Design and Implementation of Double Precision Floating Point Comparator

Rony Antony Pa, Anjana Mary Josephb*

aAssistant Professor, Dept of ECE, RSET, Ernakulam, 682039, India
bStudent, RSET, Ernakulam, 682039, India

Abstract

Floating point comparison is a fundamental arithmetic operation in DSP processor. The high dynamic range of floating point comparators find wide applications in sorting data problem, DSP algorithms etc. High performance with optimum area is a major concern for the practical implementation of these comparators. Another major concern with respect to the floating point numbers is the invalid numbers. Thus a separate module is required to handle the invalid numbers. In the present work, a double precision floating point comparator design is proposed for efficient floating point comparison. This comparator takes full advantage of the parallel prefix tree architecture. It first compares the most significant bit and proceeds towards least significant bit only when the compared bits are equal. Representation of floating point numbers is based on IEEE 754 standard. The double precision floating point comparator is modelled using Verilog HDL and synthesized in Xilinx ISE 14.6 targeting Virtex 5 and Cadence encounter tool. The results show that the new comparator architecture is efficient in handling all the invalid floating point numbers.

Keywords: Floating point comparator; Double precision; Parallel prefix tree;

1. Introduction

Comparator is an important data path element in general purpose and application specific architectures. In computing, floating point is the formulaic representation. It approximates a real number so as to support a trade-off

* Corresponding author. Tel.:+91-894-311-9202
E-mail address: anjanamary123@gmail.com
between range and precision. The comparison of floating-point numbers is commonly required in scientific calculation applications such as a math coprocessor, embedded arithmetic coprocessor, data processor, data controller, and DSP algorithms [8]. A floating-point system can be used to represent, with a fixed number of digits, numbers of different orders of magnitude: for example, the distance between galaxies or the diameter of an atomic nucleus can be expressed with the same unit of length.

In a fixed-point number representation, the radix point is always at the same location. While the convention simplifies numeric operations and conserves memory, it places a limit on the magnitude and precision of the number representation. In situations that require a large range of numbers or high resolution, a relocatable radix point is desirable. The advantage of floating-point numbers is that they can represent a much larger range of values. Very large and very small numbers can be represented in a floating-point format. The name floating point comes from the fact that the radix point is floating.

The IEEE has standardized the computer representation for binary floating-point numbers in IEEE 754. This standard is followed by almost all modern machines. Floating-point numbers are typically packed into a computer datum as the sign bit, the exponent field, and the significand or mantissa, from left to right. The standard provides for many closely related formats, differing in only a few details. Some of the formats are half precision, single precision, double precision and double extended precision.

This paper presents a novel architecture for IEEE 754 compliant floating point comparator. The comparator takes the advantage of parallel prefix tree structure. A double precision comparator is designed which compares the input operands and produce four outputs: less than, greater than, equal and unordered.

The paper is organized as follows: In section 2, we briefly review the basic concepts of floating point numbers and previous architectures of comparator. In section 3, we give a brief description of the IEEE 754 format. In section 4, we describe the double precision floating point comparator, its architecture and block diagram. Section 5 deals with the results and discussion. Finally the paper is concluded in section 6.

2. Previous work

A comparator has always been an important block in an arithmetic logic unit and also has extensive applications in many digital systems [1]. Floating point comparisons are specified by the IEEE 754 standard. Floating-point comparisons are implemented using one of two methods, both of which are addressed by the IEEE 754 standard [2], [7]. In the first method, the comparator indicates one of four mutually exclusive relations between the two input operands: Greater Than, Less Than, Equal, and Unordered. The last case, Unordered, arises when at least one operand is a Not-a-Number (NaN). In the second method, the comparator returns a true-false condition code based on a specific comparison condition (e.g., \( A < B \)). There are many existing comparator implementations [3]-[6].

In the design of combined two’s complement and floating point comparator [6] the comparator compares 32 b and 64 b two’s complement numbers and single precision and double precision floating point numbers. There are 4 outputs: greater than, lesser than, equal and unordered. Unordered arises when at least one operand is a Not a Number (NaN). It consists of 3 blocks: input conversion, comparator and exception handling. First block converts the input 32 b operands to 64b operands so that all operand sizes are handled by the same hardware. Second block do the magnitude comparison. The third block takes care of exceptions and special cases. Another comparator which is a 32 bit comparator for multi-number system [3] process double precision floating point data and single precision floating point data based on IEEE 754 standard, 32 unsigned fixed point data and 32 bit signed fixed point data.

The new comparator design proposed in [5] is having high speed of operation using digital CMOS cells. Parallel prefix tree architecture is used in this design. This comparator is a binary comparator, but it is efficient in terms of area and power. A single precision floating point comparator using parallel prefix tree structure [6] also uses the parallel prefix tree architecture, but it does not discusses about the invalid cases in comparison.
3. IEEE 754 representation

IEEE has standardized the representation of floating point numbers. The IEEE 754 format consists of sign bit, exponent and mantissa.

