Low-Power Twiddle Factor Unit for FFT Computation

Teemu Pitkänen, Tero Partanen, and Jarmo Takala

Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland
{teemu.pitkanen, tero.partanen, jarmo.takala}@tut.fi

Abstract. An integral part of FFT computation are the twiddle factors, which, in software implementations, are typically stored into RAM memory implying large memory footprint and power consumption. In this paper, we propose a novel twiddle factor generator based on reduced ROM tables. The unit supports both radix-4 and mixed-radix-4/2 FFT algorithms and several transform lengths. The unit operates at a rate of one factor per clock cycle.

1 Introduction

Fast Fourier transform (FFT) has gained popularity lately due to the fact that OFDM has been used in several wireless and wireline communication systems, e.g., IEEE 802.11a/g, 802.16, VDSL, and DVB. An integral part of the FFT computation are the twiddle factors, which, in software implementations, are typically stored into RAM memory implying large memory footprint. The twiddle factors can be generated at run-time. A traditional method is to use CORDIC as described, e.g., in [1]. The sine and cosine values are needed in direct digital frequency synthesizers and often the generation is based on polynomials, e.g., in [2]. An other approach is to use a function generator based on recursive feedback difference equation [3,4]. Typically these approaches result in smaller area than memory based approaches. However, since the computation is done at run-time, there is a huge amount of transistor switching implying higher power consumption in CMOS implementations.

Another approach is to store the twiddle factors into a ROM table. In an N-point FFT, there are $N/2$ different twiddle factors and an approach exploiting this property has been reported in [5]. Methods requiring only $N/4$ coefficients to be stored into a table are described in [6,7]. There is, however, even more redundancy since the real and imaginary parts of the factors are sine values and $N/8+1$ complex coefficients are needed to reconstruct all the factors for an N-point FFT [8]. In [9], a coefficient manipulation method is presented where only $N/8+1$ coefficients are needed to generate the twiddle factors. However, the previous methods are designed only for radix-2 algorithms containing more arithmetic operations than radix-4 algorithms.

A twiddle factor generator unit could be used as a special function unit in an application-specific instruction-set processor (ASIP) but it may not increase the performance of the software implementation. Often several instructions are needed to compute the correct index to the unit. Considerable performance increase can be expected, if the unit can also perform the index modifications to avoid additional instructions. However, the indexing of the twiddle factors varies depending on the FFT variant. More detailed discussion on twiddle factor indexing can be found from [10].

In this paper, we propose a low-power twiddle factor unit based on a ROM table. The proposed work differs from the related work such that the proposed unit a) supports radix-4 and mixed-radix-4/2 FFT algorithms, b) supports several transform sizes (power-of-two), and c) integrates index manipulation. By supporting radix-4 algorithms, the performance of FFT computation is increased significantly compared to radix-2 algorithms. In addition, the overhead of address manipulation in software implementation is omitted, which increases the performance even more. The unit can generate factors at a rate of one factor per clock cycle. The proposed unit has already been used in the FFT implementations described in our previous work [11,12] but here the twiddle factor generation is described in detail.

2 FFT Algorithms

In this work, we have used the traditional in-place radix-4 decimation-in-time (DIT) radix FFT algorithm with in-order-input, permuted output as given, e.g., in [13]. In this work, we would like to expose the different permutations, thus we formulate the traditional algorithm in the following fashion:

$$F_{2^{2n}} = R_{2^{2n}} \left[\prod_{s=n-1}^{0} [O^s_{2^{2n}}]^T (I_{2^{(2n-2)}} \otimes F_4) A^s_{2^{2n}} O^s_{2^{2n}} \right] ; \tag{1}$$

