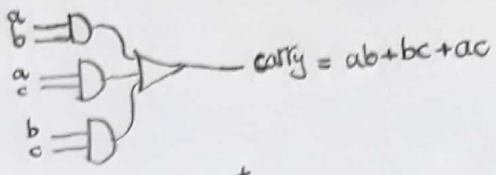
Dr. navabi	CA Report 2	Mohammad. Heydari 810197494

Q1) As we know a fulladder includes both MAJ and ODD gates as we made their in previous CA like below ↓



(1) MAJ gate

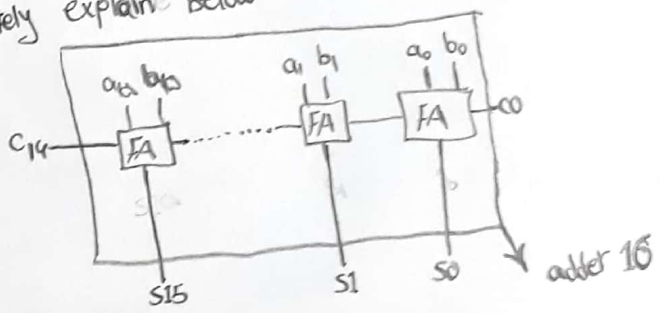
2) ODD gate

in this part we use assign statement to write systemverilog description for MAJ & ODD gates.
we also use parameter syntax to using delay values as the parameters.
but in code we have 2 gates with same input a,b,c and different output called CO for MAJ and S for ODD.

Q2) in this part we only declare ODD and MAJ gates in same module and use the #(26,22) delay for MAJ and #(22,26) delay for ODD gate as you see in mycode my module name is FA with a,b,c inputs and CO,S outputs. notice as we calculate delays of ODD and MAJ gate in previous CA I will skip that!

Q3) in this part we make a 16 bits adder using 16 instances of the Fulladder of part 2.
in my code I define a adder16 with two 16 bit inputs and an one bit input called CO and also with a 16 bits output called S and one bit output for carry out called C16. and also I have a array of wires define in this form ⇒ [16:0] c , I think there is no point about my code and the use of generate statement and the algorithm completely explain below

CO with an assign pass to an array
and at the end of array the carry out cascade to C16



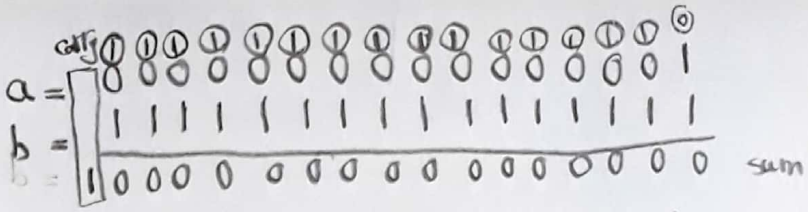
Q4) in this part we write a testbench for part 3 to test our adder. like always in module adder16_TB() I do that.

inputs ⇒ { a,b ⇒ 16 bits, cin ⇒ 1 bit } , output { sum ⇒ 16 bits, cout ⇒ 1 bit } , { as we know the output S is a 16 bits output then we ignore calculating the delay

Co ⇒ { worst case to 0 ⇒ 26×15 + 22 = 412ns, worst case to 1 ⇒ 26×16 = 416ns }

for example $C_{in}=0$

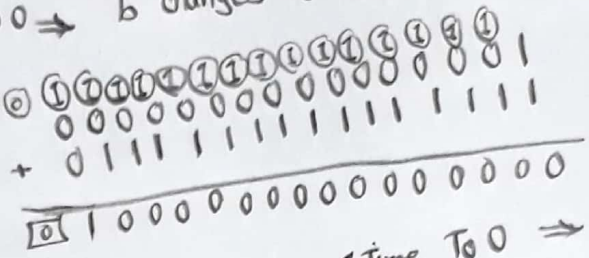
To 1 ⇒ worst case



in This example cout = 1 and we repeat 16 times with carryout = 1 → To 1

worst delay To 1 = 16 × 26 = 416ns

another example for To 0 ⇒ b changes to b = 0111111111111111



⇒ a+b ⇒ + 0111 1111 1111 1111

0 1000 0000 0000 0000

in this case we have ⇒ 15 Times To 1 + 1 Time To 0 ⇒ worst delay to 0 = 15×26+22=412

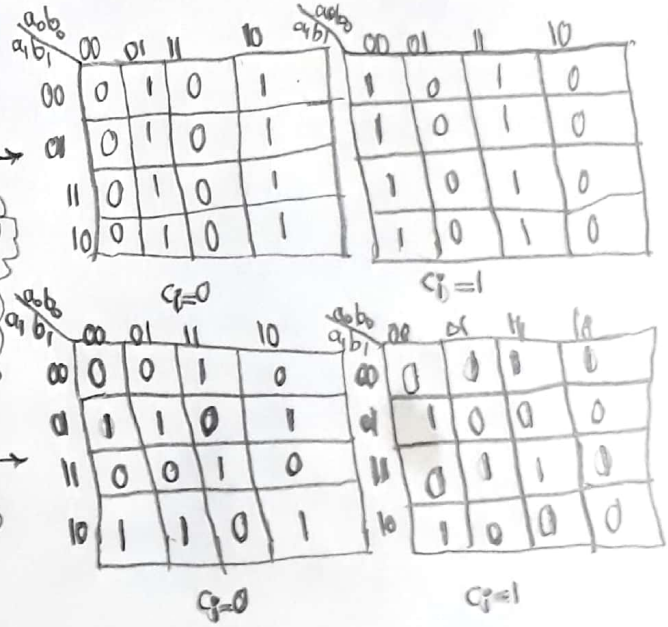
This example pointed in my code in modelsim and the screenshots will be attached

Q5)

After simplify we have ⇒ $S_0=\bar{a}_0\bar{b}_0\bar{c}+a_0\bar{b}_0\bar{c}+\bar{a}_0\bar{b}_0c+a_0b_0c$

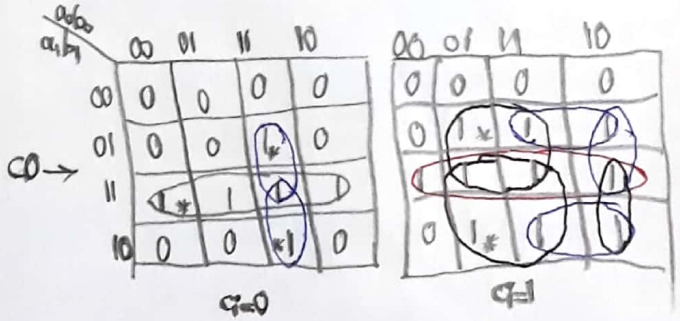
$S_1=\bar{a}_0a_1\bar{b}_1\bar{c}_i+\bar{a}_0a_1\bar{b}_0\bar{b}_1+a_0\bar{a}_1b_0\bar{b}_1+a_0a_1b_0b_1+\bar{a}_0\bar{a}_1\bar{b}_0b_1$
$+\bar{a}_0\bar{a}_1b_1\bar{c}_i+\bar{a}_1\bar{b}_0b_1\bar{c}_i+\bar{a}_1b_0\bar{b}_1c_i+a_0\bar{a}_1\bar{b}_1c_i+a_1b_0b_1c_i$
$+a_0a_1b_1c_i+\bar{a}_1\bar{b}_0b_1c_0$

$cout=a_1b_1+a_0b_0b_1+a_0b_0a_1+b_0b_1c+a_0b_1c+b_0a_1c+a_0a_1c$

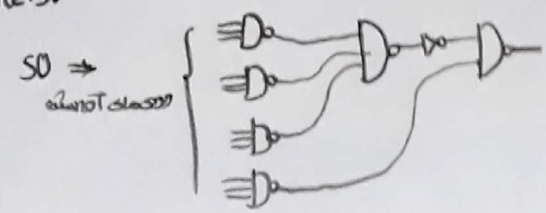


in this part I write a single assign in my code and made a 2bitsadder.

but we have to calculate the worst path that includes all of our inputs After that we choose the worst delays and use their for parameters of my assign statement

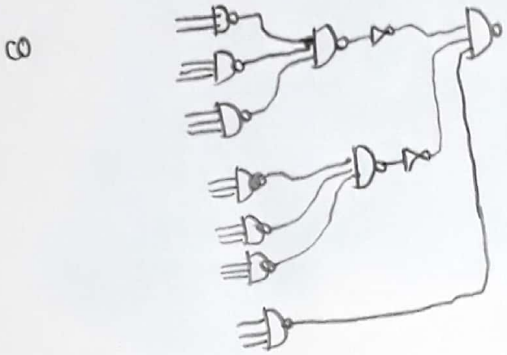


Q5) The way I choose to calculate my delay is make on all NAND circuit with inverters.
obviously we find out that the worst delays ole belong to S1 because it has more NAND gates in compare to others.

