An Efficient FFT Engine Based on Twin-Precision Computation

Martin Brinck, Kristian Eklund, Magnus Själander, and Per Larsson-Edefors
VLSI Research Group, Department of Computer Science and Engineering
Chalmers University of Technology, SE-412 96 Göteborg, Sweden

***Abstract*— Applications for System-on-Chips have very different requirements regarding operand precision. We propose a twin-precision FFT engine that can efficiently perform either the common full-precision operation or two simultaneous and independent half-precision operations in the same hardware block. We provide an evaluation on energy, delay and area for synthesized 0.13-μm butterfly circuits, and show how many lower-precision operations that are needed for the twin-precision FFT butterfly to perform better than a conventional, dedicated FFT butterfly.**

I. Introduction

With the concept of System-on-Chip (SoC) followed a new level of integration, where fairly large and dissimilar resources could be deployed in a single chip. As embedded applications are in high demand, so are SoCs with richer and richer functionality. Not only is the trend towards a larger number of applications on each SoC significant, but this trend is superposed with the trend of an increasing number of versions or standards for almost every application.

The limit to integration onto a SoC lies not only in die size, but also in power dissipation. We cannot afford to use one dedicated hardware unit for every conceivable version of every application, because this will waste far too much area and power. Therefore, it is important to find ways to efficiently share hardware resources!

It is clear that the dynamic range of the digital words that make up the operands can vary greatly from one application to another. Even within a narrow application domain, different versions of an application can have very different requirements on operand precision. The Fast Fourier Transform (FFT) is used in numerous electronic applications; wireless communication and multimedia devices are examples on applications where large FFT calculations with different operand precision are needed. We will show how we can take advantage of varying operand precisions, by introducing an efficient and flexible FFT engine.

II. Preliminaries

A. The Fast Fourier Transform

The Discrete Fourier Transform (DFT) is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi \cdot k \cdot n}{N}} = \sum_{n=0}^{N-1} x[n] \cdot {W_N}^{k \cdot n}$$

and the inverse formula (IDFT) as:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j\frac{2\pi \cdot k \cdot n}{N}} = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot {W_N}^{-k \cdot n}$$

where W_N is the twiddle factor and N is the number of points in the transform. The Fast Fourier Transform (FFT) is an algorithm that calculates the transformation in less operations than the DFT. The algorithm used for this paper is a radix-2 Decimation-In-Frequency (DIF) FFT, where the calculation complexity is reduced to $N \cdot \log N$ instead of N^2 as for the DFT [1].

The basic calculation performed in the radix-2 DIF FFT algorithm is shown in Equations 1 and 2, where X_{in} and Y_{in} are complex values and W_N is the twiddle factor on complex form.

$$X_{out} = X_{in} + Y_{in} \tag{1}$$

$$Y_{out} = X_{in} - Y_{in} \cdot W_N \tag{2}$$

An illustration of the basic element of a radix-2 DIF FFT algorithm, commonly known as a butterfly, is shown in Figure 1.

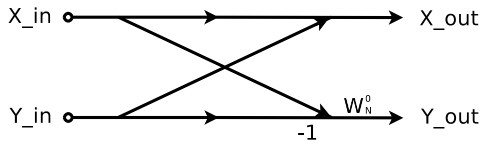


Fig. 1. A DIF FFT butterfly.

The butterfly element is used to construct larger structures to perform larger transformations. A structure of an 8-point FFT using a DIF butterfly is shown in Figure 2, where it also is shown that there are three stages in the 8-point FFT calculation.

B. Twin-Precision Computation

The general idea behind the twin-precision technique [2], [3] is to optionally calculate two lower-precision arithmetic operations simultaneously and independently in a unit designed for an operand precision which is the sum of the two lower-precision operands[1]. It is here possible to partition

[1]Typically, the twin-precision technique uses a lower-precision operand that is half the size of the full-precision operand.

