Name ..... Sami ullah

ID # .....6985

Programme ..... B - Tech ( Electrical)

Subject ..... VLSI Technology

Submitted to : Engr ; Zulqarnain Abbasi

Date : 25 jun 2020

Full text: 233

(Q No 1)

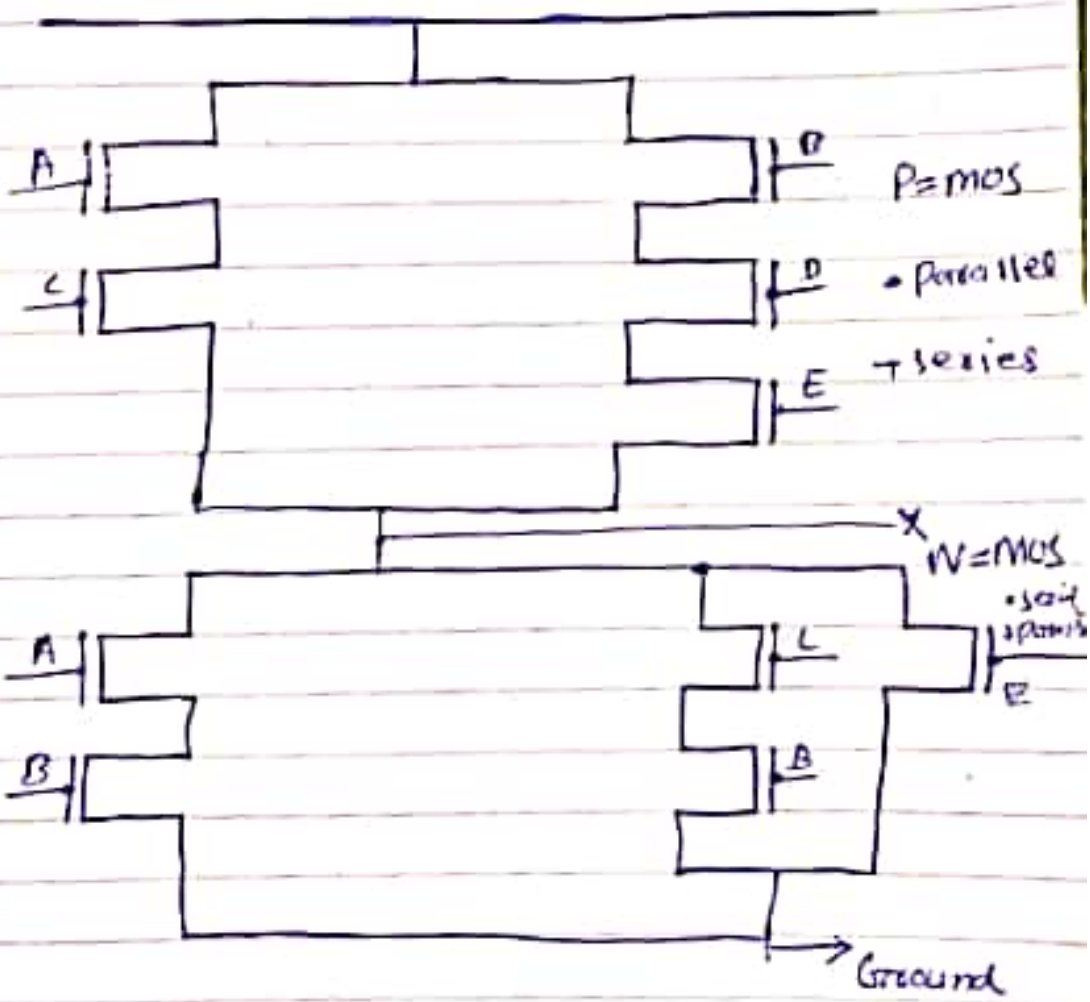
Design an area efficient layout For the cmos shown below

$F = AB + C(D)E$

* NF . Series
\+ Parallel

P = . parallel
\+ Series

$F = AB + C(.D)E$

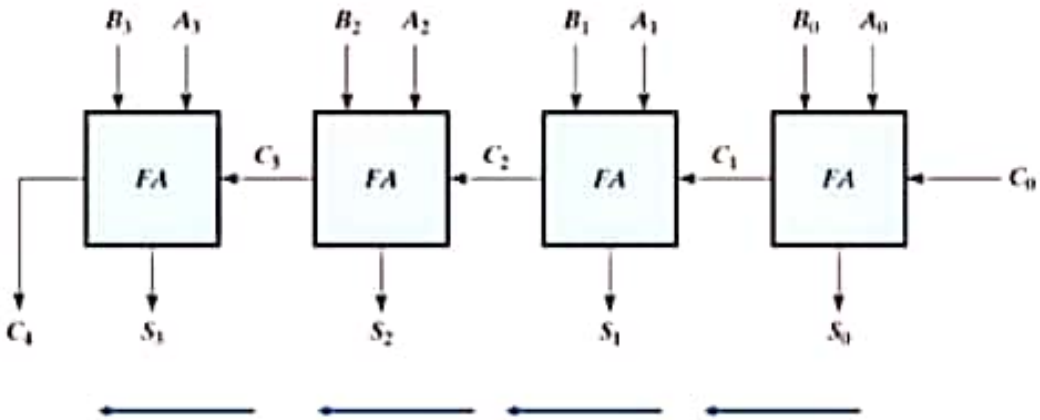


## Questions No :- 02

Anser :- The Subsystem to design of A Four bit Adder

**4-Bit Binary Adder with Fast Carry. General Description. These adders perform the addition of two 4-bit binary numbers. The sum (∑) outputs are provided for each bit and the resultant carry (C4) is obtained from the fourth bit. These adders feature full internal look ahead across all four bits.**

