Fused Floating Point Arithmetic Unit for Radix 2 FFT Implementation

Prasanna Palsodkar\(^1\), Ajay Gurjar\(^2\)

\(^1\)(Department of Electronics Engineering, Yeshwantrao Chavan College of Engineering, Nagpur, India)
\(^2\)(Department of Electronics & Telecomm. Engineering, Sipna’s College of Engineering & Technology, Amravati, India)

Abstract: This paper describes the design and implementation of user defined fused floating-point arithmetic operations that can be used to implement Radix 2 butterfly Fast Fourier Transform (FFT) for complex numbers used in Digital Signal Processing (DSP. C) processors. This paper reports the comparison of area, delay and power of fused floating point modules as compared to discrete floating point with reference to Radix 2 butterfly structure. The design is implemented and simulated by targeting Xilinx vertex 5 FPGA device. Here we have achieved reduction in area (in terms of LUT required) by 27.09%, reduced delay by 17.10%, reduction in power consumption by 11 % and energy is reduced by 26.22% as compared to discrete implementation.

Keywords: Floating point, DSP, Area, Power, Delay, FFT

I. INTRODUCTION

In past, many Digital Signal Processing (DSP) applications used fixed point arithmetic due to the high cost (in delay, silicon area, and power consumption) of floating-point arithmetic units [8]. Floating-point arithmetic is much useful in the implementation of various DSP applications as it allows the designer and the user to concentrate on the algorithms and architecture without worrying about the numerical issues [1, 6]. Many applications use floating-point hardware to perform DSP tasks in real time and hence overcome the limitations imposed by the use of fixed-point numeric systems. In realization of modern general purpose processors, fused floating-point multiply add module 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) [3]. This work extends the consideration of fused floating-point arithmetic operations that are frequently encountered in DSP applications [6]. The Fast Fourier Transform is a case in consideration since it uses a complex butterfly operation. For a radix-2 implementation, the butterfly consists of complex operations of multiplication, addition and subtraction of the same pair of data. This butterfly operations can be implemented with fused primitives, i.e Fused Two-Term Dot-Product Module, and Fused Add-Subtract module [1].

II. RELATED WORK

This section introduces fixed-point computer arithmetic and its limitations, the IEEE-754 floating-point standard, and current usage of combined (fused) arithmetic functions, a quick introduction to the Fast Fourier Transform (FFT), floating-point and FFT error analysis.

II.I DSP processors Arithmetic Overview

DSP processors arithmetic is concerned with the hardware realization of mathematical formulas, algorithms and complex models from a theoretical world. Hardware functions calculate arithmetic’s in both fixed-point and floating-point (Scientific Notations) [6].

II.II IEEE-754 Floating Point Standard

The IEEE Standard for Floating-Point Arithmetic (IEEE-754) is a technical standard established by the Institute of Electrical and Electronics Engineers [5]. It is the most ubiquitous standard for floating-point computations representation in today’s microprocessors, including Intel-based Processor’s, Macintoshes and UNIX platforms. IEEE floating point numbers have three basic components: a sign, an exponent and a significant. The significant is composed of the fraction and an implicit leading digit. The exponent base (2) is implicit and is not stored [4]. This standard specifies the basic types of representation like

- Half Precision (16-bits or 2-bytes)
- Single Precision (32-bits or 4-bytes)
Double Precision (64-bits or 8-bytes)

TABLE 1 shows the Bit-wise distribution of Floating point number.

<table>
<thead>
<tr>
<th>Types of Representation</th>
<th>Sign</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Precision</td>
<td>1[31]</td>
<td>8 [30-23]</td>
<td>23 [22-0]</td>
</tr>
<tr>
<td>Double Precision</td>
<td>1[63]</td>
<td>11[62-52]</td>
<td>52 [51-0]</td>
</tr>
</tbody>
</table>

II.III Overview of Floating-Point Fused Multiply-Add (FMA) operation

In 1990, IBM introduced the floating-point fused multiply-add operation on the RISC System 6000 (IBM RS/6000) chip [1]. IBM recognized that several advanced applications, specifically those with dot products are routinely performed with a floating-point multiplication immediately followed by a floating-point addition i.e. (A x B) + C, ad infinitum. To increase the performance of these applications, a new module was created that merged a discrete floating-point multiplier and floating-point adder into a single hardware block: the floating-point fused multiply-add (FMA) module. This floating-point arithmetic module executes the equation (A x B) + C in a single instruction.

