Low Latency, Area Optimized, High Throughput Double Precision Pipelined Floating Point Multiplier Using VHDL on FPGA

Tushar S. Muratkar

Electronics & Telecommunication Department, JDCOEM, Nagpur tushar_16m@yahoo.co.in

I. INTRODUCTION

Field-programmable gate arrays (FPGAs) have long been attractive for accelerating fixed-point applications. Early on, FPGAs could deliver tens of narrow, low latency fixed-point operations. As FPGAs matured, the amount of parallelism to be exploited grew rapidly with FPGA size. This was a boon to many application designers as it enabled them to capture more of the application. It also meant that the performance of FPGAs was growing faster than that of CPUs [7]. Every computer has a floating point processor or a dedicated accelerator that fulfils the requirements of precision using detailed floating point arithmetic. The main applications of floating point operations that operate on these values. It also specifies four rounding modes and five exceptions. IEEE 754 specifies four formats for representing floating-point values: single precision (32-bit, usually implemented with 80 bits), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits). Many languages specify that IEEE formats and arithmetic be implemented, although sometimes it is optional. For example, the C programming language, which pre-dated IEEE 754, now allows but does not require IEEE arithmetic (the C float typically is used for IEEE single-precision and double uses IEEE double-precision).

A. Floating Point Arithmetic

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) is the most widely used standard for floating-point computation, and is followed by many CPU and FPU implementations. The standard defines formats for representing floating-point number (including ±zero and denormals) and special values (infinities and NaNs) together with a set of floating-point operations that operate on these values. It also specifies four rounding modes and five exceptions. IEEE 754 specifies four formats for representing floating-point values: single precision (32-bit), double-precision (64-bit), single-extended precision (≥ 43-bit, not commonly used) and double-extended precision (≥ 79-bit, usually implemented with 80 bits). Many languages specify that IEEE formats and arithmetic be implemented, although sometimes it is optional. For example, the C programming language, which pre-dated IEEE 754, now allows but does not require IEEE arithmetic (the C float typically is used for IEEE single-precision and double uses IEEE double-precision).

B. Double Precision Floating Point Numbers

Thus, a total of 64 bits is needed for double-precision number representation. To achieve a bias equal to $2^{63}$, 1 is added to the actual exponent in order to obtain the stored exponent. This equal 1023 for an 11-bit exponent of the double precision format.
The addition of bias allows the use of an exponent in the range from \(-1023\) to \(+1024\), corresponding to a range of \(0.2047\) for double precision number. The double precision format offers a range from \(2^{-1023}\) to \(2^{+1023}\), which is equivalent to \(10^{-308}\) to \(10^{+308}\).

<table>
<thead>
<tr>
<th>SIGN</th>
<th>EXPONENT</th>
<th>FRACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Bit</td>
<td>11 Bits</td>
<td>52 Bits</td>
</tr>
</tbody>
</table>

Fig 1. Double precision Floating point format

C. Floating-Point Multiplication

Multiplication of two floating point normalized numbers is performed by multiplying the fractional components, adding the exponents, and an exclusive or operation of the sign fields of both of the operands. The most complicated part is performing the integer-like multiplication on the fraction fields. Essentially, the multiplication is done in two steps, partial product generation and partial product addition. For double precision operands (53-bit fraction fields), a total of 53 53-bit partial products are generated.

The general form of the representation of floating point is:

\[ (-1)^S \cdot M \cdot 2^E \]

Where

- \(S\) represents the sign bit,
- \(M\) represents the mantissa,
- \(E\) represents the exponent.

Given two FP numbers \(n1\) and \(n2\), the product of both, denoted as \(n\), can be expressed as:

\[
\begin{align*}
  n &= n1 \times n2 \\
  &= (-1)^{S1} \cdot p1 \cdot 2^{E1} \times (-1)^{S2} \cdot p2 \cdot 2^{E2} \\
  &= (-1)^{S1+S2} \cdot (p1 \cdot p2) \cdot 2^{E1+E2}
\end{align*}
\]

In order to perform floating-point multiplication, a simple algorithm is realized:

- Add the exponents and subtract 1023.
- Multiply the mantissas and determine the sign of the result.
- Normalize the resulting value, if necessary.

