Block Floating Point Implementations for DSP Computations in Reconfigurable Computing

Georgina Binoy Joseph

Associate Professor, KCG College of Technology, Chennai, India

***Abstract*—The IEEE-754 standard prescribes standards for 32 bit single precision and 64 bit double precision formats. For DSP applications that require a large dynamic range floating point implementations are more suitable than fixed point representation. This advantage is offset by the cost of the implementation. The block floating point (BFP) concept combines the precision and cost effectiveness of fixed point representations with the increased dynamic range of floating point representations. BFP is of particular importance in FPGA implementations of DSP algorithms. In this paper the embedded multipliers available in present day reconfigurable devices facilitate the implementation of efficient BFP architectures for DSP applications.**

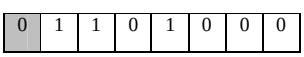
***Keywords*—block floating point; dynamic range; fixed point; reconfigurable computing; DSP applications**

I. INTRODUCTION

Reconfigurable computing using FPGAs provide a platform for efficient implementations of operations required in DSP algorithms. The bit level granularity of FPGAs permits the choice of standard and non-standard number of representations. This allows the use of just the right number of bits and the right number of operations on these bits [1]. The ratio of the largest number to the smallest number in any number representation is called the dynamic range. Certain DSP applications require an extended dynamic range which requires the use of floating point applications. Single precision and double precision representations of the IEEE-754 standard can be used to increase the dynamic range. This however comes at the cost of increased complexity in the multiply and add operations due to separate exponent and mantissa components in floating point numbers. Block Floating Point (BFP) algorithm provides a technique to reduce the complexity to that of fixed point operations. Current day FPGAs provide embedded multipliers The implementation of FFT using the concepts of block floating point is used as an illustration to highlight the advantages while performing the multiply, add and subtraction operation intensive DSP algorithms.

A. Fixed and Floating Point Processors

DSP processors use fractional fixed point representations for signal processing applications. Signed numbers between -1 and 1 can be represented with the binary point immediately following the sign bit.



Sign 2^{-1} 2^{-2} 2^{-4}

= 0.5 + 0.25 + 0.0625 = 0.8125

Fig 1. 8 bit Fixed point number

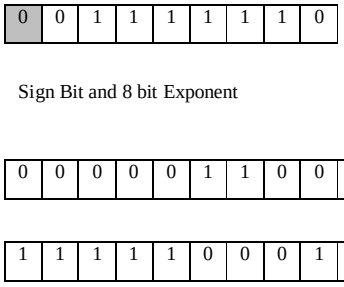


Fig 2. 32 bit Floating point number

B. Embedded Multipliers in FPGAs

DSP applications are multiplication rich and their implementation in FPGAs is facilitated by the construction of multipliers using different techniques. Xilinx provides – the following implementations for efficient, low latency and high performance multipliers – LUT based, DSP48 based or Hybrid constructions. This is supported by the Virtex-5, Virtex-6 and Spartan-6 families. These signed and unsigned fixed point multipliers can support input data widths from 2-64 bits. The embedded multipliers are optimized for 18 X 18 bit multiplications and hybrid implementations can be used for input data sizes larger than 18 bits [4].

The Altera Cyclone II FPGAs have around 150 18 X 18 multipliers. The Stratix V FPGA has a variable precision DSP block optimized for 27 X 27 bit or 18 X 36 bit multiplication. The availability of these multipliers combined with the flexibility of FPGAs makes them suitable for multiply rich DSP applications like the Fast Fourier Transform.

II. BLOCK FLOATING POINT

Fixed point representations have a higher precision than floating point representations of the same word length. This is because precision which is given by the size of the LSB of the fraction depends upon the word length for fixed point numbers and on length of mantissa for floating point numbers. However, for the same word length, floating point numbers have greater dynamic range than fixed point numbers. In addition,

processors using floating point numbers automatically scale the numbers to use the full range of the mantissa. DSP applications requiring large dynamic range would benefit from the use of floating point numbers. This however comes at increased cost of complexity of the hardware required as well as increased power dissipation.

