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 0.18\(\mu\)m 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.

II. RADIX-4 FFT

The \(N\)-point discrete Fourier transform is defined by

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

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 \(\frac{N}{r}\log_N N\) 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.
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.

### 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.
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 counter B (see Fig. 3).

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

\[ RR(\text{counter } B, 2p) \]

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

For twiddle factors \( W_b, W_c \) and \( W_d \), three memory banks are used with same address generation logic. For pass \( p \), this address is given as:

\[ B_{m-1}B_{m-2}…B_{2p} \quad 000…0 \quad (2p \text{'s follow}) \]

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 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.

<table>
<thead>
<tr>
<th>Area</th>
<th>64-point FFT</th>
<th>256-point FFT</th>
<th>1024-point FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Memory + Butterfly units</td>
<td>Address Generation unit</td>
</tr>
<tr>
<td>36662 cells</td>
<td>27591 cells</td>
<td>9071 cells</td>
<td>65547 cells</td>
</tr>
<tr>
<td>Delay</td>
<td>5.47 ns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REFERENCES