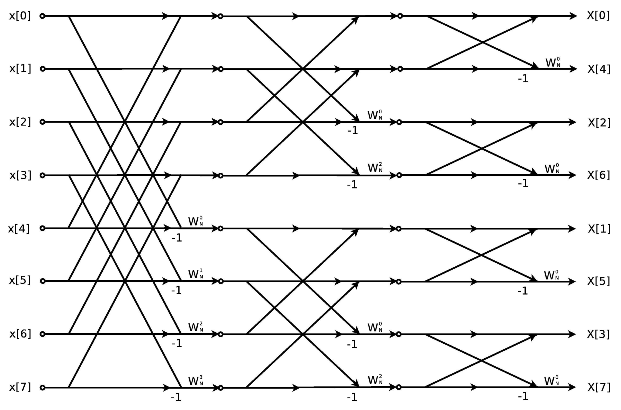


Fig. 2. An 8-point DIF FFT structure.

the bit vector of an operand so that one arithmetic operation is made in the less significant bits and the second one in the most significant bits. In the following, we will refer to two distinct modes of operation in a twin-precision unit: TP-mode is when two half-precision operations are carried out simultaneously, whereas SP-mode is when one full-precision operation is executed.

By reading two locations of the memory, two pairs of data are available at the same time and two butterflies can be calculated simultaneously, which is shown in Figure 3.

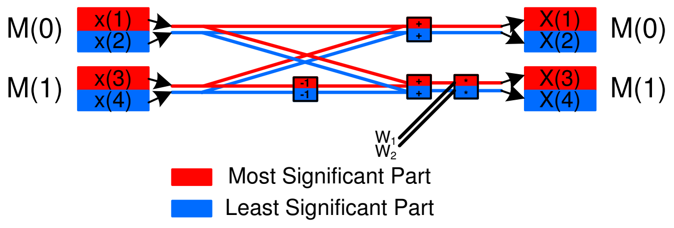


Fig. 3. The twin-precision technique used in a butterfly.

III. Flexible FFT engine

In an FFT implementation there are many parameters that can be made flexible and reconfigurable. In the FFT design of this paper, the focus of flexibility features was on the number of points (N), operand scaling and, more interestingly, the twin-precision technique. The general architecture for the proposed implementation is a radix-2 DIF FFT butterfly which is fed with complex values from an internal RAM [4]; see Figure 4. The twiddle factors are stored in a local ROM and are also fed to the butterfly. The RAM stores the input data values and the calculated values from the butterfly. Since the data that once were used in a butterfly calculation will not be used for any future calculations, the calculated data from the butterfly can be written into the RAM address that the input was taken from. The address generator feeds the RAM with read and write addresses, and read and write signals. The control unit supervises the state of the calculation and makes sure that the different components of the system collaborate in an efficient way.

When comparing a conventional FFT engine with a twin-precision enabled FFT engine, the datapath and control circuits that surround the butterfly are very similar. However, the calculation core of the design, the butterfly unit, proves to be of significance when comparing conventional and twin-precision butterfly implementation quality in aspects of area, delay and energy. Therefore, focus was put on evaluating only the butterfly unit.

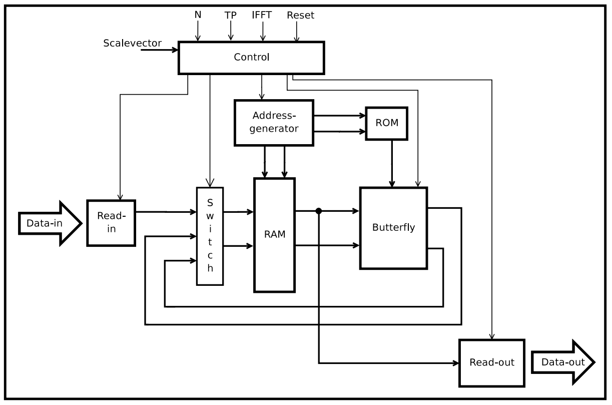


Fig. 4. Overview of the FFT engine.

IV. FFT Butterfly

The butterfly component is the core of the FFT calculation. The proposed twin-precision butterfly is combinatorial and operates on the data in a uniform manner. The data flow in the twin-precision butterfly in TP-mode will be exactly the same as in the SP-mode, except for the last stage in the FFT calculation. In this stage, data need to be rerouted because the two data points, which should be used in the same butterfly equation, are located in the same vector as shown in Figure 5. The necessary rerouting is illustrated in Figure 6. The algorithmic modification needed for the TP-mode is handled by the multipliers [2] and the adders. Addition and subtraction operations in TP-mode utilize the breaking of a carry chain in the middle of the data vector, so that overflow does not affect other bits in the same vector.

