International Journal of Electrical and Electronic Engineering & Telecommunications

ISSN 2319 – 2518 www.ijeetc.com
Vol. 2, No. 2, April 2013
© 2013 IJEETC. All Rights Reserved

Research Paper

EFFICIENT IMPLEMENTATION OF ROM-LESS FFT/IFFT PROCESSOR USING FUSED MULTIPLY AND ADD UNIT

M Arunkumar[1]* and N Kirthika

*Corresponding Author: **M Arunkumar,** ✉ arunnkumarrm@gmail.com

Wireless systems is mainly based on the Orthogonal Frequency Division Multiplexing (OFDM).The orthogonal frequency division multiplexing is mainly used to split the input signals into number of sub carrier signals. The orthogonal frequency division multiplexing requires an Inverse Fast Fourier Transform (IFFT).The inverse fast Fourier transform is mainly used to produce multiple sub-carriers signal. The FFT/IFFT processor for OFDM applications is described in this paper. The architecture uses single path delay feedback style and it occupies less memory space. The read only memory is mainly used to store the twiddle factors. To eliminate the Read Only Memory (ROM) a ROM-less FFT/IFFT processor is used. The reconfigurable complex multipliers is mainly used to achieve the ROM-less FFT/IFFT processor and the bit parallel multipliers is mainly used for square root evaluation. The Fused Multiply Add (FMA) operation is very important in many scientific and engineering fields. The floating point unit increases the performance and accuracy of the floating point. The Fused Multiply and Add unit is mainly used to reduce the latency and power.

Keywords: FFT, IFFT, Fused multiply add unit, Complex multiplier

INTRODUCTION

The Fast Fourier Transform (FFT) is mainly used as the fundamental component of many Digital Signal Processing (DSP) systems. The FFT is useful for frequency domain analysis. The FFT mainly used to convert a signal, from time domain into frequency domain. By decomposing the Fast Fourier Transform operates an N point time domain signals each composed of single point. The Fast Fourier Transform (FFT) is the heart of OFDM that enables its fast and efficient modulation of signal. Is based mainly on the Discrete Fourier Transform the FFT algorithm is the fast computation algorithm of which is an essential component for the

[1] Sri Ramakrishna Engineering College, Coimbatore, India.

modulation scheme used for OFDM applications.

The Fast Fourier Transform is mainly based on the divide-and-conquer model, by which the discrete transform is divided into smaller and simpler transforms. The divide-and-conquer model is based on the idea that aN-point DFT computation can be divided into two N/2-point DFT computation. The Fast Fourier Transform can be used in many applications such as (WIMAX) terrestrial (DVB-T). Cooley and Tukey proposed fast Fourier transform (FFT) efficiently used to reduce the time complexity.

The FFT processor can be classified into two types they are memory based and pipeline architectures styles. Memory based architecture is mainly used to design an FFT processor and it is also known as the single Processing Element (PE) approach. The FFT design uses main PE and several memory units thus the hardware cost and power consumption is lower. The pipeline FFT processor has two approaches. The first approach uses pipeline architecture with Single path Delay Feedback (SDF) and other uses Multiple-path Delay Commutator (MDC) pipeline architecture. The Single path Delay Feedback (SDF) pipeline FFT has less memory space and its control unit is easy to design. The single path pipeline architecture is used in low power design especially for applications in DSP devices.

The FFT computation need to multiply with different twiddle factors which results in higher hardware cost because of large size of ROM is needed to store the unwanted twiddle factor.The complex multipliers is mainly used to eliminate the twiddle factor ROM and to achieve a ROM-less FFT/IFFT processor. The complex multipliers used in the FFT processor are realized with shift-and-add operations. Hence, the processor uses only a two-input digital multiplier and does not need any ROM for internal storage of coefficients. The complex multiplier design is mainly used for ROM-size reduction and it is used to produce twiddle factor as well as to compact the chip area.

