AN AREA-EFFICIENT AND HIGH SPEED PROCESSOR FOR FLOATING POINT FUSED DOT PRODUCT AND FUSED ADD-SUBTRACT FOR FFT

P.Priyadharshini[1], S.Rinisanzes[2], M.Shalini[3]

[1] *(B.E),Department of ECE,Prince Shri Venkateshwara Padmavathy Engineering College,Tamil Nadu,India*
[2] *(B.E) ,Department of ECE,Prince Shri Venkateshwara Padmavathy Engineering College,Tamil Nadu,India*
[3] *Assistant professor ,Department of ECE,Prince Shri Venkateshwara Padmavathy Engineering College,Tamil Nadu,India*

ABSTRACT

The Fast Fourier Transform (FFT) is one of the rudimentary operations in field of digital signal. Some of the application of the Fast Fourier Transform include Signal analysis, Data compression, Partial differential equation, Multiplication of larges integers etc.,. The FFT can be designed by radix-4 butterfly algorithm which requires needless computations and data storage. The presenting the design and simulation of a 32 bit floating point FFT processor. In our proposed method we used concept of normal Booth encoding algorithm and designed a new DADDA algorithm which can be reduced area. The proposed design is to be implemented for single precision and will be synthesized with 90nm standard cell library. The proposed design fused dot product and fused add /subtract unit take four normalized operands and compute the sum and difference as (AB ± CD) respectively. Experiment result shows that the proposed floating point FDP and FAS unit by VLSI gives high speed, low area hardware floating point FDP and FAS unit.

Keyword: - *Fast Fourier Transform(FFT), Fused Dot Product(FDP) , Fused Add-Subtract(FAS)*

1. INTRODUCTION

Floating-point arithmetic is attractive for implementation for a variety of Digital Signal Processing (DSP) applications because it allows the designer and user to concentrate on the algorithms and architecture without worrying about numerical issues. In the past, many DSP applications used fixed point arithmetic due to the high cost (in delay, silicon area, and power consumption) of floating-point arithmetic unit.In the realization of modern general purpose processors, fused floating-point multiply add units have become attractive since their delay and silicon area is often less than that of a discrete floating-point multiplier followed by a floating point adder. Further the accuracy is improved by the fused implementation since rounding is performed only once (after the multiplication and addition).operations that are frequently encountered in DSP. The Fast Fourier Transform is a case in point since it uses a complex butterfly operation. For a radix-2 implementation, the butterfly consists of a complex multiply and the complex addition and subtraction of the same pair of data. For a radix-4 implementation, the butterfly consists of three complex multiplications and eight complex additions and subtractions.

Both of these butterfly operations can be implemented with two fused primitives, a fused two-term dot-product unit and a fused add-subtract unit. The fused two-term dot-product multiplies two sets of operands and adds the products as a single operation. The two products do not need to be rounded (only the sum is normalized and rounded) which reduces the delay by about 15% while reducing the silicon area by about 33%. For add-subtract unit, much of the complexity of a discrete implementation comes from the need to compare the operand exponents

and align the significant prior to the add and the subtract operations. For the fused implementation, sharing the comparison and alignment greatly reduces the complexity. The delay and the arithmetic results are the same as if the operations are performed in the conventional manner with a floating-point adder and a separate floating-point subtracted. In this case, the fused implementation is about 20% smaller than the discrete equivalent.

2. PROPOSED SYSTEM

2.1 Dadda multiplier

The **Dadda multiplier** is a hardware multiplier design invented by computer scientist Luigi Dadda in 1965. It is similar to the Wallace multiplier, but it is slightly faster (for all operand sizes) and requires fewer gates (for all but the smallest operand sizes)**.**

In fact dadda multiplier are the following steps

- Multiply (logical AND) each bit of one of the arguments, by each bit of the other, yielding results. Depending on position of the multiplied bits, the wires carry different weights, for example wire of bit carrying result is 32.
- Reduce the number of partial products to two by layers of full and half adders.
- Group the wires in two numbers, and add them with a conventional adder.

