** Introduction :-

(1) To decrease the failure rate of system, it must be tolerable of faults within it & hence is the need of fault tolerance.

(2) There are two main approaches for "fault-tolerant" design :-
   (a) Static
   (b) Dynamic

(3) ↳ Static methods are robust and has higher "fault-coverage rate", but has a high area and power overhead.

↳ More economical way of fault tolerant in terms of "area and power" is "dynamic recovery" where spare modules become active on identification of faulty modules.

(4) Efficient testing methodology and proper reconfiguration unit to bypass the faulty modules in dynamic "fault tolerant".

(5) The speed of digital arithmetic processor depends on the choice of adders that's why "different digital adders" are essential.

(6) This paper proposes a philosphy for designing a fault tolerant architecture for two types of widely used adders :-
   (1) ripple carry adder (RCA)
   (2) conditional sum adder (CSA)

(7) Usually more than one module "near the error" effecting part of system becomes faulty. Hence the system must have the capability to tolerate multiple faults.

** Fault modelling :-

(1) High area overhead is a major concern in many applications like satellite or defence, where reliability is another major issue.

② Hence we have adopted dynamic recovery technique as our fault tolerant methodology. primary requirement for applying dynamic recovery is that the system must have the "structural regularity."

③ Most of the adders have the regularity in structure and are examples of "iterative logic array."

↳ Test complexities increases with the size of the ILA, To overcome this problem, a method "Truth table Augmentation" where an ILA is made c-testable.

↳ c-testable refers that "the size of the "test vector is independent of the size of the ILA.

④ Our first objective is to find out the "symmetrical modulale blocks" within the adder, identify the ILA-like structure within it and then make it c-testable, so that the size of the test vector becomes independent of size of the adder.

* Single Fault Tolerant Model :-

Ⓐ Design of a 4-bit fault tolerant ripple carry adder :-

① Ripple carry adder has itself an ILA structure with each full adder is refferred to a cell. A simple 4-bit RCA is Consisted of 4 full adders (FA). Any fault in one of FA can affect "carry or Sum or both the outputs.

② But carry lines are internal to the RCA.

③ Table I shows that only "eight test vectors" suffice to test each FA exhaustively and if single FA is faulty, then "atleast one of the two Sum outputs of the faulty cell and it's next

	A_4	B_4	A_3	B_3	A_2	B_2	A_1	B_1	A_0	B_0	S_4	S_3	S_2	S_1	S_0	C_0	C_{-1}
0	0	0	1	1	0	0	1	1	0	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	0	0
2	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	0	0
3	1	1	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	1
6	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

cell will be erroneous.

④ One redundant FA is used with four operating FAs and if any of the working FAs are found to be faulty in testing phase, the functionality of that FA is bypassed by the spare one using multiplexers (MUX) at input and output levels.

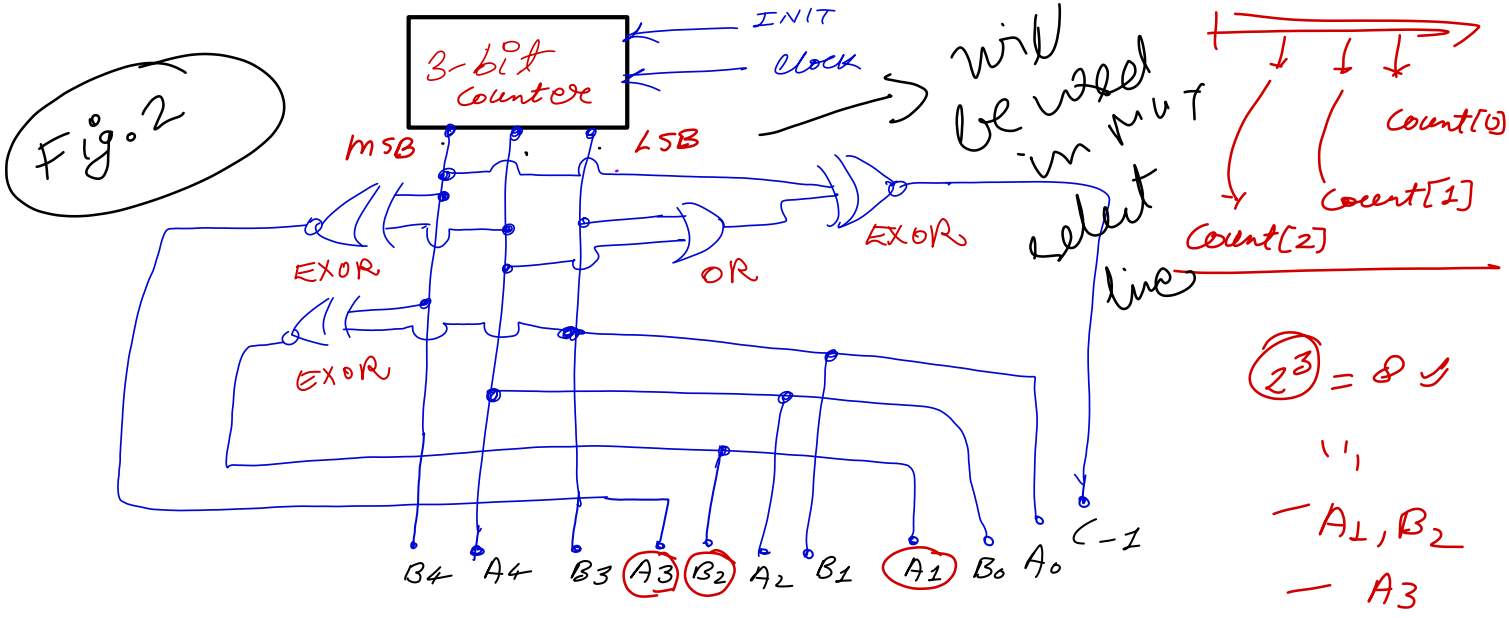
↳ The circuit diagram for fault tolerant 4-bit RCA is shown :-