## Four-Bit Adder




$C_4$ is calculated last because it takes $C_0$ 8 gates to reach $C_4$
Each FA uses 2 XOR, 2 AND and 1 OR gate.
A four-bit adder uses 8 XOR, 8 AND and 4 OR gate.

Full text: 3985

# Design of an ALU Subsystem

- **Design of 4-bit adder:**

| Inputs | | | Outputs | |
|---|---|---|---|---|
| $A_k$ | $B_k$ | $C_{k-1}$ | $S_k$ | $C_k$ |
| 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 |

From the table one form of the equation is:

Sum

$S_k = H_k C_{k-1}' + H_k' C_{k-1}$

New carry

$C_k = A_k B_k + H_k C_{k-1}$

Where Half sum

$H_k = A_k' B_k + A_k B_k'$

# Design of an ALU Subsystem

- **Adder element requirement:**

Table reveals that the adder requirement may be stated as:

If $A_k = B_k$ then $S_k = C_{k-1}$

Else $S_k = C_{k-1}'$

And for the carry $C_k$

If $A_k = B_k$ then $C_k = A_k = B_k$

Else $C_k = C_{k-1}$

Thus the standar[illegible] shown in the figure 6.11

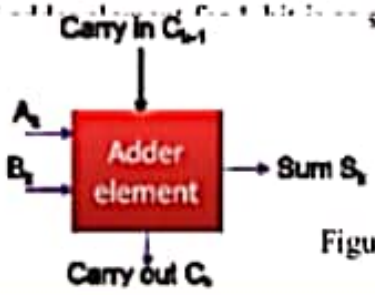



Figure 6.11: Adder element

Full text: 3985

Questions No :- 03 ( Part :- A )

Anser :- Some Architectural Issues in Vlsi Design :-

In all design process ,a logical and systematic approach is essential. This is particularly so in the case of the design of aVLSI system which could otherwise take so long as to render the whole system obsolete before it is off the drawing board.

But now some sensible concepts applied in larger system design requirements.

Guideline set as follows:

1. Define the requirements (properly & carefully).

2. Partition the overall architecture into appropriate subsystem.

3. Consider communication paths carefully in order to develop sensible interrelationships between subsystem.

4. Draw a floor plan of how the system is to map onto the silicon (and alternate between 2,3 and 4 as necessary).

5. Aim for regular structures so that design is largely a matter of replication.

6. Draw suitable (stick or symbolic) diagrams of the leaf_cells of the subsystems.

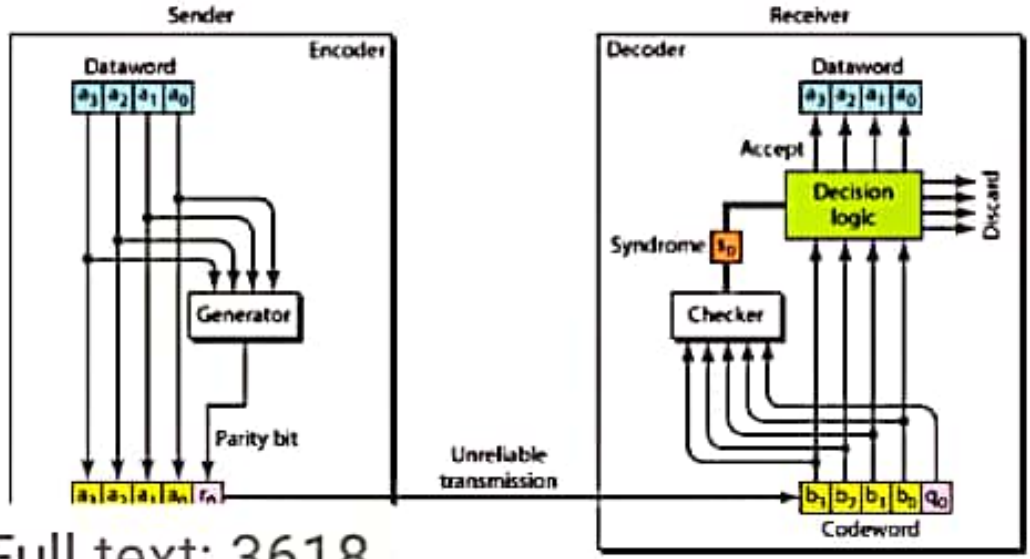
7. Convert each cell to layout.