However, unlike Wallace multipliers that reduce as much as possible on each layer, Dadda multipliers do as few reductions as possible. Because of this, Dadda multipliers have a less expensive reduction phase, but the numbers may be a few bits longer, thus requiring slightly bigger adder.

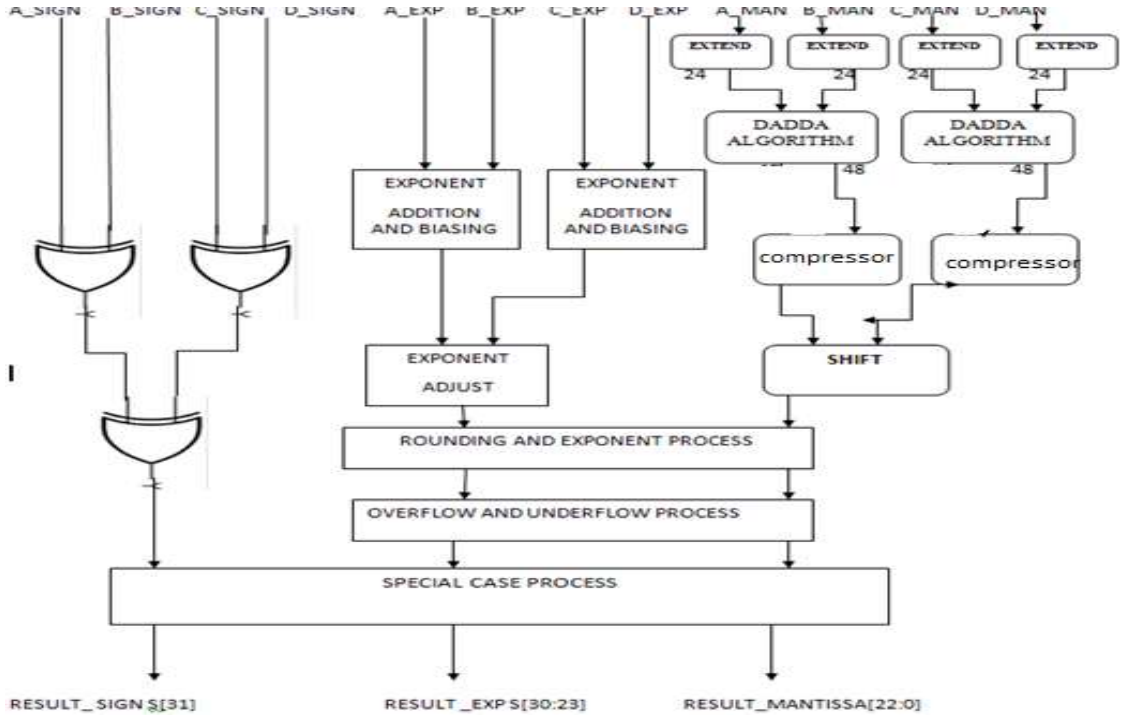

Figure 1 Detailed structure of the fused 2-term dot product unit

To achieve this, the structure of the second step is governed by slightly more complex rules than in the Wallace tree. As in the Wallace tree, a new layer is added if any weight is carried by three or more wires. The reduction rules for the Dadda tree, however, are as follows in figure 1.

- Take any three wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
- If there are two wires of the same weight left, and the current number of output wires with that weight is equal to 2 (modulo 3), input them into a half adder. Otherwise, pass them through to the next layer.
- If there is just one wire left, connect it to the next layer.

This step does only as many adds as necessary, so that the number of output weights stays close to a multiple of 3, which is the ideal number of weights when using full adders as 3:2 compressors.

However, when a layer carries at most three input wires for any weight, that layer will be the last one. In this case, the Dadda tree will use half adder more aggressively (but still not as much as in a Wallace multiplier), to ensure that there are only two outputs for any weight. Then, the second rule above changes as follows:

- If there are two wires of the same weight left, and the current number of output wires with that weight is equal to 1 or 2 (modulo 3), input them into a adder.