* Test pattern generator circuit :-

① To generate the 8 test patterns since ($2^3 = 8$), a 3-bit binary counter is used

② The outputs of the counter are fed into combinational logic like (XORs, ANDs, ORs) to produce the actual input value (A, B, C_in) for each FA.

↳ Since the test pattern repeats every two stages, the same set of logic can be reused for all FAs in the adder.



$2^3 = 8$ ✓

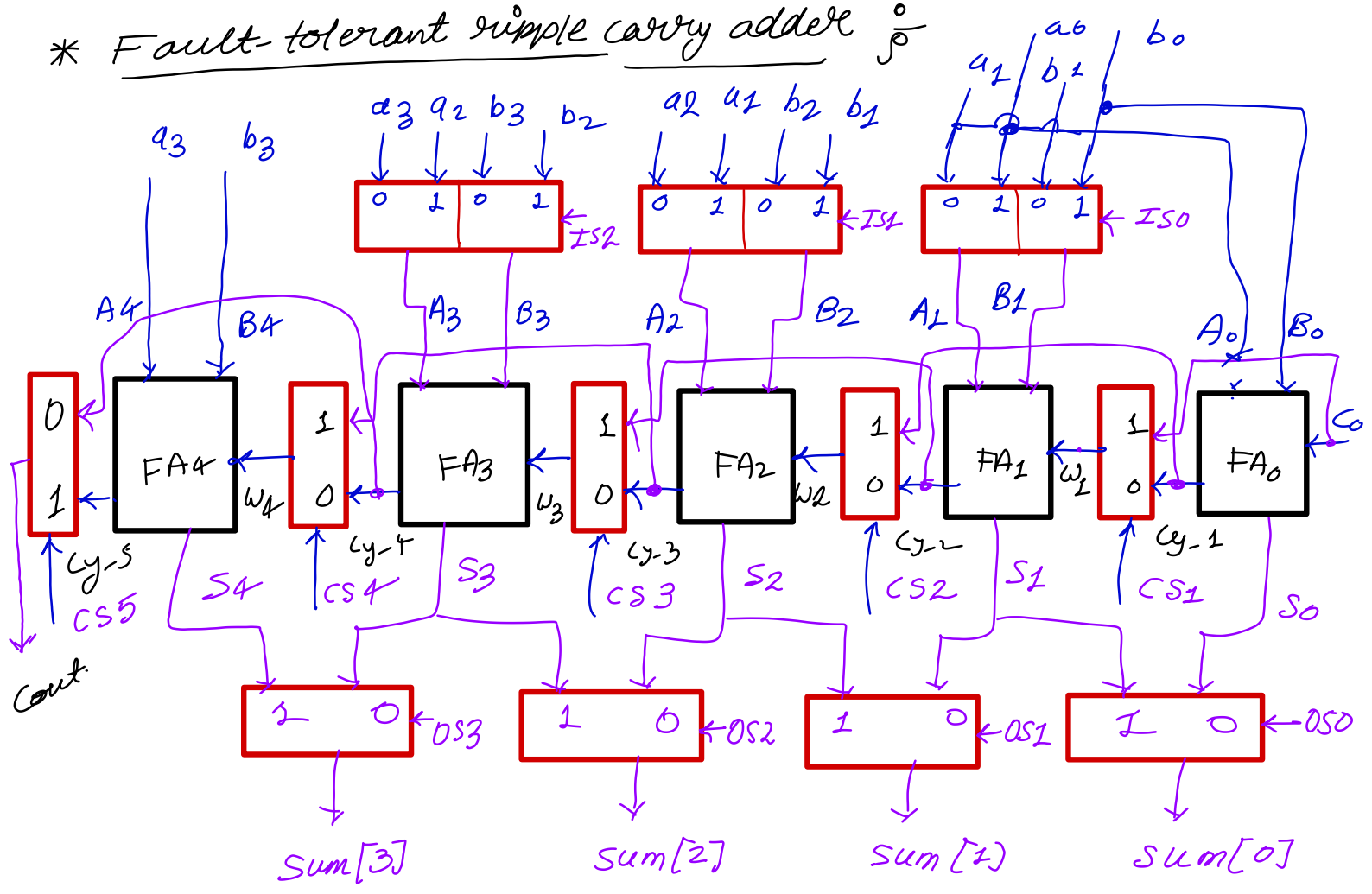
A_1, B_2

A_3

③ Carry-select Mux select lines are ANDed with the Test/Data line so that same carry is given to carry inputs of FAs.