## Q No :- 03 (Part :- B )

## Anser :- Simple Parity-Check Code:

The simple parity-check code is the most familiar error-detecting code. In this code, a k-bit data word is changed to an n-bit code word where n = k + 1. The extra bit, called the parity bit, is selected to make the total number of 1s in the code word even. Although some implementations specify an odd number of 1s. The minimum Hamming distance for this category is dmin =2, which means that the code is a single-bit error-detecting code and it cannot correct any error.

The following figure shows possible structure of an encoder (at the sender) and a decoder (at the receiver).



The encoder uses a generator that takes a copy of a 4-bit data word (a0, a1, a2 and a3) and generates a parity bit r0. The data word bits and the parity bit create the 5-bit code word. The parity bit that is added makes the number of 1s in the code word even.This is normally done by adding the 4 bits of the data word (modulo-2).

. r0=a3+a2+a1+a0 (modulo-2)

For example, the sender sends the data word 1011. The code word created from this data word is 10111, which is sent to the receiver. We examine five cases:

1. No error occurs; the received code word is 10111. The syndrome is 0. The data word 1011 is created.

2. One single-bit error changes a1. The received code word is 10011. The syndrome is 1. No data word is created.

3. One single-bit error changes r0. The received code word is 10110. The syndrome is 1. No data word is created. Note that although none of the data word bits are corrupted, no data word is created because the code is not sophisticated enough to show the position of the corrupted bit.

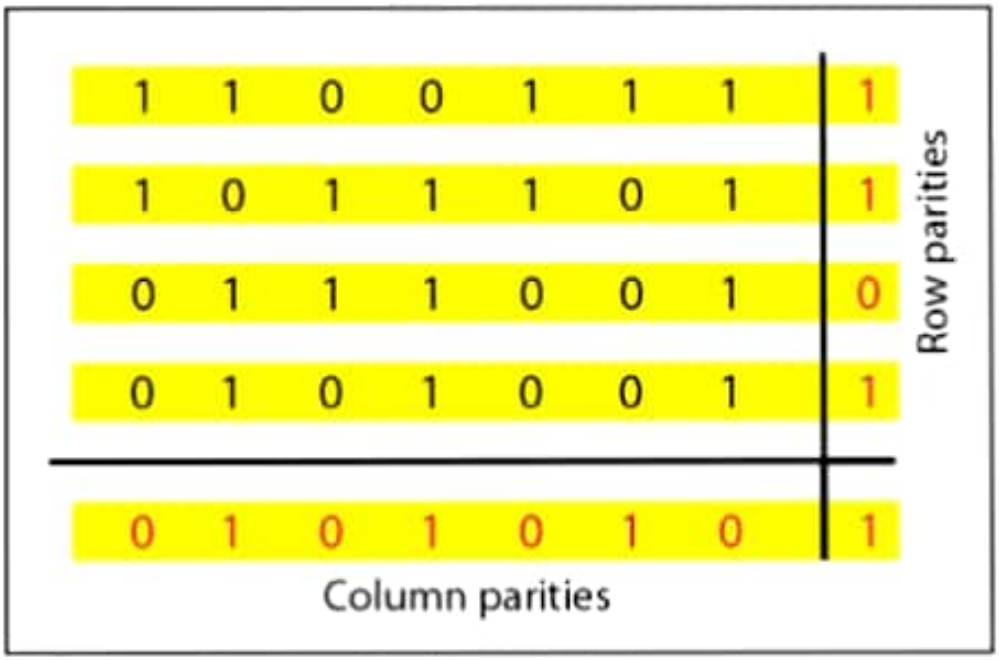
4. An error changes r0 and a second error changes a3. The received code word is 00110. The syndrome is O. The data word 0011 is created at the receiver. Note that here the data word is wrongly created due to the

number of errors. The errors cancel each other out and give the syndrome a value of

5. Three bits-a3, a2, and a1-are changed by errors. The received code word is 01011. The syndrome is 1. The data word is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

A better approach is the two-dimensional parity check. In this method, the data word is organized in a table (rows and columns). In the following figure, the data to be sent, five

7-bit bytes, are put in separate rows. For each row and each column, 1 parity-check bit is calculated. The whole table is then sent to the receiver, which finds the syndrome for each row and each column.




a. Design of row and column parities

Full text: 3618

# Questions No :- 04

**Anser :-** The inverter have always lower limit of the power depends on the sum of the threshold voltages of the nMOS and Vdd .