FFT AND IFFT ALGORITHMS

An efficient the Fast Fourier Transform (FFT) is algorithm for the computating DFT. The Fast Fourier Transform is mainly on the decomposing of the discrete Fourier transform principle computation of a sequence into smaller discrete Fourier transforms. The FFT generates the same result as DFT, however the computation for N numbers complexity is O(N2) reduced to O(Nlog(N)). After Cooley and Tukey publishing for faster computation of the FFT algorithm of discrete transform to perform computation algorithms were proposed. Based on different algorithms for decomposition could be obtained for the discrete Fourier transform computation.

Cooley-Tukey FFT Algorithms

For efficient computation the technique of DFTs is based on divide and conquerapproach. The sub-problems are then independently solved and their solutions are combined. Computation DFT by dividing the sequence ofdata into sequences of smaller data DFTs for small data sequences can be computed efficiently. Cooley and Tukey efficiency demonstrated the simplicity and of the divide and conquer approach for DFT accepted for the divide and conquer approach.

We give a simple example.

COMPLEX MULTIPLIERS

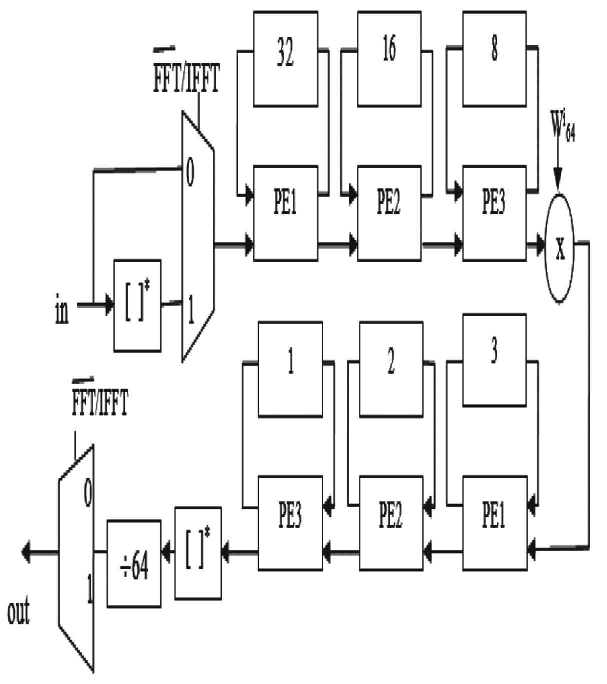
The block diagram of the 64-point FFT/IFFT processor consists of an input unit (I/P unit), two 8-point FFT units, a multiplier unit, an internal storage register bank (CB unit), an output unit (O/P unit), and a 5-bit binary counter that acts as the master controller for the entire design (Figure 1). The main performance bottlenecks in such a scheme. First, there is a large number of global wires resulting from multiplexing of the complex data to the 8-point FFT. Second, the construction of the multiplier unit to attain the required speed with minimal silicon area . Two bottlenecks and to make efficient algorithm-to-architecture mapping several special strategies have been adopted in the current architecture. The key component in the complex multiplier is the data path. The direct implementation of complex multiplier uses real multipliers the number of real multipliers can be reduced cost. The twiddle factors are known in advance, in the FFT processor which can be simplified the Distributed Arithmetic (DA) with complex multiplier.

The complex multiplier dissipates of total power in the work. A low power multiplier. Multipliers can be divided into three types. Although thechip area is less in bit-serial multiplier than that ofa high-speed clockrequires in bit-parallel multiplier. The bit-serial or digit-serial multiplier is usedto achieve high throughput, often needs several parallel units. The bit-parallel structure is used to meet the speed requirement.

The memory reduced complex multiplier is used to deal with the twiddle factor. To reduce the size of Look-Up Table (LUT), the twiddle factors with phases between 0 and –45° are selected to be stored, while the others can be reconstructed by these values. The data mapping scheme note that the twiddle factors within region a are defined as p-jq, and the data input are a+jb. According to the computation results, it can be seen that the multiplication with twiddle factors in region A is used to calculate the results in regions. Result, only 32 rather than 190 values are needed to be stored in the LUT.