With the continued demand for 3D graphics, multimedia applications and new advanced processing algorithms, IEEE has included the fused multiply-add operation into the IEEE 754-2008 standard [5]. Even though the fused multiply-add architecture has troublesome latencies, high power consumption and performance degradation with single-instruction execution, more and more microprocessor designs implement floating-point fused multiply-add module in their silicon.

II.IV Fast Fourier Transform (FFT) Algorithm

Fourier analysis is a family of mathematical techniques, based on decomposing signals into sinusoids. The Discrete Fourier Transform (DFT) is used with digitized signals [13]. The DFT of a sequence of N complex numbers is given by

\[ X_k = \sum_{n=0}^{N-1} x_n e^{2\pi i n / N} \], \quad k = 0, 1, ..., N - 1

(1)

Fast Fourier Transform (FFT) is an efficient method for calculating the DFT. Even if it produces the same result as the other approaches, it often reduces the computation time by a factor of ten or more for large sequences [14].

There are two types of the FFT algorithm: Decimation in Time (DIT) where the time domain sequence is split into even and odd parts for processing and Decimation in Frequency (DIF) where the frequency components are divided into even and odd parts for processing. Both the DIT and DIF can accept inputs either in order or in bit reversed order to produce bit reversed or in order outputs respectively.

Fig. 1 shows the radix-2 DIT FFT and DIF FFT butterflies, which are the basic computation element in performing the FFT. Fig. 2 shows the data flow diagram for performing a radix-2 DIT FFT and Fig. 3 shows the data flow diagram for performing a radix-2 DIF FFT.

The X0-X8 are the input data samples, \( W_N^k \) is the twiddle factor for butterflies which is given by equation:

\[ W_N^k = e^{-i 2\pi k / N} = \cos(2\pi K / N) - i \sin(2\pi K / N) \]  

(2)

Figure 1: Radix -2 Butterfly Structure
III. Fused Floating Point Modules

This section describes three proposed Fused Floating Point Arithmetic (FFPA) modules i.e. Fused Floating Point Add-Subtract (FFPAS) module, Fused Floating Point Multiply-Add (FFPMA) module and Fused Floating Point Two-Term Dot-Product (FP2TDP) module.

III.1 Fused Floating Point Add-Subtract (FFPAS) Module
Discrete Floating-Point Adder (FPA) and Floating-Point Subtractor (FPS) can be designed using two approaches as follows:
- Parallel implementation where two adders operate in parallel
- Serial implementation where a single adder is used twice with the same operands.

The architecture of the proposed fused floating point add-subtract (FFPAS) module is derived from the floating-point add module. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set of hardware with results shared by both add and the subtract operations. Fig. 4 shows the flowchart of Proposed FFPAS module.
Fused Floating Point Arithmetic Unit for Radix 2 FFT Implementation

Figure 4: Flowchart of Proposed Fused floating point add-subtract module (FFPAS)

```
X (31:0) Y (31:0)

Check floating point validity checker

Align & selection of bigger and smaller number

Proper addition algorithm with normalize and rounded result

Proper subtraction algorithm with normalize and rounded result

Floating point adder result (31:0)
Floating point subtract result (31:0)
```

**TABLE 2:** Testing Input for Proposed FFPAS module

<table>
<thead>
<tr>
<th>Input</th>
<th>32 Bit Floating Point</th>
<th>Equivalent Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>32b0 0111 1000 1011 1010 0000 1111 0110 110</td>
<td>1.3490607e-2</td>
</tr>
<tr>
<td>Y</td>
<td>32b0 0111 0011 0101 0000 0000 0011 1111 111</td>
<td>3.204494e-4</td>
</tr>
</tbody>
</table>

X + Y = (+1.3490607e-2)+(+3.204494e-4)= +1.381105644e-2
X – Y = (+1.3490607e-2)+(+3.204494e-4)= +1.317015756e-2

TABLE 2 indicates the testing inputs for the proposed FFPAS module and the manual calculations done for the inputs. Figure 5 shows the timing simulation of FFPAS module. From the timing simulation of FFPAS module the output sum is (X + Y) = +1.3811056e-2 and difference is (X - Y) = +1.3170158e-2, which are nearly same as that of manual calculation.