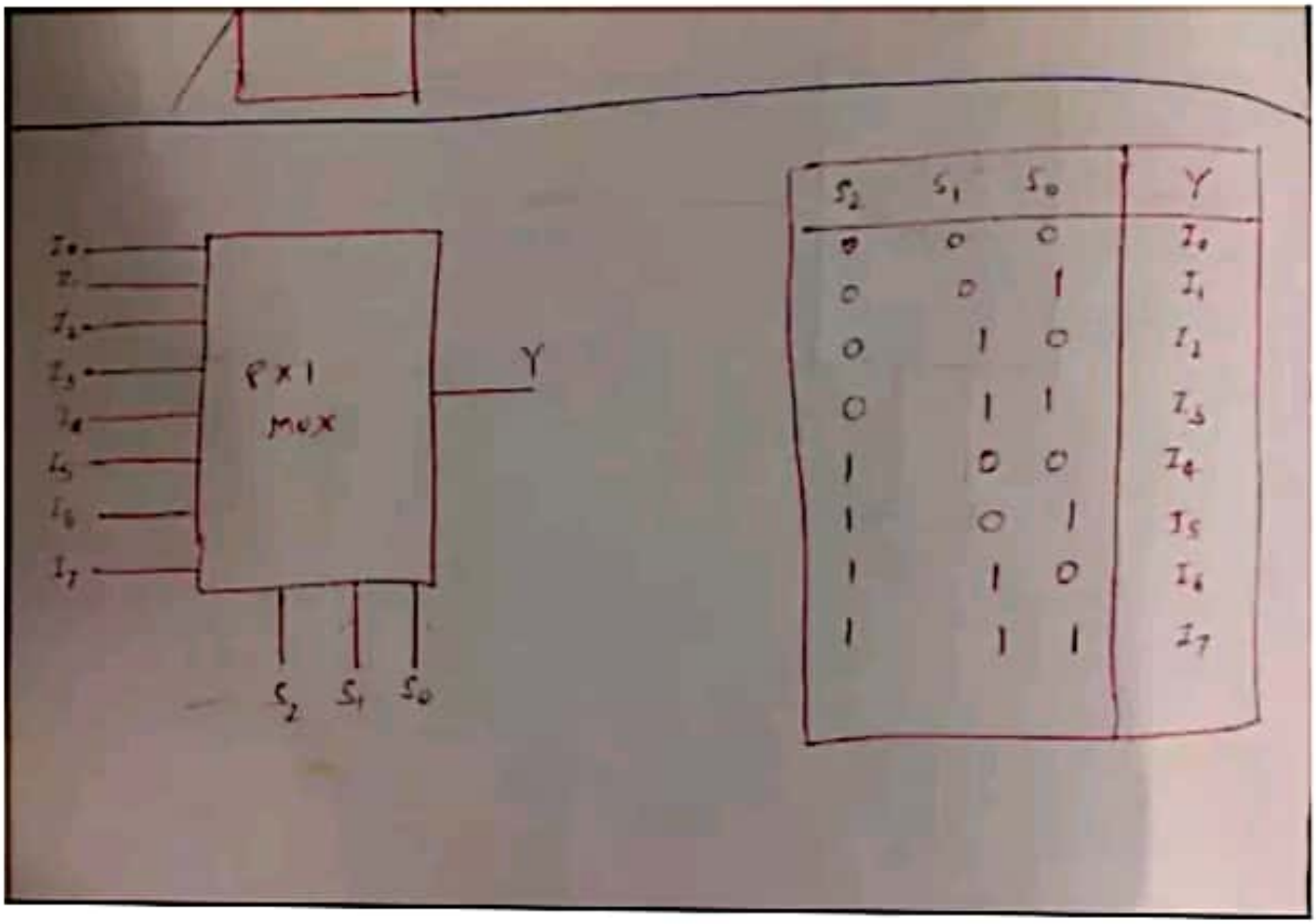
- Logic function is implemented by pull-down network only.
- Full swing outputs (V OL = GND and V OH = VDD).
- Non-ratioed.
- Faster switching speeds.