To the data mapping scheme, it is clear that only twiddle factors with are in region A of therefore, it is sufficient for the modified complex multiplier to compute twiddle factors the architecture of the modified complex multiplier is depicted. First, the real and the imaginary parts of data are separately fed into the constant multiplier to produce intermediate terms.

Figure 1: Without Fused Multiply Add Unit Using Complex Multiplier



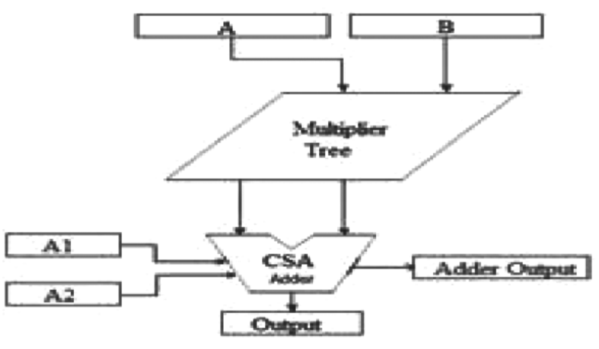
After that, these terms are fed into a data mapped to carry out the results final. Binary index of each region corresponds to the content of 3-bit control signal. Various approaches have been proposed to implement the multiplier constant. To according prior arts, hard-wired constants are efficient for multiplying four or eight distinct twiddle factors simultaneously. Specifically, there are eight hardwired constants within the constant multiplier and each one of them is built to deal with the specified twiddle factor. However, using hard-wired constants might not be an efficient solution because there is no need to perform so many multiplications at the same time in this work. Therefore, an alternative constant multiplier is designed to efficiently realize these constants.

In butterfly operation, the input signal is multiplied with twiddle factor. This operation results in multiplication complex. The complex multiplication is FFT processor major block. This complex multiplication put the constraint on computational performance of communication. The multiplications dominate the time execution. Results in high consumption power, huge area performance is poor. So we need to design good multiplier which should be fast, occupies less area and low power. Feasible only if we reduce the number operation. In literature, we could get various multiplier structures

FUSED MULTIPLY ADD UNIT

The Fused multiply add unit performs multiplication followed by addition and so that the calculation is done as operation single (Figure 2). This greatly increases the Floating-Point Unit (FPU). Recent advancement in performance and accuracy of FPGA architecture in order to improve performance. Many floating-point fused multiply add algorithms are developed to reduce the overall latency. The paper claims an estimated 15-20% reduction in latency as compared to a standard fused multiply add. Floating-Point Unit (FPU) is one of the most important custom applications needed in most hardware.

Figure 2: Fused Multiply Add Unit



Recently, the floating-point units of several commercial processors like IBM PowerPC, Intel/HP Itanium, MIPS-compatible Loongson-2F and HP PA-8000 have included a floating-point fused multiply add (FMA) unit to execute the fused multiply add operation using double-*AA* + (*BB* × *CC*) as an operation indivisible, intermediate rounding. The first FMA is introduced in 1990 by IBM RS/6000. After that FMA is implemented by several companies like HP, MIPS, ARM and Intel. It is a key feature of the floating-point unit because it greatly increases the floating-point performance and accuracy since rounding is performed only once for the result *AA* + (*BB* × *CC*) rather than twice for the multiplier and then for the adder. It also realizes reduction in the latency and hardware cost. FMA can be used instead of floating-point addition and floating-point multiplication by using constants, e.g., 0.0 +

(*BB* × *CC*) for multiplication and *AA* + (*BB* × 1.0) for addition.

Floating-point fused multiply add unit is one of the most important blocks that exist in floating-point unit as it increases its accuracy and performance. It is useful in many computations which involve the accumulation of products such as scientific and engineering fields. Algorithms are developed on floating-point fused multiply add unit to decrease its latency.

The multiplier is the process of generation and addition of the partial products. The differ in multiplication algorithms is used to generate partial products and the partial products are added together to produce the final result.

