International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 12 (2015) pp. 32233-32258

http://www.ripublication.com

An Efficient Multipath Delay Commutator FFT Processor for MIMO-OFDM Systems

Mr.S.P.Valan Arasu[1], K.Saranya[2], Dr.S.Baul Kani[3]

[1]Associate Professor & ECE Department,
Dr.Sivanthi Aditanar College of Engineering,Tiruchendur.
[2]PG Scholar & Dr.Sivanthi Aditanar College of Engineering,Tiruchendur.
[3]Associate Professor & ECE Department,
Government College of Engineering,Tirunelveli.
[1]kakkurichi@yahoo.com [2]saranlakshmi92@gmail.com
[3]ramabaulkani@yahoo.co.in

Abstract

Fast Fourier Transform (FFT) is an important transformation used in many applications of digital signal processing and communication systems. In orthogonal frequency division multiplexing, Inverse fast Fourier transform (IFFT) converts the modulated information from frequency domain to time domain for transmission of radio signals, while FFT gathers samples from the time domain, transforming them to the frequency domain. With multiple input multiple output (MIMO) devices, data throughput can be increased dramatically. Hence MIMO-OFDM systems provide promising data rate and reliability in wireless communication. The proposed FFT processor uses multipath delay commutator (MDC) and memory scheduling methods to handle multiple input streams which results in full utilization in memory usage. The inputs to the FFT processor are scheduled by using multipath delay commutator. The proposed memory scheduling scheme can effectively reduce the computation time for collecting input data of the FFT processor from the memory. The proposed FFT processor uses floating point arithmetic to improve the resolution and accuracy. Since, floating point arithmetic takes little larger computation time, in order to reduce the computation time pipelined floating point arithmetic is introduced. The proposed FFT processor is adopted for variable length such as 2048, 1024, 512, 128 for MIMO-OFDM systems.

IndexTerms-Fast Fourier Transform (FFT), memory scheduling, multiple input multiple output, orthogonal frequency division multiplexing, pipeline multipath delay commutator.

I. INTRODUCTION

Fast Fourier Transform algorithms are mathematical simplifications of the Discrete Fourier Transform (DFT). They exploit symmetries and periodicity in the transform in order to reduce the number of mathematical computations. There have since been many variations of this algorithm aimed at reducing the complexity of the DFT calculations. These families of fast algorithms for computing the DFT are commonly known as FFT algorithms. Fast Fourier transform (FFT) is a crucial block in orthogonal frequency division multiplexing (OFDM) systems. OFDM has been adopted in a wide range of applications from wired-communication modems, such as digital subscriber lines (xDSL) to wireless-communication modems, such as IEEE802.11 WiFi, IEEE802.16 WiMAX or 3GPPlong term evolution (LTE), to process baseband data. Pipeline FFTs are a class of parallel algorithms that contain an amount of parallelism equal to $\log_r N$ where N is the number of points for an FFT and r is the radix. A Pipeline implementation consists of a series of computational blocks each composed of delay lines, coefficient storage, commutators, multipliers, and adders. Pipeline FFT processor consists of butterfly unit, twiddle factor multiplication unit and buffers. Butterfly unit carries out addition and subtraction. Twiddle factor multiplied into the butterfly unit result. The main structures of pipeline FFT processor are Single-path Delay Feedback (SDF), Multiple-path Delay Commutator (MDC), and Single-path Delay Commutator (SDC). Each structure is suitable for application in some specific areas. For example, the MDC structure is suitable for MIMO communications, and the SDF structure is suitable for reconfigurable FFT processors. SDF schemes provide feedback paths to manage partially computed results in each pipe and to generate seamless output without delay.

II. MDC ARCHITECTURE FOR MIMO FFT/IFFT

In the radix 4 MDC architecture input sequence is divided into four parallel data streams, and then proper delay of three of four streams butterfly operation and twiddle factor multiplications are executed. Storage elements dominate most of the area in conventional MDC architecture. That is, the input buffering stage for radix-4 based FFT/IFFT needs $N/4+ N/2+3N/4$ words of memory, and each computing stage needs $3N/4s$ words of memory, where s is the stage index. For a 2048-point MDC FFT/IFFT processor, 5112 words of memory are required. If MDC is applied in MIMO-OFDM systems, the memory size grows linearly with the number of data streams. As for the utilization rate of butterflies and multipliers, since 3/4 of the computing time is used to gather the input data, the utilization rate is only 25% in single stream radix-4 MDC FFT/IFFT. However, for MIMO FFT/IFFT, we found that if the data streams are properly scheduled, the utilization rate can increase from 25% to 100%. This makes MDC very suitable for MIMO-OFDM systems. As for the utilization rate of butterflies and multipliers, each one of the four input symbols after memory scheduling takes 25% of one symbol time for radix-4 butterfly computation. Consequently one radix-4 butterfly in each pipeline stage can process four data streams without any idle period, that is, the utilization rate of butterflies and multipliers is 100%. Furthermore, the radix-8 butterfly at the last stage can be

configured as a radix-4 butterfly. With such flexibility, radix-2 computation can be incorporated at the last radix-8 stage, and thus for any *N* in power-of-2 fashion can be computed with this proposed method. Finally, the serial blocks of output symbol format helps to reduce the memory usage for output sorting and the complexity of the modules followed by the FFT/IFFT processor. Figure 3 shows the block diagram of the proposed MIMO FFT/IFFT computing core with $N = 2048$.

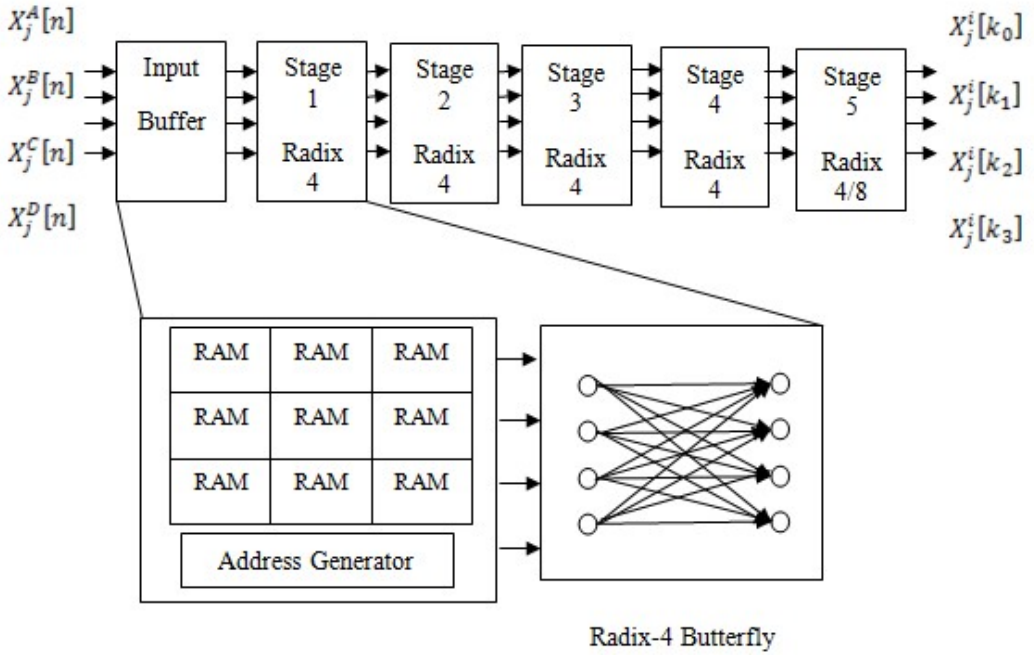


Figure 1. Block diagram of the proposed MIMO FFT Processor

A. Input Memory Scheduling

The goal is to convert the input streams in Fig. 2 (a) to the format in Fig. 2 (b). There are 12 memory banks at the input stage for converting the parallel input streams into serial blocks, such that one butterfly at each stage can compute the four data streams without idle period.

.....	1	0	2047	.....	2	1	0	A
.....	1	0	2047	.....	2	1	0	B
.....	1	0	2047	.....	2	1	0	C
.....	1	0	2047	.....	2	1	0	D

Symbol j+1 Symbol j Time index

(a)

511	.....	1	0	511	.....	1	0
1023	.....	513	512	1023	.....	513	512
1535	.....	1025	1024	1535	.....	1025	1024
2047	.....	1537	1536	2047	.....	1537	1536

STREAM B STREAM A Time index

(b)

Figure 2 (a) Initial input order (b) Sorted input order at the output of input buffer

The 16 memory banks are grouped into four memory sets as shown in Figure 3, that is, memory sets a, b, c, d which are used to store the input streams, A,B,C,D respectively. The 16 memory banks are logically grouped into four sets {a1, a2, a3, a4}, {b1, b2, b3, b4}, {c1, c2, c3, c4}and {d1, d2, d3, d4} as shown in Figure 3. For the case of N=2048, the memory banks {a1, a2, a3, a4}, {b1, b2, b3, b4}, {c1,c2,c3, c4},and {d1,d2,d3,d4} store the samples 1th-512th, 513th-1024th,1025th-1536th,1537th-2048th of the first, the second, the third, and the fourth input streams respectively.