Full text: 4268

## Questions No :- 05

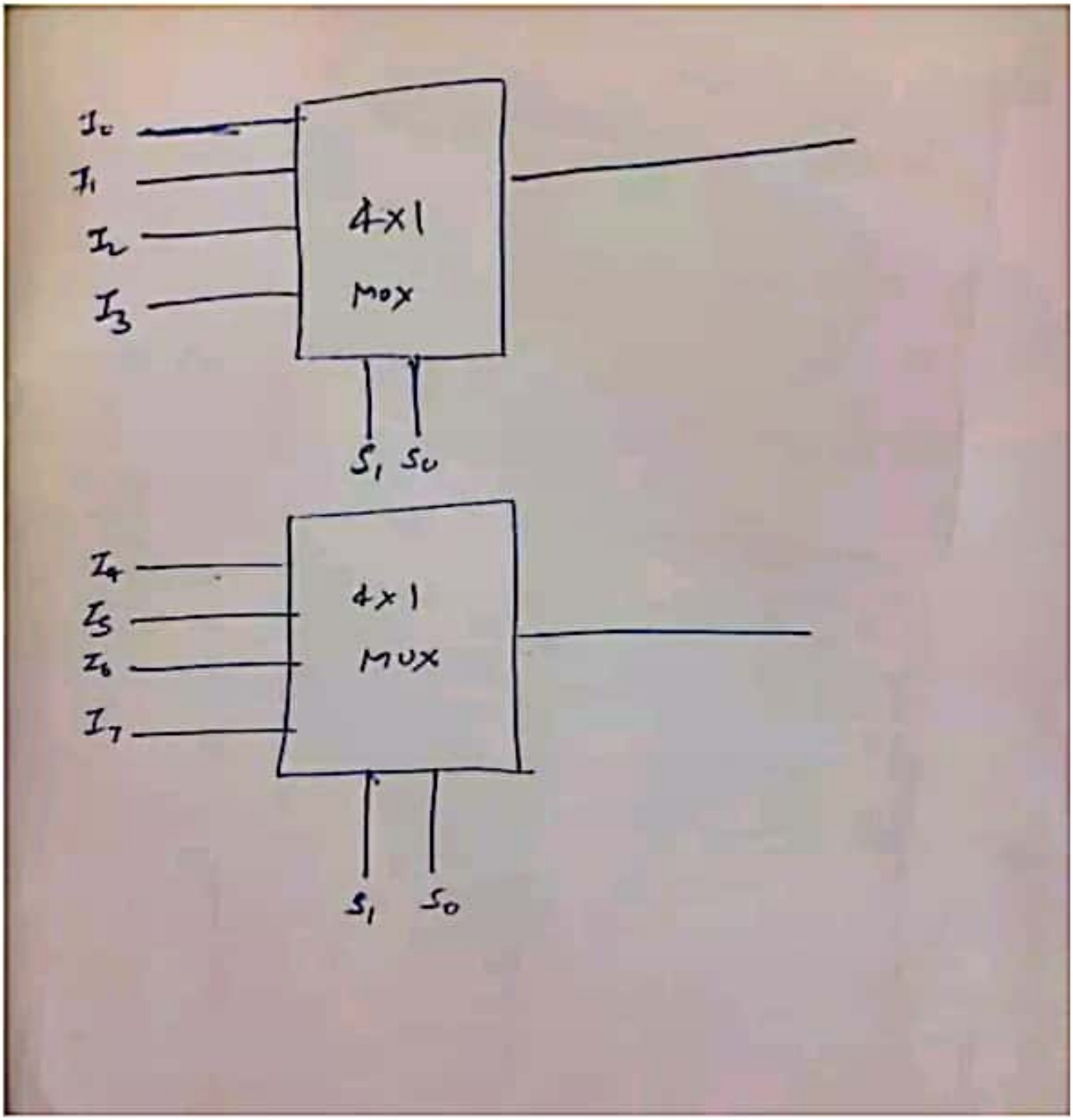
Anser :-

This is an 8X1 MUX with inputs I0,I1,I2,I3,I4,I5,I6,I7 , Y as output and S2, S1, S0 as selection lines. The output will depend upon the combination of S2,S1 & S0 as shown in the truth table.



