ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 4, April 2017

Implementation of Proposed Highspeed, Low Power 16 Bit Multiplier

[1]Ankita Gupta, [2]Mandeep Singh [3]Narula Sandeep Shrivastava
[1][2][3]ECE Department JIIT, Noida- 201309, India

***Abstract*- — We have proposed a 16-bit high-speed multiplier using VHDL. With this method, the number of partial products has been reduced and the carry during addition has been eliminated. It also reduces the switching power which makes this multiplier a low power multiplier as compared to other multipliers. Due to the elimination of carry, the delay has been reduced which makes it a faster multiplier as well. The RTL circuit is much simpler, this decreased circuit complexity which leads to a better circuit with lesser delay and easier, cost-efficient way to implement the hardware.**

***Index Terms*— VHDL, Xilinx, Multiplier, Partial Addition.**

I. INTRODUCTION

VLSI (Very Large Scale Integration) design methodology ba- sically follows two design flows: (a) Front–end design flow. (b) Back-end design flow. Back–end design flow is to lower down and optimize the area, delay, and power of the circuits. Here, designers deal with the transistor level schematics and its corresponding layouts defining the original geometry of the design that gets to fabrication. In front-end design flow, the major focus remains on the functionality of the individual components allocated to the processor along with its overall functionality. Now, in industry, both these design methodol- ogy work hand in hand. Front-end flow does the functional analysis of the design system.

We are giving input to a known system and getting an output as RTL description using the back–end for circuit implementa- tion. But on the other hand, if we assume that we do not know the system and try to estimate the system using the output and the input; it becomes difficult in regular cases. Therefore, the main purpose in doing this work is to have an RTL description of a binary multiplier which can be synthesized [1][5] without even using the costly synthesis tools. The complete design has been made by using hardware description language (HDL) broadly in VHDL (Very High Specific Integrated Circuits HDL). VHDL is a most commonly used and popular standard language in the industry used to describe the hardware from the abstract form to the concrete level, technically from the design specifications to the RTL (Register Transfer Level) de- scriptions. We have shown the implementation of the designed multiplier.

The entire paper has been constructed in sections. Section II defines the details of the multiplier architecture. Section III shows the synthesized binary multiplier results with its RTL circuit and the synthesis report. Section IV is the conclusion.

II. MULTIPLIER ARCHITECTURE

Input Block- The input block has two input registers of 16 bits each. We have enlisted them in VHDL as entities. They are defined as input port A: in std_logic_vector (15 downto 0); B: in std_logic_vector (15 downto 0); where "std_logic" depicts the standard logic set by ieee.std_logic_1164.all, which is standard library setup by IEEE [11]. Vector (15 downto 0) is used as both the input ports have 16 bits each.

Operating Clock- Necessity of clock comes in when every job, task or operation assigned to the processor follows up a machine cycle. Every processor made till date has its own ma - chine cycle. To make every task assigned to the processor work according to the machine cycle, operating clock is needed. The clock is modeled in VHDL as: clk: in std_logic; in the entity. The clock signal is named as "clk".

Output Block- This block consists of a single register of 32 bits. The VHDL modeling for this block at entity level is: re- sult: out std_logic_vector (31 downto 0).

Multiplier- This section states the operation of multiplication of two unsigned binary numbers (16 bits), five signals (tempo- rary value holder) with the name of tmp, tmp2, tmp3, tmp4, tmp5 each of 32-bits and count and this operation is being synchronized with every

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 4, April 2017

change in the operating clock. Therefore, the inputs and output taken in consideration for the design are multiplier operand, multiplicand operand, clock, tmp, tmp2, tmp3, tmp4, tmp5, count and result. The VHDL model at entity level is:

multiplier: in std_logic_vector (15 downto 0);
multiplicand : in std_logic_vector (15 downto 0); clk : in std_logic;
signal tmp : std_logic_vector (31 downto 0) := "00000000000000000 000000 00000000" ;
signal tmp2 : std_logic_vector (31 downto 0) := "00000000000000000000000000000000";
signal tmp3 : std_logic_vector (31 downto 0) :=

"00000000000000000000000000000000";
signal tmp4 : std_logic_vector (31 downto 0) := "00000000000000000000000000000000";
signal tmp5 : std_logic_vector (31 downto 0) := "00000000000000000000000000000000";
signal variable count : INTEGER:=0;
result : out std_logic_vector (31 downto 0);

where clk is the clock. Here tmp, tmp2, tmp3, tmp4 and tmp5 are used as temporary signals to store the intermediate calcu- lations of the multiplier and count variable is used to keep a track of a number of execution of multiplication between mul- tiplier and multiplicand.