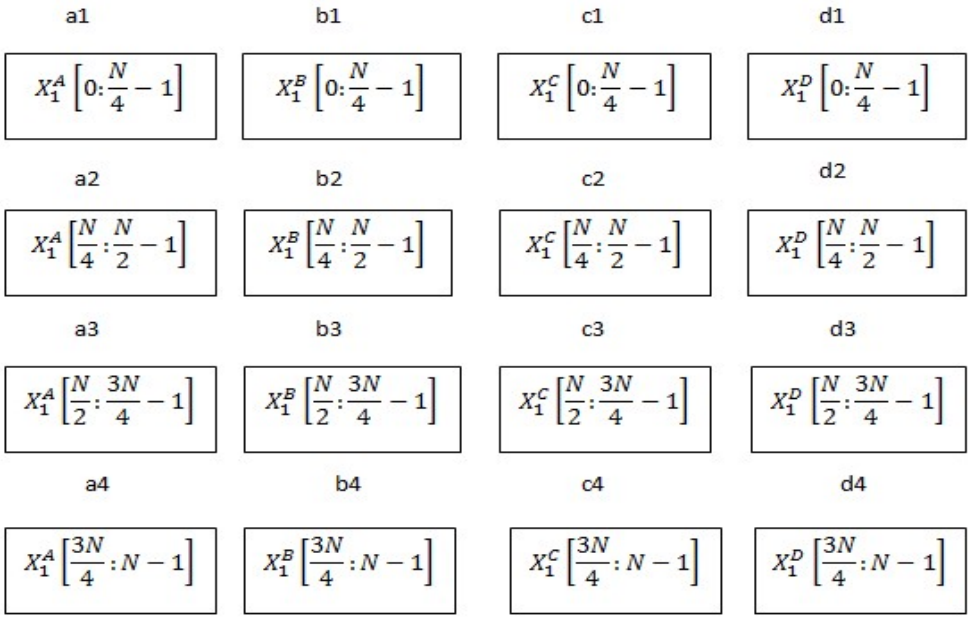
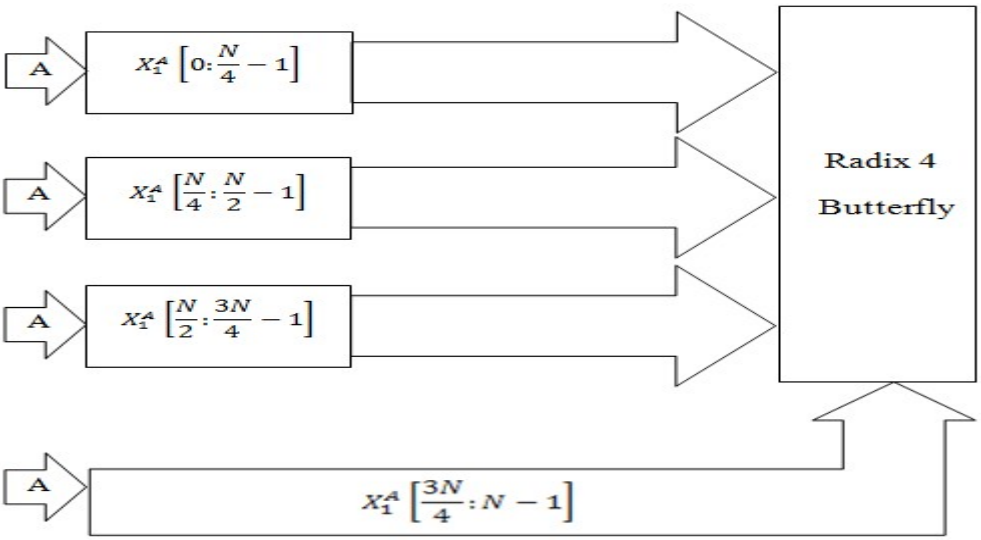
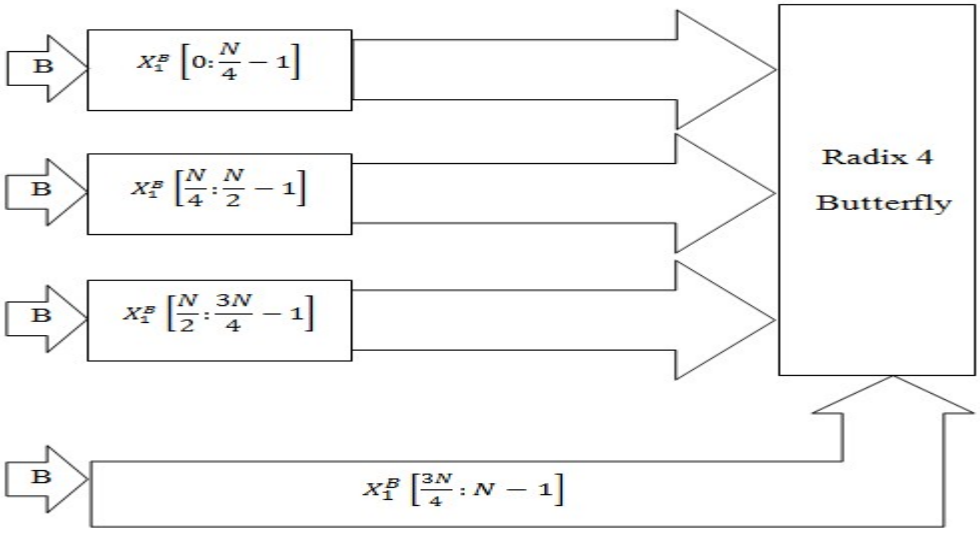


Figure 3. Logic Group of Initial Memory Banks

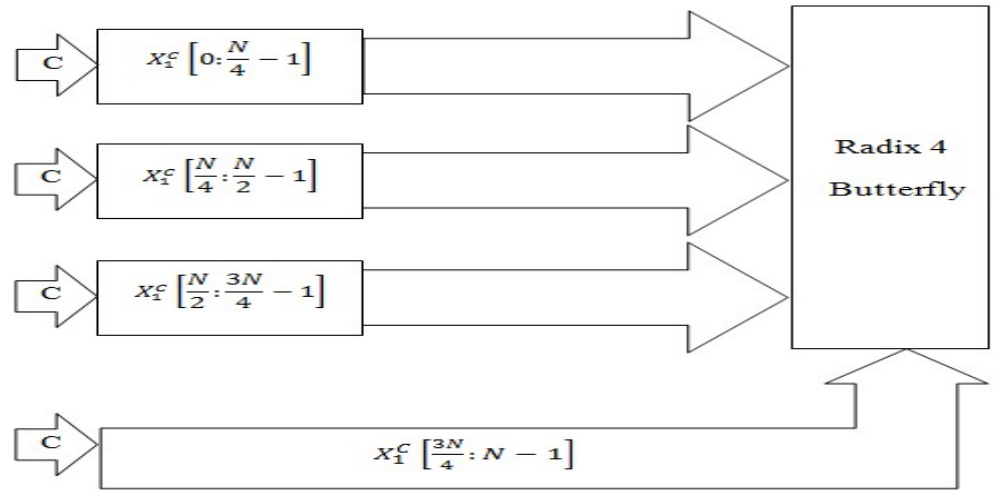
In the first clock cycle, stage 1 radix 4 butterfly process the input data from the memory bank{a1,a2,a3,a4} i.e stage1 radix 4 butterfly process 0 to N-1 samples of stream A is shown in figure 4 (a). In the second clock cycle, stage 2 radix 4 butterfly process the input data from the memory bank{b1,b2,b3,b4} i.e stage2 radix 4 butterfly process 0 to N-1 samples of stream B shown in figure 4 (b). In the third clock cycle, stage 3 radix 4 butterfly process the input data from the memory bank{c1,c2,c3,c4} i.e stage1 radix 4 butterfly process 0 to N-1 samples of stream C is shown in figure 4 (c). In the fourth clock cycle, stage 4 radix 4 butterfly process the input data from the memory bank{d1,d2,d3,d4} i.e stage 4 radix 4 butterfly process 0 to N-1 samples of stream D is shown in figure 4 (d).