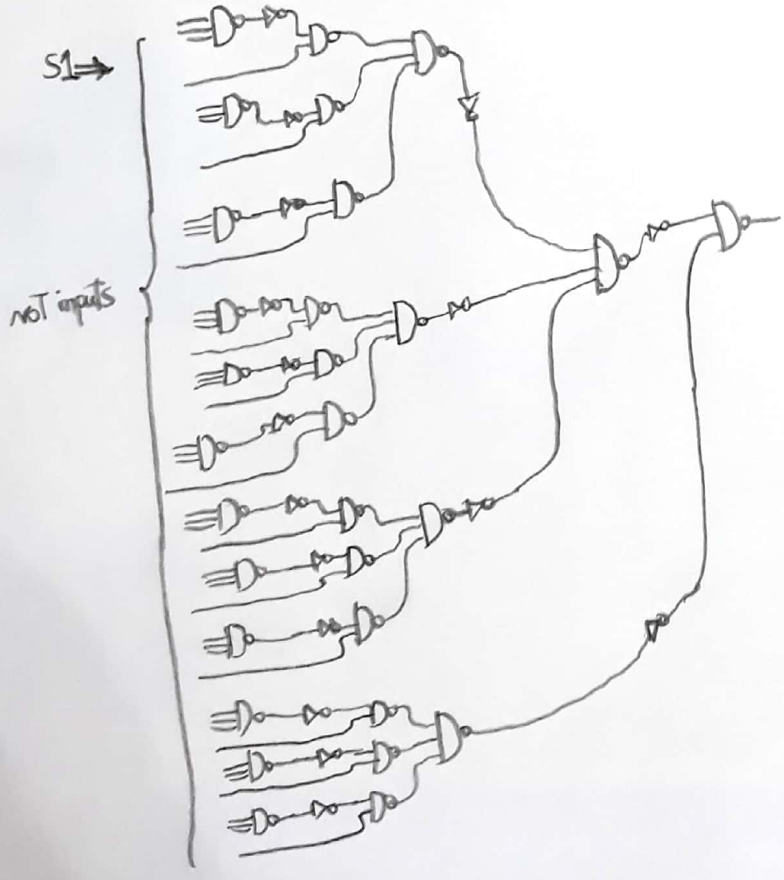


2input NAND { To0 8ns ; To1 6ns
3input NAND { To0 12ns ; To1 15ns
NOT { To1 5ns ; To0 7ns

To0 → 8+5+7+12+15 = 47ns
To1 ⇒ 7+15+15+7+10 = 54ns



{ To0 → 12+10+12+15 = 49ns
To1 ⇒ 15+8+15+15 = 53ns



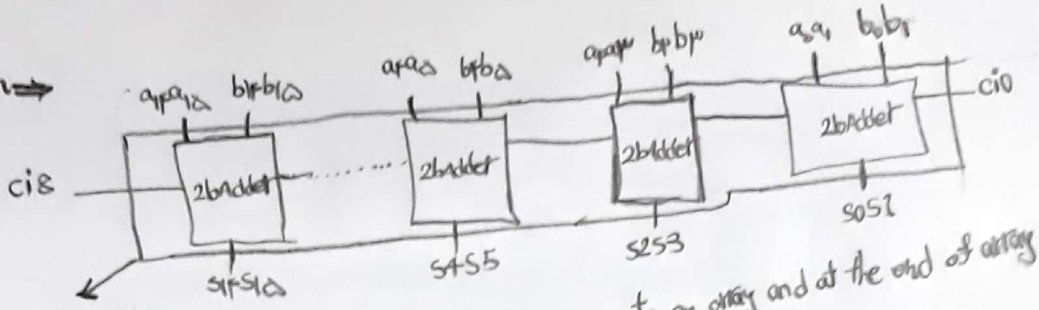
{ To0 ⇒ 8+10+12+10+12+10+8+15+10 = 95ns
To1 ⇒ 10+8+15+8+15+10+8+15+10 = 99ns

TOTAL delay = { To0 = 95ns ; To1 = 99ns

Q6) in This part we use generate statement to make a 16 bit adder with instances of 8, 2-bit adder of part 5. but in my code similarly I define a module with →

inputs { 16 bits a,b ; 1 bit cin0 } , output { 16 bits S ; 1 bit cin8 } wire → [8:0] ci

The algorithm →



adder 16 bit using 2bit adder

* cin0 with an assign pass to an array and at the end of array the carry out cascade to cin8

a,b for 1bit adders are 2bits inputs then in This module the steps of a,b,S are dual.

Q7) in this part I write a testbench to examine that the adder works correctly like always in module just adder16withadder2_TB0 I do that.

inputs → { a,b → 16 bits ; cin → 1 bit } , output { sum → 16 bits ; cout → 1 bit }

calculating delay → { T0 1 → worst → 8 * 99 = 792 ns ; T0 0 → worst → 8 * 99 = 792 ns }

The testvector and other descriptions attach completely in my code.

The screenshots will be attached in final file.

Test vector → { T0 0 → a = 0000000000000000 , b = 1111111111111111 , cin = 0 ; T0 1 → a = 1111111111111111 , b = 1111111111111010 , cin = 0 }

Q8) in this part I instantiate both adders of part 3 and part 6 in a test bench with below data.

{ adder16_withadder2 → (part 6) input a,b,cin , output → sum,cout ; adder16_withfulladder → (part 3) input a,b,cin , output sum1,cout1 }

compare → both adders have same outputs That was foreseen. However the adder which make with 16 instances of the fulladder is fater than other.

as you see the delays of adder which make with 16 FAs are less than other.