↳ Test vectors are fed to the FAs and functionality of them is checked and compared with the desired ones generated by simple Combinational circuitary.

* Fault-tolerant ripple carry adder ☺

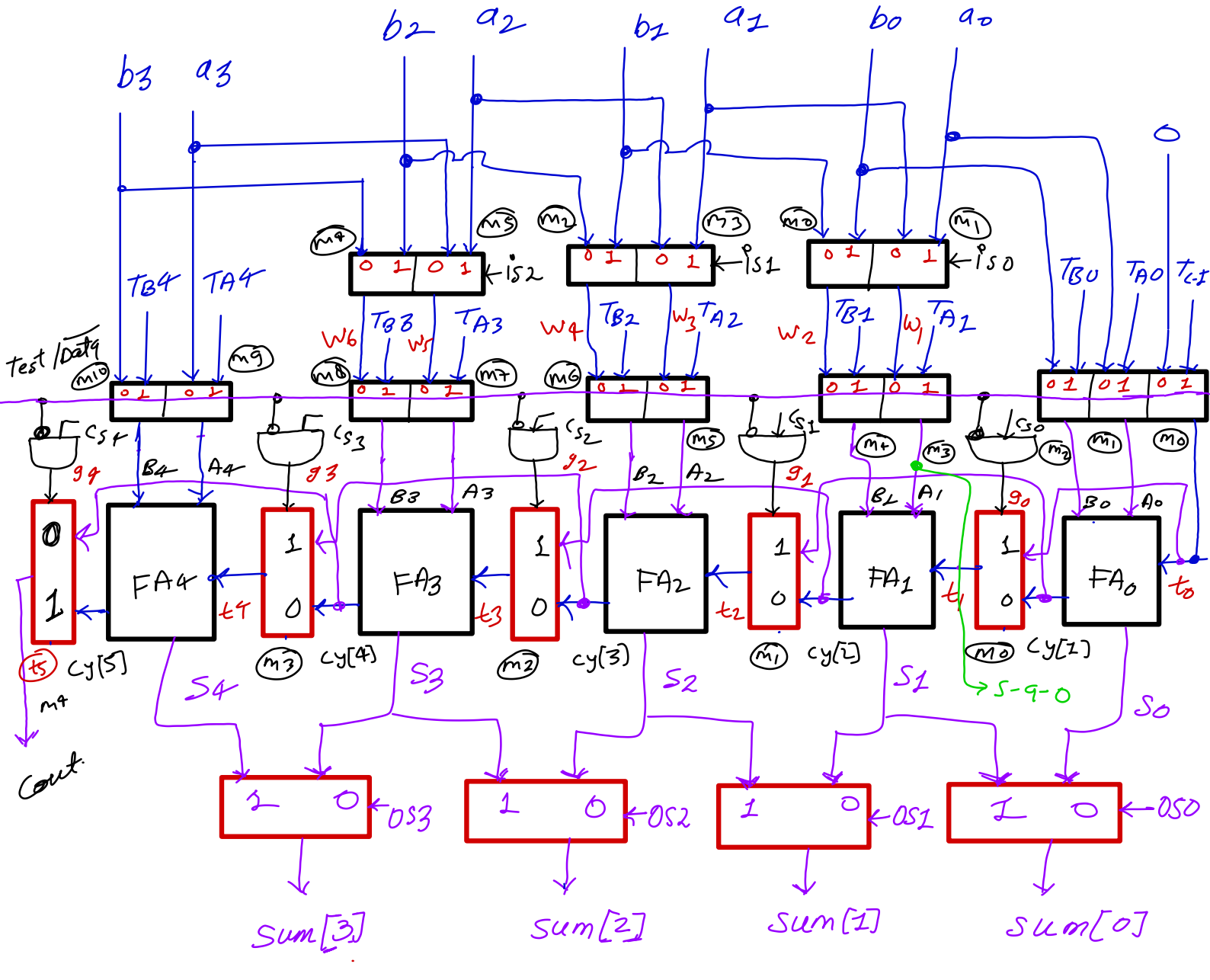


④ The desired outputs can also be stored in a look up table of size 2×8 only.

↳ The lookup table stored the value for S_0, S_1 only and can be used to compare with all other generated output.