The algorithm used for this multiplier design states that the multiplication can be carried out by partial addition and shift- ing. It says that:
a) If the least significant bit of the multiplier is '1', then the multiplicand is simply copied as a partial addition in (31 downto 16);
b) If the least significant bit the multiplier is '0', then the par- tial addition is '0'.
c) Whenever a partial addition is done, it is shifted one bit to
the right of the previous partial addition. This lead to no carry, hence no use of adders.

This process is continued until all the multiplier bits are checked. Therefore, to perform it, we take a temporary regis- ter (32 bits) “tmp” and initialize it with all zeroes. A variable is taken, named “count” which is also initialized by '0'. This “count” is used to note how many bits of multiplier get exe - cuted per clock cycle. That is the whole operation of multipli - cation has to finish within a stipulated time (count<16). At first, the all temporary register (tmp, tmp2, tmp3, tmp4, tmp5) each of 32 bits has been initialized as “tmp” value mentioned above. They are taken to reduce data over-writing.

In the architecture of the design, if the clock “clk” signal is high then we run a process triggered by multiplicand, multi- plier, clock and count. The least significant 16 bits of register “tmp” {tmp(15)='0', tmp(14)='0', tmp(13)='0', tmp(12)='0', tmp(11)='0',t mp(10)='0', tmp(9)='0', tmp(8)='0', tmp(7)='0', tmp(6)='0',tmp(5)='0', tmp(4)='0', tmp(3)='0',tmp(2) ='0', tmp(1)='0', tmp(0)='0'} are replaced by the bits of the multi- plier. Now, if tmp(0)='1', then the partial addition is done fol- lowed by 1 bit right shift otherwise if tmp(0)=’0’ only 1 bit right shift is performed, the result is then stored in “tmp3”. In the following clock cycles, the least significant bit of tmp3 is checked and the same process is repeated, i.e. If tmp3(0)=’1’, the most significant 16 bits of register “tmp4” {tmp4(31)='0', tmp4(30)='0', tmp4(29)='0', tmp4(28)='0', tmp4(27)='0', tmp4(26)='0', tmp4(25)='0', tmp4(24)='0', tmp4(23)='0', tmp4(22)='0', tmp4(21)='0', tmp4(20)='0', tmp4(19)='0', tmp4(18)='0', tmp4(17)='0', tmp4(16)='0'} are replaced by the result of the partial addition obtained due to the addition of tmp3(31 downto 16) and the multiplicand, and the least 16 bits of tmp3, i.e. tmp3(15 downto 0) is copied to tmp4(15

downto 0). The result is then right shifted by 1 bit [2] and is stored in tmp5. Whereas if tmp3(0)=’0’ then 1 bit right shift is done in tmp3 and the result is stored directly in tmp5. At the end of the clock cycle, tmp5 is copied to tmp3 and count is in- creased.

This execution continues till the count gets greater than or equal to 16. Finally, the result is loaded with the final value of “tmp3” register obtained in final clock cycle.