Partial Product Generation

Floating point fused multiply add unit includes a multiplier which uses a modified Booth's algorithm to generate partial products. Multiplier operand C in algorithm Booth's the recoded often into a radix higher than 2 in orders to reduce the number of products partially. Common recoding is radix-4 recoding (modified booth's recoding) with the digit set {–2, –1, 0, 1, 2} is shown .Series of consecutive 1's, the recoding algorithm. Which has the potential of reducing switching activity.Each three consecutive bits of the multiplier C represent the input to booth recoding block and the output from this block selects the right operation on the multiplicand B which may be "shift " (–2B, –B, 0, B, 2B) respectively due to the bits of multiplier.

Partial Product Reduction

After generation of the partial products, begin compression using 3-2 CSA tree. The reduction occurs by rows where each three partial product in same level will be input to CSA adder and output 2 operands (i.e., partial products) to the next level, and so on. For 27 partial products, 8 stages are required to produce a product in carry-save, or a carry vector and sum vector that need only to be added for a complete multiply.

Carry Save Adder (CSA)

The multiplier produce 106-bit sum and carry vectors that are reduced together with the aligned A using 3:2 CSA. Although the output of the multiplier must be positive number because we multiply two positive numbers (sign and magnitude representation), one of the two output vectors of the multiplier (sum and carry) may be negative because of using booth algorithm which use negative sets {–1, –2} which convert a positive number with sign and magnitude representation to a negative number with two's complement representation. The addition of sum and carry vectors must be a positive number but one of them, not both, may be negative.

Instead of using 161-bit CSA, Only the 106 least-significant bits of the aligned A are needed as input to the 3:2 CSA, because the product (i.e., sum and carry vectors) has only 106 bits and The 55 most-significant bits will be sign extension bits which have two cases {0, 0} if both sum and carry vectors are positive or {0, 1} if one of them is negative. For the 55 most bits significant, two multiplexers, one to select between A and inverted A as a sum vector and the second one to select between zeros and A as a carry vector by Xor-ing sign extension bits then the outputs of the two multiplexers are the CSA to obtain at the output concatenated of the 161-bit sum and carry words.

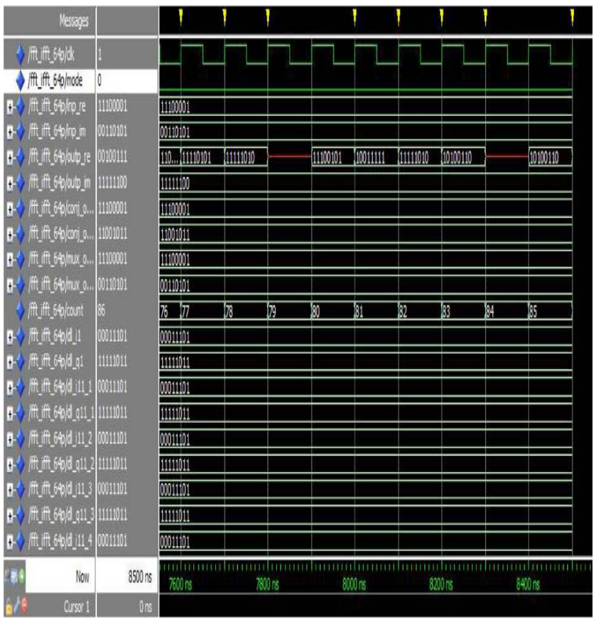
The Rounding

The rounding block rounds the result to nearest floating point number due to the round mode and performs post-normalization in case of an overflow. The round decision is taken by knowing also sticky and round bits. The sticky bit is calculated from the result by OR-ing all least significant bits after the round bit.

RESULTS AND ANALYSIS

The simulation results of without fused multiply add unit using complex multiplier is obtained using mentor graphics tool shown in Figure 3 and the figure has input, clk, and mode In the simulation results of proposed complex multipliers the clk is used to represent the clk signals and the mode signal is zero it acts as FFT and if the mode signal is one it acts as IFFT. The ouput is taken at the 64 count.