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} ; \tag{2}$$

$$R_{2^{2n}} = \prod_{k=2}^{n} I_{2^{(2n-2k)}} \otimes P_{2^{2k},4} ; \tag{3}$$

$$O^s_{2^m} = I_{4^s} \otimes P_{2^{(m-2s)},2^{(m-2s-2)}} \tag{4}$$

where j is the imaginary unit, I_n is an identity matrix of order n, and the permutation matrices R_N and O_N are based on stride-by-S permutation matrices [14] $P_{N,S}$ defined as

$$[P_{N,S}]_{mn} = \begin{cases} 1, \text{ iff } n = (mS \bmod N) + \lfloor mS/N \rfloor \\ 0, \text{ otherwise} \end{cases} , \; m,n = 0,1,\ldots,N-1 \tag{5}$$

The matrix A^s_N contains N twiddle factors $W^k_K = e^{j2\pi k/K}$ as follows

$$A^s_N = Q^s_N \left[\bigoplus_{b=0}^{N/4-1} \operatorname{diag}\left(W^0_{4^{s+1}}, W^{\lfloor \frac{b4^{s+1}}{N} \rfloor}_{4^{s+1}}, W^{2\lfloor \frac{b4^{s+1}}{N} \rfloor}_{4^{s+1}}, W^{3\lfloor \frac{b4^{s+1}}{N} \rfloor}_{4^{s+1}} \right) \right] ; \tag{6}$$

$$Q^s_N = \prod_{l=0}^{s} P_{4^{(s-l)},4} \otimes I_{N/4^{(s-l)}} . \tag{7}$$

Examples of signal flow graphs of this algorithm are depicted in Fig. 1a) and 1c).

As the Fig. 1 shows the output data is not in order, thus to give it in order, input permutation is needed at each column and it complicates the index modifications in the coefficient generator.

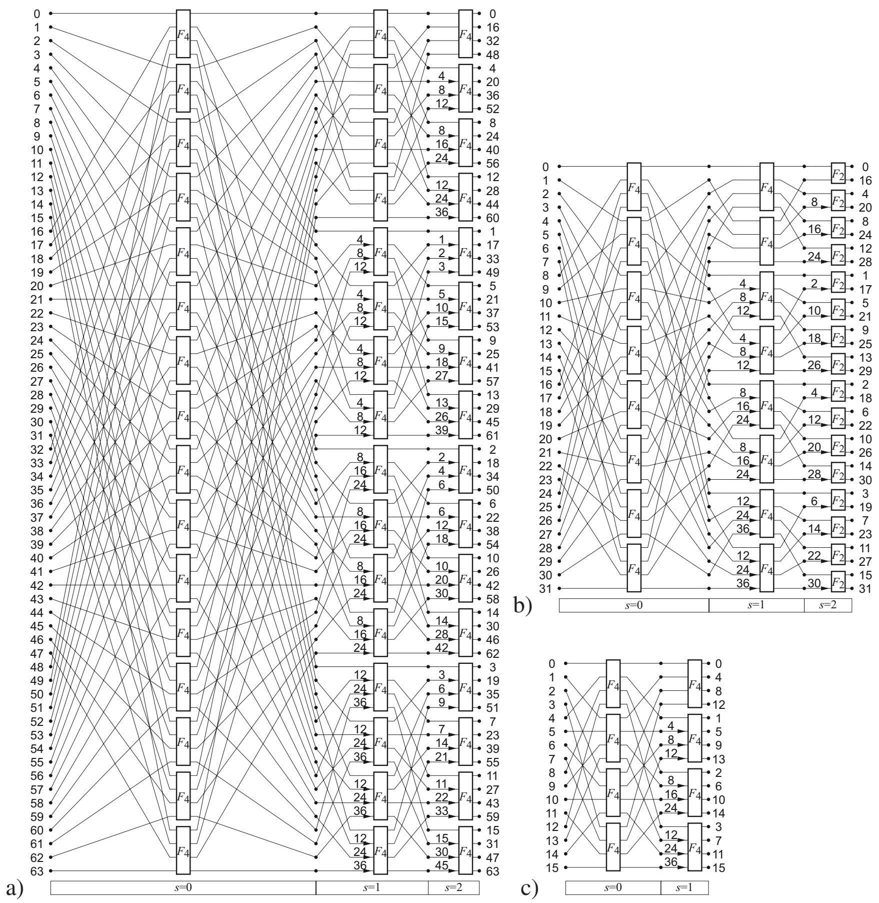

Fig. 1. Signal flow graph of a) 64-point radix-4, b) 32-point mixed-radix, and c) 16-point radix-4 FFT. A constant k in the signal flow graph represents a twiddle factor W_{64}^{k}.