- Sign bit: 1 bit, 0 for positive number and 1 for negative number
- Exponent: for single precision it is 8 bit and it provides the exponent range from E (min) = -126 to E (max) = 127 and for double precision it is 11 bit and it provides the exponent range from E (min) = -1023 to E (max) = 1024
- Mantissa: for single precision it is 23 bit and for double precision it is 52.

The main classification is into single precision and double precision floating point number.

3.1 Single precision

In single-precision format, the most significant bit (MSB) is a sign bit, followed by 8 intermediate bits to represent an exponent, and 23 least significant bits (LSB) to represent the mantissa. As a result, the total width for single precision is 32 bits. The bias for single precision is 127. The format is shown in figure 1 (a).

3.2 Double Precision

In double-precision format, the MSB is a sign bit, followed by 11 intermediate bits to represent an exponent, and 52 LSB to represent the mantissa. As a result, the total width for double precision is 64 bits. The bias for double precision is 1023. The format is shown in figure 1 (b).

![Fig.1. IEEE 754 format. (a) Single precision format (b) Double precision format](image)

4. Special cases in floating point numbers

Table 1 shows the special case numbers as defined by the IEEE-754 standard and their data bit representations.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Sign Field</th>
<th>Exponent Field</th>
<th>Mantissa Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>Don’t care</td>
<td>All 0s</td>
<td>All 0s</td>
</tr>
<tr>
<td>Positive Denormal</td>
<td>0</td>
<td>All 0s</td>
<td>Non-zero</td>
</tr>
<tr>
<td>Negative Denormal</td>
<td>1</td>
<td>All 0s</td>
<td>Non-zero</td>
</tr>
<tr>
<td>Positive Infinity</td>
<td>0</td>
<td>All 1s</td>
<td>All 0s</td>
</tr>
<tr>
<td>Negative Infinity</td>
<td>1</td>
<td>All 1s</td>
<td>All 0s</td>
</tr>
<tr>
<td>Not a Number(NaN)</td>
<td>Don’t care</td>
<td>All 1s</td>
<td>Non-zero</td>
</tr>
</tbody>
</table>

In floating-point calculations, NaN is not the same as infinity, although both are typically handled as special cases in floating-point representations of real numbers as well as in floating-point operations. An invalid operation is also not the same as an arithmetic overflow (which might return an infinity) or an arithmetic underflow (which would return the smallest normal number, a denormal number, or zero).
5. Double precision floating point comparator

The double precision floating point comparator using parallel prefix tree structure[5] compares two input operands and produces the comparison result as less than or greater than or equal or unordered. If A and B are two operands, then the output will be greater than if A>B, less than if A<B, equal if A=B and unordered if either A or B is (Not a Number) NaN.

The advantage of the proposed comparator design is the handling of invalid numbers. A separate module is implemented for handling the invalid floating point numbers. The invalid numbers are represented as Not a Number (NaN). The NaN is a special case in floating point format. NaN values are generated from 0/0, ∞, -∞, and any operation with a NaN operation. NaN’s have the property that they are unequal to anything, even themselves.

Figure 2 shows the block diagram of the floating point comparator. It consists of three blocks. They are comparison resolution module, decision module and exception handling module. The comparison resolution module and decision module performs the magnitude comparison [4]. The exception handling module handles the exceptions in the operation.

The Comparison resolution module is MSB to LSB parallel prefix tree structure. In this structure the MSB’s are first compared, if that is equal then it moves on to the next significant bit. The parallel prefix tree structure is divided into five hierarchical prefixing sets, where each group performs specific function. The output of each group serves as the input to next group. It consists of XOR and multiplexer network.

![Fig. 2. Block diagram](image)

The decision module makes the decision that whether the numbers are equal, greater than or less than. It consists of OR network. The exception handling module takes care of the exceptions or special cases in IEEE 754 representation.
5.1 Architectural overview of comparator

The comparison resolution module in fig.2 is a novel parallel prefix tree structure. The parallel tree structure is shown in figure 3. It performs bitwise comparison of two \( N \)-bit operands \( A \) and \( B \). The comparison resolution module performs the bitwise comparison asynchronously from left to right, such that the comparison logic’s computation is triggered only if all bits of greater significance are equal. The comparison resolution module performs the bitwise comparison asynchronously from left to right, such that the comparison logic’s computation is triggered only if all bits of greater significance are equal [5]. To reduce switching activities, as soon as a bitwise comparison is not equal, the bitwise comparison of every bit of lower significance is terminated and all such positions are set to zero. The decision module uses two OR-networks to output the final comparison decision based on separate OR-scans of all of the bits on the left side and all of the bits on the right side as shown in fig.3.

