BUTTERFLY UNIT SUPPORTING RADIX-4 AND RADIX-2 FFT

Jarmo Takala and Konsta Punkka

Tampere University of Technology,
P.O.Box 553, FIN-33101 Tampere, FINLAND, jarmo.takala@tut.fi, konsta.punkka@tut.fi

ABSTRACT

This paper considers partial-column radix-2 and radix-2/4 FFT processors and realizations of butterfly operations. The proposed processor organization allows the area of the FFT implementation to be traded against the computation time, thus the final structure can be easily tailored according to the requirements of the given application. The key properties, e.g., area and power consumption, of the FFT processor depend mainly on the implementation of butterfly operations. The butterfly operation can be realized with several different principles. Radix-2 butterfly units based on bit-parallel multipliers, coordinate rotations (CORDIC), and distributed arithmetic (DA) are described with VHDL and synthesized on the same state of the art standard cell ASIC technology than the proposed radix-2/4 butterfly units. After this the area and energy-efficiency of the different alternatives are compared. The comparison shows that butterfly units based on bit-parallel multipliers are energy-efficient. This energy-efficiency can be further enhanced by utilizing radix-4 operation. The length of the critical path, i.e., delay is smallest for butterfly units based on distributed arithmetic. If very long FFTs are needed, the CORDIC based butterfly units are applicable.

1. INTRODUCTION

Discrete Fourier transform (DFT) is one of the most important tools in the field of digital signal processing. Due to its computational complexity, several fast Fourier transform (FFT) algorithms have been developed over the years. The most popular FFT algorithms are the Cooley-Tukey algorithms originally proposed in [1]. It has been shown that the decimation-in-time (DIT) algorithms provide better signal-to-noise-ratio than the decimation-in-frequency algorithms when finite word length is used [2]. In addition, in-place algorithms are often used due to the principal simplicity of the operand storage; the memory locations holding the operands for a computational recursion are replaced by the results of recursion.

Based on the previous, the most popular Cooley-Tukey FFT is the radix-2 DIT FFT with bit-reversed input and in-order output. A 2^k-point FFT can be presented with tensor product formulation as [3]

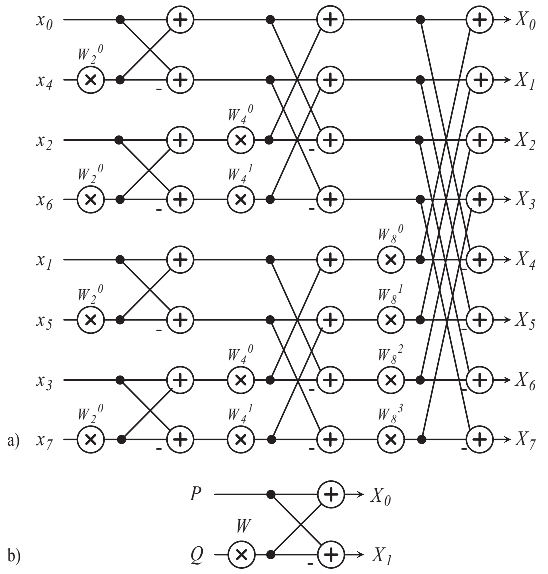


Figure 1. (a) Signal flow graph of 8-point radix-2 DIT FFT (b) radix-2 DIT butterfly operation.

$$\begin{aligned} F_{2^k}^{DIT} &= \Bigg[\prod_{i=0}^{k-1} (I_{2^i} \otimes F_2 \otimes I_{2^{k-i-1}}) \\ &\qquad (I_{2^i} \otimes T_{2^{k-i}})\Bigg] P_{2^k}^r \\ T_J &= (I_{J/2} \oplus D_{J/2}) \\ D_{J/2} &= diag(W_J^k), \quad 0 \le k < J/2 \\ F_2 &= \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \end{aligned} \tag{1}$$

where $\oplus$ and $\otimes$ denote Kronecker sum and product, respectively, I_k is the identity matrix of order k, and F_2 is the 2-point DFT matrix. P_N^r is a bit-reversed permutation matrix of order N reordering a vector $X = (x_0, x_1, ..., x_{N-1})^T$ to a vector $Y = (x_{rev(0)}, x_{rev(1)}, ..., x_{rev(N-1)})^T$, where the bit-reverse function rev(), reverses the order of the bits in the binary representation of the element indices. The complex coefficients W_J^k are called twiddle factors and are defined as

$$W_J^k = exp(\frac{-j2\pi k}{J}) \tag{2}$$

where j is the imaginary unit. The signal flow graph of 8-point radix-2 DIT FFT with bit-reversed input and

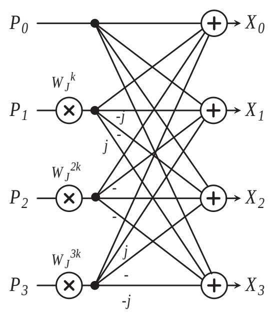


Figure 2. Signal flow graph of radix-4 DIT butterfly.