The radix-4 algorithms can be used only when the FFT size is a power-of-four. Power-of-two transforms can be supported by using mixed-radix approach and a mixed-radix-4/2 FFT consists of radix-4 processing columns followed by a single radix-2 column as follows

$$F_{2^{2n+1}} = S_{2(2n+1)}\left(I_{4^n} \otimes F_2\right) B_{2(2n+1)} \cdot \left[\prod_{s=n-1}^{0} [O^s_{2(2n+1)}]^T \left(I_{2(2n-1)} \otimes F_4\right) A^s_{2(2n+1)} O^s_{2(2n+1)}\right] \; ; \; F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad (8)$$

where the matrices O^s_N and A^s_N are defined in (4) and (6), respectively. The matrix S_N is a permutation matrix given as

$$S_N = \left(I_2 \otimes R_{4^n}\right) P_{N,2} \; , N = 2^{2n+1}. \qquad (9)$$

The matrix B_N contains the twiddle factors for the radix-2 processing column and it is defined as

$$B_N = Q_N^{\log_4(N/2)} \bigoplus_{b=0}^{N/2-1} \text{diag}\left(W_N^0, W_N^b\right) \ , N = 2^{2n+1} \tag{10}$$

where the permutation matrix Q_N^s is defined in (7). Example of signal flow graph of the mixed-radix-4/2 algorithm is shown in Fig. 1b).

3 Twiddle Factor Access

Our objective is to design a unit, which can generate twiddle factors for several power-of-two size transforms. By investigating the structure of the twiddle factors in FFTs of different size, we find that the twiddle factors of a shorter transform are included in the larger transform. Our approach is based on lookup tables (LUT) containing the twiddle factors, thus we need to define the maximum FFT size supported, $N_{max} = 2^{n_{max}}$, and the twiddle factors for shorter transforms can be generated from the same LUT.

The unit generates a twiddle factor based on index from an iteration counter, which may be updated by software, if the unit is used as a special function unit in a processor. When targeting to an application-specific fixed-function FFT processor, the iteration counter is the counter, which used to generate all the control signals in the architecture.

An N-point radix-4 FFT contains $\log_4(N)$ iterations of butterfly columns divided into $N/4$ four-input radix-4 butterfly computations while, in a mixed-radix-4/2 algorithm, $\log_4(N/2)$ iterations of $N/4$ radix-4 computations is followed by $N/2$ two-input radix-2 computations. Therefore, we need $\log_2(N)$ bits to identify each butterfly input in a butterfly column and $\lceil\log_2(\log_4(N))\rceil$ bits to express the butterfly column, i.e., s in definitions (1) and (8).

The input operands for the unit are the iteration counter and parameter indicating the transform size. Let us denote the $(\lceil\log_2(\log_4(N))\rceil + \log_2(N))$-bit iteration counter by $c = (c_{\lceil\log_2(\log_4(N))\rceil+\log_2(N)-1}, \ldots, c_1, c_0)^T$. The transform size is indicated by parameter $f = \log_2(N_{max}) - \log_2(N)$. The structure of the proposed function unit is discussed in the following sections with an example design supporting FFT sizes of 16, 32, and 64. In this example case, the input parameter f can have values 0, 1, or 2 to indicate FFT sizes 64, 32, or 16, respectively. A 5-bit iteration counter c is used when FFT size is 16 and, for a 64-point FFT, an 8-bit counter is needed. The block diagram of the example design is illustrated in Fig. 2. The input parameters are written into registers f and c and the final twiddle factor is obtained from the output registers.

