King Saud University	**FALL 2022**
College of Computer & Information Sciences	**CEN 415**
Department of Computer Engineering	**Dr. KHURSHEED AURANGZEB**

Student Name: Muhannad Funissan Alanizy.	**Section:** 32693	**Student ID:** 439101758.
Student Name: Rami Khaled Almutiry.	**Section:** 32693	**Student ID:** 439102284.
Student Name: Sultan Alotaibi.	**Section:** 32693	**Student ID:** 439101605.

CEN 415_Project

Phase (2)

Introduction:

Arithmetic Logic Unit (ALU) is the computing Heart of the microprocessors and is part of the Computer Datapath Operator that pretty much executes arithmetic and logic instructions.

In this project we would like to discover the ALU in more detail and be able to take some form of an Overview model of the architecture, to be able to apply it later on when needed on different Cmos Circuits for Application specific purposes.

Problem Statement and Specification:

Arithmetic and logic unit is the executing datapath operator that carries out Arithmetic and logic instructions and moves the program counter forward with instructions it’s a Necessity for any microprocessor to contain one, however if we were to apply one what are the most common components to be used for one?

In our case, our humble user has requested ALU with this subset of Specification has the following cases:

⇨ Design and implement a circuit that has **two 8-bit inputs and one 16-bit output** to perform the following tasks based on the user choice determined C [1:0].

- Case 00:

  The circuit generate the complement of the first input & show the result in the 8 least significant bits of the output.

- Case 01:
  The circuit perform a modulo 256 count down with the ability to hold the result if (H) is asserted. The result is shown in the 8 most significant bits of the output.

- Case 10:
  The circuit multiply both inputs using shift and add method.

- Case 11:
  the circuit generates the even parity of the second input and show the result in the most significant bit of the output.

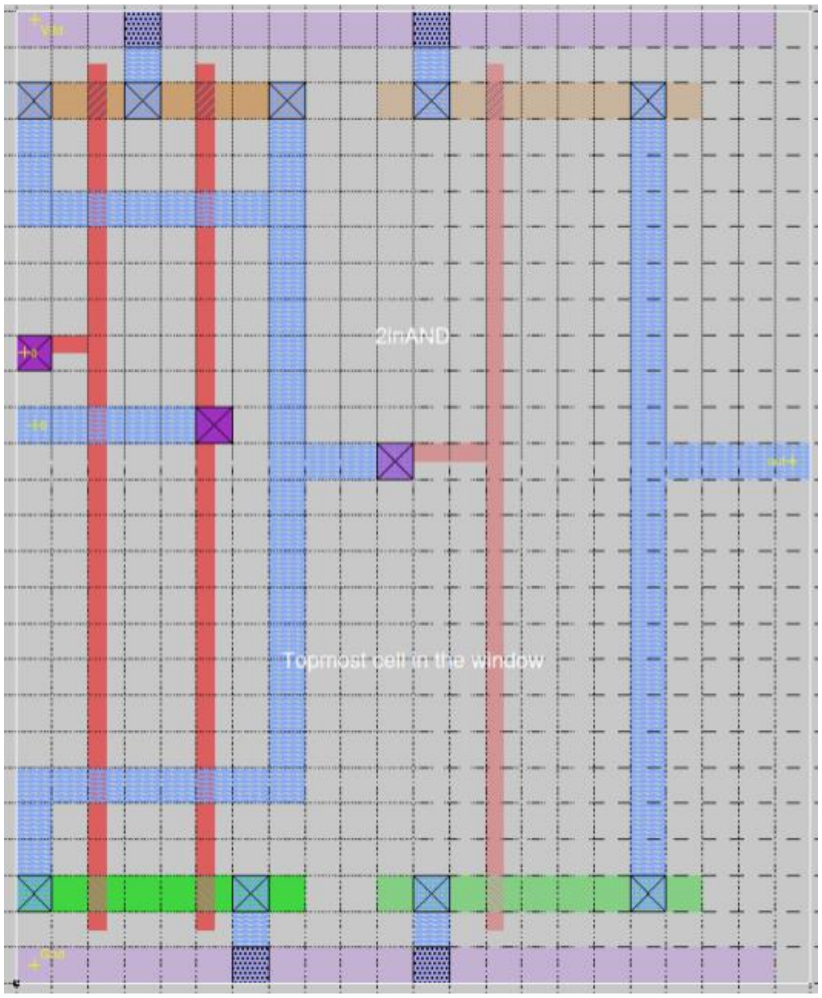
Motivation:

Our main goal is to get hands-on experience and learn to put our obtained skillsto practice, not-to-forget to model our approach such that when we want to redesign the ALU or a microprocessor, we'd had backdoors up our sleeves.

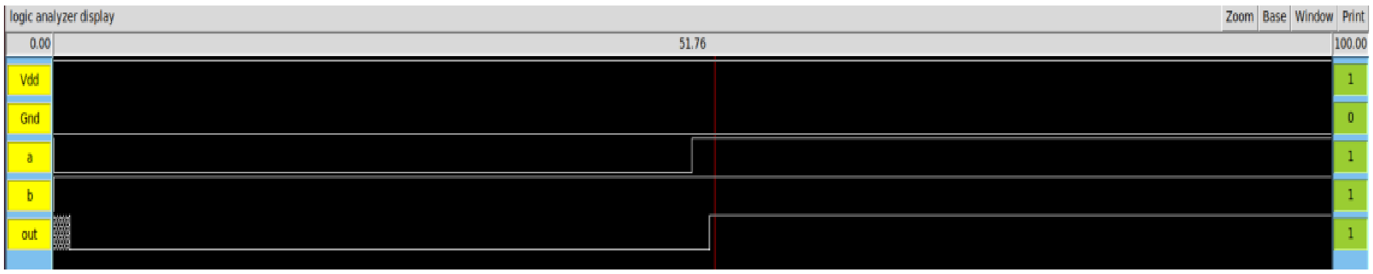
Standard Gates that we used in our design for the cases:

- **Two input AND gate:**

- Layout:

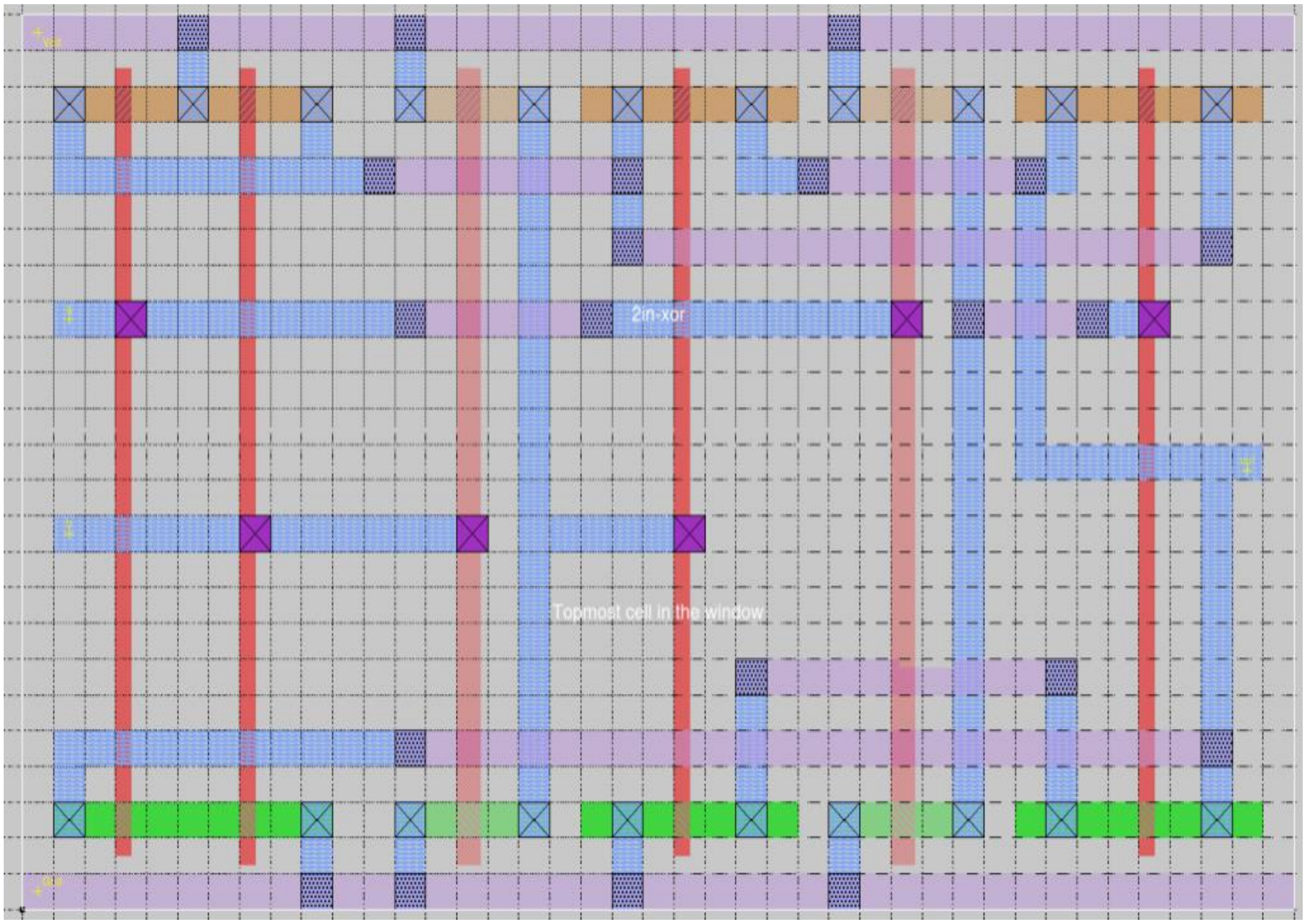


- Test the gate:

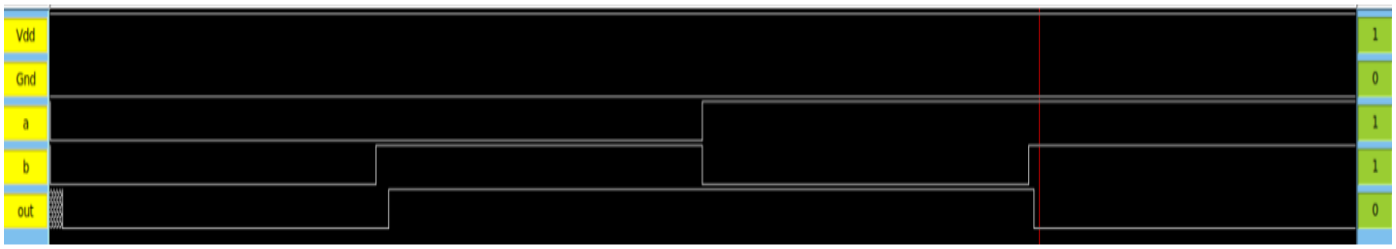


- **Two input XOR gate:**

- Layout:

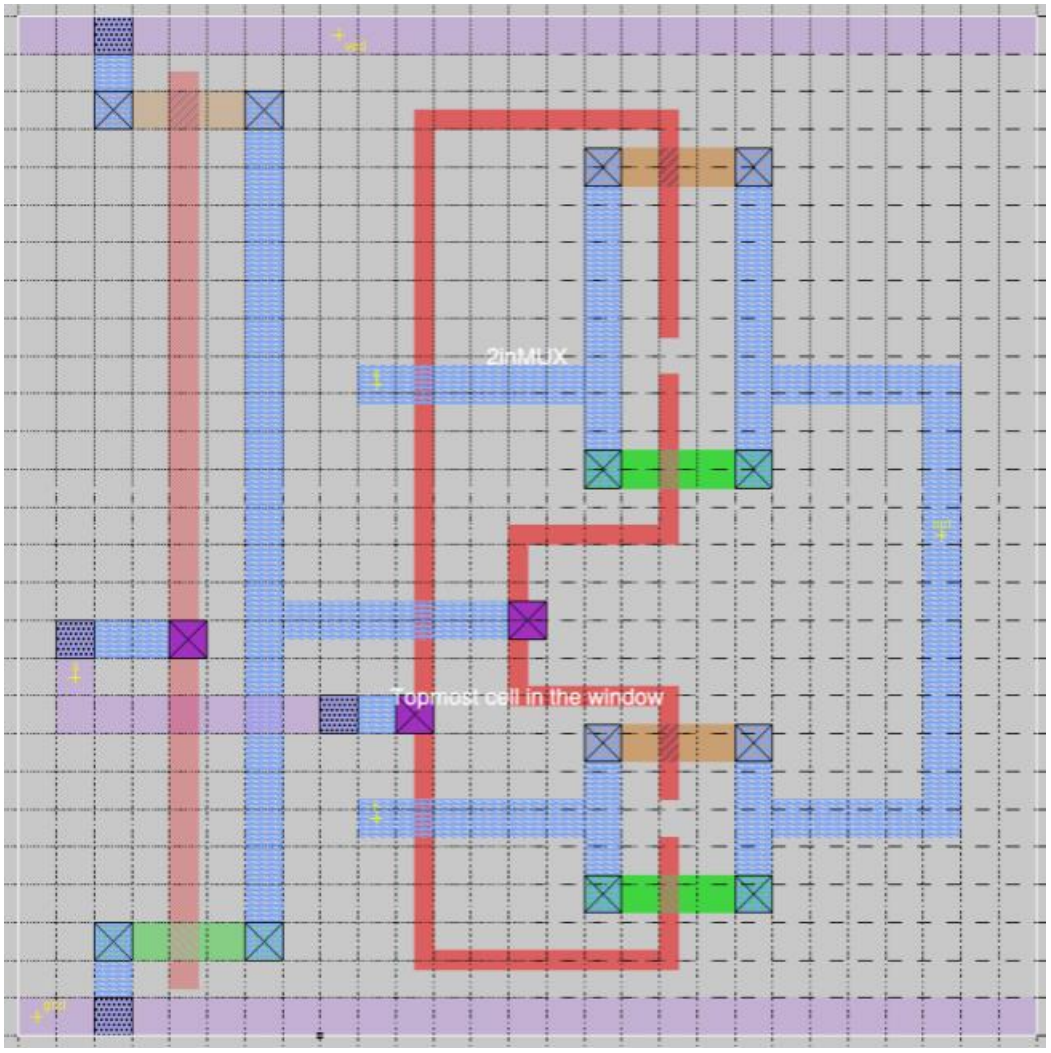


- Test the gate:

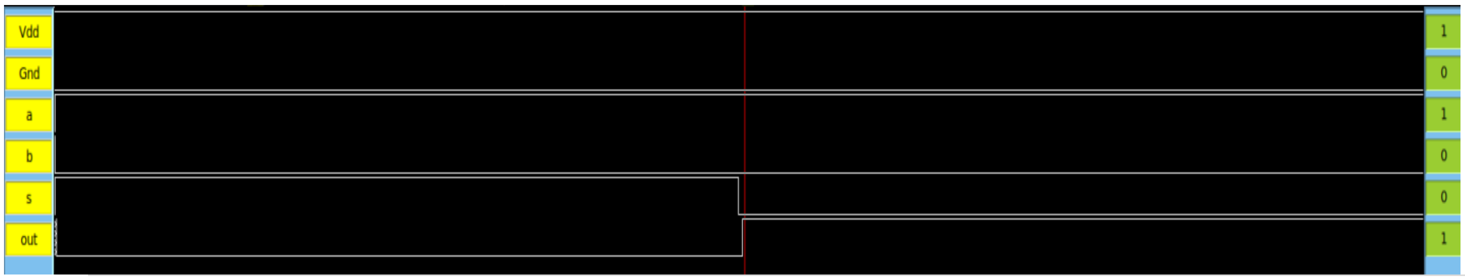


- **MUX2-1:**

- Layout:

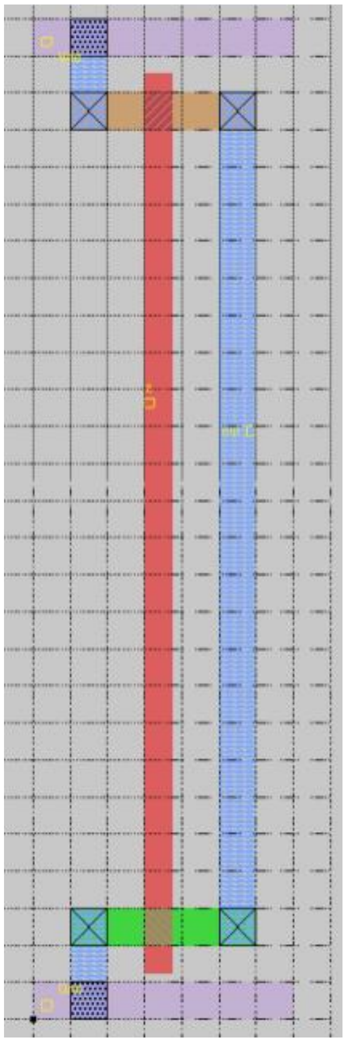


- Test the MUX2-1:

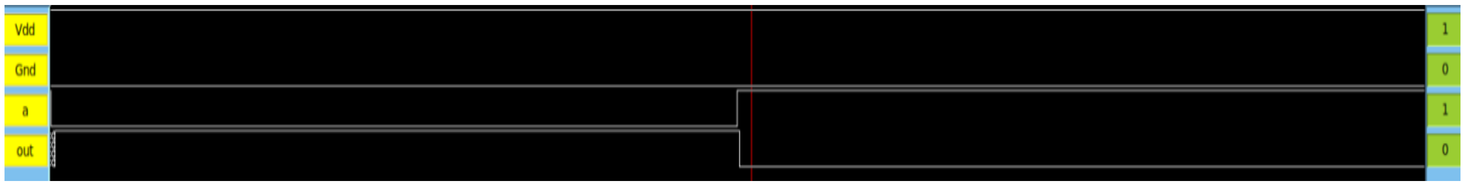


- **Inverter gate:**

- Layout:

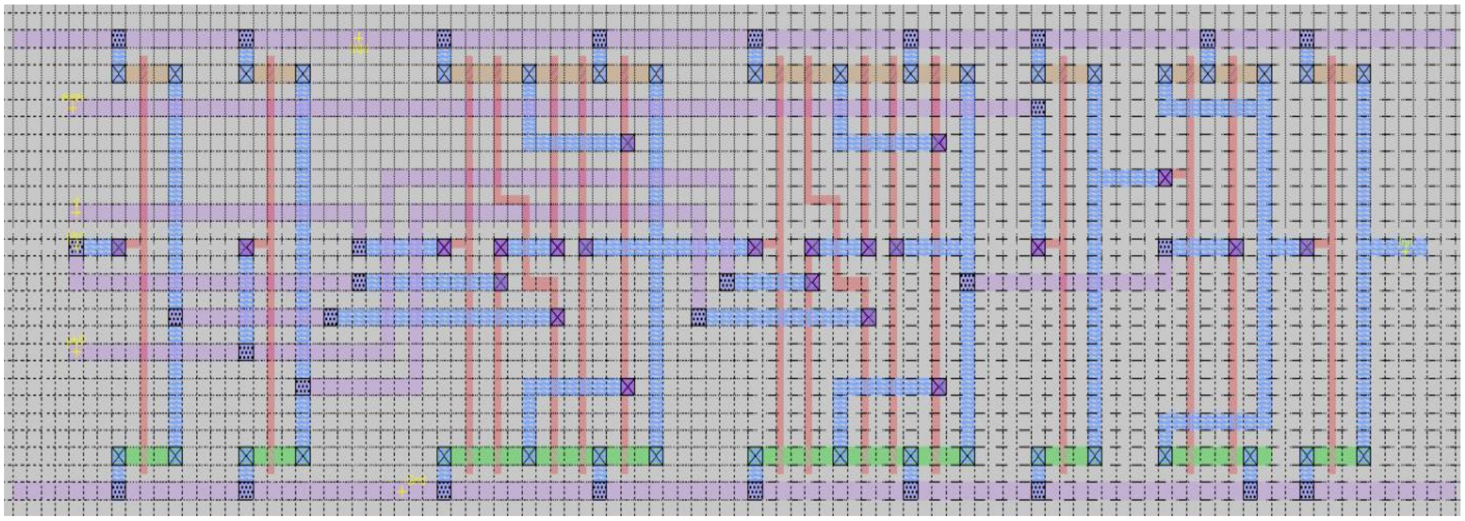


- Test the gate:

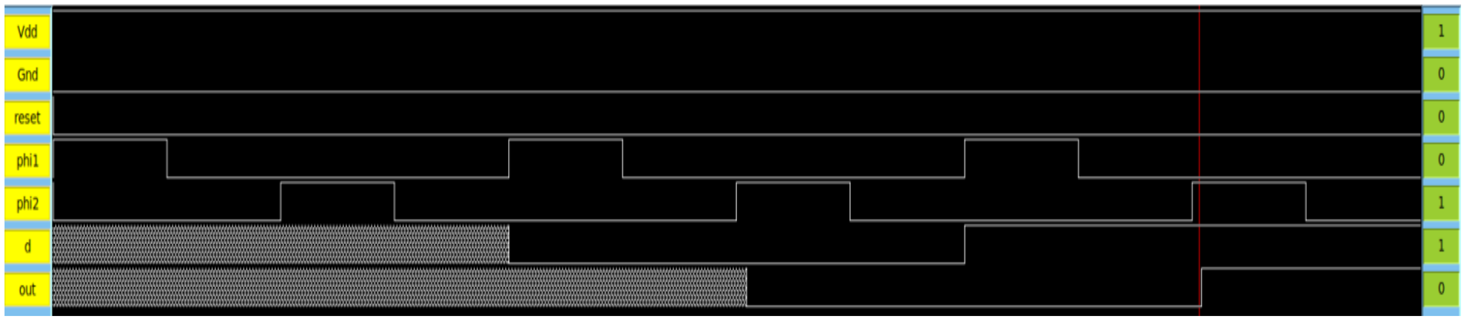


- **D flip flop:**

- Layout:

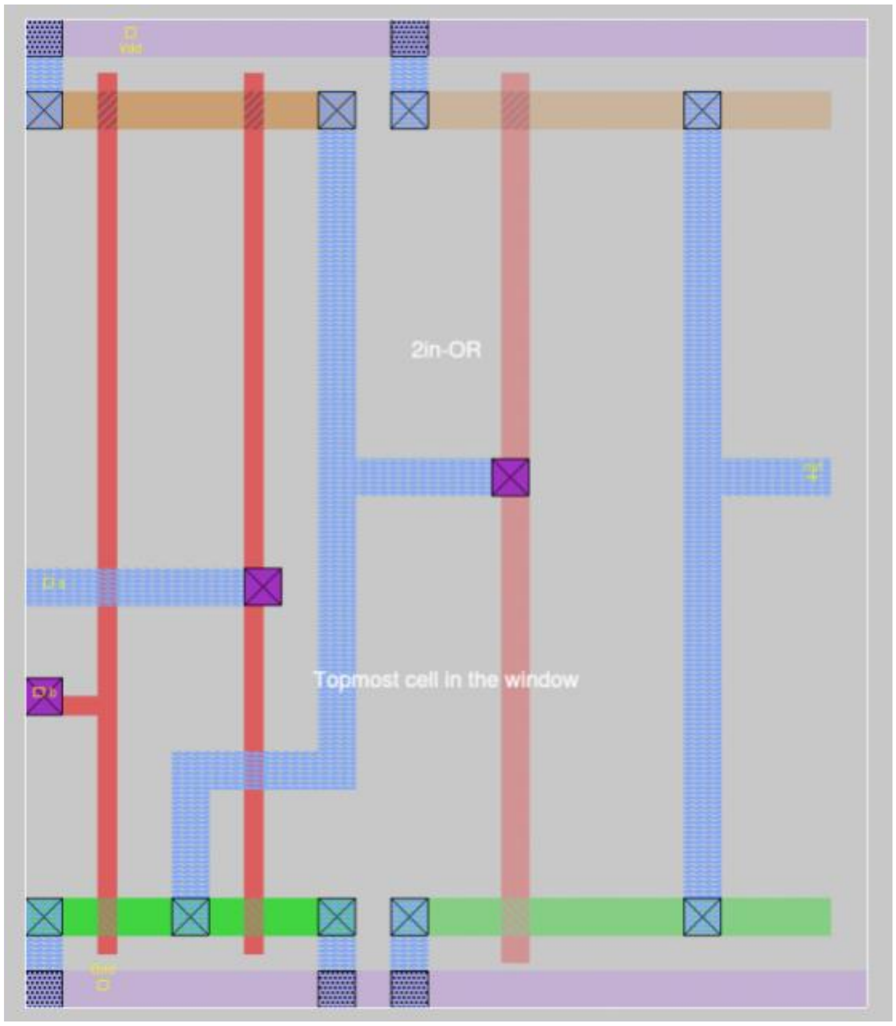


- Test the flip flop:

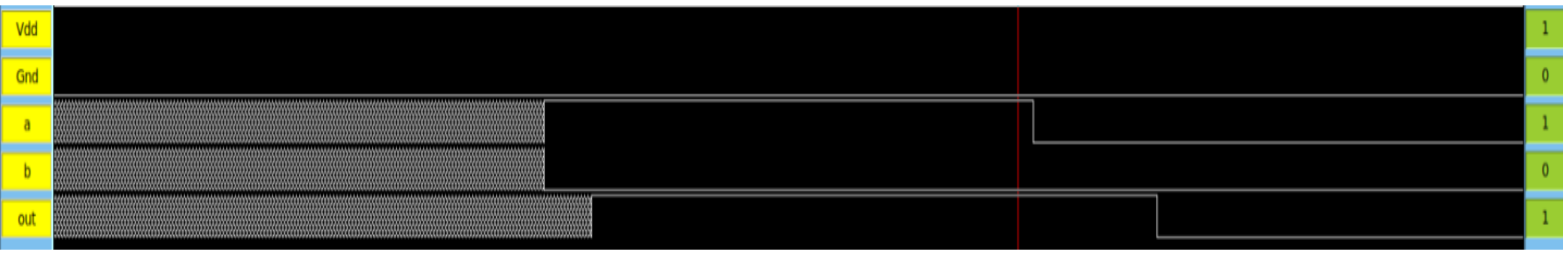


- **Two input OR:**

- Layout:

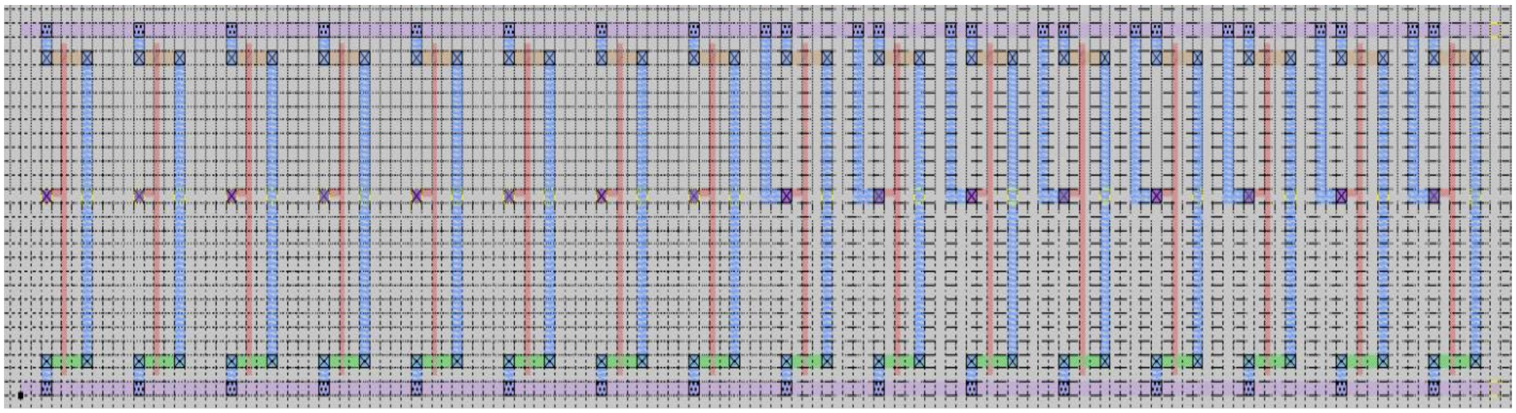


- Test the OR gate:



Case 00:

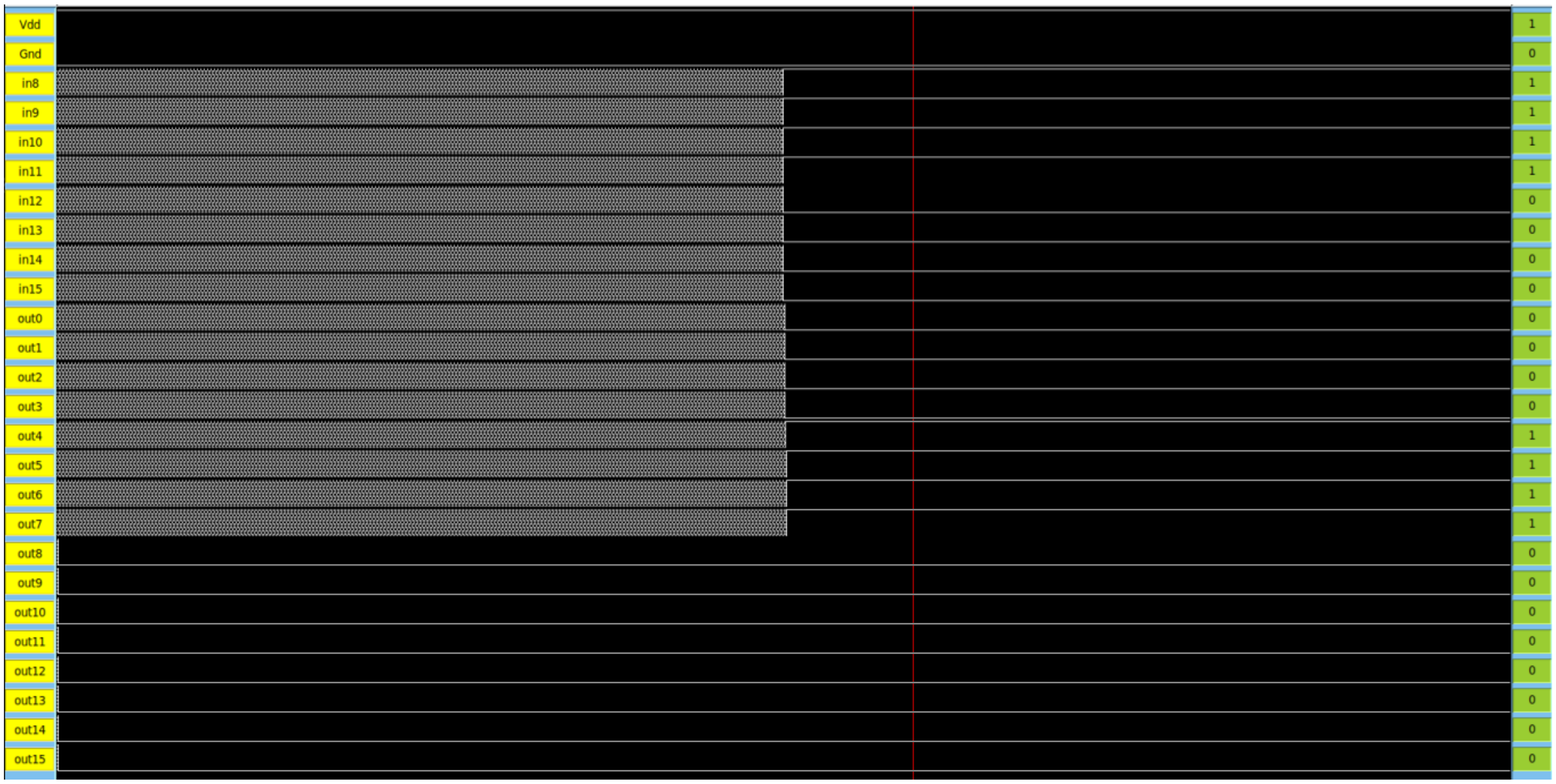
- Layout:



- Test the Case 00:

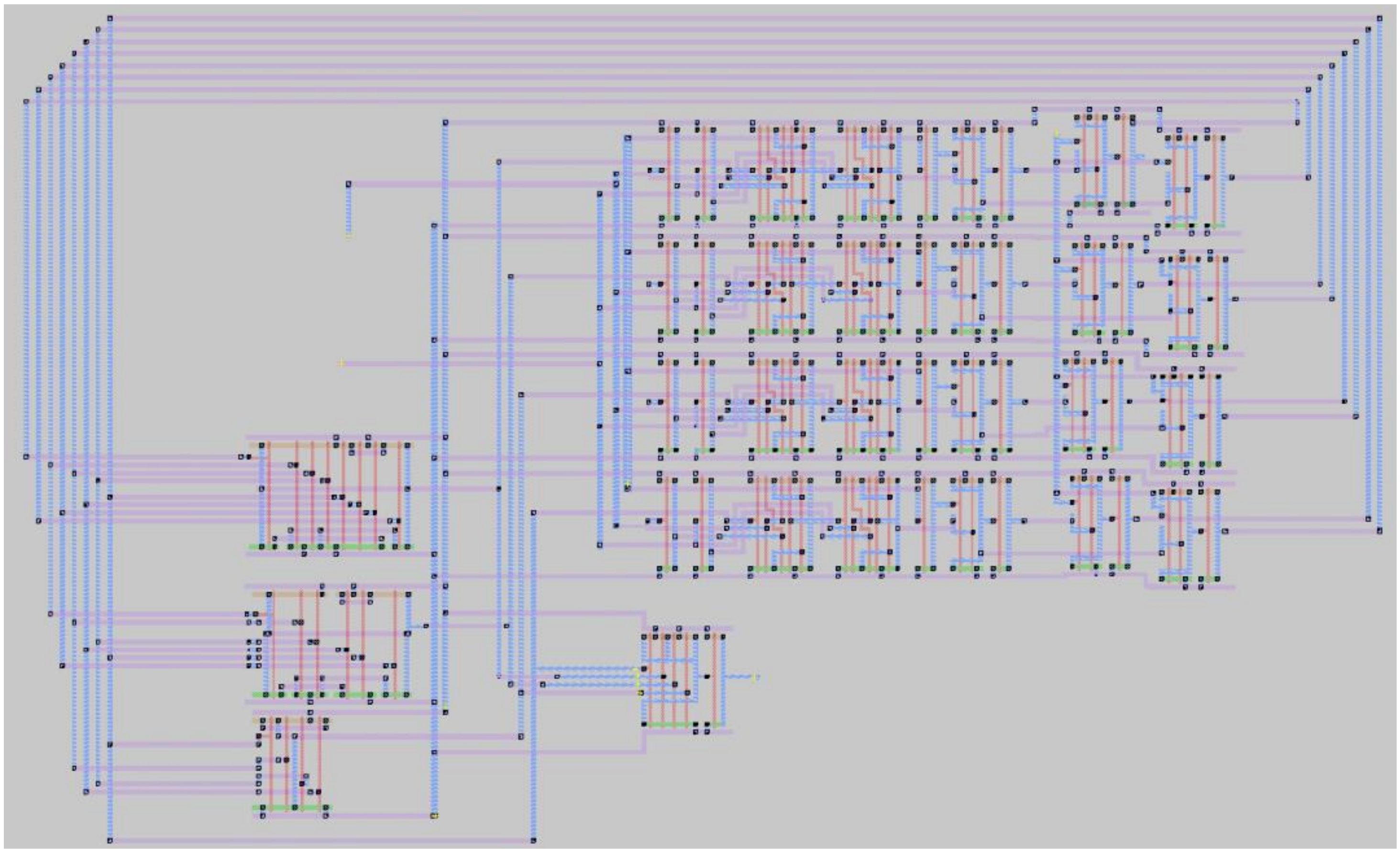
⇨ Input:
From In15 to In8: 0000 1111

⇨ Output will be from out7 to out0:

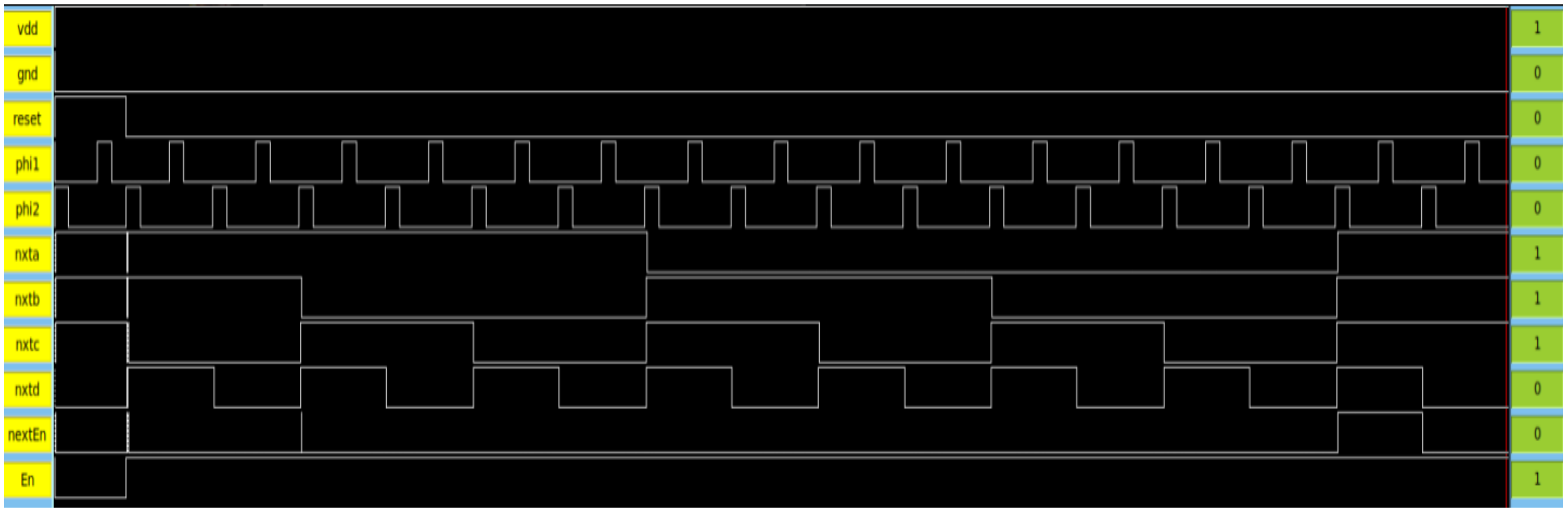


Case 01: ***Down Counter 4-bit it counts from 15 to 0***

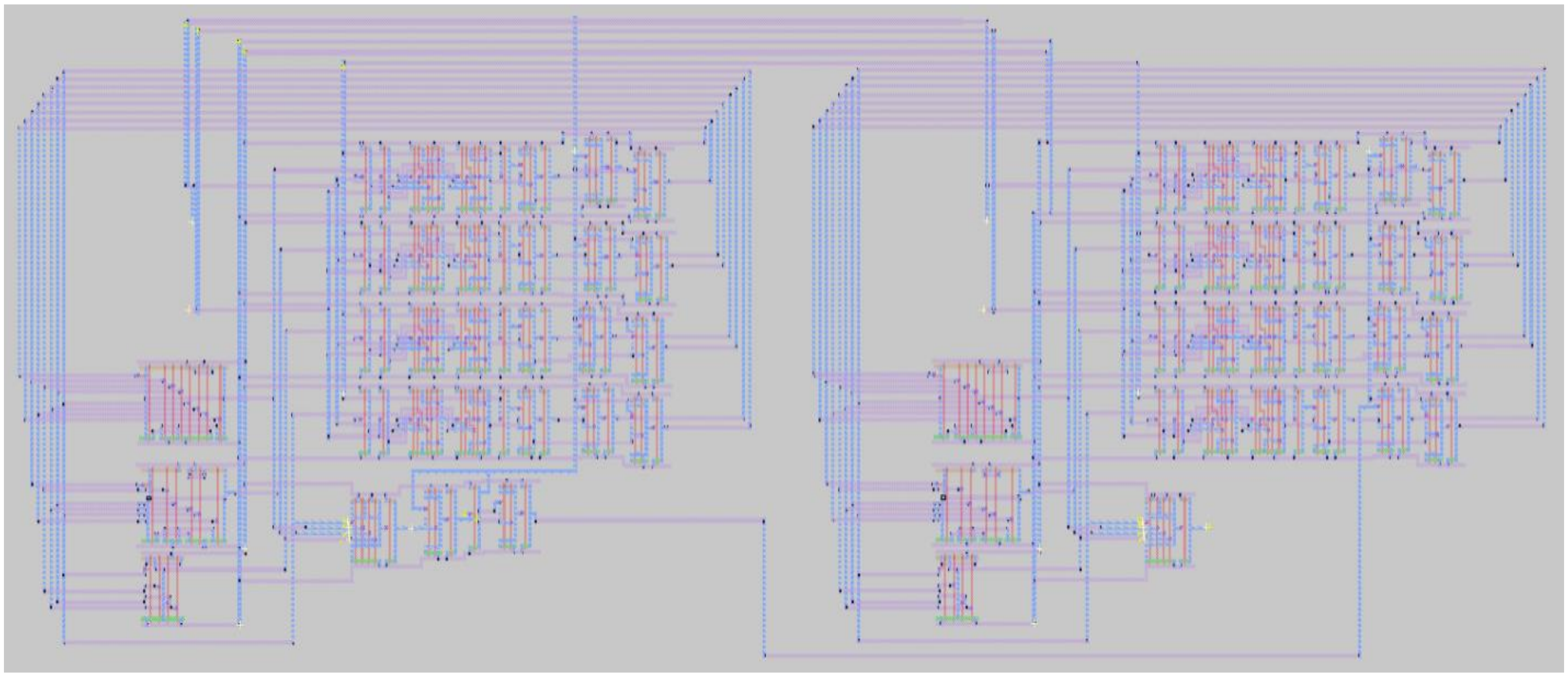
- Layout:



- Test the 4-bit counter:



Down Counter 8-bit it counts from 255 to 0

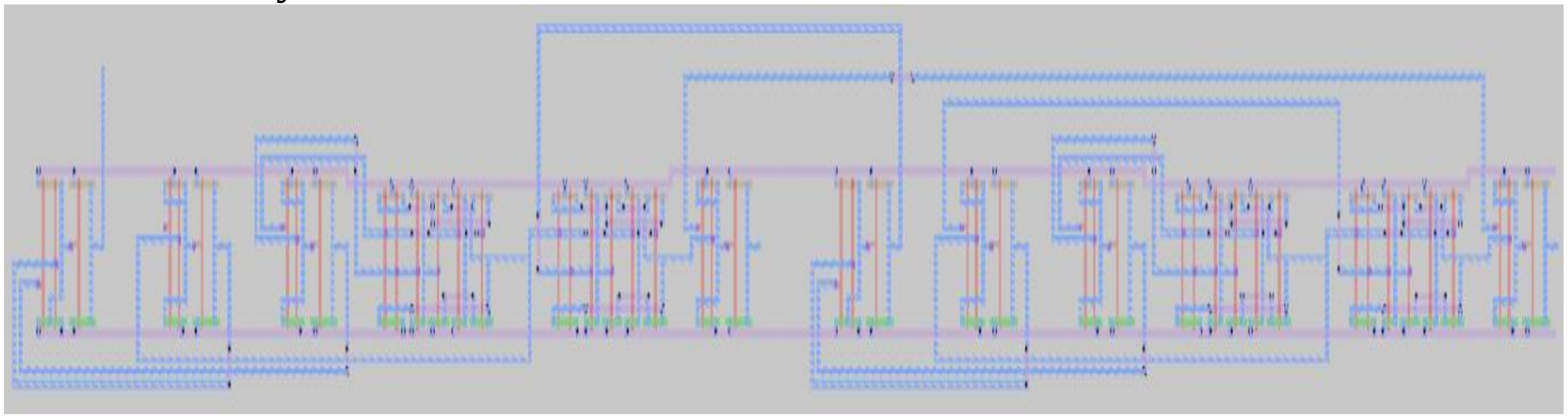


- Test the 8-bit counter:

Case 10:

- ***Two-bit ALU we repeat the two-bit for whole 16-bit:***

- Layout:



- Test#1 the ALU:

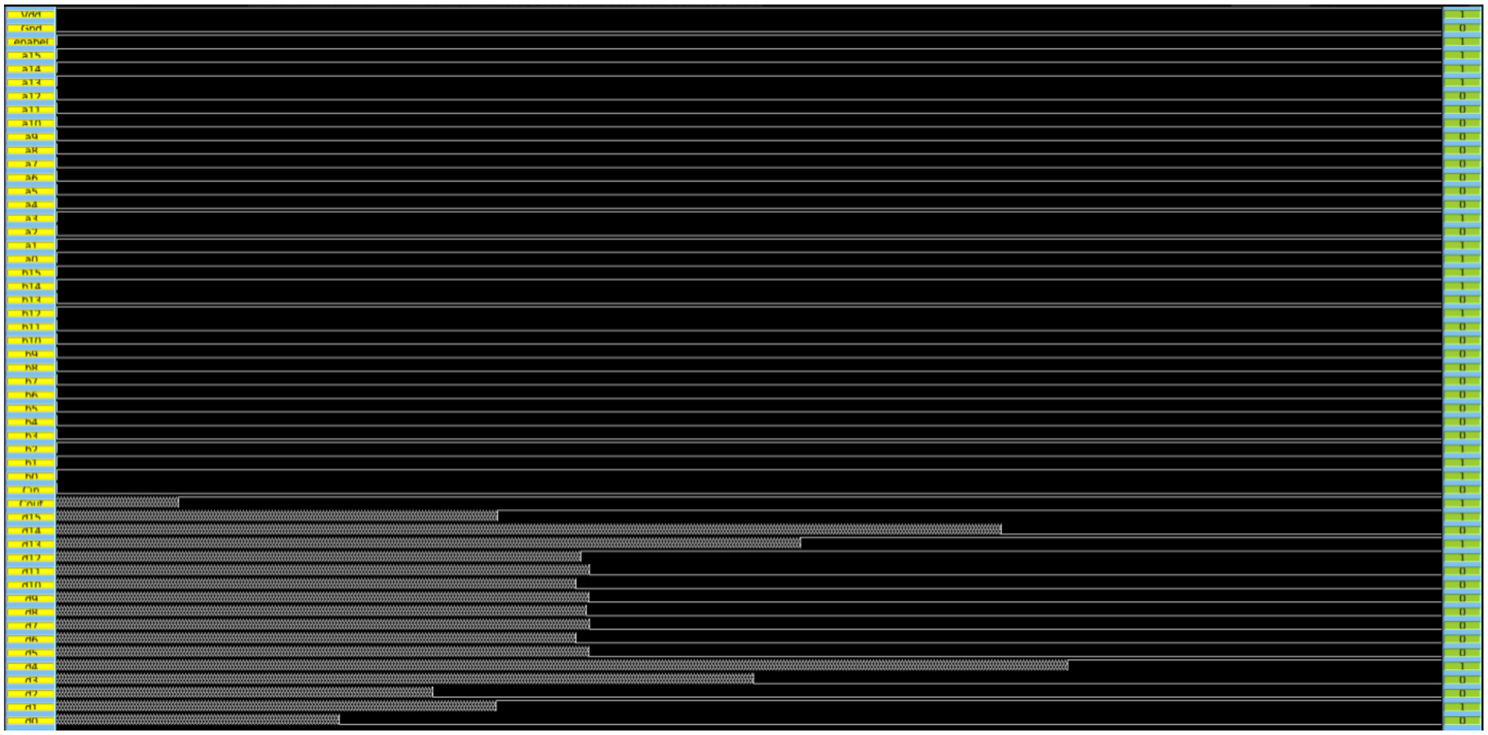
⇨ Input:

a = 1110000000001011

b = 1101000000000111

⇨ Output will be:

d= 1011000000010010



- Test#2 the ALU with Carry in:
  - ⇨ Input:

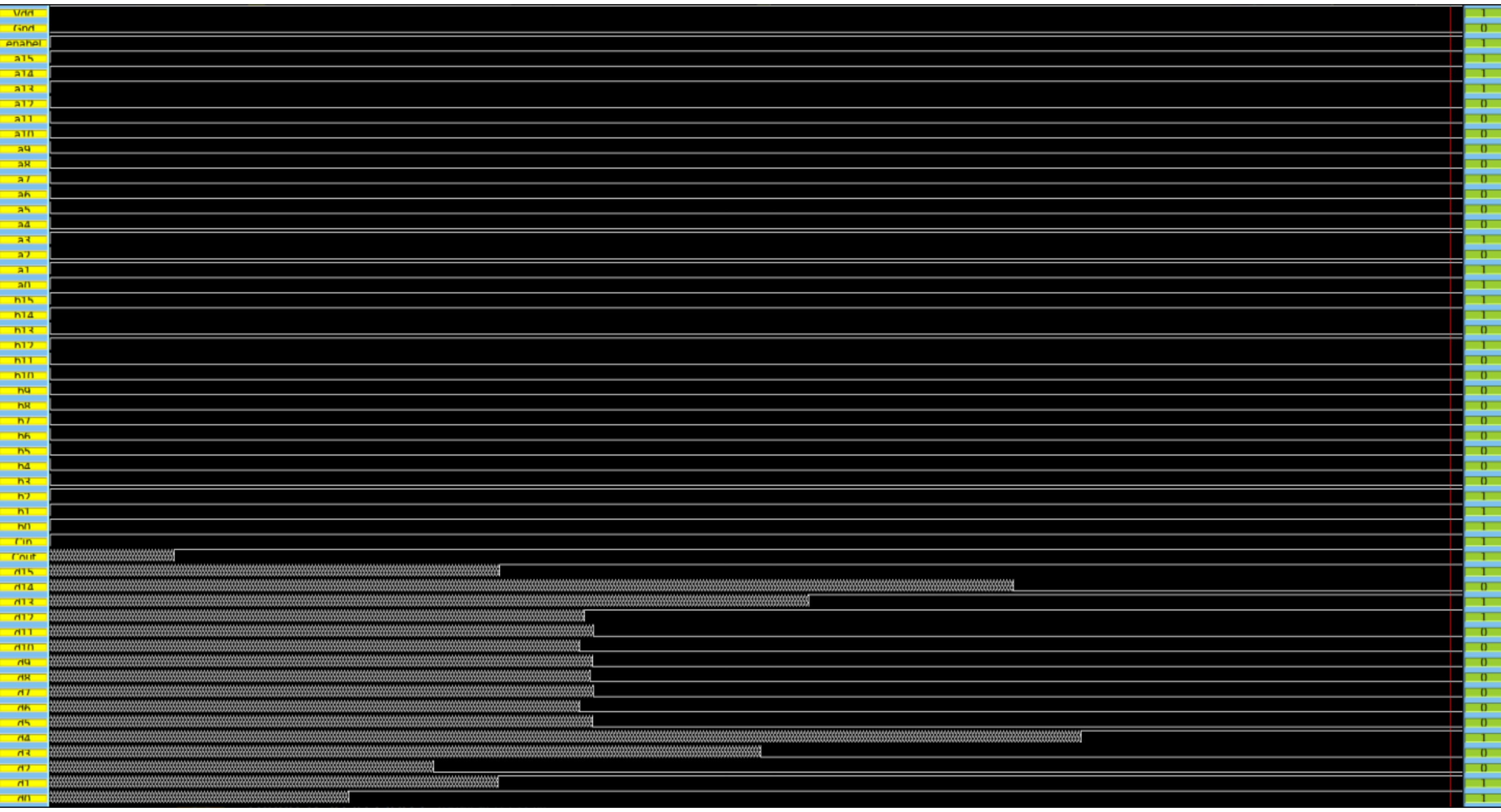
    C_{in} = 1

    a = 1110000000001011

    b = 1101000000000111

  - ⇨ Output will be:

    d = 1011000000010011

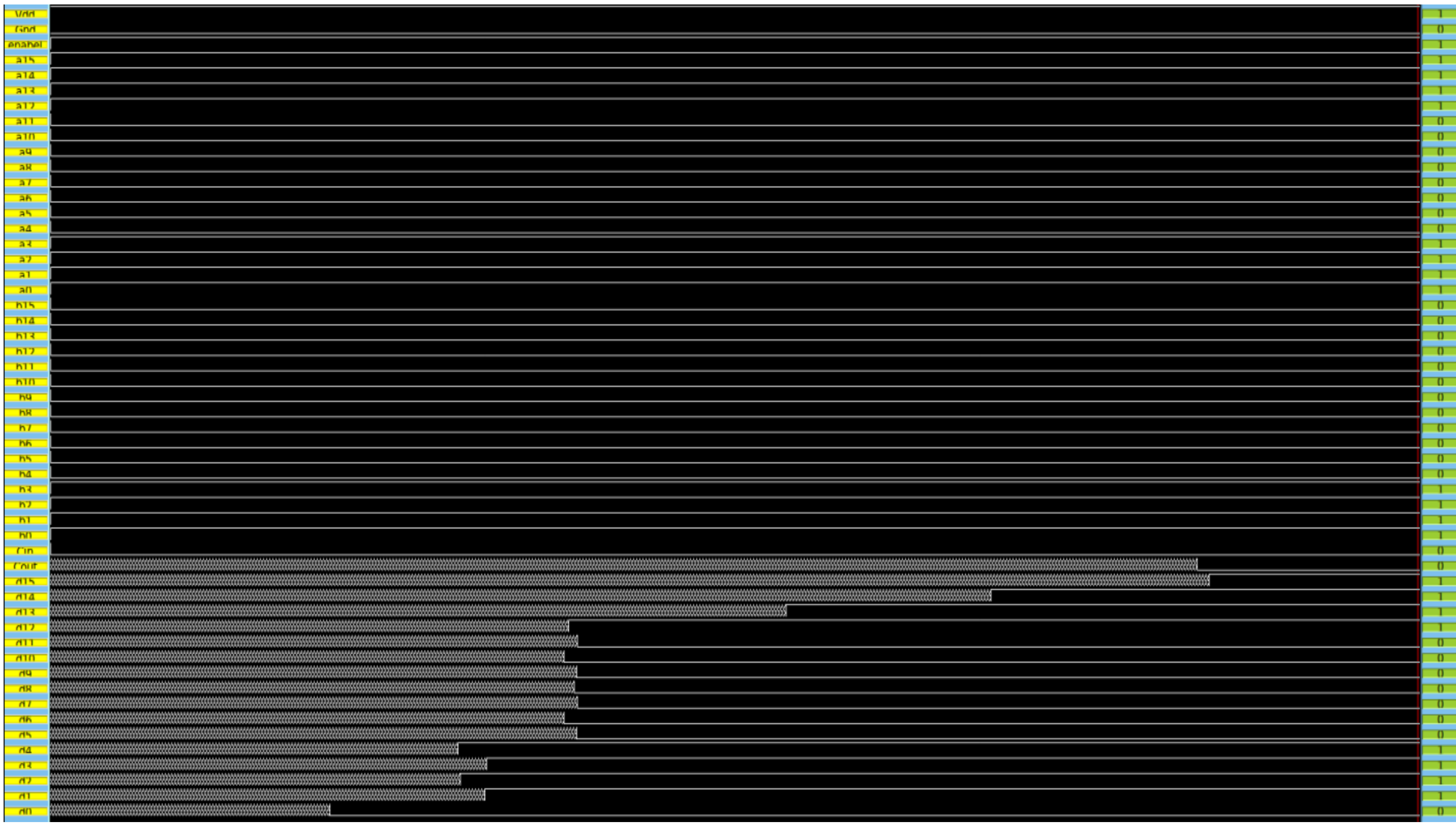


- Test#3 the ALU Extreme Numbers:
  - ⇨ Input:

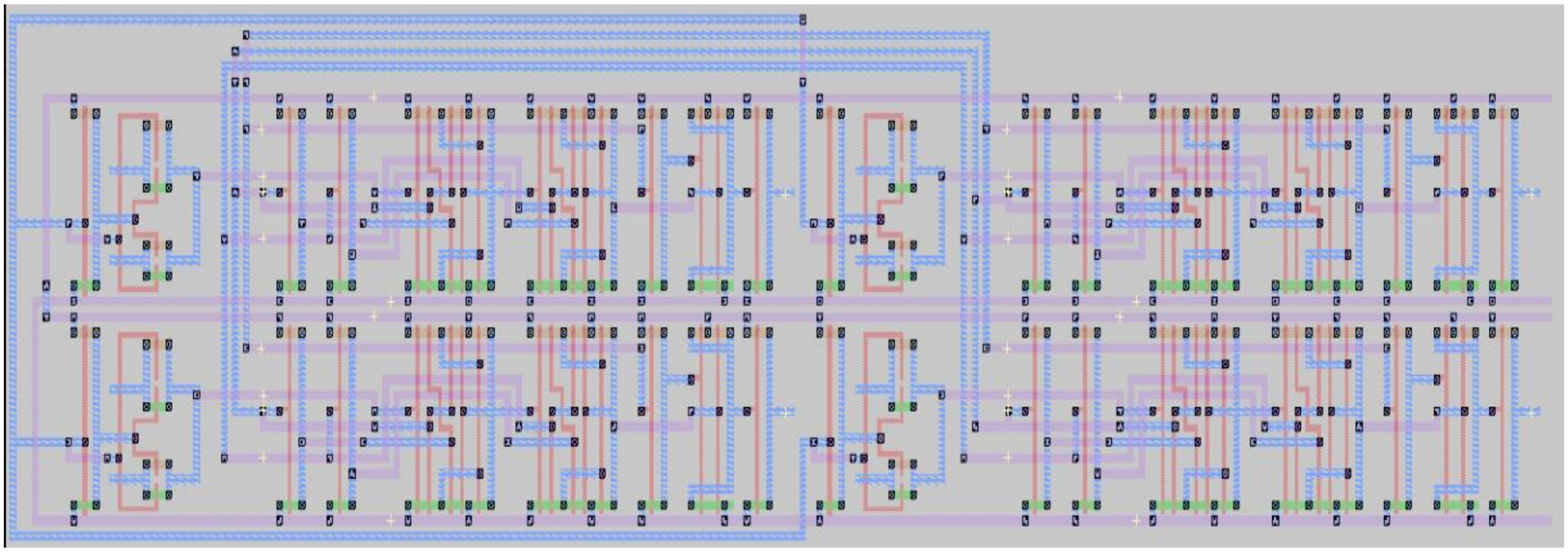
    a = 1111000000001111

    b = 0000000000001111
  - ⇨ Output will be:

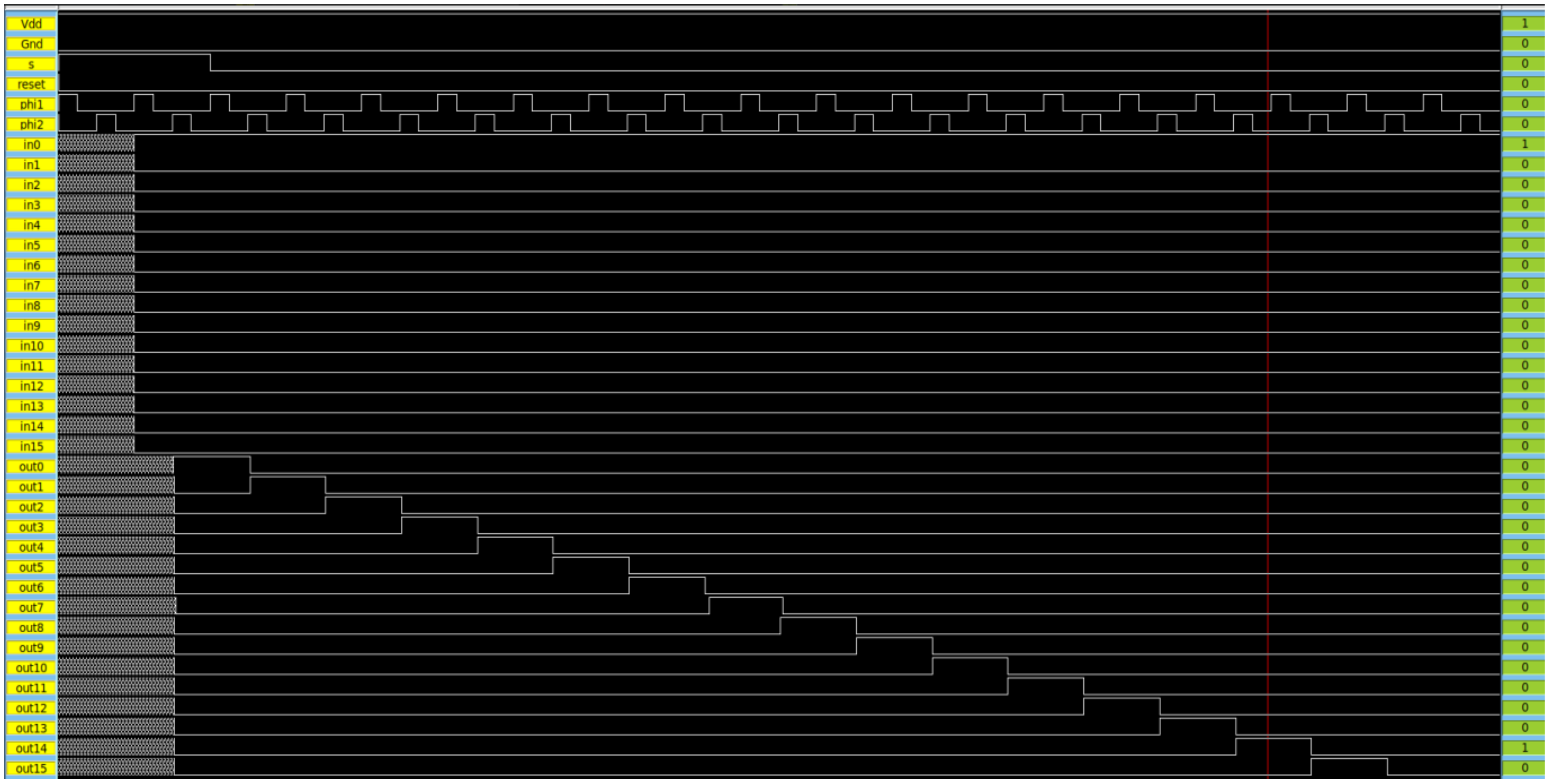
    d = 1111000000011110



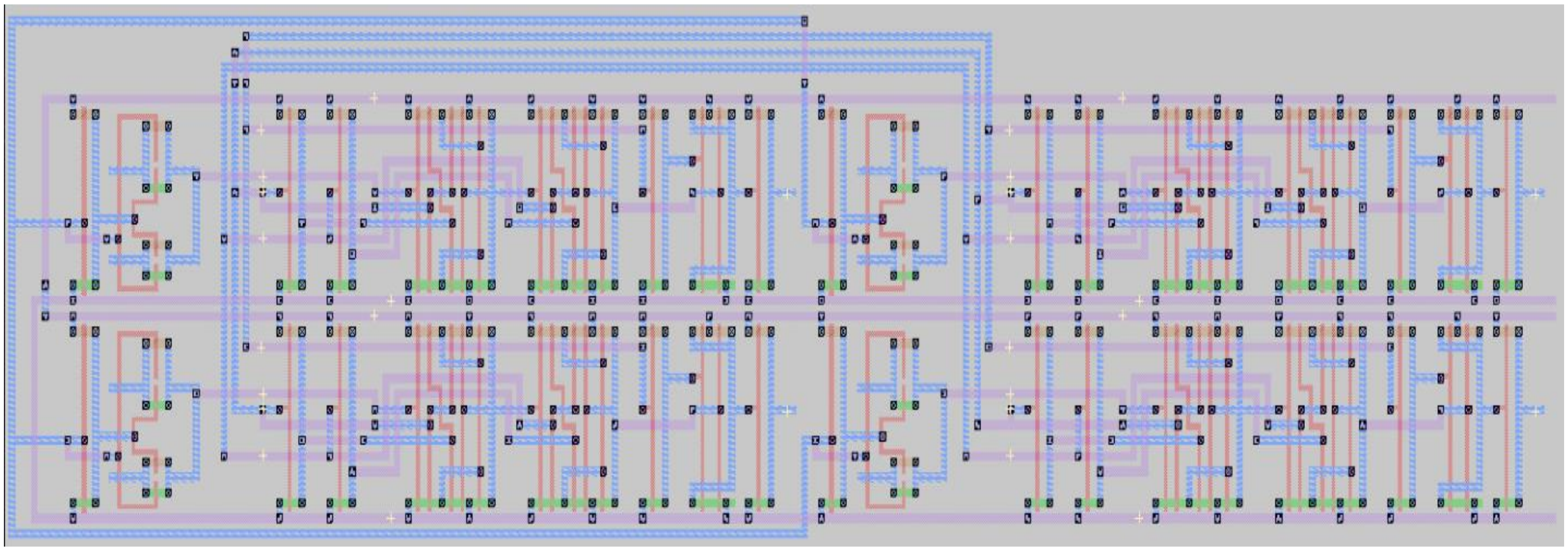
- ***Four-bit Multiplicand-shift lift (We make from 4-bit a 16-bit):***



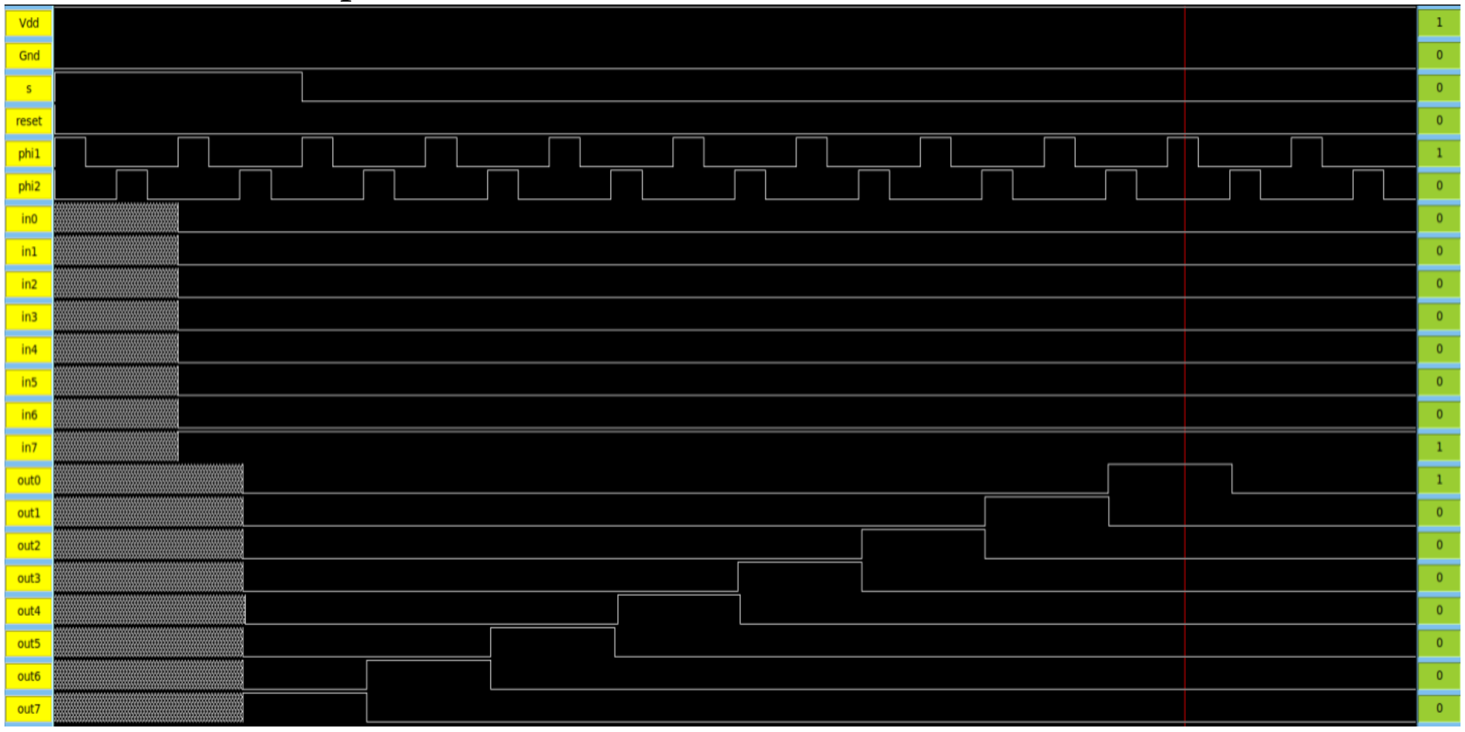
- Test 16-bit multiplicand:
  ⇨ Input in0 = 1



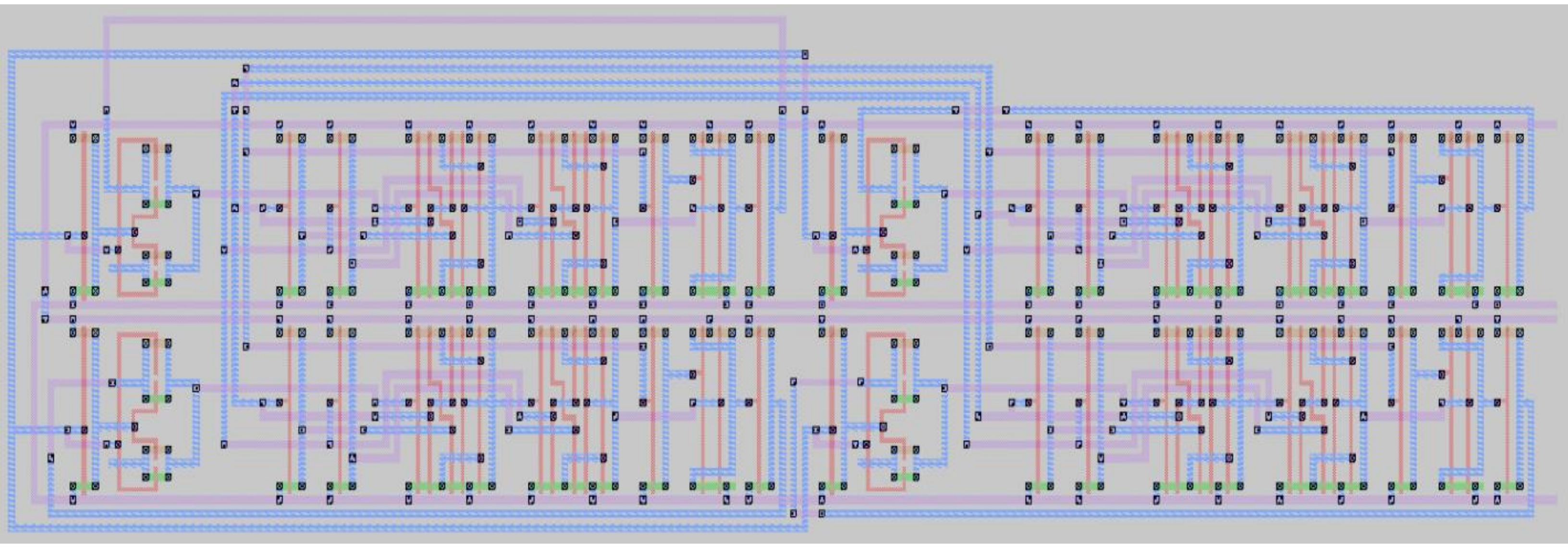
- ***Four-bit Multiplier-shift right (4bit-8bit):***



- Test 8-bit multiplier:
  - ⇨ Input in7 = 1

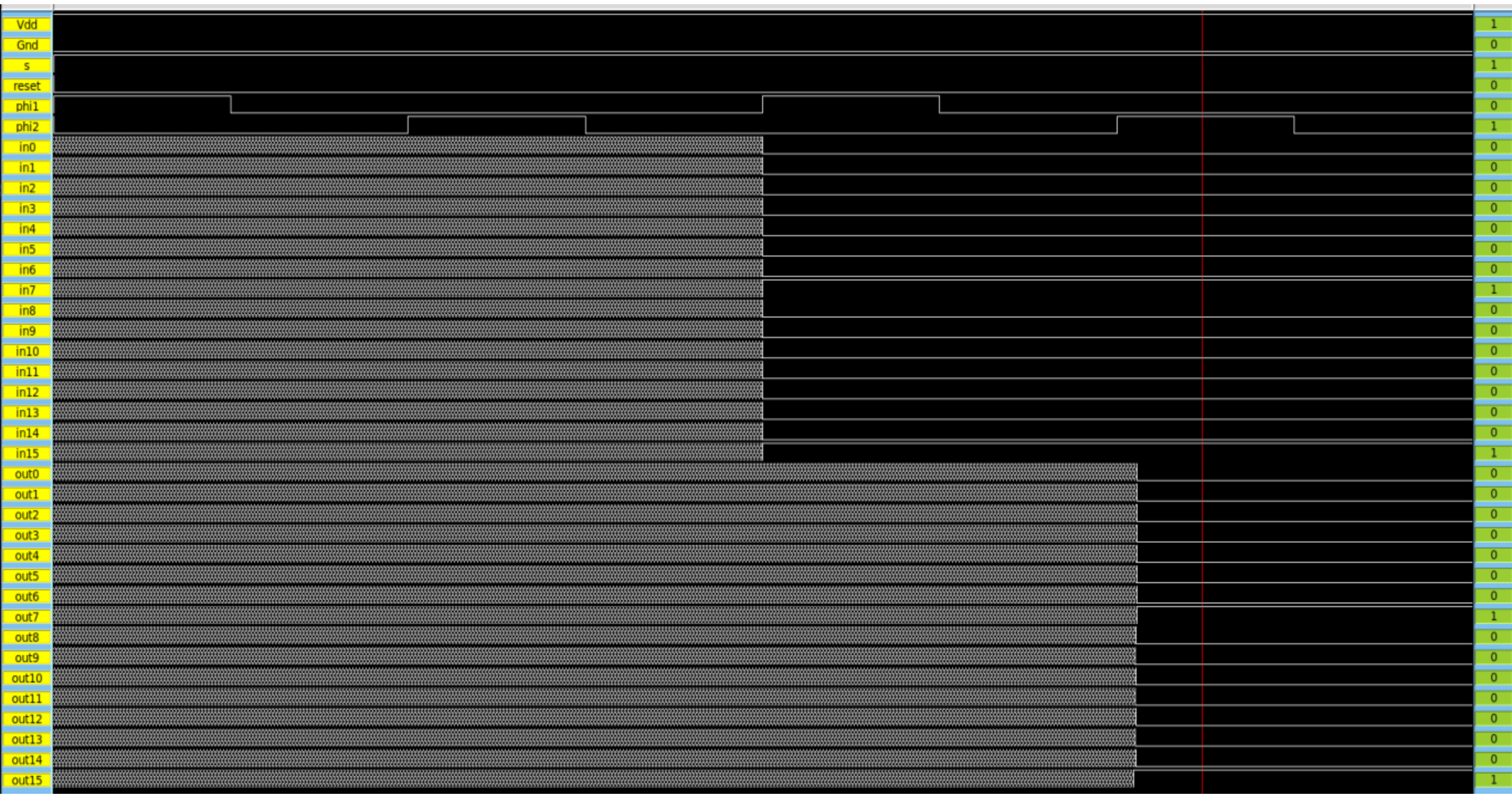


- ***Four-bit Product-shift (4bit-8bit-16bt):***

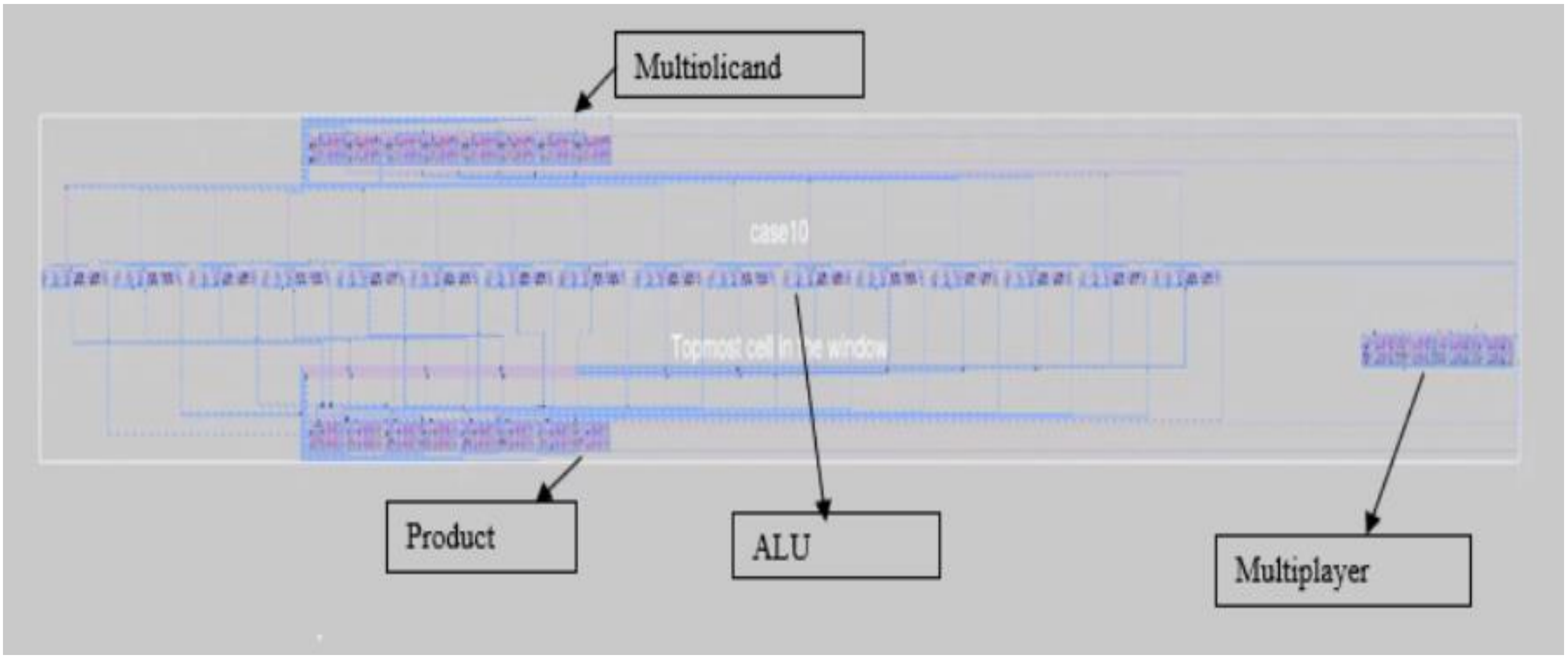


- Test 16-bit product:

  ⇨ Input: in7 = 1 **and** in15 = 1



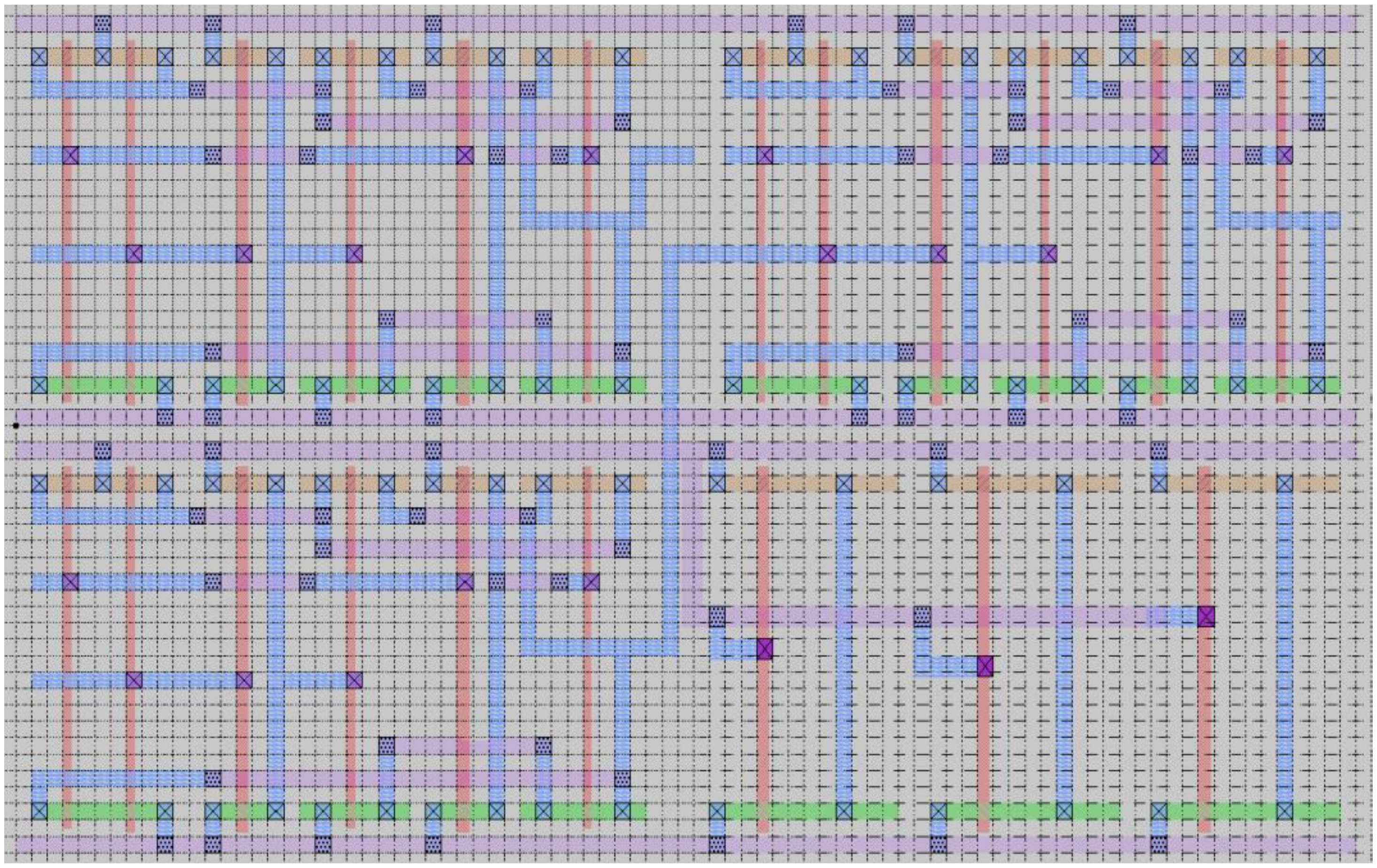
- Final design of case10:



Case 11: even parity:

Input 8 – 14 we added an invertor so we can set the values to zero.

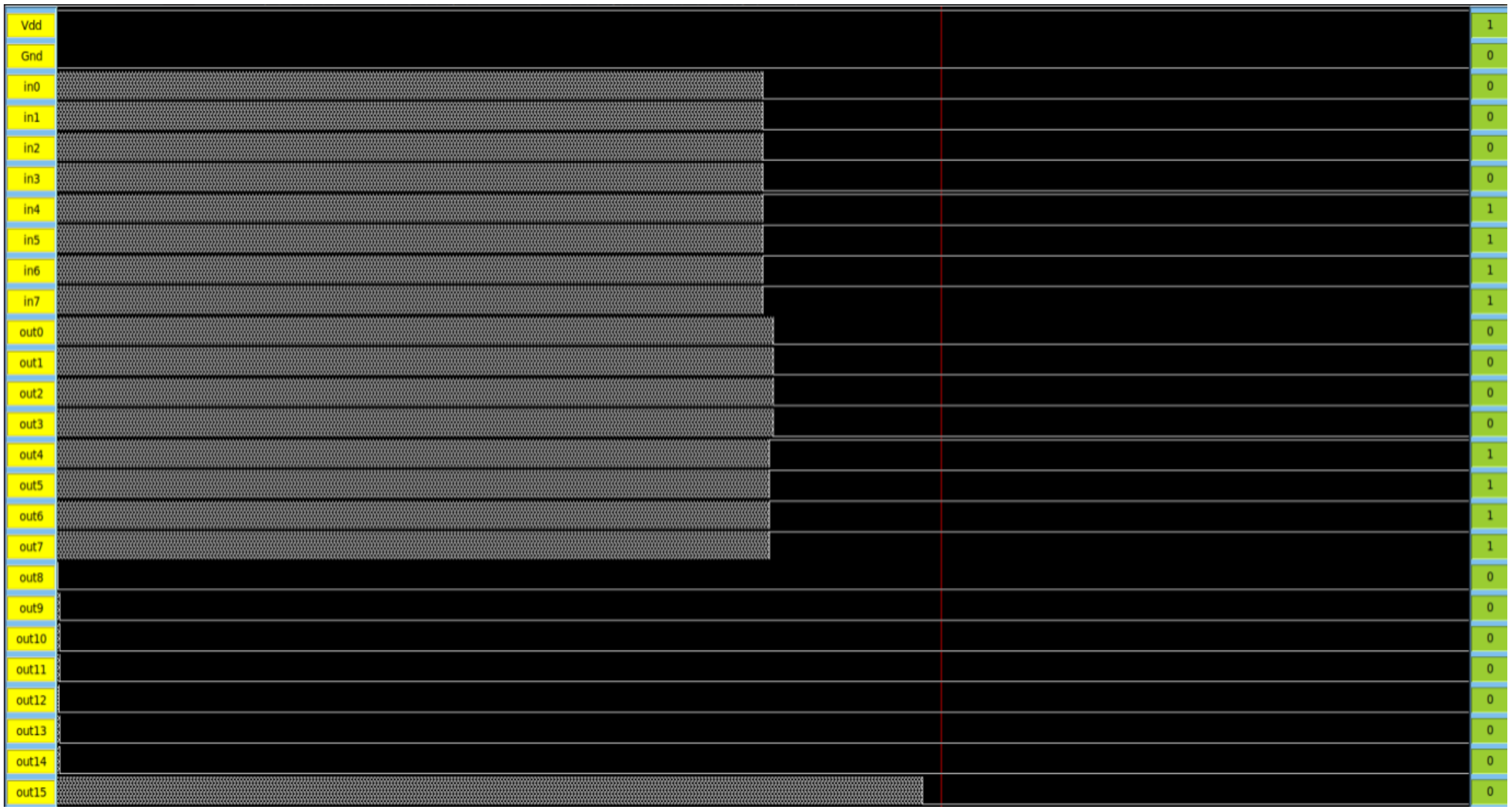
- Layout:



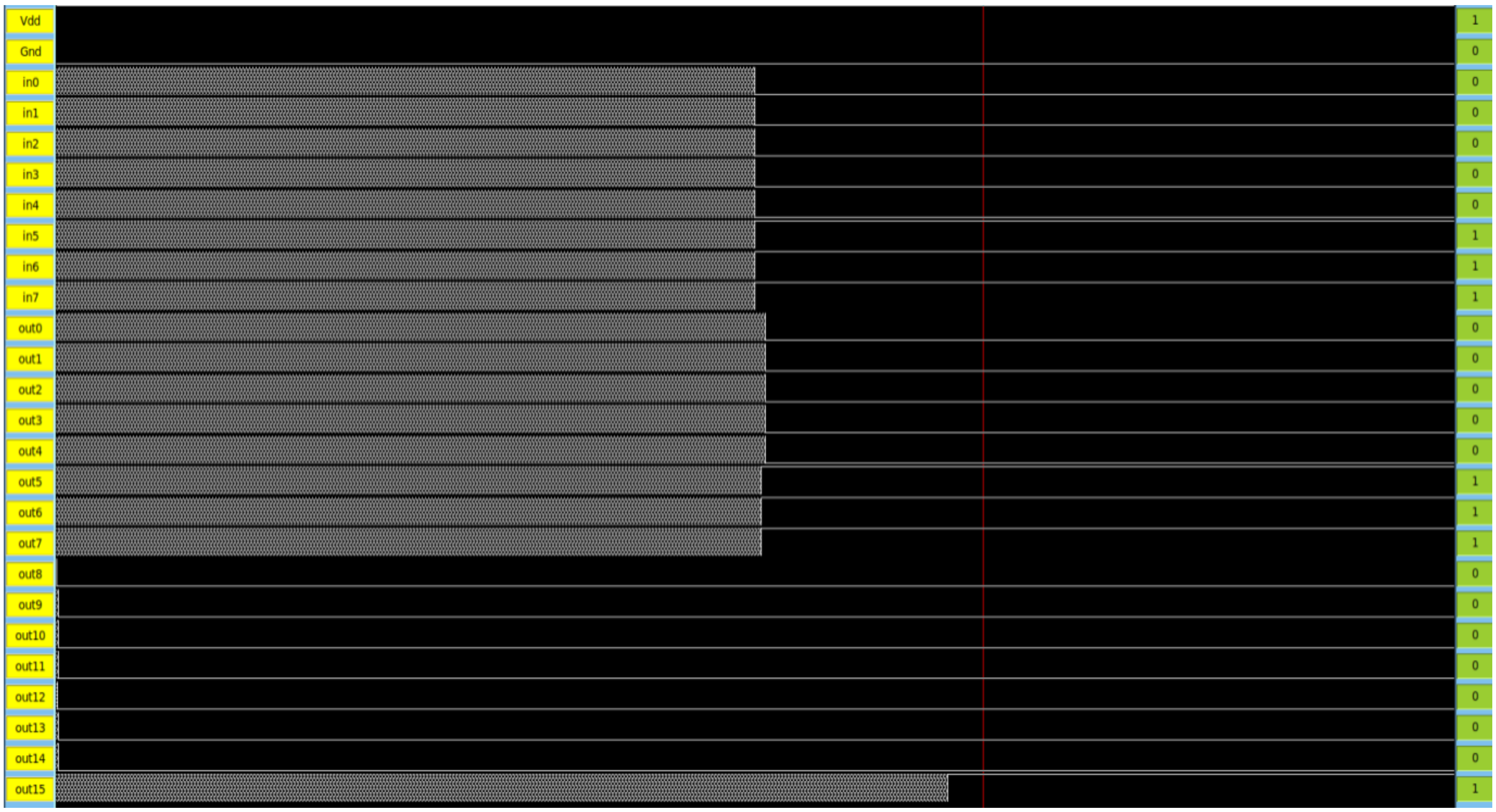
- Test the Case 11:

  ⇨ Input:
  From In7 to In0: 1111 0000

  ⇨ Output will be from out15 to out0:



- Test the Case 11:
  - ⇨ Input:
    From In7 to In0: 1110 0000
  - ⇨ Output will be from out15 to out0:



- **Testing strategy and result:**
  To verify the project work an efficient way, we used a random and extreme input to make sure that the project output is right.