Journal of VLSI Design Tools & Technology
Volume 1, Issue 1-3 Compilation 2011

STM JOURNALS

Analysis of Resource Utilization for a Floating-Point Complex Multiplication in FPGA

Anitha Mary. X[1], Dojin Domnic[2] & K. Rajasekaran[3]*
[1]Assistant Professor, [2]PG Scholar, [3]Head of the Department
Department of Electronics and Instrumentation, Karunya University, Coimbatore, India

ABSTRACT

Complex multiplication is an important operation frequently used in the digital signal processing. This paper shows the resource utilization for a complex multiplication on Spartan3E by using a 9 bit floating point numbers. The proposed method clearly shows that resource utilization for complex multiplication is less than the IP cores proposed by the Xilinx Company. More over this paper shows how the significant and the exponent of the floating point number effect the resource utilization for a complex multiplication.

Keywords: *Complex multiplication, Floating point, IP cores*

Author for Correspondence E-mail: anithajohnson2003@gmail.com

INTRODUCTION

Digital Signal Processing (DSP) is the widely used technology in engineering discipline. It is a sector which grows very fast bringing tremendous challenges to the engineering community. All the computations in the DSP domain are additions and multiplications. Faster additions and multiplications are of great importance in this domain. In the case of floating point numbers the additions is much complex than multiplication and therefore to improve the efficiency and resource utilization, a newer algorithms are necessary. The number which is understandable by the computer is binary and it is limited by the word size. It is very difficult to represent very large numbers with the help of fixed point numbers. This problem can be resolved by using floating point numbers because a large or a very small number can be represented by fewer number of bits as compared to the fixed point numbers. But the floating point numbers take more resources and computation time as compared to fixed point numbers [2]. In a complex multiplication it requires 4 multiplication and 2 additions to be performed, it is given by the equation below

$$(a+bj)(c+dj)= (ac-bd) + (ad+bc)j$$

The proposed method uses efficient use of VHDL for targeting FPGAs in order to match constraints in area and timing issues [1]. The method is implemented in SPARTAN 3E. Section II deals with the floating point numbers, section III deals with the resource utilization by Xilinx IP cores, section IV deals with the resource utilization by the proposed method and section V shows the comparative result.

FLOATING-POINT NUMBERS

Floating point number divides the number into significant part and exponent part. The small

numbers are represented by the significant part which decides the precision of the floating point numbers. The range is decided by the exponent part. Larger the exponent, wider will be the range. In 1985, the Institute of Electrical and Electronic Engineers published IEEE Standard 754 for floating-point arithmetic [5].

All the processors built today uses IEEE standard 754. The IEEE Standard 754 consists of one sign bit, eight exponent bit and 23 significant bit. A binary fraction is being used in IEEE 754 and exponent is considered to be a power of two. The format of a single-precision floating-point number is shown in Figure 1. The leftmost bit indicates the sign of the number, with a zero indicating positive and a one indicating negative. The exponent occupies eight bits and is also signed. A negative exponent indicates that the fraction is multiplied by a negative power of two. The exponent is stored as an excess 127 number, which means that the value stored is 127 more than the true value. A stored value of one indicates a true value of -126. The fraction part is a 23-bit binary fraction with the binary point assumed to be to the left of the first bit of the fraction. The approximate range of such a number is given by ± 10^-38 to ±10^38.

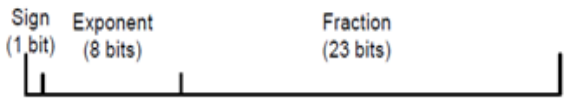

Fig. 1 Format of an IEEE 754 Single-Precision Format

USING IP CORES

Using IP cores given in the Xilinx 10.1 for floating point arithmetic the 9 bit floating point number is given by 1 sign bit, 5 exponent and 3 significant. The IP core is used for converting the fixed point to floating point and performs floating point multiplication and addition and also converting floating point back to the fixed point. The above mentioned IP cores are being used and the resource utilized is given in Table I.

Table I Resource Utilization using IP Cores

```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s100etq144-5

 Number of Slices:                      844  out of    960    87%
 Number of Slice Flip Flops:           1148  out of   1920    59%
 Number of 4 input LUTs:               1098  out of   1920    57%
    Number used as logic:              1026
    Number used as Shift registers:      72
 Number of IOs:                          55
 Number of bonded IOBs:                  55  out of    108    50%
 Number of GCLKs:                         1  out of     24     4%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------
```

PROPOSED METHOD

In this method we are dealing with 9 bit floating point number with 1 sign bit, 5 exponent and 3 significant.

Fixed to floating point Conversion

First the 9 bit fixed point number is first converted into the 9 bit floating point number and the algorithm used is given below.

1. Shift the integer part until the last '1' remains LSB of the register.
2. If right shifting is done then

Exponent = bias + (no. of shifting operations)

3. If left shifting is done then

Exponent = bias – (no. of shifting operations)

4. Shifted out numbers forms the significant part. [4]

Floating point Multiplication

The floating point multiplication is performed using the following algorithm.

A binary floating-point number x is represented as a significant and an exponent, x = s .2^e. The formula

(s1 × 2^e1) • (s2 × 2^e2) = (s1 • s2) × 2^ (e1+e2)

shows that a floating-point multiply algorithm has several parts. Significant multiplication is done by using radix 2 multiplication algorithm which is given by,

The simplest multiplier computes the product of two unsigned numbers, one bit at a time, as illustrated in Figure 2. The numbers to be multiplied are an–1an–2…..a0 and bn–1bn–2 ……b0, and they are placed in registers A and B, respectively. Register P is initially 0. Each multiply step has two parts.

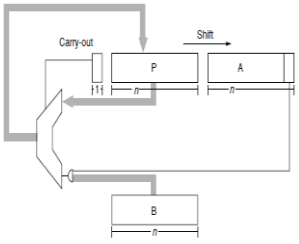

Fig. 2 Radix 2 Multiplication Algorithm.

1. If the least-significant bit of A is 1, then register B, containing bn–1bn–2 .....b0, is added to P; otherwise 00 …. 00 is added to P. The sum is placed back into P.
2. Registers P and A are shifted right, with the carry-out of the sum being moved into the high-order bit of P, the low-order bit of P being moved into register A, and the rightmost bit of A, which is not used in the rest of the algorithm, being shifted out. After n steps, the product appears in registers P and A, with A holding the lower-order bits.

The next step is the rounding .If p is the number of bits in the significant, and then the A, B, and P registers should be p bits wide. Multiply the two significant to obtain a 2p-bit product in the (P, A) registers as shown in the Figure 3.

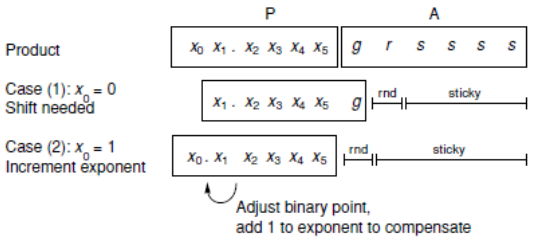

Fig. 3 Rounding Techniques

During the multiplication, the first p -2 times a bit is shifted into the A register, OR it into the sticky bit. This will be used in halfway cases. Let s represent the sticky bit, g (for guard) the most-significant bit of A and r (for round) the second most-significant bit of A. There are two cases:1. The high-order bit of P is 0. Shift

left 1 bit, shifting in the g bit from A. Shifting the rest of A is not necessary.

2. The high-order bit of P is 1. Set s: = s v r and r: = g, and add 1 to the exponent. [3]

Floating Point Addition Algorithm

To find the sign bit:

Check whether both the numbers are having the same sign bit, if so then the answer also has the same. If the sign are different first the exponents are compared and bigger exponents sign is fixed, if exponents (e) are the same then bigger significant holding number sign is taken.

To find the exponent and significant part:

1. If e1<e2, swap the operands.
2. If signs are different significant 2s' complement is taken.
3. Significant is shifted right with '1' accordingly to make e1=e2 and e1 is assigned as exponent if signs are different.
4. Significant is shifted with '0' accordingly to make e1=e2 and e1 is pushed to exponent out if signs are same.
5. Compute preliminary significant 'SIG'=s1+s2
6. If sign are same and carry is present in the above addition carry is taken in and LSB is removed and e1 is added by '1'.
7. If carry is not present we need to left shift the SIG to have a '1' in the MSB.
8. For every left shift we need to decrement the exponent.
9. In step '5' and '6' we have right shifted the significant, those shifted out bits are XORed together and if '1' is obtained then SIG is incremented.

Float point to fixed point Conversion

Finally the result obtained in floating point is converted to fixed point and the algorithm performed is given by

1. Exponent will be subtracted by the bias.
2. If difference is positive then left shift with an added '1' as the msb..
3. If difference is negative take the 2's complement and right shift with a '1'.

Resource utilization

The resource utilized for the above mentioned complex multiplication which consisting of one sign bit 5 exponent and 3 significant is given in table II. It shows that the number of slices and LUTs used are less when compared to IP cores.

Table II Resource Utilization using Proposed Method with 5 EXPONENt and 3 Significant.

```
--------------------------------------------------------------------------------

Device utilization summary:
---------------------------

Selected Device : 3s100etq144-5

 Number of Slices:                      275  out of    960    28%
 Number of 4 input LUTs:                493  out of   1920    25%
 Number of IOs:                          54
 Number of bonded IOBs:                  54  out of    108    50%
    IOB Flip Flops:                      16

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.
```

To understand how the significant and the exponent of the floating point number effect the resource utilization for a complex multiplication**.** The complex multiplication is performed on a 9 bit floating point number which consists of one sign bit, 4 exponent bit and 4 significant bit.

Here we can see that the range of the floating point number has came down from -122880 - +122880 to -496 to +496. But we can see that the accuracy has been increased in a great way. The resource utilization is given in table III.

Table III Resource utilization using proposed method with 4 exponent and 4 significant.

```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s100etq144-5

 Number of Slices:                      307  out of    960    31%
 Number of 4 input LUTs:                543  out of   1920    28%
 Number of IOs:                          54
 Number of bonded IOBs:                  54  out of    108    50%
    IOB Flip Flops:                      16

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------
```

COMPARITIVE RESULTS

Table IV depicts that the proposed method is more efficient than the IP cores provided by the Xilinx Company. We also see that the time required to compute the floating point arithmetic with the help of Xilinx IP core takes about 25 clocks where as the proposed method could give the results in 5 clocks. Also we find that the number of resource utilization increases with the significant number.

Table IV Comparative Analysis of Floating Point numbers

Methods	Slice Used (%)	Range	Bits	Clk req
Floating point number with 1 sign bit, 5 exponent and 3 significant by using IP core	87	-122880 to +122880	9	25
Floating point number with 1 sign bit, 5 exponent and 3 significant	28	-122880 to +122880	9	5
Floating point number with 1 sign bit, 4 exponent and 4 significant	31	-496 to 496	9	5

CONCLUSION

In DSP applications, floating point numbers plays an important role especially for addition

and multiplication. It is found that the proposed method is highly efficient than IP cores proposed by Xilinx. The clock frequency of SPARTAN 3E is 20 MHz. The time required to implement the proposed method is 0.25µs (5x0.05µs) whereas the time taken by IP cores is 1.25µs (25x0.05µs). We could also make a conclusion that as the number of the significant number get increased in the floating point number the resource utilization also get increased.

REFERENCES

1. Iakovos Stamoulis, Nicky Ford, Martin White et al. VHDL Methodologies for effective implementation in FPGA devices and subsequent transition to ASIC technology Designer Track DATE '98, Paris. February 23–26, 1998.
2. Bob Brown Floating Point Numbers Computer Science Department Southern Polytechnic State University, http://www.spsu.edu/cs/faculty/bbrown/papers/floating.pdf
3. David Goldberg Computer arithmetic Appendix H, Xerox Palo Alto Research Center Elsevier Science. 2003.