3.1 Scaling

In order to minimize the bit-level shifts due to different transform sizes, we first shift the iteration counter c to the left by the number of bits indicated by the parameter f. This implies that after the shift we obtain a bit-field where the $\lceil\log_2(\log_4(N_{max}))\rceil + \log_2(N_{max})$ bits indicate the butterfly column s and the $\log_2(N_{max}) = n_{max}$ least significant bits contain the index of the twiddle factor in the column to be generated. Let us denote this part by $d = (d_{n_{max}-1}, \ldots, d_0)^T$. However, the actual index is in the most significant bits index and d contains $(n_{max} - \log_2(N))$ zeros in the least significant bits. The rest of the operation is based on operands s and d.

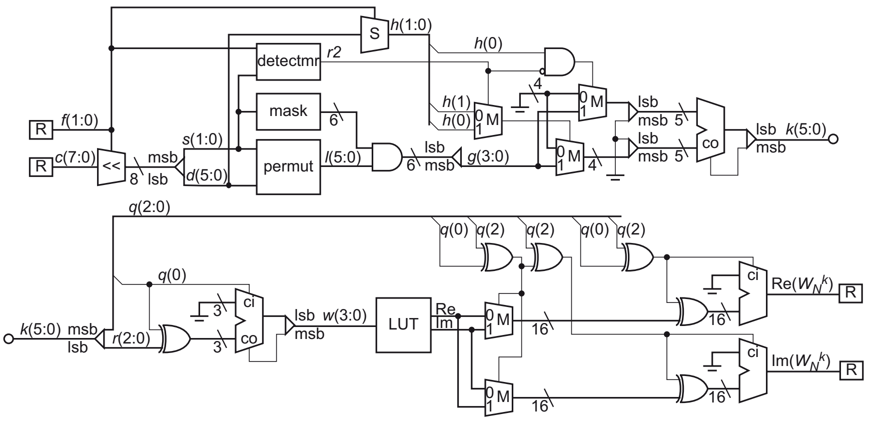

Fig. 2. Block diagram of twiddle factor generator supporting transform sizes of 16, 32, and 64. R: register. M: multiplexor. co: carry out. ci: carry in. <<: left shifter.

3.2 Permutation

The order of twiddle factors depends on the FFT algorithm and, in this work, we concentrate on the in-order input, permuted output FFTs given in (1) and (8). In these particular cases, we need to consider the implementation of matrices A_N^s and B_N^s defined in (6) and (10), respectively.

Our approach is based on index modifications, thus first we need to perform the permutation Q_N^s in (7). The the permutation can be investigated by considering the bit-level rotations as discussed in [15,16]. This shows that the permutation is actually the traditional bit-reversed permutation but here 2-bit fields are used instead of a single bit. It should also be noted that the permutation varies according to the butterfly column s. The permutation in our case is actually independent on the transform size, since we have shifter the index earlier, thus the permuted index, $l = (l_{\log_2(N)}, \ldots, l_1, l_0)$, of an N-point FFT can be expressed in bit-level in matrix form as follows

$$l = \begin{pmatrix} \bar{I}_s \otimes I_2 & \\ & I_{n_{max}-2s} \end{pmatrix} d \ ; \ \bar{I}_s = \begin{pmatrix} & & & 1 \\ & & \cdot & \\ & \cdot & & \\ 1 & & & \end{pmatrix} \tag{11}$$

where $\bar{I}_m$ is an antidiagonal matrix of order m. The bit-level permutations in a general case are illustrated in Fig. 3. In our example case, the permutations are performed in the block "permut" and the first two, i.e. the maximum butterfly column s is 2, permutations from Fig. 3 are needed.

