# Fast Memory Addressing Scheme for Radix-4 FFT Implementation

Xin Xiao, Erdal Oruklu and Jafar Saniie
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois, 60616
erdal@ece.iit.edu

***Abstract*-- In this study, an efficient addressing scheme for radix-4 FFT processor is presented. The proposed method uses extra registers to buffer and reorder the data inputs of the butterfly unit. It avoids the modulo-r addition in the address generation; hence, the critical path is significantly shorter than the conventional radix-4 FFT implementations. A significant property of the proposed method is that the critical path of the address generator is independent from the FFT transform length N, making it extremely efficient for large FFT transforms. For performance evaluation, the new FFT architecture has been implemented by FPGA (Altera Stratix) hardware and also synthesized by CMOS 0.18µm technology. The results confirm the speed and area advantages for large FFTs. Although only radix-4 FFT address generation is presented in the paper, it can be used for higher radix FFT.**

## I. INTRODUCTION

Fast Fourier transform (FFT) is one of the key components for various signal processing and communications applications such as software defined radio [1] and OFDM [2]. A typical FFT processor is composed of butterfly calculation units, an address generator and memory units. This study is primarily concerned with improving the performance of the address generation unit of the FFT processor by eliminating the complex critical path components.

Pease [3] observed that the two data addresses of every butterfly differ in their parity. Parity check can be realized by modulo-r addition in hardware. Based on Pease's observation, Cohen [4] proposed a simplified control logic for radix-2 FFT address generation. Johnson [5] proposed a similar way to realize radix-r FFT addressing. In this method, the address generator is composed of several counters, barrel shifters, multiplexers and adder units. Other FFT processors [2,6] have been designed to realize high-radix FFT. A common drawback of all these methods is the need for successive addition operations to realize the address generation. The number of addition operations depends on the length of the FFT, so the address generation speed is slower as the FFT transform length increases. Several methods have been proposed to avoid the addition for radix-2 FFT [7,8] but these methods cannot be used for higher radix FFT.

This study presents a new architecture to realize the address generation for radix-4 FFT. The new address generator is composed of counters, barrel shifters, multiplexers and registers, but no addition operation is required. The critical path of the address generator is shorter, and furthermore, the critical path of this address generator is independent of the FFT length making it extremely efficient for long length FFT.

## II. RADIX-4 FFT

The $N$-point discrete Fourier transform is defined by

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk} \quad k=0,1,...,N-1, \quad W_N^{nk}=e^{-j\frac{2\pi}{N}nk} \tag{1}$$

The $N$-point FFT can be decomposed to repeated micro-operations called *butterfly* operations. When the size of the butterfly is $r$, the FFT operation is called a radix-$r$ FFT. For FFT hardware realization, if only one butterfly structure is implemented in the chip, this butterfly unit will execute all the calculations recursively. If parallel and pipeline processing techniques are used, an $N$ point radix-$r$ FFT can be executed by $\frac{N}{r}\log_r N$ clock cycles. This indicates that a radix-4 FFT can be four times faster than a radix-2 FFT. Fig. 1 shows the signal flow graph of 64-point radix-4 FFT, and Fig. 2 shows the general structure of the radix-4 butterfly. For hardware realization of FFT, multi-bank memory and "in-place" addressing strategy are often used to speed-up the memory access time and minimize the hardware consumption. For radix-$r$ FFT, $r$ banks of memory are needed to store data, and each memory bank could be two-port memory. With "in-place" strategy, the $r$ outputs of the butterfly can be written back to the same memory locations of the $r$ inputs, and replace the old data. In this case, to realize parallel and pipelined FFT processing, an efficient addressing scheme is needed to avoid the data conflict. A popular addressing scheme for radix-$r$ ($r$>2) was presented by Johnson [5], however due to the modulo-$r$ addition, this method is slow and the speed depends on the length of FFT.

