Efficient FFT Engine with Reduced Addressing Logic

Xin Xiao, Erdal Oruklu, and Jafar Saniie

Electrical and Computer Engineering Department
Illinois Institute of Technology
Chicago, Illinois 60616

Abstract- In this study, an efficient address generation method for pipelined Radix-2 fast Fourier transform (FFT) is presented. Conventional pipelined FFT addressing schemes utilize dedicated address generators in order to enable parallel access to the memory units and in-place calculation. These address generator units require substantial hardware logic including counters and rotational shifters to generate the addresses and registers to buffer the outputs of each butterfly operation. In the proposed method, the new parallel addressing scheme utilizes only counters and inverters to generate the addresses; replacing the barrel shifter units and reducing the hardware requirements. In order to replace the barrel shifters, two outputs of the butterfly unit switch places depending on the status of the butterfly pass operation. The only additional hardware unit is extra multiplexers for switching the butterfly outputs. We present the signal flow graph for 8, 16, and 32-point FFT and derive the methodology for $2^r = N$-point transforms. Furthermore, as a case study, 16-point FFT with 64-bit complex numbers is synthesized using an FPGA device. The results indicate that approximately 20% logic reduction can be achieved with exactly same throughput.

I. INTRODUCTION

Many high-performance FFT processors use a butterfly calculation unit to realize Radix-2 FFT, and the in-place strategy is a practical choice to minimize the amount of memory. With in-place method, the outputs of a butterfly are stored within the same memory location used by the inputs. If this strategy is not possible due to inefficient data ordering, the amount of implementation memory would be doubled. This bears a significant problem especially for large (> 1024-point) transform implementations. In this study, reduced logic addressing schemes for in-place radix-2 decimation in-frequency type FFT is examined. The discrete Fourier transform of the $N$-point is defined by

$$X(k) = \sum_{n=0}^{N-1} x(n)W_n^k$$

$$k = 0,1,...,N-1, W_n = e^{-j\frac{2\pi}{N}}$$

(1)

Fig. 1 shows the signal flow graph of 16-point FFT algorithm [1]. Conventional Radix-2 butterfly operation is shown in Figure 2 and Equation 2.

$$Xa' + iYa' = Xa + Xb + i(Ya + Yb);$$
$$Xb' + iYb' = (Xa - Xb)W + (Ya - Yb)W;$$
$$+ i[(Xa - Xb)W - (Ya - Yb)W];$$

(2)

From Fig. 2, it can be seen that any butterfly operation has two data inputs and two data outputs. Therefore, parallel data access with in-place strategy necessitates the use of a block of 4-port memory for simultaneous two data read and two data write operations. However 4-port memory blocks are costly and inefficient. In most of the cases, only two-port high-speed memories are provided in hardware, and
parallel addressing can be achieved using two banks of two-port memories by a special addressing algorithm.

II. RELATED WORK

Memory segmentation and simplified control logic have been studied in [2], [3], [4]. In particular, Ma [5], [6], [7] proposed a method to realize parallel addressing of radix-2 FFT by using two banks of two-port memories. The signal flow graph of 16-point FFT with Ma’s [6] method is shown in Fig. 3. (For signal flow graphs, the following expression is used: if the two inputs of the butterfly is x and y, we express the two outputs as xy and yx). In this method, memory is composed of two separate memory banks M0 and M1, and each memory bank is a two-port RAM, which can read one datum and write one datum simultaneously at one time. Each input of any butterfly unit arrives from different memory banks, which indicates that the inputs can be accessed in parallel. However, butterfly outputs are written to the same memory bank (see Fig. 3); hence, registers are necessary to buffer the signals before parallel write back to the memory. Multiple counters and barrel shifters are utilized to generate all the read and write addresses. Table I shows the address generation for this implementation. RR(x,i) operations denote rotating x right over i bits and they are implemented via shifter blocks.

III. REDUCED ADDRESS GENERATION LOGIC FOR FFT

The drawback to the method given in [6] are the requirements that the barrel shifters and the extra buffers are used for writing two simultaneous outputs to memory. However, the addresses of the two outputs of a butterfly could be exchanged, and parallel data access can be realized without any registers or any delay. The addresses of butterflies’ outputs will be exchanged based on the status of the butterfly pass. The resulting signal data flow is very similar to standard FFT butterfly operation. (See Fig. 2 for the standard butterfly structure). Fig. 4 shows the new butterfly design whose two outputs have been exchanged.