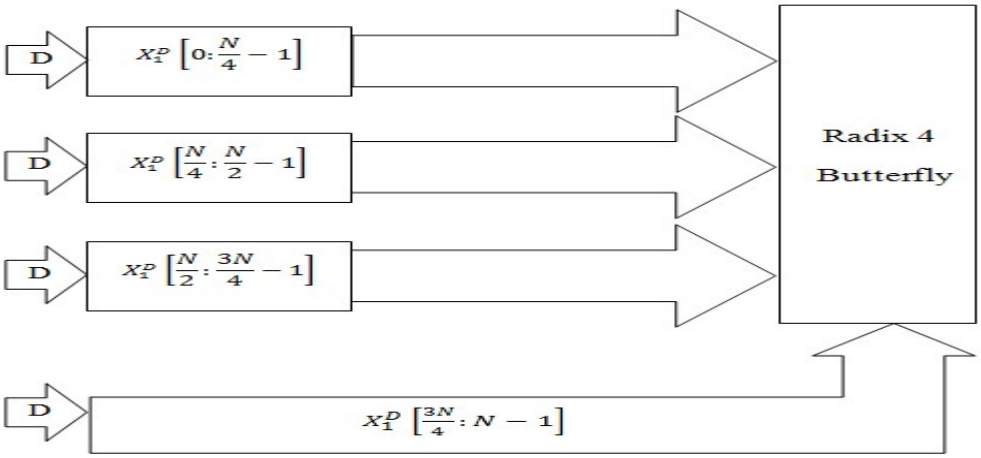
(a)



(b)



(c)



(d)

Figure 4. Input memory scheduling (a) Memory Scheduling for stream A Figure (b) Memory Scheduling for stream B Figure (c) Memory Scheduling for stream C Figure (d) Memory Scheduling for stream D

B. Butterfly Operations

The proposed FFT/IFFT processor uses radix-4 butterflies as fundamental computing elements. Each stage adopts the same radix-4 butterfly, while the last stage uses a radix-8 butterfly which can also be configured as a radix-4 butterfly. As for the complex multiplications, each radix-4 butterfly needs three multipliers and five real adders. The last stage uses radix-8/radix-4 butterfly, where the multiplications of twiddle factor can be realized by constant multipliers. This butterfly is composed by one radix-4 and four radix-2 butterflies. When a radix-4 instead of a radix-8

computation is needed, this butterfly enables only the internal radix-4 computations. The radix-4 FFT recursively partitions a DFT into four quarter-length DFTs of groups of every fourth time sample. The radix-4 FFTs require only 75% as many complex multiplications as the radix-2 FFTs. Figure 5 shows the butterfly diagram for radix-4 FFT. The radix-4 decimation-in-time and decimation-in-frequency Fast Fourier transforms (FFTs) gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs. The length-N DFT can be computed as the sum of the outputs of four length-N/4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors W_N^n, W_N^{2n}, W_N^{3n}.

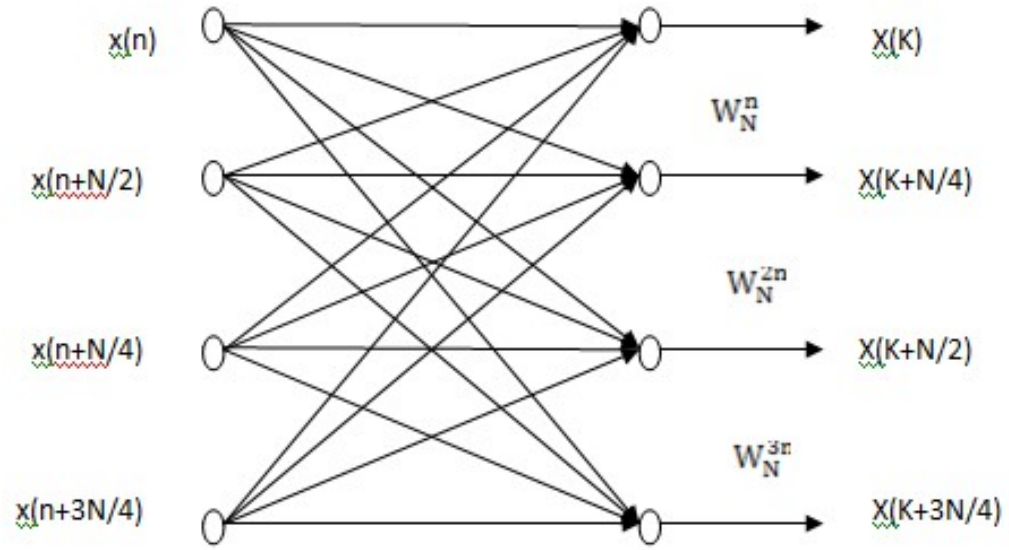


Figure 5. Butterfly Diagram for radix-4 FFT

III. COMPLEX MULTIPLIER IN RADIX-4 FFT

Most tedious part in FFT is the complex multiplication. Complex numbers are divided into two parts real and imaginary.

A. Complex multiplier with 4 multipliers and 2 adders/subtractors

a+jb is a complex number which is multiplied by another complex number c+jd, we will get

(a*c)-(b*d)+ j (b*c+ a*d) (1)

There are four multiplication and two additions are involved. It requires larger chip area in hardware implementation as shown in Figure 6

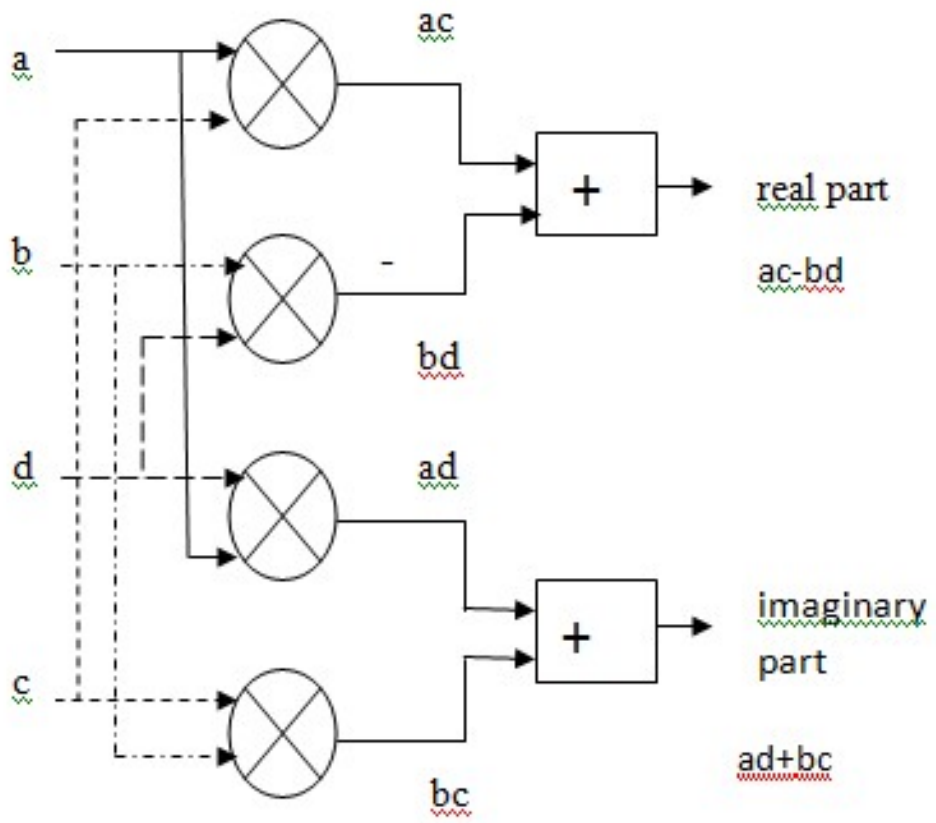


Figure 6. Complex multiplier with 4 multipliers and 2adders/subtractors

B. Complex multiplier with 3 multipliers and 5 adders/subtractors

Using subexpression elimination method in equation (1)

(a*c)-(b*d)+ j (b*c+a*d)= (a*c-b*c + b*c-b*d) + j (b*c+b*d-b*d+a*d)

={ (a*c – b*c) + (b*c-b*d)}+ j{ (a*d+b*d) + (b*c-b*d)}

(a*c)-(b*d)+j (b*c+a*b)= (c* (a-b) + b* (c-d))+j (d* (a+b)+b* (c-d)) (2)

Equation (2) shows b* (c-d) term computed only once, only three real multipliers and 5 adder/subtractors are required instead of four real multipliers and 2 adders/subtractors as shown in figure7.Complex multiplier with four multipliers and two adders/subtractors requires larger chip area in hardware implementation. Complex multiplier with three multiplier and adders/subtractors requires smaller chip area in hardware implementation compared to four multiplier and two adders/subtractors. As for complex multiplication in FFT processor complex multiplier with three multiplier and five adders/subtractors are used.

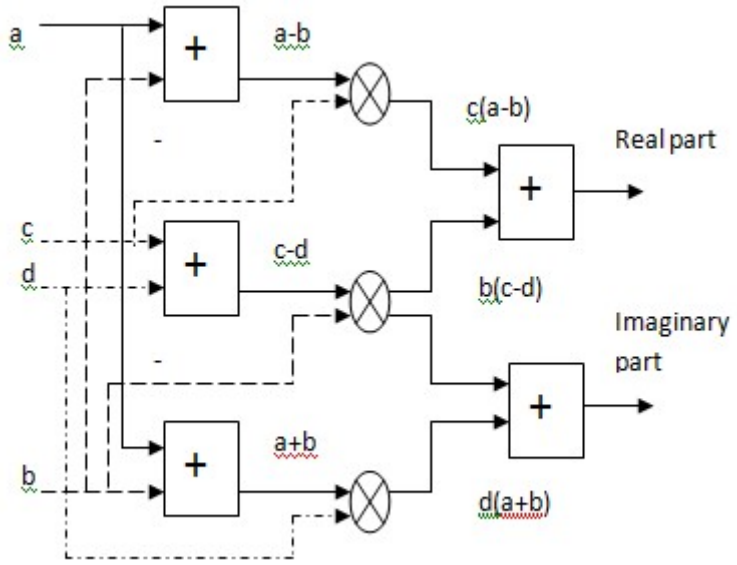


Figure 7. Complex multiplier with 3 multipliers and 5 adders/subtractors

IV FLOATING POINT ARITHMETIC

Floating point representation has its advantages of its resolution and accuracy compared to fixed point number representation. Numbers in the floating point are represented in the form of bit string. This bit string is combination of sign bit, mantissa and exponent power. This representation is called IEEE 754 standard. The single precision of floating Point is shown in figure8.