Block Floating Point provides a technique to combine the advantages of fixed point and floating point representations. Block floating point was initially proposed for software emulations of floating point operations but is now being used in FPGAs. Arithmetic efficiency involves tailoring an operator to the right size required for a particular application [1]. FPGAs are ideal platforms to tailor operations and number representations and operations to the particular application.

A. *Concepts of Block Floating Point*

In block floating point all numbers in the representation share a common exponent value. The difference between the numbers lies only in the mantissa. Hence the exponent can be stored separately and all the operations can be considered as being performed on the mantissa components. The operations are then similar to those performed by fixed point processors on fixed point numbers. This considerably simplifies the hardware required and also leads to reduction in power consumption [2].

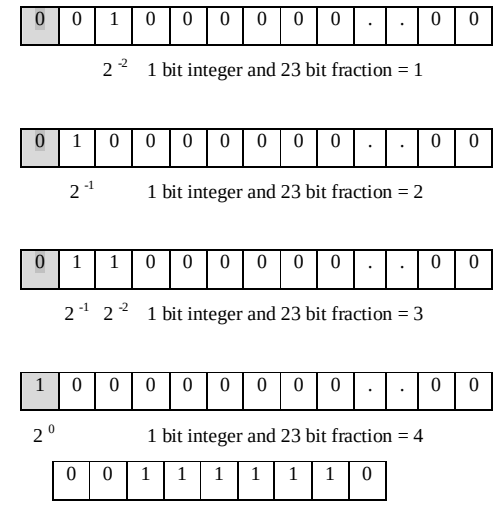


Fig 4. Block Floating Point Representation

The value of the exponent used as the common exponent is that of the largest in the block of numbers under consideration. The exponents of the numbers in the final results can then be adjusted so that the dynamic range is completely utilized. The scaling of the numbers by shifting them so as to obtain a common exponent increases the dynamic range and computations on the mantissa emulate the precision of fixed point numbers.

B. *Radix 2 FFT*

Radix 2 FFT is more suitable for maintaining precision levels and therefore two Radix 2 stages are preferred over a single Radix 4 stage. Figure 3 shows the butterfly diagram of a single stage of Radix 2 Decimation in Time (DIT) FFT.

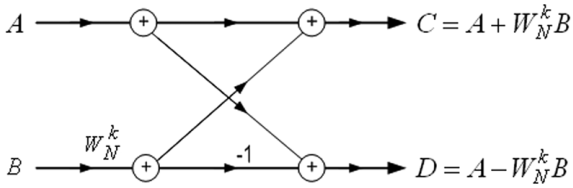


Fig 3. Radix 2 DIT FFT Butterfly Stage

The input, output and twiddle factor values can be complex valued. The real and imaginary parts of the input and twiddle factor must be considered separately while determining bit growth.

$$C = (A_r + jA_i) + (W^n_{Nr} + j\,W^n_{Ni})\,(B_r + jB_i)$$
$$= (A_r + W^n_{Nr}\,B_r - W^n_{Ni}B_i) + j(A_i + W^n_{Nr}\,B_i + W^n_{Ni}B_r) \quad (1)$$
$$D = (A_r + jA_i) - (W^n_{Nr} + j\,W^n_{Ni})\,(B_r + jB_i)$$
$$= (A_r - W^n_{Nr}\,B_r + W^n_{Ni}B_i) + j(A_i - W^n_{Nr}\,B_i - W^n_{Ni}B_r) \quad (2)$$

The Figure 4 below shows an 8 point DIT FFT which will be used as the basic DSP computation to illustrate the block floating point algorithm implementation using FPGAs.

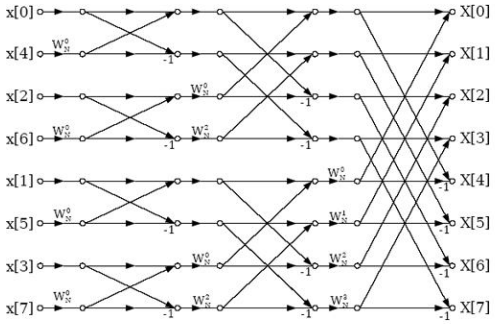


Fig 5. 8 Point DIT FFT

C. *Block Floating Point Algorithm*

Block AGC (Automatic Gain Control) is a technique that scales the values at the input stage of the DSP application like the Fast Fourier Transform (FFT). This scaling is done to increase the precision while at the same time providing an extended dynamic range. This concept is extended to the inputs of each stage of the FFT to predict the bit growth at each stage. This predicted bit growth is used to scale the inputs so that the full dynamic range can be used.