<table>
<thead>
<tr>
<th>Counter B</th>
<th>Pass 0 [RR(b,1)]</th>
<th>Pass 1 [RR(b,2)]</th>
<th>Pass 2 [RR(b,3)]</th>
<th>Pass 2 [RR(b,4)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>100</td>
<td>010</td>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>100</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>011</td>
<td>101</td>
<td>110</td>
<td>011</td>
<td>101</td>
</tr>
<tr>
<td>100</td>
<td>010</td>
<td>001</td>
<td>100</td>
<td>010</td>
</tr>
<tr>
<td>101</td>
<td>110</td>
<td>011</td>
<td>101</td>
<td>110</td>
</tr>
<tr>
<td>110</td>
<td>011</td>
<td>101</td>
<td>110</td>
<td>011</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>111</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

Fig. 3. 16-point FFT Signal Flow Graph in [6]

Fig. 4. Butterfly Operation with Outputs Exchanged

We can build a new butterfly structure whose two outputs can be exchanged by a control signal C, the two outputs will exchange their addresses when C=0, or they will not change their addresses when C=1. Fig. 5 shows this kind of new butterfly where:

If \( C = 0 \):
\[ Z_a = Xb+iYb, Z_b = Xa+iYa; \]
Else:
\[ Z_a = Xa+iYa, Z_b = Xb+iYb; \]
Fig. 6. Reduced Addressing Scheme for 8, 16, and 32-point FFT
Table II: Address Generation Table of the Proposed Method for 16-point FFT

<table>
<thead>
<tr>
<th>Counter $B(b_2b_1b_0)$</th>
<th>Counter $\overline{B}(\overline{b_2}\overline{b_1}\overline{b_0})$</th>
<th>Pass 0 (exchange control signal: $C=\overline{b_2}$)</th>
<th>Pass 1 (exchange control signal: $C=\overline{b_1}$)</th>
<th>Pass 2 (exchange control signal: $C=\overline{b_0}$)</th>
<th>Pass 3 (exchange control signal: $C=1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank 1 address $b_2b_1b_0$</td>
<td>Bank 1 address $\overline{b_2b_1b_0}$</td>
<td>Bank 1 address $b_2b_1b_0$</td>
<td>Bank 1 address $\overline{b_2b_1b_0}$</td>
<td>Bank 1 address $b_2b_1b_0$</td>
<td>Bank 1 address $\overline{b_2b_1b_0}$</td>
</tr>
<tr>
<td>000</td>
<td>111</td>
<td>000</td>
<td>000</td>
<td>100</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>110</td>
<td>001</td>
<td>001</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>101</td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>110</td>
</tr>
<tr>
<td>011</td>
<td>100</td>
<td>011</td>
<td>011</td>
<td>111</td>
<td>011</td>
</tr>
<tr>
<td>100</td>
<td>011</td>
<td>100</td>
<td>100</td>
<td>000</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
<td>101</td>
<td>101</td>
<td>000</td>
<td>101</td>
</tr>
<tr>
<td>110</td>
<td>001</td>
<td>110</td>
<td>110</td>
<td>110</td>
<td>010</td>
</tr>
<tr>
<td>111</td>
<td>000</td>
<td>111</td>
<td>111</td>
<td>111</td>
<td>011</td>
</tr>
</tbody>
</table>

Table III: Address Generation of Twiddle Factors for 16-point FFT

<table>
<thead>
<tr>
<th>Counter $B(b_2b_1b_0)$</th>
<th>W address of Pass 0 $b_2b_1b_0$</th>
<th>W address of Pass 1 $b_0b_0b_0$</th>
<th>W address of Pass 2 $b_0b_0$</th>
<th>W address of Pass 3 $b_00$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>010</td>
<td>100</td>
<td>000</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
<td>100</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>011</td>
<td>011</td>
<td>110</td>
<td>100</td>
<td>000</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>100</td>
<td>100</td>
<td>000</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
<td>100</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>110</td>
<td>100</td>
<td>000</td>
</tr>
</tbody>
</table>

A. 8, 16 and 32-point FFT examples

Fig. 6 shows signal flow graphs for 8, 16, and 32-point FFT realized by the proposed method. In Figure 6(b), the straight lines indicate the butterflies without output address exchanging and the broken line indicates the butterflies whose two outputs addresses are exchanged. The memory is also composed of two separate memory banks M0 and M1. We can use a single three-bit counter, three inverters and one three-bit shifter to generate all the addresses required for the 16-point FFT: $CounterB$ counts every clock cycle, the inverters generate an inverted count value $Counter\overline{B}$. 2-bit $CounterP$ is used to indicate which butterfly pass is in process, and the three-bit shifter is used to combine certain bits of $counterB$ and $\overline{B}$. Table II and III shows the details of the implementation. The proposed method needs only one counter, one multi-bit inverter and one multi-bit shifter to generate all the addresses needed and the control signal $C$, therefore the structure of the address generator is reduced.

