Design and Implementation of Half Adder, Full Adder and CMOS Full Adder

Alishba Batool

Department of Electrical Engineering, Information Technology University Lahore, Pakistan

***Abstract*—In this paper, 1 bit full adder is designed in Microwind and DSCH. It is analysed using different implementation methods like AOI logic and using transmission gates. It is converted into an equivalent RC model to make the model simpler but less accurate for calculations. However, accurate results are obtained and verified using DSCH, Verilog code and Mircrowind.**

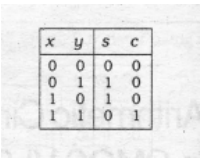
***Keywords*—Bit Adder, Half Adder, Full Adder, DSCH, Verilog**

I. INTRODUCTION

The need for counting items and adding items is almost as old as mankind itself. Addition and subtraction are the two earliest operations of mathematics. Their need has been very essential in history and is till today. These 2 arithmetic operations are in all blocks of life to add things and subtract them. Technology has made life easier and it has not left back even this small arithmetic operation. Just like addition in decimal numbers, addition also takes place in binary numbers. Simple logic gates have been used to implement adders to perform addition in binary numbers. These circuits are called bit adder circuits.

II. HALF ADDER

The smallest circuit is the half adder circuit. It takes two binary inputs and then adds them up. The output also has 2 bits, a sum bit and a carry bit. The working is such that if both the inputs are zero, both the output bits are zero because 0 + 0 = 0 and if any of the two inputs is a 1, then the sum bit is 1 and the carry bit is 0, because 1 + 0 = 1. If both the inputs are 1, then the sum bit is 0 and the carry bit is 1 because 1 + 1 = 2 and 2 is written as 10 in binary form. The 1 of the digit "10" is the but that carried just like in decimal when we add 9 + 1 the answer 10 has its 1 carried. The truth table of the half adder circuit is given below.



x	y	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 1 Truth table of Half adder (Figure 12.1 from [1])

By using this truth table in K-map, equations of sum and carry bit output is described as;

$$s = x \text{ XOR } y$$

$$c = x.y$$

A. Logic Diagram

The bit adder is implemented using simple logic gates. An AND gate and an XOR gate are used to implement the half bit adder because the sum bit is given by XOR logic and the carry bit is given by the AND logic. Circuit diagram of the half adder implemented using AND logic and XOR logic is given below.

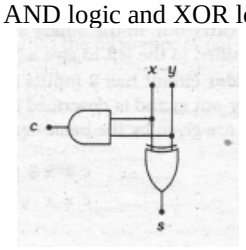


Fig. 2a Half adder logic diagram (Figure12.2 from [1])

This logic has also been designed and verified using DSCH and Vivado[2-6].

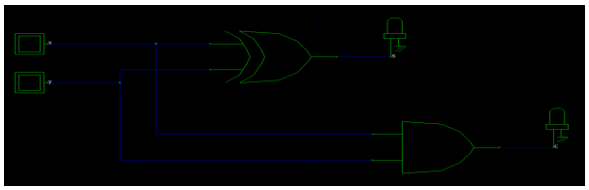


Fig. 2b Schematic diagram of Half Adder using DSCH

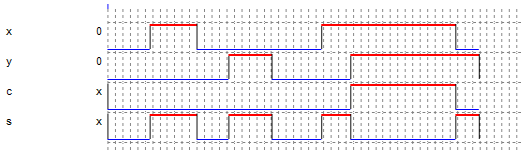


Fig. 2b Timing diagram of Half Adder using DSCH

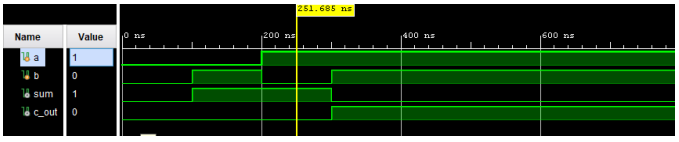


Fig. 2c Behavioural simulation of Half Adder using Verilog code

B. Alternate Half Adder Logic:

Other logics are also used to implement the half adder, for example the NAND2 logic and NOR-based design as shown below. Many possible logics exist which may be used for implementing the half adder circuit but the logic which is the most efficient is used majorly or the logic closer to the available resources is used. For example, using the NAND2 logic, we may avoid the chaining of pFET's unlike the NOR-based

network, but as the half adder circuit is a very small circuit the distance created due to the chaining does not matter much so the latter is also used. Just for example we may see for both the circuits that if we apply 1 at both the inputs whether it gives the desired answer i.e. sum bit = 0 and carry bit = 1 or not.