2.2 Algorithm dadda multiplier

The section explains the Dadda dot-diagram reduction example

- Let $d_1 = 2$ and $d_{j+1} = \text{floor}(3*d_j/2)$
- This generates the sequence: $d_1=2$, $d_2=3$, $d_3=4$, $d_4=6$, $d_5=9$, $d_6=13$,.
- Find the largest d_j that is less than the maximum number of bits in any column.
- For our example to the right, this would be 6.
- For every column, use full adders (FA) and half adders (HA) to ensure that the number of elements in each column will be $\leq d_j$.

When doing this, keep in mind that any column n that has an adder within it, will pass its sum bit to the next stage in column n and the carry bit to the n+1 column. The n+1 column will need to take this into account when determining the number of adders to use.

2.3 First bank

- Columns 0-5 don't need any adders, since they all have ≤ 6 bits
- Column 6 needs 1 HA (7 > 6) which reduces it to 6 bits and passes one carry bit to column 7.
- Column 7 can use a FA since it has 8 bits which would reduce the column to 6 bits, but since column 6 is passing in a carry bit, it needs one more HA to bring the total to 6 bits
- Column 8 needs a FA and a HA since it is getting 2 carry bits from column 7's adders.
- Column 9 only needs one FA
- Columns 10-14 do not need any adders since any carry bits from the previous columns do not result in a total greater than 6.

2.4 Second bank

The next bank's $d_j = 4$

- Columns 0-3 don't need any adders since they have ≤ 4 bits
- Column 4 needs a HA, (5 > 4)
- Column 5 needs a FA and a HA due to the carry bit
- Columns 6-10 need two FA since they all have 2 carry bits coming from the previous stage
- Column 11 only needs 1 FA to get to 4 bits after the carry bits come in
- Columns 12-14 don't need any adders since they all have < 4 bits

2.5 Third bank

The next bank's $d_j = 3$

- Columns 0-2 don't need any adders since they have ≤ 3 bits
- Column 3 only needs one HA to get to 3 bits
- Column 4-12 need a FA since they all have one carry-in bit coming in from the previous column
- Columns 13-14 don't need any adders since they have < 3 bits

2.6 Fourth bank

The next bank's $d_j = 2$

- Columns 0-1 don't need any adders since they have ≤ 2 bits
- Column 2 only needs one HA to get to 2 bits

- Column 3-13 need a FA since they all have one carry-in bit coming in from the previous column
- Column 14 doesn't need an adder (1 < 2)

2.7 Fifth bank

At this point, everything is reduced to two bits and the resultant can be calculated from a 14 bit adder.

2.8 Dadda multiplication and partial product reduction

- In this improvement in speed of multiplication of mantissa is done using Dadda multiplier there by replacing booth encoding.
- The design achieves high speed with maximum frequency of 526 MHz compared to existing floating point multipliers. The floating point multiplier is developed to handle the underflow and overflow cases.
- The significant multiplication time is reduced by using Dadda Algorithm. DADDA MULTIPLIER Dadda proposed a sequence of matrix heights that are predetermined to give the minimum number of reduction stages using 15:4 compressor and 4:2 compressor stages.

2.9 4:2 Compressor design

In the multiplication process, compressors are normally used to serve two major purposes, (i) to speed up the operation (ii) to reduce the number of stages in the partial product generation. The main goal of this exact compressor is to provide an accurate output but with high power consumption, area and delay. Basically 4:2 compressor came into existence to replace the usage of more number of full adders. The general block diagram of 4:2 compressor is shown in Fig.2. It consists of X1, X2, X3, X4 as 4 inputs with a carry in (Cin) and sum, carry as outputs with a carry out (Cout) for propagation.