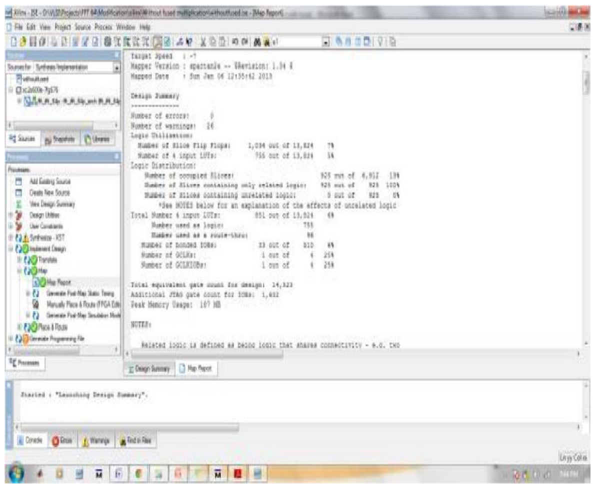
Figure 3: Simulation Results of Without Fused Multiply Add Unit Using Complex Multiplier


Messages
/fft_ifft_64p/clk 1
/fft_ifft_64p/mode 0
/fft_ifft_64p/inp_re 11100001
/fft_ifft_64p/inp_im 00110101
/fft_ifft_64p/outp_re 00100111
/fft_ifft_64p/outp_im 11111100
/fft_ifft_64p/conj_o... 11100001
/fft_ifft_64p/conj_o... 11001011
/fft_ifft_64p/mux_o... 11100001
/fft_ifft_64p/mux_o... 00110101
/fft_ifft_64p/count 86
/fft_ifft_64p/d_i1 00011101
/fft_ifft_64p/d_q1 11111011
/fft_ifft_64p/d_i11_1 00011101
/fft_ifft_64p/d_q11_1 11111011
/fft_ifft_64p/d_i11_2 00011101
/fft_ifft_64p/d_q11_2 11111011
/fft_ifft_64p/d_i11_3 00011101
/fft_ifft_64p/d_q11_3 11111011
/fft_ifft_64p/d_i11_4 00011101
11100001
00110101
110... 11110101 11111010 11100101 10011111 11111010 10100110 10100110
11111100
11100001
11001011
11100001
00110101
76 77 78 79 80 81 82 83 84 85
00011101
11111011
00011101
11111011
00011101
11111011
00011101
11111011
00011101
Now 8500 ns
Cursor 1 0 ns
7600 ns 7800 ns 8000 ns 8200 ns 8400 ns

Area Analysis of Without Fused Multiply Add Unit Using Complex Multiplier

The area analysis of the without fused multiply add unit using complex multiplier is analyzed by using mentor graphics is shown in the Figure 4.

Figure 4: Area Analysis of Without Fused Multiply Add Unit Using Complex Multiplier



```
Target Speed   : -7
Mapper Version : spartan3e -- $Revision: 1.34 $
Mapped Date    : Sun Jan 06 12:35:42 2013

Design Summary
--------------
Number of errors:      0
Number of warnings:   26
Logic Utilization:
  Number of Slice Flip Flops:     1,034 out of 13,824    7%
  Number of 4 input LUTs:           755 out of 13,824    5%
Logic Distribution:
    Number of occupied Slices:                       925 out of  6,912   13%
    Number of Slices containing only related logic:  925 out of    925  100%
    Number of Slices containing unrelated logic:       0 out of    925    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          851 out of 13,824    6%
      Number used as logic:                    755
      Number used as a route-thru:              96
   Number of bonded IOBs:            33 out of    510    6%
   Number of GCLKs:                   1 out of      4   25%
   Number of GCLKIOBs:                1 out of      4   25%

Total equivalent gate count for design:  14,523
Additional JTAG gate count for IOBs:  1,632
Peak Memory Usage:  167 MB

NOTES:

   Related logic is defined as being logic that shares connectivity - e.g. two

Started : "Launching Design Summary".
```

Here the total power consumed by the without fused multiply add unit using complex multiplier is noted as 14,523. The total area is mainly depends on the total number of gate counts.