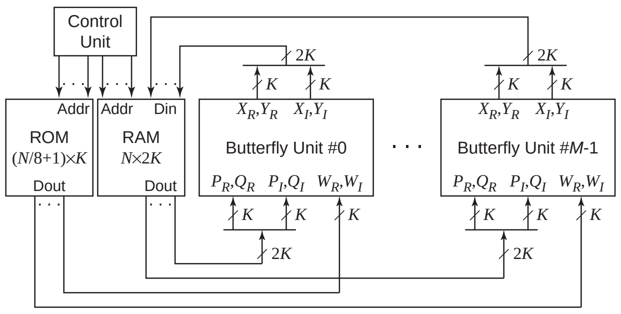


Figure 3. Block diagram of partial-column FFT processor.

in-order output is illustrated in Figure 1 (a). The arithmetic kernel of radix-2 DIT FFT is the butterfly operation defined as

$$X_0 = P + WQ; \quad X_1 = P - WQ \tag{3}$$

The signal flow graph of radix-2 DIT butterfly operation is shown in Figure 1 (b).

FFTs in which $N{=}r^k$, and where the butterflies used in each stage are the same, are called radix-r algorithms. A radix-r FFT uses N/r radix-r butterflies for each stage and has $log_r N$ stages. For example, in Figure 1 (a) there is $N/r = 4$ butterflies in each of the $log_r N = 3$ stages. FFTs in which the radices of butterflies are not equal are called mixed-radix FFTs. A 2^k-point radix-4 DIT FFT can be presented with tensor product formulation as

$$\begin{aligned} F_{4^k}^{DIT} &= \left[\prod_{i=0}^{k-1} (I_{4^i} \otimes F_4 \otimes I_{4^{k-i-1}}) \right. \\ &\qquad \left. (I_{4^i} \otimes T_{4^{k-i}})\right] P_{4^k,4^{k-1}} \\ T_J &= (D_{J/4}^{(0)} \oplus D_{J/4}^{(1)} \oplus D_{J/4}^{(2)} \oplus D_{J/4}^{(3)}) \\ D_{J/4}^{(n)} &= diag(W_J^{nk}), \quad 0 \le k < J/4 \\ F_4 &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \end{aligned} \tag{4}$$

where $P_{N,R}$ is a stride-by-R permutation matrix of order N [3]. The arithmetic kernel of radix-4 DIT FFT is the butterfly operation defined as

$$\begin{aligned} X_0 &= P_0 + W_1P_1 + W_2P_2 + W_3P_3 \\ X_1 &= P_0 - jW_1P_1 - W_2P_2 + jW_3P_3 \\ X_2 &= P_0 - W_1P_1 + W_2P_2 - W_3P_3 \\ X_3 &= P_0 + jW_1P_1 - W_2P_2 - jW_3P_3 \end{aligned} \tag{5}$$

The signal flow graph of radix-4 DIT butterfly operation is illustrated in Figure 2. The coefficients W_J^k, W_J^{2k}, and W_J^{3k} are all of the same form as W_J^k in (2).

Radix-4 algorithms have a computational advantage over radix-2 algorithms because one radix-4 butterfly does the work of four radix-2 butterflies, and the radix-4 butterfly requires only three complex multipliers compared to four complex multipliers of four radix-2 butterflies. Radix-2 and radix-4 FFTs are the most commonly used algorithms, it is also possible to design FFTs with even higher radix butterflies. The reason they are not often used is because the control and dataflow of their butterflies are so complicated that the additional efficiency gained is lost.

2. PARTIAL-COLUMN RADIX-2 AND RADIX-2/4 FFTS

Parallel FFT processors can be divided into the following principal classes [4]: fully parallel, pipelined, column, and partial-column. In general, the pipelined FFTs have been popular due to their principal simplicity [5].The advantage in using partial-column organization is that partial-column processing is scaleable where as in pipeline and column FFTs the number of butterfly units is dependent on the FFT size. The partial-column organization can also be combined to pipeline class, which results in parallel pipelines as proposed, e.g., in [4, 6]. The high-level organization of the FFT processor is not the only characteristic, that defines the key properties of the implementation. Most of the power in FFT processor is consumed by the butterfly operations [7]. Therefore, the power optimizations should be performed for butterfly units. butterfly units can be realized with several different prinziples. butterfly units based on bit-parallel multipliers [7, 8], CORDIC [9], and DA [10, 11] have been reported. The organization of the proposed energy-efficient partial-column FFT processor is described in the following sections.

2.1. Organization

In general, in the partial-column processing all operands to the butterfly computations are transferred simultaneously from the memory implying need for high memory bandwith. In our approach, the butterfly units are pipelined in a sense that a single operand ($2K$-bit word if real and imaginary parts take K bits each) is transferred to the butterfly unit at a time, thus each butterfly unit has a dedicated bus to and from memory as illustrated in Figure 3. Such an arrangement increases the computation time but this can be compensated by increasing the number of butterfly units.