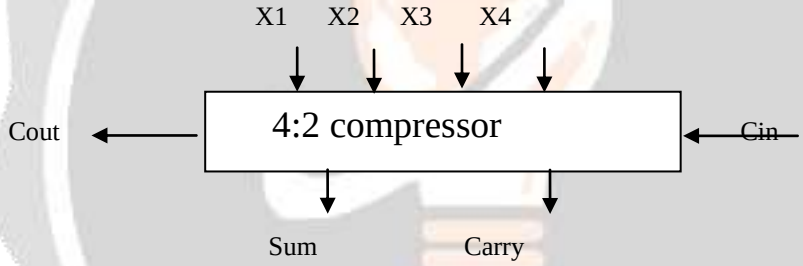

Figure 2 4:2 Compressors

The carry out (Cout) is a one bit binary number with higher significance whereas the four inputs and sum output carry the same weight. Since the carry in (Cin) bit acts as one of the inputs to the compressor, it will have lower significance while the output Carry out (Cout) comes with higher significance. There are types of implementation for the exact compressor as follows in figure 3 ,figure 4 ,figure 5 and figure 6.

- Using two Full Adders (FA).

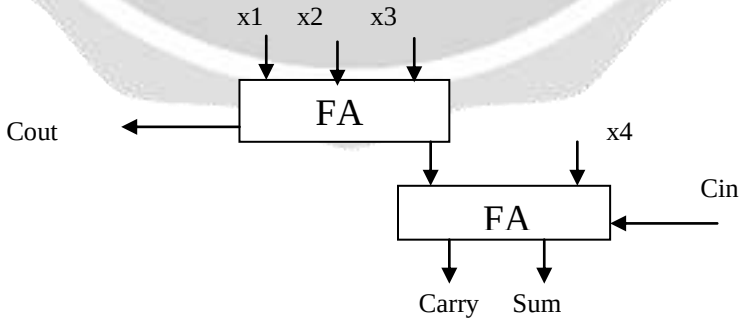

Figure 3 Full adder

- 15:4 and 5:3 Compressor

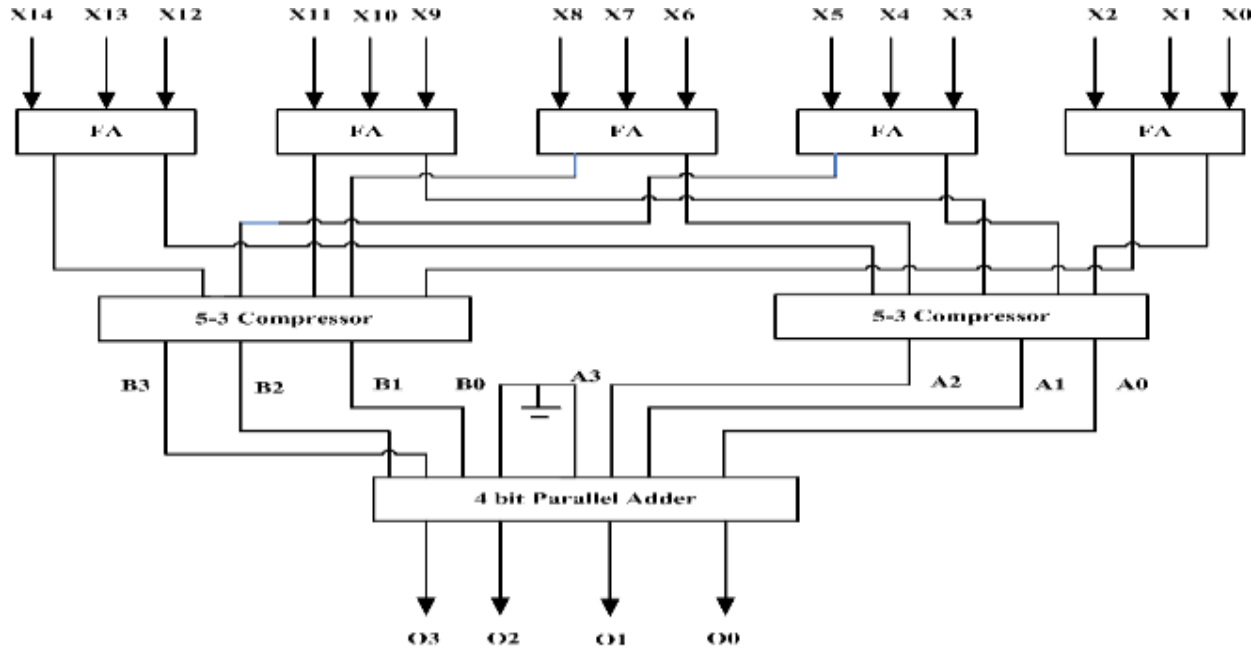

Figure 5 Reduction of partial stage using 15:4 and 5:3 compressor

- 5:3 Compressor

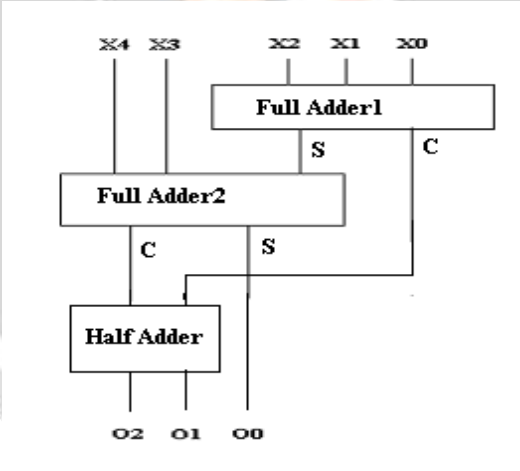

Figure 6 5:3 compressor

2.10 Fused floating-point 2-term dot product unit

The 2-term dot product computation is defined as the computation which has been widely used in complex multiplications.

$$(A.B)\pm(C.D)$$

There are various stages involved in the fused floating point 2-term dot product unit.

- First stage
- Second stage
- Third stage
- Fourth stage
- Fifth stage
- Sixth stage

3. CONCLUSIONS

Thus Radix-4 is much efficient algorithm than Radix-2 . Generic-gate level implementation of Fast Fourier Transform using existing model Booth-encoder Multiplier has been done using VLSI 90nm technology. The circuit has been studied and analyzed for data accuracy and efficient performance. An accuracy of ± 0.1% has been obtained for the twiddle factor = 0.7071. On increasing the number of significant digits of the twiddle factor, accuracy of output can be improved but it will share a trade-off with the area of the circuit. Physical level chip design and further optimization of the circuit in terms of area and power and increasing the number of points for FFT computation form the future