Power Analysis of Without Fused Multiply Add Unit Using Complex Multiplier

The power analysis of the without fused multiply add unit using complex multiplier is analyzed by using xilinux design suite is shown in the Figure 5.

The simulation results of fused multiply add unit using complex multiplier is obtained using mentor graphics tool shown in Figure 6 and the figure has input, clk, and mode In the simulation results of proposed complex

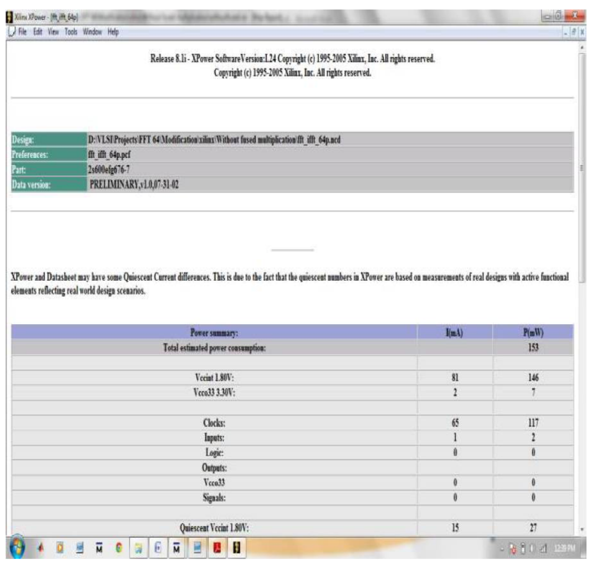


Figure 5: Power Analysis of Without Fused Multiply Add Unit Using Complex Multiplier

Release 8.1i - XPower SoftwareVersion:L24 Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.

Design:	D:\VLSI\Projects\FFT 64\Modification\xilinx\Without fused multiplication\fft_ifft_64p.ncd
Preferences:	fft_ifft_64p.pcf
Part:	2s600efg676-7
Data version:	PRELIMINARY,v1.0,07-31-02

XPower and Datasheet may have some Quiescent Current differences. This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		153
Vccint 1.80V:	81	146
Vcco33 3.30V:	2	7
Clocks:	65	117
Inputs:	1	2
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.80V:	15	27

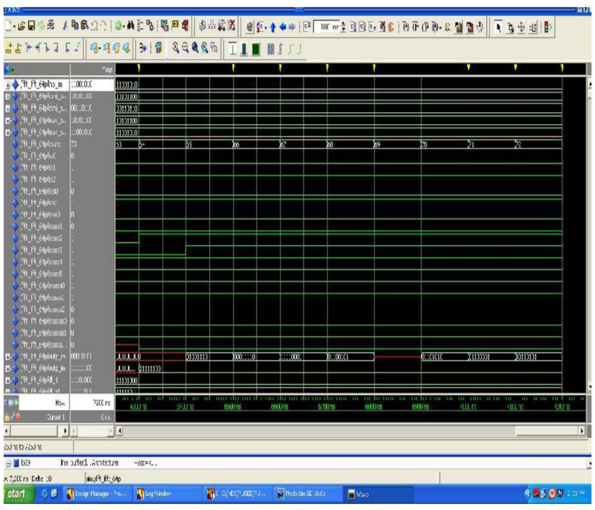


Figure 6: Simulation Results of Fused Multiply Add Unit

multipliers the clk is used to represent the clk signals and the mode signal is zero it acts as FFT and if the mode signal is one it acts as IFFT. The ouput is taken at the 64 count.

Area Analysis of Fused Multiply Add Unit Using Complex Multiplier

The area analysis of the fused multiply add unit using complex multiplier is analyzed by using mentor graphics is shown in the Figure 7.

