Implementation of Non Restoring Interval Divider unit

1Milind R. Patel, 2Prof. Deepali H. Shah

1,2 Instrumentation and Control Engineering Department, 
L. D. College of Engineering, Gujarat Technological University, 
Ahmedabad, Gujarat, India.

Abstract—Interval arithmetic provides a general methodology for bounding errors. Interval arithmetic provides reliability and accuracy by computing a lower and upper bound in which each result is guaranteed to reside. An interval’s width ensures the maximum possible error. The approaches consisted on developing hardware support for interval arithmetic division unit by incorporating existing conventional floating point units. This unit requires slightly more area than a conventional floating point divider. It provides a significant performance improvement over software implementations of interval divider. This paper presents the implementation of non restoring interval division algorithm for double precision binary data.

Keywords—Interval arithmetic, non restoring algorithm and floating point arithmetic.

I. INTRODUCTION

In many applications, such as scientific computing, the large number of arithmetic operations and the reliance placed on the results make it extremely important to provide accurate and reliable numerical computations. Unfortunately, round off error and catastrophic cancellation can quickly lead to results that are completely inaccurate [4]. Consequently, an efficient method is needed for monitoring and controlling errors in floating-point computations [4].

Interval arithmetic is faster than other error constraining techniques like various high precision and exact method, and can use arithmetic precisions that allow efficient processing [8]. Interval arithmetic does not deal with a single floating point number, which is an approximation of the desired real number, but deals with an interval [9]. It contains two floating point numbers, the upper bound of the interval and the lower bound of the interval. The interval has the property that it surely contains the desired real number. The width of the interval is a measure of the accuracy for the arithmetic computations, or better said is a measure of the lack of accuracy [9].

The software implementation of interval arithmetic is slow due to function calls, memory management, error and range checking, changing rounding modes, and exception handling, while conventional floating point arithmetic is provided by fast hardware, interval arithmetic is simulated with software routines based on integer arithmetic [5]. It has benefits of high speed interval division using share hardware with existing floating point divider with relatively minor hardware modification [5].

II. INTERVAL DIVISION

For the values of X = [x_l, x_u] and Y= [y_l, y_u] interval divider can be computed as [5]:

\[ Z = X \div Y \]

\[ = \left[ \min \left( \nabla \left( x \div y \right), \nabla \left( x \div y \right), \nabla \left( x \div y \right), \nabla \left( x \div y \right) \right), \max \left( \Delta \left( x \div y \right), \Delta \left( x \div y \right), \Delta \left( x \div y \right), \Delta \left( x \div y \right) \right) \right] \]

Where \( \nabla \) denotes rounding downward toward negative infinity and \( \Delta \) denotes rounding upward toward positive infinity. Based on this definition, computing the interval endpoints for the quotient \( Z \) requires eight floating point divider and six comparisons. It is also necessary to handle special cases in interval divider. Interval divider results in \( Z = [-\infty, \infty] \) when \( 0 \in Y \).

III. COMBINED INTERVAL AND FLOATING POINT DIVIDER UNIT

Figure 1 shows a block diagram of the combined interval and floating-point divider unit. Compared to the floating point divider, the combined unit requires two additional input registers and one additional output register [5]. It also requires control logic and multiplexers to select the interval endpoints, and control logic to set the rounding mode appropriately. By setting certain control bits, this unit is capable of performing either interval or floating-point division.
In figure 1 two multiplexers select the endpoints to be divided based on the toggle bits, \( t_x \) and \( t_y \). If the toggle bit is one, the lower interval endpoint is selected; otherwise the upper interval endpoint is selected. The values for the toggle bits are determined based on the sign bits of the interval endpoints, \( s_{x1}, s_{xu}, s_{y1}, s_{yu} \). A control bit \( le \) is set to one when the lower interval endpoint is being computed and zero when the upper interval endpoint is being computed. Table I shows the value of the sign and toggle bits for the seven cases for divider. Case 7 shows exceptional behavior in interval division [5].