Output of the first NAND gate would be zero. This zero when complemented, becomes the carry bit, that is 1. As the input of the second and the third NAND gates are 0 and 1, this makes the outputs of both the NAND gates 1. Both these 1's is the input of the final NAND gate which gives the output 0 as the sum bit. The same for the NOR-based network. Both the inputs would be inverted to 0's. The output of the NOR gate which is also the carry bit gives the output as 1 because NOR of two 0's is a 1. For the first NOR gate, NOR of two 1's is zero and for the second NOR gate, NOR of a 1 and a 0 is 0, which is the sum bit. The above-mentioned equation for the half adder using AND logic and XOR logic was the simplest, the circuits shown below will have complex equations compared to the previous circuit. These outputs are also verified in DSCH and Vivado.

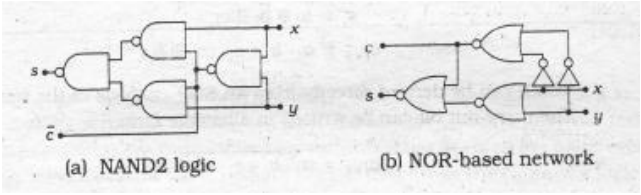


Fig. 3 Alternate Half Adder logic diagrams (Figure 12.3 from [1])

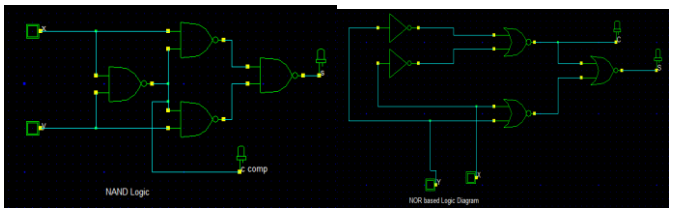


Fig. 3b DSCH designs of Alternate Half Adder Logic Networks

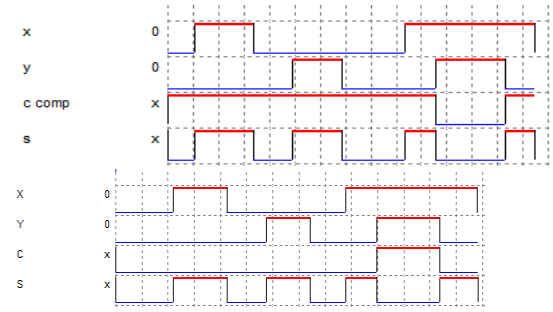
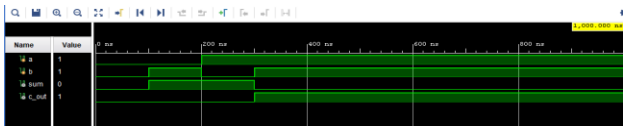


Fig. 3c Timing diagrams of Alternate Half Adder Logic Networks using DSCH



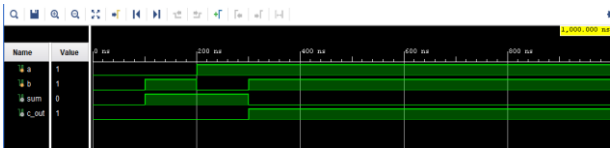
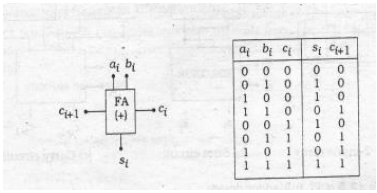


Fig. 3d Behavioural Simulations of Alternate Half Adder Logics using Verilog code

