

***100 DAYS RTL PART-1***

# Day 1: To verify Half Adder

# Day 2: To verify Full Adder

# Day 3: To verify Half subtractor

# Day 4: To verify Full subtractor

# Day 5: To verify AND gate

# Day 6: Design 4:1 mux

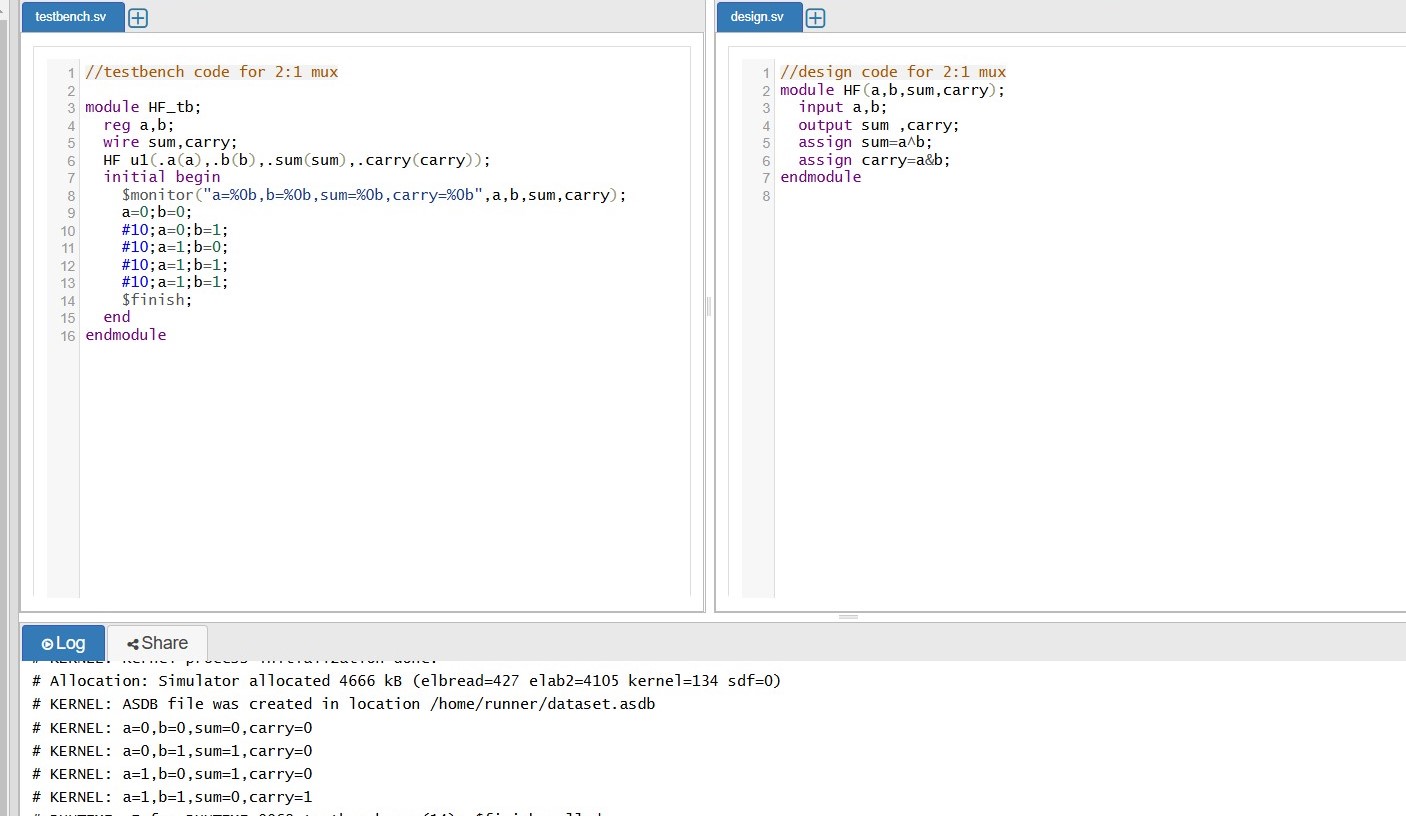
# Day 7: Design 2-bit comparator

# Day 8: Parallel Adder

# Day 9: UNIVERSAL SHIFT REGISTER

# Day 10: 8:3 Encoder

***Day 1: To verify Half Adder:***

******

Half Adder

A Half Adder is a digital circuit that adds two single-bit binary numbers. It has two inputs:

- A (Augend)

- B (Addend)

And two outputs:

- Sum (S)

- Carry (C)

Truth Table:

| A | B | Sum (S) | Carry (C) |

| --- | --- | --- | --- |

| 0 | 0 | 0 | 0 |

| 0 | 1 | 1 | 0 |

| 1 | 0 | 1 | 0 |

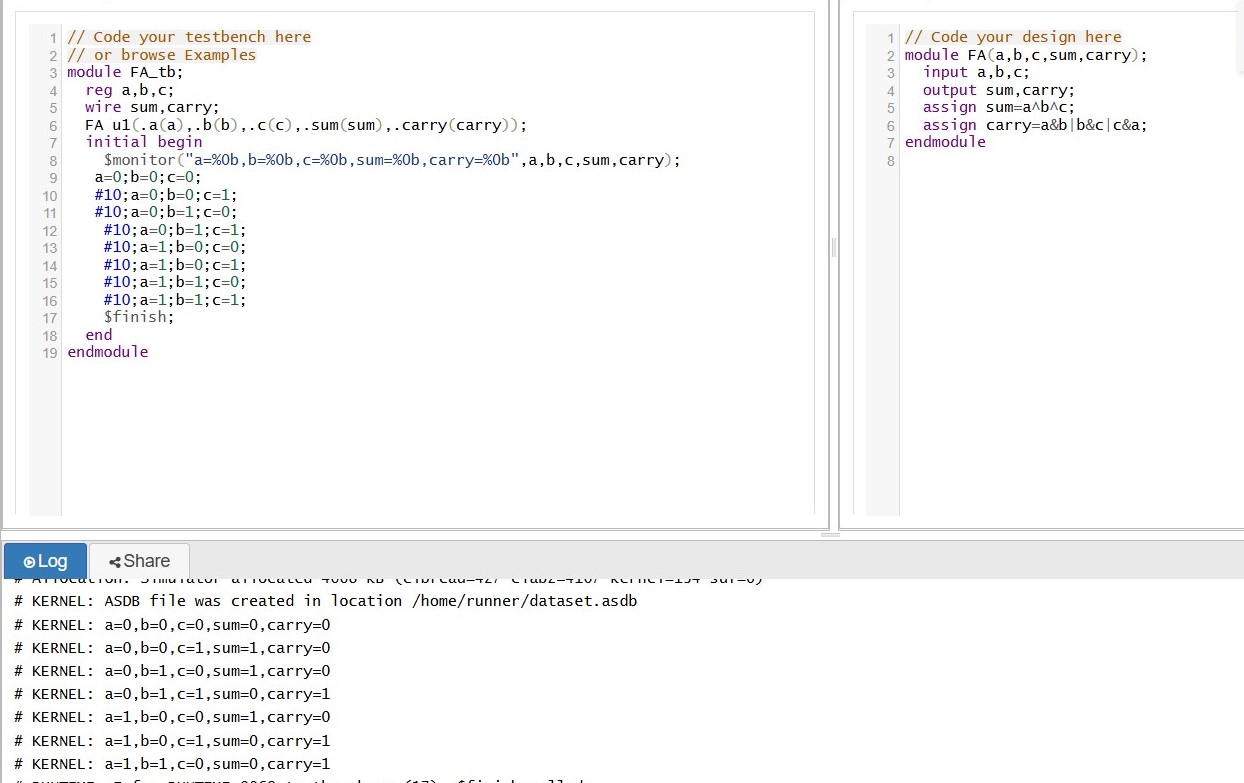
| 1 | 1 | 0 | 1 |

Boolean Equations

Sum (S) = A ⊕ B (XOR operation)

Carry (C) = A ∧ B (AND operation)

***Day 2: To verify Full Adder:***

******

🔹 Theory of full Adder (with 3 inputs: a, b, c)

⚙ Inputs:

a → 1st binary input

b → 2nd binary input

c → 3rd binary input

🔚 Outputs:

sum → Result of binary addition

carry → Carry-out of the addition

🔣 Logic Description:

➤ Sum:

sum = a ⊕ b ⊕ c

(XOR of all three inputs)

➤ Carry:

carry = (a & b) | (b & c) | (a & c)

🧾 Truth Table:

a b c sum carry

0 0 0 0 0

0 0 1 1 0

0 1 0 1 0

0 1 1 0 1

1 0 0 1 0

1 0 1 0 1

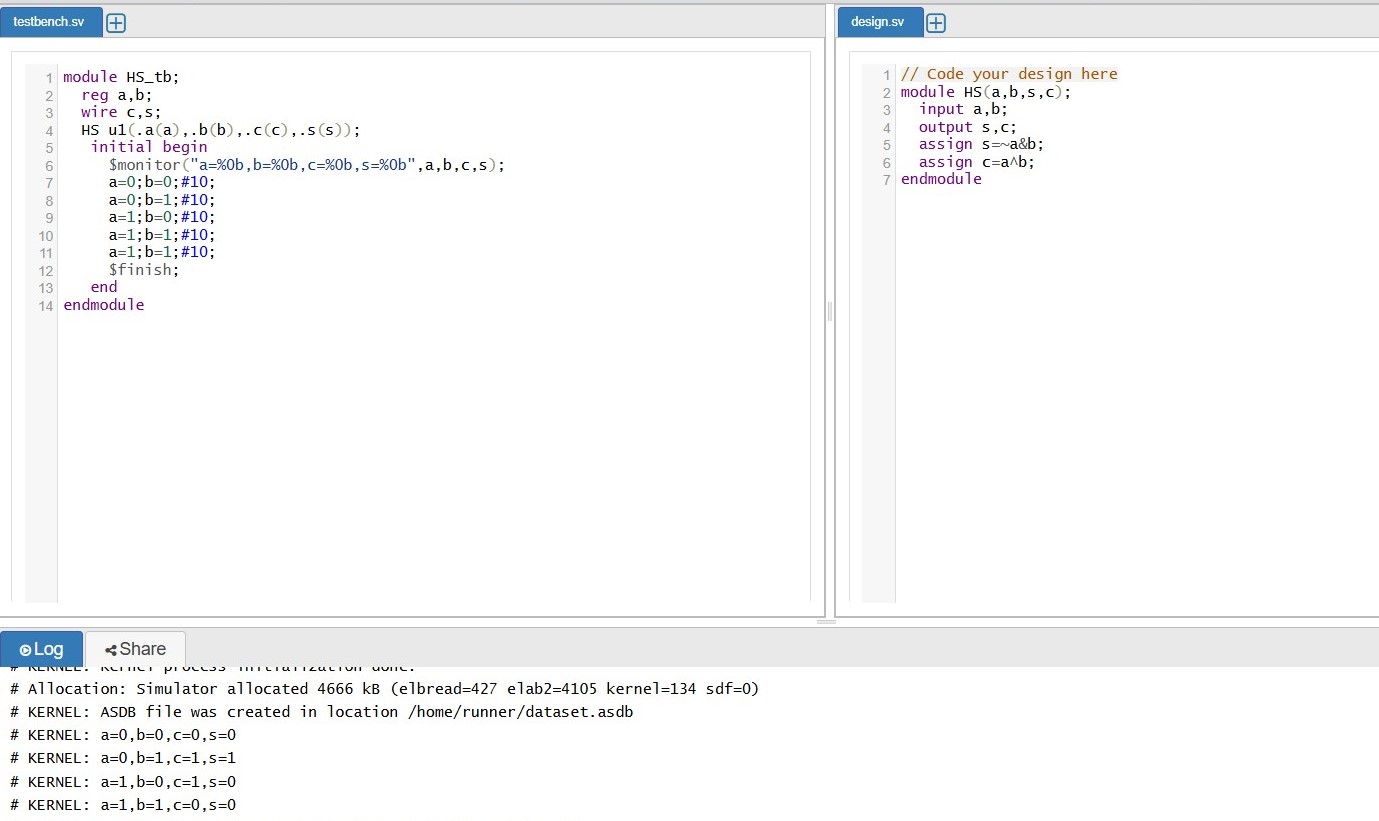
1 1 0 0 1

1 1 1 1 1

🛠 Use Case:

Although traditional half adders only take 2 inputs (a, b), this logic resembles a 1-bit full adder (without considering a carry-in from a previous stage). It can be used in : Arithmetic logic unit

***Day 3: To verify Half Subtractor:***



Half Subtractor:

A Half Subtractor is a combinational logic circuit used to perform the subtraction of two binary bits. It calculates the difference (s) and borrow (c) between two input bits.

⚙ Inputs:

a → Minuend (the number from which another number is subtracted)

b → Subtrahend (the number to be subtracted)

🔚 Outputs:

s → Difference (a - b)

c → Borrow (1 when subtraction requires borrowing)

🔣 Logic Description:

➤ Difference (s):

s = a ⊕ b

(XOR of a and b)

➤ Borrow (c):

c = ¬a & b

(Borrow is needed when a = 0 and b = 1)

🧾 Truth Table:

a b s (Diff) c (Borrow)

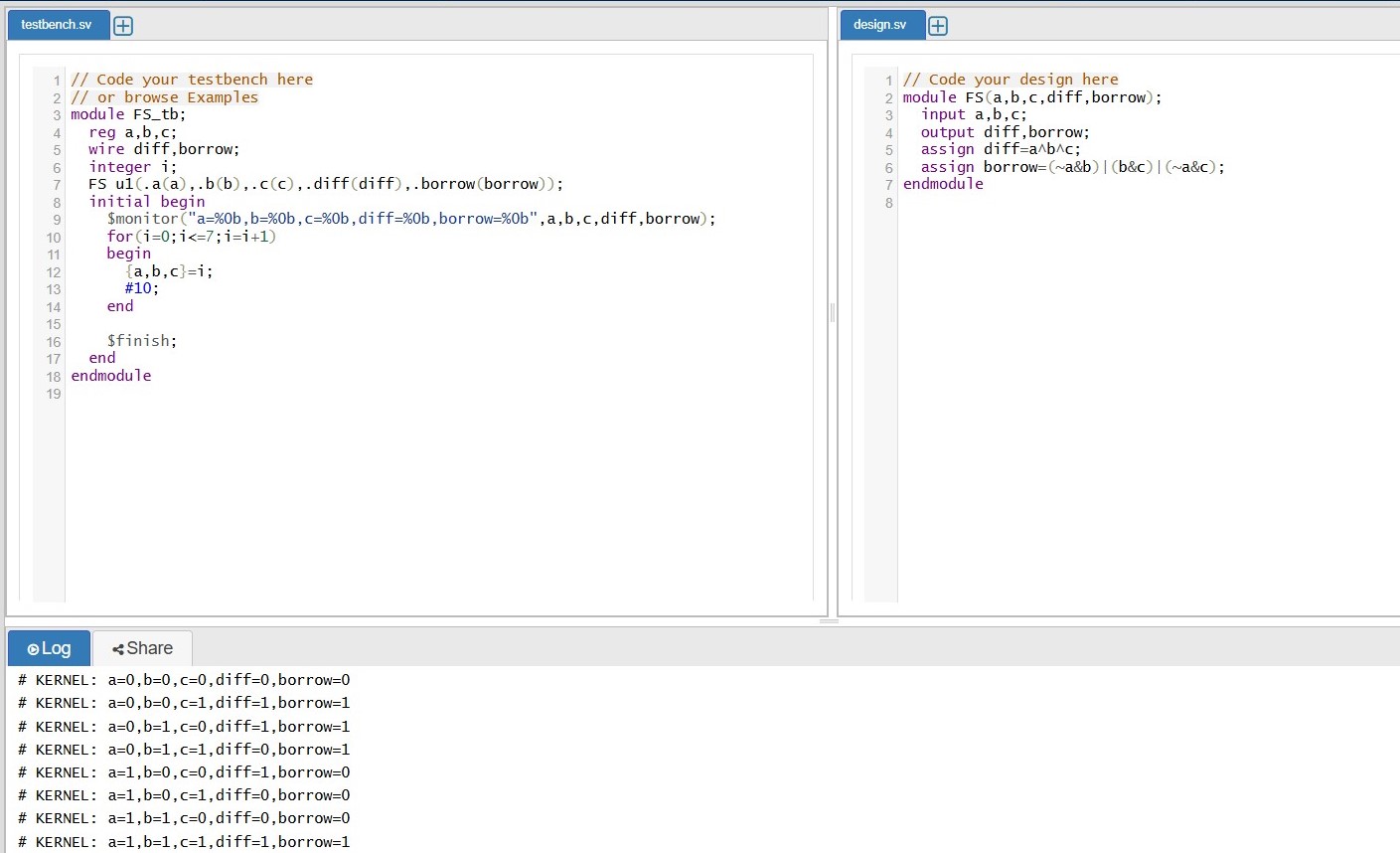
0 0 0 0

0 1 1 1

1 0 1 0

1 1 0 0

***Day4: To verify Full Subtractor:***



Full Subtractor

⚙ Inputs:

a → Minuend (bit to subtract from)

b → Subtrahend (bit to subtract)

c → Borrow-in from previous stage

🔚 Outputs:

diff → Result of subtraction

borrow → Borrow generated if subtraction needs it

🔣 Logic Description:

➤ Difference (diff):

diff = a ⊕ b ⊕ c

(XOR of all three inputs)

➤ Borrow :

borrow = (¬a & b) | ((¬a | b) & c)

🧾 Truth Table:

a b c diff borrow

0 0 0 0 0

0 0 1 1 1

0 1 0 1 1

0 1 1 0 1

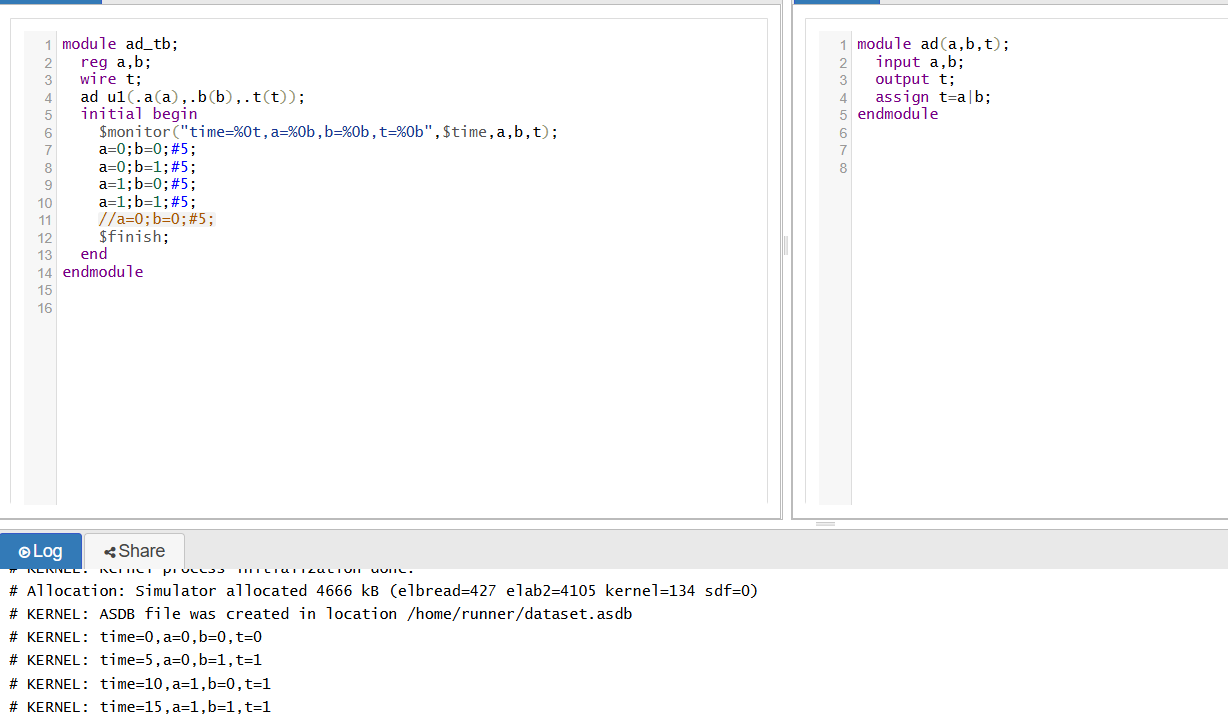
1 0 0 1 0

1 0 1 0 0

1 1 0 0 0

1 1 1 1 1

***Day5: To verify AND Gate***



1. AND Gate

Symbol: t = a AND b or t = a · b

Operation: The output t is high (1) only when both inputs a and b are high.

Truth Table:

a b t = a AND b

0 0 0

0 1 0

1 0 0

1 1 1

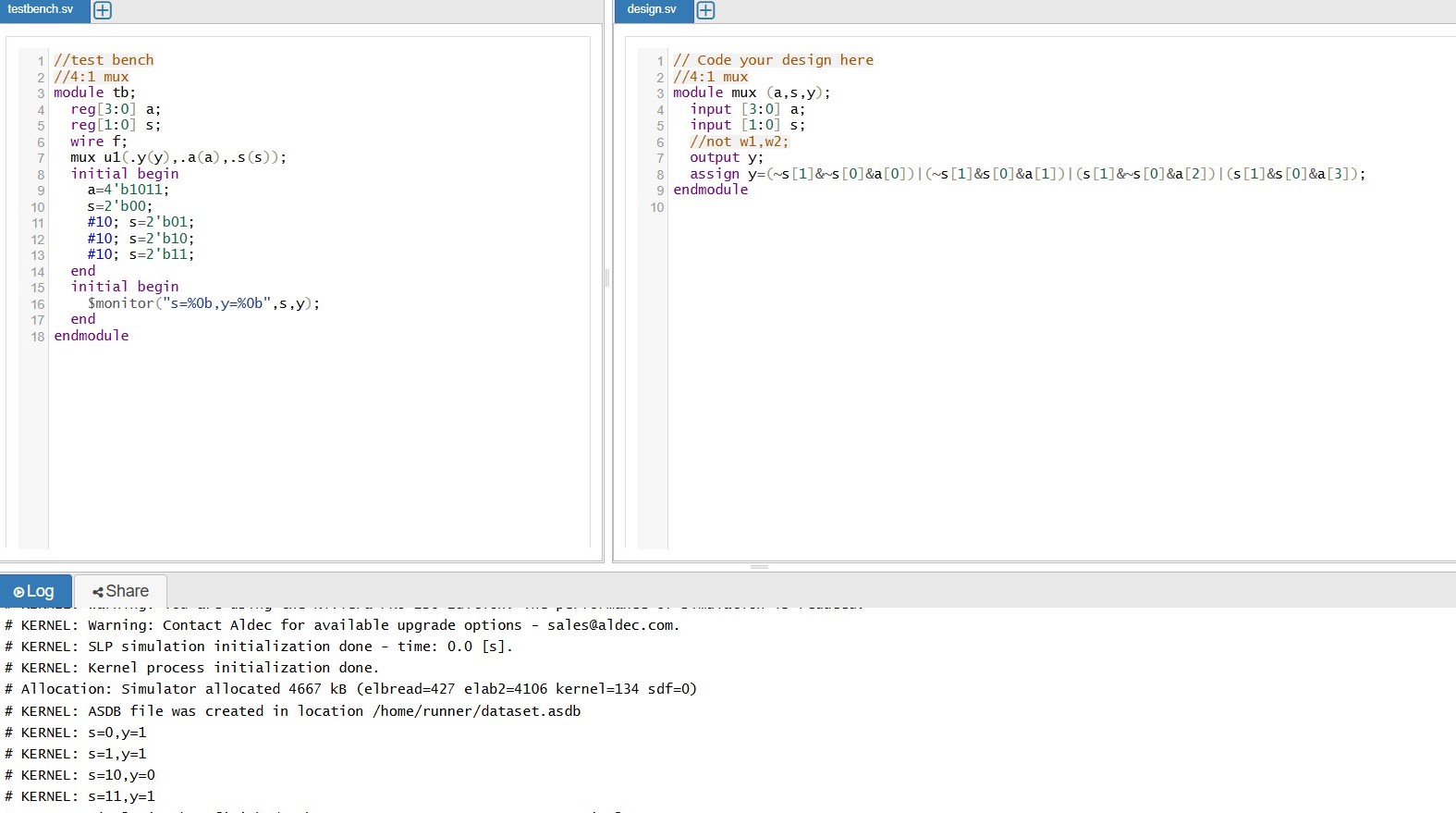
Key Points:

- The AND Gate produces an output of 1 only when all inputs are 1.

- If any input is 0, the output is 0.

AND Gates are widely used in digital electronics and computer systems for various applications, such as:Logical Operations

***Day6: To Design 4:1 mux***

******

4:1 Multiplexer (MUX)

Definition:

A Multiplexer (MUX) is a combinational circuit that selects one input from multiple data inputs and routes it to a single output line. The selection of input is controlled by select lines.

s1 s0

↓ ↓

┌───────┐

a0 ─────▶│ │

a1 ─────▶│ 4:1 │────▶ y

a2 ─────▶│ MUX │

a3 ─────▶│ │

└───────┘

If s1 s0 = 00, then y = a0

If s1 s0 = 01, then y = a1

If s1 s0 = 10, then y = a2

If s1 s0 = 11, then y = a3

Truth Table:

s1 s0 Output y

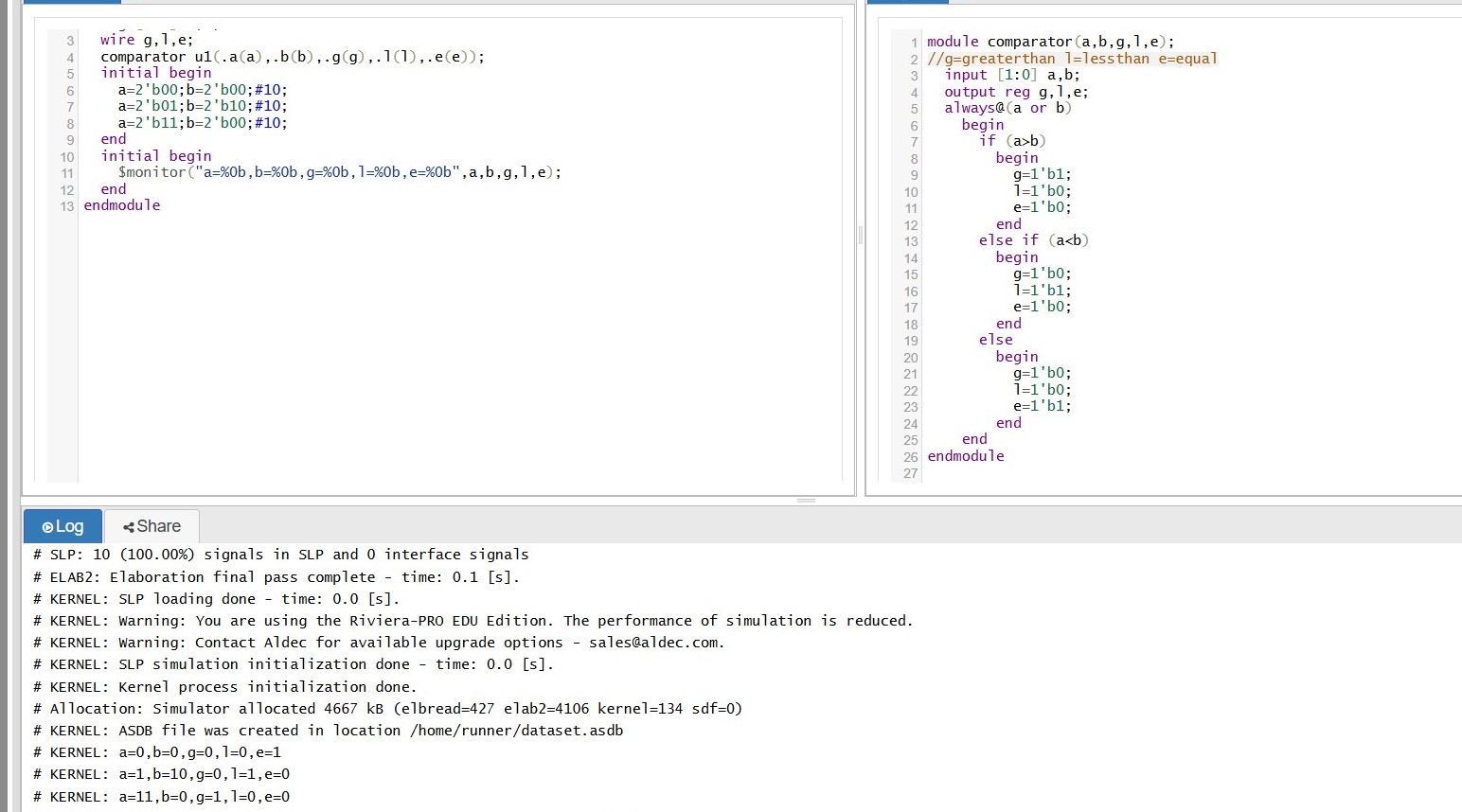
0 0 a0

0 1 a1

1 0 a2

1 1 a3

# ***Day7: To Design 2-Bit Comparator***

******

2-Bit Comparator

g (greater): 1 if a > b

l (less): 1 if a < b

e (equal): 1 if a = b

Inputs:

a = a1a0 (2-bit number)

b = b1b0 (2-bit number)

Outputs:

g: 1 if a > b

l: 1 if a < b

e: 1 if a = b

Working Principle:

The comparator performs a bit-wise comparison, starting from the most significant bit (MSB) to the least significant bit (LSB). The logic follows:

1. If a1 > b1, then a > b → g = 1

2. If a1 < b1, then a < b → l = 1

3. If a1 = b1, then compare a0 and b0

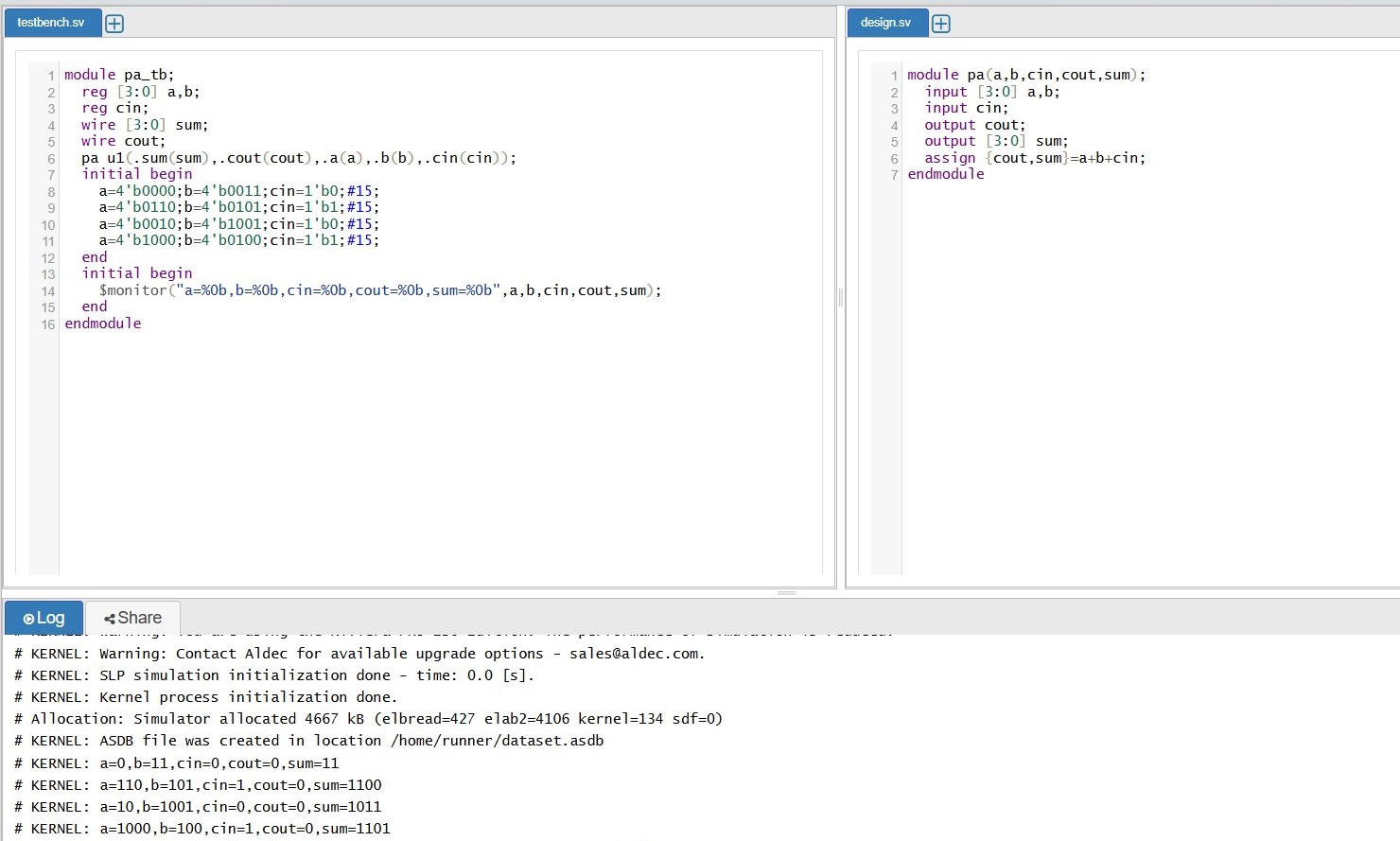
If a0 > b0, then a > b

If a0 < b0, then a < b

If a0 = b0, then a = b

Truth Table:

***Day:8 To Design Parallel Adder***

******

Parallel Adder

Definition:

A Parallel Adder is a digital circuit that performs the arithmetic sum of two or more multi-bit binary numbers simultaneously. It uses multiple full adders, one for each bit position, connected in series. The carry output of one full adder is passed as the carry input to the next.