Fig. 3. Parallel tree structure
The structure can be partitioned into five hierarchical prefixing sets, as depicted in Fig. 3. Set 1 compares the \(N\)-bit operands \(A\) and \(B\) bit-by-bit, using XOR cells. Set 2 consists of nor cells, which combine the termination flags for each of the four XOR cells from set 1. Set 3 consists of NOR type cells, which are similar to the NOR type cells in set 2, but can have more logic levels, different inputs. Set 4 consists of AND cells, whose outputs control the select inputs of multiplexer cells (two-input multiplexors) in set 5, which in turn drive both the left most bits and the right rightmost bits. The inputs to the AND cells are all inverted except one from the set 5. Set 5 consists of multiplexer cells (two-input, 2-b-wide multiplexors). One input is \((A_k, B_k)\) and the other is hardwired to 00. The select control input is based on the AND cell output from set 4. The output of multiplexer is 2-bit. The left most bits are combined and given to the left side OR gate. The right most bits are combined and given to the right side OR gate. The output of left OR gate is \(G\) and right OR gate is \(L\) which represents greater than and less than respectively.

All components within each set operate in parallel. This is a key fact to increase operating speed while minimizing the transitions to a minimal set of leftmost bits needed for a correct decision. This prefixing set structure bounds the component's fan-in and fan-out regardless of comparator bit width. It eliminates heavily loaded global signals with parasitic components and thus improving the operating speed and reducing the power consumption.

An 8-b comparison of input operands \(A = 01111101\) and \(B = 01101001\) can be taken as example. In the first step, a parallel prefix tree structure generates the encoded data on the left side signals and right side signals for each pair of corresponding bits from \(A\) and \(B\). In this example, \(A_7 = 0\) and \(B_7 = 0\) encodes as left7 = right7 = 0, \(A_6 = 1\), and \(B_6 = 1\) encodes as left6 = right6 = 0, \(A_5 = 1\) and \(B_5 = 1\) encodes left5 = right5 = 0, \(A_4 = 1\) and \(B_4 = 0\) encodes as left4 = 1 and right4 = 0. At this point, since the bits are unequal, the comparison terminates and a final comparison decision can be made based on the first four bits evaluated. The parallel prefix structure forces all bits of lesser significance on each side to 0, regardless of the remaining bit values in the operands. In the second step, the OR-networks perform the bus OR-scans, resulting in 1 and 0, respectively, and the final comparison decision is \(A > B\).

6. Results and discussion

The double precision floating point comparator is designed and simulated in verilog HDL. It is also simulated using cadence encounter tool. The simulation results are shown in figure 4.

![Simulation result using Xilinx ISE](image)

The simulation result using Isim (Incisive simulator) simulator of cadence is shown in figure 5. When input operand \(A\) is greater than \(B\), then the output is \(G\) (denotes greater than) is high. The output \(E\) (denotes equal) is high when both inputs are equal. When the input \(A\) is less than input \(B\), then output \(L\) (denotes less than) is high. The output \(U\) (denotes unordered) becomes high when any one of the operands is NaN as shown in figure 4 and figure 5.

The design of the proposed floating point comparator is implemented using Virtex-5 XC5VLX20T as the targeted device. The synthesis report is given in Table 2. The results show that device utilization is less. The time delay is only 6.237 ns. The schematic diagram obtained from cadence encounter tool is shown in figure 6.
The proposed comparator is also synthesized using the RTL compiler of cadence encounter tool. Area report, power report, timing report and gate report are obtained from the same. The cell area is 1415 µm², leakage power is 2.828µW and dynamic power is 31.419µW.

Table 2. Device utilization and timing summary

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice LUTs</td>
<td>262</td>
</tr>
<tr>
<td>Number of Slice LUTs in use</td>
<td>128</td>
</tr>
<tr>
<td>Number of IOs</td>
<td>132</td>
</tr>
<tr>
<td>Number of Bonded IOBs</td>
<td>132</td>
</tr>
<tr>
<td>Time Delay</td>
<td>6.237ns</td>
</tr>
</tbody>
</table>

Fig. 5. Simulation result using Isim

Fig. 6. Schematic diagram
The comparator proposed in [4] is a 32 bit floating point comparator which has an IEEE encoder circuit and it uses the same parallel prefix tree structure but it does not handle invalid cases. The proposed comparator resolves the problem of handling invalid cases and the precision is also high. The leakage power is less for the proposed comparator.

7. Conclusion

This paper proposes the design of a double precision floating point comparator. The proposed comparator compares double precision floating point numbers with an added advantage that it handles the invalid cases. This comparator can therefore be used in high precision operations. The simulation results are shown for IEEE 754 compliant double precision floating point comparator using Xilinx ISE targeting Virtex-5 fpga and Cadence encounter tool with TSMC 180 nm technology. The logic utilization is less and the time delay is only 6.237ns. As a future work this comparator can be incorporated to an FPU.

References