ESE 570 Digital Integrated circuits and VLSI Fundamentals

Final Project

Synchronous 8-bit Fixed Point Arithmetic Logic Unit

Joonyi Koh; Xiangning Li; Han Wang

2014 May 11th

1. Topic investigation

1.1. Background and theory

Arithmetic logic unit (ALU) is a digital circuit that can perform integer arithmetic, logical and shift operations under different operation codes. [1] It is a fundamental building block of the central processing unit (CPU) and many other digital electronics. Typically, the ALU has direct input and output access to the processor controller, main memory, and input/output devices. [2]

In this project, we have successfully designed an 8-bit fixed point arithmetic logic unit with a series of functions. The schematic symbol of the ALU is shown in Fig 1.1. There are three inputs in this ALU: two for 8-bit integer input (A and B), and one for operation code. The output can give information for arithmetic/logical operations (8-bit output with carry out) or comparison (larger, smaller, equal). Besides, in order to keep the outputs synchronous, another input is needed for the clock signal.

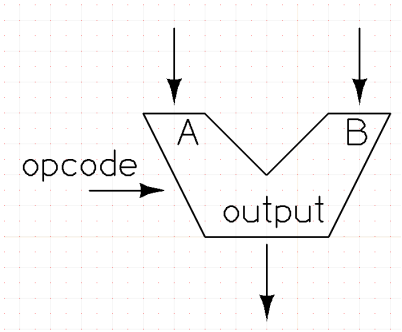


Figure 1.1: Arithmetic and Logic Unit Schematic Symbol (ALU)

1.2. System specification

The architecture of ALU is shown in Fig 1.2 and input/output pins are listed in Table 1.1. There are four main compute blocks: algorithm block, comparator block, logic block and shift/rotate block. They can operate under different operation codes as listed in Table 1.2. Some other blocks here are dynamic DFF with a 2-phase clock to make the outputs synchronous, and several MUXs to select the desired output. By using the MUXs, 4 different blocks can work together.