II. LITERATURE SURVEY

A few research works have been conducted to explain the concept of Floating Point Numbers. D. Goldberg [1] explained the concept of Floating Point Numbers used to describe very small to very large numbers with a varying level of precision. They are comprised of three fields, a sign, a fraction, and an exponent field. B. Parhami [2] proposed IEEE-754 standard defining several floating point number formats and the size of the fields that comprise them. This Standard defines several rounding schemes, which include round to zero, round to infinity, round to negative infinity, and round to nearest. Michael L. Overton [3] performed the multiplication of two floating point normalized numbers by multiplying the fractional components, adding the exponents, and an Exclusive OR operation of the sign fields of both of the operands. Cho, J. Hong et al. and N. Besli et al.[4][5] multiplied double precision operands (53-bit fraction fields), in which a total of 53 53-bit partial products are generated. To speed up this process, the two obvious solutions are to generate fewer partial products and to sum them faster. Sumit Vaidya et al.[6] compared the different multipliers on the basis of power, speed, delay, and area to get the efficient multiplier. It can be concluded that array Multiplier requires more power consumption and gives optimum number of components required.

III. METHODOLOGY AND THE PIPELINED FLOATING POINT MODULE

There are several techniques that can be used to perform multiplication. In general, the choice is based upon factors such as latency, throughput, area, and design complexity. Thus I had used Array Multiplier for implementing the multiplier.

Array multiplier is an efficient layout of a combinational multiplier. Multiplication of two binary number can be obtained with one micro-operation by using a combinational circuit that forms the product bit all at once thus making it a fast way of multiplying two numbers since only delay is the time for the signals to propagate through the gates that forms the multiplication array.

Fig 2. Pipelined Floating point multiplier

The details of each block is as given below:

1) Check Zero Modules:

Here both operands are checked to determine whether they contain a zero. If one of them is zero, zero _flag is set to zero. If none of them are zero, then inputs in IEEE 754 format is unpacked and

Jhulelal Institute Of Technology, Lonara, Nagpur 40 | P a g e
assigned to the check sign, add exponent, and multiply mantissa module.

2) Add Exponent Module:
This module is activated if both the operands are non-zero. Two extra bits are also added to indicate the overflow and underflow conditions. The resulting sum has a double bias, so the extra bias is subtracted from the exponent sum. After this, Exp_Flag is set to 1.

3) Multiply Mantissa module:
Here zero_flag is checked first. If zero_flag is set to zero the no calculation and normalization is performed. The Mantissa_Flag is set to zero. If both the operands are not zero then operation is done with multiplication operator. Mantissa_Flag is set to 1, indicating that the operation is executed.

4) Check Sign Module:
It determines the sign of the two operands. The resultant sign is positive if both the operands have same sign else it is negative. For this XOR circuit is used.

5) Normalize and concatenate all modules:
It checks the overflow and underflow after adding the exponent. Overflow occurs if 8th bit is 1, underflow occurs if 9th bit is 1. If Exp_Flag, Mantissa_Flag, Sign_Flag are set, then normalization is carried out. Lastly all are concatenated and are normalized.

IV. IMPLEMENTATION
The black box view of the double precision floating point multiplier is shown in figure 3. The Multiplier receives two 64-bit floating point numbers. First these numbers are unpacked by separating the numbers into sign, exponent, and mantissa bits. The sign logic is a simple XOR. The exponents of the two numbers are added and then subtracted with a bias number i.e., 1023. Mantissa multiplier block performs multiplication operation. After this the output of mantissa division is normalized, i.e., if the MSB of the result obtained is not 1, then it is left shifted to make the MSB 1. If changes are made by shifting then corresponding changes has to be made in exponent also.

V. RESULTS
The double precision floating point multiplier designs were simulated in Modelsim 6.6c and synthesized using Xilinx ISE 13.1i which are mapped on to Virtex-6 FPGA. The simulation results of 64-bit floating point double precision multiplier are shown in Figure 4 below. The 'opa' and 'opb' are the inputs and sign, exponent, product are the parts of output. Table 1 gives the comparison of device utilization between my work and [11].

![Fig 3. Black box view of floating point double precision multiplier](image)

![Fig 4. Simulation results of double precision floating point multiplier](image)

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>Device utilization summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Logic Utilization</td>
<td>Present work</td>
</tr>
<tr>
<td>Number of Slice Registers (Flip-Flops)</td>
<td>952</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>1758</td>
</tr>
</tbody>
</table>
VI. CONCLUSIONS

The double precision pipelined floating point multiplier supports the IEEE-754 binary interchange format, targeted on a Xilinx Virtex-6 xc6vlx75t-3ff484 FPGA. The designs achieved the operating frequency of 306.937 MHz which boost the performance to a great extent.

REFERENCES