Our approach is to minimize the RAM storage, thus the computation is performed in-place, i.e., results of butterfly units are stored into the same memory locations they were obtained. Therefore, N complex-valued memory locations are needed for an N-point FFT. The organization requires that there are $2M$-ports in the RAM memory

# of Multipliers	Area [gates]		Power [mW]		Min. Critical Path [ns]
	50 MHz	100 MHz	50 MHz	100 MHz	
4, Eqn. (6)	6986	9483	2,96	4,22	8
3, Eqn. (7)	5616	8149	2,94	4,34	9
2, Eqn. (8)	3915	5181	1,44	2,13	8

Table 1. Characteristics of 16-bit complex multipliers on an 0,11μm ASIC technology.

when M butterfly units with throughput of one is used. This can be arranged with multi-port memories, but more area-efficient approach is to use interleaved memories with a conflict-free access scheme [12]. This arrangement requires that for an N-point FFT, there are $2M$ single- port memories with $N/2M$ words and the memories are interconnected with a permutation network.

2.2. Radix-2 Butterfly Units

An essential portion of the cost and characteristics of butterfly unit is defined by the realization of complex multiplier. In this paper, we have considered three different methods based on bit-parallel multipliers, CORDIC, and distributed arithmetic. All these result in different structures for computing the radix-2 butterfly operation and these are discussed in the following sections.

2.2.1. Bit-Parallel Multiplier

The multiplication of two complex numbers W and P is shown below:

$$\begin{aligned} WP = (W_R + jW_I)(P_R + jP_I) = W_R P_R \\ - W_I P_I + j(W_R P_I + W_I P_R) \end{aligned} \tag{6}$$

Direct structure that implements the equation (6) requires four real multipliers and two adders. The critical path in this structure is through a multiplier and adder. The number of multipliers can be reduced by rewriting (6) into a different form and there are several ways to perform such a reduction as described in [13]. In this paper, we have considered the following version:

$$\begin{aligned} WP = W_I(P_R - P_I) + P_R(W_R - W_I) \\ + j[W_I(P_R - P_I) + P_I(W_R + W_I)] \end{aligned} \tag{7}$$

This equation implies that only three real multipliers but five adders are needed for complex multiplication. This suggests that more efficient implementation can be obtained. However, the drawback is longer critical path, i.e., path through multiplier and two adders, which requires the logic synthesis tool to use faster components to fulfill the timing requirements compared to the previous case. This is illustrated in Table 1. In delay and power consumption point of view four multiplier case is advantageous, thus we have used this type of complex multiplier in our butterfly units. The Table also shows that noteworthy savings in power consumption can be obtained if two clock cycles can be allocated for the complex multiplication, i.e., the critical path through the complex multiplier is longer than the clock period. This is possible in our case when

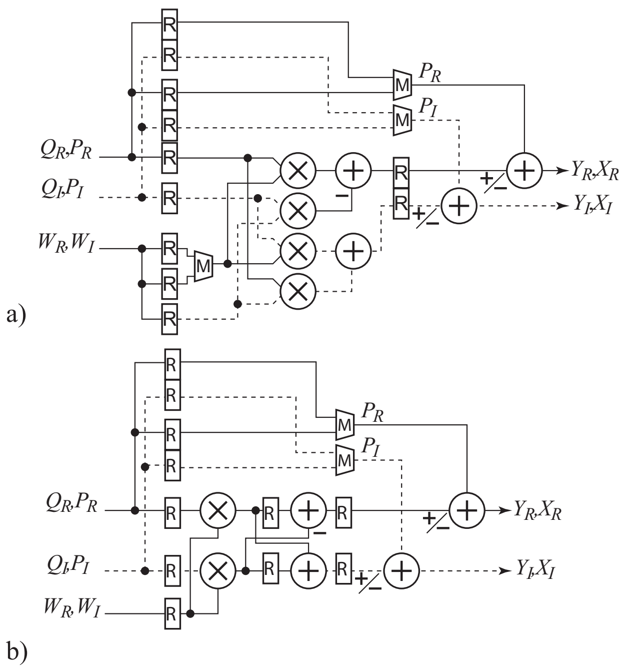


Figure 4. Block diagram of radix-2 butterfly units based on (a) four and (b) two real multipliers. Dashed lines indicate imaginary part data.

operands enter through a common bus and operand Q in (3) enters the butterfly unit before the operand P. The two clock cycles can also be exploited in pipelined fashion according to the following formulation:

$$\begin{aligned} &A_1 = P_R W_R; \quad B_1 = P_I W_R \quad (cycle\ \#1) \\ &A_2 = P_R W_I; \quad B_2 = P_I W_I \quad (cycle\ \#2) \\ &WP = (A_1 - B_2) + j(B_1 + A_2) \end{aligned} \tag{8}$$

This approach requires only two real multipliers and two adders. The previous equation also indicates that the real and imaginary parts of the twiddle factor are not needed at the same time, thus it is sufficient to provide them at consecutive clock cycles. This approach provides power efficiency as shown in Table 1 but, on the other hand, the critical path limits the applicable clock frequencies as shown in the forthcoming sections.

