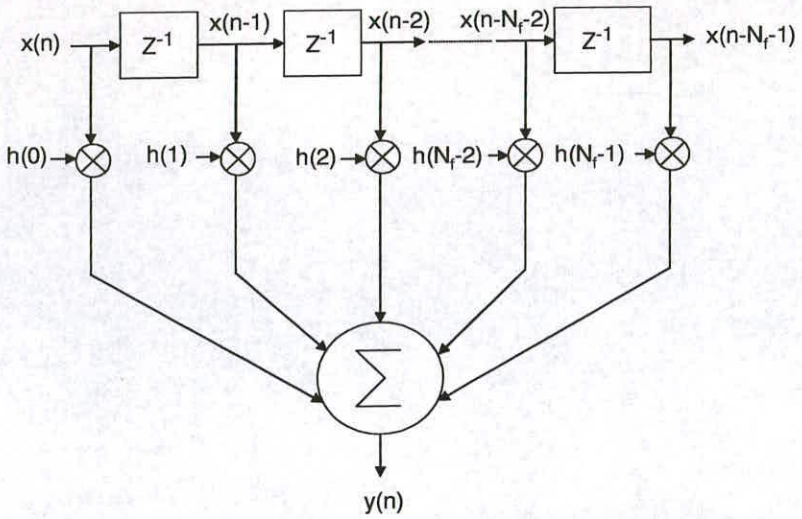


Figure 9.1: *Direct form FIR filter structure.*

separate two or more signals combined together for efficient use of a communication channel or to extract information from signals.

A digital filter is a mathematical algorithm that produces a digital output from a digital input for achieving a filtering objective. It can be implemented in hardware or software depending upon the application.

A Finite impulse response (FIR) filter is represented by the following equation:

$$y(n) = \sum_{k=0}^{N_f-1} h(k)x(n-k) \tag{9.1}$$

Where x(n) and y(n) are the input and output samples of the filter respectively, h(k) is the k^{th} impulse response coefficient and $'N_f'$ is the filter order. Equation 9.1 clearly indicates that the filter's output response to any input x(n) that eventually goes to zero will eventually go to zero. The structure of the Direct form FIR filter (*DF*) is shown in Figure 9.1. It is nonrecursive which means that the present filter output depends only on the present and past inputs and not on the past output values. This characteristic is responsible for the stability and popularity of FIR filters.

The *DF* filter structure consists of a series of delay elements (z^{-1}). The output of each of these delay elements along with the input sample are called the taps of the filter. An N_f-tap filter has $N_f - 1$ delayed samples. The filter order depends upon the number of taps. The filter response tends towards the ideal response by increasing the number of taps. Each data tap is multiplied by

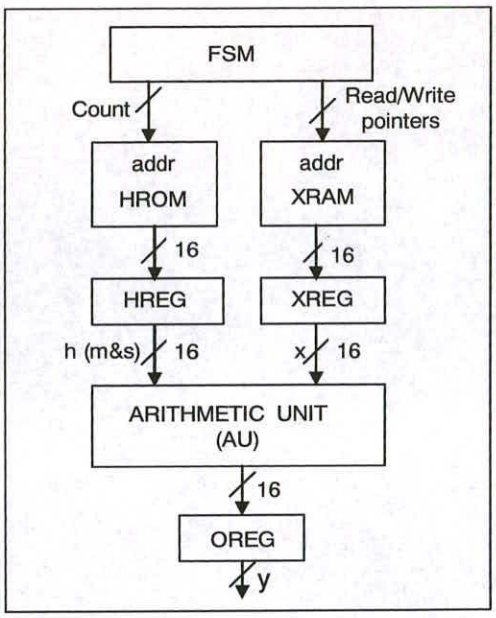


Figure 9.2: *Generic direct form FIR core.*

its respective coefficient and then all the product terms are summed together to generate the final output at that point of time. The delay elements are realised by clocked registers. The output $y(n)$ at any time depends upon the current input $x(n)$ and $N_f - 1$ previous inputs for an N_f-tap filter. The *DF* structure can be realised in a parallel fashion by replacing each multiplication operation by a hardware multiplier, each delay element by a register, and a summer for adding the outputs of all the multipliers. This thesis explores a single multiply-accumulator based low power FIR filter architectures. The filtering operation in this architecture has to be carried out in a sequential fashion for N_f clock cycles to generate a single output. This filter implementation saves area but is much slower than the parallel implementation.

9.2 Conventional *DF* FIR filter architecture

The block diagram of a generic *DF* FIR core is shown in Figure 9.2. It consists of two memory blocks for storing the coefficients (*HROM*) and input data (*XRAM*), two registers for holding the coefficient and input data namely *HREG* and *XREG* respectively, and the *FSM* along with the main arithmetic unit (*AU*). The *XRAM* is realised in the form of a latch based circular buffer for reducing its power consumption. The *FSM* is responsible for applying the appropriate coefficients and input data to *AU*. The *AU* architecture along with that of the *FSM* mainly varies from one algorithm to the other.

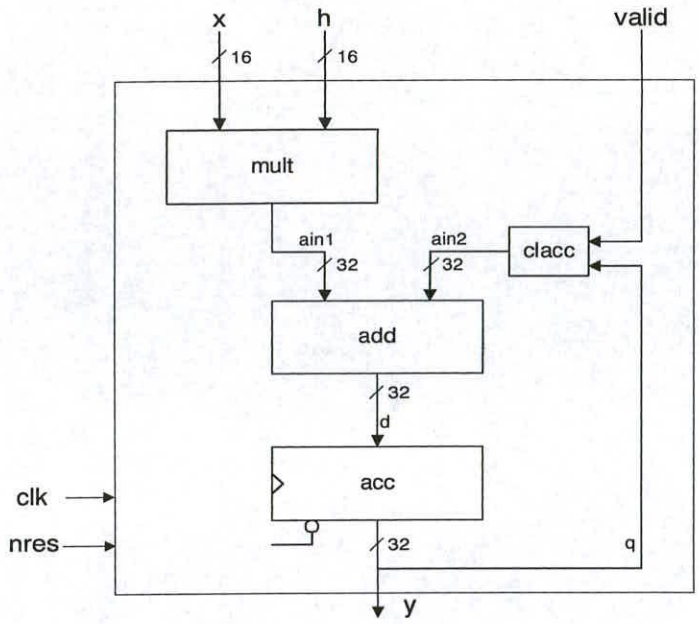


Figure 9.3: *Conventional arithmetic unit.*

In the conventional implementation of an FIR filter, the *AU* consists of a multiplier (*mult*), an adder (*add*), an accumulator (*acc*) and a clearing logic block (*clacc*) in the form of a multiplexer as shown in Figure 9.3. The *clacc* carries out the dual operation of feeding the accumulated values back to the adder and also for clearing the accumulator when the valid input to the *AU* is asserted high. The valid input is asserted high after the generation of a filter output. The *mult* has three different implementations namely a carry-save array type (*csa*), a Non-Booth-coded Wallace tree type (*nbw*) and a Booth-coded Wallace tree type (*wall*). The power evaluation of the filter architectures was carried out for all of the three multipliers. The *AU* is used to multiply a coefficient h(k) with an input data sample x(k) and adding the previous stored accumulator register value to the product at the same time in each clock cycle. Data samples and the filter coefficients are represented as 16-bit two's complement numbers whereas the filter output is of 32-bit.

A conventional implementation of a direct form FIR filter is executed such that at each clock cycle a new data sample, x(k), and the corresponding filter coefficient, h(k), are fetched from *XRAM* and *HROM* simultaneously and stored in the respective registers namely *XREG* and *HREG*. The register outputs are directly connected to the multiplier. Therefore, for each multiplication both inputs of the multiplier receive new data. Due to this continuous change at both the inputs, there will be a high level of switching activity within the multiplier leading to higher power consumption

9.3 Coefficient ordering based FIR filter architecture

The multiplier is a major bottleneck governing the performance of a DSP algorithm. In addition, the power dissipated within the multiplier represents a significant proportion of the overall power dissipated by the DSP device [114]. A reduction in the switching activity within the multiplier block can be achieved by implementing the filter such that respective data samples are multiplied with filter coefficients in a non-conventional order [115]. This order can be obtained by minimising the Hamming distance between those filter coefficients used in successive multiplication operations. It must be noted that the ordering of coefficients is performed only once prior to the commencement of filtering. Subsequent use of the filter will utilise the same order of coefficients. For this reason, coefficient ordering has no implications on the speed of the filtering process.

Filtering commences by fetching a coefficient and the corresponding data sample. These are then presented to the multiplier inputs and the result is added to the accumulator. The rest of the coefficients are processed in a similar manner. Once all the coefficients are processed the filter output is obtained from the accumulator. For the next filter output, the accumulator is cleared, a new data sample is read into the data memory (replacing the oldest data in the memory), and the above steps are repeated.

A block diagram of the ordered FIR core is illustrated in Figure 9.4. It consists of *HROM*, *XRAM*, *AU* and a *FSM* containing an address generation logic (ADDR-GEN) block to support ordering. The *ADDR-GEN* block governs the non-conventional processing of coefficients. It consists of a look-up-table (*LUT*) and an adder. The *XRAM* is realised in the form of a circular buffer for reducing its power consumption. The *XRAM* position to be currently written is tracked by the Write-pointer generated by the *FSM*. The Write-pointer always points towards the most recently entered data value x(0). In a conventional implementation, the access of the first coefficient h(0) in *HROM* and the data sample x(0) in *XRAM* must only be aligned and all other combinations of coefficients and data will automatically fall in place. In an ordered implementation, the coefficients in the *HROM* can be in any order depending upon the ordering algorithm and the original coefficient set. Let us assume that the first *HROM* location stores h(5) instead of h(0) as per the ordered list. In order to generate the Read-pointer or address of the correct data corresponding to h(5), the ADDR-GEN block should be able to generate the address of the corresponding data x(5) which is always located five positions away from x(0) in an *XRAM*. The x(0) position is always pointed at by the Write-pointer. Hence, the first

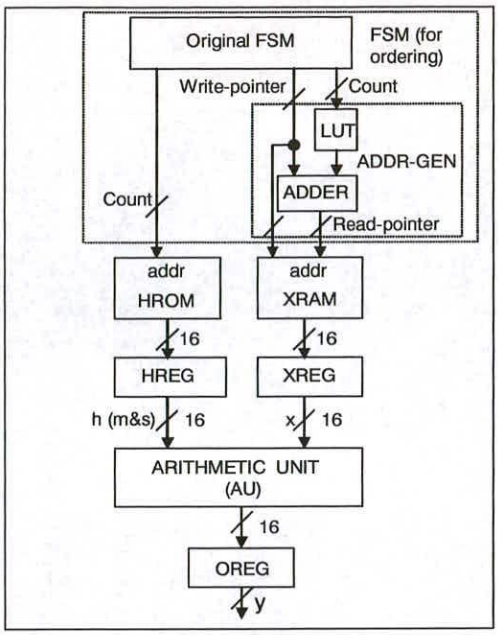


Figure 9.4: *Coefficient ordering based FIR filter architecture.*

offset in the *LUT*, for the Write-pointer corresponding to the first entry of *HROM*, must be 5. The remaining entries of the *LUT* can be obtained by storing appropriate offsets with respect to the Write-pointer's position by examining the corresponding coefficients in the *HROM*. The *HROM* and *LUT* are both addressed by the same counter. The final *XRAM* Read-pointer for a given counter value is obtained by adding the corresponding offset to the Write-pointer. The Write-pointer is always decremented by one after the generation of every output as the *XRAM* receives a new data sample value only after every output generation. This work is reported in [116].

9.4 Coefficient segmentation based FIR filter architecture

According to this technique [17], the fixed coefficients are divided into two components such that one of the components can be realised by using a single shift operation. The second component with a reduced wordlength is applied to the coefficient input of the multiplier. This leads to significant reduction in the switching activity at the coefficient input of the multiplier and hence the power consumption. It is important to note that a shifter consumes much less power as compared to the multiplier and consequently the first component does not consume much power. In order to reduce the switched capacitance at the coefficient input of the multiplier, the segmentation must be done in such a way that the consecutive values of the second component

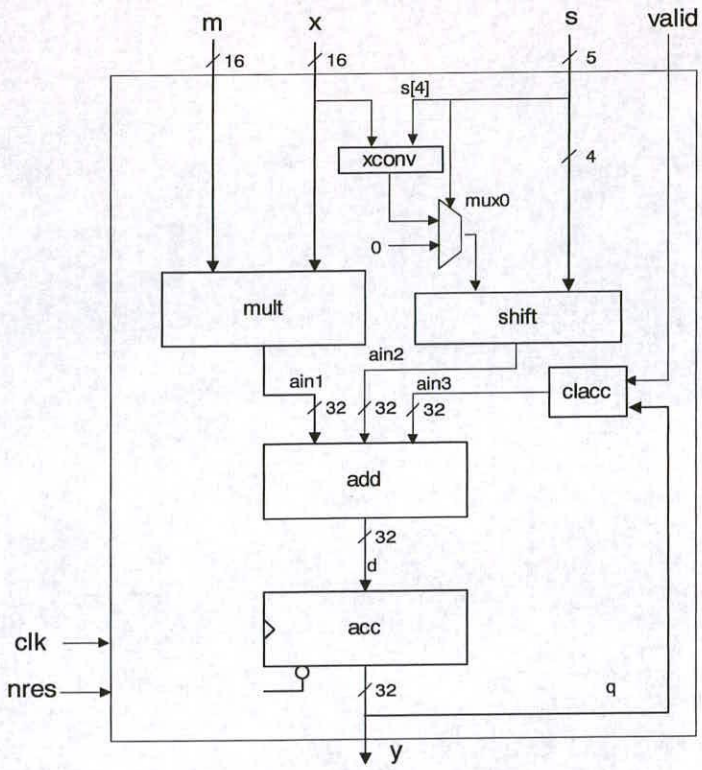


Figure 9.5: *Arithmetic unit for coefficient segmentation.*

should have the same polarity. This approach will also minimise the effective wordlength of the second component which is applied to the multiplier.

Here a 16-bit coefficient h(k) is segmented into two numbers namely a 16-bit decomposed coefficient m(k) and a 5-bit shift value s(k). The MSB of s(k) acts as a sign bit and the remaining four bits are a measure of shift. The input data sample x(k) and the number m(k) are applied to the multiplier while a shift operation of x(k) will be performed according to the shift value s(k). One more 5-bit register has to be included in the main architecture to store the s(k) value. The result of the multiplication and the shifted input data are added to the previous value stored in the accumulator register. The *AU* for the coefficient segmentation algorithm is shown in Figure 9.5. It consists of a multiplier (*mult*), an adder (*add*), a logarithmic shifter (*shift*) implemented using arrays of 2-to-1 multiplexers, a conditional two's complementor (*xconv*), a multiplexer (*mux*) to load and clear the shifter and a clearing block (*clacc*) identical to the one in the conventional FIR filtering block. The MSB of the shift value s(k) determines if a negative shift has to be performed and therefore controls the conversion unit *xconv*. The output of *xconv* is the two's complement of the data only if the MSB of s(k) is one, otherwise the output is equal to the input data. When h(k) is zero (m(k)=0, s(k)=0) or one (m(k)=1, s(k)=0), the shift value

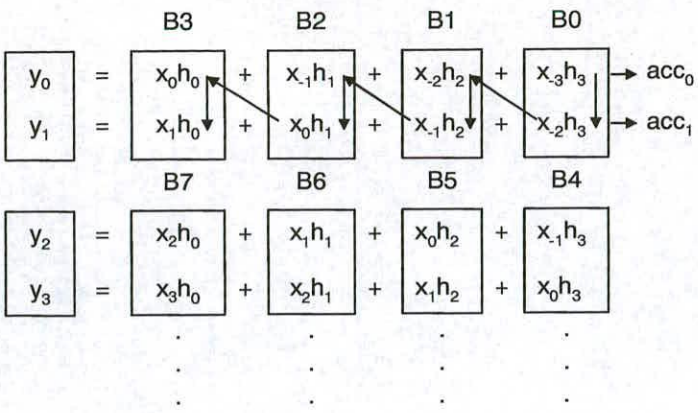


Figure 9.6: *Illustration of block processing with a block size of two.*

will be zero. In these cases, the output of the shifter must be zero as well. In order to guarantee this behaviour, a multiplexer is needed between the conversion unit and the shifter that applies a zero vector when s(k) equals to zero. Since three values (multiplier, shifter and accumulator outputs) are to be added, a single multi-input adder carries out this addition.

9.5 Block processing based FIR filter architecture

In the direct form realisation of the filter a new data sample x(k) and the corresponding coefficient h(k) are multiplied at each clock cycle followed by accumulation in a conventional *AU*. This leads to high switching activity because both inputs of the multiplier receive new data at every clock cycle. Another source of power consumption in DSP's is the activity on data and address buses. Since each time a new data sample is to be multiplied with a new coefficient, both data and address buses experience high switching activity. According to the block processing technique [18], data samples are processed in blocks. By processing multiple data samples at the same time rather than one, it is possible to reduce the switching activity not only at the multiplier inputs but also on the address and data busses, thereby, achieving considerable power saving. As per earlier results [18], blocks of size two yields best results because both inputs to the multiplier are held constant for two clock cycles rather than only one for the block sizes greater than two. The block operation for the block of size two is shown in Figure 9.6. It is clear from this Figure that if one starts the filtering process from the first entry of Block B0, and moves vertically down by one level then the filter coefficient is not changing within this block. Moreover, if one moves diagonally from the second entry in Block B0 to the first entry in Block B1 then the data is not changing as well. This pattern is repeated for all the blocks up to B3.

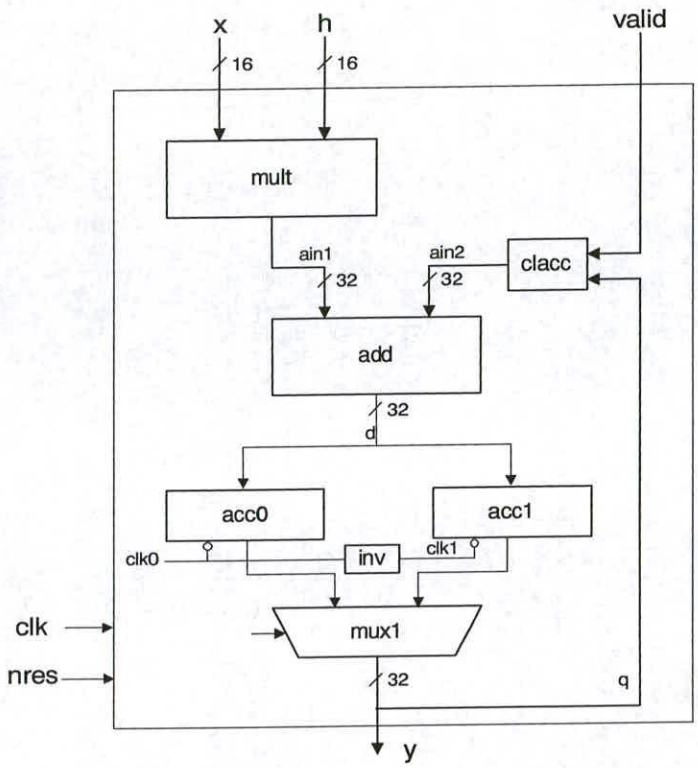


Figure 9.7: *Arithmetic unit for block processing.*

This characteristic is exploited to retain the data and coefficient in filtering block registers for two clock cycles rather than one, thereby, reducing switching activity at the multiplier inputs and address and data busses resulting in power saving.

The *AU*, shown in Figure 9.7, is designed for a block size of two. It consists of a multiplier (mult), an adder (add), two accumulators for storing the results for two outputs namely acc0 and acc1, a multiplexer (mux) to select the proper output of the accumulators to be fed back to the adder and also a clearing logic (*clacc*). The *clacc* initialises the accumulators in response to the active high valid signal which goes up at the time of switching from one block of outputs (y0 and y1) to the next (y2 and y3). The appropriate accumulators are selected by the controls generated by the *FSM*. Once the block processing for the first two outputs (y0 and y1) is over, the *FSM* directs the loading of *HREG* and *XREG* with the new set of coefficient and data values corresponding to the outputs y2 and y3. This sequence has to be repeated for all future outputs. The accumulator acc0 and acc1 are clocked by complementary clocks namely clk0 and clk1 generated by dividing the main clock, clk, by two.

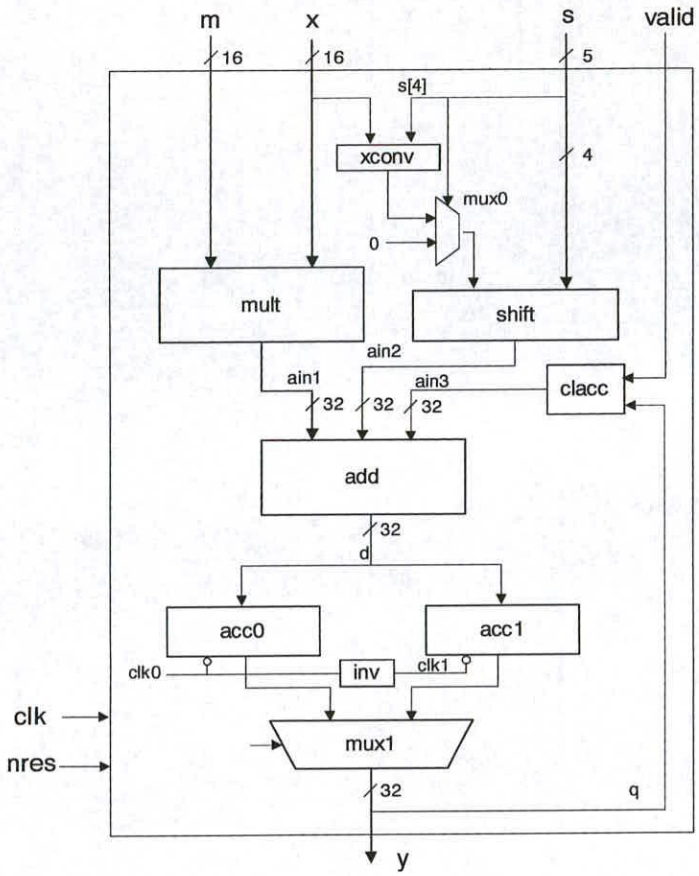


Figure 9.8: *Arithmetic unit for the combination.*

9.6 Combination of block processing and coefficient segmentation based filter architecture

The architectures corresponding to coefficient segmentation and block processing can be combined together to yield even more significant reduction in power with a slight area overhead. The architecture of the *AU* for the combination of block processing and coefficient segmentation is shown in Figure 9.8. It is basically the combination of the architectures described in Figure 9.5 and Figure 9.7. The power saving occurs not only by segmenting the coefficients but also by holding the segmented coefficients and data values at the input of the multiplier for two clock cycles rather than one. The coefficient segmentation, block processing and their hybrid architectures are reported in [117].

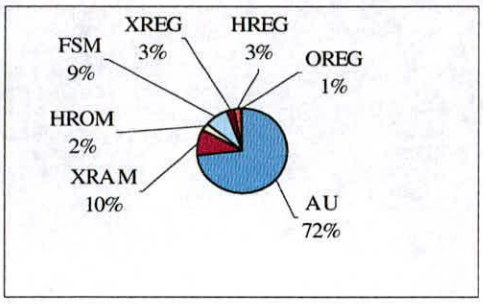
Cell	*CONV* (mW)	*ORDER* (mW)	*CSEG* (mW)	*BP* (mW)	*COMB* (mW)
AU	0.896	0.756	0.631	0.712	0.504
XRAM	0.122	0.141	0.121	0.065	0.065
HROM	0.026	0.020	0.026	0.013	0.012
FSM	0.105	0.159	0.111	0.1	0.108
XREG	0.034	0.035	0.036	0.026	0.027
HREG	0.031	0.023	0.025	0.020	0.018
OREG	0.007	0.007	0.007	0.008	0.008
Total	1.221	1.141	0.957	0.944	0.742

Table 9.1: *FIR Cell power.*

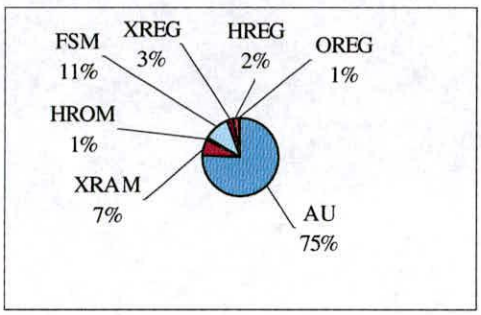
9.7 Results

The different FIR cores for the three algorithms namely coefficient ordering (*ORDER*), coefficient segmentation (*CSEG*), block processing (*BP*), and the combination of CSEG and BP (*COMB*) have been analysed with regard to area usage and power consumption with respect to the conventional architecture (*CONV*). The cores were designed using Verilog HDL and then synthesized using *Ambit BuildGates* targeting the UMC 0.18μ standard cell CMOS library. The requirements for the synthesis were identical for all the cores. This was necessary in order to allow for a consistent power consumption and area usage comparisons. A maximum circuit delay of 35ns has been defined for all the cores. A layout for each core was generated using the Cadence *Silicon Ensemble* place-and-route software. This was followed by extracting RC information and then performing RC back-annotated post-layout gate-level netlist simulations for a uniformly distributed random input data sample set equal to 1000 using *Verilog-XL* simulator. The resulting data including switching activity of the circuit nets and the capacitive load information extracted from the layouts was then used by the Synopsys *DesignPower* tool to compute power consumption figures for the different FIR cores. In all of the above stages, a clock rate of 10MHz and a supply voltage of 1.8 Volts were used. The power profile of the filter has been verified up to 100MHz.

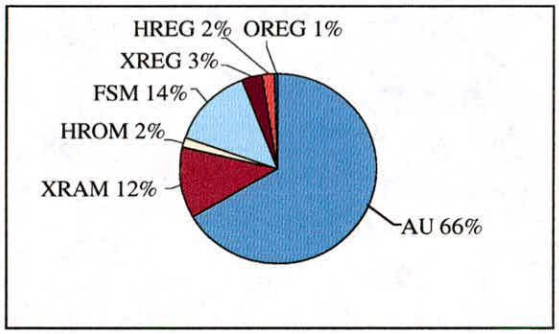
The results are shown in Tables 9.1 to 9.6 and Figures 9.9 and 9.10 for a 73-tap band-pass filter. The power saving in this filter was gauged for three different multiplier types from Synopsys *DesignWare* namely *csa*, *nbw* and *wall*. Table 9.1 and Table 9.2 list the power consumption for the different blocks of the overall filter and the *AU* respectively using a *csa* multiplier type. According to Table 9.1 and Figure 9.9, the power consumption of the *AU* is in the range of



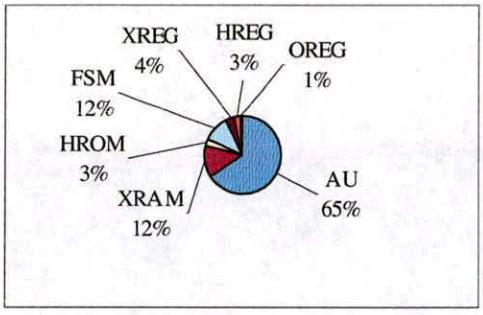
(a) CONV



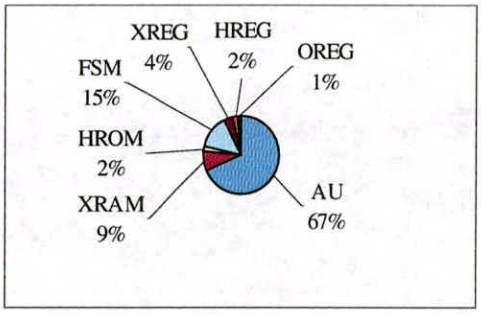
(d) **BP**



(b) ORDER



(c) CSEG



(e) **COMB**

Figure 9.9: *Distribution of cell power for the overall FIR core.*

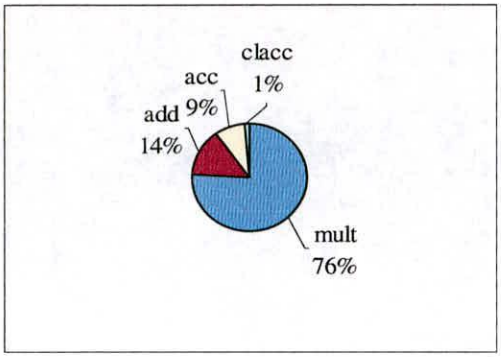
Cell	*CONV* (mW)	*ORDER* (mW)	*CSEG* (mW)	*BP* (mW)	*COMB* (mW)
mult	0.682	0.556	0.331	0.466	0.207
add	0.126	0.117	0.157	0.103	0.127
acc0	0.077	0.072	0.071	0.055	0.051
acc1	-	-	-	0.055	0.051
clacc	0.012	0.011	0.012	0.015	0.016
xconv	-	-	0.013	-	0.008
mux0	-	-	0.006	-	0.004
mux1	-	-	-	0.017	0.018
shift	-	-	0.041	-	0.024
Total	0.896	0.756	0.631	0.712	0.504

Table 9.2: *Arithmetic unit Cell power.*

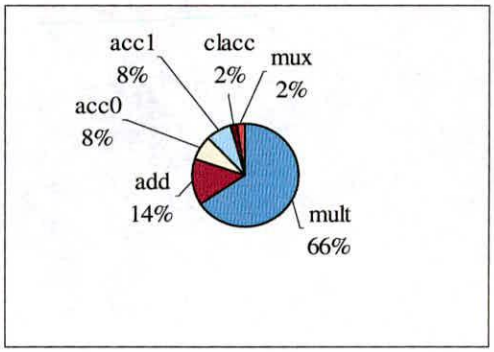
Algorithm	*csa*		*nbw*		*wall*	
	(mW)	%	(mW)	%	(mW)	%
CONV	0.682	-	0.566	-	0.687	-
ORDER	0.556	18	0.425	25	0.373	46
CSEG	0.331	51	0.325	43	0.597	13
BP	0.466	32	0.388	31	0.438	36
COMB	0.207	70	0.227	60	0.349	49

Table 9.3: *Power consumption analysis for different multipliers.*

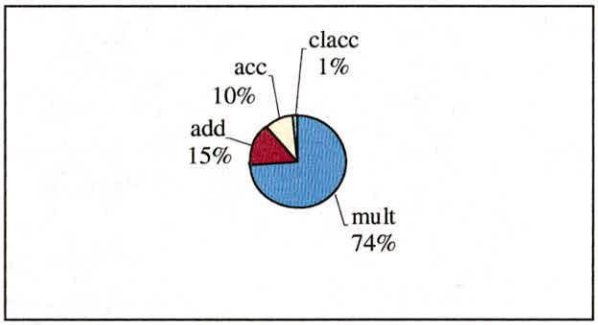
(65-75%) of the overall FIR core power. It is also evident from Table 9.2 and Figure9.10 that the multiplier contributes most to the power consumption (40-76%) in the *AU* and therefore all power saving efforts must be directed to reduce its power consumption. Table 9.3 shows that the percentage saving in power at the multiplier is substantially higher for *CSEG* as compared to *ORDER* and *BP* for the *csa* and *nbw* multiplier types whereas the reverse is true for the *wall* multiplier. The higher savings in power at the multiplier side in *CSEG* is primarily attributed to the significant reduction in the switching activity and effective wordlength of its coefficient input. The *wall* multiplier performs very well for similar *ORDER* and *BP* algorithms where the adjacent coefficient inputs are having less switching activity. The *wall* multiplier saves 46% and 36% power with *ORDER* and *BP* respectively as compared to only 18% and 32% for *csa* and 25% and 31% for *nbw* multipliers. The power saving at the multiplier will obviously be maximum for the *COMB*. The performance of the *csa* multiplier type is best in terms of power reduction for *CSEG* and *COMB* whereas the worst performance is obtained for the *wall* multiplier. The *wall* multiplier proves to be the best at power saving for *ORDER* and *BP*.



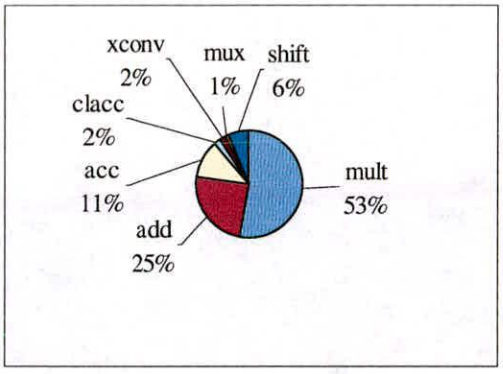
(a) CONV



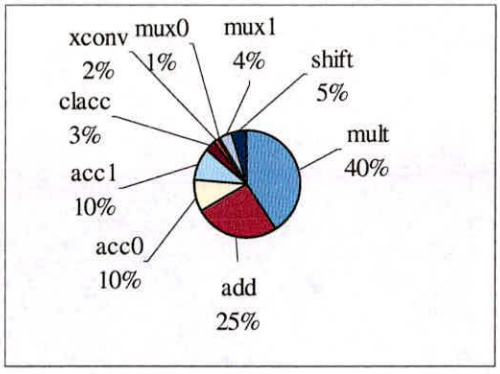
(d) BP



(b) ORDER



(c) CSEG



(e) COMB

Figure 9.10: *Distribution of cell power for the* AU's.

Algorithm	*csa*		*nbw*		*wall*	
	(mW)	%	(mW)	%	(mW)	%
CONV	1.221	-	1.082	-	1.219	-
ORDER	1.141	07	0.987	09	0.942	23
CSEG	0.957	22	0.958	11	1.263	-4
BP	0.944	23	0.844	22	0.898	26
COMB	0.742	39	0.765	29	0.901	26

Table 9.4: *Power consumption analysis for FIR cores.*

Algorithm	*csa*		*nbw*		*wall*	
	$[mm^2]$	%	$[mm^2]$	%	$[mm^2]$	%
CONV	0.184	-	0.186	-	0.186	-
ORDER	0.187	-2	0.189	-2	0.189	-2
CSEG	0.191	-4	0.193	-4	0.194	-4
BP	0.190	-3	0.192	-3	0.192	-3
COMB	0.197	-7	0.199	-7	0.199	-7

Table 9.5: *Area comparison for FIR cores.*

The overall power reduction of the different filter cores is listed in Table 9.4. It is clear from this table that the overall power reduction is more in *BP* as compared to *ORDER* and *CSEG* for all the multiplier types. This is because in *BP* power saving occurs not only in the multiplier but also on the address and data busses for both data and coefficient memories on account of less memory accesses. This is evident from the power consumption of the two memory blocks namely *HROM* and *XRAM* in Table 9.1. Moreover, it is clear from Table 9.5 that the extra hardware introduced to support *CSEG* is around 4% as compared to only 3% for *BP*. This additional hardware in the form of shifter, two's complementor, three-input adder etc. (listed in Table 9.2) is also responsible for the inferior power saving in *CSEG* over *BP*. The overall power consumption of the *wall* based FIR core for *CSEG* has gone up over *CONV* because the power saving in the multiplier is to the tune of just 13%. This is not good enough to offset the power consumed by the additional hardware provided to support the algorithm. The best power saving to the tune of 39% and 29% are obtained for *COMB* with *csa* and *nbw* multiplier types respectively. This is not true for *wall* multiplier type because of the poor performance of *CSEG* with this type of multiplier. The best performance with *wall* multiplier is obtained by *BP* both in terms of power and area. The *ORDER* algorithm also saves 23% power for the *wall* multiplier with an area overhead of just 2%.

Algorithm	*Delay(ns)*		
	csa	*nbw*	*wall*
CONV	10.01	9.02	9.61
ORDER	10.01	9.02	9.61
CSEG	9.94	9.70	9.70
BP	10.03	9.05	9.64
COMB	9.98	9.72	9.72

Table 9.6: *Delay analysis of* AU*'s using different multipliers.*

The area overhead as per Table 9.5 is around 2% for *ORDER*, 3% for *BP* and 4% for *CSEG* for all multiplier types. The results have proved that these algorithms are capable of reducing power consumption in the range of 7-39% with a small increment in area. It is finally inferred from the tables that the best performance is obtained with *COMB* using *csa* and *nbw* multipliers at an expense of 7% increase in area whereas BP gives best result using *wall* multiplier at the expense of only 3% increase in area. The *ORDER* algorithm performs best with the *wall* multiplier saving 23% power at an expense of only 2% in area. Any of these cores can be used for low power depending on the area overhead budget. The results obtained have also been verified with other filter lengths and data samples in addition to the filter considered here.

The delay analysis for *AU*s using different multipliers is given in Table 9.6. It is clear from the table that the difference in delays is negligible for the different algorithms and the multipliers. The critical path for the faster *nbw* and *wall* multipliers runs through the shifter-adder combination rather than the multiplier-adder combination. This is due to the reduction in the delay of the multiplier on account of its smaller size in case of *CSEG* and *COMB* algorithms. Therefore, the critical path for these algorithms remains the same for *nbw* and *wall* multipliers. The critical path for these cases is slightly longer compared to *CONV* or *ORDER* algorithms due to the additional delay introduced by a three-input adder. The critical path in case of a slower *csa* multiplier for *CSEG* and *COMB* algorithms runs through the multiplier-adder combination and hence it is longer than for the other multipliers. The critical path for *CONV*, *ORDER*, *CSEG* and *COMB* algorithms remains almost same for *csa* multiplier because the reduction in the delay of the smaller multiplier is partially compensated by the delay introduced by the three-input adder. In the case of *BP*, the critical path is almost same as compared to *CON* for all the multipliers. The slight increase of critical path in *BP* is attributed to the higher fanout of the adder in the form of two accumulators rather than one. The critical path is the same for *CONV* and *ORDER* for all types of multipliers.

9.8 Summary

This chapter has presented low power direct form FIR filter architectures for the existing low power algorithms. The performance of each of these algorithms has been evaluated only at the multiplier level till now. It is very important to examine the effect of these algorithms on the overall FIR core especially in the presence of hardware overhead. The power reduction capability of the low power algorithms are multiplier dependent and hence the results are presented for the commonly used three multiplier types. The area overhead of the algorithm is also an important measure of its effectiveness. The area overhead for all the algorithms are evaluated. For instance, coefficient segmentation based FIR core provides up to 22% power reduction with an area overhead of 4%. Block processing based core, on the other hand, results in a power reduction of up to 26% at the expense of only 3% in area. Coefficient ordering provides power saving up to 23% with only 2% increase in area. However, if more power saving is desired then the combination of coefficient segmentation and block processing based core can be used providing up to 39% power reduction, but at the cost of 7% increase in area. Any architecture can be considered depending upon the power and area constraints.

This thesis has proposed low power algorithms and architectures for the important blocks of the multi-carrier receiver. It also proposes a concept of power reduction through real time adjustment of receiver hardware on the basis of channel parameters. The next and the last chapter concludes the thesis and describes the roadmap for the future.

Chapter 10
Summary and Conclusions

10.1 Introduction

The aim of this thesis is to investigate low power architectures for the basic blocks of a MC-CDMA and other multicarrier receivers as well as the techniques which can be used for reducing the power of the whole receiver. The key receiver blocks investigated for low power in this thesis are the FFT, Combiner and a direct form FIR filter. The power consumption of the whole receiver is reduced by introducing the concept of an adaptive receiver instead of a fixed receiver designed for the worst case channel conditions.

This chapter is organised into four sections. The first section presents the summary of the work presented in each chapter. The second section provides the conclusions drawn from the results obtained in each chapter. The third section summarises the achievement and the last section outlines areas for future investigation.

10.2 Summary

The power consumption in portable devices has to be reduced to extend the battery life. Multicarrier mode of transmission and reception has become very popular in the wireless world. MC-CDMA has significant potential to be included as a standard in future generations of mobile communications and so power consumption is important. The theme of this thesis has been to investigate low power architectures for the individual blocks of a MC-CDMA receiver as well as for the whole receiver. The receiver blocks namely the FFT, Combiner and FIR filter have been investigated for low power. Three different low power techniques namely, the coefficient addressing scheme, the coefficient memory reduction scheme and the order based processing scheme have been proposed for the most power consuming FFT block of the receiver. A low power radix-2 single butterfly and a pipelined radix-4 FFT processor architectures have also been proposed based on the order based processing scheme. The radix-4 pipelined FFT pro-

cessor has been combined with the low power combiner to realise a low power MC-CDMA receiver.

This thesis has presented low power FIR filter architectures for the existing low power algorithms. Most of these algorithms have been tested only up to the multiplier level. It is important to test the effect of each of these algorithms on the power of the overall filter especially in the presence of hardware overhead. Any proposed low power filter architecture can be chosen depending upon the area/power budget.

This thesis has also introduced a novel concept of adjusting the receiver hardware dynamically as per the changing channel parameters like the delay spread, transmission rate, signal to noise ratio and the Doppler frequency. This adaptive receiver consumes much less power as compared to a receiver designed for the worst case channel conditions. Two 256 sub-carrier reconfigurable receiver architectures have been proposed that can also be configured for 64 sub-carriers instead of using a fixed 256 sub-carrier receiver.

Chapter 2 provided a description of general and block specific low power techniques for reducing the switched capacitance and therefore the power consumption. Most of the techniques exploited signal correlations, hardware size reduction through some transformations and by shutting down unused portion of the architecture in real time. The previous work on MC-CDMA receiver architecture involved processing symbols in block. This technique was suitable only for the combiner but not for the FFT due to the presence of memory overhead. This memory overhead was needed to store the whole block of symbols rather than one symbol after every FFT stage.

The Discrete Fourier transform (DFT) and the FFT for efficiently computing the DFT were introduced in Chapter 3. The radix-2 decimation-in-time FFT algorithm was covered in detail whereas the overview of the radix-4 decimation-in-time algorithm was given. The comparison of radix-4 and radix-2 FFT algorithms in terms of butterfly complexity and number of stages was also discussed.

Chapter 4 presented three schemes for reducing the power consumption in FFT processors namely the order based processing scheme, the coefficient memory reduction scheme and the coefficient addressing scheme. The order based processing scheme was based on the novel concept of using either the real part of the coefficient or its two's complement as per the Hamming distance with the preceding real part. A flag bit was stored along with the coefficients to

indicate the form of the real part of the coefficient. The coefficient memory reduction scheme exploited the relationship among the contents of the coefficient memory to reduce the memory size from N/4 to N/8+1 locations. This resulted in both area and power saving for long FFTs. The coefficient addressing scheme for single butterfly FFT processor was realised using a simple multiplexer instead of a cascade of Barrel shifters. This saved both area and power for all FFT lengths.

The architecture of radix-2 single butterfly FFT processor was presented in Chapter 5. This low power architecture was based on the order based processing scheme. The order based processing scheme was only applied to the last stage of the FFT processor where the switching activity between successive coefficients was the highest. The hardware overhead was in the form of a ROM, a multiplexer and a set of Ex-OR gates.

Chapter 6 presented architectures for the 16-point and 64-point low power radix-4 pipelined FFT processors. The power consumption of the low power radix-4 FFT processor architecture, proposed by Bi and Jones, was further reduced by incorporating order based processing scheme to its penultimate stage. The coefficient ordering required corresponding data ordering. A novel triple port RAM based commutator architecture was proposed for the first stage of a 16-point FFT processor to handle data ordering. This commutator architecture saved power compared to the traditional dual port RAM architecture due to the use of only three triple port RAMs instead of six dual port RAMs.

Chapter 7 presented architectures for a 64 sub-carrier MC-CDMA receiver. This architecture was obtained by combining the low power ordered 64-point FFT processor with the Combiner. The divider modules were needed only in the channel estimation phase and not in the data demodulation phase. Therefore, the divider modules could be disabled in the data demodulation phase by keeping their inputs fixed during this phase.

Chapter 8 introduced a novel concept of designing an adaptive receiver which could be configured as per the channel parameters like the delay spread, signal to noise ratio, transmission rate and Doppler frequency instead of a fixed receiver designed for the worst case channel conditions. Two 256-point reconfigurable FFT processor architectures have been proposed depending upon the extent of clock gating which could also be configured as a 64-point or a 16-point as per the channel requirements in real time. The power saving in going down to 64-point or 16-point as compared to the fixed 256-point FFT processor was 55% and 85% for

RFFT-I and 26% and 43% for *RFFT-II* respectively. The power overhead of 256-point *RFFT-I* and 256-point *RFFT-II* with respect to fixed 256-point FFT was 26% and only 1% respectively. Two reconfigurable 256 sub-carrier MC-CDMA receiver architectures namely, *RECEIVER-I* and *RECEIVER-II* have also been proposed depending upon the extent of clock gating in their modules. These architectures could also be configured for 64 sub-carriers. This approach led to power saving of 47% and 19% for *RECEIVER-I* and *RECEIVER-II* respectively as compared to a fixed 256 sub-carrier receiver designed for the worst case channel conditions.

Various direct form low power FIR filter architectures were proposed in Chapter 9. These architectures were based on the existing low power algorithms like coefficient ordering, block processing, coefficient segmentation and their combination. Most of the low power algorithms were only tested at the multiplier level. It was important to evaluate the effect of these algorithms on the power consumption of the overall filter in the presence of associated hardware overhead. Each of the above mentioned algorithms was evaluated in terms of area and power with respect to a conventional 73-tap FIR filter.

10.3 Conclusions

This thesis proposed novel low power techniques and architectures for the various blocks of the MC-CDMA receiver along with the receiver itself. It can be concluded from Chapter 4 that the order based processing scheme resulted in switching activity reduction of close to 50% as compared to only 20% by the traditional ordering scheme based on Hamming distance for all FFT lengths. This significant reduction in switching activity was responsible for the power savings obtained in both radix-2 single butterfly and radix-4 pipelined FFT processors.

It can be concluded from Chapter 4 that the memory reduction scheme was more effective both in terms of area and power saving for long FFTs (1024-points onwards). The savings in area ranged from 48% to 54% whereas the power savings varied from 0.62% to 59% for long FFT's. It has been found that the scheme was most effective for an intermediate wordlength of 12-bits rather than 8 or 16-bits. This scheme was more suitable for OFDM applications requiring long FFTs. The scheme would lead to the design of more power and area efficient long length FFT processors.

It has been shown in Chapter 4 that the proposed coefficient addressing scheme maintained an improvement of more than 80% over the existing addressing schemes both in terms of area and

power for almost all FFT lengths. The scheme would lead to the design of faster and more power and area efficient systems with embedded radix-2 single butterfly FFT processors.

The results in Chapter 5 concluded that the percentage power saving continued to reduce for longer length FFT's because *Our* order based processing scheme was directed only at reducing power by lowering the switching activity at the coefficient input of the multipliers in the butterfly structure. The butterfly complexity in an FFT remains fixed with FFT length whereas the RAM size goes on increasing. This means that the power consumed in the butterfly increases slightly with FFT length as compared to the power consumed in the RAM blocks. This resulted in lowering of the percentage savings in power for longer FFT's. The *Others* ordering scheme led to no power savings in most cases because the switching activity reduction was much less compared to the hardware overhead required to support that scheme. This was not true for *Our* scheme due to the significant reduction in the switching activity of the ordered coefficient set. The power saving of *Our* order based processing scheme ranged from 1% to 25% for 512-point to 16-point single butterfly FFT processors respectively over the *Others* approaches. The power saving results would improve considerably using customised dual port RAMs rather than the SYNOPSYS *DesignWare* RAMs. The customised RAM consumed very less power than the *DesignWare* RAM. Moreover, more power saving would be obtained for shorter wordlengths due to proportionally less power consumed in the dual port RAMs as compared to the butterfly in radix-2 single butterfly FFT processor architecture.

It was experimentally found that the order based processing scheme gave best power results when applied to the last stage (last stage has high switching activity because of different coefficients) of the radix-2 FFT processor. It has been found that the saving in switching activity was lesser for the inner FFT stages as compared to the hardware overhead needed to support the scheme in these stages. This was responsible for the inferior power performance of FFT with ordering incorporated into the inner stages. Hence, the ordering was limited only to the last stage of the FFT signal flowgraph.

It can be concluded from Chapter 6 that the order based processing scheme was effective only in the penultimate stage of the FFT processor due to the need of corresponding data ordering followed by restoration to the normal data order for the subsequent stage. This approach was not feasible for the higher stages due to the complexity of the commutator and also due to the need of data order restoration hardware overhead for the subsequent stage. This FFT architecture was attractive for wireless LAN applications requiring short FFTs. It could also be used in

the penultimate stages of long FFTs. The switching activity of the successive coefficients has been reduced by 54% in the penultimate stage of the pipelined FFT processor after order based processing. The power saving was in the range of 4-30% for the 64-point FFT and 14-37% for the 16-point FFT for the ordered FFT processor depending upon the multiplier type and FIFO implementation.

From the results of Chapter 6, it can be concluded that the dual port RAM (ADM) in the last commutator stage of the ordered pipelined FFT processor was the hardware overhead needed for incorporating order based processing. The power consumed by this hardware overhead would be reduced significantly by using a customised dual port RAM instead of a SYNOPSYS *DesignWare* RAM. Moreover, the power consumption in the hardware overhead would be considerably lower for shorter wordlength. This means that the power saving percentage would significantly increase by using a customised dual port RAM, and for wordlengths shorter than 16.

Chapter 7 presented a 64 sub-carrier MC-CDMA receiver architecture. The power saving was obtained both in the FFT and the Combiner. The power saving in the Combiner was significant by keeping the inputs to the divider fixed during the long data demodulation phase.

It can be concluded from Chapter 8 that significant power saving could be obtained by using an adaptive receiver tuned to the changing channel parameters instead of using a fixed receiver designed for the worst case channel conditions. This work has presented a novel concept of real time adjustment of the FFT size in a wireless receiver as per the channel requirements. Two reconfigurable 256-point FFT processor architectures namely *RFFT-I* and *RFFT-II* have been proposed which could be configured for 64-point or 16-point as well. The difference between *RFFT-I* and *RFFT-II* lied in the extent of clock gating. The clock gating was applied to all the modules of stage 1 and stage 2 in case of *RFFT-I* whereas it was limited to FSM and registers in case of *RFFT-II*. The selective clock gating helped in reducing the power overhead of *RFFT-II* to just 1% as compared to 26% for *RFFT-I* with respect to the fixed 256-point FFT processor. On the other hand, the power saving in going down to smaller FFT size was much less for *RFFT-II* due to limited clock gating. *RFFT-I* architecture is recommended over *RFFT-II* when the probability of the largest FFT size (256 here) is less than 50% due to the power overhead. Two reconfigurable 256 sub-carrier receivers namely, *RECEIVER-I* and *RECEIVER-II* have also been proposed depending upon the extent of clock gating. These receivers could also be configured for 64 sub-carriers. The power saving results for receiver were similar to

the reconfigurable FFT processor. The power saving, in going down to smaller number of sub-carriers, has been clearly established. This flexible architecture is ideal under variable channel conditions.

In Chapter 9, novel FIR filter cores have been developed based on a number of low power algorithms. The algorithms could be used alone or in combination depending upon the power and area constraints. For instance, coefficient segmentation based FIR core provided up to 22% power reduction with an area overhead of 4%. Block processing based core, on the other hand, resulted in a power reduction of up to 26% at an expense of only 3% in area. Order based processing led to 23% power reduction at an expense of 2% in silicon area. However, if more power saving was desired then the combination of coefficient segmentation and block processing based core could be used providing up to 39% power reduction, but at the cost of 7% increase in area. Any algorithms could be chosen depending upon the area/power budget.

10.4 Achievements

- A novel order based processing scheme is proposed which is much better than the conventional coefficient ordering scheme.
- A low power radix-2 single butterfly FFT processor architecture is proposed based on the order based processing scheme.
- A low power radix-4 pipelined FFT processor arechitecture is proposed based on the order based processing scheme.
- A low power 64 sub-carrier MC-CDMA receiver architecture is proposed.
- A novel concept of adaptive receiver is introduced. According to this concept, the hardware size of the receiver is dynamically adjusted as per the channel requirements instead of designing it for the worst case channel conditions.
- A novel coefficient memory reduction scheme is proposed that reduces the memory size to N/8+1 locations instead of N/4 locations in an N-point FFT processor.
- A novel coefficient addressing scheme for generating the coefficient address using only one multiplexer instead of two barrel shifters in cascade is also proposed for an N-point FFT processor.

- Various novel low power FIR cores for the existing low power algorithms are proposed.

10.5 Future work

This thesis has tried to provide a thorough investigation into the research proposal outlined in section 10.1. However, a number of additional issues need to be explored which might further add to the knowledge gained from the research presented. The additional issues are highlighted as follows:

- In all FFT processors, a fixed point 2's complement number representation is employed. The dynamic range can be extended by using block floating point representation.
- The wordlength is assumed to be fixed for all stages of the pipelined FFT processors. It is important to use shortest wordlength in the most complex highest or first FFT stage and gradually increase it for the lower stages for reducing power consumption on the basis of the desired SNR.
- The architecture of a MC-CDMA receiver contains only the basic blocks in the form of FFT and combiner for BPSK reception. Some additional blocks like de-interleaver, signal mapper and Viterbi decoder can be added to support more complex modulating schemes for better performance in terms of bit error rates etc.
- Some more aspects like hardware for synchronisation has to be explored in the MC-CDMA receiver in the presence of frequency offset.
- A scheme for altering the receiver complexity in real time needs to be formulated. This scheme should clearly establish the relationship between the receiver hardware size and the channel conditions for all the important blocks of the receiver. Channel monitoring hardware will have to be designed as well for real time adaptive receiver. Moreover, the frequency of hardware size adjustment on the basis of channel conditions has to be studied for minimising power consumption.
- This concept of hardware size adjustment on the basis of changing channel parameters can be extended to other receiver blocks like the Viterbi decoder. The ultimate objective is the realisation of a complete adaptive receiver.

- Some of the cores are realised in 0.35μ technology and implemented up to the gate level. It will be better to realise all the cores up to the layout level in 0.18μ technology. Although, it has been found that the comparative power results at the gate level using wire load models with SDF (standard delay format) are not quite different from those obtained after layout.

- Only the direct form of the FIR filter is explored in this work. This work can be extended to other filter structures as well. Moreover, the performance of an architecture derived from the combination of coefficient segmentation and ordering needs to be explored.

References

[1] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery driven system design: a new frontier in low power design," in *15th International Conference on VLSI design*, vol. 1, pp. 261–267, 2002.

[2] L. D. Paulson, "Low-power chips for high-powered handhelds," *Computer*, vol. 36, pp. 21–23, January 2003.

[3] M. Pedram, "Power minimisation in IC design: principles and applications," in *ACM transactions on Design Automation of Electronic systems*, vol. 1, pp. 3–56, January 1996.

[4] A. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, April 1992.

[5] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS design*. Kluwer, 1995.

[6] A. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *IEEE Proceedings*, vol. 83, pp. 498–523, April 1995.

[7] C. Small, "Shrinking devices put the squeeze on system packaging," *EDN*, vol. 39, pp. 41–46, February 1994.

[8] F. Swarts, P. Rooyan, I. Oppermann, and M. Lotter, *CDMA techniques for third generation mobile systems*. Kluwer, 1999.

[9] A. McCormick and E. Al-Susa, "Multicarrier CDMA for future generation mobile communication," *Electronics and Communication Engineering Journal*, vol. 14, pp. 52–60, April 2002.

[10] N. Zhang and R. Brodersen, "Architectural evaluation of flexible digital signal processing for wireless receivers," in *34th Asimolar Conference on Signals, Systems and Computers*, vol. 1, pp. 78–83, 2000.

[11] P. Merakos, K. Masselos, O. Koufopavlou, S. Nikolaidis, and C. Goutis, "A novel transformation for reduction of switching activity in FIR filter implementation," in *International Conference on Digital Signal Processing*, vol. 2, pp. 653–656, July 1997.

[12] A. Erdogan and T. Arslan, "An order based segmentation algorithm for low power implementation of digital filters," in *IEEE Int. Conference on Acoustics, Speech, and Signal Processing (ICASSP'2000)*, pp. D441 – D444, June 2000.

[13] B. Guoan and E. Jones, "A pipelined FFT processor for word sequential data," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, pp. 1982–1985, December 1989.

[14] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high performance FFT processors," *IEEE Transactions on Signal Processing*, vol. 48, pp. 917–921, March 2000.

[15] D. Cohen, "Simplified control of FFT hardware," in *IEEE Transaction on Acoustics, Speech and Signal Processing*, vol. 24, pp. 577–579, December 1976.

[16] S. Hara and R. Prasad, "Design and performance of multicarrier CDMA system in frequency selective rayleigh fading channels," *IEEE Transactions on Vehicular Technology*, vol. 48, pp. 1584–1595, September 1999.

[17] A. Erdogan and T. Arslan, "A coefficient segmentation algorithm for low power implementation of FIR filters," in *IEEE International Symposium on Circuits and Systems*, pp. III.359 – III.362, June 1999.

[18] A. Erdogan and T. Arslan, "Data block processing for low power implementation of direct form FIR filters on single multiplier CMOS based DSPs," in *IEEE International Symposium on Circuits and Systems*, pp. D441 – D444, June 1998.

[19] J.M.Rabaey and M. Pedram, *Low power design methodologies*. Kluwer, 1996.

[20] G. Tiwary, "Below the half-micron mark," *IEEE Spectrum*, vol. 31, pp. 84–87, November 1994.

[21] K. Khouri and N. Jha, "Leakage power analysis and reduction during behavioral synthesis," *IEEE Transactions on VLSI Systems*, vol. 10, pp. 876–885, December 2002.

[22] D. B. Lidsky and J. Rabaey, "Low power design of memory intensive functions," in *IEEE International Symposium on Low Power Electronics and Design*, vol. 1, pp. 16–17, October 1994.

[23] D. Garrett, M. Stan, and A. Dean, "Challenges in clockgating for a low power ASIC methodology," in *IEEE International Symposium on Low Power Electronics and Design*, pp. 176–181, 1999.

[24] C. Chen, C. Kang, and M. Sarrafzadeh, "Activity-sensitive clock tree construction for low power," in *IEEE International Symposium on Low Power Electronics and Design*, pp. 279–282, August 2002.

[25] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 14, pp. 12–30, January 1995.

[26] R. Amritatrajah, T. Xanthapoulas, and A. Chandrakasan, "Power scalable processing using distributed arithmetic," in *IEEE International Symposium on Low Power Electronics and Design*, pp. 170–175, 1999.

[27] M. Mehendale, A. Sinha, and S. Sherlekar, "Low power realisation of FIR filters implemented using distributed arithmetic," in *Design Automation Conference*, pp. 151–156, February 1998.

[28] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *International Conference on Computer-aided Design*, pp. 398–402, November 1993.

[29] C. Schimpfle, S. Simon, and J. Nossek, "Optimal placement of registers in data paths for low power design," in *IEEE International Symposium on Circuits and Systems*, pp. 2160–2163, June 1997.

[30] A. Raghunathan, S. Dey, and N. Jha, "Glitch analysis and reduction in register transfer level power optimisation," in *33rd ACM/IEEE Design Automation Conference*, June 1996.

[31] M. Alidina, J. Monteiro, S. Devdas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," in *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 2 no.4, pp. 425–436, December 1994.

[32] J. Monteiro, S. Devdas, and A. Ghosh, "Sequential logic optimization for low power using input-disabling precomputation architectures," in *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 17 no.3, pp. 279–284, March 1998.

[33] J. Sacha and M. Irwin, "Number representations for reducing switched capacitance in subband coding," in *International Conference on Acoustic, Speech and Signal Processing*, pp. 3125–3128, 1998.

[34] S. Wuytack, F. Catthoor, F. Franssen, L. Nachtergale, and H. Deman, "Global communication and memory optimising transformation for low power design," in *International Workshop on Low Power Design*, pp. 178–187, April 1994.

[35] C. Su, C. Tsui, and A. Despain, "Low power architecture design and and compilation techniques for high performance processors," in *Digest of papers COMPCON spring*, pp. 489–498, February 1994.

[36] M. Stan and W. Burleson, "Bus-invert coding for low power I/O," *IEEE Transactions on VLSI Systems*, vol. 3, pp. 49–58, March 1995.

[37] L. Benini, L. Macchiarulo, A. Macii, and M. Poncino, "Layout-driven memory synthesis for embedded systems-on-chip," *IEEE Transactions on VLSI Systems*, vol. 10, pp. 96–105, April 2002.

[38] L. Benini, L. Macchiarulo, A. Macii, and M. Poncino, "Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation," in *IEEE Design and Test*, vol. 17, pp. 74–85, April 2000.

[39] P. Panda and N. Dutt, "Low-power memory mapping through reducing address bus activity," *IEEE Transactions on VLSI Systems*, vol. 7, pp. 309–320, September 1999.

[40] L. Benini and G. Micheli, "State assignment for low power dissipation," in *IEEE Journal of Solid State Circuits*, vol. 30, pp. 258–267, March 1995.

[41] K. Roy and S. Prasad, "Circuit activity based logic synthesis for low power reliable operations," in *IEEE Transactions on VLSI System*, vol. 1, pp. 503–513, December 1993.

[42] E. Musoll and J. Cortadella, "Scheduling and resource binding for low power," in *International Symposium on System Synthesis*, pp. 104–109, 1995.

[43] T. Callaway and E. Swartzlander, "Optimising adders for WSI," in *IEEE International Conference on Wafer Scale Integration*, pp. 251–260, 1992.

[44] T. Callaway and E. Swartzlander, "Optimising multipliers for WSI," in *IEEE International Conference on Wafer Scale Integration*, pp. 85–94, 1993.

[45] G. Keane, J. Spanier, and R. Woods, "The impact of data characteristics and hardware topology on hardware selection for low power DSP," in *International Symposium on Low Power Electronics and Design*, pp. 94–96, August 1998.

[46] C. Tsui, M. Pedram, and A. Despain, "Power efficient technology decomposition and mapping under an extended power consumption model," in *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 13(9), pp. 1110–1122, September 1994.

[47] C. Yeh, C.-C. Chang, and J.-S. Wang, "Technology mapping for low power," in *Design Automation Conference*, vol. 1, pp. 145–148, January 1999.

[48] H. Choi and W. Burleson, "Search-based wordlength optimization for VLSI/DSP synthesis," in *Workshop on VLSI Signal Processing*, pp. 198–207, 1994.

[49] D.-S. Chen and M. Sarrafzadeh, "An exact algorithm for low power library-specific gate re-sizing," in *33rd ACM/IEEE Design Automation Conference*, June 1996.

[50] H. Vaishnav and M. Pedram, "Pcube: A performance driven placement algorithm for low power design," in *European Design Automation Conference*, pp. 72–77, September 1993.

[51] R. Mehra, L. Guerra, and J. Rabaey, "A partitioning scheme for optimising interconnect power," *IEEE Journal of Solid State Circuits*, vol. 32, pp. 433–443, March 1997.

[52] J. Cong, C.-K. Koh, and K.-S. Leung, "Simultaneous driver and wire sizing for performance and power optimisation," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 408–425, December 1994.

[53] B. M. Bass, "A low-power high-performance 1024-point FFT processor," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 380–387, March 1999.

[54] B. M. Bass, "An energy efficient single-chip FFT processor," in *Symposium on VLSI Circuits*, pp. 164–165, June 1996.

[55] K. Masselos, P. Merakos, T. Stouratis, and C. Goutis, "Novel techniques for bus power consumption reduction in realizations of sum-of-product computation," *IEEE Transactions on VLSI Systems*, vol. 7, pp. 492–497, December 1999.

[56] K. Masselos, S. Theoharis, P. Merakos, T. Stouratis, and C. Goutis, "Low power synthesis of sum-of-product computation," in *IEEE International Symposium on Low Power Electronics and Design*, pp. 234–237, 2000.

[57] K. Masselos, P. Merakos, T. Stouratis, and C. Goutis, "Low power synthesis of sum-of-product computation in DSP algorithms," in *IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 420–423, July 1999.

[58] S. Johansson, S. He, and P. Nilsson, "Wordlength optimization of a pipelined FFT processor," in *42nd Midwest Symposium on Circuits and Systems*, vol. 1, pp. 501–503, 1999.

[59] L. Jia, Y. Gao, J. Isoaho, and H. Tenhunen, "A new VLSI-oriented FFT algorithm and implementation," in *11th Annual IEEE International ASIC Conference*, pp. 337–341, September 1998.

[60] L. Jia, Y. Gao, J. Isoaho, and H. Tenhunen, "Implementation of a low power 128-point FFT," in *6th International Conference on Solid-state and Integrated Circuit Technology*, pp. 369–372, 1998.

[61] L. Rabiner and B. Gold, *Theory and application of digital signal processing*. Prentice Hall, 1975.

[62] B. M. Bass, *An approach to low-power, high-performance, Fast Fourier transform processor design*. PhD thesis, Stanford University, 1999.

[63] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-state Circuits*, vol. 30, pp. 300–305, March 1995.

[64] S. Sridhara and N. Shanbhag, "Low-power FFT via reduced precision redundancy," in *IEEE Workshop on Signal Processing Systems*, pp. 117–124, 2001.

[65] R. Hegde and N. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *IEEE International Symposium on Low Power Electronics and Design*, pp. 30–35, 1999.

[66] K. Stevens and B. Suter, "A mathematical approach to a low power FFT architecture," in *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 21–24, 1998.

[67] B. Hunt, K. Stevens, B. Suter, and D. Gelosh, "A single chip low power asynchronous implementation of an FFT algorithm for space applications," in *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 216–223, March 1998.

[68] Y. Wang, H. Lam, C.-Y. Tsui, R. Cheng, and W. Mow, "Low complexity OFDM receiver using log-FFT for coded OFDM system," in *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 445–448, 2002.

[69] Y. Wang, C.-Y. Tsui, R. Cheng, and W. Mow, "Performance study of OFDM receiver using FFT based on log number system," in *IEEE 55th Vehicular Technology Conference*, vol. 3, pp. 1257–1259, 2002.

[70] J. Ludwig, S. Nawab, and A. Chandrakasan, "Low power digital filtering using approximate processing," in *IEEE Journal of Solid State Circuits*, vol. 31, pp. 395–399, March 1996.

[71] M. Mehendale, S. Sherlekar, and G. Venkatesh, "Low power realisation of FIR filters using multirate architectures," in *9th International Conference on VLSI Design*, pp. 370–375, January 1996.

[72] M. Mehendale, S. Sherlekar, and G. Venkatesh, "Coefficient optimisation for low power realisation of FIR filters," in *IEEE Workshop on VLSI Signal Processing*, pp. 352–361, 1995.

[73] N. Sankarayya, K. Roy, and D. Bhattacharya, "Algorithms for low power and high speed FIR filter realisation using differential coefficients," in *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 44, pp. 488–497, June 1997.

[74] Z.Yu, M.-L. Yu, K. Azadet, and A. W. Jr., "A low power FIR filter design technique using dynamic reduced signal representation," in *International Symposium on VLSI Technology, Systems and Applications*, pp. 113–116, 2001.

[75] J. Park, W. Jeong, H. Choo, H. Mahmoodi-Meimand, Y. Wang, and K. Roy, "High performance and low power FIR filter design based on sharing multiplication," in *International Symposium on Low Power Electronics and Design*, pp. 295–300, August 2002.

[76] A. McCormick, P. Grant, J. Thompson, T. Arslan, and A. Erdogan, "Low power receiver architectures for multi-carrier CDMA," *IEE Proceedings*, vol. 149, pp. 227–233, August 2002.

[77] A. McCormick, P. Grant, J. Thompson, T. Arslan, and A. Erdogan, "A low power MMSE receiver architecture for multi-carrier CDMA," in *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 41–44, May 2001.

[78] A. Oppenheim and R. Schafer, *Discrete-time signal processing*. Prentice Hall, 1989.

[79] J. Proakis and D. Manolakis, *Digital signal processing: Principles, algorithms and applications*. Macmillan publishing company, 1992.

[80] J.W.Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *International Journal of Mathematics of Computation*, vol. 19, pp. 297–301, April 1965.

[81] M. Hasan and T. Arslan, "Implementation of low power FFT processor cores using a novel order based processing scheme," *IEE Proceedings on Circuits, Devices and Systems*, vol. 50, pp. 149–154, June 2003.

[82] Y. Chang and K. Parhi, "Efficient FFT implementation using digit-serial arithmetic," in *IEEE Workshop on Signal Processing Systems*, pp. 645–653, 1999.

[83] Y. Ma, "An effective memory addressing scheme for FFT processors," *IEEE Transactions on Signal Processing*, vol. 47, pp. 907–911, March 1999.

[84] M. Hasan and T. Arslan, "Scheme for reducing size of the coefficient memory in FFT processor," *Electronic Letters*, vol. 38, pp. 163–653, February 2002.

[85] M. Hasan and T. Arslan, "FFT coefficient memory reduction technique for OFDM applications," in *International Conference on Acoustics, Speech and Signal processing*, vol. 1, pp. 1085–1088, May 2002.

[86] M. Hasan and T. Arslan, "Coefficient memory addressing scheme for high performance FFT processors," *Electronic Letters*, vol. 37, pp. 1322–1324, October 2001.

[87] M. Hasan and T. Arslan, "A coefficient memory addressing scheme for VLSI implementation of FFT processors," in *International Symposium on Circuits and Systems*, vol. 4, pp. 850–853, May 2002.

[88] M.C.Pease, "Organization of large scale Fourier processors," *JACM*, vol. 16, pp. 474–482, July 1969.

[89] W. Li and L. Wanhammar, "A pipeline FFT processor," in *IEEE workshop on Signal Processing Systems*, vol. 4, pp. 654–662, October 1999.

[90] N. Zhang and R. Brodersen, "Architectural evaluation of flexible digital signal processing for wireless receivers," in 34^{th} *Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 78–83, 2000.

[91] L. Fanucci, M. Forliti, and P. Terreni, "Fast: FFT ASIC automated synthesis," in *INTEGRATION the VLSI journal*, vol. 33, pp. 23–37, 2002.

[92] B. Gold and T. Bially, "Parallelism in fast fourier transform hardware," *IEEE Transactions audio Electroacoustics*, vol. 21, no. 1, pp. 5–16, 1973.

[93] C. Hui, T. Ding, J. McCanny, and R. Woods, "A 64-point Fourier Transform Chip for Video Motion Compensation using Phase Correlation," *IEEE Journal of Solid-State Circuits*, vol. 49, pp. 1751–1761, November 1996.

[94] M. Hasan, T. Arslan, and J. Thompson, "A novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications," *IEEE Transactions on Consumer Electronics*, vol. 49, pp. 128–134, February 2003.

[95] M. Hasan and T. Arslan, "A triple port RAM based low power commutator architecture for a pipelined FFT processor," in *International Symposium on Circuits and Systems*, vol. 5, pp. 353–356, May 2003.

[96] N. Yee, J. Linnartz, and G. Fettweis, "Multi-carrier CDMA in indoor wireless radio networks," in *IEEE PIMRC*, pp. 109–113, September 1993.

[97] K. Fazel and L. Papke, "On the performance of convolutionally-coded CDMA/OFDM for mobile communication system," in *IEEE PIMRC*, pp. 468–472, September 1993.

[98] R. Prasad and S. Hara, "An overview of multi-carrier CDMA," in *International Symposium on Spread Spectrum Techniques and Applications*, pp. 107–114, September 1996.

[99] J. Bingham, "Multi-carrier modulation for data transmission: An idea whose time has come," in *IEEE Communication Magazine*, vol. 36, pp. 112–117, February 1998.

[100] S. Hara, T.-H. Lee, and R. Prasad, "BER comparison of DS-CDMA and MC-CDMA for frequency selective fading channels," in *7th Tyrrhenian International Workshop on Digital Communications*, pp. 3–14, September 1995.

[101] N.Yee and J. Linnartz, "Wiener filtering of multicarrier CDMA in a rayleigh fading channel," in *IEEE PIMRC*, pp. 1344–1347, September 1994.

[102] Y. Bar-Ness, J. Linnartz, and X.Liu, "Synchronous multi-user multi-carrier CDMA communication system with decorrelating interference canceler," in *IEEE PIMRC*, pp. 184–188, September 1994.

[103] S. Hara and R. Prasad, "DS-CDMA, MC-CDMA and MT-CDMA for mobile multimedia communications," in *IEEE Vehicular Technology Conference*, pp. 1106–1110, April 1996.

[104] V.M.DaSilva and E. Sousa, "Multicarrier orthogonal CDMA signals for quasi-synchronous communication systems," *IEEE Journal on Select Areas of Communication*, vol. JSAC-12, pp. 1106–1110, June 1994.

[105] E. Sourour and M. Nakagawa, "Performance of orthogonal multicarrier CDMA in a multipath fading channel," *IEEE Transactions on Communications*, vol. 44, pp. 356–367, March 1996.

[106] R. V. Nee and R. Prasad, *OFDM wireless multimedia communications*. Artech House, 2000.

[107] R. Prasad, *CDMA for wireless personal communications*. Artech House, 2000.

[108] J.G.Proakis, *Digital Communications*. Mc-Graw Hill, 1995.

[109] D. R.L. Pickholtz and L. Milstein, "Theory of spread spectrum communication- a tutorial," *IEEE Transactions on Communcations*, vol. 30, pp. 855–884, May 1982.

[110] M. Hasan, T. Arslan, and J. Thompson, "A novel low power pipelined architecture for a MC-CDMA receiver," in 3^{rd} *International Symposium on Image and Signal Processing and Analysis*, To appear in September 2003.

[111] I. Kang and A. W. Jr., "Low-power Viterbi decoder for CDMA mobile terminals," *IEEE Journal of Solid-state Circuits*, vol. 33, pp. 473–482, March 1998.

[112] R. Henning and C. Chakrabarti, "Low-power approach for decoding convolutional codes with adaptive Viterbi algorithm approximations," in *International Symposium on Low Power Electronics and Design*, pp. 68–71, 2002.

[113] M. Hasan, T. Arslan, and J. Thompson, "A delay spread based low power reconfigurable FFT processor architecture for wireless receivers," in *International Symposium on System on Chip*, To appear in November 2003.

[114] I. Abu-Khater, A. Bellaouar, and M. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers," *IEEE Journal of Solid-state Circuits*, vol. 31, pp. 1535–1546, October 1996.

[115] A. Erdogan and T. Arslan, "Low power implementation of linear phase FIR filters for single multiplier CMOS based DSPs," in *International Symposium on Circuits and Systems*, vol. D, pp. 425–428, May 1998.

[116] A. Erdogan, M. Hasan, and T. Arslan, "A low power FIR filtering core," in *International Conference on ASIC/SOC*, pp. 271–275, September 2001.

[117] A. Erdogan, M. Hasan, and T. Arslan, "Algorithmic low power FIR cores," *IEE Proceedings on Circuits, Devices and Systems*, vol. 150, pp. 155–160, June 2003.

Appendix A
Publications

A.1 Refereed Journals

1. M. Hasan, T. Arslan and J. Thompson, "A Novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications", IEEE Transactions on Consumer Electronics, vol. 49, no. 1, pp.128-134, February, 2003.

2. M. Hasan and T. Arslan, "Implementation of low power FFT processor cores using a novel order based processing scheme", IEE proceedings on Circuits, Devices and Systems, vol. 150, no. 3, pp. 149-154, June 2003.

3. A.T. Erdogan, M. Hasan and T.Arslan, "Algorithmic low power FIR cores", IEE proceedings on Circuits, Devices and Systems, vol. 150, no. 3, pp. 155-160, June 2003.

4. M. Hasan and T. Arslan, "Scheme for reducing size of the coefficient memory in FFT processor", Electronic letters, vol. 38, no. 4, pp.163-164, February, 2002.

5. M. Hasan and T. Arslan, "Coefficient memory addressing scheme for high performance FFT processors", Electronic letters, vol. 37, no. 22, pp.1322-1324, October, 2001.

A.2 Refereed Conferences

1. A.T. Erdogan, M. Hasan and T. Arslan, "A low power FIR filtering core", IEEE International Conference on ASIC/SOC, pp. 271-275, September, 2001.

2. M. Hasan and T. Arslan, "A coefficient memory addressing scheme for VLSI implementation of FFT processors", International Symposium on Circuits and Systems, Volume : 4, pp. 850-853, May, 2002.

3. M. Hasan and T. Arslan, "FFT coefficient memory reduction technique for OFDM applications", International Conference on Acoustics, Speech and Signal processing, Volume : 1, pp. 1085-1088, May, 2002.

4. M. Hasan and T. Arslan, "A Triple Port RAM Based Low Power Commutator Architecture for a Pipelined FFT Processor", International Symposium on Circuits and Systems, Volume : 5, pp. 353-356, May, 2003.

5. M. Hasan, T. Arslan and J. Thompson, "A Novel Low Power Pipelined Architecture for a MC-CDMA receiver", 3^{rd} International Symposium on Image and Signal processing and Analysis, September, 2003 in Rome, Italy. To appear.

6. M. Hasan, T. Arslan and J. Thompson, "A Delay spread based low power reconfigurable FFT processor architecture for wireless receivers", International Symposium on System on Chip, November, 2003, in Tampere, Finland. To appear.

Appendix B
C-code for the order based processing algorithm

```
/*  This programme reads in coefficient set, performs quantisation
    based on a specified wordlength and arrange the coefficients as
    per Our order based processing scheme. */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>

#define max_wordlength   32
#define maxcoeff_number  4096

int wordlength,count1,count2,count, hold_bit_number,exchange_index,
coeff_number,rcoef_min[maxcoeff_number][2],
icoef_min[maxcoeff_number][2],
dsignal[maxcoeff_number],rcoef_norm[maxcoeff_number],
icoef_norm[maxcoeff_number],signal[maxcoeff_number],
nrcoef_min[maxcoeff_number][2],nicoef_min[maxcoeff_number][2],
rcoef_comp[maxcoeff_number];
double imag_coef[maxcoeff_number], real_coef[maxcoeff_number],
m_factor,tempr,tempi;

int main(int argc, char *argv[])
{
   int xor,temp,r;
   int i,j,k,trhnorm,trhmin,p,li;
   FILE  *coefi, *rcoef, *report1, *report2 ;
```

```
/* read in command line arguments */

for(i=0;i<argc;i++)
    if(strcmp(argv[i],"-w") == NULL) {
       wordlength = atoi(argv[i+1]);
             }

for(i=0;i<argc;i++)
    if(strcmp(argv[i],"-f1") == NULL) {
       coefi = fopen(argv[i+1], "r+");
       if(coefi==NULL) printf("Can't open coef\n");
             }

for(i=0;i<argc;i++)
    if(strcmp(argv[i],"-f2") == NULL) {
       rcoef = fopen(argv[i+1], "r+");
       if(rcoef==NULL) printf("Can't open coef\n");
             }

/* Step 1 - Read in the imaginary and real parts of the
            coefficient in decimal and store it in files
            coefi and rcoef respectively */

/* Read in the imaginary part */

coeff_number = 0;
while(!feof(coefi)){
    fscanf(coefi, "%lf", &imag_coef[coeff_number]);
    coeff_number++;
             }
coeff_number--;
fclose(coefi);
printf("\n\n\ncoeff_number is %d\n\n\n", coeff_number);

/* Read in the real part */

coeff_number = 0;
while(!feof(rcoef)) {
```

```
    fscanf(rcoef, "%lf", &real_coef[coeff_number]);
     coeff_number++;
              }
coeff_number--;
fclose(rcoef);
printf("\n\n\ncoeff_number is %d\n\n\n", coeff_number);

/* Step 2 - Quantise coefficients as per wordlength  */

m_factor = pow(2,wordlength-1) - 1;
for(i=0;i<maxcoeff_number;i++) {
     tempr = real_coef[i] * m_factor;
     tempi = imag_coef[i] * m_factor;
  if(tempr >=0) {
    rcoef_norm[i]   = (int)(tempr + 0.9);
    rcoef_min[i][1] = (int)(tempr + 0.9);
    rcoef_min[i][0] = i;
              }
  else {
    rcoef_norm[i]   = (int)(tempr + 0.9 + (pow(2,wordlength)-1));
    rcoef_min[i][1] = (int)(tempr + 0.9 + (pow(2,wordlength)-1));
    rcoef_min[i][0] = i;
      }
  if(tempi >= 0) {
    icoef_norm[i]   = (int)(tempi + 0.9);
    icoef_min[i][1] = (int)(tempi + 0.9);
    icoef_min[i][0] = i;
              }
  else {
    icoef_norm[i]  = (int)(tempi + 0.9 + (pow(2,wordlength)-1));
    icoef_min[i][1] = (int)(tempi + 0.9 + (pow(2,wordlength)-1));
    icoef_min[i][0] = i;
      }
  if(rcoef_norm[i] == 0)
    rcoef_comp[i] = rcoef_norm[i];
  else
    rcoef_comp[i] = (int)(pow(2,wordlength)-rcoef_norm[i]);
                    } /* for */
```

```
/* Step 3 - Order coefficients for minimum Hamming distance */
/* Count the number of transitions in the initial
   imaginary coefficient set */

trhnorm=0;
for(j=0;j<coeff_number-1;j++)
   {
    xor=icoef_norm[j]^icoef_norm[j+1];
    for(k=0;k<wordlength;k++)
           {
         if(xor & 01) trhnorm++;
         xor >>= 1;
           }
  }



/*  Count the number of transitions between the last and the
    first imaginary coefficient */

xor=icoef_norm[coeff_number-1]^icoef_norm[0];
for(k=0;k<wordlength;k++)
      {
      if(xor & 01)  trhnorm++;
      xor >>= 1;
      }
/*  Count the number of transitions in the initial real
    coefficient set */
for(j=0;j<coeff_number-1;j++)
  {
      xor=rcoef_norm[j]^rcoef_norm[j+1];
      for(k=0;k<wordlength;k++)
         {
           if(xor & 01) trhnorm++;
       xor >>= 1;
           }
  }
```

```
/*  Count the number of transitions between the last and the
    first real coefficient */

xor=rcoef_norm[coeff_number-1]^rcoef_norm[0];
for(k=0;k<wordlength;k++)
    {
      if(xor & 01)  trhnorm++;
      xor >>= 1;
    }

/* Display the normal switching activity of the coefficient set */
printf("Number of transitions for icoef_norm vector and
      rcoef_norm vector= %4d\n",trhnorm);

/* Sort the imaginary parts of the input coefficient vector
   for the minimum transition icoef_min */
/* Find an imaginary coefficient with minimum 1's in it and
   place it at the top */

hold_bit_number=wordlength;
for(j=0;j<coeff_number;j++)
   {
    count=0;
    xor=icoef_min[j][1];
    for(k=0;k<wordlength;k++)
    {
     if(xor & 01) count++;
     xor >>=1;
    }
      if(count < hold_bit_number)
        {
        hold_bit_number = count;
        exchange_index = j;
        }
      }
    if(hold_bit_number != wordlength)
    {
```

```
      r=icoef_min[0][0];
      temp=icoef_min[0][1];
      icoef_min[0][0]=icoef_min[exchange_index][0];
      icoef_min[0][1]=icoef_min[exchange_index][1];
      icoef_min[exchange_index][0]=r;
      icoef_min[exchange_index][1]=temp;
      }

/* Now sort the rest of the imaginary coefficients */

for(j=0;j<coeff_number-1;j++)
   {
      hold_bit_number=wordlength;
      for(k=j+1;k<coeff_number;k++)
      {
       count=0;
       xor=icoef_min[j][1]^icoef_min[k][1];
           for(i=0;i<wordlength;i++)
            {
         if(xor & 01 ) count++;
         xor >>=1;
            }
         if(count < hold_bit_number)
           {
            hold_bit_number = count;
            exchange_index = k;
           }
      }
      if(hold_bit_number != wordlength)
        {
         r=icoef_min[j+1][0];
         temp=icoef_min[j+1][1];
         icoef_min[j+1][0]=icoef_min[exchange_index][0];
         icoef_min[j+1][1]=icoef_min[exchange_index][1];
         icoef_min[exchange_index][0]=r;
         icoef_min[exchange_index][1]=temp;
         }
   }
```

```
/* The imaginary coefficient has already been sorted
   on the basis of minimum Hamming distance and now start
   arranging the real part */
/* Choose the first normal real part or its complement
   corresponding to the already aligned imaginary part
   depending upon the no.of 1's */

count1=0;
j=icoef_min[0][0];
xor=rcoef_norm[j];
for(k=0;k<wordlength;k++)
   {
     if(xor & 01) count1++;
     xor >>=1;
   }

count2=0;
xor=rcoef_comp[j];
for(k=0;k<wordlength;k++)
   {
     if(xor & 01) count2++;
     xor >>=1;
   }
if(count1 < count2)
         {
           rcoef_min[0][1] = rcoef_norm[j];
           rcoef_min[0][0] = j;
           signal[0] = 0;
         }
         else
         {
           rcoef_min[0][1] = rcoef_comp[j];
           signal[0] =1;
           rcoef_min[0][0] = j;
         }
```

```
/* Choose the normal real part or its complement
   depending upon the Hamming distance with the
   preceding real coefficient */

for(j=0;j<coeff_number-1;j++)
       {
         p=icoef_min[j+1][0];
     count1=0;
     xor=rcoef_min[j][1]^rcoef_norm[p];
         for(i=0;i<wordlength;i++)
         {
           if(xor & 01 ) count1++;
           xor >>=1;
         }
     count2=0;
     xor=rcoef_min[j][1]^rcoef_comp[p];
     for(k=0;k<wordlength;k++)
         {
           if(xor & 01) count2++;
           xor >>=1;
         }
     if(count1 < count2)
             {
               rcoef_min[j+1][1] = rcoef_norm[p];
               rcoef_min[j+1][0] = p;
               signal[j+1]=0;
           }
         else
           {
               rcoef_min[j+1][1] = rcoef_comp[p];
               rcoef_min[j+1][0] = p;
               signal[j+1]=1;
           }
   }
```

```
/* Count the number of transitions in the ordered
   imaginary vector */

trhmin=0;
for(j=0;j<coeff_number-1;j++)
    {
     xor=icoef_min[j][1]^icoef_min[j+1][1];
     for(k=0;k<wordlength;k++)
         {
          if(xor & 01) trhmin++;
      xor >>= 1;
         }
    }
     xor=icoef_min[coeff_number-1][1]^icoef_min[0][1];
     for(k=0;k<wordlength;k++)
     {
      if(xor & 01) trhmin++;
      xor >>= 1;
      }
/* Count the number of transitions in the ordered real
   vector */

for(j=0;j<coeff_number-1;j++)
    {
     xor=rcoef_min[j][1]^rcoef_min[j+1][1];
     for(k=0;k<wordlength;k++)
         {
       if(xor & 01) trhmin++;
       xor >>= 1;
     }
    }
     xor=rcoef_min[coeff_number-1][1]^rcoef_min[0][1];
     for(k=0;k<wordlength;k++)
     {
      if(xor & 01) trhmin++;
      xor >>= 1;
     }
```

```
/* Display the switching activity of the ordered
   coefficient set */

printf("Number of transitions for icoef_min vector
      and rcoef_min vector = %4d\n\n",trhmin);

/* Store the old and new order along with the
   indices in file coef_order.dat*/

report = fopen("coef_order.dat", "w");
   if(report1==NULL) printf("Can't open
                          coef_numbered.dat\n");
for(i=0;i<coeff_number;i++) {
   fprintf(report1, "h[%2d] = %6d (%0x) %6d (%0x)  ",i,
   icoef_norm[i],icoef_norm[i],rcoef_norm[i],rcoef_norm[i]);
   fprintf(report1, "%6d (%0x) %6d %6d (%0x) %6d  %d\n",
          icoef_min[i][1],icoef_min[i][1],icoef_min[i][0],
          rcoef_min[i][1],rcoef_min[i][1],rcoef_min[i][0],
          signal[i]);
                           }
fprintf(report1, "\nNumber of transitions for
       (r+i)coef_norm = %3d\n",trhnorm);
fprintf(report1, "Number of transitions for
       (r+i)coef_min = %3d\n",trhmin);

/* Store the modified set in original order in a specific
   format and print it into file mem.dat*/

report1 = fopen("mem.dat", "w");
   if(report2==NULL) printf("Can't open mem.dat\n");

for(i=0;i<coeff_number;i++) {
    for(li=0;li<coeff_number;li++) {
          if(rcoef_min[li][0]==i) {
                printf("%d \n",li);
                dsignal[i]=signal[li];
                nrcoef_min[i][1]= rcoef_min[li][1];
                nicoef_min[i][1]= icoef_min[li][1];
```

```
                                    }
                                    }
fprintf(report2, "7'h %02x: out = 33'h %d%04x%04x;\n",
       i,dsignal[i],nrcoef_min[i][1],nicoef_min[i][1]);
                                }

fclose(report1);
fclose(report2);
return 0; }
```

Appendix C
MATLAB code for the MC-CDMA transceiver

The MATLAB code has a main section which calls the transmitter and receiver functions. The mobile channel is also defined in the main section. The code accepts as inputs the simulation length or the number of data bits to be transmitted, the number of users and the signal to noise ratio. This appendix starts with the main section of the code. It will be followed by a section of the code for the mc-cdma transmitter. The appendix ends with a section having the code for the mc-cdma receiver.

C.1 Main section of code

The main section of the code accepts as inputs the length of the simulation or the number of data bits to be transmitted, the number of active users and the signal to noise ratio. The main section of the code defines the channel with the help of a five tap filter having exponentially decreasing random coefficients. The exponentially decreasing behaviour is due to the fact that the delayed signals have a lower average power than the direct path signal.

```
% Simulation length
  L=1000;
% No. of users
  N=32;
% Signal to noise ratio
  snr = 20;

% Generation of random coefficients for the five-tap FIR filter
  used to model the channel
  b = (randn(1,5)+j*randn(1,5)).*exp(-(0:4));
  b = b/(norm(b));
```

```
% Function used to model the mc-cdma transmitter
  [y] = mccdma_transmitter(L,64,N,4,31);

% Complex noise generation
  n=1/sqrt(2)*(randn(length(y),1)+j*randn(length(y),1));

% Signal at the input of receiver
  y1=filter(b,1,y) + 10^(-snr/20)*n;

% Function used to model the receiver
  [d1] = mc_receiver(L,64,N,4,31,y1,ts);
```

C.2 Function used to model the MC-CDMA transmitter

```
% Function used to model the mc-cdma transmitter
function [y] = mccdma_transmitter(n,L,N,x,ti)
% n - no of data bits
% L - code length
% N - number of users
% x - cyclic-extend length
% ti - training interval

% Code Definition (Normalised)
C = hadamard(L)/sqrt(L);

% Setup output vector (Column vector)
y=zeros(n*(L+x)+floor(n/ti)*(L+x),1);

% Create training symbol
trs = sign(randn(L,1));
ts=ifft(trs);
ts=[ts(L-x+1:L);ts];

% Create random data
(Each row contains n-bits corresponding to a user)
d=sign(randn(N,n));
```

```
index=1;
for k = 1:floor(n/ti)

% Transmit training symbol
y((k-1)*(ti+1)*(L+x)+(1:L+x)) = ts;

% Transmit ti data symbols
% (C:,(1:N)- First N-columns of C out of L)
% d(:,1) - first column of data
  for l = 1:ti
     td = ifft(C(:,1:N)*d(:,index));
     y((k-1)*(ti+1)*(L+x)+(1:L+x)+l*(L+x))=[td(L-x+1:L);td];
     index=index+1;
  end
end
%normalise y to average power of 1
y=y*sqrt(L);
```

C.3 Function used to model the MC-CDMA receiver

```
% Function used to model the receiver
function [d1] = mc_receiver(n,L,N,x,ti,y,ts)
% n - no of data bits
% L - code length
% N - number of users
% x - cyclic-extend/guard interval
% ti - training interval
% y - receiver signal
% ts - training sequence

% normalise power of y
y = y/sqrt(L);

% Code Definition (Normalised)
C = hadamard(L)/sqrt(L);
d1=zeros(N,n);
index=1;
```

```
for k = 1:floor(n/ti)

% Get channel estimate from training symbol
tr_position =(k-1)*(ti+1)*(L+x)+x; rts=fft(y(tr_position+(1:L))); ce=rts./ts;
eq=conj(ce)./abs(ce)**2;

% Data Reception
 for l = 1:ti
     dint= eq.*fft(y((k-1)*(ti+1)*(L+x)+(1:L)+l*(L+x)+x));
     d1(:,index) = sign(real(C(:,1:N)'*dint));
     index=index+1;
 end
end
```

Appendix D
Verilog code for the MC-CDMA receiver

The two key blocks of the MC-CDMA receiver are the FFT and the Combiner.

D.1 Verilog code for the FFT

```
// Top level FFT module

'timescale 1ns/1ps

'define dwidth  32
'define width 16
'define depth1 16
'define depth2 8
'define cwidth 5
'define swidth 4

module FFT(clk,in,Xfro,Xfio,reset,count1);

parameter dwidth  = 'dwidth;
parameter depth1  = 'depth1;
parameter depth2  = 'depth2;
parameter swidth = 'swidth;
parameter cwidth = 'cwidth;
parameter width = 'width;

input clk,reset;
input [dwidth-1:0] in;
output[width-1:0] Xfro,Xfio;
output[cwidth:0] count1;
wire c42,c52,c62,cm2,c41,c51,c61,cm1,c43,c53,c63,cm3;
wire [dwidth-1:0] o11,o21,o31,o41,o12,o22,o32,o42,o13,o23,o33,o43,w1,wo1,w2,wo2;
wire [width-1:0] Xtro1,Xtio1,Xmro1,Xmio1,Xfmro1,Xfmio1,Xtro2,Xtio2,Xmro2,Xmio2,Xfmro2,Xfmio2;


// ROM for storing the coefficient for stage 1

rom1 romb1(.out(wo1),.addr(count1));

// ROM for storing the coefficient for stage 2

rom2 romb2(.out(wo2),.addr(count1[3:0]));

// Registers for storing the current coefficient inputs

reg0 regr1(.clk(clk), .in(wo1),.out(w1));

regm regr2(.clk(clk), .in(wo2),.out(w2));

// First stage Commutator

comu1  comutator1(.clk(clk),.in(in),.o11(o11),.o21(o21),.o31(o31),.o41(o41),.reset(reset),.c41(c41),.c51(c51),
                  .c61(c61),.cm1(cm1),.count1(count1));

// First stage Butterfly

butter butter1(.Xro(Xtro1),.Xio(Xtio1),.xr0(o11[31:16]),.xi0(o11[15:0]),.xr1(o21[31:16]),.xi1(o21[15:0]),
               .xr2(o31[31:16]),.xi2(o31[15:0]), .xr3(o41[31:16]),.xi3(o41[15:0]),.c4(c41),.c5(c51),.c6(c61),
               .cm(cm1));

// Intermediate register

reg0 regs1(.clk(clk), .in({Xtro1,Xtio1}),.out({Xmro1,Xmio1}));
```

```
// First stage complex multiplier

cmult  multi1(.xr(Xmro1),.xi(Xmio1),.wr(w1[31:16]),.wi(w1[15:0]),.xro(Xfmro1),.xio(Xfmio1));

// Second stage commutator

comu2  comutator2(.clk(clk),.in({Xfmro1,Xfmio1}),.o12(o12),.o22(o22),.o32(o32),.o42(o42),.reset(reset),
                  .c42(c42),.c52(c52),.c62(c62),.cm2(cm2), .count2(count1[3:0]));

// Second stage Butterfly

butter butter2(.Xro(Xtro2),.Xio(Xtio2),.xr0(o12[31:16]),.xi0(o12[15:0]),.xr1(o22[31:16]),.xi1(o22[15:0]),
               .xr2(o32[31:16]),.xi2(o32[15:0]), .xr3(o42[31:16]),.xi3(o42[15:0]),.c4(c42),.c5(c52),.c6(c62),
               .cm(cm2));

// Intermediate register

reg0 regs2(.clk(clk), .in({Xtro2,Xtio2}),.out({Xmro2,Xmio2}));

// Ordered complex multiplier

cmulto  multi2(.xr(Xmro2),.xi(Xmio2),.wr(w2[31:16]),.wi(w2[15:0]),.xro(Xfmro2),.xio(Xfmio2),.si(w2[32]));

// Third stage commutator

comu3 comutator3(.clk(clk),.in({Xfmro2,Xfmio2}),.o13(o13),.o23(o23),.o33(o33),.o43(o43),.reset(reset),
                 .c43(c43),.c53(c53), .c63(c63),.cm3(cm3),.count3(count1[4:0]));

// Third stage butterfly

butter butter3(.Xro(Xfro),.Xio(Xfio),.xr0(o13[31:16]),.xi0(o13[15:0]),.xr1(o23[31:16]),.xi1(o23[15:0]),
               .xr2(o33[31:16]),.xi2(o33[15:0]),.xr3(o43[31:16]),.xi3(o43[15:0]),.c4(c43),.c5(c53),
               .c6(c63),.cm(cm3));
endmodule

// Butterfly module

module butter(Xro,Xio,xr0,xi0,xr1,xi1,xr2,xi2,xr3,xi3,c4,c5,c6,cm);

input [width-1:0] xr0,xi0,xr1,xi1,xr2,xi2,xr3,xi3;
output [width-1:0] Xro,Xio;
wire [width-1:0] m0,m1,m2,m3,xmr0,xmr1,xmr3,xmi1,xmi2,xmi3;
input c4,c5,c6,cm;

addon  ex1(.A(xr1), .SI(c5), .B(xmr1));
addon  ex2(.A(xr3), .SI(c6), .B(xmr3));
addon  ex3(.A(xi1), .SI(c5), .B(xmi1));
addon  ex4(.A(xi3), .SI(c6), .B(xmi3));
addon  ex5(.A(xi2), .SI(cm), .B(xmi2));
addon  ex6(.A(xr0), .SI(cm), .B(xmr0));
sum sumr(.INPUT({m0,m2,xmr1,xmr3}), .SUM(Xro));
sum sumi(.INPUT({m1,m3,xmi1,xmi3}), .SUM(Xio));
mux16 mux0(.out(m0),.in0(xmr0),.in1(xi0),.cont(c4));
mux16 mux1(.out(m1),.in0(xi0),.in1(xmr0),.cont(c4));
mux16 mux2(.out(m2),.in0(xr2),.in1(xmi2),.cont(c4));
mux16 mux3(.out(m3),.in0(xmi2),.in1(xr2),.cont(c4));

endmodule

// Control inversion module for the butterfly

module addon(A, SI, B);

input [width-1 : 0] A;
input SI;
output [width-1 : 0] B;
reg [width-1 : 0] B;

always @(A or SI)
begin
   if(SI)
     B = ~A;
   else
     B = A;
end
endmodule

// Summer module for the low power butterfly

module sum(INPUT, SUM);

input [swidth*width-1 : 0] INPUT;
output [width-1 : 0] SUM;

// Instance of DW02_sum
DW02_sum #(swidth, width)
    U1 (.INPUT(INPUT), .SUM(SUM));

endmodule
```

```
// 16-bit 2:1 multiplexer for the first stage butterfly

module mux16(out,in0,in1,cont);

output [width-1:0]    out;
input [width-1:0]     in0,in1;
input cont;
reg [width-1:0]       out;

always @(in0 or in1 or cont)
begin
     if(cont)
    out=in1;
     else
    out=in0;
   end

endmodule

// Complex multiplier for the first stage of FFT processor

module cmult(xr,xi,wr,wi,xro,xio);

input [width-1:0] xr,xi,wr,wi;
output[width-1:0] xro,xio;
wire  [width+width-1:0] m1,m2,m3,m4;

mult mult1(.A(xr),.B(wr),.TC(1'b1),.PROD(m1));
mult mult2(.A(xr),.B(wi),.TC(1'b1),.PROD(m2));
mult mult3(.A(xi),.B(wr),.TC(1'b1),.PROD(m3));
mult mult4(.A(xi),.B(wi),.TC(1'b1),.PROD(m4));
add add1(.A(m2[width+width-2 : width-1]),.B(m3[width+width-2 : width-1]),.CI(1'b0),.S(xio));
sub sub1(.A(m1[width+width-2 : width-1]),.B(m4[width+width-2 : width-1]),.CI(1'b0),.DIFF(xro));

endmodule // cmult

// Adder module definition

module add(A,B,CI,S,CO);

input [width-1:0] A,B;
output [width-1:0] S;
input CI;
output CO;

DW01_add #(width) add(.A(A),.B(B),.CI(CI),.SUM(S),.CO(CO));

endmodule

// Subtractor module definition

module sub(A, B, CI, DIFF, CO);

input [width-1 : 0] A,B;
input CI; output [width-1 : 0]
DIFF; output CO;

    // Instance of DW01_sub
    DW01_sub #(width)
      sub(.A(A), .B(B), .CI(CI), .DIFF(DIFF), .CO(CO));

endmodule

// Two's complement multiplier module definition

module mult(A, B, TC, PROD);

input [width-1 : 0] A,B;
input TC;
output [width+width-1 : 0] PROD;

    // Instance of DW02_mult
    DW02_mult #(width, width)
      mult ( .A(A), .B(B), .TC(TC), .PRODUCT(PROD) );

endmodule

// Ordered complex multiplier

module cmulto(xr,xi,wr,wi,xro,xio,si);

input [width-1:0] xr,xi,wr,wi;
input si;
output[width-1:0] xro,xio;
wire  [width+width-1:0] m2,m4;
wire [width-1:0] b1,b3;

block block1(.A(xr),.B(wr),.SI(si),.PRODUCT(b1));
```

```
mult mult2(.A(xr),.B(wi),.TC(1'b1),.PROD(m2));
block block3(.A(xi),.B(wr),.SI(si),.PRODUCT(b3));
mult mult4(.A(xi),.B(wi),.TC(1'b1),.PROD(m4));
add add1(.A(m2[width+width-2 : width-1]),.B(b3),.CI(1'b0),.S(xio));
sub sub1(.A(b1),.B(m4[width+width-2 : width-1]),.CI(1'b0),.DIFF(xro));

endmodule // cmult

// Combination of two' complement multiplier module and a control inversion addon module to
   implement order based processing scheme in the second stage complex multiplier

module block(A, B, SI, PRODUCT);

input SI;
input [width-1 : 0] A,B;
wire  [width+width-1 : 0] PRO;
output[width-1 : 0] PRODUCT;

  mult mult1(.A(A),.B(B),.TC(1'b1),.PROD(PRO));
  addon mux(.A(PRO[width+width-2 : width-1]),.SI(SI),.B(PRODUCT));

endmodule

// First stage commutator of the FFT processor,

module comu1(clk,in,o11,o21,o31,o41,reset,c41,c51,c61,cm1,count1);

input clk,reset;
input [dwidth-1:0] in;
output [dwidth-1:0] o11,o21,o31,o41;
output c41,c51,c61,cm1;
output[counter_width-1:0] count1;
wire [dwidth-1:0] out0,out1,out2,out3,out4,out5;
wire [swidth-1:0] addr1,addr2;

assign o11=out2;

// Dual port RAMs are used as FIFOs (Six FIFOs)

duramc1 ram0(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addr2),
        .wr_addr(addr1), .data_in(in), .data_out(out0));

duramc1 ram1(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addr2),
        .wr_addr(addr1), .data_in(out0), .data_out(out1));

duramc1 ram2(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addr2),
        .wr_addr(addr1), .data_in(out1), .data_out(out2));

duramc1 ram3(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addr2),
        .wr_addr(addr1), .data_in(out2), .data_out(out3));

duramc1 ram4(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addr2),
        .wr_addr(addr1), .data_in(out3), .data_out(out4));

duramc1 ram5(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addr2),
        .wr_addr(addr1), .data_in(out4), .data_out(out5));

mux32 mux1(.out(o21),.in0(out3),.in1(in),.cont(c11));

mux32 mux2(.out(o31),.in0(out4),.in1(out0),.cont(c21));

mux32 mux3(.out(o41),.in0(out5),.in1(out1),.cont(c31));

// FSM of first stage commutator

fsmc1 fsm1(.c11(c11),.c21(c21),.c31(c31),.addr1(addr1),.addr2(addr2),.clk(clk),.reset(reset),
          .c41(c41),.c51(c51),.c61(c61),.cm1(cm1),.count1(count1));

endmodule

// Dual port RAM for the first stage commutator

module duramc1(clk, rst_n, cs_n, wr_n, rd_addr, wr_addr, data_in, data_out);

parameter rst_mode = 0;

input clk;
input rst_n;
input cs_n;
input wr_n;
input [swidth-1 : 0] rd_addr,wr_addr;
input [dwidth-1 : 0] data_in;
output[dwidth-1 : 0] data_out;

    // Instance of DW_ram_r_w_s_dff
    DW_ram_r_w_s_dff #(dwidth, width, rst_mode)
      duramc1 (.clk(clk), .rst_n(rst_n), .cs_n(cs_n), .wr_n(wr_n), .rd_addr(rd_addr),
       .wr_addr(wr_addr),.data_in(data_in), .data_out(data_out));
```

```
endmodule

// FSM for the first stage of the FFT processor

module fsmc1(c11,c21,c31,clk,addr1,addr2,reset,c41,c51,c61,cm1,count1);

   output c11,c21,c31,c41,c51,c61,cm1;
   output [swidth-1:0] addr1,addr2;
   reg [swidth-1:0] addr1,addr2;
   input clk,reset;
   output [count_width-1:0]       count1;
   reg [count_width-1:0]       count1;
   reg c11,c21,c31,c41,c51,c61;

   assign cm1 = c51^c61;

   always @(posedge clk)

   begin
        if(reset) begin
         count1=6'h0;
         addr1=4'b0000;
         addr2=4'b0001;
         c11=0;
         c21=1;
         c31=1;
         c41=1;
         c51=0;
         c61=1;
         end

         else if(count1<16)
              begin
              c11=0;
              c21=1;
              c31=1;
              c41=1;
              c51=0;
              c61=1;
              end

         else if(count1>15 && count1<32)
              begin
              c11=0;
              c21=0;
              c31=1;
              c41=0;
              c51=1;
              c61=1;
              end

         else if(count1>31 && count1<48)
              begin
              c11=0;
              c21=0;
              c31=0;
              c41=1;
              c51=1;
              c61=0;
              end

        else if(count1>47 && count1<64)
              begin
              c11=1;
              c21=1;
              c31=1;
              c41=0;
              c51=0;
              c61=0;
              end

         count1=count1+1;
         addr1=addr1+1;
         addr2=addr2+1;
   end

endmodule

// 32-bit multiplexer for the first stage commutator

module mux32(out,in0,in1,cont);

   output [dwidth-1:0]    out;
   input [dwidth-1:0]     in0,in1;
   input cont;
   reg [dwidth-1:0]       out;

   always @(in0 or in1 or cont)
   begin
```

```
    if(cont)
    out=in1;
    else
    out=in0;
   end

endmodule

// 16-bit adder for the first stage Commutator

module reg0(out,in,clk);

   output [dwidth-1:0]    out;
   input [dwidth-1:0]     in;
   input clk;
   reg [dwidth-1:0]       out;

   always @(posedge clk)
   begin
    out=in;
   end

endmodule

// Second stage commutator for the 64-point FFT processor

module comu2(clk,in,o12,o22,o32,o42,reset,c42,c52,c62,cm2,count2);

input clk,reset;
input [cwidth-2:0] count2;
input [dwidth-1:0] in;
output c42,c52,c62,cm2;
output [dwidth-1:0]o12,o22,o32,o42;
wire [dwidth-1:0] out0,A,B,C,D,E,F;
wire [cwidth-3:0] addr1,addr2;
wire [rwidth-1:0] ro;

assign cm2=ro[27]^ro[26];
assign c42= ro[28];
assign c52= ro[27];
assign c62= ro[26];

// Triple port RAM as FIFOs (Three of double size instead of six)

duramc2 ram0(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd1_addr(addr2),
     .rd2_addr(ro[10:8]), .wr_addr(addr1), .data_in(in), .data_rd1_out(B), .data_rd2_out(A));

duramc2 ram1(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd1_addr(ro[16:14]),
     .rd2_addr(ro[13:11]), .wr_addr(addr1), .data_in(B), .data_rd1_out(D), .data_rd2_out(C));

duramc2 ram2(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd1_addr(ro[19:17]),
     .rd2_addr(ro[22:20]), .wr_addr(ro[25:23]), .data_in(D), .data_rd1_out(E), .data_rd2_out(F));

rmux32 mux0c1(.out(o12), .in0(A), .in1(D),.in2(F), .sel(ro[1:0]));

rmux32 mux1c1(.out(o22), .in0(in), .in1(C),.in2(E), .sel(ro[3:2]));

rmux32 mux2c1(.out(o32), .in0(A), .in1(D),.in2(F), .sel(ro[5:4]));

muxc2  mux3c1(.out(o42), .in0(A), .in1(B),.in2(C),.in3(E), .sel(ro[7:6]));

// ROM for addressing the triple port RAMs

romc2 rom2(.out(ro), .addr(count2));

// FSM for stage 2 commutator

fsmc2 fsm12(.addr1(addr1),.addr2(addr2),.clk(clk),.reset(reset));

endmodule

// Triple port RAM module used in the second stage commutator (Two read ports and one write port)

module duramc2(clk, rst_n, cs_n, wr_n, rd1_addr,
     rd2_addr, wr_addr, data_in, data_rd1_out, data_rd2_out);

input clk,rst_n,cs_n, wr_n;
input [cwidth-3 : 0] rd1_addr,rd2_addr,wr_addr;
input [dwidth-1: 0] data_in;
output [dwidth-1 : 0] data_rd1_out,data_rd2_out;

    // Instance of DW_ram_2r_w_s_dff
    DW_ram_2r_w_s_dff #(dwidth, depth, rst_mode)
      U1 ( .clk(clk), .rst_n(rst_n), .cs_n(cs_n), .wr_n(wr_n), .rd1_addr(rd1_addr),
      .rd2_addr(rd2_addr), .wr_addr(wr_addr), .data_in(data_in), .data_rd1_out(data_rd1_out),
      .data_rd2_out(data_rd2_out));

endmodule
```

```
// Four channel multiplexer for the second stage commutator

module muxc2(out, in0, in1,in2,in3, sel);

   input [cwidth-4:0] sel;
   output [dwidth-1:0]    out;
   input [dwidth-1:0]    in0,in1,in2,in3;
   reg [dwidth-1:0]       out;

   always @(sel or in0 or in1 or in2 or in3)
begin
      case (sel)
           2'h 0: out = in0;
           2'h 1: out = in1;
           2'h 2: out = in2;
           2'h 3: out = in3;
           default : out =  32'h x;
      endcase // case(sel)
 end
endmodule //

// Three channel multiplexer for the second stage commutator

module rmux32(out, in0, in1,in2, sel);

   input [cwidth-4:0] sel;
   output [dwidth-1:0]    out;
   input [dwidth-1:0]    in0,in1,in2;
   reg [dwidth-1:0]       out;

   always @(sel or in0 or in1 or in2)
begin
      case (sel)
           2'h 0: out = in0;
           2'h 1: out = in1;
           2'h 2: out = in2;
           default : out =  4'h x;
      endcase // case(sel)
 end
endmodule //

// 33-bit ROM for storing the second stage coefficient along with a flag bit.

module romc2(out,addr);

input [cwidth-2:0] addr;
output [rwidth-1:0]    out;
reg [rwidth-1:0]       out;

   always @(addr)

      case (addr)
4'h 0:   out = 29'h 0192cb41;
4'h 1:   out = 29'h 02130c41;
4'h 2:   out = 29'h 16934d49;
4'h 3:   out = 29'h 0f1a4d29;
4'h 4:   out = 29'h 1bd3cde4;
4'h 5:   out = 29'h 0c3ecf29;
4'h 6:   out = 29'h 14359689;
4'h 7:   out = 29'h 17382089;
4'h 8:   out = 29'h 188595e5;
4'h 9:   out = 29'h 14f7dd9a;
4'h a:   out = 29'h 0ca0b5a9;
4'h b:   out = 29'h 0ccd05a9;
4'h c:   out = 29'h 18f7dde5;
4'h d:   out = 29'h 18e820e5;
4'h e:   out = 29'h 00924941;
4'h f:   out = 29'h 01128a41;
default : out = 29'h x;
      endcase // case(addr)

endmodule // ROM

// FSM for the second stage of FFT

module fsmc2(clk,addr1,addr2,reset);

output [cwidth-3:0] addr1,addr2;
reg [cwidth-3:0] addr1,addr2;
input clk,reset;

always @(posedge clk)

   begin
        if(reset) begin
                  addr1=3'b000;
                  addr2=3'b001;
                  end
```

```
        else     begin
                 addr1=addr1+1;
                 addr2=addr2+1;
                 end
   end

endmodule

// 33-bit register for holding the second stage coefficient and a flag bit

module regm(out,in,clk);

   output [dwidth:0]    out;
   input [dwidth:0]     in;
   input clk;
   reg [dwidth:0]       out;

   always @(posedge clk)
   begin
    out=in;
   end

endmodule

// Commutator for the third stage of FFT

module comu3(clk,in,o13,o23,o33,o43,reset,c43,c53,c63,cm3,count3);

input clk,reset;
input [dwidth-1:0] in;
output c43,c53,c63,cm3;
output [dwidth-1:0] o13,o23,o33,o43;
input [cwidth-1:0] count3;
wire [cwidth-3:0] addrx,addry;
wire [dwidth-1:0] out1,out2,out3,out4,out5,out6,A;
wire c13,c23,c33; wire [cwidth-1:0] count3;

assign o13= out3;

// Dual port RAM is used for restoring the data order

duramc3 ADM(.clk(clk), .rst_n(1'b1), .cs_n(1'b0), .wr_n(1'b0), .rd_addr(addrx),
      .wr_addr(addry), .data_in(in), .data_out(A));

// ROM for addressing the dual port RAM (ADM)

romc3 (.out({addrx,addry}), .addr(count3));

// Six FIFOs having unity length

reg0 reg1(.clk(clk), .in(A),.out(out1));

reg0 reg2(.clk(clk), .in(out1),.out(out2));

reg0 reg3(.clk(clk), .in(out2),.out(out3));

reg0 reg4(.clk(clk), .in(out3),.out(out4));

reg0 reg5(.clk(clk), .in(out4),.out(out5));

reg0 reg6(.clk(clk), .in(out5),.out(out6));

muxc3 mux1(.out(o23),.in0(out4),.in1(A),.cont(c13));

muxc3 mux2(.out(o33),.in0(out5),.in1(out1),.cont(c23));

muxc3 mux3(.out(o43),.in0(out6),.in1(out2),.cont(c33));

// FSM for the third stage commutator

fsmc3 fsm32(.c13(c13),.c23(c23),.c33(c33),.clk(clk),.reset(reset),.cnt(count3[1:0]),.c43(c43),
            .c53(c53),.c63(c63),.cm3(cm3));

endmodule

// Dual port RAM definition for the third stage

module duramc3(clk, rst_n, cs_n, wr_n, rd_addr, wr_addr, data_in, data_out);

parameter rst_mode = 0;

input clk,rst_n,cs_n,wr_n;
input [swidth-2 : 0] rd_addr,wr_addr;
input [dwidth-1 : 0] data_in;
output[dwidth-1 : 0] data_out;

    // Instance of DW_ram_r_w_s_dff
    DW_ram_r_w_s_dff #(dwidth, depth2, rst_mode)
      duramc3 (.clk(clk), .rst_n(rst_n), .cs_n(cs_n), .wr_n(wr_n), .rd_addr(rd_addr),
```

```
                .wr_addr(wr_addr), .data_in(data_in), .data_out(data_out));
endmodule

// FSM for the third stage of FFT

module fsmc3(c13,c23,c33,clk,reset,c43,c53,c63,cm3,cnt);

   input clk,reset;
   input [swidth-3:0]  cnt;
   output c13,c23,c33,c43,c53,c63,cm3;
   reg c13,c23,c33,c43,c53,c63;

   assign cm3 = c53^c63;

   always @(posedge clk)

   begin
        if(reset)
            begin
              c13=0;
              c23=1;
              c33=1;
              c43=1;
              c53=0;
              c63=1;
             end

         else if(cnt==0)
              begin
              c13=0;
              c23=1;
              c33=1;
              c43=1;
              c53=0;
              c63=1;
              end

         else if(cnt==1)
              begin
               c13=0;
               c23=0;
               c33=1;
               c43=0;
               c53=1;
               c63=1;
              end

         else if(cnt==2)
              begin
              c13=0;
              c23=0;
              c33=0;
              c43=1;
              c53=1;
              c63=0;
              end

        else
              begin
              c13=1;
              c23=1;
              c33=1;
              c43=0;
              c53=0;
              c63=0;
              end
   end

endmodule

// 16-bit multiplexer for the third stage

module muxc3(out,in0,in1,cont);

   output [dwidth-1:0]    out;
   input [dwidth-1:0]     in0,in1;
   input cont;
   reg [dwidth-1:0]       out;

   always @(in0 or in1 or cont)

   begin
    if(cont)
    out=in1;
    else
    out=in0;
   end
```

```
endmodule

// ROM contents for generating the read and write addresses for ADM in the third stage
   of the 64-point FFT processor.

module romc3(out, addr);

input [cwidth-1:0] addr;
output [cwidth:0]    out;
reg [cwidth:0]       out;

   always @(addr)

      case (addr)

5'h 0:  out = 6'h 21;
5'h 1:  out = 6'h 12;
5'h 2:  out = 6'h 33;
5'h 3:  out = 6'h 2c;
5'h 4:  out = 6'h 3d;
5'h 5:  out = 6'h 06;
5'h 6:  out = 6'h 0f;
5'h 7:  out = 6'h 10;
5'h 8:  out = 6'h 19;
5'h 9:  out = 6'h 22;
5'h a:  out = 6'h 03;
5'h b:  out = 6'h 1c;
5'h c:  out = 6'h 08;
5'h d:  out = 6'h 29;
5'h e:  out = 6'h 23;
5'h f:  out = 6'h 3c;
5'h 10:  out = 6'h 05;
5'h 11:  out = 6'h 36;
5'h 12:  out = 6'h 17;
5'h 13:  out = 6'h 08;
5'h 14:  out = 6'h 19;
5'h 15:  out = 6'h 22;
5'h 16:  out = 6'h 2b;
5'h 17:  out = 6'h 34;
5'h 18:  out = 6'h 3d;
5'h 19:  out = 6'h 06;
5'h 1a:  out = 6'h 27;
5'h 1b:  out = 6'h 38;
5'h 1c:  out = 6'h 2c;
5'h 1d:  out = 6'h 0d;
5'h 1e:  out = 6'h 07;
5'h 1f:  out = 6'h 18;

default : out = 6'h x;
      endcase // case(addr)

endmodule // ROM

// ROM for storing the first stage coefficient

module roml(out, addr);

   input [counter_width-1:0] addr;
   output [dwidth-1:0]    out;
   reg [dwidth-1:0]       out;

   always @(addr)

      case (addr)
6'h  0: out = 32'h     7fff0000;
6'h  1: out = 32'h     7fff0000;
6'h  2: out = 32'h     7f61f373;
6'h  3: out = 32'h     7d89e706;
6'h  4: out = 32'h     7a7cdad7;
6'h  5: out = 32'h     7641cf04;
6'h  6: out = 32'h     70e2c3a9;
6'h  7: out = 32'h     6a6db8e3;
6'h  8: out = 32'h     62f1aecc;
6'h  9: out = 32'h     5a82a57d;
6'h  a: out = 32'h     51339d0e;
6'h  b: out = 32'h     471c9592;
6'h  c: out = 32'h     3c568f1d;
6'h  d: out = 32'h     30fb89be;
6'h  e: out = 32'h     25288583;
6'h  f: out = 32'h     18f98276;
6'h  10: out = 32'h     0c8c809e;
6'h 11: out = 32'h     7fff0000;
6'h 12: out = 32'h     7d89e706;
6'h 13: out = 32'h     7641cf04;
6'h 14: out = 32'h     6a6db8e3;
6'h 15: out = 32'h     5a82a57d;
6'h 16: out = 32'h     471c9592;
6'h 17: out = 32'h     30fb89be;
6'h 18: out = 32'h     18f98276;
```

```
6'h 19: out = 32'h      00008000;
6'h 1a: out = 32'h      e7068276;
6'h 1b: out = 32'h      cf0489be;
6'h 1c: out = 32'h      b8e39592;
6'h 1d: out = 32'h      a57da57d;
6'h 1e: out = 32'h      9592b8e3;
6'h 1f: out = 32'h      89becf04;
6'h 20: out = 32'h      8276e706;
6'h 21: out = 32'h      7fff0000;
6'h 22: out = 32'h      7a7cdad7;
6'h 23: out = 32'h      6a6db8e3;
6'h 24: out = 32'h      51339d0e;
6'h 25: out = 32'h      30fb89be;
6'h 26: out = 32'h      0c8c809e;
6'h 27: out = 32'h      e7068276;
6'h 28: out = 32'h      c3a98f1d;
6'h 29: out = 32'h      a57da57d;
6'h 2a: out = 32'h      8f1dc3a9;
6'h 2b: out = 32'h      8276e706;
6'h 2c: out = 32'h      809e0c8c;
6'h 2d: out = 32'h      89be30fb;
6'h 2e: out = 32'h      9d0e5133;
6'h 2f: out = 32'h      b8e36a6d;
6'h 30: out = 32'h      dad77a7c;
6'h 31: out = 32'h      7fff0000;
6'h 32: out = 32'h      7fff0000;
6'h 33: out = 32'h      7fff0000;
6'h 34: out = 32'h      7fff0000;
6'h 35: out = 32'h      7fff0000;
6'h 36: out = 32'h      7fff0000;
6'h 37: out = 32'h      7fff0000;
6'h 38: out = 32'h      7fff0000;
6'h 39: out = 32'h      7fff0000;
6'h 3a: out = 32'h      7fff0000;
6'h 3b: out = 32'h      7fff0000;
6'h 3c: out = 32'h      7fff0000;
6'h 3d: out = 32'h      7fff0000;
6'h 3e: out = 32'h      7fff0000;
6'h 3f: out = 32'h      7fff0000;

default : out = 32'h       xxxxxxx;
      endcase // case(addr)

endmodule // ROM

// ROM for storing the second stage coefficient

`define dwidth 32
`define swidth 4        // counter word size

module rom2(out, addr);

parameter dwidth = `dwidth;
parameter swidth = `swidth;

   input [swidth-1:0] addr;
   output [dwidth:0]    out;
   reg [dwidth:0]       out;

   always @(addr)
      case (addr)

4'h 0:    out = 33'h 180010000;
4'h 1:    out = 33'h 180010000;
4'h 2:    out = 33'h 180010000;
4'h 3:    out = 33'h 180010000;
4'h 4:    out = 33'h 180010000;
4'h 5:    out = 33'h 000008000;
4'h 6:    out = 33'h 07641cf04;
4'h 7:    out = 33'h 1cf0589be;
4'h 8:    out = 33'h 1cf0589be;
4'h 9:    out = 33'h 05a82a57d;
4'h a:    out = 33'h 05a82a57d;
4'h b:    out = 33'h 15a83a57d;
4'h c:    out = 33'h 15a83a57d;
4'h d:    out = 33'h 1764230fb;
4'h e:    out = 33'h 180010000;
4'h f:    out = 33'h 180010000;

default : out = 33'h xxxxxxx;
      endcase // case(addr)

endmodule // ROM
```

D.2 Verilog code for the Combiner

```
// Top level Combiner module

'define width 16
'define depth3  64
'define c_inputs 3
'define cwidth 5

module Combiner(clk,in,acc,reset,lambda);

parameter depth3  = 'depth3;
parameter cwidth = 'cwidth;
parameter width = 'width;
parameter c_inputs = 'c_inputs;

input clk,reset;
input [width+width-1:0] in;
output [width-1:0] acc;
input [width-1:0] lambda;
wire c11,c12,c13,c21,enf,enr,csram;
wire [width-1:0] xdr,xdi,emr,emi,eqr,eqi,acco,mor,moi;
wire [cwidth:0] waddr,raddr;
wire x;

// Multiplication module

block1 blocka(.mor(mor),.moi(moi),.xdr(xdr),.xdi(xdi),.xr(in[31:16]),.xi(in[15:0]),.emr(emr),.emi(emi),
              .c11(c11),.c12(c12),.c13(c13),.clk(clk));

// Accumulation and summing module

block2 blockb(.mor(mor),.moi(moi),.accso(acco),.c21(c21),.clk(clk),.acco(acc));

// Division module

block3 blockc(.eqr(eqr),.eqi(eqi),.xdr(xdr),.xdi(xdi), .acci(acco),.lambda(lambda),.clk(clk),.enf(enf),
              .enr(enr));

// Memory for storing the equaliser coefficients

cduram mem(.clk(clk), .rst_n(1'b1), .cs_n(csram), .wr_n(1'b0), .rd_addr(raddr), .wr_addr(waddr),
           .data_in({eqr,eqi}), .data_out({emr,emi}));

// FSM for the Combiner

cfsm  fsm1(.clk(clk), .c11(c11),.c12(c12),.c13(c13),.c21(c21),.enf(enf),.enr(enr),.csram(csram),

           .waddr(waddr),.raddr(raddr),.reset(reset));

endmodule

// FSM for the Combiner

module cfsm(clk, c11, c12, c13, c21, enf, enr,csram,waddr,raddr, reset);

   output [cwidth:0]  raddr,waddr;
   output c11,c12,c13,c21,enf,enr,csram;
   reg c11,c12,c13,c21,enf,enr,csram;
   input clk,reset;

parameter pilot=1'b0;
parameter data =1'b1;

   reg  ps,ns;
   reg [cwidth:0] carrier_count,next_carrier_count;
   reg [cwidth-1:0] symbol_count,next_symbol_count;

   assign waddr=carrier_count;
   assign raddr= carrier_count+3;

always @(posedge clk)
  begin
   if(reset)
        begin
        ps=pilot;
        carrier_count=0;
        symbol_count=0;
        end
  else  begin
        ps=ns;
        carrier_count=next_carrier_count;
        symbol_count= next_symbol_count;
        end
end

always @(ps or carrier_count or symbol_count) begin
```

```
    case(ps)
    pilot:begin

          c11 =1'b0;
          c12 =1'b0;
          c13 =1'b0;
          csram=1'b0;
          enf = 1'b1;
          enr = 1'b1;
          next_carrier_count = carrier_count + 1;

          if(carrier_count < 2)
             c21=1'b1;
          else
             c21=1'b0;

          if(carrier_count < 3)
              csram=1'b1;
          else
              csram=1'b0;

        if(carrier_count < 63)
          begin
            next_symbol_count = symbol_count;
            ns=pilot;
          end
        else
          begin
           ns=data;
           next_symbol_count = symbol_count+1;
          end
        end

    data: begin
      c11 =1'b1;
          c12 =1'b1;
          enf=1'b1;
          enr=1'b1;
          next_carrier_count = carrier_count+1;

       if(symbol_count == 1)
             if (carrier_count<3)
              begin
                enf=1'b1;
                enr=1'b1;
                csram=1'b0;
                c21=1'b0;
                 if(carrier_count < 1)
                     c13 = 1'b0;
                 else
                     c13 = 1'b1;
              end
            else
              begin
               c13=1'b1;
               csram=1'b1;
               c21=1'b1;
               enf=1'b0;
               enr=1'b0;
              end
       else
             begin
             c13=1'b1;
             csram=1'b1;
             enf=1'b0;
             enr=1'b0;
             if(carrier_count==2)
                  c21=1'b0;
             else
                  c21=1'b1;
             end

          if (symbol_count==31 && carrier_count==63)
             ns =pilot;
          else
             ns =data;

          if (carrier_count < 63)
              next_symbol_count = symbol_count;
          else
              next_symbol_count = symbol_count + 1;
        end
  endcase
end

endmodule

// Dual port RAM for storing the coefficients
```

```
module cduram(clk, rst_n, cs_n, wr_n, rd_addr, wr_addr, data_in, data_out);

input clk,rst_n,cs_n,wr_n;
input [cwidth : 0] rd_addr,wr_addr;
input [width+width-1 : 0] data_in;
output[width+width-1 : 0] data_out;

    // Instance of DW_ram_r_w_s_dff
    DW_ram_r_w_s_dff #(width+width, depth3, rst_mode)
     cduram (.clk(clk), .rst_n(rst_n), .cs_n(cs_n), .wr_n(wr_n), .rd_addr(rd_addr), .wr_addr(wr_addr),
       .data_in(data_in), .data_out(data_out));

endmodule

// Verilog code for Block1 of Combiner

module block1(mor,moi,xdr,xdi,xr,xi,emr,emi,c11,c12,c13,clk);

   output [width-1:0]    mor, moi,xdr,xdi;
   input [width-1:0]     xr,xi,emr,emi;
   input clk,c11,c12,c13;
   wire [width-1:0] m1,m2,m3,m4,xro,xio,PI;
   wire [width+width-1:0] PRODUCTR,PRODUCTI;


 assign  mor = PRODUCTR[width+width-2:width-1];
 assign  PI  = PRODUCTI[width+width-2:width-1];
 assign xdr = m1;
 assign xdi = m3;
 cmux16 muxr(.out(xro),.in0(xr),.in1(emr),.cont(c11));
 cmux16 muxi(.out(xio),.in0(xi),.in1(emi),.cont(c12));
 creg16 regr(.out(m1),.in(xr),.clk(clk));
 creg16 eqr(.out(m2),.in(xro),.clk(clk));
 creg16 regi(.out(m3),.in(xi),.clk(clk));
 creg16 eqi(.out(m4),.in(xio),.clk(clk));
 mult  multr(.A(m1), .B(m2), .TC(1'b1), .PROD(PRODUCTR));
 mult  multi(.A(m3), .B(m4), .TC(1'b1), .PROD(PRODUCTI));
 caddon compli(.A(PI), .SI(c13), .B(moi));

 endmodule

// Control inversion module for Block1 of Combiner

module caddon(A, SI, B);

input [width-1 : 0] A;
input SI;
output [width-1 : 0] B;
reg [width-1 : 0] B;

  always @(A or SI)
  begin
   if(SI)
     B = ~A;
   else
     B = A;
   end
endmodule

// 16-bit multiplexer for Block1 of Combiner

module cmux16(out,in0,in1,cont);

   output [width-1:0]    out;
   input [width-1:0]     in0,in1;
   input cont;
   reg [width-1:0]       out;

   always @(in0 or in1 or cont)
   begin
    if(cont)
    out=in1;
    else
    out=in0;
   end

endmodule

// 16-bit register for Block1 of Combiner

module creg16(out,in,clk);

   output [width-1:0]    out;
   input [width-1:0]     in;
   input clk;
   reg [width-1:0]       out;

   always @(posedge clk)
   begin
```

```
    out=in;
   end

endmodule

// Verilog code for Block2 of the Combiner

module block2(mor,moi,accso,acco,c21,clk);

   input [width-1:0]    mor,moi;
   output [width-1:0]     accso,acco;
   input clk,c21;
   wire [width-1:0] sr,si,mo,accio;

  assign accso = {4'h0,accio[width-1:4]};

  creg16 reg21(.out(sr),.in(mor),.clk(clk));
  creg16 reg22(.out(si),.in(moi),.clk(clk));
  csum  summer(.INPUT({sr,si,mo}), .SUM(accio));
  creg16 regacc(.out(acco),.in(accio),.clk(clk));
  cmux16 mux21(.out(mo),.in0(16'h0000),.in1(acco),.cont(c21));
 endmodule

// Summer for Block2 of the Combiner

module csum(INPUT, SUM);

input [c_inputs*width-1 : 0] INPUT;
output [width-1 : 0] SUM;

    // Instance of DW02_sum
    DW02_sum #(c_inputs, width)
      U1 (.INPUT(INPUT), .SUM(SUM));

endmodule

// Verilog code for Block3 of the Combiner

module block3(eqr,eqi,xdr,xdi,acci,lambda,clk,enf,enr);

   output [width-1:0]    eqr,eqi;
   input [width-1:0]     xdr,xdi,acci,lambda;
   input clk,enf,enr;
   wire [width-1:0] deno,den,fr,fi,fmi,ieqr,ieqi;

   assign eqr={ieqr[width-6:0],5'h0};
   assign eqi={ieqi[width-6:0],5'h0};
   assign fmi= ~fi;

   cadd  adder(.A(acci),.B(lambda),.CI(1'b0),.S(den));
   reg16E regR(.out(deno),.in(den),.clk(clk),.enr(enr));
   fifo fifo3(.out({fr,fi}),.in({xdr,xdi}),.clk(clk),.enf(enf));
   divide dividerr( .A(fr), .B(deno), .TC(1'b1), .QUOTIENT(ieqr));
   divide divideri( .A(fmi), .B(deno), .TC(1'b1), .QUOTIENT(ieqi));

endmodule

// FIFO module for Block3 of Combiner

module fifo(out,in,clk,enf);

   output [width+width-1:0]    out;
   input [width+width-1:0]     in;
   input clk,enf;
   wire enclk;
   wire [width+width-1:0]     i1;

   creg32 regf0(.out(i1),.in(in),.clk(clk),.enf(enf));
   creg32 regf1(.out(out),.in(i1),.clk(clk),.enf(enf));

  endmodule

// Register with enable for Block3 of Combiner

module reg16E(out,in,clk,enr);

   output [width-1:0]    out;
   input [width-1:0]     in;
   input clk,enr;
   reg [width-1:0]       out;

   always @(posedge clk)
   begin
     if (enr)
     out=in;
   end

endmodule
```

```
// Adder for Block3 of Combiner

module cadd(A,B,CI,S,CO);

input [width-1:0] A,B;
output [width-1:0] S;
input CI;
output CO;

DW01_add #(width) cadd(.A(A),.B(B),.CI(CI),.SUM(S),.CO(CO));

endmodule

// 32-bit register for FIFO module of Block3

module creg32(out,in,clk);

   output [width+width-1:0]    out;
   input [width+width-1:0]     in;
   input clk;
   reg [width+width-1:0]       out;

   always @(posedge clk)
   begin
     out=in;
   end

endmodule

// Divider module for Block3 of Combiner

module divide( A, B, TC, DIVIDE_BY_0, QUOTIENT );

parameter TC_mode = 1;

input [width-1:0] A,B;
input TC;
output DIVIDE_BY_0;
output [width-1:0] QUOTIENT;

    // Instance of DW02_divide
    DW02_divide #(width, width, TC_mode)
      U1 ( .A(A), .B(B), .TC(TC), .DIVIDE_BY_0(DIVIDE_BY_0), .QUOTIENT(QUOTIENT) );

endmodule
```