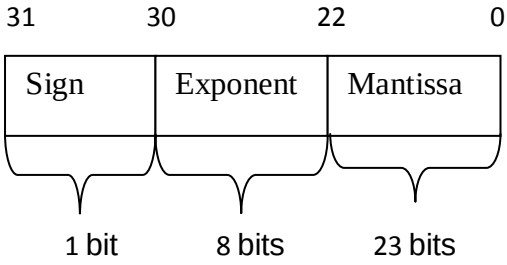


Figure 8. IEEE 754 single precision floating point format

Floating point number consists of three fields:

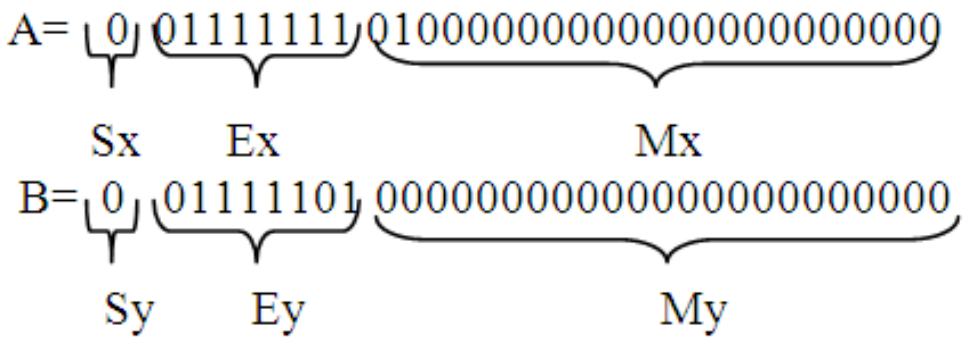
1. Sign (S): It is used to denote the sign of the number i.e. 0 represent positive number and 1 represent negative number.
2. Mantissa (M): Mantissa is part of a floating point number which represents the magnitude of the number.
3. Exponent (E): Exponent is part of the floating point number that represents the number of places that the decimal point (or binary point) is to be moved.

A.Steps required to carry out floating point addition/subtraction

1) Allign sign bit,exponent bits,mantissa bits.
2) Compute the sign of the result S=Sx+Sy.
3) The exponent of the operands must be made equal for addition and subtraction.
4) Compare exponents if Ey>Ex Right shift Mx by Ex-Ey, if Ex>Ey Right shift My by Ey-Ex.
5) Append hidden bit 1 in the msb of the mantissa bits.
6) Add or subtract the mantissa bits to get sum result/subtraction result.
7) Normalize the result, the msb of the result is 1,then right shift result and increment result exponent otherwise no shift is required.
8) Check result,overflow/underflow, if result mantissa is 0,may need to set the exponent to zero to return a zero.
9) Round the appropriate number of bits.

Floating point Addition Example:

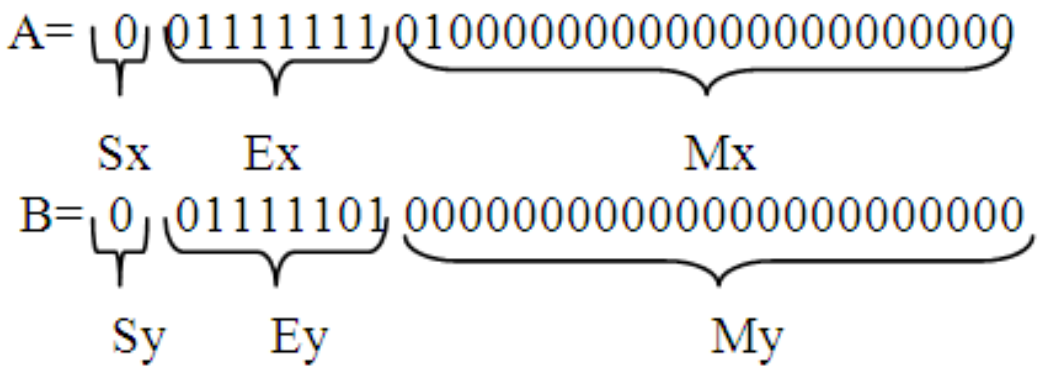
Consider the floating point addition 1.25+0.25



Append 01 (hidden bit) in the mantissa bits 01010000000000000000000 (Mx larger) 01000000000000000000000 (My smaller)
Ex>Ey Right shift smaller mantissa by exponential difference we get,
00010000000000000000000 (Mr)
ADD Mr+Mx
01010000000000000000000 (Mx)
00010000000000000000000 (Mr)

01100000000000000000000 (Ms)
Check carry =0
No need to normalize
Remove the hidden bit 01 from (Ms)
exponent=larger exponent
Final Answer:
{0 01111111 10000000000000000000000}=1.50

Floating point subtraction example:
Consider the floating point subtraction 1.25-0.25



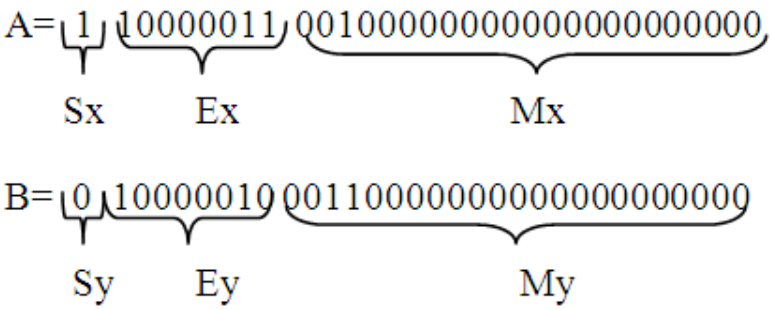
Append 01 (hidden bit) in the mantissa bits 01010000000000000000000 (Mx larger) 01000000000000000000000 (My smaller)
Ex>Ey Right shift smaller mantissa by exponential difference we get,
00010000000000000000000 (Mr)
SUBTRACT Mr-Mx
01010000000000000000000 (Mx)
00010000000000000000000 (Mr)

01000000000000000000000 (Ms)
Check carry =0
No need to normalize
Remove the hidden bit 01 from (Ms)
exponent=larger exponent
Final Answer:
{0 01111111 00000000000000000000000} (1.00)

B. Steps required to carry out Floating point multiplication:

1) Allign sign bit,exponent bits,mantissa bits.
2) Compute the sign of the result Sx xor Sy.
3) Add exponents,biased exponent (Ex)+biased exponent (Ey)-bias.
4) Append hidden bit 1 in the msb of the mantissa bits.
5) MultiplymantissaMx*My.
6) Round the result to the allowed number of mantissabits.
7) Normalize the result, the msb of the result is 1,then right shift result and increment result exponent otherwise no right shift is required.

Floating point multiplication example:
Consider the floating point multiplication-18*9.5



Sign =Sx^Sy=0^1=1
Append hidden bit 1 in the mantissa bits
10010000000000000000000
10011000000000000000000
Multiply mantissa bit
10010….0 x 10011 ….0

101010110…….0
remove hidden bit from the result 01010110……
0 Add exponents (Ex+Ey-bias)
10000011 + 10000010

100000101-001111111

010000110//normalize the result (10000110)
Final answer:
1 10000110 01010110000000000000000 =-171

C. Pipelined Floating point Arithmetic Unit

Pipelining is a special technique to give the faster output and reduce the delay in the design. It allows many operations to occur in parallel. Pipelining reduces the critical path in the circuit hence increases the speed. Generally in Pipelining, each operation of the stage is performed at each clock pulse and concurrently the output of the previous stage is given to the next stage so there is no waste of clock pulse in the pipelining.