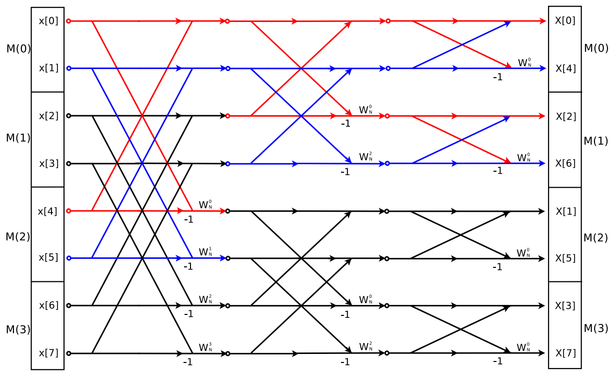


Fig. 5. An 8-point DIF FFT structure, where it can be seen that rerouting is needed for the last stage (Color graph).

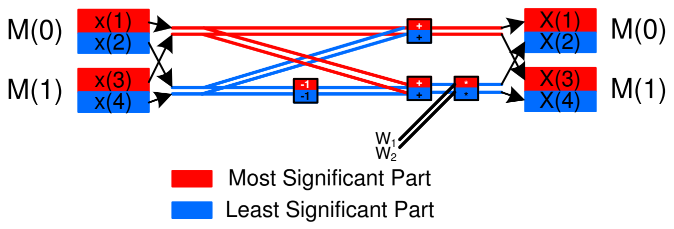


Fig. 6. Data rerouting during the last stage in TP-mode.

The butterfly performs a calculation on two complex input values, which are multiplied with their respective complex twiddle factors. The datapath of the butterfly is mainly divided into three different parts. The first part is a ripple-carry adder, the second is a signed Baugh-Wooley multiplier, while the third part is another ripple-carry adder. Since the bit width of the result after the multiplier is twice as wide as the input data, and since the output data should have the same bit width as the input data, truncation is performed in the last part of the butterfly. To obtain the correct representation, the twiddle factors are scaled up before they are put into the ROM to compensate for the truncation. Consequently, the most significant bits become the output data of the butterfly.

When calculating an FFT, the values returned from the butterfly component to the RAM are potentially larger than the initial values. To prevent overflow, a strategy must be developed to accommodate the expansion of the calculated values. The solution implemented in our design is a scaling vector, in which each bit is represented by a stage in the FFT calculation. If the bit in the scaling vector is set for the actual stage, scaling is performed within the butterfly by shifting the bit vector one step, which is equivalent to a division by two.

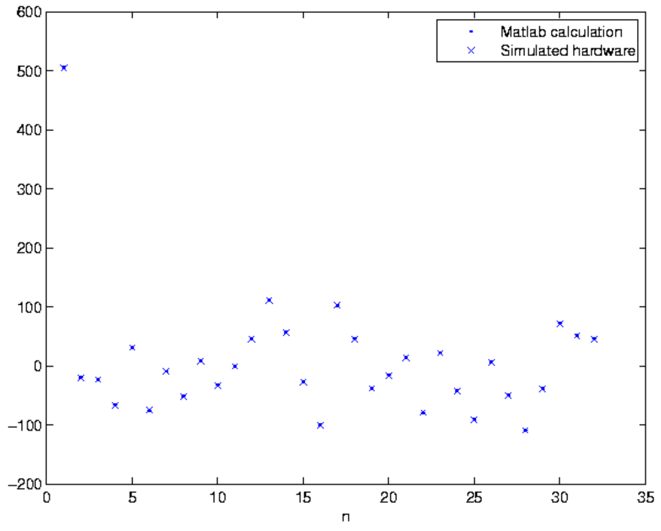
A. Simulation and Results

To verify our twin-precision FFT engine design, randomized test-vector values from Matlab [5] were imported to the Modelsim [6] simulator. The output results of our FFT engine were then compared with the reference results from Matlab. The result from one verification is shown in Figure 7. Due to the complex-valued representation, a 32-bit calculation actually is made up of a 16-bit real and a 16-bit imaginary calculation, in the same way as a 16-bit calculation is represented by an 8-bit real and an 8-bit imaginary one.