978-1-4244-3355-1/09/$25.00©2009 IEEE

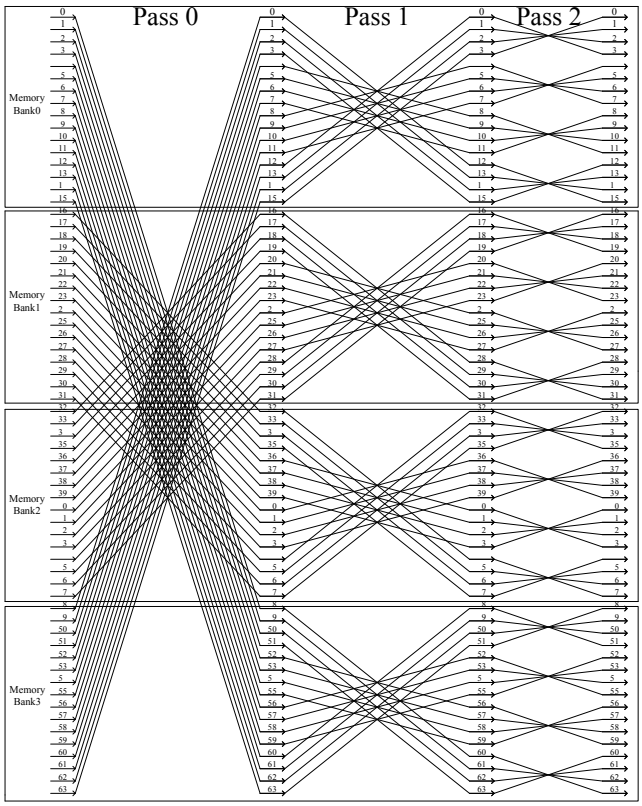
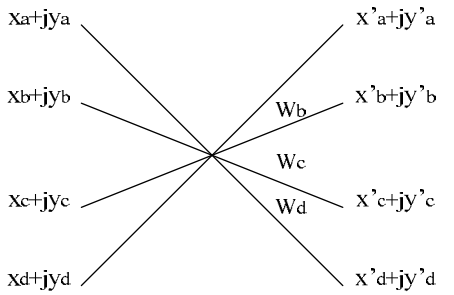



Figure 1. 64-point FFT using radix-4 butterfly units




$W_b=R_b+jI_b \quad W_c=R_c+jI_c \quad W_d=R_d+jI_d$

$x'_a=x_a+x_b+x_c+x_d$

$y'_a=y_a+y_b+y_c+y_d$

$x'_b=(x_a+y_b-x_c-y_d)R_b-(y_a-x_b-y_c+x_d)I_b$

$y'_b=(y_a-x_b-y_c+x_d)R_b+(x_a+y_b-x_c-y_d)I_b$

$x'_c=(x_a-x_b+x_c-x_d)R_c-(y_a-y_b+y_c-y_d)I_c$

$y'_c=(y_a-y_b+y_c-y_d)R_c+(x_a-x_b+x_c-x_d)I_c$

$x'_d=(x_a-y_b-x_c+y_d)R_d-(y_a+x_b-y_c-x_d)I_d$

$y'_d=(y_a+x_b-y_c-x_d)R_d+(x_a-y_b-x_c+y_d)I_d$

Figure 2. Butterfly structure of radix-4 FFT

TABLE I. ADDRESS SEQUENCES FOR THE FIRST REGISTER SET (R0-R15)

| CLK | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | | | |
| R1 | | 4 | 4 | 4 | 17 | 17 | 17 | 17 | 6 | 6 | 6 | 6 | 19 | 19 | 19 | 19 | 19 | | |
| R2 | | | 8 | 8 | 33 | 33 | 33 | 33 | 33 | 33 | 10 | 10 | 35 | 35 | 35 | 35 | 35 | 35 | |
| R3 | | | | 12 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 14 | 51 | 51 | 51 | 51 | 51 | 51 | 51 |
| R4 | 16 | 16 | 16 | 16 | 16 | 5 | 5 | 5 | 18 | 18 | 18 | 18 | 18 | 7 | 7 | 7 | | | |
| R5 | | 20 | 20 | 20 | 20 | 21 | 21 | 21 | 21 | 22 | 22 | 22 | 22 | 23 | 23 | 23 | 23 | | |
| R6 | | | 24 | 24 | 24 | 37 | 37 | 37 | 37 | 37 | 26 | 26 | 26 | 39 | 39 | 39 | 39 | 39 | |
| R7 | | | | 28 | 28 | 53 | 53 | 53 | 53 | 53 | 53 | 30 | 30 | 55 | 55 | 55 | 55 | 55 | 55 |
| R8 | 32 | 32 | 32 | 32 | 32 | 32 | 9 | 9 | 34 | 34 | 34 | 34 | 34 | 34 | 11 | 11 | | | |
| R9 | | 36 | 36 | 36 | 36 | 36 | 25 | 25 | 25 | 38 | 38 | 38 | 38 | 38 | 27 | 27 | 27 | | |
| R10 | | | 40 | 40 | 40 | 40 | 41 | 41 | 41 | 41 | 42 | 42 | 42 | 42 | 43 | 43 | 43 | 43 | |
| R11 | | | | 44 | 44 | 44 | 57 | 57 | 57 | 57 | 57 | 46 | 46 | 46 | 59 | 59 | 59 | 59 | 59 |
| R12 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 13 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 15 | | | |
| R13 | | 52 | 52 | 52 | 52 | 52 | 52 | 29 | 29 | 54 | 54 | 54 | 54 | 54 | 54 | 31 | 31 | | |
| R14 | | | 56 | 56 | 56 | 56 | 56 | 45 | 45 | 45 | 58 | 58 | 58 | 58 | 58 | 47 | 47 | 47 | |
| R15 | | | | 60 | 60 | 60 | 60 | 61 | 61 | 61 | 61 | 62 | 62 | 62 | 62 | 63 | 63 | 63 | 63 |

TABLE II. ADDRESS SEQUENCES FOR THE SECOND REGISTER SET (R16-R31)

| Clk | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R16 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | | | |
| R17 | 4 | 4 | 4 | 4 | 4 | 17 | 17 | 17 | 6 | 6 | 6 | 6 | 6 | 19 | 19 | 19 | | | |
| R18 | 8 | 8 | 8 | 8 | 8 | 8 | 33 | 33 | 10 | 10 | 10 | 10 | 10 | 10 | 35 | 35 | | | |
| R19 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 49 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 51 | | | |
| R20 | | 16 | 16 | 16 | 5 | 5 | 5 | 5 | 5 | 18 | 18 | 18 | 7 | 7 | 7 | 7 | 7 | | |
| R21 | | 20 | 20 | 20 | 20 | 21 | 21 | 21 | 21 | 22 | 22 | 22 | 22 | 23 | 23 | 23 | 23 | | |
| R22 | | 24 | 24 | 24 | 24 | 24 | 37 | 37 | 37 | 26 | 26 | 26 | 26 | 26 | 39 | 39 | 39 | | |
| R23 | | 28 | 28 | 28 | 28 | 28 | 28 | 53 | 53 | 30 | 30 | 30 | 30 | 30 | 30 | 55 | 55 | | |
| R24 | | | 32 | 32 | 9 | 9 | 9 | 9 | 9 | 9 | 34 | 34 | 11 | 11 | 11 | 11 | 11 | 11 | |
| R25 | | | 36 | 36 | 36 | 25 | 25 | 25 | 25 | 25 | 38 | 38 | 38 | 27 | 27 | 27 | 27 | 27 | |
| R26 | | | 40 | 40 | 40 | 40 | 41 | 41 | 41 | 41 | 42 | 42 | 42 | 42 | 43 | 43 | 43 | 43 | |
| R27 | | | 44 | 44 | 44 | 44 | 44 | 57 | 57 | 57 | 46 | 46 | 46 | 46 | 46 | 59 | 59 | 59 | |
| R28 | | | | 48 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 50 | 15 | 15 | 15 | 15 | 15 | 15 | 15 |
| R29 | | | | 52 | 52 | 29 | 29 | 29 | 29 | 29 | 29 | 54 | 54 | 31 | 31 | 31 | 31 | 31 | 31 |
| R30 | | | | 56 | 56 | 56 | 45 | 45 | 45 | 45 | 45 | 58 | 58 | 58 | 47 | 47 | 47 | 47 | 47 |
| R31 | | | | 60 | 60 | 60 | 60 | 61 | 61 | 61 | 61 | 62 | 62 | 62 | 62 | 63 | 63 | 63 | 63 |

## III. PROPOSED METHOD

This study presents a new addressing scheme for radix-r FFT, which avoids complex addition steps in the address generation unit at the expense of more registers and multiplexers.

In proposed approach, four memory banks are used to store the data, as shown in Figure 1. In pass 0, four inputs and four outputs of any butterfly stage belong to different memory banks. However, for pass 1 and pass 2, four inputs and four outputs of any butterfly stage belong to same memory bank. Since each memory bank is a two-port memory, at each clock cycle, each memory bank can export (read) once and import (write) once. Four clock cycles are necessary to perform four read and four write accesses in pass 1 and pass 2. Ideally, in four clock cycles, 16 imports and 16 exports can be accomplished in the four memory banks. This can facilitate four radix-4 butterfly operations to be executed in four clock cycles. For 100% utilization of the butterfly unit as described above, two sets of registers are necessary to buffer these data. Each set contains 16 registers; the first register set (register 0 to register 15) is employed to buffer the outputs of the memory units before they are imported to the butterfly unit, and the second register set (register 16 to register 31) is used to buffer the outputs of the butterfly unit before they are imported to the memory banks. Eight 16-to-1 multiplexers are used in each set to re-order the data for avoiding any data conflict.

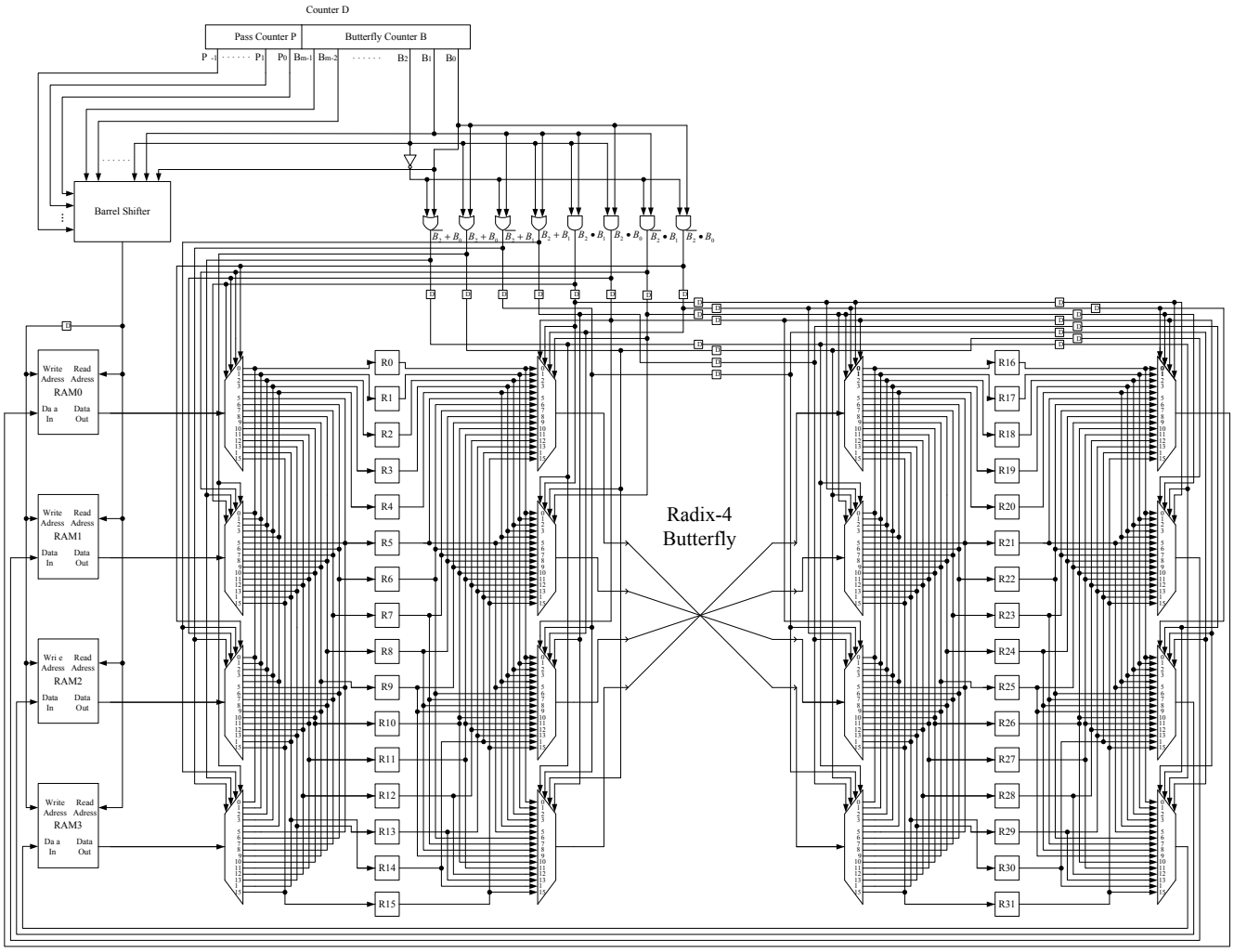



Figure 3. Address generation circuit for Radix-4 FFT

Fig. 3 shows the proposed scheme for *N*-point radix-*4* FFT processor. Table I lists the address sequence of the first register set, whereas Table II lists the address sequence of the second register set (assume the butterfly calculation delay is four clock cycles). Together, they present the address sequence order for *pass 1* of 64-point radix-4 FFT. For other passes, the sequence tables for the register sets are similar. Tables I and II show that for different clock cycles, the data in the registers follow a very regular sequence and the hardware components to realize these sequences are very simple: After logic minimization, it results in only primitive logic gates such as AND/OR gates using the least significant three bits of the butterfly counter *B* (see Fig. 3).

