International Journal of Science, Environment and Technology, Vol. 4, No 2, 2015, 440 – 445

ISSN 2278-3687 (O)
2277-663X (P)

ROUNDED CONSTANT DIVISION VIA ADD-SHIFT IN VERILOG

Fouziah Md Yassin[1], Ag. Asri Ag. Ibrahim[3], Noor Syamimi Abd Manah[1], Zaturrawiah Ali Omar[2] and Saturi Baco[1]
[1]Physics with Electronics Programme
[2]Mathematics and Graphics Programme
Faculty of Science and Natural Resources
Universiti Malaysia Sabah (UMS), Jalan UMS,
88400 Kota Kinabalu, Sabah, Malaysia
[3]Faculty of Computer and Informatics,
Universiti Malaysia Sabah, Labuan International Campus,
Kampung Sungai Pagar, 87000 Labuan, Malaysia
Email: fouziahy@ums.edu.my / fouziahy@yahoo.com (**Corresponding Author*)

Abstract: An implementation of division in hardware is expensive. One of the alternatives is by replacing it with cheaper adder and shifter. This paper presents the condition of add-shift schemes that had been modifiedfrom existing algorithm. The constant denominators are 3, 5, 6, 7 and 9. The modifications are to eliminate the integer multiplication and to round the unsigned result to the nearest integer. The comparison results of the outputs between C++ and Verilog codes are used to verify the accuracy of the division process. Verilog code needs to be changed for any incorrect results. The required results were obtained. The outputs (div_out) of all denominators (deno) have been rounded to the nearest integer. However, the maximum bit widths of numerators (n) are only 13 except for the divisor of 3 that produces the maximum bit width up to 16.

Keywords: Constant division, add-shift, Verilog.

Introduction

Integer division instructions are considerably slower than multiplication while multiplication instructions are several times slower than addition (Möller and Granlund, 2011). Therefore, a lot of work has been done on optimizing constant division. Digit recurrence division is simple and has a lower complexity compared to other division methods (Kaur et *al.*, 2013). The digit recurrence is a subtractive division technique that uses addition or subtraction and shift in a manner similar to paper-and-pencil approach (Chiang et *al.*, 2000). It gives out a fixed number of quotient bits in iteration. Schwarzbacher et *al.*, 2000, had published a paper to find the best algorithm of division with add-shift design to optimise the area, speed and power consumption of the integrated circuit.

*Received Mar 16, 2015 * Published April 2, 2015 * www.ijset.net*

The output of unsigned division in Verilog is normally rounded towards zero. However, certain algorithm of block requires the division value to be rounded to the nearest integer. This paper presents the development of algorithm round the division output to the nearest integer using add-shift scheme. The algorithm was inspired by the problem encountered in designing block that applies division in Verilog and the concept done by Warren (2012).

Overview of Add-Shift Scheme in Constant Division

Srinivasan and Petry (1994) proposed a constant division algorithm which is an iterative algorithm for a division of the form 2^n±1. They found that the optimum formula can be generalised as:

$$Q = A\sum_{i=1}^{4} 2^{-2i} = A * (2^{-2} + 2^{-4} + 2^{-6} + 2^{-8}) \qquad (1)$$

whereQ is quotient and A is dividend. Equation (1) shows that the division by three is reducible to a multiplication by value, which is a close approximation of 1/3.

Warren (2012) also proposed a quite similar design where the constant division is changed into a sequence of shift and adds instruction. In addition, he also proposed an approximation of reciprocal of 3 that can be described in equation2 below:

$$q=(n\gg 2)+ (n\gg 4)+ (n\gg 6)+ \ldots +(n\gg 30) \qquad (2)$$

where q is quotient and n is dividend. However, he found that when the first term shifts out two bits, the next four bits and so on, it will result of almost 1 in the least significant bits. The shifts may contribute an error of almost sixteen as there are sixteen terms and resulting maximum total error of sixteen (Warren, 2012). As a result, a faster and more accurate method is used as shown in equation 2.3. The method results only five shifts operations and contributes only five maximum total errors which are lower and better than procedure in equation (3).