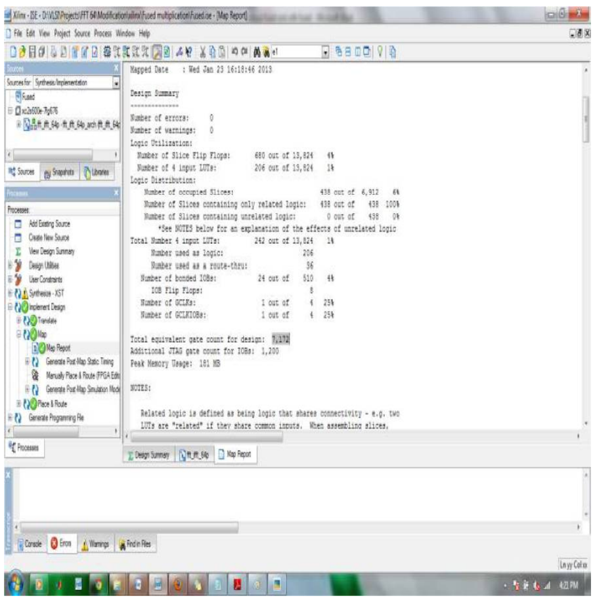


Figure 7: Area Analysis of Fused Multiply Add Unit Using Complex Multiplier

```
Mapped Date    : Wed Jan 23 16:18:46 2013

Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:       680 out of 13,824    4%
  Number of 4 input LUTs:           206 out of 13,824    1%
Logic Distribution:
    Number of occupied Slices:                         438 out of  6,912    6%
    Number of Slices containing only related logic:    438 out of    438  100%
    Number of Slices containing unrelated logic:         0 out of    438    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          242 out of 13,824    1%
      Number used as logic:                       206
      Number used as a route-thru:                 36
   Number of bonded IOBs:            24 out of    510    4%
      IOB Flip Flops:                               8
   Number of GCLKs:                   1 out of      4   25%
   Number of GCLKIOBs:                1 out of      4   25%

Total equivalent gate count for design:  7,172
Additional JTAG gate count for IOBs:  1,200
Peak Memory Usage:  181 MB

NOTES:

   Related logic is defined as being logic that shares connectivity - e.g. two
   LUTs are "related" if they share common inputs.  When assembling slices,
```

Power Analysis of Fused Multiply Add Unit Using Complex Multiplier

The area analysis of the fused multiply add unit using complex multiplier is analyzed by using xilinux design suite is shown in the Figure 8.

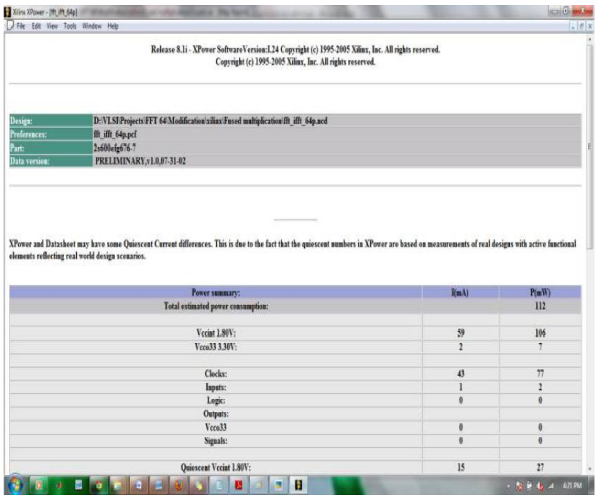


Figure 8: Power Analysis of Fusedmultiply Add Unit Using Complex Multiplier

Release 8.1i - XPower SoftwareVersion:L24 Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.
Copyright (c) 1995-2005 Xilinx, Inc. All rights reserved.

Design:	D:\VLSI\Projects\FFT 64\Modification\xilinx\Fused multiplication\fft_ifft_64p.ncd
Preferences:	fft_ifft_64p.pcf
Part:	2s600efg676-7
Data version:	PRELIMINARY,v1.0,07-31-02

XPower and Datasheet may have some Quiescent Current differences. This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		112
Vccint 1.80V:	59	106
Vcco33 3.30V:	2	7
Clocks:	43	77
Inputs:	1	2
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.80V:	15	27

Table 1: Power and Area Comparison of the Proposed Design with Fused Multiply Add Unit

Parameter	Without Fused Multiply and Add Unit	With Fused Multiply Add Unit
Power (mW)	153	128
Gate Counts	14,523	7,172

CONCLUSION

The Fused Multiply Add (FMA) operation in 1990 was introduced RS/6000 by IBM for execution of the single instruction by the equation $A + (B \times C)$ floating-point with single and double precision operands. This hardware unit was designed to reduce the latency of dot product calculations and provided greater floating-point arithmetic accuracy since only a single rounding is performed. FMA is implemented by several commercial processors like IBM, HP, MIPS, ARM and Intel. FMA can be used instead of floating-point addition and floating-point multiplication by using constants for multiplications and for addition. The Fused Multiply Add (FMA) is mainly used to decrease the latency. The existing architecture of without fused multiply add unit occupies a power of 153 (mw) and the area of without fused multiply add unit is 14,523. The proposed architecture of fused multiply add unit occupies a power of 128 (mw) and the area of fused multiply add unit is 7,172. The power and area of the proposed fused multiply add unit is lower than the existing work which results in lower power consumption and lower area.

ACKNOWLEDGEMENT

The authors thank the Management and Principal of SriRamakrishna Engineering College, Coimbatore for providing excellent computing facility and encouragement.

REFERENCES

1. Bass B M (1999), "A Low-Power, High Performance, 1024-Point FFT Processor", *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 3, pp. 380-387.

2. Chu Yu, Yi-Ting Liao, Mao-Hsu Yen, Pao-Ann Hsiung and Sao-Jie Chen (2011), "A Novel Low-Power 64-Point Pipelined FFT/IFFT Processor for OFDM Applications", in Proc. IEEE Int'l Conference on Consumer Electronics, pp. 452-453.

3. Chin-Teng Lin, Yuan-Chu Yu and Lan-Da Van (2006), "A Low-Power 64-Point FFT-IFFT Design for IEEE 802.11a WLAN Application", in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), pp. 4523-4526.

4. Cooley J W and Tukey J W (1965), "An Algorithm for the Machine Calculation of Complex Fourier Series", *Math.Computer*, Vol. 19, pp. 297-301.

5. Gold B and Bially T (1973), "Parallelism in Fast Fourier Transform Hardware", *IEEE Trans. Audio and Electroacoustics*, Vol. AU-21, No. 1, pp. 5-16.

6. Groginsky H L and Works G A (1970), "A Pipeline Fast Fourier Transform", *IEEE Transactions on Computers*, Vol. C-19, No. 11, pp. 1015-1019.

7. He S and Torkelson M (1998), "Designing Pipeline FFT Processor for OFDM (de)Modulation", in Proc. URSI Int. Symp. Signals, Systems, and Electronics, Vol. 29, pp. 257-262.

8. Hokenek E, Montoye R K and Cook (1990), "Second-Generation RISC Floating Point with Multiply-Add Fused", *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, pp. 1207-1213.

9. McClellan J H and Purdy R J (1978), "Applications of Digital Signal Processing to Radar", A V Oppenheim (Ed.), *Applications of Digital Signal Processing*, pp. 239-329, Prentice-Hall.

10. Montoye R K, Hokenek E and Runyon S L (1990), "Design of the IBM RISC System/6000 Floating-Point Execution Unit", *IBM J. Research and Development*, Vol. 34, pp. 59-70.

11. Saleh H H and Swartzlander E E Jr (2008), "A Floating-Point Fused Dot-Product Unit", *Proc.* IEEE Int'l Conf. Computer Design (ICCD), pp. 427-431.

12. Takahashi D (2003), "A Radix-16 FFT Algorithm Suitable for Multiply-Add Instruction Based on Goedecker Method", Proc. Int'l Conf. Multimedia and Expo, Vol. 2, July, pp. II-845-II-848.