VOL. 12, NO. 10, MAY 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

TRANSLATION OF DIVISION ALGORITHM INTO VERILOG HDL

Yusmardiah Y.[1], Darmawaty Mohd A.[1], Abdul Karimi H.[1], Abdul Aziz Abdul R.[2] and Ahmad Kamsani S.[2]
[1]Faculty of Electrical Engineering, Universiti Teknologi Mara, Shah Alam, Malaysia
[2]Signal Processing Lab, Computational Science Division, TM Innovation Centre, Cyberjaya, Malaysia
E-Mail: ymardiahyusuf@gmail.com

ABSTRACT

This paper deals with the design of sixteen bit division algorithms, programmed by using Xilinx ISE 14.4 software for translating the arithmetic operation for division operation. In recent, many researchers have proposed the algorithms to carry out the computation task in hardware instead of software, with the aim to increase the performance of computation. We explained and translated the non-restoring method for division operation. This method is simple to implement since it requires only adder or subtractor in each iteration. Hence, it does not require any other hardware components such as multipliers and multiplexers. The algorithm is translated into Verilog Hardware Description Language that simulated using Integrated Synthesis Environment (ISE) Simulator and then synthesized using Synopsys Design Compiler. The system will only process unsigned binary division hence producing in fixed point value.

Keywords: division, non-restoring algorithm, verilog HDL, Xilinx.

INTRODUCTION

In the world of digital signal processing, the division operation is in widely used, for example in wireless signal processing, image processing, computer graphics and etc. Mostly, the algorithms are implemented using fixed point arithmetic due to expected area and power savings [1]. In many computer applications, division is less commonly used than multiplication, subtraction or addition and takes much longer to compute.

Division is the most complicated of all the elemental operations, whether to implement the algorithm in hardware or software. However, it has been shown that ignoring its implementation can result in significant system performance degradation for many applications [2]. There are number of binary division algorithm such as Multiplicative Algorithm, Approximation Algorithms, CORDIC Algorithm and Continued Product Algorithm.

Fixed-point arithmetic is regularly used in the calculations because of the costing since floating point calculations could significantly increase the size of the design and make it more complex. However, the existing solutions for the fixed-point division are limited in terms of input and output widths of the modules. Among the solutions is the Xilinx IP Core Divider which is presented in [3]. There are radix 4, 8, 16 and even 256 algorithms which are faster but difficult to implement [4].

The disadvantages of floating-point representation are slower and less precise than the fixed-point. There are three basic components for floating-point representation: mantissa, exponent and sign. Based on Goldschmidt's algorithm, the division operations associated with FMA designed for single precision floating point have been proposed [5].

In this paper, the Verilog HDL code for non-restoring algorithm is proposed. However, the size of bit is limited to 16 bit value for the input dividend and divisor. The non-restoring division gives the exact value of the quotient and remainder, besides the implementation required less hardware since the calculation only involves shifting process, arithmetic addition and subtraction.

This paper is organized as follows. In section 2, the restoring division, SRT division and Vedic Architecture: Nikhilam Sutra algorithms are presented. The Non-Restoring Division Algorithms is explained in section 3. Section 4 presents the proposed Verilog code for the algorithm. In section 5, the simulation result is explained. Finally, the summarization and future recommendations is presented in the last section.

DIVISION ALGORITHMS

In recent years, many researchers have proposed different algorithms. They aimed to perform fast division operation and at the same time enhancing the performance. Some of the proposed division algorithms to include:

Restoring division

Restoring division functions on fixed-point fractional numbers and depends on the following assumptions [6], $D < N$ and $0 < N, D < 1$.

The quotient digits Q are formed from the digit set {0, 1}. To begin the operation, the dividend and the divisor is broke into the right half of the 2n-bit A register and into the left half of the 2n-bit B register respectively. The divisor B is subtracted from the remainder register A. If the result of previous step is negative, set the quotient, $Q0 = 0$ and restore the old remainder. This is the reason why this method is called restoring division. Else, set Q0 to 1. In the next step, the divisor is shifted to the right, aligning the divisor with the dividend for the next iteration. Repeat the steps until there is no bit left

SRT division

The name of the SRT division stands for Dura W. Sweeney, James E. Robertson and Keith D. Tocher who proposed a fast algorithm for 2's complement numbers that use the technique of shifting over zeros for division. [7].