⑤ Depending on the comparator result, the faulty FA is identified and the corresponding MUX select lines are generated using the circuitary in fig. 3 for proper input and output signals selection

⊛ modified fault tolerant ripple carry adder :-



⊛ mux-select line circuit ∘/∘

#5 a = 4'b0010 b = 4'b0001, C_0 = 1

#5 a = 4'b0010 b = 4'b0000; C_0 = 1;

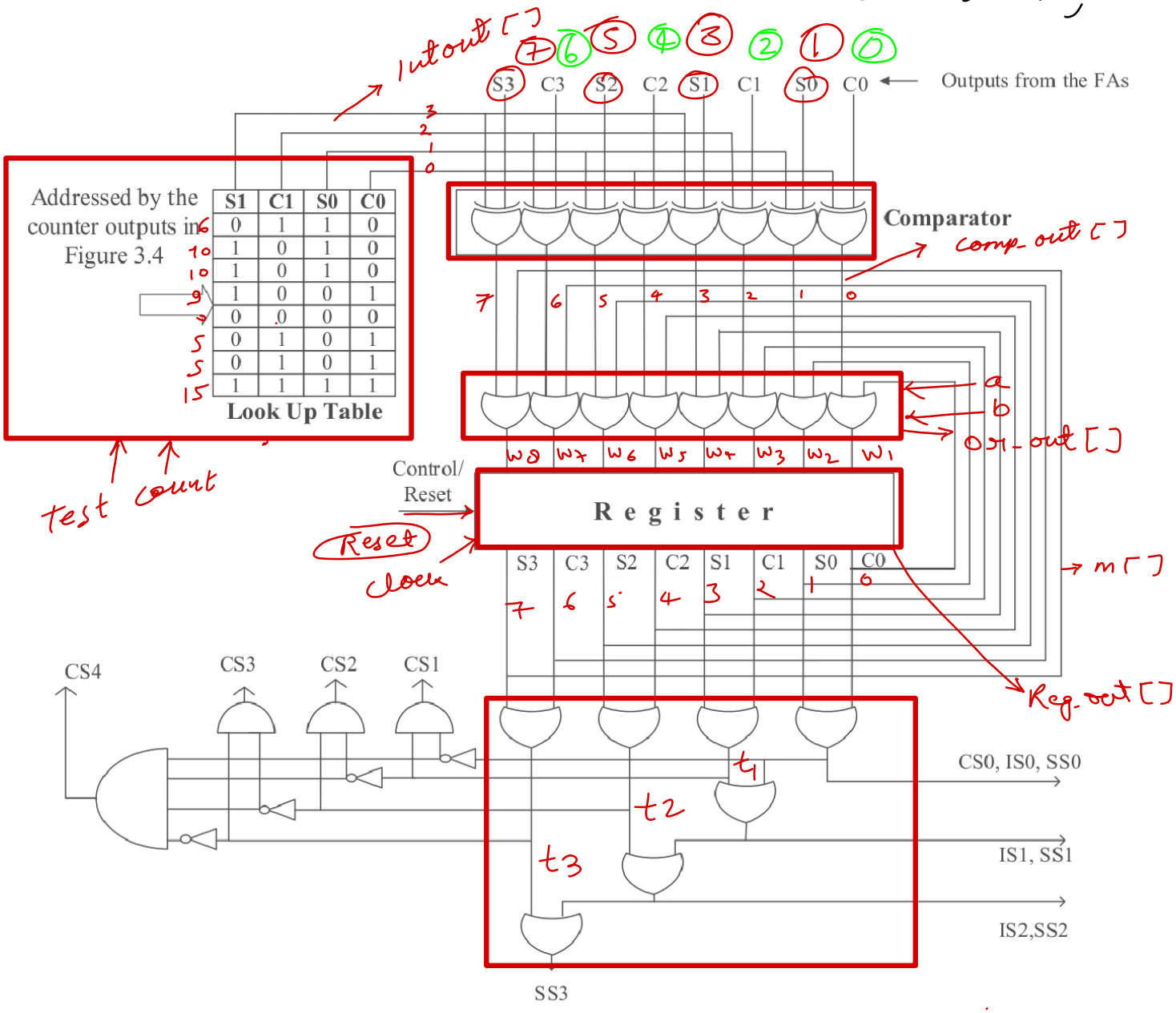


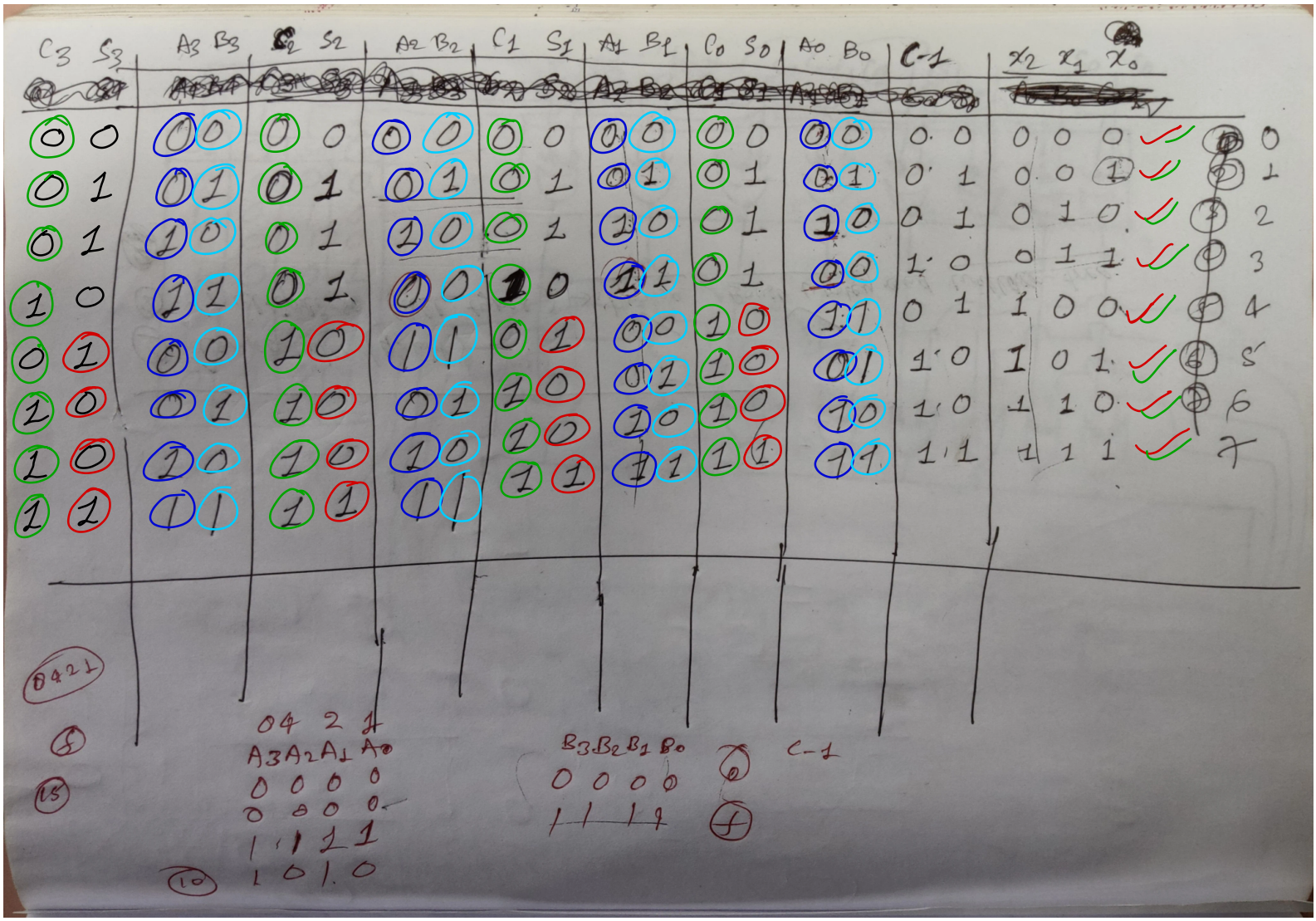
Figure 3.5: Output comparison & MUX select signals generation unit

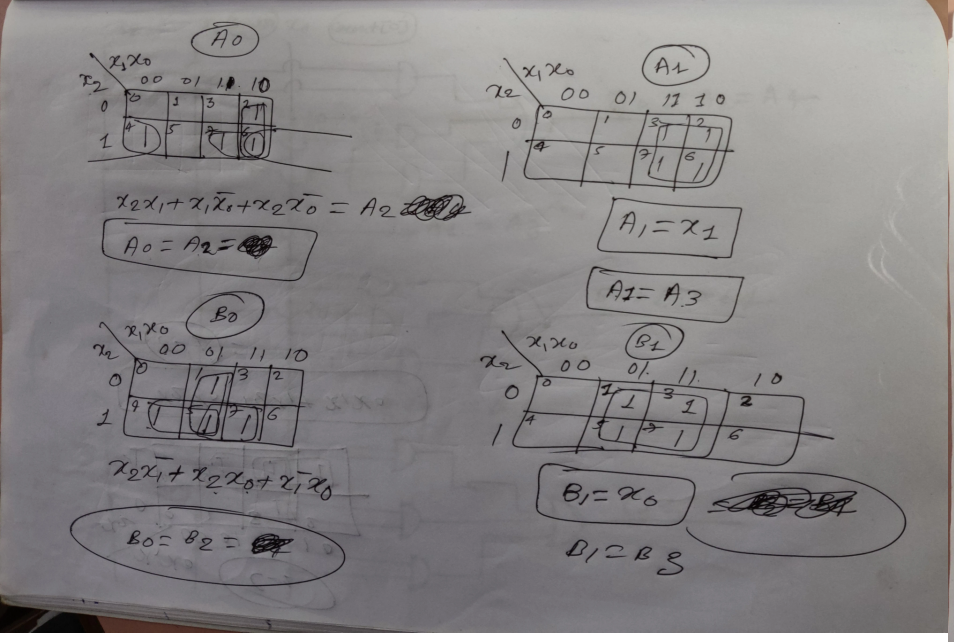
S_3	S_2	S_1	S_0	
0	1	0	1	5
1	1	1	1	15
1	1	1	1	15
1	0	1	0	10
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
1	1	1	1	15