Inputs:

a = an−1...a1a0: First binary number (n bits)

b = bn−1...b1b0: Second binary number (n bits)

cin: Initial carry input (usually 0)

Outputs:

sum = sn−1...s1s0: Sum output (n bits)

cout: Final carry output (carry-out from the most significant bit)

Working Principle:

A Parallel Adder uses a series of Full Adders, one per bit, as shown below for a 4-bit adder:

a3 a2 a1 a0

| | | |

b3 b2 b1 b0

| | | |

FA FA FA FA

| | | |

s3 s2 s1 s0 ← sum

↑ ↑ ↑

c3 c2 c1 ← carry chain

Truth Table for a Single Full Adder (1-bit block):

a b cin sum cout

0 0 0 0 0

0 0 1 1 0

0 1 0 1 0

0 1 1 0 1

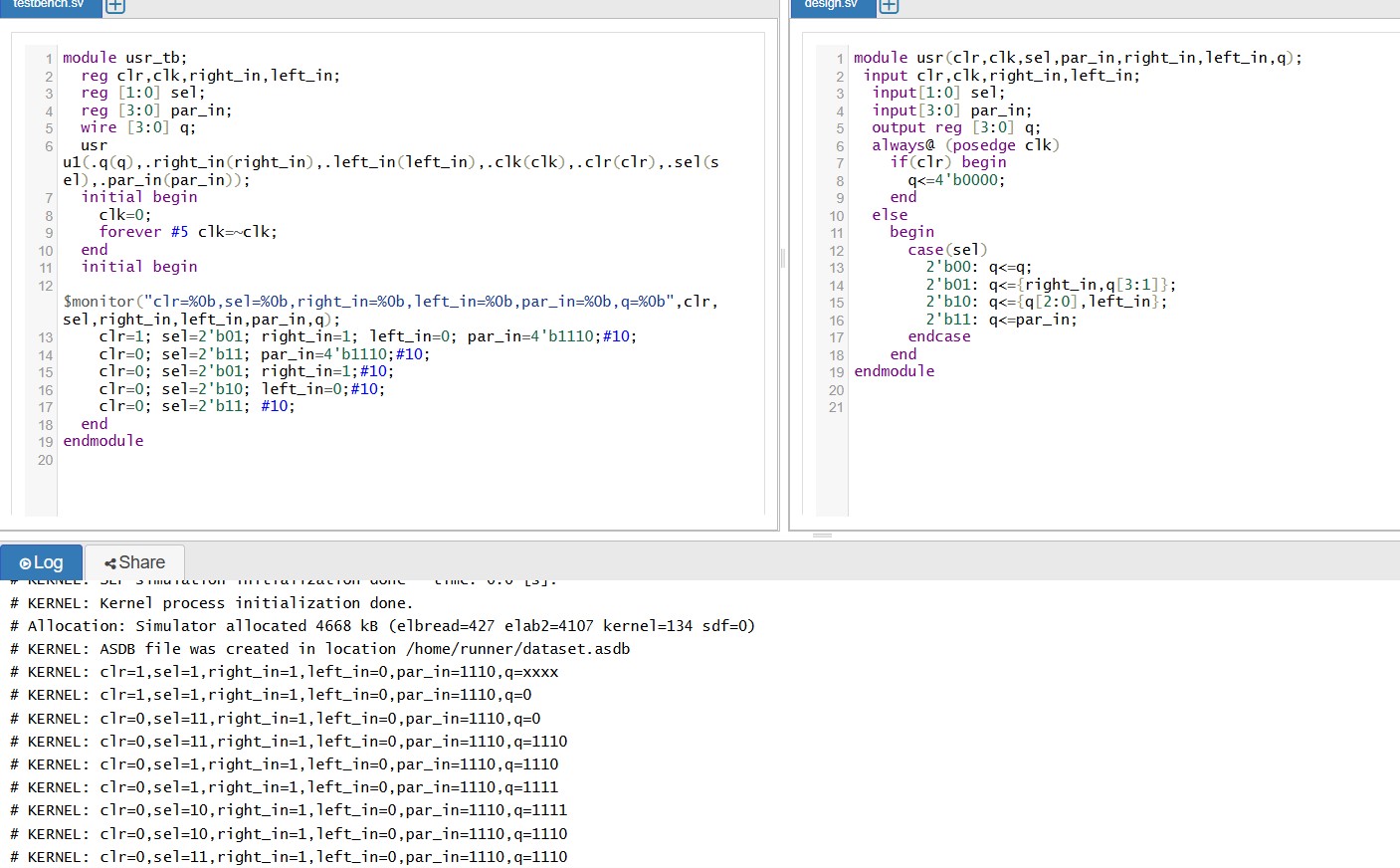
1 0 0 1 0

1 0 1 0 1

1 1 0 0 1

1 1 1 1 1

***Day:9 To Design universal shift register***

******Universal Shift Register

It is called "universal" because it can operate in all common modes of data transfer.

Inputs:

clk: Clock input — synchronizes data operations

clr: Clear/reset input — resets all outputs to 0

right\_in: Serial data input for right shift operation

left\_in: Serial data input for left shift operation

par\_in[n:0]: Parallel data input (n-bit)

mode[1:0]: Control lines to select operation mode (optional, often used)

Output:

q[n:0]: Current data output of the register (n-bit)

Mode (M1 M0) Operation

00 Hold (no change)

01 Right Shift

10 Left Shift

11 Parallel Load

Working Principle:

1. Clear (clr = 1):

All bits of the register are reset to 0 regardless of other inputs.

2. Hold Mode (M1 M0 = 00):

Register retains its previous state; no data is shifted or loaded.

3. Right Shift (M1 M0 = 01):

All bits shift one position to the right.

MSB gets data from right\_in, and LSB is discarded.

4. Left Shift (M1 M0 = 10):

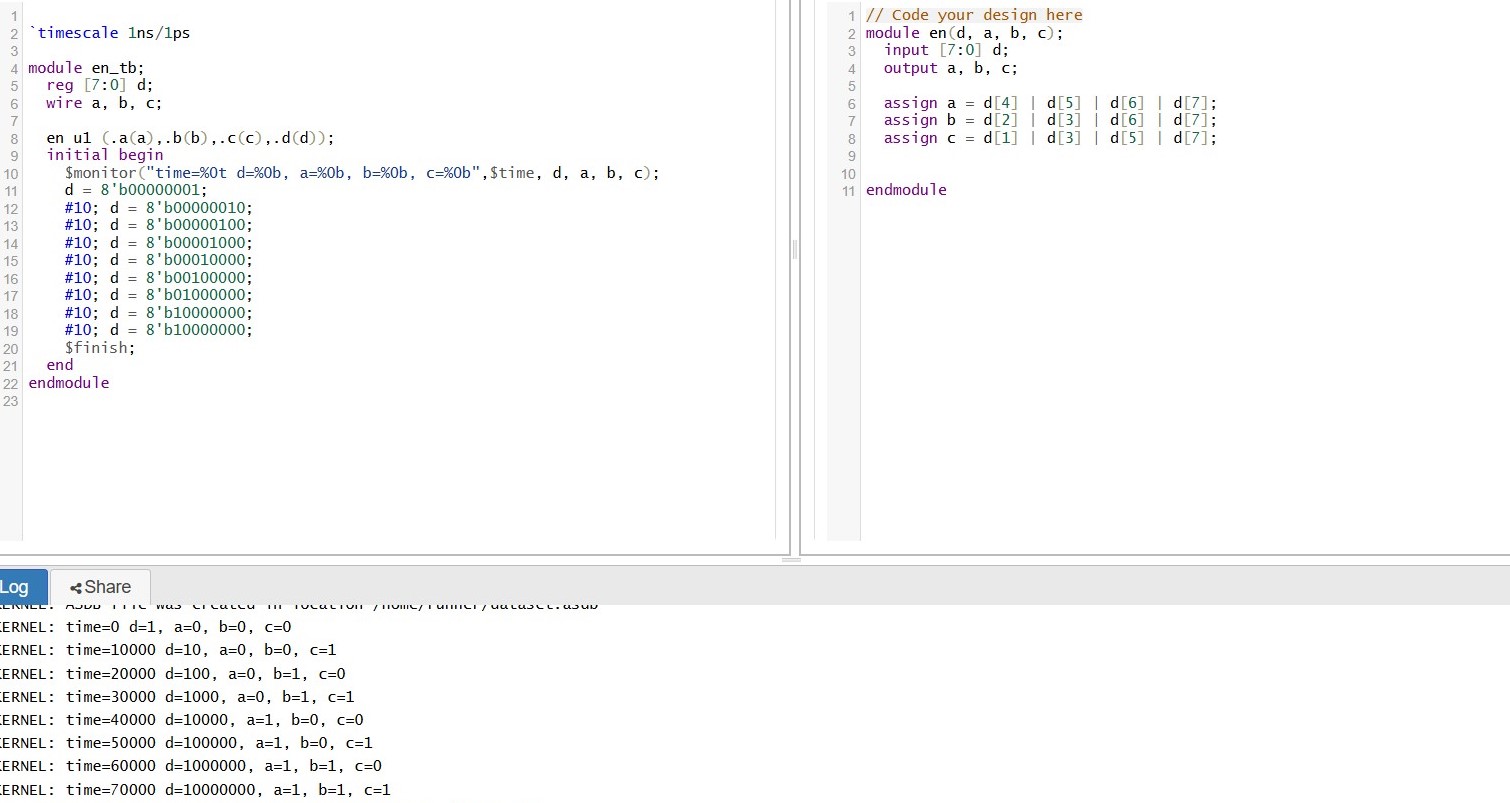
All bits shift one position to the left.

LSB gets data from left\_in, and MSB is discarded.

5. Parallel Load (M1 M0 = 11):

Data on par\_in is loaded into the register simultaneously on the rising edge of clk

***Day:10 To Design 8:3 Encoder***

******

8:3 Encoder

Definition:

An 8:3 Encoder is a combinational logic circuit that encodes 8 input lines into a 3-bit binary code. Only one input should be active (HIGH = 1) at any time. The output gives the binary equivalent of the position of the active input.

Inputs:

d0 to d7 (8 input lines)

Outputs:

a, b, c (3 output bits representing binary code)

Working Principle:

The 8:3 encoder checks which one of the 8 inputs is HIGH (1) and encodes the position/index of that input into a 3-bit binary number

If d0 = 1, output is 000

If d1 = 1, output is 001

….

If d7 = 1, output is 111

Truth Table:

d7 d6 d5 d4 d3 d2 d1 d0 a b c

0 0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 0 1 0 0 0 1

0 0 0 0 0 1 0 0 0 1 0

0 0 0 0 1 0 0 0 0 1 1

0 0 0 1 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0 1 0 1

0 1 0 0 0 0 0 0 1 1 0

1 0 0 0 0 0 0 0 1 1 1

Logic Expressions:

Let the output bits be:

a: MSB (Most Significant Bit)

b: Middle Bit

c: LSB (Least Significant Bit)

The outputs are expressed as:

a = d4 + d5 + d6 + d7

b = d2 + d3 + d6 + d7

c = d1 + d3 + d5 + d7

******