3.3 Lookup Table Index

Our approach is to store the twiddle factors to a lookup table and the indexing into the table is based on the exponent k in the twiddle factor W_N^k as defined in (6). Different

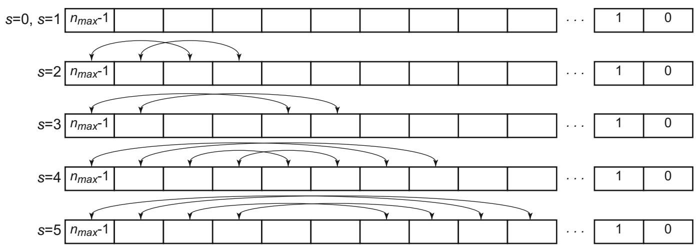

Fig. 3. Examples of bit-level index permutations according to (11)

values of k in FFTs can also be seen in Fig. 1. By exploiting the property $W_{aN}^{a} = W_N$, we can express the twiddle factors as powers of $W_{N_{max}}$. The twiddle factors for radix-4 algorithm are defined in (6) and we can rewrite this equation as follows:

$$A_N^s = Q_N^s \bigoplus_{b=0}^{N} W_{N_{max}}^{(b \bmod 4)\lfloor \frac{\lfloor b/4 \rfloor 4^{s+1}}{N} \rfloor \frac{N_{max}}{4^{s+1}}} \tag{12}$$

where mod is the modulus operation. Here we need an equation for the exponent k for factor $W_{N_{max}}^{k}$, which can be found from the previous. In addition, we have used a shifted index and, therefore, the index b can be replaced with the permuted index l from (11) by the relation $l = bN_{max}/N$, thus we obtain

$$k = (b \bmod 4)\lfloor \frac{\lfloor b/4 \rfloor 4^{s+1}}{N} \rfloor \frac{N_{max}}{4^{s+1}} = \left[\lfloor \frac{lN}{N_{max}} \rfloor \bmod 4\right] \left[\lfloor \frac{\lfloor l/4 \rfloor 4^{s+1}}{N_{max}} \rfloor \frac{N_{max}}{4^{s+1}}\right]. \tag{13}$$

We may denote the first term as h and the second as g. Then the operation at bit-level representation can defined as follows:

$$k = hg\ ;\ g = \begin{pmatrix} I_{2s} & \\ & 0_{n_{max}-2s-2,n_{max}-2s} \end{pmatrix} l\ ;\ h = (0_{2,n_{max}-f-2}, I_2, 0_{2,f})\, l \tag{14}$$

where $0_{n,m}$ is an $n \times m$ matrix containing zeros and f is the input operand defining the index shift, $f = n_{max} - \log_2(N)$. In the example case in Fig. 2, the block "mask" generates a 6-bit mask, where the $2s$ most significant bits are ones and the rest are zeros. This is used to mask the 6-bit permuted index l. Then the two least significant bits are omitted and the 4-bit result is passed to multiplication with h. Since h is a 2-bit variable, a simple solution is to us adder, where the same operand is fed but the second one is shifted one bit to right, i.e., multiplied by two. Multiplexers can be used to feed either the operand or zero to the adder and these multiplexers are controlled by the multiplicand h.

The 2-bit variable h needs to be extracted from l with the aid of multiplexer controlled with f. Figure 2 indicates that h can be extracted also from d, which shortens the critical path. The block "S" performs the extraction, i.e., $h = (h_1, h_0)^T = (d_{f+1}, d_f)^T$.

In the last butterfly column of mixed-radix-4/2 algorithm, the twiddle factors have a bit different form and by using the fact that, in the last column, $s = \log_4(N/2)$ we may rewrite (10) as follows:

$$B_N = Q_N^s \bigoplus_{b=0}^{N} W_{N_{max}}^{(b \bmod 2)\lfloor \frac{\lfloor b/2 \rfloor 4^{(s+1)}}{N} \rfloor \frac{N_{max}}{4^{(s+1)}}}. \tag{15}$$

By following the procedure used to define the exponent k in radix-4 case, we can define k in this case as follows

$$k = (b \bmod 2)\lfloor \frac{\lfloor b/2 \rfloor 4^{s+1}}{N} \rfloor \frac{N_{max}}{4^{s+1}} = \left[\lfloor \frac{lN}{N_{max}} \rfloor \bmod 2 \right] \left[\lfloor \frac{\lfloor \frac{l}{2} \rfloor 4^{s+1}}{N_{max}} \rfloor \frac{N_{max}}{4^{s+1}} \right]. \tag{16}$$