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 4, April 2017

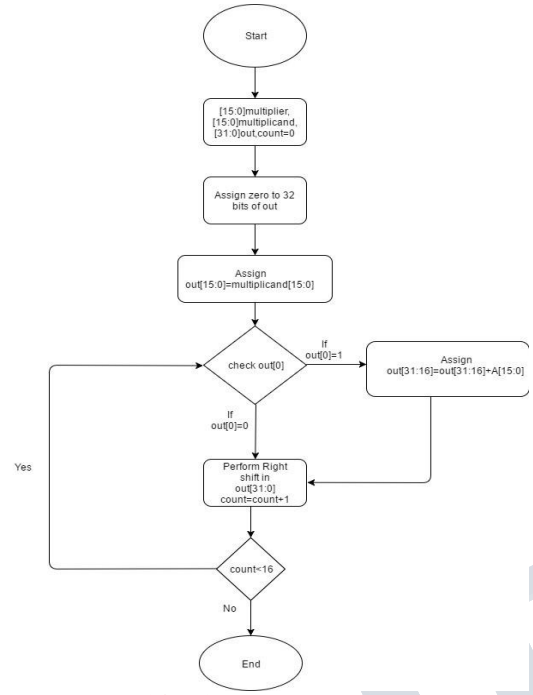

Fig 1. Flowchart of Multiplier

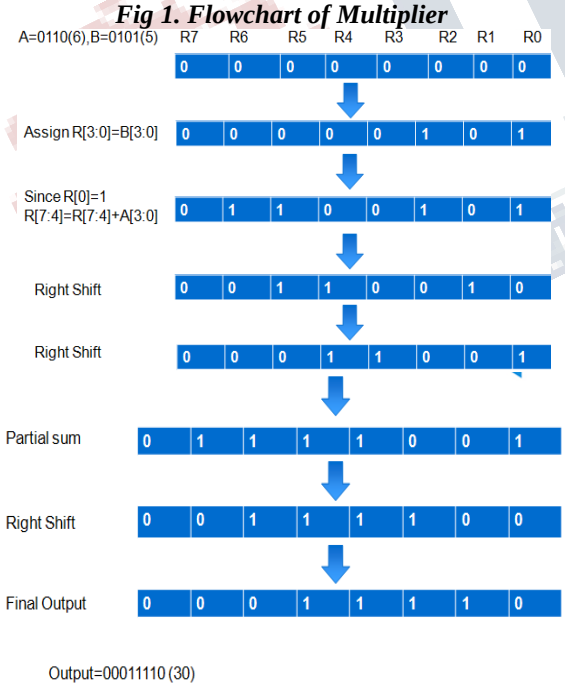

Fig 2. An Example of our algorithm using 4-bit

III. RESULTS

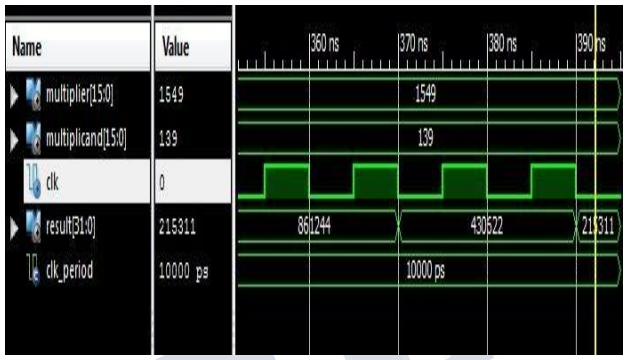

Fig 3. Result of the Multiplier (in 16-bit)

```
Data Path: tmp5_21 to tmp3_15
                                   Gate    Net
Cell:in->out      fanout  Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
FD_1:C->Q              5  0.447   1.079  tmp5_21 (tmp5_21)
LUT6:I0->O             2  0.203   0.961  tmp5[31]_GND_7_o_not_equal_5_o7_SW0 (N3)
LUT6:I1->O            16  0.203   1.005  tmp5[31]_GND_7_o_not_equal_5_o7 (tmp5[31]_GND_7_o_not_equal_5_o)
LUT4:I3->O             1  0.205   0.000  Mmux_GND_7_o_tmp5[31]_mux_6_OUT241 (GND_7_o_tmp5[31]_mux_6_OUT<30>)
FD_1:D                    0.102          tmp3_30
----------------------------------------
Total                     4.205ns (1.160ns logic, 3.045ns route)
                          (27.6% logic, 72.4% route)
```