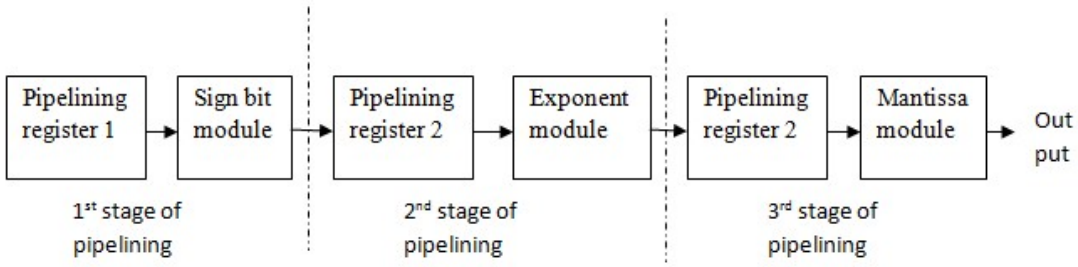


Figure 9. Pipelined floating point arithmetic unit

Floating point arithmetic unit is splitted into three parts sign bit module, exponent module, mantissa module. Pipelining register 1 stores the sign bit of the operands, pipelining register 2 stores the exponent bit of the operands, pipelining register 3 stores the mantissa bits of the operands. In the first stage of pipelining compute the sign bit of the result. In the second stage of pipelining compute the exponent bit of the result. In the third stage of pipelining compute the mantissa bit of the result.

V RESULTS AND DISCUSSION

The design was simulated in Xilinix 14.5 & ModelSim 6.3f. This chapter comprises of the simulation and synthesis results of Radix-4 FFT processor, complex multiplier with 4 multipliers and 2 adders/subtractors, complex multiplier with 3 multipliers and 5 adders/subtractors, input memory scheduling algorithm and MDC FFT processor and Floating point arithmetic unit results.

A.Radix-4 FFT Processor Simulation Results

The simulation result of radix-4 4 point FFT processor is shown in Figure 10. In the simulation result the inputs are x0r, x0i, x1r, x1i, x2r, x2i, x3r, x3i and the outputs are or0, oi0, or1, oi1, or2, or2, oi3, or3. Variables x0r, x1r, x2r, x3r represents the real part input. Variables x0i, x1i, x2i, x3i represents the imaginary part input. Variables o0r, o1r, o2r, o3r represents the real part output. Variables o0i, o1i, o2i, o3i represents the imaginary part output. The input sequence x (n) is {0, 1, 1, 2} and the output sequence x (k) is{4,-1+j,-2,-1-j }.

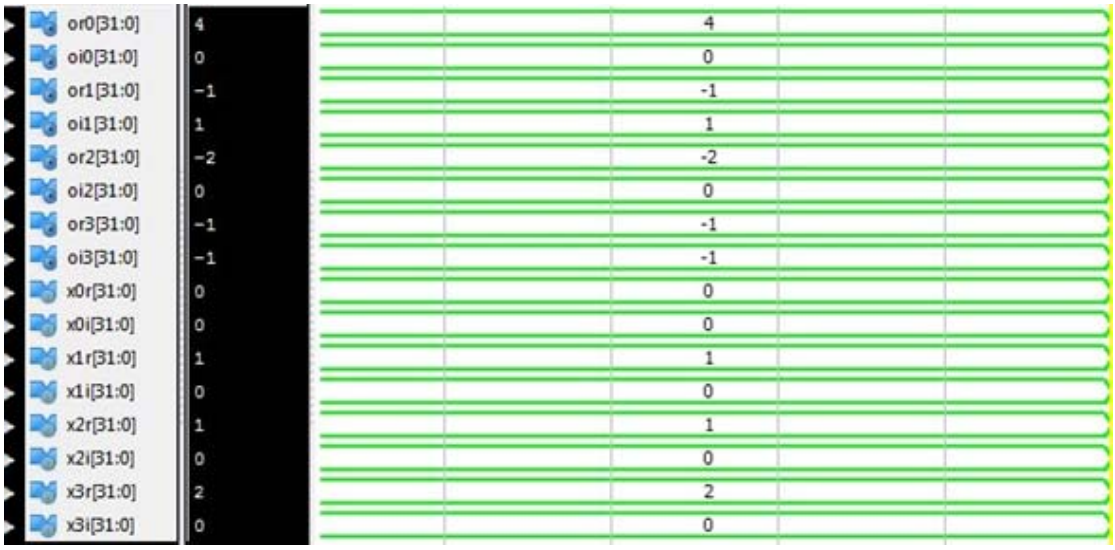


Figure 10. Simulation result of Radix 4 FFT Processor

B. Device Utilization Summary of Radix-4 FFT Processor

Device utilization shown in Table 1 depicts the number of registers, LUT's, slices, IOs, buffers used by the Radix-4 FFT processor design.

Table 1. Device Utilization Summary of Radix-4 FFT processor

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	404	63400	0%
Number of fully used LUT-FF pairs	0	404	0%
Number of bonded IOBs	320	210	152%
Number of DSP48E1s	22	240	9%

C. Simulation Result of Complex Multiplier with 4 multipliers and 2 adders/subtractors

The simulation result of complex multiplier with 4 multipliers and 2 adders/ subtractors is shown in Figure 11. In the simulation output, the inputs variables are c, d, e, f, and the output variables are xr, xi. The input complex number 2+1j is multiplied by another complex number 1+2j the output is 0+j5. xr represent the real part output and its value is 0. xi represent the imaginary part output and its value is 5.

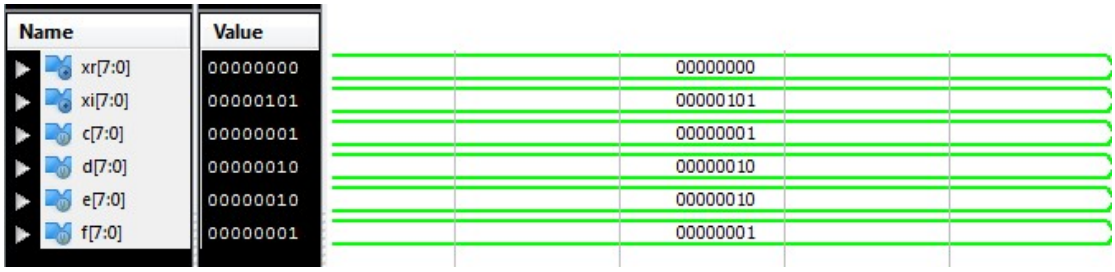


Figure 11. Simulation Result of Complex multiplier with 4 multipliers and 2 adders/subtractors

D. Simulation Result of Complex Multiplier with 3 multipliers and 5 adders/subtractors

The simulation result of complex multiplier with 3 multipliers and 5 adders/ subtractors is shown in Figure 12. In the simulation output, the input variables are c, d, e, f, and the output variables are xr, xi. The input complex number 2+1j is multiplied with another complex number 1+2j and the output is 0+j5. xr represent the real part output and its value is 0. xi represent the imaginary part output and its value is 5. This is the same as the one obtained in Figure 11but here, only 3 multiplications are involved instead of 4 multiplications. Complex multiplier with four multiplier and two adders/subtractors requires larger chip area in hardware implementation. So, Complex Multiplier with 3 multiplier and 5 adders/subtractors is used to implement MDC FFT processor.

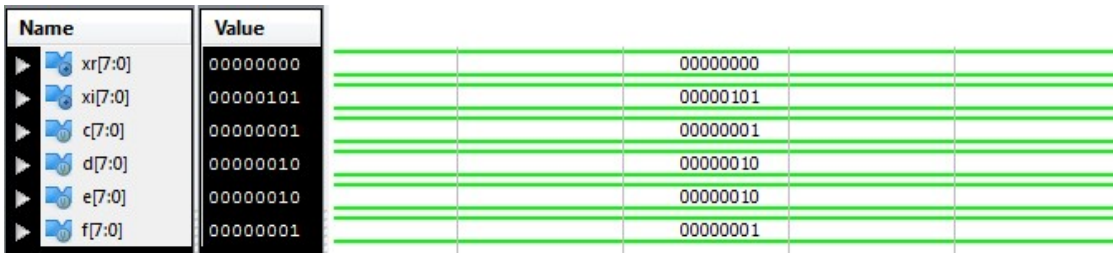


Figure 12. Simulation Result of Complex multiplier with 3 multipliers and 5adders/subtractors

E. Device Utilization Summary of Complex Multiplier with 4 multipliers and 2 adders/subtractors of Complex Multiplier with 3 multipliers and 5 adders/subtractors

The Device Utilization Summary of Complex Multiplier with 4 multipliers and 2 adders/subtractors and complex multiplier with 3 multipliers and 5 adders/subtractors are shown in Table 2 and Table 3 respectively. Device utilization summary explains the number of registers, LUT's, slices, IOs, buffers used by the Complex Multiplier with 3 multipliers and 5 adders/subtractors. Compared to Table 2 the number of devices used in the design decreases. So Complex Multiplier with 3 multipliers and 5adders/subtractors is used in the MDC FFT processor.