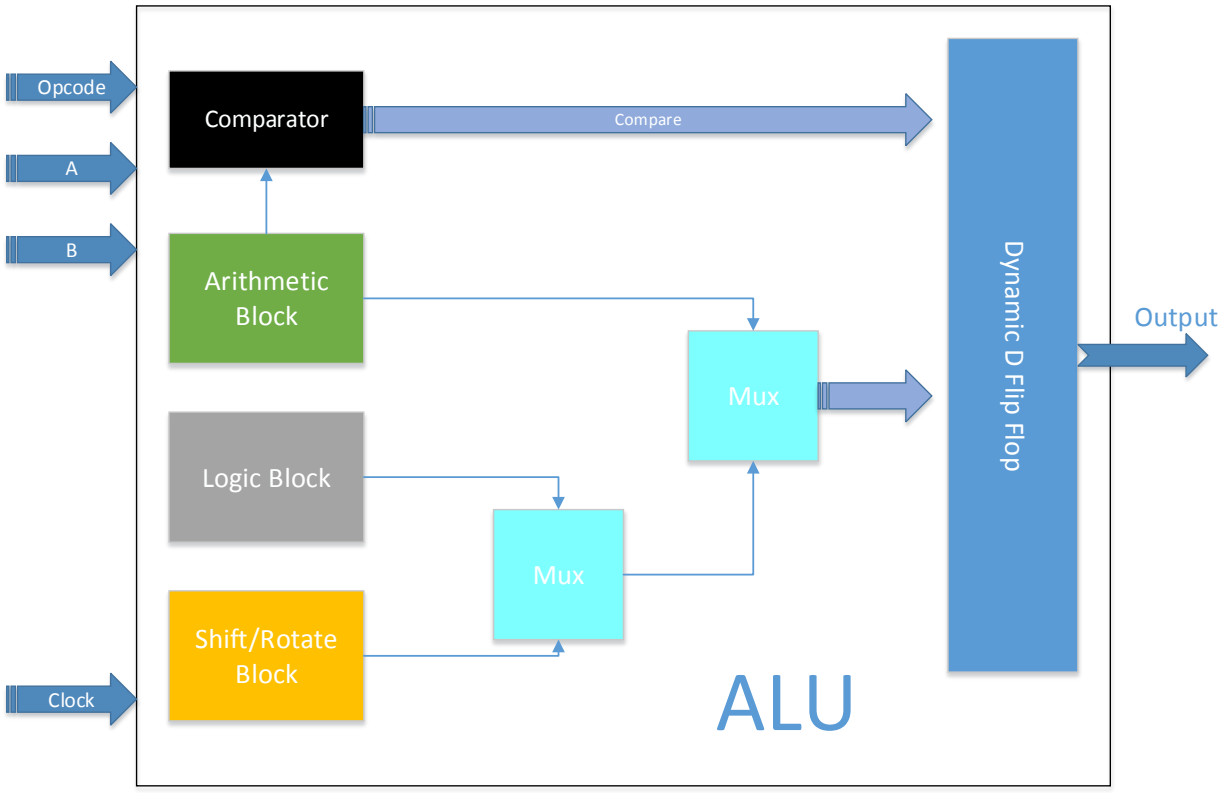


Figure 1.2: Internal Architecture of ALU design

Table 1.1 Pin Descriptions for ALU

	Pin	Description
Input	A0 – A7	8-bit integer input A
	B0 – B7	8-bit integer input B
	S0 – S4	Operation Code
	cin	Carry in for algorithm block
	clkin	Clock input
Output	o0 – o7	8-bit integer output
	cout	Carry out
	A_larger	Comparison output: A>B
	A_smaller	Comparison output: A<B
	equal	Comparison output: A=B

In implementing this project, a bottom-up approach was used to design from the basic gates to the 4 main blocks and finally the whole chip-level ALU. The whole system is under the AMI 0.6 micron technology, and sizes chosen are Wp = 3 um, Lp = 0.6 um for pMOS, and Wn = 1.5 um, Ln = 0.6 um for nMOS. The height for the standard layout design is 24 um. During each design procedure, Verilog functional simulation, transistor level schematic simulation and post layout simulation (PLS) were performed to ensure each module has the desired functions. The ALU chip has a power consumption of 2.672mW and can operate at or slightly larger than 100MHz.

Table 1.2 Operation Code For Different Functions

	Operation Code					**Cin**	**Operation**
	S4	S3	S2	S1	S0		
Algorithm Block	0	X	0	0	0	0	Add (A + B)
	0	X	0	1	0	1	Subtract (A – B)
	0	X	1	1	0	0	Increment (A + 00000001)
	0	X	1	0	0	0	Decrement (A – 00000001)
	0	X	0	1	1	0	1’s complement (Invert B)
	0	X	0	1	1	1	2’s complement (Invert B + 00000001)
Logic Block	1	0	X	0	0	X	XOR
	1	0	X	0	1	X	NOR
	1	0	X	1	0	X	AND
	1	0	X	1	1	X	OR
Shift/Rotate Block	1	1	X	0	0	X	Shift Left
	1	1	X	1	0	X	Shift Right
	1	1	X	0	1	X	Rotate Left
	1	1	X	1	1	X	Rotate Right

2. Design

2.1. ALU Architecture

The schematic architecture for the whole chip level ALU is shown in Fig 2.1 with 4 compute blocks. This ALU is capable to perform algorithm operations (Add, Subtract, Increment, Decrement, 1's complement, 2's complement), comparison operations (Larger, Smaller, Equal), logic operations (XOR, AND, OR, NOR), and shift/rotate left/right operations. Two 8-bit MUX were used for output selection, while four 1-bit dynamic DFF and one 8-bit dynamic DFF were used to make the outputs synchronous. Information of basic gates can be found in the Appendix.