By comparing this equation to (13), we find that there is a scaling difference in the second term and, if the same hardware is used to generate exponent for both radix-4 and mixed-radix-4/2 twiddle factors, this needs to be compensated. In bit-level representation, this can be defined as

$$k = 2hg \; ; \; h = (0_{1,n_{max}-f-1}, 1, 0_{1,f})\, l \tag{17}$$

where g is obtained as in (14). The example case in Fig. 2 shows a block “detectmr”, which is used to detect when mixed-radix algorithm is used and the twiddle factors are for the last butterfly column consisting of the radix-2 butterflies. In this case, the least significant bit of f can be used to detect the mixed-radix transform and the detection of the last butterfly column is detected with the aid of hard-coded detection. Signal “r2” is active-high, which masks the signal “h(1)” since h is only a 1-bit parameter. In addition, the operand g is directed to the lower input of the adder, where the operand is shifted one bit to the left, thus the additional multiplication by two is realized.

3.4 Memory Reduction

Here we propose a method to reconstruct twiddle factors for radix-4 and mixed-radix-4/2 FFT from a ROM table containing $N/8+1$ coefficients. The twiddle factors in 64-point radix-4 FFT are shown in Table 1 and it can be seen that by reordering the coefficients into six blocks, $B0 \dots B5$, all the twiddle factors can be retrieved from coefficients in block $B0$ containing nine complex coefficients. Since we need to support several transform sizes up to an N_{max}-point FFT, we store $(N_{max}/8+1)$ complex-valued coefficients into a table, $M = (M_0, M_1, \dots, M_{N/8}) \mid M_k = W_{N_{max}}^k$. The rest of the twiddle factors can be obtained from the table M as follows:

$$W_{N_{max}}^k = \begin{cases} M_k & , \quad k \leq \frac{N_{max}}{8} \\ -jM_{\frac{N_{max}}{4}-k} & , \frac{N_{max}}{8} < k \leq \frac{N_{max}}{4} \\ -jM^*_{k-\frac{N_{max}}{4}} & , \frac{N_{max}}{4} < k \leq \frac{3N_{max}}{8} \\ -M^*_{\frac{N_{max}}{2}-k} & , \frac{3N_{max}}{8} < k \leq \frac{N_{max}}{2} \\ -M_{k-\frac{N_{max}}{2}} & , \frac{N_{max}}{2} < k \leq \frac{5N_{max}}{8} \\ jM^*_{\frac{3N_{max}}{4}-k} & , \frac{5N_{max}}{8} < k \end{cases} \tag{18}$$

where M_k^* is the complex conjugate of M_k.

Table 1. Twiddle factors in 64-point radix-4 FFT. The decimal value is shown as (real,imaginary).

B0	B1	B2	B3	B4	B5
W_{64}^{0} (1.0,0.0)	W_{64}^{16} (.00,-1.0)				
W_{64}^{1} (1.0,-.10)	W_{64}^{15} (.10,-1.0)			W_{64}^{33} (-1.0,.10)	
W_{64}^{2} (.98,-.20)	W_{64}^{14} (.20,-.98)	W_{64}^{18} (-.20,-.98)	W_{64}^{30} (-.98,-.20)		
W_{64}^{3} (.96,-.29)	W_{64}^{13} (.29,-.96)				W_{64}^{45} (-.29,.96)
W_{64}^{4} (.92,-.38)	W_{64}^{12} (.38,-.92)	W_{64}^{20} (-.38,-.92)	W_{64}^{28} (-.92,-.38)	W_{64}^{36} (-.92,.38)	
W_{64}^{5} (.88,-.47)	W_{64}^{11} (.47,-.88)	W_{64}^{21} (-.47,-.88)	W_{64}^{27} (-.88,-.47)		
W_{64}^{6} (.83,-.56)	W_{64}^{10} (.56,-.83)	W_{64}^{22} (-.56,-.83)	W_{64}^{26} (-.83,-.56)		W_{64}^{42} (-.56,.83)
W_{64}^{7} (.77,-.63)	W_{64}^{9} (.63,-.77)			W_{64}^{39} (-.77,.63)	
W_{64}^{8} (.71,-.71)		W_{64}^{24} (-.71,-.71)			