### IV. NON RESTORING DIVISION ALGORITHM

For non restoring division sequence of operations performed is as follow [12]:

1) The divisor (\( Y \)) is subtracted from the partial remainder (\( R \)).
   \[ R \leftarrow R - Y \]
2) If the result is negative, the partial remainder is restored in the same step.
   \[ R \leftarrow R - Y + Y \]
3) In the next step, the partial remainder is shifted left with one position.
   \[ R \leftarrow 2 \times R \]
4) The divisor is subtracted from the partial remainder.
   \[ R \leftarrow 2 \times R - Y \]

The same result can be obtained by another sequence of operations:

1) The divisor is subtracted from the partial remainder.
   \[ R \leftarrow R - Y \]
2) If the result is negative, the partial remainder is not restored, but it is shifted left with one position in the next step.
   \[ R \leftarrow 2 \times R - 2 \times Y \]
3) The divisor is added to the partial remainder.
   \[ R \leftarrow 2 \times R - 2 \times Y + Y \]

### TABLE I

**SETTING OF SIGN BIT AND TOGGLE BIT FOR INTERVAL DIVISION**

<table>
<thead>
<tr>
<th>Case</th>
<th>Condition</th>
<th>( s_{x1} )</th>
<th>( s_{xu} )</th>
<th>( s_{y1} )</th>
<th>( s_{yu} )</th>
<th>( Z )</th>
<th>( le=1 )</th>
<th>( le=0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( x_l, y_l \ &gt; 0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>([x_l \div y_u, x_u \div y_l])</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>( x_l, y_u \ &lt; 0 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>([x_u \div y_u, x_u \div y_l])</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>( x_l \ &lt; 0, y_l \ &gt; 0 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>([x_l \div y_u, x_u \div y_u])</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>( x_l \ &lt; 0, y_u \ &lt; 0 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>([x_u \div y_l, x_u \div y_u])</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>( x_l \ &lt; 0 \times x_u, y_l \ &lt; 0 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>([x_l \div y_u, x_u \div y_u])</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>( x_l \ &lt; 0 \times x_u, y_u \ &lt; 0 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>([x_u \div y_l, x_u \div y_u])</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>( y_l \ &gt; 0 \times y_u )</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>([-\infty, +\infty])</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
V. SIMULATION RESULTS

Three different simulation results are shown, which are case selection module, non restoring division algorithm and interval non restoring division algorithm. These modules are programmed into Verilog HDL language, implemented by Xilinx ISE 13.2 synthesis tools and simulated using Modelsim 10.0b SE tools.

A. Case selection module

Case selection module selects the seven different cases of the interval division as shown in Table I. In case selection module first of all four binary 64 bit data splitted into the 64 bit floating format. From the sign bits of the floating format data, toggle bits t_x and t_y of two multiplexer are selected. On the basis of toggle bits interval input end points are selected. Figure 2 shows the simulation result of case selection module.

Fig. 2 Simulation result of case selection module

Fig. 3 Simulation result for the interval non restoring division algorithm
B. Interval non restoring division algorithm

Interval non restoring division algorithm is a top module. It combines both the case selection module and non restoring division algorithm module into top module. Unsigned non restoring division algorithm is performed for the mantissa of 64 bit floating format data. The dividend and divider are of 52 bit for mantissa. It provides output in the form of 52 bit quotient and remainder. le bit is use for the computation of the interval endpoints of the output. If the le bit is one lower interval endpoints computation occurs otherwise upper interval end point computation occurs. In this algorithm sign bit is selected using XOR logic, exponents are subtracted and mantissas are divided. Figure 3 shows simulation result for the interval non restoring division algorithm. In the simulation result first four rows show 64 bit binary data and last six rows show data of output interval end points.

VI. CONCLUSION

Interval arithmetic provides reliability and accuracy by computing a lower and upper bound in which each result is guaranteed to reside. Interval arithmetic removes the round off error and catastrophic cancellation generated due to the numerical computation. An interval’s width indicates the maximum possible error. The simulation result of the main component of the interval non restoring divider unit is shown. Analysis is done between the non restoring floating point algorithm and interval non restoring division algorithm with respect to the area and time response. Table II shows the area requirement for both algorithms, area requirement for the interval division algorithm is more than non restoring floating point algorithm due to the two extra input register and one output register. Table III shows the timing report of both algorithms, total delay for the output execution in non restoring interval division is less than non restoring floating point algorithm.

ACKNOWLEDGMENT

The authors wish to thank Instrumentation and Control Engineering Department of L.D.C.E. for valuable discussions and suggestions. We are also indebted to the referees whose detailed and insightful comments have greatly contributed to the clarity of the paper.

REFERENCES