scope of work. Thus the proposed approximate compressors with an implementation on Dadda multiplier has been discussed. The first and second multipliers have significant reduction on delay, area and power. The third and fourth multipliers have modest reduction with best accuracy It was observed that Dadda multiplier was around 14% faster and consumed 27%–45% lower power, hence it was selected to build the FPPE.The data paths are scalable and parameterizable. This was demonstrated through the implementation of a new FPPE.The generalized structure of the data paths makes them ideal implementation platforms for soft-processing-based systems.Also the power-delay product of the proposed design is significantly lower than that of the regular Wallace multiplier.Our future efforts in this area will involve integrating these data path structures into a hybrid, multigranular FPGA as well as soft-processing reconfigurable array for low-cost, high-speed multimedia processing.

4. ACKNOWLEDGEMENT

We wish to express our great deal of gratitude to our Guide, **Mrs.M.Shalini,M.E.,** for her cooperation towards us at all times of need, for her guidance and valuable suggestions in every aspects for completion of this project.

5. REFERENCES

1. R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC system/6000 floating-point execution unit," *IBM J. Res. Develop., vol. 34*,pp. 59–70, 1990.
2. E. Hokenek, R. K. Montoye, and P. W. Cook, "Second-generation RISC floating point with multiply-add fused," *IEEE J. Solid-State Circuits*,vol. 25, pp. 1207–1213, 1990.
3. T. Lang and J. D. Bruguera, "Floating-point fused multiply-add with reduced latency," *IEEE Trans Comput.,vol 53, pp. 988–1003, 2004.*
4. J. D. Bruguera and T. Lang, "Floating-point fused multiply-add: Reduced latency for floating-point addition," in *Proc. 19th IEEE Symp. Comput. Arith., 2005, pp. 42–51*.