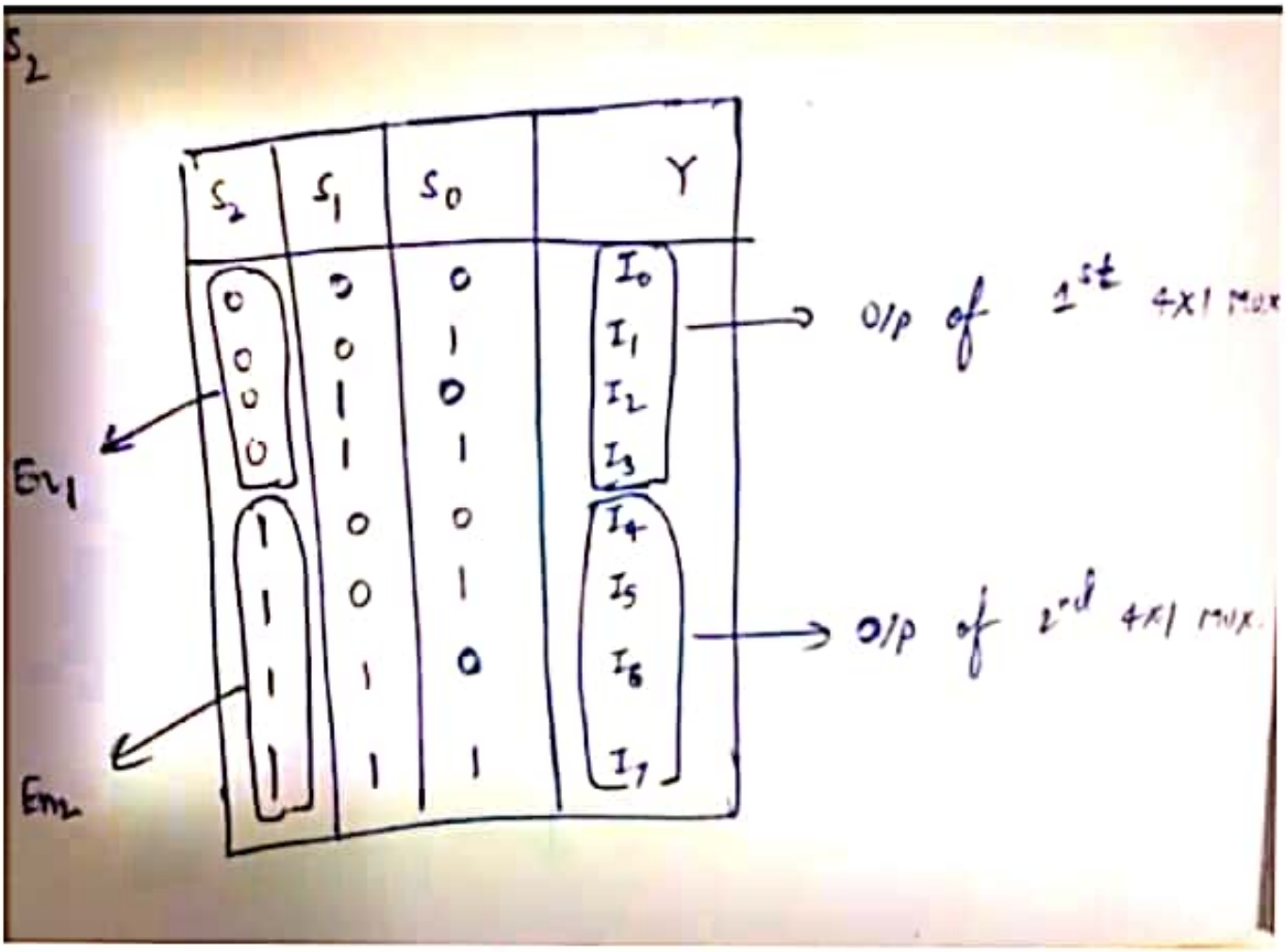

| $S_2$ | $S_1$ | $S_0$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | $I_0$ |
| 0 | 0 | 1 | $I_1$ |
| 0 | 1 | 0 | $I_2$ |
| 0 | 1 | 1 | $I_3$ |
| 1 | 0 | 0 | $I_4$ |
| 1 | 0 | 1 | $I_5$ |
| 1 | 1 | 0 | $I_6$ |
| 1 | 1 | 1 | $I_7$ |

Now, to implement this 8X1 MUX using 4X1 MUX we need two 4X1 MUX, since to take 8 inputs atleast two 4X1 MUX required, 4 inputs on each of the muxes having selection lines S1 & S0 as shown in the figure.