![Figure 5: Timing Simulation waveform of Proposed FFPAS module](image)

**III. II Fused Floating Point Multiply-Add (FFPMA) Module**

The architecture of the proposed Fused Floating Point multiply-add (FFPMA) module is derived from the floating-point add and multiplier module. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set of hardware with results shared by both the multiply and the add operations. Fig. 6 shows the flowchart of Proposed Fused Floating Point Multiply-Add Module (FFPMA). TABLE 3 shows the testing input for proposed FFPMA module.
TABLE 3:- Testing Input for Proposed FFPMA module

<table>
<thead>
<tr>
<th>Input</th>
<th>32 Bit Floating Point</th>
<th>Equivalent Floating Point Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>32b 0 0110_0000 1010_0000_1110_1110_0000_111</td>
<td>7.5827738e-10</td>
</tr>
<tr>
<td>Y</td>
<td>32b 0 0111_0000 0101_1111_0000_0010_0000_101</td>
<td>4.1843410e-5</td>
</tr>
<tr>
<td>Z</td>
<td>32b 0 0101_0001 0111_0001_0000_1010_0110_100</td>
<td>2.0485871e-14</td>
</tr>
</tbody>
</table>

X*Y = (+7.5827738e-10) * (+4.1843410e-5) = +3.172891131e-14
(X*Y) + Z = (+3.172891131e-14) + (+2.0485871e-14) = +5.221478231e-14

Fig. 6 Flowchart of Proposed Fused floating multiply- add module (FFPMA)

Fig. 7 shows the timing simulation of proposed FFPMA module. Also the output (Mul-Add = +5.2413355e-14) from the timing simulation and manual calculations are same.

III.Fused Floating Point Two-Term Dot Product (FFP2TDP) Module
The architecture of the proposed fused Floating Point two term dot product (FFT2TDP) module is derived from the floating-point add and multiplier module. Input A and input B multiply and input C and input D multiply after adjusting result of multiplication. The exponent difference, significant shift and exponent adjustment functions can be performed once with a single set of hardware with results shared by both the multiply and the add operations. Figure 8 shows Flowchart of Fused Floating Point two-term Dot-Product (FFP2TDP) module.
Figure 8 Flowchart of Proposed Fused floating point Two-term Dot Product (FFP2TDP) module

<table>
<thead>
<tr>
<th>Input</th>
<th>32 Bit Floating Point</th>
<th>Equivalent Floating Point Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>in A</td>
<td>32b0_0110 0000 1010 0000 1110 0000 1110 0000 1111</td>
<td>7.5827738e-10</td>
</tr>
<tr>
<td>in B</td>
<td>32b0_0111 0000 0101 1111 0000 0010 0000 0010 1001</td>
<td>4.1843410e-5</td>
</tr>
<tr>
<td>in C</td>
<td>32b0 0110 0111 1110 0000 0000 1010 0110 1000</td>
<td>1.1176817e-7</td>
</tr>
<tr>
<td>in D</td>
<td>32b0_0110 0100 1111 0000 0000 0000 0000 0000</td>
<td>1.4435499e-8</td>
</tr>
</tbody>
</table>

\[
\text{inA} \times \text{inB} = (+7.5827738 \times 10^{-10}) \times (+4.1843410 \times 10^{-5}) = +3.172891131 \times 10^{-14} \\
\text{inC} \times \text{inD} = (+1.1176817 \times 10^{-7}) \times (+1.4435499 \times 10^{-8}) = +1.613429306 \times 10^{-15} \\
\text{(inA*inB) + (inC*inD)} = (+3.172891131 \times 10^{-14}) + (+1.613429306 \times 10^{-15}) = +3.334234061 \times 10^{-14}
\]

TABLE 4 shows the testing input for the proposed fused floating point 2 term dot product module. Fig. 9 shows the timing simulation for the proposed fused floating point 2 term dot product module and the calculation from the simulations is Two Term Dot Product = +3.33423406e-14 which appears to be equal to the manual calculations.