In order to generate correct twiddle factor W_N^k for the given exponent k defined earlier, we need to create an index to the table M. Such a method can be obtained by noting the fact that the twiddle factors are defined by vectors with equal spaced angles along a unit circle, thus when starting from zero angle the indices to the table M increase by one until $k = N/8$. Then the indices decrease until $k = N/4$ and they start to increase again. This behavior results in six regions as shown in Table 1.

In bit-level, we may generate the index to lookup table by dividing the bit-field k into two parts; the three most significant bits of k are denoted as $q = (k_{n_{max}-1}, k_{n_{max}-2}, k_{n_{max}-3})^T$ and the least significant bits by $r = (k_{n_{max}-4}, \ldots, k_1, k_0)^T$. The index to the lookup table is obtained as follows

$$w = \begin{cases} r & \text{, if } q_0 = 0 \\ \sim r + 1 & \text{, otherwise} \end{cases} \tag{19}$$

where $\sim r$ denotes inversion of bits in r. This can be seen in the lower part in Fig. 2. The index w is used to access the lookup table M ("LUT" in Fig. 2) and the obtained complex value M_w needs to be modified according to (18), which shows that the correct twiddle factor can be obtained as follows

$$W_{N_{max}}^{k} = \begin{cases} (-1)^{q_0 \triangledown q_2} \Re(M_w) + j(-1)^{q_0 \triangledown q_1 \triangledown q_2} \Im(M_w) & \text{, if } q_0 \triangledown q_1 = 0 \\ (-1)^{q_0 \triangledown q_2} \Im(M_w) + j(-1)^{q_0 \triangledown q_1 \triangledown q_2} \Re(M_w) & \text{, otherwise} \end{cases} \tag{20}$$

where $\triangledown$ denotes bitwise exclusive-OR operation and $\Re(x)$ and $\Im(x)$ denote real and imaginary part of x, respectively. Figure 2 shows that this modification requires two multiplexors and two real adders with XOR-gates in inputs.

4 Experiments

We have described the proposed twiddle factor unit in VHDL language such that $N_{max} = 2^{14}$, i.e., the unit supports power-of-two FFTs up to 16K, thus lookup table contains 2049 complex-valued coefficients. The inputs to the unit are 17-bit c register and 4-bit

Table 2. Power dissipation and area of twiddle factor unit designs: proposed unit, pipelined (two stages) and non-pipelined, and unit based on ROM table [5]

	pipelined@250MHz		non-pipelined@140MHz		ROM table [5]@250MHz	
	Power [mW]	Area [kgates]	Power [mW]	Area [kgates]	Power [mW]	Area [kgates]
LUT	1.50	12	2.24	15.8	43.00	20.5
Pipeline	0.95	0.3				
Total	3.70	14.3	4.11	18.4	43.00	20.5

f register. The lookup table contains complex-valued coefficients with 16-bit real and imaginary parts, i.e., word width of lookup table is 32 bits.

The design has been synthesized with Synopsys tools onto a 130 nm standard cell technology. Then power estimates have been obtained with Synopsys tools with aid of gate level simulations. The analysis results are listed in Table 2.

The analysis results show that the critical path limits the clock frequency to 140 MHz when no pipelining is exploited. When two pipeline stages are used, the maximum clock frequency is 275 MHz. The lookup table has been designed with hard-wired logic for reducing the power consumption. If the lookup table was implemented as a ROM memory, the power consumption would have been eight times higher, although the area had been half smaller.

For comparison, we have also implemented a unit based on the traditional ROM table approach where $N_{max}/2 = 8192$ coefficients are stored ("ROM table" in Table 2). The method in [9] is not compared since it does not support radix-4 algorithms.