The basic algorithm for binary (radix 2) SRT division is initially by inserting dividend and divisor into A and B registers respectively. If register B has k leading

zeros, shift all the registers (B and A) positions left k bits. Then, the following steps are repeated n times. If the top three bits of the A register are equal, shift the A registers one position left and set Qi = 0. If the top three bits of the A register are unequal and negatives, shift the A registers one position left, set Qi = –1 and add B to A. Otherwise, shift A one bit left, set Qi = 1 and subtract B. After the steps are repeated by n times and the final remainder is negative, correct the remainders by adding B also correct the quotient by subtracting 1 from Qi. Shift remainder k bits right.

Vedic architecture: Nikhilam Sutra

Vedic Sutra, Nikhilam [8] algorithm is started by loading the dividend and divisor into A and B register. Firstly, set the incrementer/quotient with '0'. Then, determine the complement B with respect to 2n. Add B with A. If the carry is '1', then feed the result to the adder (remainder). Next, increment the content of the incrementer by one. Repeat the steps until the result is less than the divisor, B. To sum up, the result of the addition is the remainder and the incrementer is the quotient.

NON-RESTORING DIVISION ALGORITHM

The non-restoring algorithm comes from restoring division and it calculates the remainder by successively subtracting the shifted divisor from the dividend until the remainder is in the appropriate range. The method is [9]:

Assume that we have dividend, D and divisor, X as an input data, quotient, Q as division result and R as remainder. The steps of the non-restoring algorithm are calculated as visible in Figure-1.

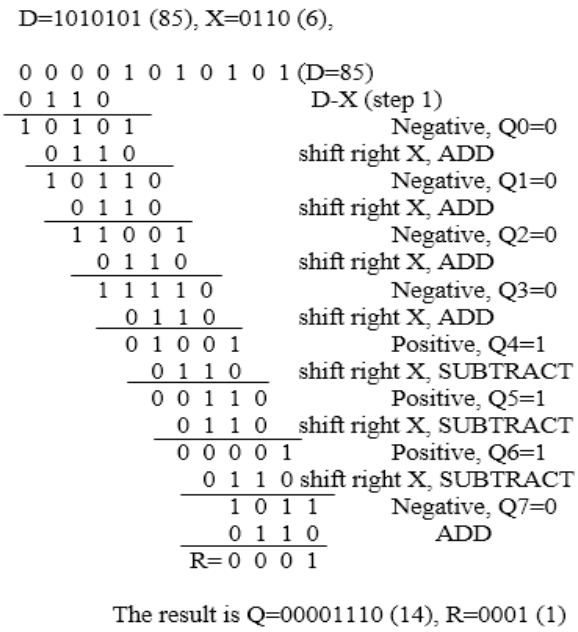

Figure-1. Steps to calculate binary non-restoring division algorithm.

As illustrated in Figure-1, the process starts by subtracting from the most significant bit of dividend with divisor. After making the subtraction process, bring down the next MSB of dividend and attached to the results from the first step. These steps are repeated until all the bits of dividend are calculated as well as the bits of quotient are determined.

From the algorithm, it can be concluded that if the result of subtraction is negative, 0 is selected as the quotient Q. On the other hand, if the result of subtraction is positive which gives 0 as a different, quotient, Q is selected as 1. In summary, Table-1 provides the steps to calculate the binary number for non-restoring division algorithm.

Table-1. Methods to solve binary non-restoring division algorithm.

Step	Method
1	Subtract the divisor, X from the most significant bit (MSB) of the dividend, D.
2	Bring down the next MSB (left) of the divisor and attach it to the result of step 1
3	Check the sign for the result of step 2. If the result of step 2 is positive: a. Set the next MSB of the Q to 1. b. Subtract the divisor from the result to produce a new result. If the result from step 2 is negative: a. Set the next MSB of the quotient, Q to 0. b. Add the divisor to the result to produce a new result.
4	Repeat steps 2 and 3 until all bits of the quotient are determined.