Table 2 Device Utilization Summary of Complex Multiplier with 4 multipliers and 2 adders/subtractors

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of bonded IOBs	48	210	22%
Number of DSP48E1s	4	240	1%

Table 3. Device Utilization Summary of Complex Multiplier with 3 multipliers and 5 adders/subtractors

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of bonded IOBs	48	210	22%
Number of DSP48E1s	3	240	1%

F. Simulation Result of First In First Out

The simulation output of FIFO memory is shown in Figure 13. If we write 16 data into FIFO memory, we need 16 clock cycles. In the simulation output buf_out represent the memory output. When reset value is 1 buf_out value is zero. When reset value is zero memory write operation is taking place. Figure 13 shows the 16 data is to be written into the FIFO memory. The 16 data value is given by {3, 2, 1, 2, 1, 1, 1, 1, 2, 0, 1, 0, 3, 1, 1, 1}

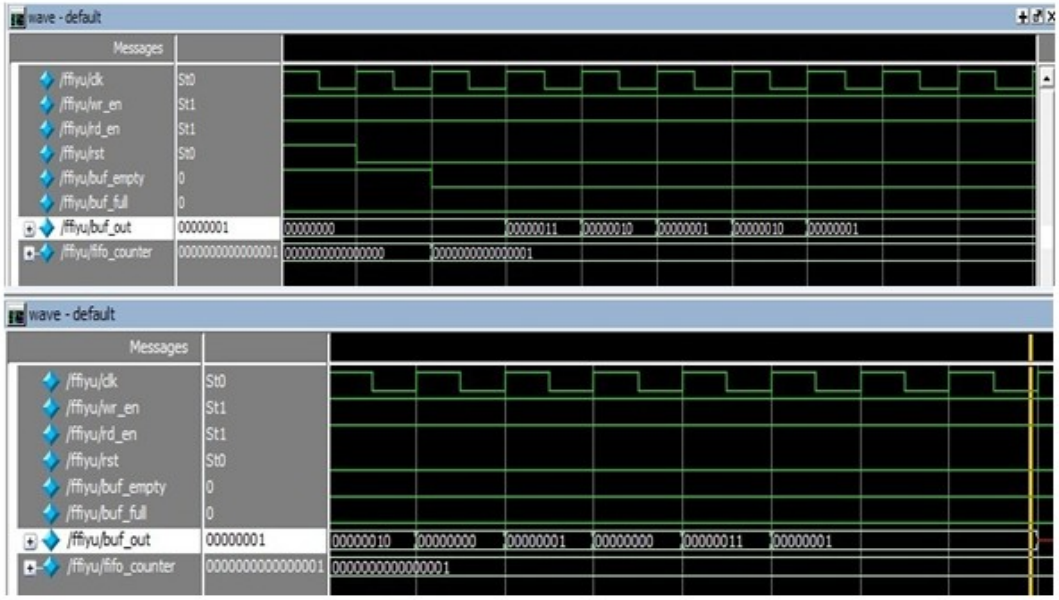


Figure 13. Simulation result of FIFO

G. Simulation Result of Memory Partitioning

The simulation output of memory partitioning is shown in Figure 14. The memory is divided into 4 memory banks. Memory partitioning scheme reduces the clock cycles needed for write operation. If we write 16 data means we need 16 clock cycles as shown in Figure 13. But the memory is divided into 4 memory bank means, we need only 4 clock cycles to write 16 data. The 16 data is divided into 4 sets. The first 4 data are stored in the memory bank1. The second 4 data are stored in memory bank2. The third 4 data are stored in the memory bank 3. The fourth 4 data are stored in the memory bank 4. This simple memory scheduling scheme is used for MDC FFT processor.

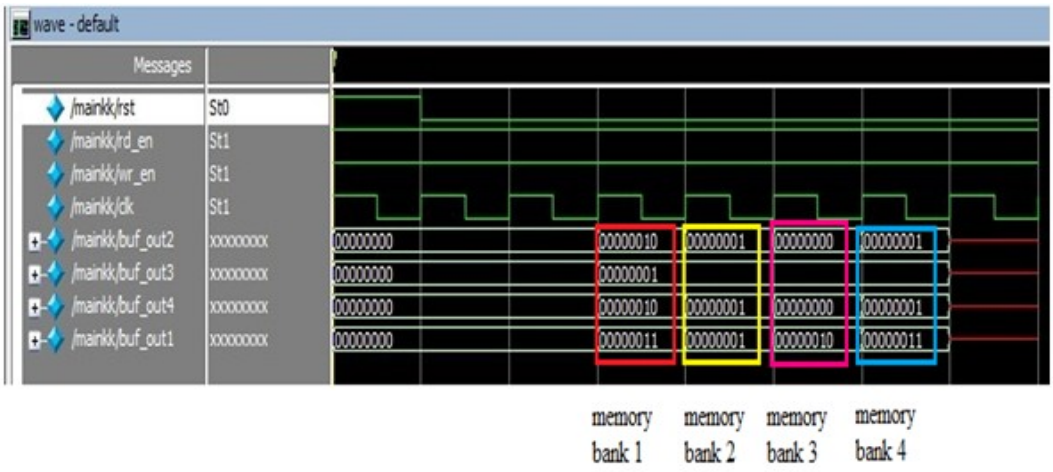


Figure14.Simulation Result of Memory Partitioning

In Figure 12, Memory bank1 stores the inputs {3, 2, 1, 2}. Memory bank2 stores the inputs {1, 1, 1, 1}. Memory bank3 stores the inputs {2, 0, 1, 3}. Memory bank 4 stores the inputs {3, 1, 1, 1}.

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 2.002ns (frequency: 499.575MHz)
  Total number of paths / destination ports: 210 / 22
---------------------------------------------------------
Delay:               2.002ns (Levels of Logic = 6)
  Source:            fifo_counter_0 (FF)
  Destination:       fifo_counter_4 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

Figure 15.Delay report for without memory partitioning

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 1.431ns (frequency: 699.007MHz)
  Total number of paths / destination ports: 62 / 19
---------------------------------------------------------
Delay:               1.431ns (Levels of Logic = 1)
  Source:            m1/fifo_counter_2 (FF)
  Destination:       m1/buf_out_1 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

Figure 16.Delay report for with memory partitioning

Table 3.Comparison result of without memory partitioning and with memory partitioning interms of delay

Parameter	Without memory partitioning	With memory partitioning
Delay (ns)	2.002	1.431

H. Simulation Result of Input Memory Scheduling

The simulation result of logical group of initial memory banks is shown in Figure 17. The 16 memory banks are logically grouped into four sets {a1, a2, a3, a4}, {b1, b2, b3, b4}, {c1, c2, c3, c4}, {d1, d2, d3, d4}. There are four input streams A, B, C, D, memory banks a1, a2, a3&a4 are used to store the samples of input stream A. Memory banks b1, b2, b3& b4 is used to store the samples of input stream B. Memory banks c1, c2, c3& c4 are used to store the samples of input stream C. Memory banks d1, d2, d3& d4 are used to store the samples of input stream D. For the case of N=16 the 4 stream input is A stream ={3, 2, 1, 2, 4, 2, 1, 2, 1, 1, 1, 1, 3, 1, 1, 1}, B stream={2, 1, 2, 1, 2, 0, 0, 0, 3, 3, 2, 2, 2, 0, 0, 0}, C stream={4, 2, 1, 2, 3, 3, 1, 2, 4, 1, 1, 1, 1, 3, 2, 2}, D stream = {2, 0, 0, 0, 2, 0, 0, 1, 3, 3, 3, 3, 4, 1, 1, 1}. a1 memory bank stores the samples{3, 2, 1, 2} a2 memory bank stores the samples {4, 2, 1, 2}. a3 memory bank stores the samples {1,1,1,1} a4 memory bank stores the samples {3, 1, 1, 1} b1 memory bank stores the samples {2, 1, 2, 1} b2 memory bank stores the samples {2, 0, 0, 0} b3 memory bank stores the samples {3, 3, 2, 2} b4 memory bank stores the samples {2, 0, 0, 0} c1 memory bank stores the samples {4, 2, 1, 2} c2 memory bank stores the samples {3, 3, 1, 2} c3 memory bank stores the samples {4, 1, 1, 1} c4 memory bank store the samples {1, 3, 2, 2} d1 memory bank stores the samples {2, 0, 0, 0} d2 memory bank stores the samples {2, 0, 0, 1} d3 memory bank stores the samples {3, 3, 3, 3} d4 memory bank stores the samples {4, 1, 1, 1}

Figure 17. Simulation result of input memory scheduling

I. Simulation Result of MDC FFT Processor

The simulation result of MDC FFT processor is shown in Figure 18. In the simulation result r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16 represent the output of the radix4 MDC FFT processor. In the first clock cycle, stage one radix 4 FFT processor process input samples of stream A. The inputs of stream A are (3, 2, 1, 2, 4, 2, 1, 2, 1, 1, 1, 1, 3, 1, 1, 1}, and the outputs of stream A are {8, 2, 0, 2, 9, 3, 1, 3, 4, 0, 0, 0, 6, 2, 2, 2}. In figure 18 the outputs are in the 16 bit binary form the first 8 bit represent the imaginary part next 8 bit represent the real part. In the second clock cycle, stage one radix 4 FFT processor process the input samples of stream B. The inputs of stream B are {2, 1, 2, 1, 2, 0, 0, 0, 3, 3, 2, 2, 2, 0, 0, 0}, and the outputs of stream B are {6, 0, 2, 0, 2, 2, 2, 2, 10, 1-j1, 0, 1+1j, 2, 2, 2, 2}. In the third clock cycle, stage three radix-4 FFT processor process the input samples of stream C. The inputs of stream C are {4, 2, 1, 2, 3, 3, 1, 2, 4, 1, 1, 1, 1, 3, 3, 2}, and the outputs of C stream are {9, 3, 1, 3, 9, 2-j1,-1, 2+1j, 7, 3, 3, 3, 9,-2-j1,-1,-2+j1}. In the fourth clock cycle, stage four radix 4 FFT processor process input samples of stream C. The inputs of stream D are {2, 0, 0, 0, 2, 0, 0, 1, 3, 3, 3, 3, 4, 1, 1, 1}, and the outputs of D stream are {2, 2, 2, 2, 3, 2+j1, 1, 2-j1, 12, 0, 0, 0, 7, 3, 3,3}.