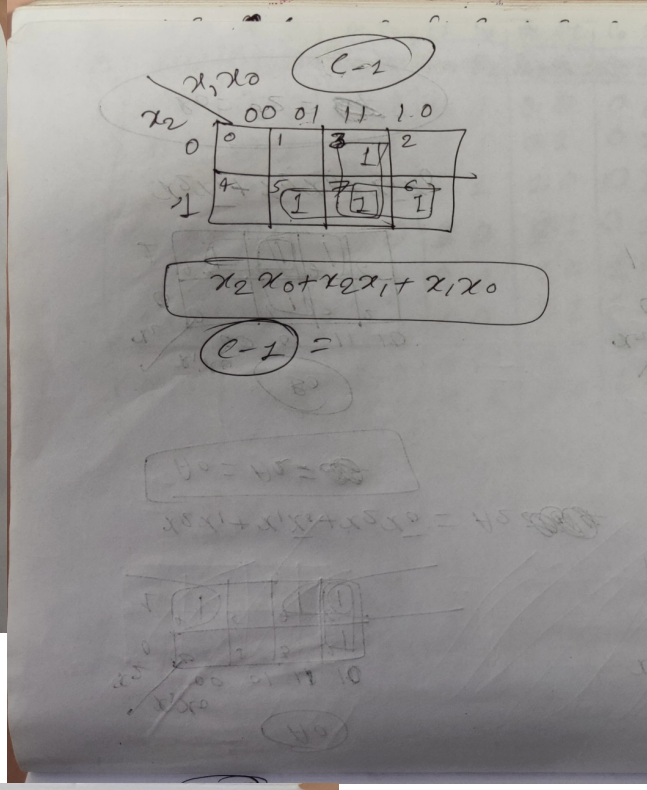
10	1	0	1	0
0	0	0	0	0
0	0	0	0	0
5	0	1	0	1
0	0	0	0	0
15	1	1	1	1
15	1	1	1	1
15	1	1	1	1

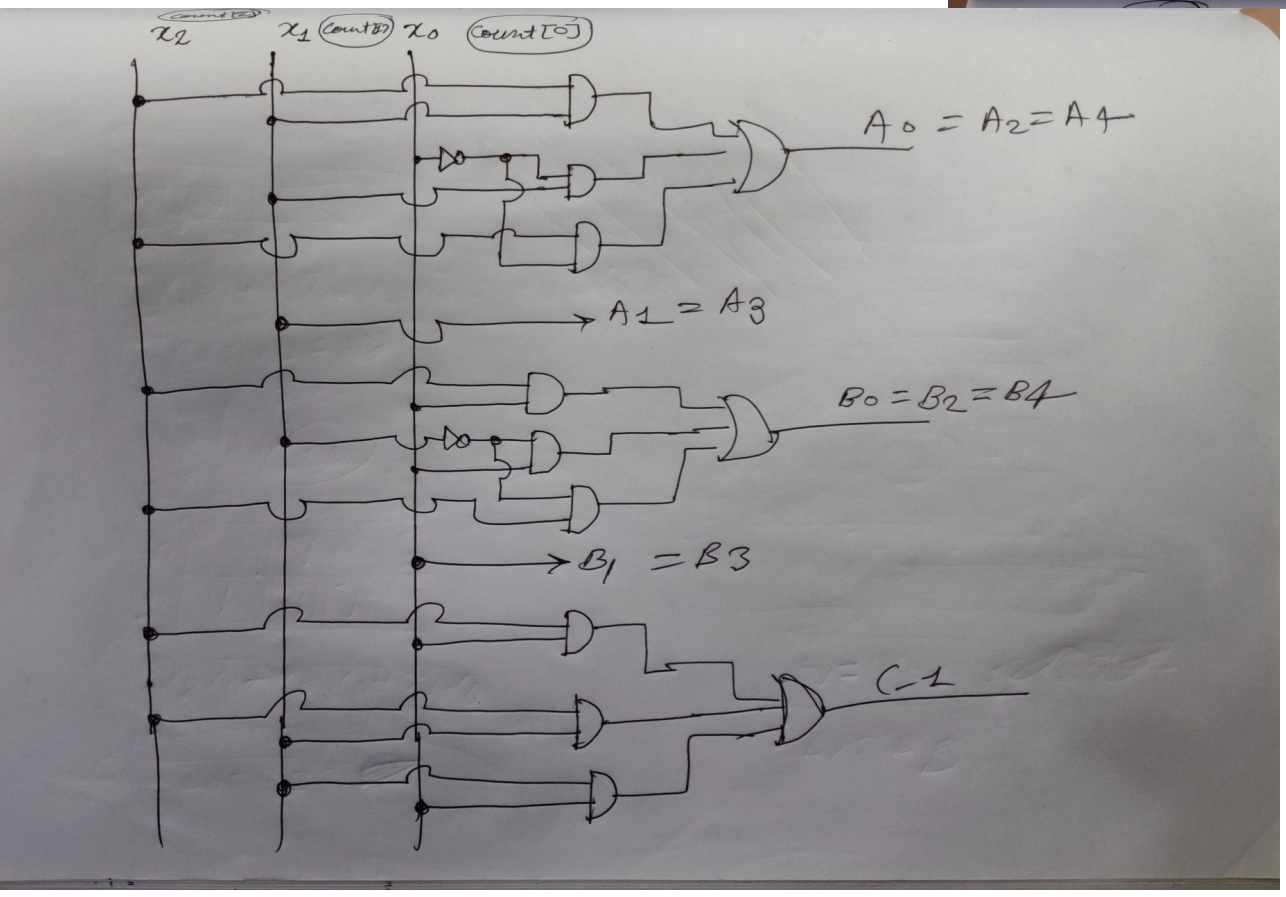
S_3	C_3	S_2	C_2	S_1	C_1	S_0	C_0
0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	0
1	0	1	0	1	0	0	1
1	0	0	1	1	0	0	1
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1

* Test pattern generation circuit :-




A0
A1
B0
B1
$x_2x_1 + x_1\bar{x}_0 + x_2\bar{x}_0 = A_2$
$A_0 = A_2 =$
$A_1 = x_1$
$A_1 = A_3$
$x_2\bar{x}_1 + x_2x_0 + x_1\bar{x}_0$
$B_0 = B_2 =$
$B_1 = x_0$
$B_1 = B_3$


C-1
$x_2x_0 + x_2x_1 + x_1x_0$
C-1 =


x_2
x_1 (count[1])
x_0 (count[0])
$A_0 = A_2 = A_4$
$A_1 = A_3$
$B_0 = B_2 = B_4$
$B_1 = B_3$
C-1

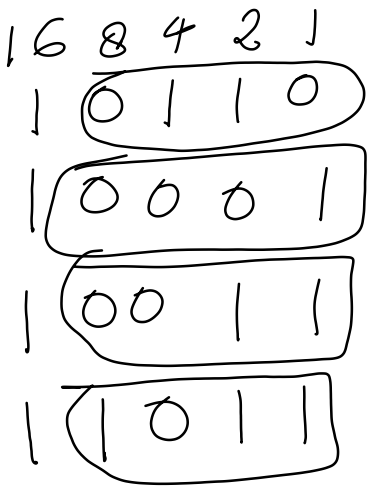
count

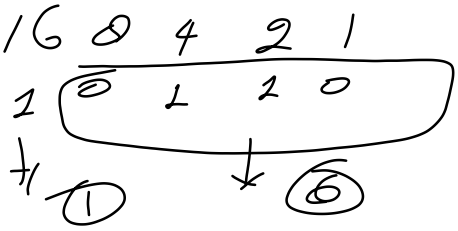
* Look up table :-

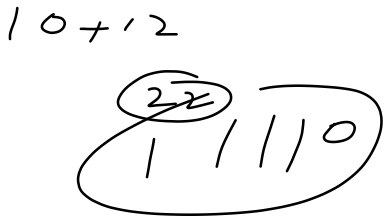
	S_1	C_1	S_0	C_0
3'b000 →	0	0	0	0
3'b001 →	1	0	1	0
3'b010 →	1	0	1	0
3'b011 →	0	1	1	0
3'b100 →	1	0	0	1
3'b101 →	0	1	0	1
3'b110 →	0	1	0	1
3'b111 →	1	1	1	1

	S_3	S_2	S_1	S_0	C_3	C_2	C_1	C_0
0	(0) 0	0	0	0	0	0	0	0
1	(15) 1	1	1	1	0	0	0	0
2	(15) 1	1	1	1	0	0	0	0
3	(5) 0	1	0	1	1	0	1	0
4	(10) 1	0	1	0	0	1	0	1
5	(0) 0	0	0	0	1	1	1	1
6	(0) 0	0	0	0	1	1	1	1
7	(15) 1	1	1	1	1	1	1	1

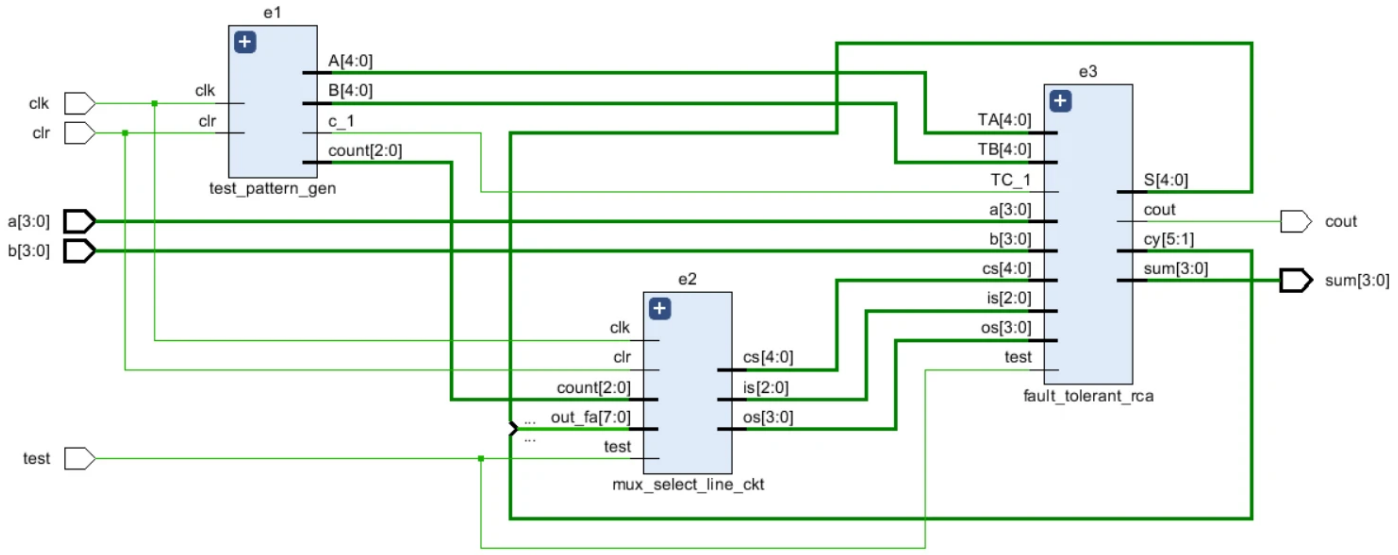
	A_3	A_2	A_1	A_0		B_3	B_2	B_1	B_0	
0	0	0	0	0	(0)	0	0	0	0	(0)
1	0	0	0	0	(0)	1	1	1	1	(15)
2	1	1	1	1	(15)	0	0	0	0	(0)
3	1	0	1	0	(10)	1	0	1	0	(10)
4	0	1	0	1	(5)	0	1	0	1	(5)
5	0	0	0	0	(0)	1	1	1	1	(15)
6	1	1	1	1	(15)	0	0	0	0	(0)
7	1	1	1	1	(15)	1	1	1	1	(15)



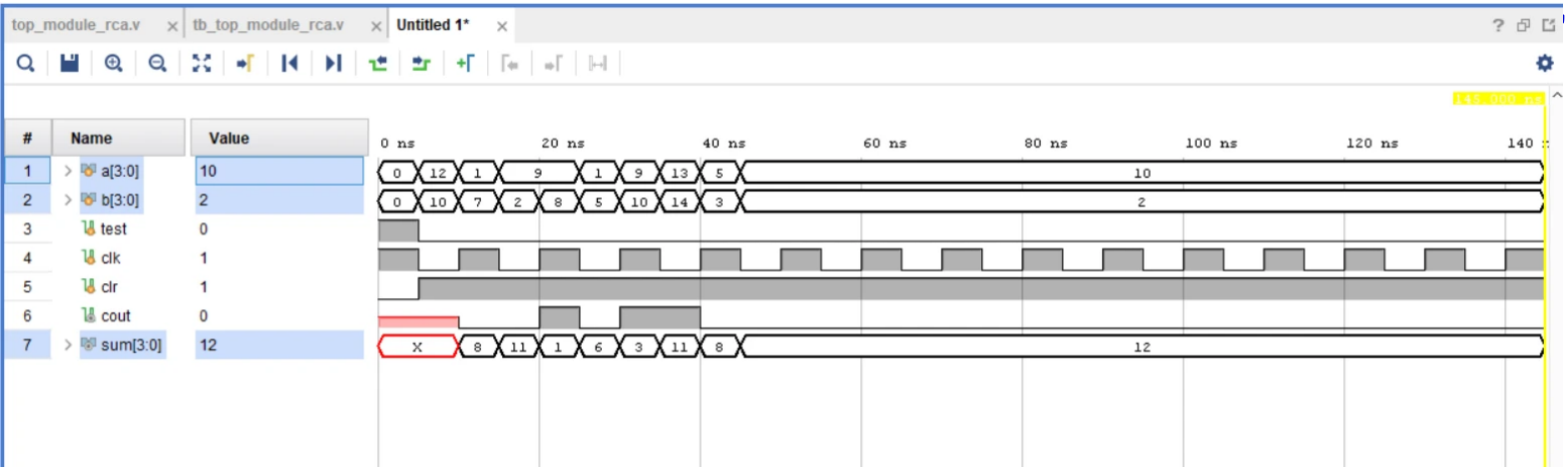




① TOP module RTL design :-



② Test bench at data mode (T=0) :-



③ Test bench at test mode (T=1) :-