III. Full Adder

The Full Adder takes 3 inputs, a, b and Ci. It works on the same principle that if all the 3 inputs are 0, then 0 + 0 + 0 = 0, so the output is also 0, which means both the sum and the carry bit are zero. If any one of the inputs is a 1 than the sum bit would be 1 and the carry bit would be zero because 1 + 0 + 0 = 1. If any two of the inputs are 1 than the sum bit would be zero and the carry bit would be 1, because 1 + 1 + 0 = 2, which is represented as 10 in binary form. In the last if all the inputs are 1 then both sum and carry bits would be 1 because 1 + 1 + 1 = 3, which is written as 11 in binary form. The output equation of this adder can be derived using the K-map. We may make the truth table for the full adder and then use the K-map to derive the equation. It is to be noted that the K-map will give the simplest possible equation and if it is implemented in circuit form the circuit would be the smallest possible but there are several logics used to implement the full adder. Below is the truth table of the full adder.



a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Fig. 4 Full Adder symbol and truth table (Figure 12.4 from [1]

A. *Logic Diagram*

With the help of the equations derived through truth table, logic gates circuit is designed.

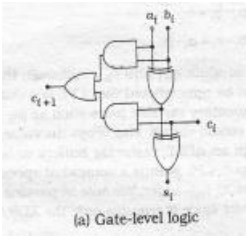


Fig. 5a Full Adder logic diagram (Figure 12.6a from [1])

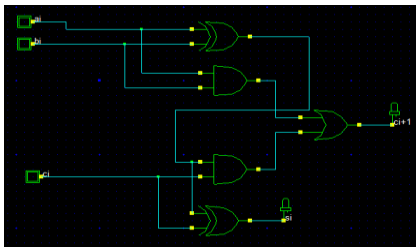


Fig. 5b DSCH logic circuit diagram

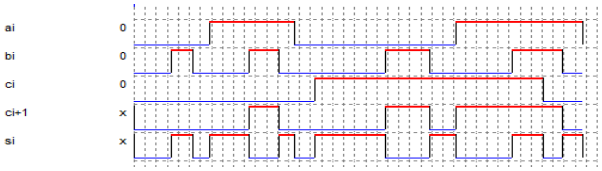


Fig. 5c Timing diagram of DSCH design

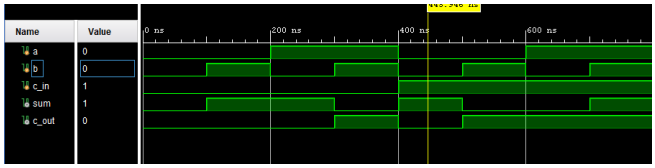


Fig. 5d Behavioural simulation using Verilog code

It can be verified that outputs of DSCH, Verilog and truth table are the same. This leads to the equations of sum and carry out, i.e.

$$s = ai \text{ XOR } bi \text{ XOR } ci$$

$$ci+1 = ai.bi + ci\ (ai \text{ XOR } bi)$$

For further verification, its optimized layout is also drawn using Microwind. In this project, 0.25-micron CMOS technology is being used. For XOR gate, the gate is converted into CMOS diagram and later into a stick diagram by using the logic that if B=0, Output= A and if B=1, Output= Inv(A). The timing diagram of the layout also shows the same output as from the equations mentioned above.

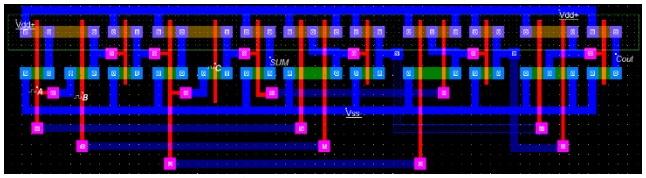


Fig. 6a Microwind layout of Full Adder

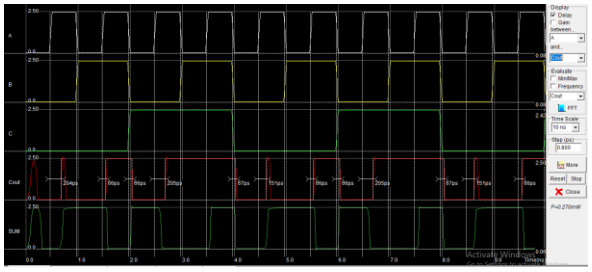


Fig. 6b Timing diagram of Full Adder in Microwind

B. AOI Full Adder

Due to the importance of full adder many implementations have been made. One of them is the AOI algorithm shown below, implemented using AND gates, OR gates and inverters.

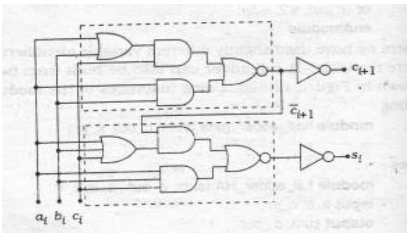


Fig. 7a AOI Full Adder logic (Figure 12.7 from [1])

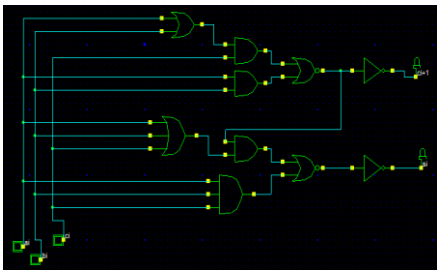


Fig. 7b AOI Full adder logic in DSCH

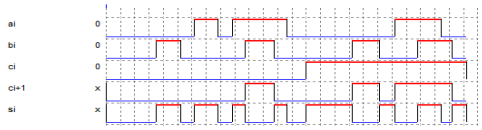


Fig. 7c Timing diagram of AOI Full Adder in DSCH

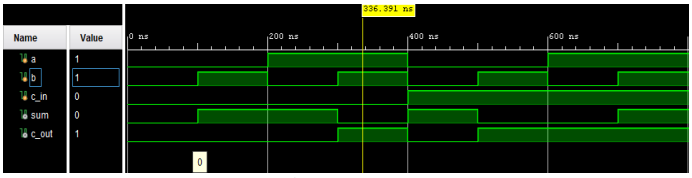


Fig. 7d Behavioural Simulation of AOI Full Adder using Verilog

The AOI logic diagram is also being drawn in Microwind. The gates are being converted into CMOS and then to stick diagrams where each gate is aligned with logic to draw an optimized layout. Hence, the outputs of sum and carry out are the same as in the truth table in Figure 5(a).

Fig 8a AOI Full Adder logic layout in Microwind

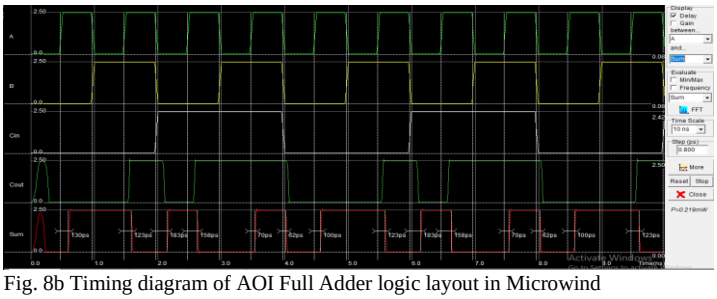


Fig. 8b Timing diagram of AOI Full Adder logic layout in Microwind

IV. CMOS FULL ADDER

Bit adders are also implemented using MOSFET's, which act as AND gates and OR gates, depending on how they are connected.

A. Carry-out nFET Logic

The truth table shown in Figure 5a, if all the input combinations are applied to the circuit and checked one by one, this circuit produces the desired output. Let's try applying all 0's as an example. The outputs of the first two OR gates would be 0. The outputs of all the AND gates would also be zero but the outputs of both the NOR gates would be 1. The 1 would be inverted by the inverters and the 2 outputs would be zero. This would give the carry bit and the sum bit, both to be 0. Basically, the upper half of the circuit generates the carry bit and the sum bit is generated by the lower half of the circuit. This can also be implemented using the nFET logic as shown below.

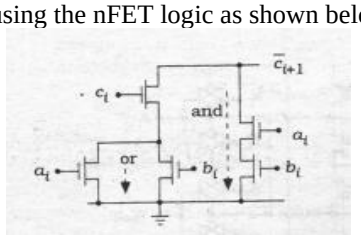


Fig. 9a Standard nFET logic (Figure 12.8a from [1])

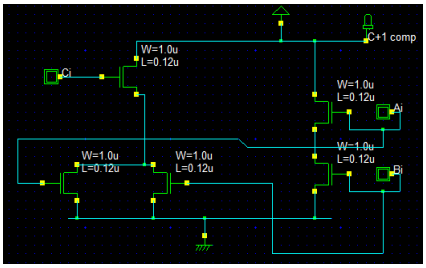


Fig. 9b nFET logic design in DSCH

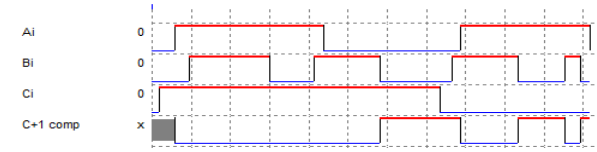


Fig. 9c Timing diagram of nFET logic in DSCH

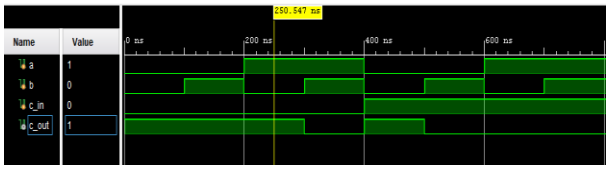


Fig. 9d Behavioural Simulation of nFET logic using Verilog code

B. Carry-out Mirror Circuit:

We may see in the AOI diagram that we need an OR gate followed by 2 AND gates and finally a NOR gate to generate the carry bit portion of the full adder The first OR gate is implemented by the 2 FET's connected in parallel. It's output and the third input go through an AND operation by the help of an FET connected above this OR gate implementation of the FET. The two FET's connected in series are an AND gate implementation which is again connected in parallel with the entire circuit on the left, which is also an OR operation. The output would be the complemented form of the carry bit so to complete the operation an inverter should be connected at the output of this circuit. The same logic is also implemented using the mirror circuit as shown below.

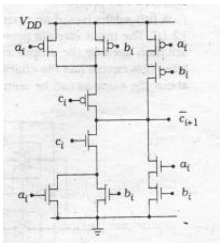


Fig. 10a Mirror circuit of Carry Out (Figure 12.8b from [1])

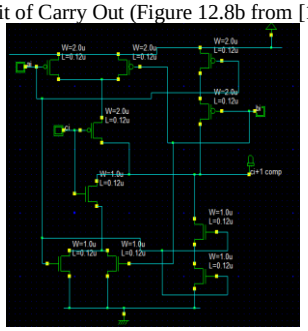


Fig. 10b Schematic diagram in DSCH

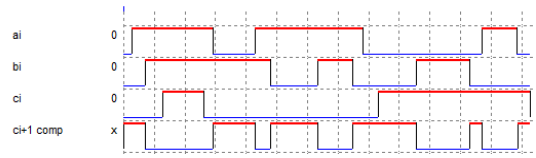


Fig. 10c Timing diagram in DSCH

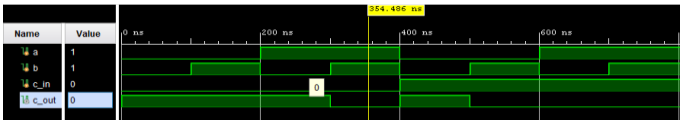


Fig. 10d Behavioural Simulation using Verilog code

To optimize the layout of the mirror circuit, Euler's path is being found first which leads towards the stick diagram. This stick diagram is then drawn in Microwind by using minimum lambda width and spacing of each metal and substrate.

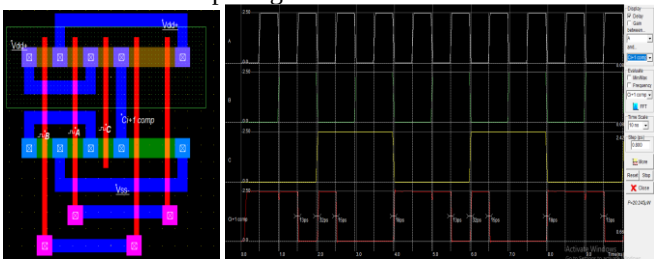


Fig. 11a & b Optimized layout timing diagram of complemented Carry out mirror circuit in Microwind

C. AOI CMOS:

This implementation includes the use of both, N-type and P-type FETs for generating the carry bit in the full adder circuit. The reason for calling it the mirror circuit is that the

lower half implementation is the same nFET implementation and a mirror circuit of it is implemented above it using pFETs. The output is still the complement of the carry bit and an inverter is needed to obtain the carry bit. Using the FET model, a mirror AOI CMOS full adder is implemented and its circuit is shown below.

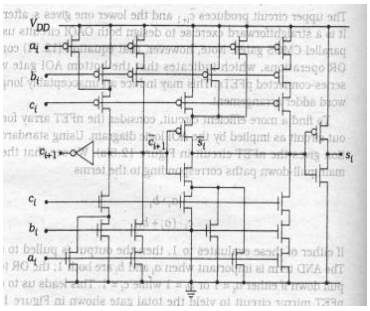


Fig. 12a Mirror AOI CMOS full adder(Figure12.9 from [1])

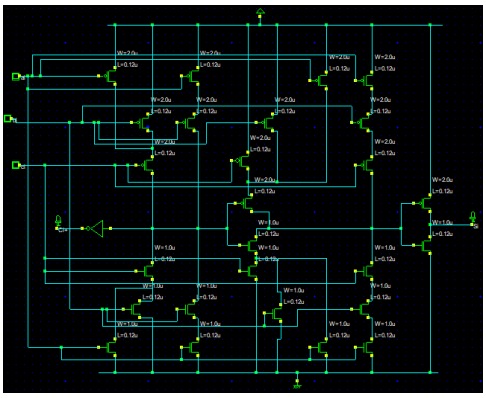


Fig. 12b Schematic diagram in DSCH

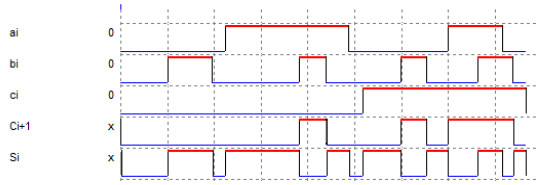


Fig. 12c Timing diagram in DSCH

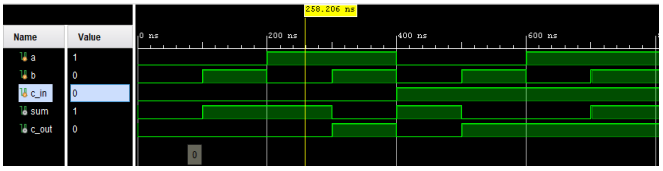


Fig. 12d Behavioural Simulation using Verilog code

V. TRANSMISSION GATES CMOS

This is much faster than a series-parallel implementation and it also facilitates the layout because of its mirrored FET arrays. Also, there is no need for inverters at the output, this circuit is complete and generates the right output. The full adder is also implemented using transmission gates as shown below in the diagram.

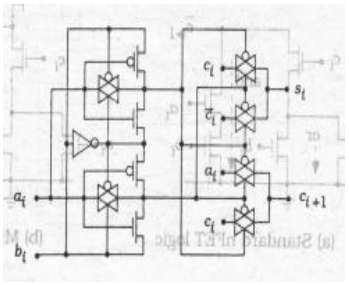


Fig. 13a Transmission gates Full Adder Circuit (Figure [1])

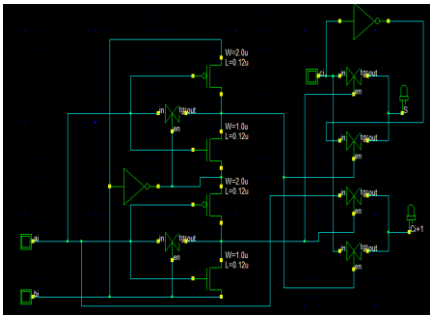


Fig. 13b Schematic diagram in DSCH

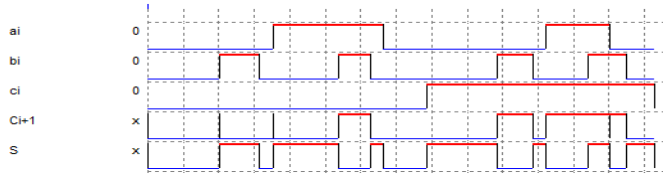


Fig. 13c Timing diagram in DSCH

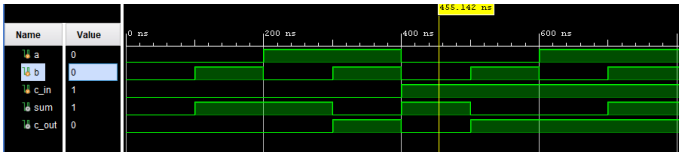


Fig. 13d Behavioural Simulation using Verilog code

The input circuit provides the desired operation which is used by the output array of the transmission gates. These transmission gates produce the sum and carry bits as the output. If the input bits are applied at the same time to the circuit, both the sum and carry bit output would be produced at the same time. This makes transmission gate implementation different from the AOI logic in which the carry bit is generated first followed by the sum bit. A transmission gate is defined as an electronic element that will selectively block or pass a signal level from the input to the output. These gates can also be called solid-state switches which comprise of a pMOS and nMOS transistor. Transmission gates are used as building blocks for logic circuitry like flip flops. A transmission gate may also isolate components from signals. When we connect PMOS and NMOS devices in parallel, this creates a CMOS switch, also known as the transmission gate. These gates are very much different from conventional logics because they are bilateral, that is inputs and outputs are interchangeable.

When the input control signal applied is 0, both the NMOS and PMOS transistors are cut-off, thus the switch is open. When the input control signals are high, both devices are biased into

conduction, thus the switch is closed. Thus, the transmission gate acts like a closed switch when the input is 1 and acts as an open switch when the input is 0. The bubble indicates it being the pFET. The output of the transmission gate relies on both, the input and control input.

Thus, the logic level value of B is defined as both A AND Control giving the expression for transmission gate output as B = A. Control input.

For the optimized version in Microwind, layout is being drawn using the logic that for XOR implementation if B= 0 Output=A and if B=1 Output= Inv(A). Here, 8 transmission gates are used in total for Sum and Carry out.

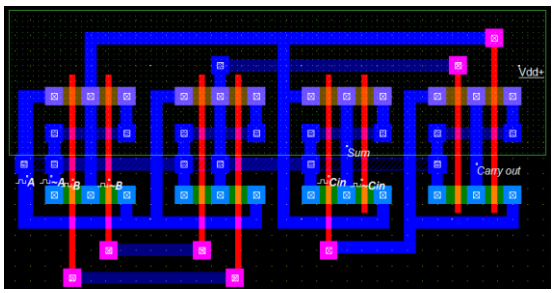


Fig 14a Transmission gates Full Adder layout in Microwind

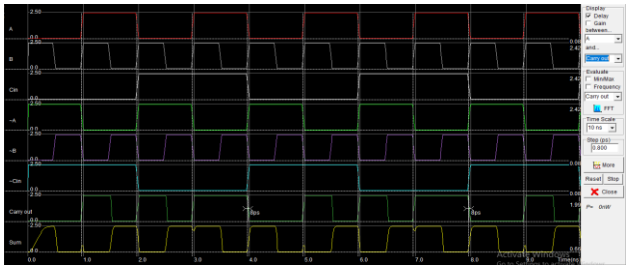


Fig. 14b Timing diagram in Microwind

REFERENCES

[1] P. Binomial, "UNIT I: FUZZY LOGIC 9 Classical logic–Multivalued logics–Fuzzy propositions–Fuzzy quantifiers. UNIT II: MATRIX THEORY 9," *VINAYAKA MISSIONS UNIVERSITY, SALEM*, p. 6.

[2] A. Cortes, I. Velez, and A. Irizar, "High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies," in *2016 Conference on design of circuits and integrated systems (DCIS)*, 2016: IEEE, pp. 1-6.

[3] T. Feist, "Vivado design suite," *White Paper*, vol. 5, p. 30, 2012.

[4] D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons, and F. Morgan, "Xilinx vivado high level synthesis: Case studies," 2014.

[5] F. Winterstein, S. Bayliss, and G. A. Constantinides, "High-level synthesis of dynamic data structures: A case study using Vivado HLS," in *2013 International conference on field-programmable technology (FPT)*, 2013: IEEE, pp. 362-365.

[6] R. Zamacola, A. G. Martínez, J. Mora, A. Otero, and E. de La Torre, "IMPRESS: automated tool for the implementation of highly flexible partial reconfigurable systems with Xilinx Vivado," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2018: IEEE, pp. 1-8.