B. N-point FFT Implementation

For $N = 2^r$ point FFT, we need an (r-1)-bit counter:

$$B = b_{r-2}b_{r-3}...b_1b_0$$

(3)

and an (r-1)-bit inverter to generate an inverse counter:

$$\overline{B} = \overline{b_{r-2}}\overline{b_{r-3}}...\overline{b_1}\overline{b_0}$$

(4)

as well as a pass counter

$$P = 0, 1, 2, r-1$$

(5)
to indicate which butterfly pass is in progress. At any pass, the read and write address of memory bank M0 is exactly the same as the value of counter B, and at pass s, the read and write address of memory bank M1 is
Pass Counter P  
Butterfly Counter B

\[
\overline{b}_{r-2}\overline{b}_{r-3}...\overline{b}_{r-s-1}b_{r-s-2}...b_b \tag{6}
\]
which is a combination of counter B and inverse counter \(\overline{B}\) outputs, the \((r-1)\)-bit shifter is used to generate the control signals for the multiplexer which combine the counter B and \(\overline{B}\). The control signal \(C\) (it decides whether or not to exchange the positions of the two outputs of any butterfly) is equal to \(b_{r-s-2}\) when pass number \(s\) is even, or equal to \(\overline{b}_{r-s-2}\) when pass number \(s\) is odd, and \(C = 1\) at the last pass \(r-1\). And the address of twiddle factors at \(s\) pass is \(b_{r-s-2}b_{r-s-3}...b_0 0...0\) (s ‘0’s, and when \(s = r-1\), it’s all ‘0’s).

Figure 7(a) and 7(b) show the circuit diagram of the read and write address generators for 1024-point FFT.

IV. IMPLEMENTATION

The proposed FFT algorithm is implemented by CMOS technology, and we evaluate the logic complexity similar to [6] with the sizes of some basic circuit and gates listed in Table IV.

Gate count comparison for 1024-point FFT processor of 40-bit complex data (20-bit each for the real part and imaginary part) is shown in the Table V.

An FPGA chip Stratix EP1S40F1508C5 has been used to simulate 16-point FFT by these two methods, the results are shown in Table VI.
TABLE VI
FPGA SYNTHESIS RESULTS FOR 16-POINT FFT

<table>
<thead>
<tr>
<th>Resource</th>
<th>Proposed method</th>
<th>Ma’s method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total logic elements</td>
<td>885</td>
<td>1012</td>
</tr>
<tr>
<td>Total LABs (Logic Array Block)</td>
<td>125</td>
<td>150</td>
</tr>
<tr>
<td>Total memory bits</td>
<td>1,536</td>
<td>1,536</td>
</tr>
<tr>
<td>DSP block 9-bit elements</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Clk frequency</td>
<td>148.26MHz</td>
<td>148.26MHz</td>
</tr>
</tbody>
</table>

V. SUMMARY

FFT is a fundamental tool for many signal processing and communications applications. Therefore, many methods had been proposed to realize conflict free memory addressing of FFT, however most methods try to reorder the addresses of the butterfly inputs or outputs to realize the parallel accessing of the memory. These implementations require shifters to build their address generators and they buffer the interim data to realize the in-place strategy. This paper proposes a reduced addressing logic by building a new butterfly unit whose two outputs’ positions can be exchanged. Based on this concept, we deduce a new addressing scheme to realize parallel access of butterfly addresses. Our method have two advantages: first, proposed method use inverters instead of shifters to generate the addresses, second, because any butterfly operation in our scheme is in-place by default, we don’t need any registers to buffer the interim data. These changes do not impact the throughput of the FFT operation, however it improves area and power metrics. For a case study, we have implemented 16-point FFT with a Stratix FPGA. The results show approximately 20% reduction in logical blocks.

REFERENCES