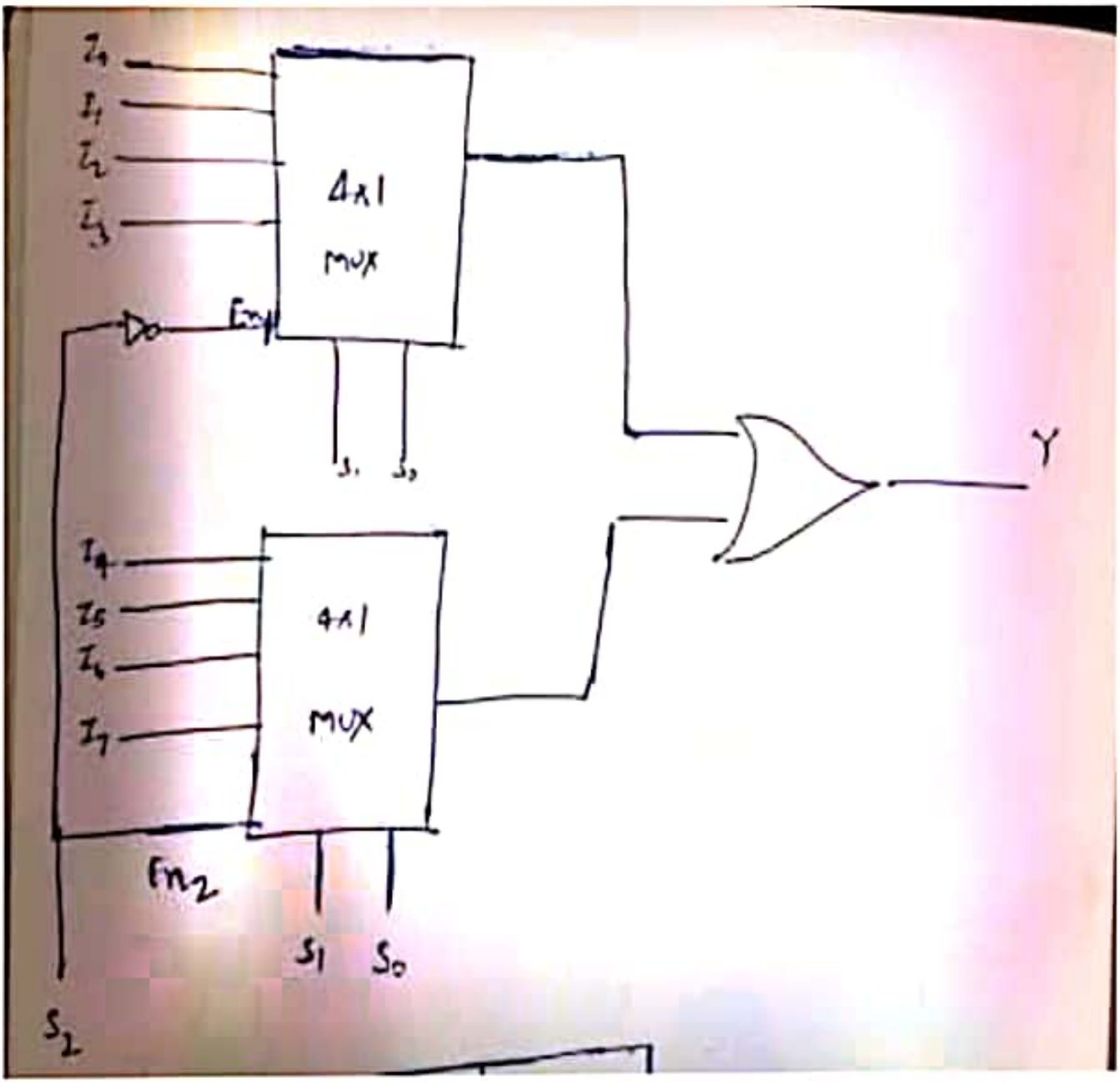



Now, as there are 3 selection lines in 8X1 MUX namely S2, S1, S0, we also need one additional selection line S2. So, question is, where to add that selection line?, as there will be only two selection lines in 4X1 MUX. Lets have a look on the truth table given below.