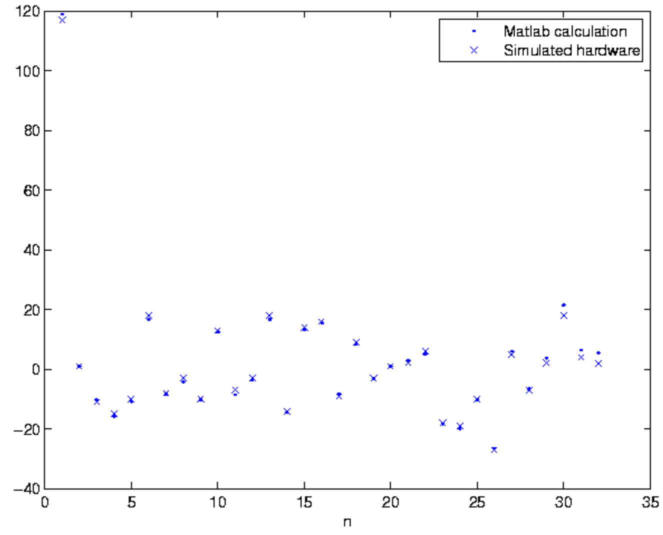
The design was synthesized with Design Compiler [7], using a cell library of a commercial 0.13-μm CMOS technology. To obtain estimations on timing and power dissipation, PrimeTime [8] and PrimePower [9] were used. A 10-fF load on the primary output signals and a medium-sized buffer as driver on the input signals were used for simulations. The simulations were performed with a supply voltage of 1.32 V and at the default temperature of the tools.

1) Delay: The delay for a twin-precision 32-bit butterfly compared to a dedicated 32-bit butterfly and a dedicated 16-bit butterfly is shown in Table I.

The delay in the twin-precision butterfly has two different values depending on in which mode it is operating. When



(a) 32-bit SP-mode: 16-bit real and 16-bit imaginary calculations.



(b) 16-bit TP-mode: 8-bit real and 8-bit imaginary calculations.

Fig. 7. Results from simulation of the hardware implementation compared to results from Matlab for a 32-point FFT calculation.

TABLE I
DELAY IN BUTTERFLY ELEMENTS.

Butterfly	32-bit twin-precision		Dedicated	
	SP-mode	TP-mode	32-bit	16-bit
Delay (ns)	4.18	3.70	3.90	2.17

the butterfly is operating in TP-mode, two 16-bit butterflies are calculated simultaneously. In this mode, the critical path is 3.70 ns which is to be compared to the SP-mode, for which the delay is 4.18 ns. This is because the logic depth of the multiplier and the adders is substantially shorter in the TP-mode. Since the 32-bit twin-precision butterfly can calculate two 16-bit butterflies simultaneously, the throughput is higher in the twin-precision butterfly than in the dedicated 16-bit butterfly. However, when comparing a 32-bit SP-mode calculation in the twin-precision implementation, with the same calculation on the dedicated 32-bit butterfly, there is some loss in delay.

2) Area: The required area for a dedicated butterfly is, as expected, less than for a twin-precision butterfly. When comparing the area of the twin-precision 32-bit butterfly to that of the dedicated 32-bit butterfly, the extra logic area is about 17%. This is the area overhead for the twin-precision flexibility. However, the area for the twin-precision 32-bit butterfly is *not* larger than implementing both a dedicated 32-bit butterfly and a dedicated 16-bit butterfly. The area of the different butterflies is shown in Table II.

TABLE II

AREA OF BUTTERFLY ELEMENTS.

Butterfly	Twin-Precision	Dedicated	
	32-bit	32-bit	16-bit
Area	82382	68223	18072

3) Energy: The total electric energy dissipation for one butterfly operation is shown in Table III. The energy has been calculated by running power simulations, in which the dedicated butterflies were operated at maximum speed while the twin-precision butterfly was operated at its maximum SP-mode speed, shown in Table I.

TABLE III

ELECTRIC ENERGY DISSIPATION IN BUTTERFLY ELEMENTS PER OPERATION.