Other main components of the FFT processor are Counter D and the barrel shifter. Counter D has two parts; pass counter *P* which is $v=log_4N$ bits ($P_{v-1}$ to $P_0$) and butterfly counter *B* which is $m = \left\lceil \log_2 \frac{N}{4} \right\rceil$ bits ($B_{m-1}$ to $B_0$).

The barrel shifter generates all the addresses for four memory banks based on the *pass number* of the FFT, which can be expressed as:

$$RR(counter\ B,\ 2p) \tag{2}$$

where RR(counter B, 2p) means rotate-right butterfly counter *B* by *2p* bits, and *p* is the pass number of FFT.

For twiddle factors *Wb, Wc* and *Wd*, three memory banks are used with same address generation logic. For *pass p*, this address is given as:

$$B_{m-1}B_{m-2}...B_{2p}000...0 \quad (2p\ 0\text{'s follow}) \tag{3}$$

For different length FFT transforms, the control logic of the multiplexers only depends on the last three bits of the counter (see Fig. 3), so the register and multiplexer structures are fixed for different length FFTs resulting in a common architecture for any *N*-point FFT.

TABLE III. SYNTHESIS RESULTS WITH CMOS 0.18μm TECHNOLOGY FOR DIFFERENT LENGTH FFT
(32-BIT COMPLEX DATA: 16-BIT EACH FOR THE REAL AND IMAGINARY PART)

| | 64-point FFT | | | 256-point FFT | | | 1024-point FFT | | |
|---|---|---|---|---|---|---|---|---|---|
| **Area** | Total | Memory + Butterfly units | Address Generation unit | Total | Memory + Butterfly units | Address Generation unit | Total | Memory + Butterfly units | Address Generation unit |
| | 36662 cells | 27591 cells | 9071 cells | 65547 cells | 56915 cells | 8632 cells | 176746 cells | 168199 cells | 8547 cells |
| **Delay** | 5.47 ns | | | 5.49 ns | | | 5.48 ns | | |

Table III shows Synopsys Design Compiler synthesis results using TSMC CMOS 0.18μm technology. For different length FFTs, the memory usage scales proportional to transform length, but the address generation circuit sizes are almost same; confirming that method is extremely efficient for long length FFT transforms. Compared to a radix-2 FFT implementation given in [8], the throughput is faster by a factor of 4.

## IV. CONCLUSION

The proposed method for radix-4 FFT avoids any addition in the address generation, enabling a fast datapath for butterfly operations. The same concept can be extended to any radix FFT, but the amount of registers and multiplexers for different radix FFT will be different: For radix-$r$ FFT, $2r^2$ registers and $4r$ multiplexers are needed.

## REFERENCES


[1] S. Mittal, Z.A. Khan, and M.B. Srinivas, "Area efficient high speed architecture of Bruun's FFT for software defined radio", *IEEE Global Telecommunications Conference GLOBECOM '07,* pages 3118 -3122, November 2007.

[2] L. Xiaojin, L. Zongsheng Lai, and C. Jianmin Cui, "A low power and small area FFT processor for OFDM demodulator", *IEEE Transactions on Consumer Electronics,* 53(2):274-277, May 2007.

[3] M. C. Pease, "Organization of large scale Fourier processors", *J. Assoc. Comput. Mach.,* 16:474–482, July 1969.

[4] D. Cohen, "Simplified control of FFT hardware", *IEEE Trans. Acoust., Speech, Signal Processing*, 24:577-579, December 1976.

[5] L.G. Johnson, "Conflict free memory addressing for dedicated FFT hardware", *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 312-316, May 1992.

[6] J.H. Takala, T.S. Jarvinen, and H.T. Sorokin, "Conflict-free parallel memory access scheme for FFT processors", *Proc. of the International Symposium on Circuits and Systems,* ISCAS '03, vol. 4, pages 524-527, May 2003.

[7] Y. Ma, "An effective memory addressing scheme for FFT processors", *IEEE Trans. on Signal Process*,47(3): 907–911, March 1999.

[8] X. Xiao, E. Oruklu, J. Saniie, "An Efficient FFT Engine With Reduced Addressing Logic", *IEEE Transactions on Circuits and Systems II Express Briefs,* vol. 55, no 11, pp. 1149 - 1153, November 2008.