For an N point FFT where N is a power of 2, the twiddle factors of the first stage, W^0_N and W^N_N have a value 1 + j0. Equations (1) and (2) show that in this case the maximum bit growth in this stage is 2 as the inputs A and B have real and imaginary values less than 1.

The second stage twiddle factors are W^0_N, $W^{N/4}_N$ and W^N_N which have values 1 + j0, 0 – j, 1 + j0 respectively. This again leads to a maximum growth factor for the second stage of 2.

In the rest of the stages the twiddle factors can have both real and complex values, resulting in growth factors of more than 2. The maximum growth factor is determined by finding the maximum magnitudes of C and D. Equations (1) and (2) can be rewritten as follows:

$$C = (A_r + B_r\cos\varphi - B_i\sin\varphi) + j(A_i + B_i\cos\varphi + B_r\sin\varphi) \quad (3)$$

$$D = (A_r - B_r\cos\varphi + B_i\sin\varphi) + j(A_i - B_i\cos\varphi - B_r\sin\varphi) \quad (4)$$

The maximum values for C and D are obtained when the real and imaginary components of A and B are at their maximum values of 1 or -1. In addition, the three components in the brackets in equations (3) and (4) have to be of the same sign and φ has a value that leads to a maximum.

To determine the value of φ the derivative of (1 ± cosφ ± sinφ) with respect to φ is set equal to 0, yielding values of

$$\varphi = \Pi/4 + n\,\Pi/2,\ n = 0,1,2,\ldots\infty$$

The maximum growth factor is determined to be 2.4142 bits (≈2bits).

This analysis shows that to maintain precision by preventing overflow, the input values to the first two stages have to be scaled to allow a maximum bit growth of 1 and the rest of the stages have to be scaled to allow a maximum bit growth of 2. At each stage the maximum bit growth in the real and imaginary parts of the output is determined and the required scaling is applied if necessary. If there is no bit growth no scaling is done. This allows maximizing of the dynamic range while preventing overflow [2].

III. Implementation and Simulation

The 8 point DIT FFT implementation was used to illustrate the use of block floating point arithmetic in DSP computations. Three hardware based designs for the Fast Fourier Transform were implemented on the Virtex6 FPGA.

The first design implemented a fixed point FFT computation using 32 bit fixed point numbers. The format of the fixed point number with one sign bit, one integer bit and 30 fractional bits limits the range of numbers from -2 to +2. Fixed point arithmetic adders and multipliers exploiting the DSP blocks of the Virtex6 FPGA have been designed for performing the required fixed point computations.

The second design is implemented using 32 bit IEEE 754 single precision floating point numbers having one sign bit, eight exponent bits, 23 fractional bits for the mantissa and an implied bit of value 1 for the integer part of the mantissa. This gives a larger dynamic range of - 2^{-126} to $(2-2^{-23})x2^{127}$ for normalized numbers. 32 bit floating point adder and multiplier modules have been used for this implementation.

In the third design, the block floating point concept has been implemented combining the increased dynamic range of floating point representations and the reduced area requirement of fixed point representations. The inputs are represented as 32 bit single precision floating point numbers. An align operation is used to align the smaller exponents so that all numbers have a common exponent which is the largest among the block of inputs. The exponents are not considered further in the FFT computations.

The de normalized mantissa of the numbers are used as inputs to the FFT computation which is now performed on 25 bit numbers that includes a 1 bit sign, 1 bit integer part and 23 bit fractional part. The arithmetic units implementing addition and multiplication perform 25 bit fixed point computation. A constant scaling of 1 bit is performed on the outputs of the first stage before applying them to the second stage. A similar scaling of 2 bits is performed on the outputs of the second stage to prevent overflow.

A final stage combines the fixed point FFT block outputs and the exponent. It performs a normalizing operation so that the outputs are in the IEEE 754 format for single precision numbers.