Figure 9 Timing Simulation waveform of proposed FFP2TDP module

### IIIIV Analysis of Proposed Fused Modules

TABLE 5 shows the analysis of the proposed fused models in terms of area, delay, power and energy.

<table>
<thead>
<tr>
<th>Name of module</th>
<th>LUT (Area)</th>
<th>DSP48E</th>
<th>I/OBS</th>
<th>DELAY (ns)</th>
<th>POWER (mW)</th>
<th>ENERGY (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ud_bch_str_add_sub</td>
<td>975</td>
<td>--</td>
<td>131</td>
<td>29.02</td>
<td>719.94</td>
<td>20.89</td>
</tr>
<tr>
<td>ud_bch_str_mul_add</td>
<td>739</td>
<td>2</td>
<td>131</td>
<td>39.96</td>
<td>692.27</td>
<td>27.67</td>
</tr>
<tr>
<td>ud_bch_str_2term_dot_prod</td>
<td>926</td>
<td>4</td>
<td>163</td>
<td>39</td>
<td>715.68</td>
<td>27.91</td>
</tr>
</tbody>
</table>

### IV. Proposed Fused Floating Point Radix 2 FFT Butterfly Structure

In this section the implementation of radix-2 FFT butterfly structure is discussed. This Radix 2 butterfly structure is designed by using Proposed fused floating point add-Subtractor (FFPAS), multiply-add (FFPMA) and two-term dot-product (FFP2TDP) module as show in Fig 10. After analysis of Discrete Implementation [13] and Proposed Fused implemented modules as shown in Table 6 it is observed that discrete radix-2 butterfly structure require three Adders, three Subtractor and four Multiplier whereas
fused floating point radix-2 butterfly structure require only two Fused FP Add-Subtract and two Fused FP Two-Term Dot Product module.

<table>
<thead>
<tr>
<th>Name of Module</th>
<th>LUT (Area)</th>
<th>DSP 48E</th>
<th>IOBs</th>
<th>Delay (ns)</th>
<th>Power (mW)</th>
<th>Energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete Radix2 Butterfly Structure</td>
<td>4799</td>
<td>8</td>
<td>323</td>
<td>73.089</td>
<td>1138.37</td>
<td>83.20</td>
</tr>
<tr>
<td>Proposed Fused Radix2 Butterfly Structure</td>
<td>3499</td>
<td>8</td>
<td>323</td>
<td>60.594</td>
<td>1013.15</td>
<td>61.39</td>
</tr>
</tbody>
</table>

TABLE 6 shows the comparative analysis of proposed fused floating point radix-2 FFT butterfly structure and the discrete implementation of radix-2 butterfly structure. From the table 6, it is clear that the proposed fused FFT structure is efficient in terms of area, delay, power and energy required for processing the signals.

![Diagram of Proposed Fused Floating Point implementation of Radix 2 FFT butterfly structure](image)

Figure 10 Proposed Fused Floating Point implementation of Radix 2 FFT butterfly structure

V. Conclusion

The Prior art to realize floating-point DSP hardware falls into one of the two categories: a serial approach used for applications with low area, power and energy budgets, while for applications that need to achieve high speed processing, the parallel approach is used with large increase in the area and power consumption. The proposed fused architectures have been specifically designed to address the problems of high latency, area and power consumption for the floating-point implementation of DSP algorithms. The implementation results using industry standard process and an automatic synthesis FPGA implementation flow shows that the fused primitives are faster, smaller, uses less power and energy than discrete implementation of Radix2 butterfly and provide more accurate result. In implementation of the proposed fused modules, area required is reduced by 27.09%, delay is reduced by 17.10%, it consumes 11 % less power and requires 26.22% less energy as compared to discrete implementation of Radix 2 butterfly structure.

References

[7]. Stuart Franklin Oberman, Design Issues in High Performance Floating Point Arithmetic Units, Ph. D. Dissertation, Stanford University, 1996.

DOI: 10.9790/4200-0602015865 www.iosrjournals.org 64 | Page