Based on the previous discussion, we propose two structures for the butterfly computation in partial-column FFT processors. The complex multiplier structures are based on either four or two real-valued multipliers as illustrated in Figure 4. The outputs of the real-valued multipliers in Figure 4 are registered. The advantage gained is that the critical path of the complex multiplier, i.e., the critical path of the butterfly unit is shortened. The complete butterfly computation can be performed by pairing the complex multiplier with two adders and registers for storing the operands. Two sets of registers are needed for operand P since the next P is arriving when the previous WQ is available. In addition, it should be noted that in both structures real and imaginary parts of the twiddle factors can be fed on consecutive clock cycles.

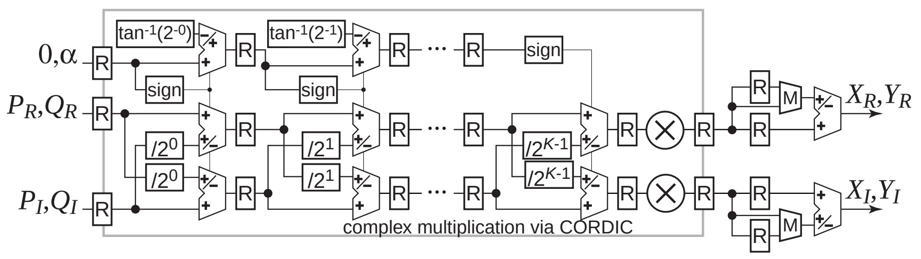


Figure 5. Block diagram of butterfly unit based on iterative CORDIC.

2.2.2. CORDIC

Since a complex number can be interpreted as a vector in imaginary plane and, in FFT, the twiddle factors are roots of unity, i.e., norm of a twiddle factor vector is one, the multiplication by twiddle factors can be realized by rotating the vector. The CORDIC algorithm is a well-known iterative method to perform such a rotation. The basic equations to be iterated are [11]

$$\begin{aligned} x_{Rk+1} &= x_{Rk} + sign(z_k)x_{Ik}2^{-k} \\ x_{Ik+1} &= x_{Ik} + sign(z_k)x_{Rk}2^{-k} \\ z_{k+1} &= z_k - sign(z_k)tan^{-1}(2^{-k}) \end{aligned} \tag{9}$$

Where k denotes the iteration, x_{Rk} and x_{Ik} are the coordinates of the vector at iteration k, and x_k is the angle at iteration k. Multiplication QW_N^n is started with initial conditions $z_0 = \frac{-2\pi n}{N}$, i.e., rotation angle of twiddle factor W_N^n, $x_{R0} = Q_R$, and $x_{I0} = Q_I$. The computation requires values of tan^{-1} in (9) to be stored into a ROM but the storage requirement is small; typically for a K-bit precision, approximately K iterations are needed implying need to store K values of tan^{-1}. The drawback of the algorithm is that the magnitude of vector increases at each iteration. This scaling needs to be compensated but fortunately the scaling is constant. Therefore, in our experiments, we have used a multiplier with constant multiplicand to compensate the magnitude.

Since the CORDIC algorithm is iterative, the butterfly unit based on CORDIC may obtain new operands after K cycles when K-bits are used for real and imaginary part implying throughput of $1/K$. In order to increase the throughput to 1, the iterations of CORDIC algorithm can be separated into independent stages implying a pipelined realization. For a K-bit precision, K pipeline stages are needed, which results in an increase in area due to the pipeline registers. The block diagram of a butterfly unit based on the pipelined CORDIC is illustrated in Figure 5. This reminds the butterfly units used in pipelined FFT processors. It should be noted that, in this approach, all the operands are fed through the complex multiplication, thus every second rotation angle needs to be zero, i.e., operand P in (3) is not rotated.

2.2.3. Distributed Arithmetic

The complex multiplication QW_N^n can be performed with the aid of distributed arithmetic as described in [14]. The principle can be described as follows. Let x be a K-bit number in two's complement representation consisting of bits $(x_{(K-1)},...,x_{(1)},x_{(0)})$ where $x_{(0)}$ is the least significant bit. Then x can be expressed as

$$\begin{aligned} x &= \frac{1}{2}x - \frac{1}{2}(-x) \\ &= \frac{1}{2}\Big[-(x_{(K-1)} - \overline{x_{(K-1)}}) \\ &\quad + \sum_{k=0}^{K-2}(2^{k+1-K}(x_{(k)} - \overline{x_{(k)}})) - 2^{1-K}\Big] \end{aligned} \tag{10}$$

where $\overline{x_k}$ is complement of bit x_k. This equation can be used in (x), which results in the following:

$$\begin{aligned} 2WQ = &-q_r(Q_{R(K-1)}, Q_{I(K-1)}) \\ &+ \sum_{k=0}^{K-2}(q_R(Q_{R(k)}, Q_{I(k)})2^{k+1-K}) \\ &+ q_R(0,0)2^{1-K} + j\Big[-q_I(Q_{R(K-1)}, Q_{I(K-1)}) \\ &+ \sum_{k=0}^{K-2}(q_I(Q_{R(k)}, Q_{I(k)})2^{k+1-K}) \\ &+ q_I(0,0)2^{1-K}\Big] \end{aligned} \tag{11}$$

where functions $q_R(\cdot,\cdot)$ and $q_I(\cdot,\cdot)$ are defined as

$$\begin{aligned} q_R(a_R, a_I) &= (a_R - \overline{a_R})W_R - (a_I - \overline{a_I})W_I \\ q_I(a_R, a_I) &= (a_R - \overline{a_R})W_I + (a_I - \overline{a_I})W_R \end{aligned} \tag{12}$$