A. *Simulation and Implementation Results*

The results of the FFT computation of the three architectures are identical due to the use of scaling to prevent overflow in the fixed point and block floating point architectures.

A comparison of the hardware complexity of the architectures is given in the table below:

TABLE I. RESOURCE UTILIZATION

Design	Resource Utilization (Target Device xc6vlx75t-3ff484)					
	No of Slice LUTs		No of Slices		No of DSP48E1s	
	Used	Utilization	Used	Utilization	Used	Utilization
32bit Fixed Point	13243	28 %	4396	37 %	126	43 %
Single Precision Floating Point	44986	96 %	11450	98 %	40	13 %
Block Floating Point	9652	20 %	2766	23 %	48	16 %

As operations are performed on 32 bit fixed point operands, the first design shows increased resource utilization over the Block floating point design in which computations are performed on 23 bit operands. The precision is more for 32 bit fixed point when compared with 23 bit operands. However, as the design uses floating point numbers the dynamic range is more. The use of scaling to prevent overflow preserves accuracy in both architectures.

A comparison of the single precision floating point architecture and the block floating point design reveals a large resource requirement for the design that does not use the block floating point concept. This can be attributed to the fact that use of single precision floating modules for operation of addition, subtraction and multiplication requires alignment of the exponents of the operands and corresponding shifting of the mantissa prior to the computation. Once the actual computation has been performed the output of each computation has to be normalized to the IEEE 754 single precision format.

This is the major advantage of the Block floating point architecture as the alignment is performed only once for all numbers to align the exponents to a common exponent at the onset of the FFT computation. The rest of the computations are performed on the mantissa of each number. Only a single stage of normalization at the end of the FFT computation has to be carried out to normalize the final FFT outputs. This has resulted in a drastic reduction in resource utilization while maintaining the dynamic range.

Table II below compares the power-delay product of the three architectures.

TABLE II. POWER-DELAY PRODUCT

Design	Delay (ns)	Power (mW)	Power-Delay Product
32bit Fixed Point	48.759	1156.21	56,375.643
Single Precision Floating Point	166.519	1065.87	177,487.607
Block Floating Point	37.189	1279.44	47,581.094

The performance of the Block floating point architecture is better than the other two architectures in terms of the delay of the critical path. The power consumption of the 32 bit fixed point architecture is the least of the three designs. The power-delay product of the Block floating point is significantly lower than the other two designs.

IV. CONCLUSION

The implementation results of the 8 point Radix 2 DIT FFT are used to highlight the application of the Block floating Point (BFP) concept to computation of DSP algorithms on FPGAs. Effort has also been taken to ensure the design utilizes the embedded DSP resources like multipliers in present day reconfigurable devices. The Block Floating Point architecture requires significantly fewer resources and is thus cost effective when compared to the designs implementing fixed point as well as floating point computations. It also shows improvement in the power-delay product. The BFP implementation can be further improved by providing dynamic scaling between stages rather than the static one used in this design. Also, reconfigurable computing, particularly for real time applications does not need to adhere to the standard formats for operand representation. Operand size can be chosen as per the requirements of the application which will further lead to more efficient implementations.

REFERENCES

[1] Florent de Dinechin and Bogdan Pasca, “High-Performance Computing using FPGAs,” Chapter, Reconfigurable Arithmetic for High Performance Computing, Springer, 2013.

[2] David Elam and Cesar Iovescu, “A Block Floating Point Implementation for an N-Point FFT on the TMS320C55x DSP,” TMS320C5000 Software Applications, Application Report SPRA948 – September 2003

[3] Shiro Kobayashi and Gerhard P. Fettweis, “A new approach for block-floating-point arithmetic,” in proceedings of IEEE Conference on Accoustic, Speech, and Signal processing, 1999, pp. 2009-2012.

[4] Xilinx Product Specification, “Virtex-6 Family Overview,” DS150 (v2.4) January 19, 2012

[5] Nicolas Brisebarre, Florent de Dinechin, Jean-Michel Muller, “Integer and Floating-Point Constant Multipliers for FPGAs,” International Conference on Application-Specific Systems, Architectures and Processors, 2008, pp. 239 - 244.