Design of a radix-8/4/2 FFT processor for OFDM systems

Jungmin Park
Computer Engineering
Iowa State University

The FFT processor is a very important component in Orthogonal Frequency Division Multiplexing(OFDM) communication system. In this paper, we propose an efficient variable-length radix-8/4/2 FFT architecture for OFDM systems. The FFT processor is based on radix-8 FFT algorithm and also supports radix-4 or radix-2 FFT computation. The pipelined radix-8/4/2 butterfly unit computes radix-8 FFT algorithm basically. For the limitation of FFT length, if it cannot run radix-8 FFT algorithm at the last stage then it computes radix-4 or radix-2 FFT algorithm. Furthermore, proposed FFT architecture use shared-memory to minimize and simplify hardware. We are using efficient In-place memory access method to maintain conflict-free data access and minimize memory size. Also we replace a very large lookup table with a twiddle factor generator which consumes less area then a ROM-based lookup table. The proposed FFT processor performs variable-length FFT including 64, 256, 512, 1024, 2048, 4096 and 8192 points which cover all the required FFT lengths used in 802.11a, 802.16a, DAB, DVB-T, VDSL and ADSL.

Categories and Subject Descriptors: CprE583 [**Project**]: Design of a radix-8/4/2 FFT processor for OFDM systems

Additional Key Words and Phrases: FFT, OFDM, Radix-8 DIF FFT, Radix-4 DIF FFT, Radix-2 DIF FFT

1. INTRODUCTION

An Orthogonal frequency division multiplexing (OFDM) signal consists of a sum of subcarriers that are modulated by using phase shift keying (PSK) or quadrature amplitude modulation (QAM)[Nee and Prasad 2000]. These days, OFDM technique is widely used for high-speed digital communications, such as xDSL, DAB, DVB-T/H, and WLAN. In OFDM system, Discrete Fourier transform (DFT)/Inverse-DFT is used and it is a very important operation. Since DFT/IDFT computation requires a large amount of arithmetic operations, we need an efficient FFT algorithm which can reduce the number of arithmetic operations to meet real time computation in OFDM systems.

There are many kinds of FFT architectures used in OFDM systems. They are mainly categorized as three types : the parallel architecture[Wold and Despain 1984], the pipeline architecture[He and Torkelson 1998] and the shared memory architecture[Lee et al. 2006]. The parallel and pipeline architecture have more butterfly processing units to achieve high performance but they consume larger area than the shared memory architecture. On the other hand, the shared memory architecture requires only one butterfly processing unit and has the advantage of area efficiency. But the shared memory architecture has a drawback of low throughput and requires a complex circuit design of memory address controller. Fortunately, the lower throughput of the shared memory architecture increases dramatically if the high radix algorithm is used. But the high radix algorithm has defects of more

complex memory scheme and limitation that FFT length(N) must be only powers of radix- $r(r^n)$.

We focus on the shared memory architecture for area efficiency and hardware simplicity which are required to make small OFDM receivers. One radix-8 butterfly processing unit is used and it has the pipeline structure in order to realize high throughput. However, the FFT computation is restricted to N points which are 8^n. We propose the structure which can perform the radix-4 or radix-2 FFT algorithm in the radix-8 butterfly processing unit to permit the FFT computation of all points which are powers of 2. Because of choosing high and mixed radix FFT algorithm, memory scheme is complex. Efficient memory assignment and addressing are proposed to reduce the complexity of memory scheme. The ROM-based lookup table storing twiddle factors consumes large area in case of long-length FFT computation. To solve the problem, the twiddle factor generator is replaced with the ROM-based lookup table.

In the next section, FFT algorithm is reviewed. An efficient In-place memory assignment and addressing for the shared memory architecture are proposed in section 3. Section 4 describes the proposed FFT architecture. Section 5 explains implementation and simulation. Performance analysis is shown in section 6. Finally, a concluding remark is given in section 7.

2. FFT ALGORITHM

The N-point Discrete Fourier Transform(DFT) of a sequence $x(n)$ is defined as

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

where W_N is $exp(-j2\pi/N)$. To compute $X[k]$ directly, N^2 multiplication and $N(N-1)$ addition are needed. If (1) is represented in the matrix form like the following equation,

$$\vec{X} = W\vec{x} \tag{2}$$

W is defined as

$$W = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_N^1 & w_N^2 & \cdots & w_N^{N-1} \\ 1 & w_N^2 & w_N^4 & \cdots & w_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_N^{N-1} & w_N^{2(N-1)} & \cdots & w_N^{(N-1)^2} \end{pmatrix} \tag{3}$$

If N points are a power of the radix-r, the N-point DFT is decomposed into a set of recursively related r-point transforms. Equation (3) is decomposed into the following matrix, F.

$$F = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_r^1 & w_r^2 & \cdots & w_r^{r-1} \\ 1 & w_r^2 & w_r^4 & \cdots & w_r^{2(r-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_r^{r-1} & w_r^{2(r-1)} & \cdots & w_r^{(r-1)^2} \end{pmatrix} \tag{4}$$

The permutation matrix, P is defined as

$$P = \begin{pmatrix} e_1 \\ e_{r+1} \\ \vdots \\ e_{(N/r-1)r+1} \\ e_2 \\ e_{r+2} \\ \vdots \\ e_{(N/r-1)r+2} \\ \vdots \\ e_r \\ e_{r+r} \\ \vdots \\ e_{(N/r-1)r+r} \end{pmatrix} \tag{5}$$

where e_j denotes a row vector of length N with 1 in the jth position and 0 in every other position. The diagonal matrix, D is defined as

$$D = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & w_N^1 & 0 & \cdots & 0 \\ 0 & 0 & w_N^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w_N^{r-1} \end{pmatrix} \tag{6}$$

$$W \cdot P = \begin{pmatrix} F & DF & D^2F & \cdots & D^{N/r-1}F \\ F & w_N^r DF & w_N^{2r} D^2F & \cdots & w_N^{(N/r-1)r} D^{N/r-1}F \\ F & w_N^{2r} DF & w_N^{4r} D^2F & \cdots & w_N^{2(N/r-1)r} D^{N/r-1}F \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ F & w_N^{(N/r-1)r} DF & w_N^{2(N/r-1)r} D^2F & \cdots & w_N^{(N/r-1)^2 r} D^{N/r-1}F \end{pmatrix}$$

$$= A \cdot \begin{pmatrix} I & 0 & 0 & \cdots & 0 \\ 0 & DF & 0 & \cdots & 0 \\ 0 & 0 & D^2F & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & D^{N/r-1}F \end{pmatrix} \tag{7}$$

where $A = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_N^r & w_N^{2r} & \cdots & w_N^{(N/r-1)r} \\ 1 & w_N^{2r} & w_N^{4r} & \cdots & w_N^{2(N/r-1)r} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_N^{(N/r-1)r} & w_N^{2(N/r-1)r} & \cdots & w_N^{(N/r-1)^2 r} \end{pmatrix}$

The $N/r \times N/r$ matrix A is also decomposed into the matrix F. Thus it is possible to evaluate the N-point DFT by performing a D^iF matrix operation recursively. When the radix-r is 8 in (4),

$$F = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w_8^1 & w_8^2 & w_8^3 & w_8^4 & w_8^5 & w_8^6 & w_8^7 \\ 1 & w_8^2 & w_8^4 & w_8^6 & 1 & w_8^2 & w_8^4 & w_8^6 \\ 1 & w_8^3 & w_8^6 & w_8^1 & w_8^4 & w_8^7 & w_8^2 & w_8^5 \\ 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 & 1 & w_8^4 \\ 1 & w_8^5 & w_8^2 & w_8^7 & w_8^4 & w_8^1 & w_8^6 & w_8^3 \\ 1 & w_8^6 & w_8^4 & w_8^2 & 1 & w_8^6 & w_8^4 & w_8^2 \\ 1 & w_8^7 & w_8^6 & w_8^5 & w_8^4 & w_8^3 & w_8^2 & w_8^1 \end{pmatrix} \tag{8}$$

In (8), the matrix F is used in the radix-8 FFT and also decomposed into the matrix when r is 2 in (4). Fig. 1 shows the radix-8 butterfly unit and Fig. 2 shows the decomposed radix-8 buttefly unit. In Fig. 2, the radix-8 butterfly unit consists of 3 sub-stages which contains radix-2 buttefly units. If computation is completed in the substage 2, it is to perform 2 radix-4 FFT computations simultaneously. And if computation finishes in the the substage 1, it is to perform 4 radix-4 FFT computations simultaneously. By using a decomposed radix-8 butterfly unit, FFT computation is possible in case of N points which is not powers of 8.

3. EFFICENT IN-PLACE MEMORY ASSIGNMENT AND ADDRESSING

In the shared memory architecture, ideally a PE should should simultaneously fetch r data from a memory then write r computed output data back to the memory. In order to avoid memory contention, the memory is divided into r seperate banks so that r data values can be read/written simultaneously. The "In-Place" strategy is used to minimized the memory size[Johnson 1992].

There are some efforts on memory addressing methods to achieve conflict-free memory access[Pease 1969][Cohen 1976][Johnson 1992][Ma 1999][Lo et al. 2001]. These methods hava a drawback that input data can come into the memory, when all output data in the memory have gone out. Chang[Chang and Park 2004]'s method can resolve this problem. But that method is applied to the fixed radix-r

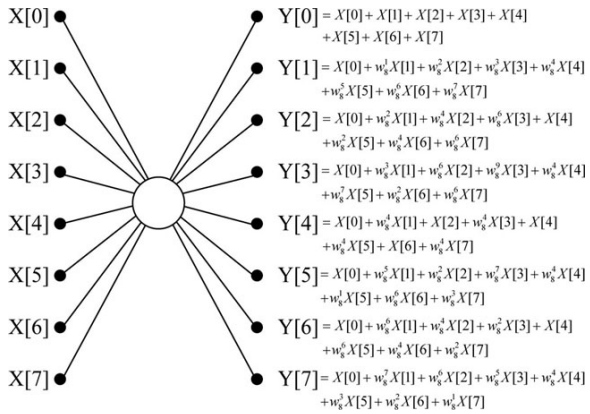

Fig. 1. The radix-8 butterfly unit.

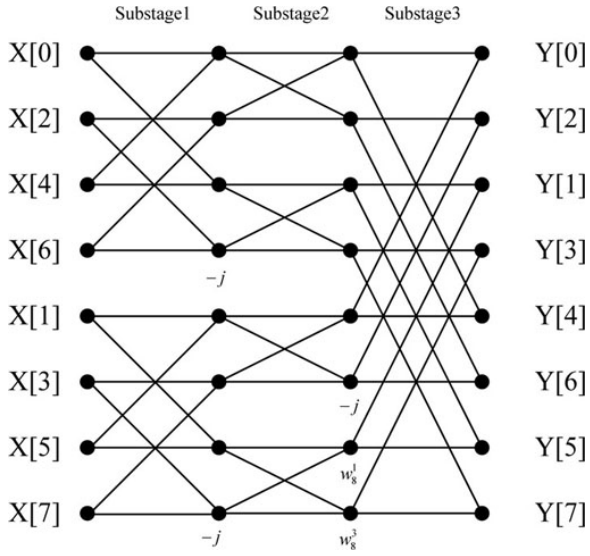

Fig. 2. The decomposed radix-8 butterfly unit.

FFT algorithm in a r-bank memory. We need to modify Chang's method to be applied to the radix-8, radix-4 and radix-2 FFT algorithm in a 8-bank memory.

At first, input data are stored from memory bank0 sequentially. The assigned address in bank and index of bank can be described by the following equations.

$$bank_{address} = (I) mod (N/8)$$

$$bank_{index} = \lfloor I \times 8/N \rfloor \tag{9}$$

where I is the index of input data, $0 \leq I < N$. In the first computation stage, 8 read bank addresses are generated from 0 to $(N/8) - 1$ in sequential order equally. 8 data in each bank are read simultaneously and then 8-point DFT operation is performed. The output data of the DFT operation are stored in 8 memory banks

of which write addresses are equal to read addresses. But some output data are assigned to different banks from read memory banks so that confliction of data can be avoided in the next stages. If the difference between indexes of two input data is $8^x, 2 \cdot 8^x$, or $4 \cdot 8^x$, where x is integer, such as (0, 8), (8, 24), (0, 32) and so on, these input data must not be stored in the same bank to avoid memory contension in all computation stages. When bank address represented in binary number are partitioned every 3 bits from MSB, each group means the data index's difference from data index in bank address 0 is $(a_2a_1a_0)_2 \cdot 2 \cdot 8^x, (a_2a_1a_0)_2 \cdot 2 \cdot 8^x$, or $(a_2a_1a_0)_2 \cdot 4 \cdot 8^x$. For example, in case of 512 points, 8 data indexes in each bank of which address is $(000000)_2$ are 0, 64, 128, 192, 256, 320, 384 and 448. When bank address is $(010000)_2$, 8 data index in each bank is 16, 80, 144, 208, 272, 336, 400 and 464. If two data groups are assigned from bank0 to bank7 sequentailly, confliction of data cannot be avoided. Thats why indexes of input data to compute in next stage are 0, 8, 16, 24, 32, 40, 48 and 56. To avoid confliction of data, 8 data in bank address $(010000)_2$ are rotation-shifted by $(010)_2 + (000)_2$ and then assigned into each bank. In conclusion, 8 data are rotation-shifted by the sum of each group and then assigned into 8 memory banks to avoid memory contention. The method of assignment can be described by the following equations.

$$n = log_2 N$$

$$bank_{address} = [a_{n-4}a_{n-5}...a_1a_0]_2$$

$$bank_{index} = (i + \sum_{t=0}^{\lceil (n-3)/3 \rceil - 1} (2^2 a_{n-4-3t} + 2a_{n-5-3t} + a_{n-6-3t}))mod8$$

$$if \;\; n - y - 3t < 0, a_{n-y-3t} = 0. \tag{10}$$

where i is the index of computed data in the butterfly unit.

From the second computation stage, the computed data in butterfly unit are assigned into the same address and memory-bank of input data. But the final outputs are stored in the different banks so that outputs of low index can be stored in low-index memory banks. Thus, the group of low-index data in low-index memory banks can go out first and then new data and old data can be accepted and go out concurrently. Table I shows the data stored in each memory bank after each stage in case of 64-point FFT.

The method of memory addressing change according to each stage and FFT length. Table II represents this method. In Table II,c_n is the nth bit of a butterfly counter which counts the number of butterfly operation in each stage and is reseted whenever each stage is complete. $n_2n_1n_0$ is 000, 001, 010, 011, 100, 101, 110 or 111.

4. THE PROPOSED FFT ARCHITECTURE

Components of proposed FFT architecture are a memory, a butterfly unit, two commutators, a data address generator, a twiddle factor generator and a control unit. The structure of proposed FFT processor is shown in Fig. 3.

Table I. Data of 64-point after stage computation in memory

After input

B0	B1	B2	B3	B4	B5	B6	B7
x[0]	x[8]	x[16]	x[24]	x[32]	x[40]	x[48]	x[56]
x[1]	x[9]	x[17]	x[25]	x[33]	x[41]	x[49]	x[57]
x[2]	x[10]	x[18]	x[26]	x[34]	x[42]	x[50]	x[58]
x[3]	x[11]	x[19]	x[27]	x[35]	x[43]	x[51]	x[59]
x[4]	x[12]	x[20]	x[28]	x[36]	x[44]	x[52]	x[60]
x[5]	x[13]	x[21]	x[29]	x[37]	x[45]	x[53]	x[61]
x[6]	x[14]	x[22]	x[30]	x[38]	x[46]	x[54]	x[62]
x[7]	x[15]	x[23]	x[31]	x[39]	x[47]	x[55]	x[63]

After Stage1 computation

B0	B1	B2	B3	B4	B5	B6	B7
g[0]	g[8]	g[16]	g[24]	g[32]	g[40]	g[48]	g[56]
g[57]	g[1]	g[9]	g[17]	g[25]	g[33]	g[41]	g[49]
g[50]	g[58]	g[2]	g[10]	g[18]	g[26]	g[34]	g[42]
g[43]	g[51]	g[59]	g[3]	g[11]	g[19]	g[27]	g[35]
g[36]	g[44]	g[52]	g[60]	g[4]	g[12]	g[20]	g[28]
g[29]	g[37]	g[45]	g[53]	g[61]	g[5]	g[13]	g[21]
g[22]	g[30]	g[38]	g[46]	g[54]	g[62]	g[6]	g[14]
g[15]	g[23]	g[31]	g[39]	g[47]	g[55]	g[63]	g[7]

After Stage2 computation

B0	B1	B2	B3	B4	B5	B6	B7
y[0]	y[9]	y[18]	y[27]	y[36]	y[45]	y[54]	y[63]
y[7]	y[8]	y[17]	y[26]	y[35]	y[44]	y[53]	y[62]
y[6]	y[15]	y[16]	y[25]	y[34]	y[43]	y[52]	y[61]
y[5]	y[14]	y[23]	y[24]	y[33]	y[42]	y[51]	y[60]
y[4]	y[13]	y[22]	y[31]	y[32]	y[41]	y[50]	y[59]
y[3]	y[12]	y[21]	y[30]	y[39]	y[40]	y[49]	y[58]
y[2]	y[11]	y[20]	y[29]	y[38]	y[47]	y[48]	y[57]
y[1]	y[10]	y[19]	y[28]	y[37]	y[46]	y[55]	y[56]

Table II. Memory Addressing

Mode	Address	64	512	4096	128	1024	8192	256	2048
1	$[c_9...c_0]_2$	First stage							
2	$[n_2n_1n_0c_9...c_3]_2$						2 stage		
3	$[c_9n_2n_1n_0c_8...c_3]_2$			2 stage					
4	$[c_9c_8n_2n_1n_0c_7...c_3]_2$								2 stage
5	$[c_9...c_7n_2n_1n_0c_6...c_3]_2$					2 stage	3 stage		
6	$[c_9...c_6n_2n_1n_0c_5...c_3]_2$		2 stage	3 stage					
7	$[c_9...c_5n_2n_1n_0c_4c_3]_2$							2 stage	3 stage
8	$[c_9...c_4n_2n_1n_0c_3]_2$				2 stage	3 stage	4 stage		
9	$[c_9...c_3n_2n_1n_0]_2$	Last stage							

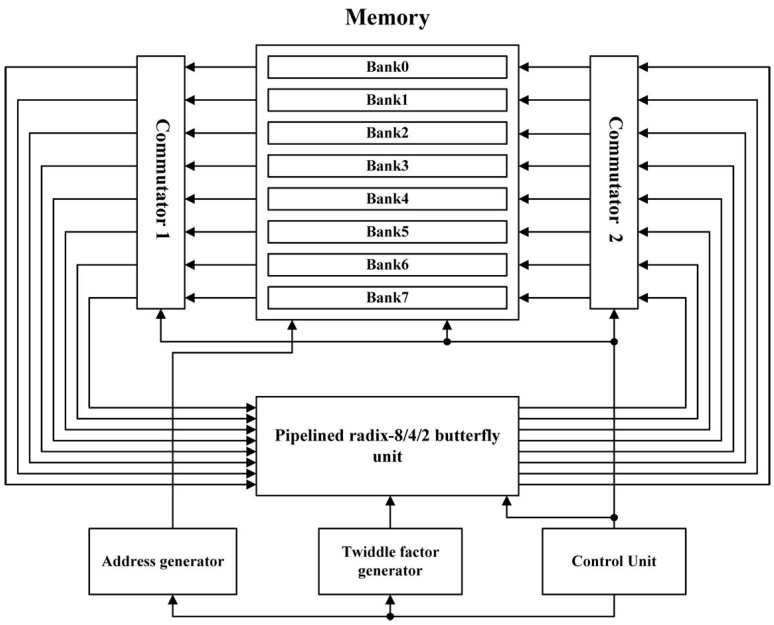

Fig. 3. The structure of proposed FFT processor.

4.1 Memory

The memory consists of 8 memory banks and has dual ports so that 8 complex data can simultaneously be read/written at one clock cycle.

4.2 Butterfly unit

The butterfly unit is designed to perform basically the radix-8 DIF FFT algorithm and also it can compute radix-4 or radix-2 DIF FFT algorithm. It computes the radix-8 FFT in all computation stage. But when the radix-8 FFT cannot be performed in the last stage for the limitation of FFT length, radix-4 or radix-2 FFT algorithm can be performed by this butterfly unit. Also, it has the pipeline structure with total 4 pipeline stages. The first, second and third pipeline stages perfome radix-2 FFT computations and multiplications by twiddle factors are computed in the last pipeline stage. Fig. 4 shows the pipelined radix-8/4/2 butterfly unit. In Fig. 4, if the control signals of MUXs, S is "00", the butterfly unit can compute four radix-2 FFT algorithm simultaneously. When the signal, S is "01", it performs two parallel radix-4 FFT algorithm. When S is "10", it can compute the radix-8 FFT algorithm without multiplication by twiddle factors. When S is "11", it can execute the radix-8 FFT computation and then multiply the results by twiddle factors.

4.3 Address generator

Memory addressing is performed by the address generator. Fig. 5 shows the structure of proposed address generator. When one of 9 modes in Table 2 are selected, 8 addresses are generated by each address generator. The position of 8 addresses are changed by a switch and a barrel shifter. The switch operates differently according

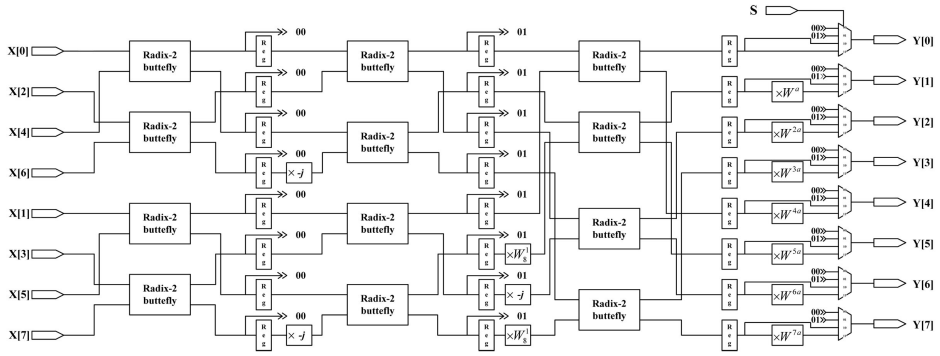

Fig. 4. The pipelined radix-8 butterfly unit.

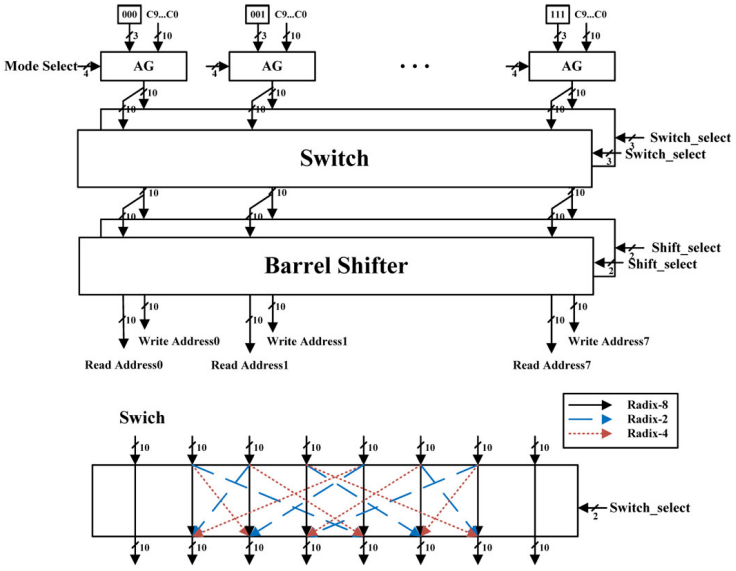

Fig. 5. The structure of proposed address generator.

to the radix r and 8 addresses are rotated right by $[c_2c_1c_0]_2$ which means 3 LSBs in the butterfly counter. Write addresses are equal to 4-clock delayed read addresses of each bank during computation because data read from each bank shoud be written to the same addresses of each bank after 4 clock cycles.

4.4 Commutator

Two commutators are barrel shifters which consist of 8 8-to-1 MUXs. The first commutator close to output ports of butterfly unit assigns the output data of the butterfly unit to correct memory banks. The second one close to input ports of butterfly unit changes the position of 8 data read from memory in index order. The control of two commutators changes according to each stage and FFT length and the control signal has 3-bit wide. Table III shows the control signal of two

Table III. Controal signal of two commutators

	1 stage	Other stages	Last stage
Comm.1	000	$c_2c_1c_0$	$c_2c_1c_0$
Comm.2	64, 512, 4 k : $c'_8c'_7c'_6 + c'_5c_4c'_3 + c'_2c'_1c'_0$	$c'_2c'_1c'_0$	000
	128, 1 k, 8 k : $c'_9c'_8c'_7 + c'_6c'_5c'_4 + c'_3c'_2c'_1 + c'_000$		
	256, 2 k : $c'_7c'_6c'_5 + c'_4c'_3c'_2 + c'_100$		

commutators. In Table III, c'_n represents the nth bit of the 4-clock delayed butterfly counter .

4.5 Twiddle-factor generator

We use the recursive sine/cosine function generator which is based on the well-known recursive feedback difference equation for the computation of sine and cosine functions. This method has the advantage of low complexity[Chi and Sau-Gee Chen 2004]. Equation (11) shows difference equations for the generation of sine function and cosine function.

$$sin(n\theta) = 2cos\theta \times sin(n-1)\theta - sin(n-2)\theta$$

$$cos(n\theta) = 2cos\theta \times cos(n-1)\theta - cos(n-2)\theta \tag{11}$$

In (11), there are feedback terms of previous computed value. It causes error propagation problem in finite precision calculation. If all terms in (11) have n-bit fraction precision, the maximum error of computed $sin(m\theta)$ and $cos(m\theta)$ is two-bit error[Chi and Sau-Gee Chen 2004]. Thus the bound of error is represented by the following equation.

$$-\frac{3}{2^n} \leq error \leq +\frac{3}{2^n} \tag{12}$$

We can know which is larger between correct value and computed value by comparing only 3 LSBs of correct value with 3 LSBs of computed value because only one of three cases($A < B, A = B or A > B$) satisfies (12). Thus error can be calculated and then compensated by using the correction table which contains 3 LSBs of correct values. The compensation method is represented by the following equations.

$$[z_2z_1z_o]_2 = unsigned([x_2x_1x_0]_2 - [y_2y_1y_0]_2)$$

$$value_{correct} = signed(value_{computed} + [z_2z_1z_0]_2) \tag{13}$$

where $[x_2x_1x_0]_2$ is the 3 LSBs in the correction table and $[y_2y_1y_0]_2$ is the 3 LSBs of computed value. Fig. 6 shows the structure of twiddle factor generator. In Fig. 6, the data width of $cos\theta_j$ in the cosine generator is 23 bits because all $cos\theta_j$ are distinguished when represented in minimum 23 bits which consists 2-bit integer and 21-bit fraction. We need 7 twiddle factor generators to be applied to the proposed FFT processor and 7 ROMs which store initial twiddle factors. 7 ROMs consume small area because initial twiddle factors of the first butterfly operation in each stage are only stored. Fig. 7 shows total structure of twiddle factor generator in the proposed FFT processor.

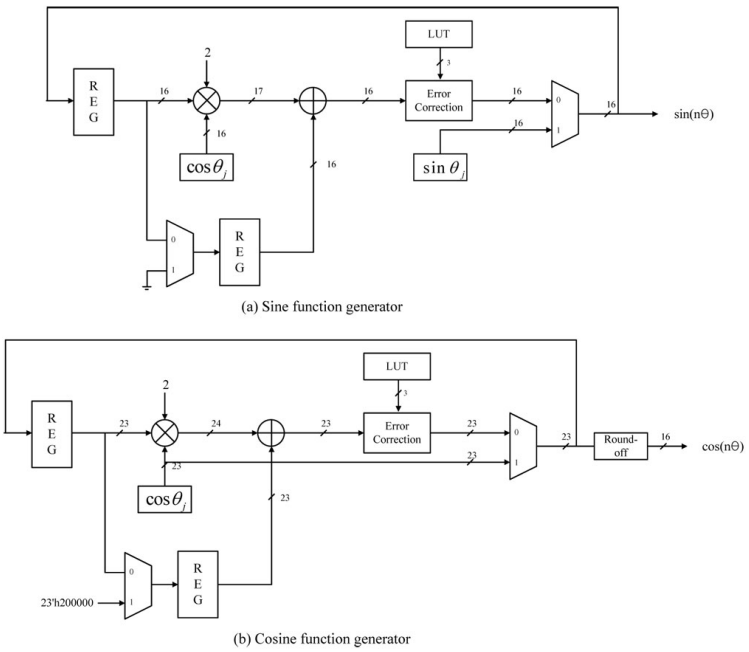

Fig. 6. The structure of twiddle factor generator.

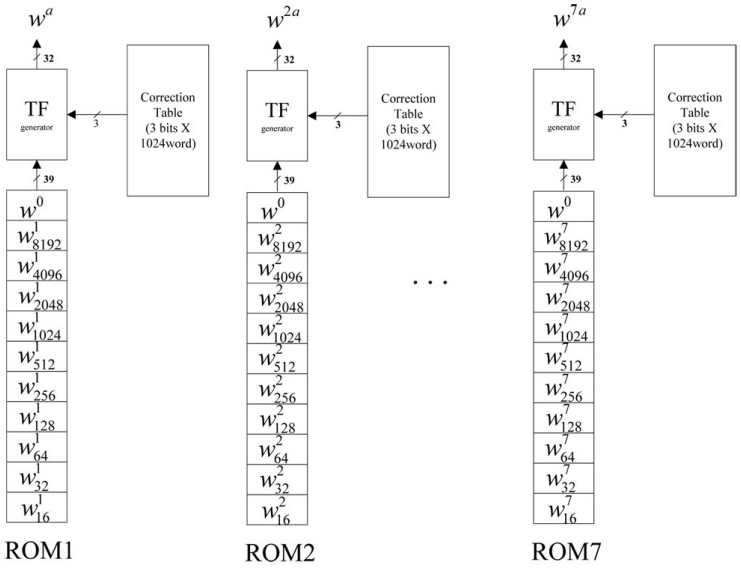

Fig. 7. Total structure of twiddle factor generator.

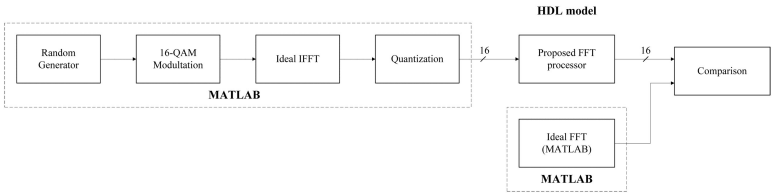

Fig. 8. The method of simulation and verification.

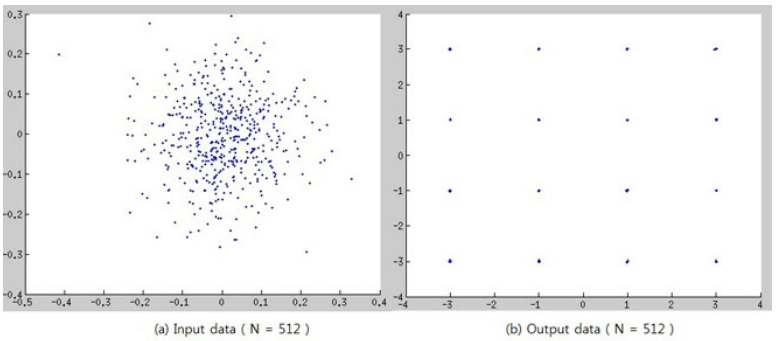

Fig. 9. The constellations of input and output data (N = 512).

Table IV. Synthesis results

Input data width	Phase factor width	LUTs	Block RAMs	DSP48s	Max. Freq.(MHz)
16	16	4811	22	10.339	96.723

5. IMPLEMENTATION AND SIMULATION

The proposed FFT processor has been modeled by VHDL in RTL-level. Input data consists of real part and imaginary part. Each part consists of 5 bits of integer and 11 bits of fraction. Thus total data width is 16 bits. In MATLAB, 16-QAM modulated data are generated randomly and these data come into IFFT module. The output data of IFFT is transformed into input format of the proposed FFT architecture. Fig. 8 shows our method to simulate and verify our FFT processor. Fig. 9 shows that input data and output data of the proposed FFT are mapped onto orthogonal coordinates in case of 512-point. The proposed FFT processor is also implemented in hardware using the Xilinx Virtex-5 FPGA. Design synthesis results are shown in Table IV.

6. PERFORMANCE ANALYSIS

The clock cycles of complete N-point FFT operation for only one frame can be computed by the following equations:

$$T_{frame} = I + L + O_{overlap} \quad cycles$$

$$I = N \quad cycles$$

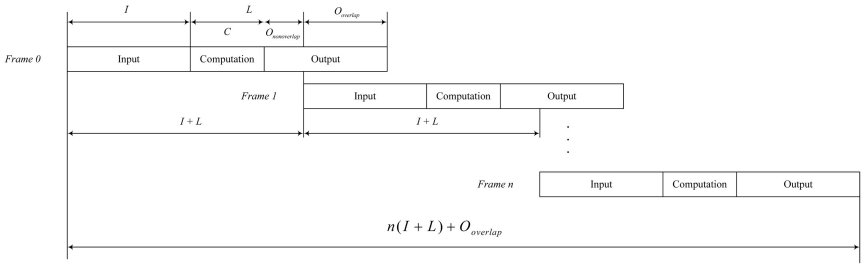

Fig. 10. The clock cycles of complete FFT operation for n frames.

Table V. Latency of N-point FFT (96 MHz)

N points	Latency cycles	Latency time (μs)
64	32	0.33
128	120	1.25
256	170	1.77
512	268	2.79
1K	1036	10.79
2K	1550	16.15
4K	2576	26.83
8K	9234	96.20

$$L = C + O_{nonoverlap} \quad cycles$$

$$C = (\lceil log_8 N \rceil - 1)(\frac{N}{8} + 4) + (\frac{N}{8} + x) \quad cycles$$

where $x = 4$ if $r_{last} = 8$, $x = 2$ if $r_{last} = 4$, $x = 0$ if $r_{last} = 2$.

$$O_{nonoverlap} = \frac{N}{r_{last}} \quad cycles$$

$$O_{overlap} = N - \frac{N}{r_{last}} \quad cycles \tag{14}$$

where r_{last} is the radix at the last stage computation. I means the cylces required to write input data in memory and C and O represent computation cycles and output cycles, respectively. L represents latency cycles from last input data of present frame to first input data of next frame and consists of computation cycles, C and output cycles which do not overlap with input cycles. $\frac{N}{8} + 4$ computation cycles are needed every stage except the last stage and the computation cycles of the last stage are different according to r_{last}. Fig. 10 shows complete clock cycles of n frames. If n is very large number, complete clock cycles per each frame is $I + L + \frac{O_{overlap}}{n} \approx I + L$. Table V shows latency cycles and time according to N points when the FFT processor operates at 96 MHz.

7. CONCLUSION

In this paper, we proposed a memory-based variable-length FFT processor architecture. We designed a pipelined radix-8/4/2 butterfly unit and proposed efficient

In-place memory assignment. Also we designed the twiddle factor generator which consume smaller area than ROM-based lookup table. This FFT processor has been modeled by VHDL and the verification in RTL-level has been completed in case of 64, 128, 256, 512, 1024, 2048, 4096 and 8192 points. This FFT processor can be applied to multi-standard OFDM systems including 802.11a, 802.16a, DAB, DVB-T, ADSL and VDSL.

REFERENCES

Chang, Y.-S. and Park, S.-C. 2004. An enhanced memory assignment scheme for memory-based fft processor. *Fundamentals of Electronics, Communications and Computer Sciences, IEICE TRANSACTIONS on E87-A,* 11 (nov), 3020–3024.

Chi, J.-C. and Sau-Gee Chen, B. 2004. An efficient fft twiddle factor generator. In *EUSIPCO 2004 : (XII. European Signal Processing Conference).*

Cohen, D. 1976. Simplified control of fft hardware. *Acoustics, Speech and Signal Processing, IEEE Transactions on 24,* 6 (dec), 577 – 579.

He, S. and Torkelson, M. 1998. Designing pipeline fft processor for ofdm (de)modulation. In *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on.* 257 –262.

Johnson, L. 1992. Conflict free memory addressing for dedicated fft hardware. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on 39,* 5 (may), 312 –316.

Lee, S.-Y., Chen, C.-C., Lee, C.-C., and Cheng, C.-J. 2006. A low-power vlsi architecture for a shared-memory fft processor with a mixed-radix algorithm and a simple memory control scheme. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on.* 4 pp. –160.

Lo, H.-F., Shieh, M.-D., and Wu, C.-M. 2001. Design of an efficient fft processor for dab system. In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on.* Vol. 4. 654 –657 vol. 4.

Ma, Y. 1999. An effective memory addressing scheme for fft processors. *Signal Processing, IEEE Transactions on 47,* 3 (mar), 907 –911.

Nee, R. v. and Prasad, R. 2000. *OFDM for Wireless Multimedia Communications*, 1st ed. Artech House, Inc., Norwood, MA, USA.

Pease, M. C. 1969. Organization of large scale fourier processors. *J. ACM 16*, 474–482.

Wold, E. and Despain, A. 1984. Pipeline and parallel-pipeline fft processors for vlsi implementations. *Computers, IEEE Transactions on C-33,* 5 (may), 414 –426.