All the possible values of $q_R(\cdot,\cdot)$ and $q_I(\cdot,\cdot)$ are listed in Table 2, which shows that for each twiddle factor W it is enough to store two real values $-(W_R - W_I)$ and $-(W_R + W_I)$. Then complex multiplication can be performed in digit-serial fashion by accumulating these store values based on the bits in the real and imaginary parts.

The original butterfly structure proposed in [14] was designed for bit-serial interface, which implies needs for 1-bit memories. Since such memories are rare in modern ASIC technologies we included parallel- to-serial converters into the input and the principal structure to perform complex multiplication with the aid of distributed arithmetic is illustrated in Figure 6 (a). It should be noted that the throughput of the multiplication is $1/K$ when K-bit operands are used. As the result of multiplication can be obtained in bit-parallel format, we have used bit-parallel

a_R	a_I	$q_R(a_R,a_I)$	$q_I(a_R,a_I)$
0	0	$-(W_R-W_I)$	$-(W_R+W_I)$
0	1	$-(W_R+W_I)$	$+(W_R-W_I)$
1	0	$+(W_R+W_I)$	$-(W_R-W_I)$
1	1	$+(W_R-W_I)$	$+(W_R+W_I)$

Table 2. All the possible values of $q_R(a_R,a_I)$ and $q_I(a_R,a_I)$ in (11) and (12).

complex add/sub units to perform the add and subtract operations in the butterfly. The block diagram of the resulting butterfly unit is ilustrated in Figure 6 (b).

The previous butterfly structure is not area-efficient since the add/sub units are used only once within K cycles. Therefore, we proposed a new structure, where several complex multipliers share the same add/sub units resulting better energy and area-efficiency. This principal structure is illustrated in Figure 6 (c) Since the throughput of the complex multiplier in Figure 6 (a) is $1/K$ when K-bit words are used, $K/2$ multipliers can share the same add/sub unit resulting in througput of 1. This, however, requires that the operand P is fed to the system $k+1$ cycles later than the operand Q.

2.3. Radix-2/4 Butterfly Unit

Radix-4 butterfly operation was described in (5). Radix-4 butterfly unit can be constructed from the radix-2 butterfly unit shown in Figure 4 (a). In order to support the radix-4 operation two add/sub units, four multiplexers, six registers, and multiplication by imaginary unit j are added to the structure. Additional add/sub units are needed, because the number of additions and subtractions is larger in (5) than in (3). Additional registers and multiplexers are needed to implement the permutations in radix-4 butterfly operation. Multiplication by imaginary unit j is accomplished by a swapping of the real and imaginary parts, and negating the imaginary part. This negation of imaginary part can be easily done by inverting the add/sub control signal of the second add/sub unit. Because this new butterfly unit is an extended version of the radix-2 butterfly unit, it can be also used for the computation of radix-2 butterfly operation. Thus, this new butterfly unit is a mixed-radix structure. Depending on the values of control signals radix-2/4 butterfly unit computes either radix-2 or radix-4 butterfly operations in a serial fashion, in which a new value is fed into and a result is obtained at the output at every cycle. Block diagram and operation of this radix-2/4 butterfly unit is illustrated in Figure 7. Figure 7 also illustrates the parst of the data path that are active in radix-2 and radix-4 operation modes. Complex multiplier in this structure is implemented with four real multipliers. Complex multiplier based on two real-valued multipliers can not be used, because in radix-4 butterfly operation complex multiplication needs to be performed on consecutive input operands, as can be seen from Figure 2.

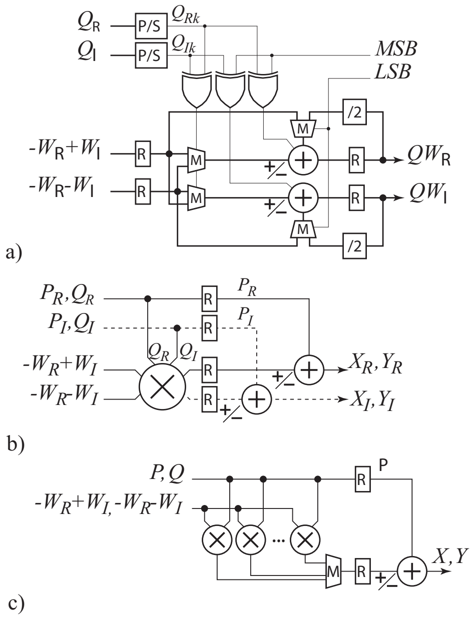


Figure 6. Block diagram of structures based on distributed arithmetic: (a) complex multiplication by twiddle factors, (b) butterfly computation with bit-parallel add-sub units, and (c) butterfly computation with shared add/sub unit. In (a) and (b), all the signals are real valued and, in (c), complex-valued. P/S: Parallel-to-serial converter.

2.4. Operand and Twiddle Factor Storage

The in-place property of the FFT algorithms in (1) and (4) allows the input operands and intermediate and final results, to share the same memory locations. This, however, requires that the input sequence is stored into the RAM memory according to the input permutations defined in (1) and (4). The in-place operand access during the actual computations can be realized with the aid of address rotation [15].

In an N-point radix-2 FFT, there are N/2 different complex-valued twiddle factors. However, these factors can be computed from N/8+1 complex-valued numbers with the aid of an add/sub unit [16]. On the other hand, there are N/4+1 different real-valued magnitudes present in the real and imaginary parts, thus the twiddle factors can be formed easily by fetching the magnitudes from a table and taking complement when needed. However, this arrangement requires that two memory accesses to the ROM table are needed. This is possible with the proposed pipelined butterfly units since the operand access takes two cycles.

The previous method can be applied directly to butterfly units based on bit-parallel multipliers. The DA based butterfly units require sum and difference of the real and imaginary parts of the twiddle factors, thus an additional add/sub unit is needed. Butterfly units based on CORDIC need no ROMs since the angles can be generated easily from counters synchronizing the overall operation.

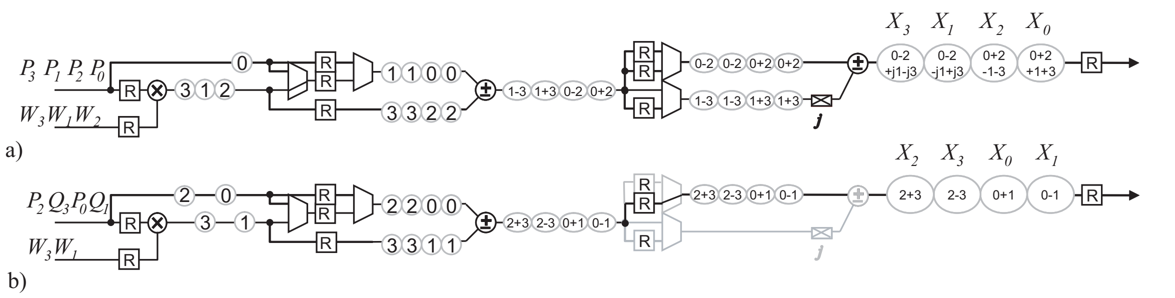


Figure 7. Block diagram and operation of radix-2/4 butterfly unit. All the signals are complex-valued.

3. COMPARISON

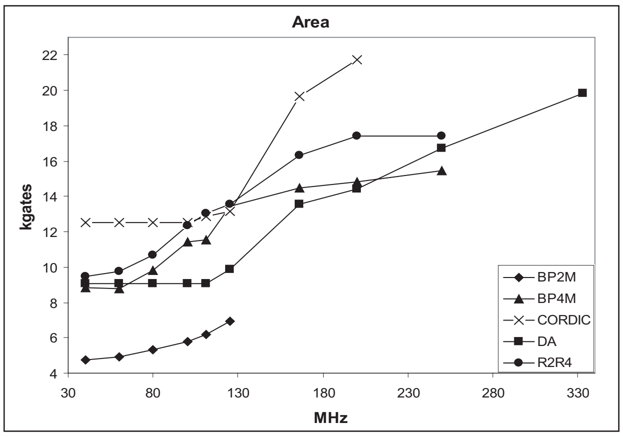
Comparison of the energy-efficiency of the different butterfly structures is not a straigthforward task since the different butterfly structures have different properties and even the same structure may have different circuit level implementation depending on the targeted clock frequency. In order to perform a fair comparison, the performance of each structure has to be the same, i.e., word width is the same and all the units have to produce the same number of results per second. Since the behavior can be obtained by either increasing parallelism or clock frequency, we have compared the structures such that the number of complex-valued results per clock cycle is one and the clock frequency is varied. The notations used for butterfly structures, described in this paper, are shown below:

- BP4M : radix-2 butterfly unit based on four real-valued multipliers, shown in Figure 4 (a)
- BP2M : radix-2 butterfly unit based on two real-valued multipliers, shown in Figure 4 (b)
- CORDIC : radix-2 butterfly unit based on pipelined CORDIC, shown in Figure 5
- DA : radix-2 butterfly unit based on distributed arithmetic with shared add/sub unit, shown in Figure 6 (c)
- R2R4 : radix-2/4 butterfly unit based on four real-valued multipliers, shown in Figure 7

These butterfly units were described with VHDL language, using 16-bit precision for arithmetic units such that all the units have registered outputs. VHDL descriptions were synthesized on a 0,11 μm standard cell ASIC technology with logic synthesis tools from Synopsys. Several clock constraints ranging from 40 to 333 MHz were used. Gate level estimations for the area and power consumption of the butterfly units are shown in Figure 8. Figure 8 (b) contains two estimations for the power consumption of radix-2/4 butterfly unit. $R2R4\_R2$ is the power consumption in radix-2 and $R2R4\_R4$ in radix-4 mode. Power consumption is larger for radix-4 than radix-2 mode due to the fact that the active datapath is longer in radix-4 mode. Estimations for the power consumption are obtained by using the switching activity information captured from the simulations of the VHDL models. Simulations were done with the ModelSim simulator from Mentor Graphics.