$$\begin{aligned} q&=(n\gg 2)+ (n\gg 4)\\ q&=q+(q\gg 4)\\ q&=q+(q\gg 8)\\ q&=q+(q\gg 16) \qquad (3)\end{aligned}$$

The remainder (r) can be calculated as shown in equation 4.

$$r=n-q*3 \qquad (4)$$

The a/b must be greater than 1/3 when 3 as a divisor, so that a shift result will be the same with the normal division. A sequence of approximations had been defined. Usually a small fraction of the sequence is easier to count. Therefore, the selected fraction is the smallest fraction and the best in this case is 11/32 (Warren, 2012). So, the division output of q is as in equation 5 below.

$$q=q+(11*r \gg 5) \tag{5}$$

Methods

Division of using add-shift method was developed to eliminate the multiplier component and to obtain the output that could be rounded up to the nearest integer. This simplifies the process in ASIC implementation while still producing the same output accuracy. This process is able to save the repetition time in the division operation. The division algorithm process of using adder and shift are as below:

i- Determine the divisor number; 3, 5, 6, 7 and 9.

ii- Calculate the output of the division for each divisor. It is changed from decimal to binary so that the value could be shifted according to the binary 2^n and further the quotient equation could be determined. The equations are arranged as shown in equation 5.

iii- Replace the multipliers component which is used for remainder (r) with adders in simple equation.

iv- Determine the equation of remainder and use the pattern of the remainder to get new quotient which the value is rounding to the nearest integer. Therefore, any output that have decimal point equal or more than 5, the output is added by one (1) in Verilog. The equation can be written as div_out<= div_out+1;

The constant division program was produced in both C++ and Verilog for functional verification as shown in Figure 1. As a result, their outputs could be compared and recorded. The output of C++ with normal division is used as the main reference. Adjustment of Verilog codes are required if the incorrect outputs detected. The process is repeated until the accurate result obtained.

Since the denominators of 3, 5, and 6 are using the same instructions as shown in Figure 2 (a), simplification could be done when compiling all the denominators. Same goes to the denominators of 7 and 9 that are using the same instructions as shown in Figure 2 (b). The simplification might be able to reduce the area size in ASIC implementation.

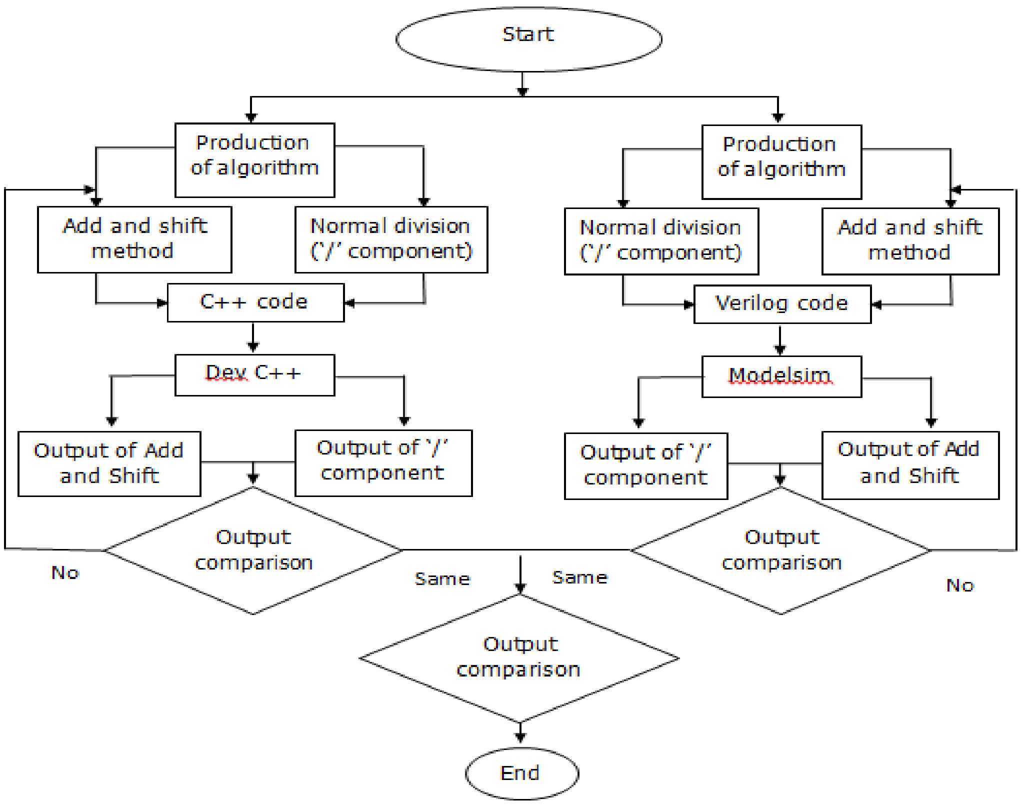

Figure 1: The process of output comparison to verify the accuracy of the division output in Verilog

```
q1<=q+(q>>4);
q2<=q1+(q1>>8);
q3<=q2+(q2>>16)
q4 <=q3>>2;
```

```
q2<= q1+ (q1>>12);
q3<= q2+ (q2>>24);
```

(a) (b)

Figure 2: Line of instructions for (a) 3, 5 and 6 as denominators (b) 7 and 9 as denominators

Results and Discussion

The functional verifications are done using Modelsim. The required results are successfully obtained for all selected denominators with the maximum numerator are 8191. Multiplier component in remainder (r) has been replaced with add-shift scheme. For an example is as shown in equation 6 for 3 as the denominator (deno).

$$r <= n5- ((q3<<1)+q3); \qquad (6)$$

The simulation results can be seen from the waveform shown in Figure 3 and 4. The numerator (n) is 13-bit wide which means it can calculate maximum number of 8191 while the denominator (deno) is 3-bit wide as 9 is the maximum constant number. Figure 3 shows the output of divide-by-3. From Figure 3(a), it shows that the output validation signal (out_sig) is asserted seven cycles after input signal (in_sig) is asserted. Both, Figure 3(a) and 3(b) shows that the output (div_out) is rounded to the nearest integer. For an example, 2/3 is 0.666 but whenever r= 2 is detected, the output (div_out) is added by 1 making the output become 1 instead of 0.Figure 4 shows the output of divide-by-7. From Figure 7(a), it shows that the output validation signal (out_sig) is also asserted seven cycles after input signal (in_sig) is asserted. Both, Figure 4(a) and 4(b) shows that the output (div_out) is added by 1 whenever r is greater or equal 4 and r is smaller or equal 6 (4≤ r ≤ 6).

The similar routines occurred for the denominators of 5, 6 and 9. The difference between these denominators is the value of r. It determines whether the ouput should be added by 1 or remains the value.

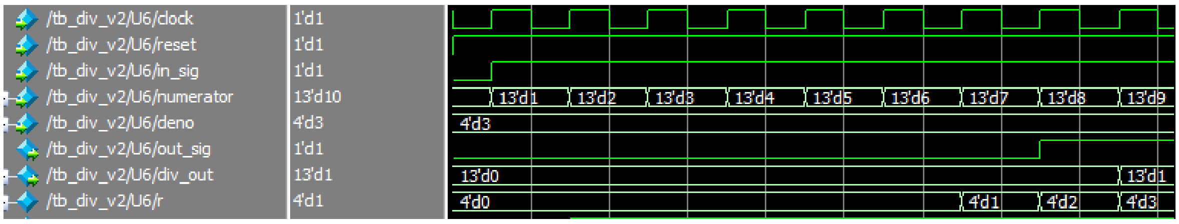

Figure 3 (a): The timing diagram when denominator=3 after in_sig is asserted.