Fig 4. Delay between tmp5 and tmp3 (Block-wise Delay)

```
Data Path: multiplier<0> to tmp3_0
                                   Gate    Net
Cell:in->out      fanout  Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
IBUF:I->O             62  1.222   1.874  multiplier_0_IBUF (multiplier_0_IBUF)
LUT5:I1->O             1  0.203   0.000  Mmux_GND_7_o_tmp5[31]_mux_6_OUT321 (GND_7_o_tmp5[31]_mux_6_OUT<9>)
FD_1:D                    0.102          tmp3_9
----------------------------------------
Total                     3.401ns (1.527ns logic, 1.874ns route)
                          (44.9% logic, 55.1% route)
```

Fig 5. Delay between multiplier and tmp3 (Block-wise Delay)

```
Data Path: result_30 to result<30>
                                   Gate    Net
Cell:in->out      fanout  Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
FD_1:C->Q              1  0.447   0.579  result_30 (result_30)
OBUF:I->O                 2.571          result_30_OBUF (result<30>)
----------------------------------------
Total                     3.597ns (3.018ns logic, 0.579ns route)
                          (83.9% logic, 16.1% route)
```

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 4, April 2017

```
Data Path: result_30 to result<30>
                                   Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 FD_1:C->Q             1   0.447   0.579  result_30 (result_30)
 OBUF:I->O                 2.571          result_30_OBUF (result<30>)
----------------------------------------
Total                      3.597ns (3.018ns logic, 0.579ns route)
                                   (83.9% logic, 16.1% route)
```

Fig 6. Delay of result (Final output)

```
MUXCY:CI->O           1   0.023   0.000  Madd_GND_1_o_x[15]_add_11_OUT_cy<8> (Mad
MUXCY:CI->O           1   0.023   0.000  Madd_GND_1_o_x[15]_add_11_OUT_cy<9> (Mad
MUXCY:CI->O           1   0.023   0.000  Madd_GND_1_o_x[15]_add_11_OUT_cy<10> (Ma
MUXCY:CI->O           1   0.023   0.000  Madd_GND_1_o_x[15]_add_11_OUT_cy<11> (Ma
MUXCY:CI->O           1   0.023   0.000  Madd_GND_1_o_x[15]_add_11_OUT_cy<12> (Ma
MUXCY:CI->O           1   0.023   0.000  Madd_GND_1_o_x[15]_add_11_OUT_cy<13> (Ma
XORCY:CI->O           3   0.370   0.389  Madd_GND_1_o_x[15]_add_11_OUT_xor<14> (G
LUT5:I3->O            1   0.097   0.000  Madd_GND_1_o_x[15]_add_12_OUT_lut<13> (M
MUXCY:S->O            1   0.353   0.000  Madd_GND_1_o_x[15]_add_12_OUT_cy<13> (Ma
XORCY:CI->O           2   0.370   0.384  Madd_GND_1_o_x[15]_add_12_OUT_xor<14> (G
LUT5:I3->O            1   0.097   0.000  Madd_GND_1_o_x[15]_add_13_OUT_lut<13> (M
MUXCY:S->O            1   0.353   0.000  Madd_GND_1_o_x[15]_add_13_OUT_cy<13> (Ma
XORCY:CI->O           2   0.370   0.383  Madd_GND_1_o_x[15]_add_13_OUT_xor<14> (G
LUT5:I3->O            1   0.097   0.000  Madd_GND_1_o_x[15]_add_14_OUT_lut<13> (M
MUXCY:S->O            1   0.353   0.000  Madd_GND_1_o_x[15]_add_14_OUT_cy<13> (Ma
XORCY:CI->O           1   0.370   0.511  Madd_GND_1_o_x[15]_add_14_OUT_xor<14> (G
LUT4:I1->O            1   0.097   0.279  Mmux_z<29>11 (z_29_OBUF)
OBUF:I->O                 0.000          z_29_OBUF (z<29>)
----------------------------------------
Total                     19.539ns (12.697ns logic, 6.842ns route)
                                   (65.0% logic, 35.0% route)
```