Gate level estimates in Figure 8 shows that butterfly units, where the complex multiplication is implemented with bit-parallel multipliers, are advantageous in power consumption point of view. Butterfly unit, where the complex multiplication is performed with the aid of distributed arithmetic, outperforms these butterfly units only in a narrow frequency range. In power consumption point of view, the best solution is the $BP2M$ butterfly unit, but the applicable clock frequencies for this unit are limited to lower frequencies than in other implementations. The energy-efficiency of the radix-2/4 butterfly unit is not evident in Figure 8. In Figure 9, the normalized energy required for the calculation of the FFT is illustrated as a function of the length of the FFT. The energy has been calculated by multiplying the power consumption values in Figure 8 by the amount of clock cycles required to perform the FFT. The number of clock cycles required is smallest for radix-2/4 butterfly unit, when it is operating in radix-4 mode. Therefore, when the target clock frequency is too large for the $BP2M$ butterfly unit, the radix-2/4 butterfly unit is the most energy-efficient alternative of the butterfly units, described in this paper. To be able to utilize the radix-2/4 butterfly unit in radix-4 mode, the length of the FFT has to be a power of 4. If the length of the FFT $N = 2^k$, FFT can be calculated mostly in radix-4 mode, since $2^k = 2^{2n} = (2^2)^n = 4^n$ when k is even and $2^k = 2^{2n+1} = 2^{2n} \cdot 2 = 4^n \cdot 2$ when k is odd.

The previous comparison does not take into account other logic needed in the FFT processor: control logic and memories. The control unit, when realized as a finite state machine, has almost the same complexity regardless of the type of the butterfly unit. In addition, according to our experiments, the cost of control unit is negligible in terms of area and power consumption, thus it has been left out from the analysis. The FFT processor will have the same memory for operand storage independent on the type of butterfly unit, thus it has been left out. The only difference is the ROM memory for twiddle factor storage. Radix-2 and radix-2/4 buttefly units based on bit-parallel multipliers and butterfly units based on distributed arithmetic need an 16-bit N/4+1 word ROM for twiddle factor storage while this is not needed in CORDIC based butterfly units. However, the power consumption of ROM memories is very

a) 

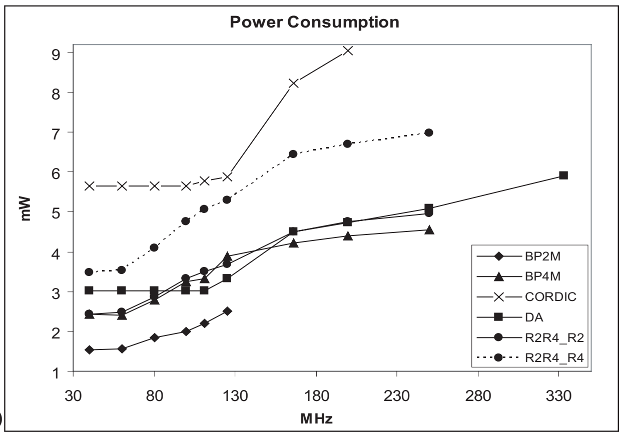
b) 

Figure 8. Gate level comparison of the area (a) and power consumption (b) of the selected butterfly structures as a function of clock frequency.

small compared to arithmetic units, e.g., according to our experiments, the CORDIC approach will provide better energy-efficiency on 100 MHz clock frequency than the other approaches when the transform length is $N > 2^{25}$.

4. CONCLUSION

In this paper, we have proposed a general organization for partial-column radix-2 and radix-2/4 FFT processors and described methods to improve the energy-efficiency of pipelined butterfly units. Several pipelined butterfly units have been synthesized onto a 0,11 μm ASIC technology and the results show that butterfly units based on bit-parallel multipliers are energy-efficient but cannot be used when high clock frequencies are used. The energy-efficiency of these butterfly units can be further enchanced by utilizing radix-4 operation. The proposed radix-2/4 butterfly unit was shown to be the most energy-efficient choise, when the target clock frequency is high. Butterflies based on distributed arithmetic can support higher clock frequencies than the butterflies based on bit-parallel multipliers. When extremely long FFTs are needed, the pipelined CORDIC should be considered due to the fact that separate twiddle factor ROMs are not needed.

5. REFERENCES

[1] J. Cooley and J. Tuckey, “An algorithm for the machine calculation of the complex fourier series,” *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.

[2] Tran-Thong and B. Liu, “Fixed-point fast fourier transform error analysis,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 24, no. 6, pp. 563–573, Dec. 1976.

[3] J. Granata, M. Conner, and R. Tolimieri, “Recursive fast algoritms and the role of the tensor product,” *IEEE Trans. Signal Processing*, vol. 40, no. 12, pp. 2921–2930, Dec. 1992.

[4] S. F. Gorman and J. M. Wills, “Partial column fft pipelines,” *IEEE Trans. IEEE Trans. Circuits Syst. II*, vol. 42, no. 6, pp. 414–423, June 1995.

[5] S. He and M. Torkelson, “Design and implementation of a 1024-point pipeline fft processor,” in *Proc. IEEE Custom Integrated Circuits Conf.*, Santa Clara, CA, May 11-14 1998, vol. 2, pp. 131–134.

[6] E. H. Wold and A. M. Despain, “Pipeline and parallel-pipeline fft processors for vlsi implementations,” *IEEE Trans. Comput.*, vol. 33, no. 5, pp. 414–426, May 1984.

[7] M. Hasan and T. Arslan, “Implementation of low-power fft processor cores using a novel order-base processing scheme,” in *Proc. IEEE Circuits Devices Syst.*, June 2003, vol. 150, pp. 149–154.

[8] M. Wosnitza, M. Cavadini, M. Thaler, and G. Tröster, “A high precision 1024-point fft processor for 2d convolution,” in *Dig. Tech Papers IEEE Solid-State Circuits Conf.*, San Francisco, CA, Feb. 5-7 1998, pp. 118–119.

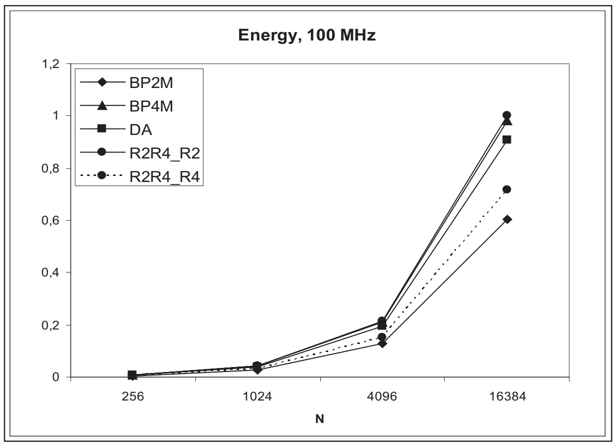
[9] A. M. Despain, “Fourier transform computers using cordic iterations,” *IEEE Trans. Comput.*, vol. 23, no. 10, pp. 993–1001, Oct 1974.

[10] A. Berkeman, V. Owall, and M. Torkelson, “A low logic depth complex multiplier using distributed arithmetic,” *IEEE Solid-State Circuits*, vol. 35, no. 4, pp. 656–659, Apr. 2000.

[11] L. Wanhammar, *DSP Integrated Circuits*, CA: Academic Press, San Diego, 1999.

[12] J. Takala and T. Järvinen, “Stride permutation access in interleaved memory systems,” in *Domain-Specific Processors: Systems, Architectures, Modeling and Simulation*, New York, NY, 2004, pp. 63–84.

[13] A. Wenzler and E. Lüder, “New structures for complex multipliers and their noise analysis,” in *Proc.*

a) 

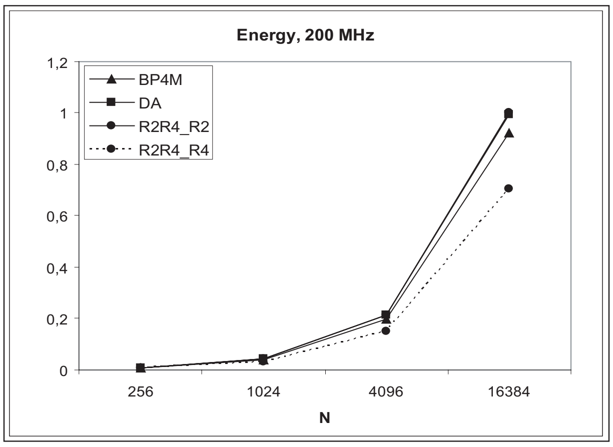
b) 

Figure 9. Normalized energy for the selected butterfly units as a function of the length of the FFT N.

IEEE ISCAS, Seatle, WA, Apr. 30- May 3 1995, vol. 2, pp. 1432–1435.

[14] S.A. White, "A simple fft butterfly arithmetic unit," *IEEE Trans. Circuits Syst.*, vol. 28, no. 4, pp. 352–355, Apr. 1981.

[15] E. Chu and A. George, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*, FL: CRC Press, Boca Raton, 2000.

[16] M. Hasan and T. Arslan, "Fft coefficient memory reduction technique for ofdm applications," in *Proc. IEEE ICASSP*, Orlando, FL, May 13-17 2002, pp. 1085–1088.