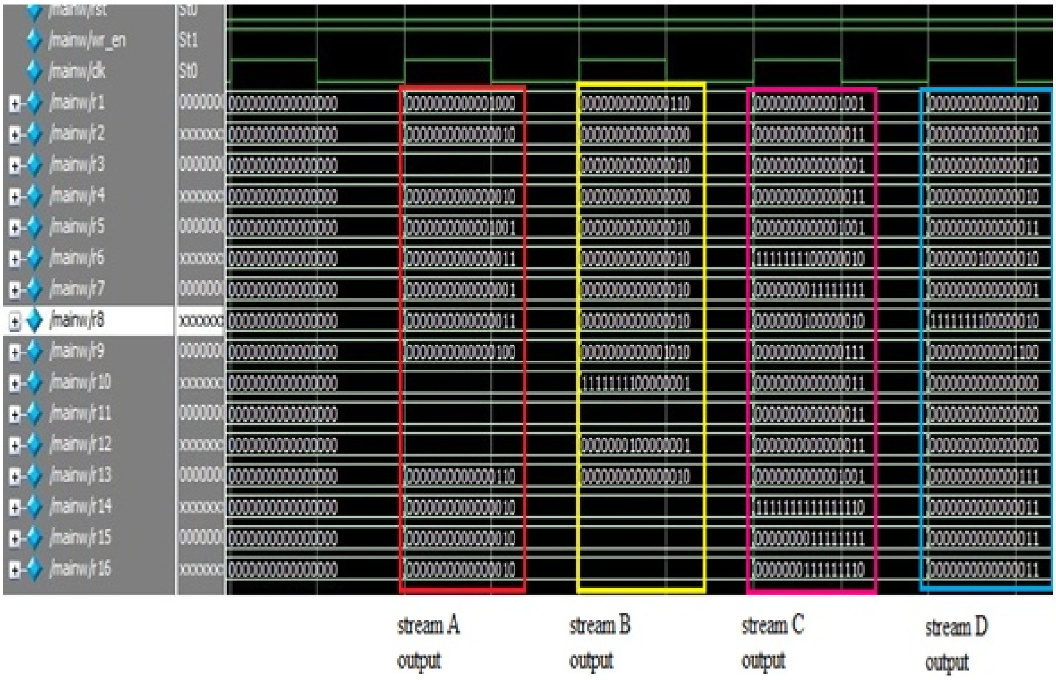


Figure 18.Simulation Result of MDC FFT processor

J. Simulation results of unpipelined and pipelined floating point adder

Figure 19 shows the unpipelined floating point addition, variables a, b represent input and o represent output. The inputs are a=0 0 1 1 11 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (1.25) b=0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (0.25).The output is o= 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (1.50)

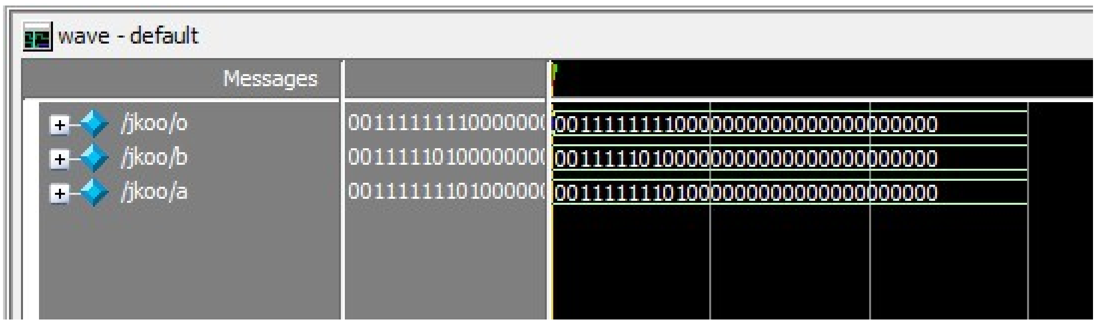


Figure19. Unpipelined floating point addition

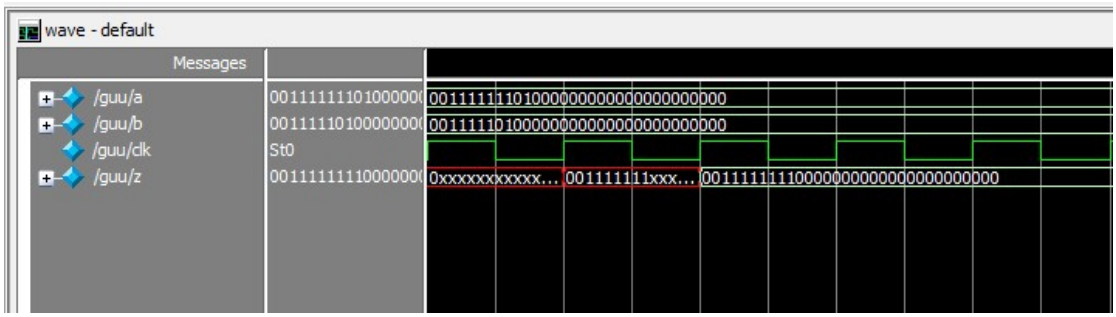


Figure20. Pipelined floating point addition

Figure 20 shows the pipelining floating point addition. Pipelined floating point adder computes sign bit result in the first clock cycle as 0, exponent bit result in the second clock cycle as 01111111,and mantissa bit result in the third clock cycle as 10000000000000000000000.

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

--------------------------------------------------------------------------------
Timing constraint: Default path analysis
  Total number of paths / destination ports: 2840290 / 32
--------------------------------------------------------------------------------
Delay:               7.485ns (Levels of Logic = 37)
  Source:            a<23> (PAD)
  Destination:       o<22> (PAD)

  Data Path: a<23> to o<22>
```

Figure 21. Delay report for unpipelined floating point adder

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

--------------------------------------------------------------------------------
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 3.330ns (frequency: 300.323MHz)
  Total number of paths / destination ports: 11687 / 197
--------------------------------------------------------------------------------
Delay:               3.330ns (Levels of Logic = 28)
  Source:            a_reg_28 (FF)
  Destination:       pip7_24 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

Figure 22. Delay report for pipelined floating point Adder

K. Simulation results of unpipelined and pipelined floating point multiplier

Figure 23 shows the unpipelined floating point subtraction, variables a, b represent input and o represent output a=0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (1.25),b = 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (0.25).The output is o=0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (1.00)

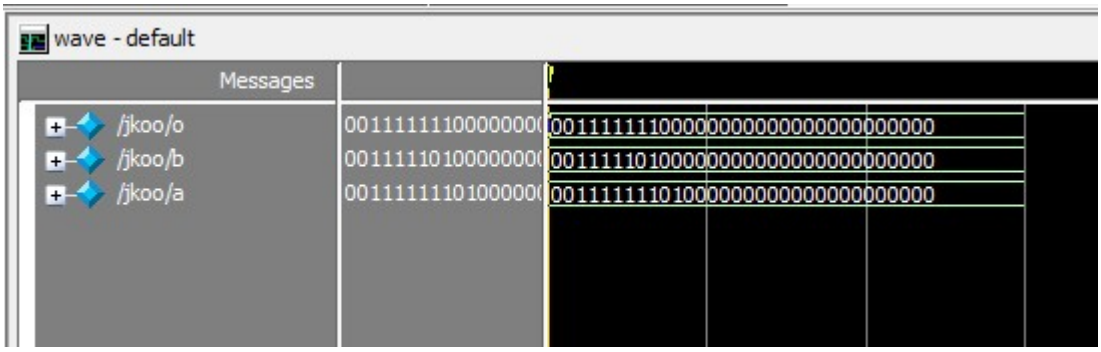


Figure23. Unpipelined floating point subtraction

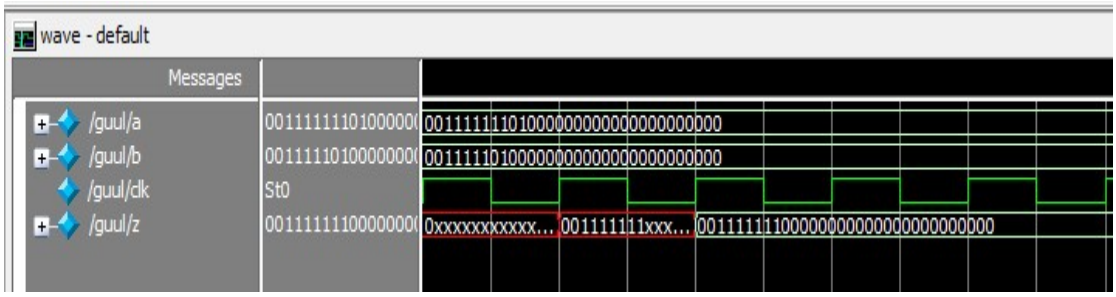


Figure24. Pipelined floating subtraction

Figure 24 shows the pipelined floating point subtraction. Pipelined floating point subtractor computes sign bit result in the first clock cycle as 0, exponent bit result in the second clock cycle as 01111111and mantissa bit result in the third clock cycle as 00000000000000000000000.

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 2840290 / 32
-------------------------------------------------------------------------
Delay:               7.485ns (Levels of Logic = 37)
  Source:            a<23> (PAD)
  Destination:       o<22> (PAD)

  Data Path: a<23> to o<22>
```

Figure25. Delay report for unpipelined floating point subtractor

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 3.330ns (frequency: 300.323MHz)
  Total number of paths / destination ports: 11687 / 197
-------------------------------------------------------------------------
Delay:               3.330ns (Levels of Logic = 28)
  Source:            a_reg_28 (FF)
  Destination:       pip7_24 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

Figure 26. Delay report for pipelined floating point subtrctor

L. Simulation results of unpipelined and pipelined floating point multiplier

Figure 27 shows the unpipelined floating point addition, variables a, b represent input and o represent output, a=1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (18), b = 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (9.5). The output is o = 1 1 0 0 0 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (-171)

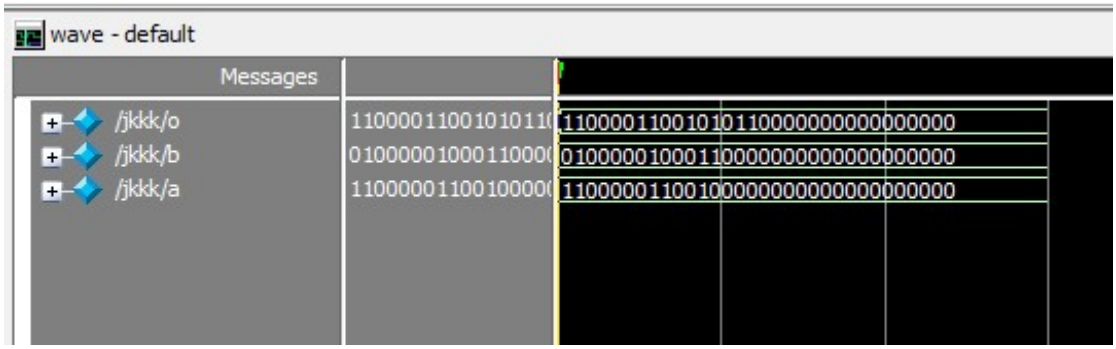


Figure 27.Unpipelined floating point multiplication

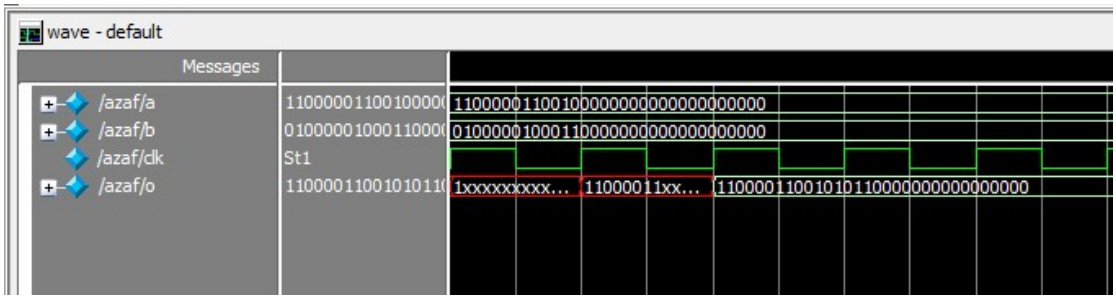


Figure28. Pipelined floating point multiplication

Figure 28 shows the pipelined floating point multiplication. Pipelined floating point multiplier computes sign bit result in the first clock cycle as 1, exponent bit

result in the second clock cycle ie) 10000110and mantissa bit result in the third clock cycle ie) 01010110000000000000000

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 150391 / 32
-------------------------------------------------------------------------
Delay:               5.524ns (Levels of Logic = 5)
  Source:            a<22> (PAD)
  Destination:       o<28> (PAD)

  Data Path: a<22> to o<28>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             1   0.001   0.279  a_22_IBUF (a_22_IBUF)
     DSP48E1:A22->PCOUT47    1   2.970   0.000  Mmult_m (Mmult_m_PCOUT
     DSP48E1:PCIN47->P30   31   1.107   0.790  Mmult_m1 (m<47>)
     LUT6:I1->O            1   0.097   0.279  Mmux_n002661 (o_28_OBUF)
     OBUF:I->O                 0.000          o_28_OBUF (o<28>)
    ----------------------------------------
    Total                      5.524ns (4.175ns logic, 1.349ns route)
                                       (75.6% logic, 24.4% route)
```

Figure 29. Delay report for unpipelined floating point multiplier

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 4.123ns (frequency: 242.542MHz)
  Total number of paths / destination ports: 323 / 149
-------------------------------------------------------------------------
Delay:               4.123ns (Levels of Logic = 0)
  Source:            Mmult_m (DSP)
  Destination:       Mmult_m1 (DSP)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

Figure 30. Delay report for pipelined floating point multiplier

Table 4.Comparison result of unpipelined &pipelined floating point Arithmetic unit interms of delay

ARITHMETIC UNIT	UnPipelined floatingpoint arithmetic unit delay (ns)	Pipelined floatingpoint arithmetic unit delay (ns)
Floating point adder	7.485	3.330
Floating pointsubtractor	7.485	3.330
Floating pointmultiplier	5.524	4.123

CONCLUSION

The proposed FFT processor is simulated using Xilinix 14.5 and ModelSim 6.3f. The inputs to the FFT processor are scheduled by using Multipath delay commutator architecture. Conversion from time domain to frequency domain of multiple data streams are carried out by FFT/IFFT processor with the memory scheduled architecture. The inputs to the FFT processor are scheduled as a set of memory banks. By doing so, the computation time is minimized and better resource utilization is achieved. MDC architecture and memory scheduling are very much suitable for FFT/IFFT processor in multiple input multiple output OFDM system. The proposed memory scheduling scheme can effectively reduce the computation time for collecting input data of the FFT processor from the memory. Floating point representation has its advantages of its resolution and accuracy compared to fixed point number representation. The simulation results of pipelined floating point arithmetic shows the reduction in delay compared to unpipelined floating point arithmetic unit. The results of these pipelined implementation are to be used in the MDC FFT processor for various set of simulations in our future work.

REFERENCES

[1] A. Cortes, I. Velez, and J. F. Sevillano, “Radix rk FFTs: Matricial representation and SDC/SDF pipeline implementation,” IEEE Trans. Signal Process., vol. 57, 7, pp. 2824-2839, Jul. 2009.

[2] B. Fu and P. Ampadu, ”An area efficient FFT/IFFT processor for MIMO-OFDM WLAN 802.11n,” J. Signal Process. Syst., vol. 56, no. 1,pp. 59-68, Jul. 2009.

[3] B. G. Jo and M. H. Sunwoo, “New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 52, no. 5, pp. 1471-1477, Sep. 2002.1024-point FFT processor,” IEEE J. Solid-State Circuits, vol. 34, no. 3, pp. 380-387, Mar.1999.

[5] C.-L. Hung, S.-S. Long, and M.-T. Shiue, “A low power and variable length FFT processor design for flexible MIMO OFDM systems,” in Proc. IEEE Int. Symp. Circuits Syst., May 2009, pp. 705-708

[6] E. E. Swartzlander, W. K. W. Young, and S. J. Joseph,"A radix 4 delay commutator for fast Fourier transform processor implementation," IEEE J. Solid-State Circuits, vol. 19, no. 5, pp. 702-709, Oct. 1984.

[7] Kai-Jiun Yang, Shang-Ho Tsai and Gene C. H. Chuang," MDC FFT/IFFT Processor With Variable Length for MIMO-OFDM Systems. IEEE Transactions on Very Large Scale Integration (VLSI) Systems.vol.21.no.April 2013.

[8] M. S. Patil, T. D. Chhatbar, and A. D. Darji, "An area efficient and low power implementation of 2048 point FFT/IFFT processor for mobile Wimax," in Proc. Int. Conf. Signal Process. Commun., 2010, pp. 1-4.

[9] Samir Palnitkar Verilog HDL A guide to Digital Design and Synthesis

[10] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in Proc. IEEE Int. Parallel Process. Symp., Apr. 1996, pp. 766-770.