PROPOSED VERILOG HDL

This paper proposed a Verilog HDL coding of non-restoring division algorithm as shown in Figure-1. The clk is the input clock signal, means that the process is begin to calculate the division operation in the first clock cycle and signal is ready when the iteration is done. However, the size of bit is limited to 16 bit value for the input dividend and divisor. In this implementation, the divisor is shifted to right by one bit until there is no bit left.

```
module Division(clk,ready,Dividend,Divisor,Q,R);

 input         clk;
 input [15:0]  Dividend;
 input [15:0]  Divisor;

 output[15:0]  Q;
 output[15:0]  R;
 output        ready;

 reg [15:0]    Q;
 reg [31:0]    Dividend_copy, Divisor_copy, diff;

 wire [15:0]   remainder = Dividend_copy[15:0];

 reg [31:0]     bit;

 wire          ready = !bit;

 initial bit = 0;

 always @( posedge clk )

   if( ready )
      begin
         bit = 16;
         Q = 0;
         Dividend_copy = {16'd0,Dividend}; //Output Dividend_copy=31 bits
         Divisor_copy = {1'b0,Divisor,15'd0}; //Output Divisor_copy=31 bits
      end
   else
      begin
         diff = Dividend_copy - Divisor_copy;   //Difference is negative:
                                                //copy dividend and put 0 i
         Q = Q << 1;            //Shift 1 bit

      if( !diff[31] )
         begin
           Dividend_copy = diff;
           Q[0] = 1'd1;
         end

         Divisor_copy = Divisor_copy >> 1;
         bit = bit - 1; //Check for count,
         //when bit = bit-1 then stop the loop

 end

endmodule
```

Figure-2. Verilog HDL for division.

In this paper, the Verilog HDL codes for division are generated and simulate using Xilinx ISE 14.4. The Verilog HDL code is broken down into modules which deal with the division of 16 bit dividend and 16 bit divisor. We chose the non-restoring algorithm because it is simple to implement since it requires basic adder or subtractor operation and the shifting process, which is to the left or right in each stage of calculation. Hence, it does not involve other hardware components such as multipliers and multiplexors. The top module connects all of the inputs and produces output as shown in Figure-2.

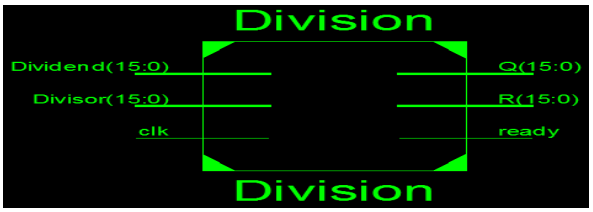

Figure-3. Top RTL view of division algorithm.

RESULTS AND DISCUSSIONS

Simulation

The simulation result of 16 bit fixed point division algorithm is shown in Figure-3. It is clearly shown that the system needs 17 clock cycles, so that the output for 16-bit input is in the ready state. READY state here means that the enumeration stage is completed. We consider the division of 16 bits two fixed point numbers, which are 32768 (1000000000000000) and 158 (0000000010011110) fed as the input dividend and divisor respectively. The desired output of quotient = 207 (0000000011001111) and remainder 62 (0000000000111110) is shown in Figure-4.

This proved that the calculation executed by the Verilog HDL code is valid. The simulation performed the calculation through the iteration per bit value until all the bits for quotient and remainder is determined.

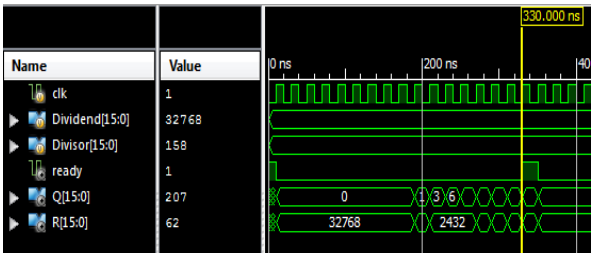

Figure-4. Simulation result of 16 bit fixed point division (zoom out).

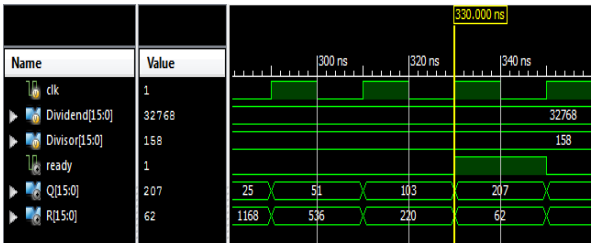

Figure-5. Simulation result of 16 bit fixed point division (zoom in).

Synthesis

Verilog HDL Code for division is then synthesized using the XC6VLX240T device with the package of Vertex 6 FPGA family implementation and the implementation clock frequency is 10 MHz. The characteristic of the device [10] is listed in Table-2.

Table-2. XILINX VIRTEX-6 XC6VLX240T features.

Device	CLBs arrays (Total slices)	Maximum I/O
XC6VLX240	37,680	720

The translation of the division algorithm is quite simple and the implementation is done using the Xilinx Vertex-6. The FPGA implementation's result for the 16 bit Non-Restoring divider is shown in Table-3.

Table-3. FPGA Implementation for 16 bit Divider.

No. of slices	No. of LUTs	No. of bonded IOBs
95	166	160

Table-3 shows the total hardware required for the implementation in FPGA. The 16 bit non-restoring method only utilizes small resources as compared to the SRTs division proposed in [11]. In conclusion, the computation time is faster but difficult to implement since it consumed a lot of hardware resources.

CONCLUSIONS

The non-restoring division algorithm is presented in this paper. This algorithm is implemented in Verilog HDL and synthesized by using Xilinx ISE 14.4. The aim is to focus on the simplicity of the algorithm which is easy to translate into Verilog codes. This is because the non-restoring method only involved the basic addition and subtraction and shifting process, which is either left or right. Hence, this method can be implemented for any value of binary numbers for division operations.

However, this algorithm has limitation on the latency because the hardware needs to calculate the number in sequence (per bit) to converge. Means that the higher the bit being processed, the longer it takes to calculate the output. Later in the future, the parallel method will be implemented for division algorithm so that the hardware is improved in terms of speed's calculation.

ACKNOWLEDGEMENT

This research was supported by University Teknologi MARA (UiTM) and Kementerian Sains, Teknologi dan Inovasi (MOSTI).

REFERENCES

[1] Bhoyar R., Palsodkar P. and Kakde S. 2015. Design and implementation of Goldschmidts algorithm for floating point division and square root. In: IEEE International Conference on Communications and Signal Processing. pp. 1588-1592.

[2] S. F. Oberman and M. J. Fiynn. 1997. Division algorithms and implementations. IEEE Transaction on Computers. 46(8): 833-854.

[3] N. Sorokin. 2006. Implementation of high-speed fixed-point dividers on FPGA. Journal of Computer Science and Technology. 6(1): 8-11.

[4] Ercegovac M. D. and Muller J. M. 2005. Variable radix real and complex digit-recurrence division. In: IEEE International Conference on Application-Specific Systems, Architecture and Processors. pp. 316-321.

[5] Floating-Point Working Group. 1987. IEEE standard for binary floating-point arithmetic. In: ACM Special Interest Group on Programming Languages. pp. 9-25.

[6] S. Kaur, Suman, M. S. Manna and R. Agarwal. 2013. VHDL implementation of non-restoring division algorithm using high speed adder/subtractor. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering. 2(7): 3317-3324.

[7] Harris D. L., Oberman S. F. and M Horowitz. A. 1997. SRT division architectures and implementations. In: 13th IEEE Symposium on Computer Arithmetic. pp. 18-25.

[8] Jain S., Pancholi M., Garg H. and Saini S. 2014. Binary division algorithm and high speed deconvolution algorithm. In: 11th IEEE International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology. pp. 1-5.

[9] Ghatte N., Patil S. and Bhoir D. 2014. Single precision floating point division. In: 5th IRF International Conference. pp. 34-38.

[10] Xilinx. 2015. Virtex-6 family overview. https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.

[11] M. R. Patel, T.V. Shah and D. H. Shah. 2012. Implementation and analysis of interval SRT radix-2 division algorithm. International Journal of Electronics and Computer Science Engineering. 1(3): 971-976.