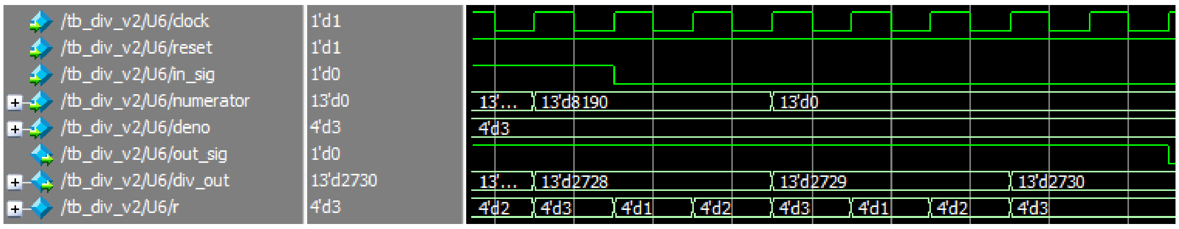

Figure 3 (b): The timing diagram when denominator=3 after in_sig is deasserted.

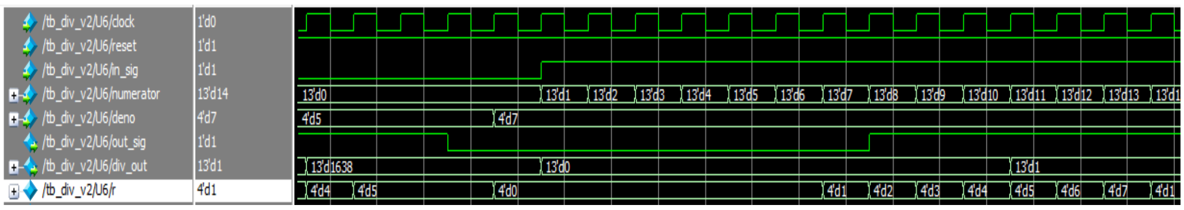

Figure 4 (a): The timing diagram when denominator=7 after in_sig is asserted.

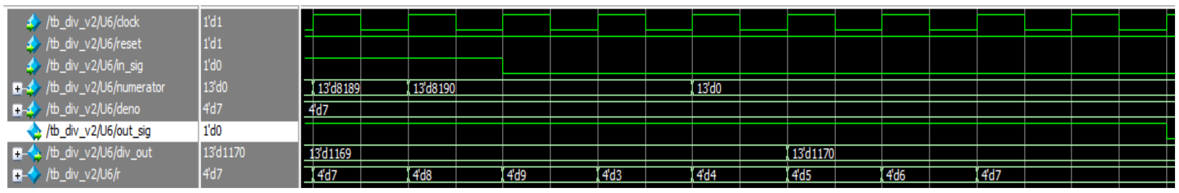

Figure 4 (b): The timing diagram when denominator=7 after in_sig is deasserted

Conclusion

It has been shown that the implementation constant division via add-shift scheme in Verilog producing outputs that is rounded to the nearest integer. However, the numerators are limited to 13 bit width and the denominators are only for 3, 5, 6, 7 and 9. The output can only be obtained in seven cycles after in_sig is asserted. The validation of the outputs is indicated by out_sig.

Acknowledgements

This research was funded by Universiti Malaysia Sabah under Skim Pensyarah Lantikan Baru (SLB0013-TK-2012)

References

[1] Chiang, J-S., Chung, H-D.& Tsai M-H.(2000), Carry-Free Tadix-2 Subtractive Division Algorithm and Implementation of the Divider. *Tamkang Journal of Science and Engineering*, 3(4), 249-255.

[2] Kaur, S., Suman, Manna, M.S. & Agarwal, R. (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.

[3] Möller, N. & Granlund, T. (2011).Improved Division by Invariant Integers.*IEEE Transactions on Computers*, 60(2), 165-175.

[4] Schwarzbacher, A. Th., Brutscheck, M., Schwingel, O. & Foley, J.B. (2000). Optimisation of Constant Divider Structures of the Form $2^{\wedge}$ n±1 for VLSI Implementation. *Irish Systems and signals Conference Proceeding*.

[5] Srinivasan, P. & Petry, F. (1994). Constant-division Algorithms. *IEEE Proceedings on Computers and Digital Techniques*, 141(6), 334-340.

[6] Warren, H.S. (2012). Hacker's Delight (Second Edition). Pearson Education.