INTERNATIONAL JOURNAL OF TECHO-ENGINEERING

ISSN : 2057-5688

A Novel Analysis Low Power German Radix and Split Radix FFT Processors Using Radix-2 Butterfly Units

[1] ANIL KUMAR PATNALA, [2] S. RAJESH

[1] M Tech Student, Dept of ECE, Avanthi's st theressa institute of engg and tech, Vizianagaram, AP, India.

[2] Assistant Professor, Dept of ECE, Avanthi's st theressa institute of engg and tech, Vizianagaram, AP, India.

Abstract:

Split-radix fast Fourier transform (SRFFT) is an ideal candidate for the implementation of a low-power and high speed FFT processor, because it has the lowest number of arithmetic operations among all the FFT algorithms. In the design of such processors, an efficient addressing scheme for FFT data as well as twiddle factors is required. The signal flow graph of SRFFT is the same as radix-2 FFT, and therefore, the conventional address generation schemes of FFT data could also be applied to SRFFT. However, SRFFT has irregular locations of twiddle factors and forbids the application of radix-2 address generation methods. This brief presents a shared-memory low-power SRFFT processor architecture. We also introduce German Radix FFT and show that SRFFT and GRFFT can be computed by using a modified radix-2 butterfly unit. The time constraint and power constraint will be calculated. Both SRFFT and GRFFT is been compared. Implementation is done using Mat Lab Tool.

Keywords:

FFT, Butterfly, Radix 2, SRFFT, GRFFT

1. Introduction:

The fast Fourier transform (FFT) is one of the most important and fundamental algorithms in the digital signal processing area. Since the discovery of FFT, have so many variants of the FFT algorithm have been developed, such as radix-2 and radix-4 FFT. In 1984, Duhamel and Hollmann [1] proposed a new variant of FFT algorithm called split-radix FFT (SRFFT). Their algorithm requires the least number of multiplications and additions among all the known FFT algorithms. Since arithmetic operations significantly contribute to overall system power consumption, SRFFT is a good candidate for the implementation of a low-power FFT processor. In general, all the FFT processors can be categorized into two main groups: pipelined processors or shared-memory processors. Examples of pipelined FFT processors can be found in [2] and [3]. A pipelined architecture provides high throughputs, but it requires more hardware resources at the same time. One or multiple pipelines are often implemented, each consisting of butterfly units and control logic. In shared-memory-based architecture requires the least amount of hardware resources at the expense of slower throughput. Examples of such processors can be found in [4] and [5]. In the radix-2 shared-memory architecture, the FFT data are organized into two memory banks. At each clock cycle, two FFT data are provided by memory banks and one butterfly unit is used to process the data. At the next clock cycle, the calculation results are written back to the memory banks and replace the old data. The scope of this brief is limited to the shared-memory architecture.

In the shared-memory architecture, an efficient addressing scheme for FFT data as well as coefficients (called twiddle factors) is required. For the fixed-radix FFT, previous works of this topic can be found in [5] and [6]. For split-radix FFT, it conventionally involves an L-shaped butterfly data path whose irregular shape has uneven latencies and makes scheduling difficult. In this brief, we show that the SRFFT can be computed by using a modified radix-2 butterfly structure. Our contribution consists of mapping the split-radix FFT algorithm to the shared-memory architecture, leveraging the lower

ISSN : 2057-5688

multiplicative complexity of the algorithm to reduce the dynamic power and developing two novel twiddle factor addressing schemes for the split-radix FFT.

2. Butterfly Diagram:

In the context of fast Fourier transform algorithms, a butterfly is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into subtransforms). The name "butterfly" comes from the shape of the data-flow diagram in the radix-2 case, as described below. The earliest occurrence in print of the term is thought to be in a 1969 MIT technical report. The same structure can also be found in the Viterbi algorithm, used for finding the most likely sequence of hidden states.

Most commonly, the term "butterfly" appears in the context of the Cooley–Tukey FFT algorithm, which recursively breaks down a DFT of composite size n = rm into r smaller transforms of size m where r is the "radix" of the transform. These smaller DFTs are then combined via size-r butterflies, which themselves are DFTs of size r (performed m times on corresponding outputs of the sub-transforms) pre-multiplied by roots of unity (known as twiddle factors). (This is the "decimation in time" case; one can also perform the steps in reverse, known as "decimation in frequency", where the butterflies come first and are post-multiplied by twiddle factors.

Radix 2 Butterfly Diagram:

In the case of the radix-2 Cooley–Tukey algorithm, the butterfly is simply a DFT of size-2 that takes two inputs (x_0, x_1) (corresponding outputs of the two sub-transforms) and gives two outputs (y_0, y_1) by the formula (not including twiddle factors):

$$y_0 = x_0 + x_1$$
$$y_1 = x_0 - x_1$$

If one draws the data-flow diagram for this pair of operations, the (x_0, x_1) to (y_0, y_1) lines cross and resemble the wings of a butterfly, hence the name (see also the illustration at right).

$$y_0 = x_0 + x_1\omega_n^k$$
$$y_1 = x_0 - x_1\omega_n^k,$$

where, $\omega_n^k = e^{-\frac{2\pi ik}{n}}$

where k is an integer depending on the part of the transform being computed. Whereas the corresponding inverse transform can mathematically be performed by replacing ω with ω^{-1} (and possibly multiplying by an overall scale factor, depending on the normalization convention), one may also directly invert the butterflies:

$$x_0 = \frac{1}{2}(y_0 + y_1)$$
$$x_1 = \frac{\omega_n^{-k}}{2}(y_0 - y_1),$$

corresponding to a decimation-in-frequency FFT algorithm.

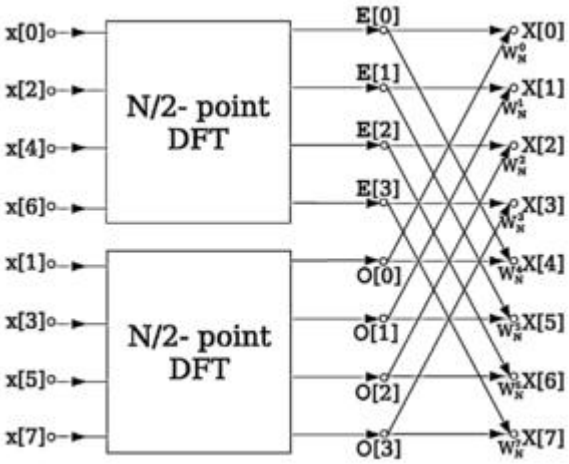

Fig 1. A decimation-in-time radix-2 FFT breaks a length-N DFT into two length-N/2 DFTs followed by a combining stage consisting of many butterfly operations.

3.Comparison of SRFFT And Radix-2 FFT:

ISSN : 2057-5688

The N-point discrete Fourier transform is defined by

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

where $k = 0, 1, \ldots, N-1$ and

$$W_N^{nk} = e^{-j2\pi nk/N}$$

If we split *X(k)*

into even and odd terms, radix-2 FFT can be derived as

$$X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)] W_{N/2}^{nk}$$

$$X(2k+1) = \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)] W_N^n W_{N/2}^{nk}.$$

The basic idea behind the SRFFT is the application of a radix-2 index map to the even-index terms and a radix-4 map to the odd-index terms.

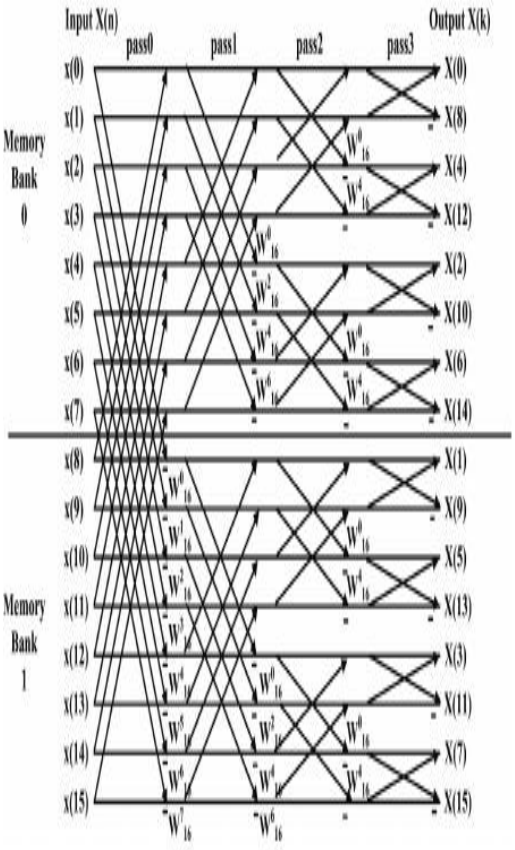

Fig 2. Signal Flow graph of Radix 2 FFT

For the even-index terms, it can be decomposed as (2). For the odd-index terms, it can be decomposed as

$$X(4k+1) = \sum_{n=0}^{N/4-1} [x(n) - x(n+N/2) - j(x(n+N/4) - x(n+3N/4))] W_N^n W_{N/4}^{nk}$$

$$X(4k+3) = \sum_{n=0}^{N/4-1} [x(n) - x(n+N/2) + j(x(n+N/4) - x(n+3N/4))] W_N^n W_{N/4}^{nk}$$

where $k = 0, 1, \ldots, N/4$.

ISSN : 2057-5688

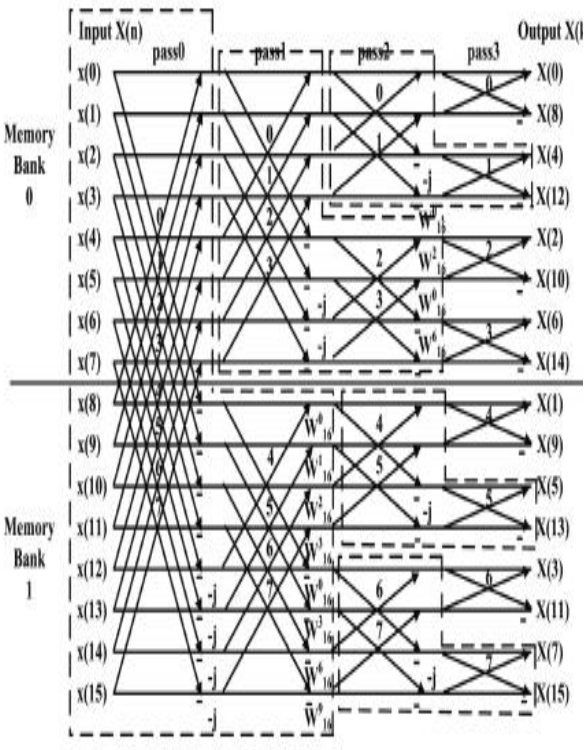

Fig 3. Signal Flow Graph of SRFFT

Each L butterfly contains two nontrivial complex multiplications, and therefore, the total number of nontrivial complex multiplications *M*SR in SRFFT is

$$M_{\mathrm{SR}} = [(3S-2)2^{S-1} + (-1)^S]2/9.$$

In the *(S* − 1*)*th pass, the number of SR butterfly N_{S-1} is

$$N_{S-1} = [2 + (-1/2)^{S-2}]N/12$$

However, in the *(S* − 1*)*th pass, each L butterfly does not contain any nontrivial twiddle factors and hence, the total number of nontrivial multiplications M'_{SR} in SRFFT is

$$M'_{\mathrm{SR}} = M_{\mathrm{SR}} - 2N_{S-1}$$

For the conventional radix-2 FFT, the total number of complex multiplications M_{R2} is

$$M_{R2} = 2^{S-1}(S-1)$$

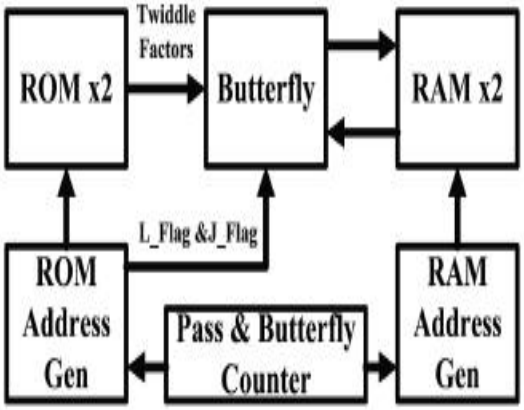

Fig 4. Shared Memory Architecture

4. SRFFT and DRFFT:

The split-radix FFT, along with its variations, long had the distinction of achieving the lowest published arithmetic operation count (total exact number of required real additions and multiplications) to compute a DFT of power-of-two sizes N.

The split-radix algorithm can only be applied when N is a multiple of 4, but since it breaks a DFT into smaller DFTs it can be combined with any other FFT algorithm as desired.

Split Radix Decomposition:

Recall that the DFT is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n \omega_N^{nk}$$

where K is an integer ranging from 0 to N-1 and w_n denotes the primitive root of unity:

$$\omega_N = e^{-\frac{2\pi i}{N}}$$

and thus $\omega_N^N = 1$

The split-radix algorithm works by expressing this summation in terms of three smaller summations. (Here, we give the "decimation in time" version of the split-radix FFT; the dual decimation in frequency version is essentially just the reverse of these steps.)

ISSN : 2057-5688

First, a summation over the even indices x_{2n_2}. Second, a summation over the odd indices broken into two pieces: x_{4n_4+1} and x_{4n_4+3}, according to whether the index is 1 or 3 modulo 4. Here, n_m denotes an index that runs from 0 to N/m -1. The resulting summations look like:

$$X_k = \sum_{n_2=0}^{N/2-1} x_{2n_2}\omega_{N/2}^{n_2 k} + \omega_N^k \sum_{n_4=0}^{N/4-1} x_{4n_4+1}\omega_{N/4}^{n_4 k} + \omega_N^{3k} \sum_{n_4=0}^{N/4-1} x_{4n_4+3}\omega_{N/4}^{n_4 k}$$

where we have used the fact that,

$$\omega_N^{mnk} = \omega_{N/m}^{nk}$$

These three sums correspond to portions of radix-2 (size N/2) and radix-4 (size N/4) Cooley-Tukey steps, respectively. (The underlying idea is that the even-index subtransform of radix-2 has no multiplicative factor in front of it, so it should be left as-is, while the odd-index subtransform of radix-2 benefits by combining a second recursive subdivision.)

These smaller summations are now exactly DFTs of length N/2 and N/4, which can be performed recursively and then recombined.

More specifically, let U_k denote the result of the DFT of length N/2 (for $k = 0,\ldots\ldots\ldots\ldots, {}^{N}/_{2-1}$), and let z_k and z'_k denote the results of the DFTs of length N/4 (for $k = 0,\ldots\ldots\ldots\ldots, {}^{N}/_{4-1}$). Then the output ${}^{N}/_{4}$ is simply:

$$X_k = U_k + \omega_N^k Z_k + \omega_N^{3k} Z'_k.$$

This, however, performs unnecessary calculations, since $K \geq N/4$ turn out to share many calculations with $K < N/4$. In particular, if we add N/4 to k, the size-N/4 DFTs are not changed (because they are periodic in k), while the size-N/2 DFT is unchanged if we add N/2 to k. So, the only things that change are the w_n^K and w_n^{3K} terms, known as twiddle factors. Here, we use the identities:

$$\omega_N^{k+N/4} = -i\omega_N^k$$
$$\omega_N^{3(k+N/4)} = i\omega_N^{3k}$$

to finally arrive at:

$$X_k = U_k + \left(\omega_N^k Z_k + \omega_N^{3k} Z'_k\right),$$
$$X_{k+N/2} = U_k - \left(\omega_N^k Z_k + \omega_N^{3k} Z'_k\right),$$
$$X_{k+N/4} = U_{k+N/4} - i\left(\omega_N^k Z_k - \omega_N^{3k} Z'_k\right),$$
$$X_{k+3N/4} = U_{k+N/4} + i\left(\omega_N^k Z_k - \omega_N^{3k} Z'_k\right),$$

which gives all of the outputs X_k if we let k range from 0 to $\frac{N}{2}$ in the above four expressions. Notice that these expressions are arranged so that we need to combine the various DFT outputs by pairs of additions and subtractions, which are known as butterflies. In order to obtain the minimal operation count for this algorithm, one needs to take into account special cases for (where the twiddle factors are unity) and for $k = \frac{N}{8}$ (where the twiddle factors are ${}^{(i\mp i)}/_{\sqrt{2}}$ and can be multiplied more quickly); see, e.g. Sorensen et al. (1986). Multiplications by ∓ 1 and $\mp i$ are ordinarily counted as free (all negations can be absorbed by converting additions into subtractions or vice versa).

DRFFT:

German style DIT varies the algorithm effectively. The level of stage reduction is more by implementing GRFFT. For example if stages N=8, then it get reduces to Log n to base 2 stages. By which the implementation is performed in less time and the processor speed can be increased.

5. Results:

ISSN : 2057-5688

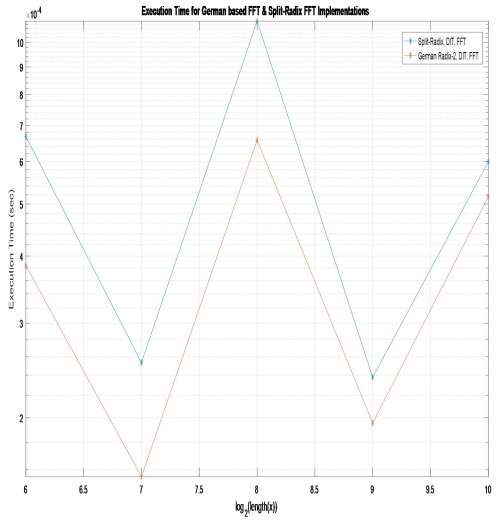

Fig 5. Execution time vs Length

The above fig 5 show the execution time taken by the processor at different lengths. The execution time using German based FFT and Split Radix FFT are compared and shown in the above figure. It also shows that GRFFT obtained better outputs.

Execution Time : 0.000668 sec

Power consumption : 0.012782 W

Execution Time : 0.000384 sec

Power consumption : 0.012775 W

	Power(W)	**Time(Sec)**
SRFFT	**0.012782**	**0.000668**
German Radix FFT	**0.012775**	**0.000384**

6. Conclusion:

In this project the comparison of HIGH SPEED SRFFT and German Radix FFT is been performed. The main aim is to improve the speed of the processor, which is one of the major constraint in VLSI processing. The power also plays a important role. The power constraint is also calculated. The results obtained shows that the time constraint is more reduced by using German Radix FFT when compared with SRFFT. Hence the German Radix FFT can be used in many processors. The power variation is very low when compared to both the techniques. The above is implemented used MATLAB tool.

References:

[1] P. Duhamel and H. Hollmann, "'Split radix' FFT algorithm," Electron. Lett., vol. 20, no. 1, pp. 14–16, Jan. 1984.

[2] M. A. Richards, "On hardware implementation of the split-radix FFT," IEEE Trans. Acoust., Speech Signal Process., vol. 36, no. 10, pp. 1575–1581, Oct. 1988.

[3] J. Chen, J. Hu, S. Lee, and G. E. Sobelman, "Hardware efficient mixed radix-25/16/9 FFT for LTE systems," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 2, pp. 221–229, Feb. 2015.

[4] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 39, no. 5, pp. 312–316, May 1992.

[5] D. Cohen, "Simplified control of FFT hardware," IEEE Trans. Acoust., Speech, Signal Process., vol. 24, no. 6, pp. 577–579, Dec. 1976.

[6] X. Xiao, E. Oruklu, and J. Saniie, "An efficient FFT engine with reduced addressing logic," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 55, no. 11, pp. 1149–1153, Nov. 2008.

[7] Z. Qian, N. Nasiri, O. Segal, and M. Margala, "FPGA implementation of low-power split-radix FFT processors," in Proc. 24th Int. Conf. Field Program. Logic Appl., Munich, Germany, Sep. 2014, pp. 1–2.

[8] A. N. Skodras and A. G. Constantinides, "Efficient computation of the split-radix FFT," IEE Proc. F-Radar Signal Process., vol. 139, no. 1, pp. 56–60, Feb. 1992.

[9] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On computing the split-radix FFT," IEEE Trans. Acoust., Speech Signal Process., vol. 34, no. 1, pp. 152–156, Feb. 1986.

[10] J. Kwong and M. Goel, "A high performance split-radix FFT with constant

ISSN : 2057-5688

geometry architecture," in Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE), Dresden, Germany, Mar. 2012, pp. 1537–1542.
[11] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," IEEE Trans. Signal Process., vol. 51, no. 3, pp. 864–874, Mar. 2003.