---

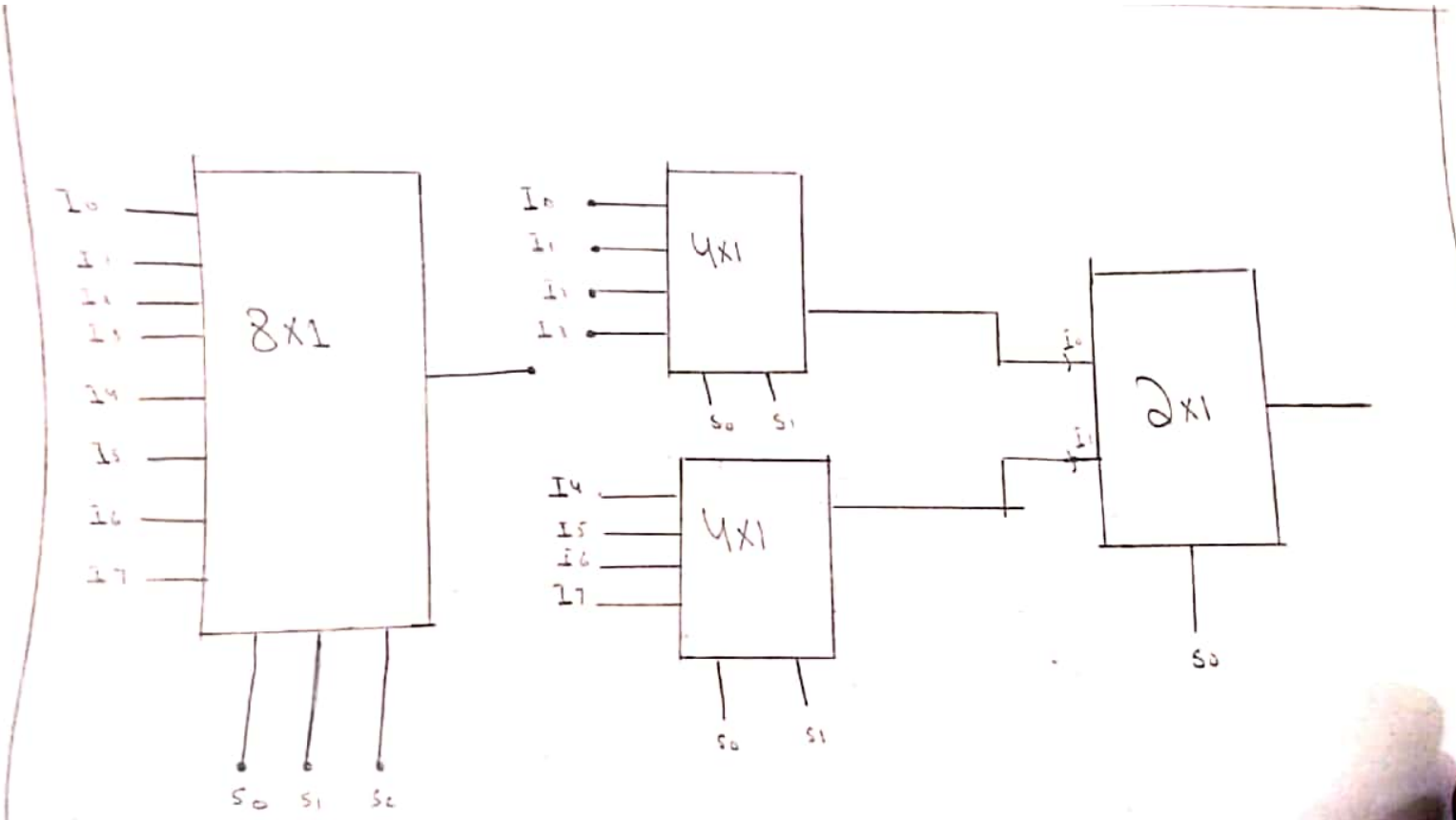


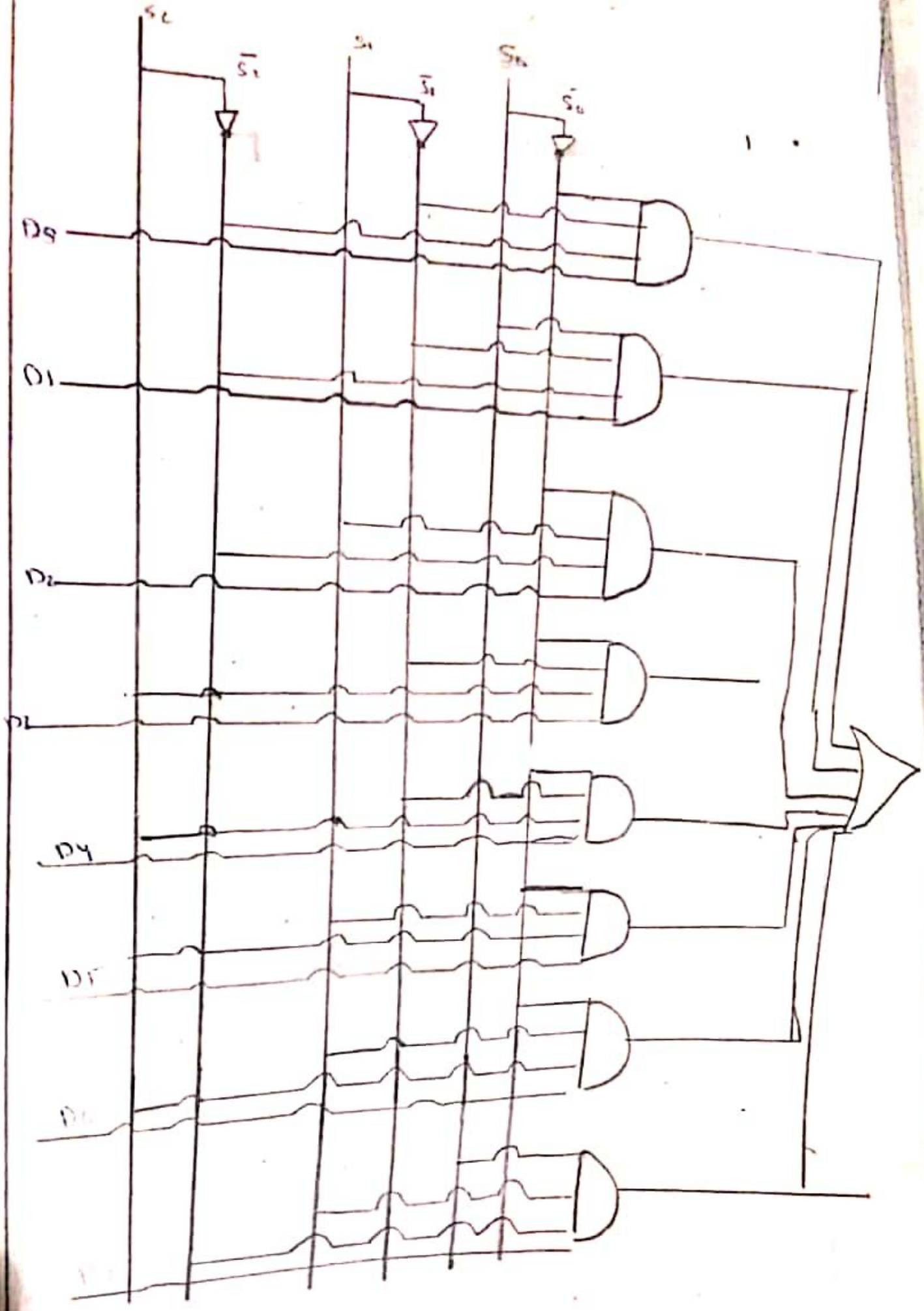

We can take S2 as enable for the two 4X1 MUX, since S2=0 will select the output from first four inputs and S2=1 will select output from last four inputs. So, finally we arrive to the result as




Here, we have applied not gate to the En1 to take S2=0 condition and directly applied S2 to En2 to take S2=1 condition. And finally we have applied output of both 4X1 MUXes to an OR gate, since at a time only one of the 4X1 MUX will activate.

Full text: 5391





# 8 x 1 mux

## Truth table

| $S_2$ | $S_1$ | $S_0$ | out put |
|---|---|---|---|
| 0 | 0 | 0 | $I_0$ |
| 0 | 0 | 1 | $I_1$ |
| 0 | 1 | 0 | $I_2$ |
| 0 | 1 | 1 | $I_3$ |
| 1 | 0 | 0 | $I_4$ |
| 1 | 0 | 1 | $I_5$ |
| 1 | 1 | 0 | $I_6$ |
| 1 | 1 | 1 | $I_7$ |

## 4 x 1

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

## 2 x 1

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |