Design of Efficient Radix-8 Butterfly PEs for VLSI

T. Widhe, J. Melander, and L. Wanhammar
Department of Electrical Engineering
Linköping University, Linköping, Sweden
email: torwi@isy.liu.se

Abstract - In this paper we discuss various aspects of VLSI implementation of radix-8 Sande-Tukey butterfly PEs. The Sande-Tukey algorithm is investigated to find regularities and simplifications. We argue that a bit-serial implementation of the butterfly PEs will be advantageous from a area point of view under most circumstances. The ideas are supported by a design example, a 512 point FFT/IFFT processor aimed at OFDM applications has been designed and will soon be implemented.

I. Introduction

Fourier transforms are among the most fundamental operations in digital signal processing today and new applications continue to arise [1]. Hence, efficient implementations of the fast Fourier transform (FFT) are of fundamental importance. The tremendous evolution of VLSI technology has changed the character of the problems faced by the FFT hardware designer. In the past, maximizing the speed was the main goal and huge FFT machines were built [2] consuming enormous amount of power. Today, it is possible to reach almost any desired throughput if no constraints concerning maximum power dissipation have to be met by the FFT processor. The real problem of today is to reach the desired throughput while dissipating a minimal amount of power. It is not only the battery powered systems, where battery lifetime is of vital important, that has put the power consumption i focus, it is also important in all systems when considering economic and environmental aspects.

A lot of FFT algorithms exists, e.g. Cooley-Tukey, Sande-Tukey, Split-radix, Prime Factor Algorithm (PFA) and Winograd [3]. Some of them are irregular or unsuitable for hardware implementation for other reasons. The Cooley-Tukey and Sande-Tukey algorithms is regular, can be computed *in place*, and thus, are suitable for VLSI implementations. The vast majority of the implementations of the Cooley-Tukey and Sande-Tukey algorithm has used the radix-2 or radix-4 versions of the algorithm, although higher radices has obvious advantages, e.g. less memory accesses and less complex multiplications, see Tab. 1. The reason is probably that higher radix implementation requires high complexity processing elements (PEs) and poses restrictions on the transform length, N. (N = radix^k, k is an integer.)

The butterfly is the basic operation in the FFT algorithm and consists basically of an adder-tree and twiddle-factor multiplications. The size and complexity of the adder-tree as well as the number of twiddle-factor multiplications depend on the radix. Thus a high radix butterfly PE is significantly more complex to design than a low radix PE.

Operation	radix-2	radix-4	radix-8
Complex multiplications	22528	15360	10752
Real multiplications	0	0	8192
Complex additions	49152	49152	49152
Real additions	0	0	8192
Memory accesses	49152	24576	16384

Table 1. Workload for a 4096 point FFT using different radices.

In this paper we will discuss the design of a radix-8, Sande-Tukey, butterfly PE and present an example of such a design aimed at a 512 point FFT processor.

II. Applications

By application for a butterfly PE we mean a dedicated FFT processor. Numerous different architectures for FFT processors exists with different advantages and disadvantages, but most of them incorporate one or more butterfly PEs. In this paper we will just briefly discuss two types of architectures, the *pipelined* architecture and the *shared memory* architecture.

A. *Pipelined Architecture*

The pipelined architecture [4] is probably the most common FFT processor architecture. There are a number of different variations but the basic principle is shown in Fig. 1.

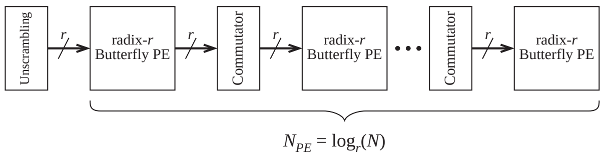

Fig. 1. Pipelined architecture.

Implementations using this architecture are typically high throughput. One disadvantage is that the minimum number of PEs, N_{PE}, is determined by the number of stages in the FFT algorithm, i.e. dependent on the radix and the transform length.

B. *Shared memory architecture*

The shared memory architecture [5], shown in Fig. 2, has the advantage that the number of PEs can be chosen freely to meet the throughput requirements.

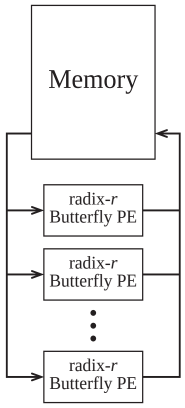

Fig. 2. Shared memory architecture.

The problem with this architecture is the memory bandwidth which can be high in high throughput applications. In [6] solutions to this problem is discussed.

III. The Radix-8 Algorithm

The discrete Fourier transform DFT is defined by

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

Let $\mathbf{x} = [x(0), x(1), \ldots, x(N\text{-}1)]^T$ be the input signal sequence and $\mathbf{X} = [X(0), X(1), \ldots, X(N\text{-}1)]^T$ be the output signal sequence. Then

$$\mathbf{X} = \mathbf{F_N} \cdot \mathbf{x}$$

where $\mathbf{F_N}$ is the $N\times N$ DFT matrix whose elements are defined by $[\mathbf{F_N}]_{ij} = W_N^{ij}$ (i, j $\in$ [0, N-1]). The radix-8 Sande-Tukey FFT algorithm can be expressed by

$$\mathbf{P_N} \cdot \mathbf{F_N} = \mathbf{C_1} \cdot \mathbf{C_2} \cdot \ \ldots \ \cdot \mathbf{C_s} \qquad (N = 8^s)$$

where

$$\mathbf{C_q} = \mathbf{I}_{\frac{\mathbf{N}}{\mathbf{8^q}}} \otimes \mathbf{D_{8^q}} \qquad q \in [1,\ s]$$

$$\mathbf{D_{8^q}} = \mathbf{diag}(\Delta^{\mathbf{0}}_{\mathbf{8^{q-1}}}, \Delta^{\mathbf{1}}_{\mathbf{8^{q-1}}}, \ \ldots \ , \Delta^{\mathbf{7}}_{\mathbf{8^{q-1}}}) \cdot \mathbf{F_8} \otimes \mathbf{I_{8^{q-1}}}$$

$$\Delta^{\boldsymbol{i}}_{\mathbf{8^{q-1}}} = \mathbf{diag}(1, W^{i}_{8^q}, W^{2i}_{8^q}, \ \ldots \ , W^{(8^{q-1}-1)i}_{8^q})$$

$\mathbf{P_N}$ is a $N\times N$ permutation matrix, $\mathbf{I_L}$ is a $L\times L$ identity matrix and $\otimes$ is the Kronecker product. The matrix $\mathbf{C_s}$ correspond to stage 1, $\mathbf{C_{s\text{-}1}}$ to stage 2 etc.

From this formulation of the radix-8 algorithm we can identify $\mathbf{F_8}$ as the *butterfly* without twiddle-factors and $\mathbf{diag}(\Delta^{\mathbf{0}}_{\mathbf{8^{q-1}}}, \Delta^{\mathbf{1}}_{\mathbf{8^{q-1}}}, \ \ldots \ , \Delta^{\mathbf{7}}_{\mathbf{8^{q-1}}})$ as the twiddle-factors in stage s-q+1. By doing some trivial transformations e.g. $W_8^2 = -j$ we get

$$\mathbf{F_8} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8 & -j & -jW_8 & -1 & -W_8 & j & jW_8 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -jW_8 & j & W_8 & -1 & jW_8 & -j & -W_8 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -W_8 & -j & jW_8 & -1 & W_8 & j & -jW_8 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & jW_8 & j & -W_8 & -1 & -jW_8 & -j & W_8 \end{pmatrix}$$

This butterfly requires 56 complex additions. However, by factoring the matrix we can perform the same computations using only 24 complex additions.

$$\mathbf{F_8} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -j \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & j \\ 0 & 1 & 0 & 0 & 0 & -j & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \end{pmatrix} \cdot$$
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & j & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -j & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -jW_8 & 0 & 0 & 0 & W_8 \\ 0 & 0 & 0 & jW_8 & 0 & 0 & 0 & -W_8 \end{pmatrix} \cdot$$
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix}$$

By doing this we get the same number of complex additions as in the radix-2 case, i.e. if 12 radix-2 butterflies would have been used as one radix-8 butterfly. A similar factorization can be performed for the radix-4 butterfly resulting in exactly the same number of complex additions, see Tab. 1.

Another noteworthy property of the algorithm is that the twiddle-factors in the last stage, $\mathbf{C_1}$, are all equal to 1. These multiplications are usually referred to as trivial multiplications and are omitted in Tab. 1.

IV. Design Considerations

The signal flow graph (SFG) of a butterfly implementing $\mathbf{F_8}$ is shown in Fig. 3. Note that all the signals in the SFG are complex valued. The inputs and the outputs of the butterfly are not in the natural order due to drawing considerations. Note that this order is also suitable in an VLSI implementation because the order minimize the length of signal wires.

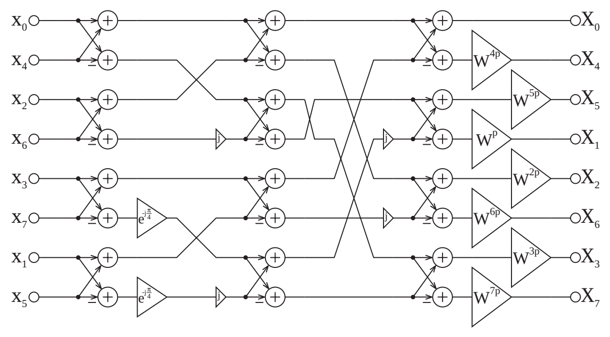

Fig. 3. SFG of the radix-8 butterfly.

The twiddle-factor multiplications, W^{ip}, are included in the SFG for the sake of completeness. We can see that the adder-tree is modular and that the interconnections are local.

A. *Modular adder-tree*

From the SFG it can be seen that the adder-tree is built up by *two-adder modules*. Figure 4 shows the two-adder module with complex valued signals and the corresponding real valued signal implementation.

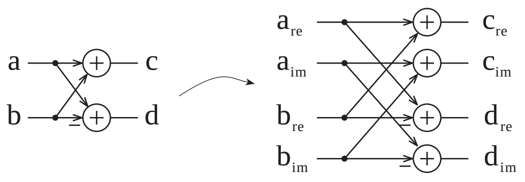

Fig. 4. Two-adder module.

This modularity in the adder-tree is a useful property when it comes to VLSI implementation of the butterfly.

B. *Multiplications in the adder-tree*

The constant multiplications, by *j* and $e^{-j\pi/4}$, in the adder tree can be implemented efficiently.

The four multiplications by *j* (= sqrt(-1)) are easily incorporated in the two-adder module by interchanging inputs and outputs as shown in Fig. 5, and thus, resulting in no extra hardware.

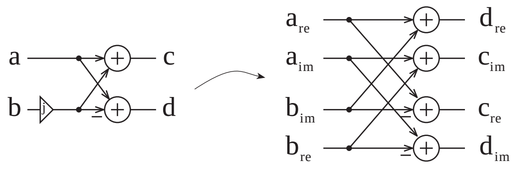

Fig. 5. Multiplication by *j* incorporated in the two-adder module.

The two multiplications by $e^{-j\pi/4}$ can not be implemented without hardware cost. One solution is shown in Fig. 6. The cost of this implementation is two real multipliers and two real adders.

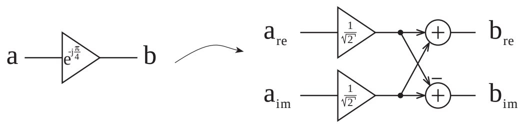

Fig. 6. Complex multiplication by $e^{-j\pi/4}$.

Note that the real multiplications have a constant coefficient and, thus, are less expensive to implement than ordinary, variable coefficient, multipliers.

C. *Twiddle-factor generation*

By studying the radix-8 algorithm, it can be seen that each of the complex multipliers (W^{ip} in Fig. 3) only multiplies with *N*/8 different twiddle-factors each. This means that in a VLSI implementation we can have a dedicated local ROM with *N*/8 complex words, representing the twiddle factors, for each complex multiplier. All the twiddle-factors can then be read out of the ROM if *p* is known, this means that we only need to supply the butterfly with the value of *p*. Table 2 shows which values of W^p that is stored in each ROM.

Multiplier	W^p stored in local ROM. Value of *p* listed.
W^p	0, 1, 2, 3, .. , 63
W^{2p}	0, 2, 4, 6, .. , 126
W^{3p}	0, 3, 6, 9, .. , 189
W^{4p}	0, 4, 8, 12, .. , 252
W^{5p}	0, 5, 10, 15, .. , 315
W^{6p}	0, 6, 12, 18, .. , 378
W^{7p}	0, 7, 14, 21, .. , 441

Table 2. Organization of the ROMs.

D. *Bit-serial PE*

In spite of the simplifications presented in this paper the cost is high in terms of silicon area and power consumption when implementing a PE.

One way to reduce the silicon area is to use bit-serial arithmetic in the implementation, i.e to use bit-serial adders/subtractors and serial-parallel multipliers (i.e. presenting the coefficients to the multipliers in parallel). A bit-serial implementation is likely to have lower throughput than a bit-parallel one, however, the area is less than in a bit-parallel solution. Assume that a parallel PE has an area of $A_{parallel}$ and a maximal throughput of $1/T_{parallel}$ and that a bit-serial PE has an area of A_{serial} and a maximal throughput of $1/T_{serial}$. To replace a parallel PE, operating at maximal speed, $T_{serial}/T_{parallel}$ bit-serial PEs are needed. If the reduction in area $A_{parallel}/A_{serial}$ is lower than $T_{serial}/T_{parallel}$ there is no idea to use bit-serial PEs since the area will increase instead. However, in the cases investigated by us, the area reduction is larger than the reduction in throughput favoring a bit-serial approach. Note also that if the maximum throughput of a parallel PE in not needed the area reduction using bit-serial PEs is even bigger.

In [7] we propose a bit-serial implementation with a parallel interface *hiding* the bit-serial implementation from the rest of the circuit. The implementation includes a local clock generator to generate the bit-serial clock and shift-registers to convert between bit-serial and bit-parallel data.

The cost for the butterfly PE, in terms of silicon area, power consumption etc., is likely to be dominated by the 7 complex multipliers, i.e. the twiddle factor multipliers. A direct implementation of a complex multiplication requires four real multiplications and two additions. This can easily be reduced to 3 multiplication at the cost of more additions and increased latency. In [5] it is shown that two distributed arithmetic (DA) units are sufficient. If the DA units are implemented using serial/parallel shift-accumulators, each of them will have approximately the same complexity as a real serial/parallel multiplier.

The fact that all complex multiplications in the last stage are equal to 1 makes it possible to reduce the power consumption by bypassing the complex multipliers and gating their clocks in this stage.

V. Example

The floorplan of a bit-serial butterfly PE is shown in Fig. 7. The design is aimed at a 0.8 µm, standard CMOS process and all the blocks in the design are implemented using the TSPC [8] logic style. It can be seen that the design is dominated by the complex multipliers even though they are implemented using distributed arithmetic. Note that the twiddle factor ROMs are local to each complex multiplier. The shimming-delays in the center of the floorplan are needed to compensate for the latency in the multipliers.

Since the design is aimed at a 512 point FFT the size of each ROM is 64 words. The butterfly runs at 125 MHz at 3 V supply voltage and the internal word length is 17 bits. The FFT processor computes a 512 point FFT in 20 µs and to meet these requirements 2 butterfly PEs are used.

VI. Conclusion

Various aspects of VLSI implementations of radix-8 butterfly PEs have been investigated. In spite of the high complexity of the implementations we regard higher radix FFT algorithms as an attractive alternative for high performance FFT processors. By utilizing regularities in the algorithm we can minimize the complexity of the implementation and by using bit-serial butterfly PEs area and power can be saved.

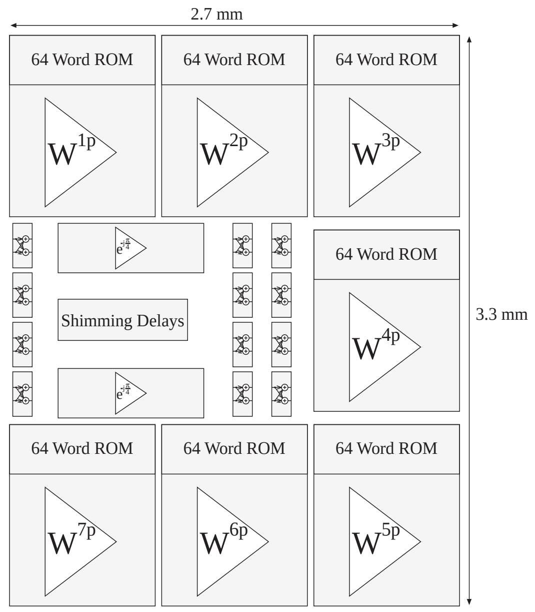

Fig. 7. Floorplan of a radix-8 butterfly PE.

References

[1] O. E. Brigham, *The Fast Fourier Transform and its Applications*, Prentice Hall, 1988.

[2] G. Bergland, "Fast Fourier Transform Hardware — A Survey", IEEE Trans. Audio and Electro-acoustics, vol. AU-17, pp. 109 - 119, June 1969.

[3] P. Duhamel and M Vetterli, "Fast Fourier Transforms: A Tutorial review and a state of the art", Signal Processing, Vol. 19, No. 4, pp. 259-299, April 1990.

[4] S. F. Gorman and J. M. Wills, "Partial Column FFT Pipelines", IEEE Trans. on Circuits and Systems-II, Vol. 42, No. 6, June 1995.

[5] L. Wanhammar, *DSP Integrated Circuits*, Linköping University, prepublication edition, 1996.

[6] J. Melander, T. Widhe, and L. Wanhammar, "A Radix-*r* FFT/IFFT Architecture With Distributed Control Unit", Proc. of NORSIG, Espoo, Finland, September 1996.

[7] J. Melander, T. Widhe, and L. Wanhammar, "An FFT Processor Based on the SIC Architecture with Asynchronous PE", Proc. MWSCAS'96, Iowa, USA, August, 1996.

[8] J. Yuan and C. Svensson, "High Speed CMOS Circuit Technique", IEEE Journal of Solid-States Circuits, vol. 24, pp. 62-70, 1989.