Fig 7. Final Path Delay of Multiplier

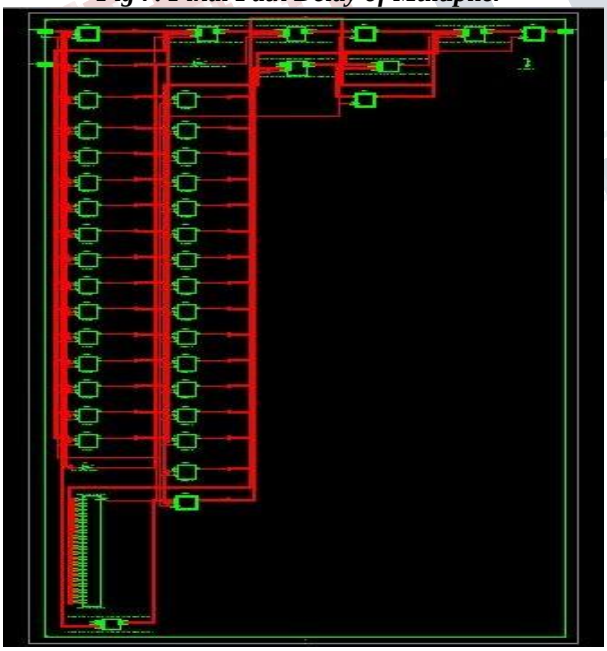

Fig 8. RTL schematic circuit of Multiplier

IV. CONCLUSION

In this paper, a 16-bit multiplier having high-speed and low power has been implemented. This multiplier does not use any carry adder method which makes the circuit less complex and has a smaller delay. It just partially adds and shifts one right. The demand of this subject is growing with recent advances in various hardware technologies, as the structure density of the new devices grows and the price-per element decreases and the demand for minimization of the current consumption rises [4].

REFERENCES

[1] Douglas L. Perry "VHDL Programming by Example" Fourth Edi- tion,Tata McGraw-Hill Publications, DOI: 10.1036/0071409548.

[2] J. Salivahanan, Arivazhagan S. Digital Circuits And Design, 3E. Vikas Publishing House Pvt Ltd, 2009.

[3] Volnei A. Pedroni, "Circuit Design with VHDL", MIT Press, Cam- bridge, Massachussetts, 2004, ISBN 0-262-16224-5 © 2004 Mass- achusetts Institute of Technology

[4]Fedra, Zbynek, and Jaromir Kolouch. "VHDL procedure for combinational divider." Telecommunications and Signal Process- ing (TSP), 2011 34th International Conference on. IEEE, 2011.

[5] Sadhu, A., & Bhattacharjee, P. (2014). Methodology of Standard Cell Library Design in. LIB Format. Journal of VLSI Design Tools & Technology, 4(1), 30-38.

[6] Pritam Bhattacharjee, Arindam Sadhu, Kunal Das. "A Register- Transfer-Level Description of Synthesizable Binary Multiplier and Binary Divider", 2015.

[7] Kuan-Hung Chen, Yuan-Sun Chu, and Yu-Min Chen, Jiun-In Guo. "A High-Speed/Low-Power Multiplier Using an Advanced Spuri- ous Power Suppression Technique", 2007, 1-4244-0921-7/07 on IEEE.

[8] Premananda B.S., Samarth S. Pai, Shashank B., Shashank S. Bhat. "Design and Implementation of 8-Bit

Vedic Multiplier", IJA- REEIE Vol. 2, Issue 12, December 2013.

[9] Ryosuke Nakamoto, Sakae Sakuraba, Takeshi Onomi, Shigeo Sato, and Koji Nakajima. "4-bit SFQ Multiplier Based on Booth Encoder", IEEE TRANSACTIONS ON APPLIED SUPERCON-DUCTIVITY, VOL. 21, NO. 3, JUNE 2011.

[10] Juny Mary Jose, Reen Paul. "A High Speed Booth Wallace Multi- plier Using Pipelining Technique", IJAREEIE Vol. 5, Special Issue 4, March 2016.

[11] Ashenden, Peter J. "VHDL standards." Design & Test of Comput- ers, IEEE 18.5 (2001): 122-123