Butterfly	Twin-precision		Dedicated		
	32-bit		32-bit		16-bit
	SP-mode	TP-mode	FP-mode	HP-mode	
Energy (nJ)	0.162	0.062	0.138	0.136	0.035

As can be seen in Table III, the electric energy dissipation for 16-bit operands that are computed on a conventional 32-bit butterfly is basically identical to that of 32-bit operands. The reason for this is that in the conventional 32-bit butterfly, signed 16-bit operands need to be sign extended to 32-bit representation.

The dedicated butterflies expend less energy than the twin-precision butterfly. The extra electric energy dissipation for a twin-precision 32-bit butterfly is about 18% when operating on 32-bit wide operands. However, the twin-precision butterfly only dissipates 46% of the energy dissipated by a conventional 32-bit butterfly when operating on 16-bit wide operands. This clearly points out that it is inefficient to calculate lower-precision operands on dedicated butterflies.

Figure 8 shows the expended total energy for 1000 butterfly operations when the ratio of half-precision to full-precision operations is varied. If our proposed FFT engine calculates at least 25% 16-bit calculations, it becomes more efficient than a dedicated 32-bit solution.

Because the FFT algorithm operates with iteration over the butterfly, values are read and written to a local memory many times throughout the FFT calculation. This reading and writing to the RAM is a great part of the whole energy dissipation in the FFT calculation. By using the twin-precision technique, the number of memory accesses in the calculation phase can be reduced by 50% compared to a dedicated full-precision butterfly. This will reduce the energy dissipation substantially. Thus, considering the ratio of low- to full-precision operations, the break-even point when the usage of a twin-precision FFT engine makes sense is reduced further.

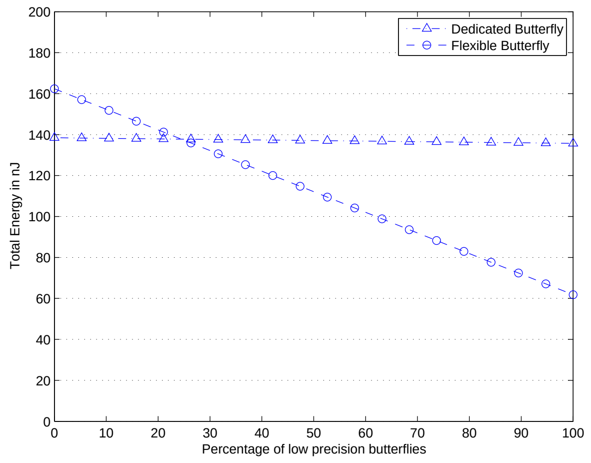


Fig. 8. Total energy expended.

V. CONCLUSION

A reconfigurable FFT engine using the twin-precision technique has been implemented in hardware. The calculation core of the design, the butterfly unit, is shown to be of significance for the final implementation quality in aspects of area, delay and power dissipation. The 32-bit twin-precision butterfly has been compared to both a dedicated 32-bit butterfly and a dedicated 16-bit butterfly. When calculating at least 25% half-precision calculations, we can effect savings in electric energy dissipation by using a flexible twin-precision butterfly, instead of one or two dedicated butterfly(ies). The twin-precision butterfly makes it possible to calculate two half-precision butterflies simultaneously with half the number of accesses to the RAM during calculation, compared to a dedicated full-precision butterfly.

REFERENCES

[1] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput*, vol. 19, pp. 297–301, 1965.
[2] M. Själander, "Efficient Reconfigurable Multipliers Based on the Twin-Precision Technique," Licentiate of Engineering Thesis, Chalmers University of Technology, 2006.
[3] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An Efficient Twin-Precision Multiplier," in *IEEE International Conference on Computer Design*, 2004, pp. 507–510.
[4] M. Brinck and K. Eklund, "A Flexible FFT/DCT Engine Using the Twin-Precision Technique," Master's thesis, Chalmers University of Technology, 2006.
[5] *Matlab version 7.1 help.*
[6] *Modelsim SE PLUS 6.0c.*
[7] *Design Compiler Users Guide Version W-2004.12.*
[8] *PrimeTime Users Guide X-2005.06.*
[9] *PrimePower Manual W-2004.12.*