We have also the twiddle factor unit in an ASIP tailored for FFT computations [12] and, in this 32-bit processor containing, e.g., complex multiplier and adder, the twiddle factor unit uses about 23% of the core area (instruction and data memories not included), while the power consumption is only 7% of the total power consumption of the core. However, the unit improved significantly the performance of the FFT software implementation; the unit provides twiddle factor once per instruction cycle without additional address manipulation instructions.

5 Conclusions

In this paper, we have described a twiddle factor unit supporting radix-4 and mixed-radix-4/2 FFT algorithms and several power-of-two FFT sizes. The unit can be used as a special unit in an ASIP architecture or a coefficient generator in application-specific FFT processors. The unit shows significant power savings compared to the popular approach where the twiddle factors are stored into a ROM table.

Acknowledgement

This work has been supported in part by the Academy of Finland under project 205743 and the Finnish Funding Agency for Technology and Innovation under research funding decision 40441/05.

References

1. Wu, C.S., Wu, A.Y.: Modified vector rotational CORDIC(MVR-CORDIC algorithm and its application to fft. In: Proc. IEEE ISCAS, Geneva, Switzerland, vol. 4, pp. 529–532 (2000)
2. Xiu, L., You, Z.: A new frequency synthesis method based on flying-adder architecture. IEEE Trans. Circuits Syst. 50(3), 130–134 (2003)
3. Fliege, N.J., Wintermantel, J.: Complex digital oscillator and FSK modulators. IEEE Trans. Signal Processing 40(2), 333–342 (1992)
4. Chi, J.C., Chen, S.G.: An efficient FFT twiddle factor generator. In: Proc. European Signal Process. Conf., Vienna, Austria, pp. 1533–1536 (2004)
5. Cohen, D.: Simplified control of FFT hardware. IEEE Trans. Acoust., Speech, Signal Processing 24(6), 577–579 (1976)
6. Chang, Y., Parhi, K.K.: Efficient FFT implementation using digit-serial arithmetic. In: Proc. IEEE Workshop Signal Process. Syst., Taipei, Taiwan, pp. 645–653. IEEE Computer Society Press, Los Alamitos (1999)
7. Ma, Y., Wanhammar, L.: A hardware efficient control of memory addressing for high-performance FFT processors. IEEE Trans. Signal Processing 48(3), 917–921 (2000)
8. Wanhammar, L.: DSP Integrated Circuits. Academic Press, San Diego, CA (1999)
9. Hasan, M., Arslan, T.: FFT coefficient memory reduction technique for OFDM applications. In: Proc. IEEE ICASSP, Orlando, FL, vol. 1, pp. 1085–1088 (2002)
10. Chu, E., George, A.: Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms. CRC Press, Boca Raton, FL (2000)
11. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., Takala, J.: Low-power, high-performance TTA processor for 1024-point fast Fourier transform. In: Vassiliadis, S., Wong, S., Hämäläinen, T.D. (eds.) SAMOS 2006. LNCS, vol. 4017, pp. 227–236. Springer, Heidelberg (2006)
12. Pitkänen, T., Mäkinen, R., Heikkinen, J., Partanen, T., Takala, J.: Transport triggered architecture processor for mixed-radix FFT. In: Proc. Asilomar Conf. Signals, Systems, and Computers, Pacific Grove, CA (2006)
13. Rabiner, L.R., Gold, B.: Theory and Application of Digital Signal Processing. Prentice Hall, Englewood Cliffs (1975)
14. Granata, J., Conner, M., Tolimieri, R.: Recursive fast algorithms and the role of the tensor product. IEEE Trans. Signal Processing 40(12), 2921–2930 (1992)
15. Akopian, D., Takala, J., Astola, J., Saarinen, J.: Multistage interconnection networks for parallel Viterbi decoders. IEEE Trans. Commun. 51(9), 1536–1545 (2003)
16. Bóo, M., Argüello, F., Bruguera, J., Doallo, R., Zapata, E.: High-performance VLSI architecture for the Viterbi algorithm. IEEE Trans. Commun. 45(2), 168–176 (1997)