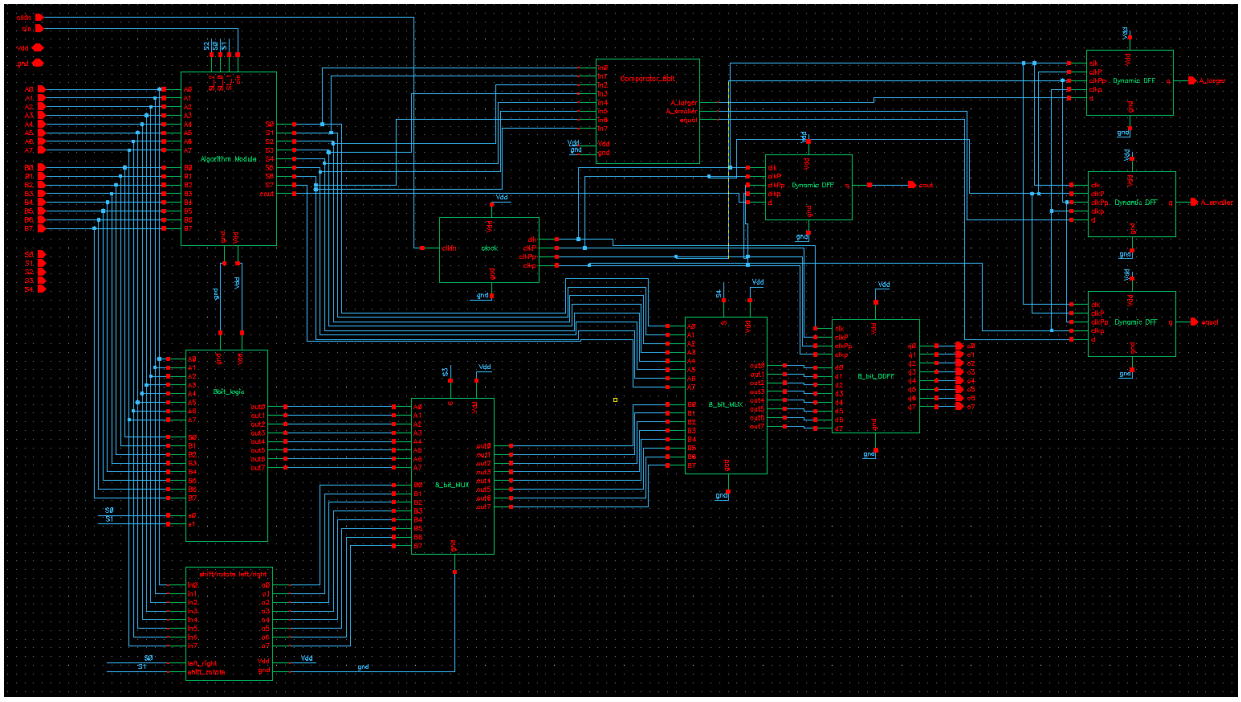


Fig 2.1 Schematic structure of ALU

2.2. Individual block description

2.2.1. Basic gates, clock, Dynamic D-flip flop and Multiplexer

In order to save chip area, we used Dynamic D-flop flop. The output of MUX is input A when the signal is 0; B when the signal is 1. Please check Appendix for the Schematic, layout and PLS of Basic gates, clock, D-flip flop and Multiplexer.

2.2.2. Algorithm Block

2.2.2.1. Adder

The most basic building block of algorithm is the 1-bit adder, which are made of XOR, AND, OR gates. The logic of 1-bit adder is shown in table 2.2.1. The 8-bit full adder was obtained by cascading eight 1-bit adders with inputs A0-A7, B0-B7.

Table 2.2.1 Truth table of 1-bit adder.

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	1	0	1	0
1	1	1	1	1

So the adder logic is:

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\overline{C_{in}} + A\bar{B}\overline{C_{in}} + ABC_{in} = A \oplus B \oplus C_{in}$$

$$C_{out} = BC_{in} + AC_{in} + AB$$

2.2.2.2. Algorithm block

Algorithm block performs add, subtract, 1's complement, 2'scomplement, increment, decrement, and comparison. The core of Algorithm block is an 8-bit full adder. The detailed realization of each operation is explained in the table 2.2.2.

Table 2.2.2 Realization of functions in Algorithm Module

Function	Realization
add	8 bit full adder
substract	By adding the 2st complement of the subtrahend (8 bit input B) which means let B pass an 8-bit inverter and set Cin as 1
1st complement	Let B go pass an 8-bit inverter, set Cin as 0
2nd complement	Let B go pass an 8-bit inverter, set Cin as 1
increment	Set B as 00000001
decrement	Set B as 11111111

2.2.3. Comparator Block

The input for comparator block is the result of the subtract operation of the algorithm block. That is to say, the comparator block needs to go after the algorithm block. By checking the last bit of the subtract operation S7, We could there the not less than (>=) result (When S7 shows 1) and less than result (When S7 shows 0). We could further separate "larger than" and "equal to" by comparing the subtract result to an 8-bit zero.

2.2.4. Logic Block

The 1-bit logic gate can perform operations of XOR, NOR, AND, OR. The outputs are selected by three MUX gates (2-stage) under 2 control signals (s0, s1). The operation code for each logic function is listed in Table 2.2.3. The 8-bit logic block is obtained by cascading the 1-bit logic blocks, just as the 8-bit adder does.

Table 2.2.3 Operation code for 1-bit logic gate

s1	s0	Logic
0	0	XOR
0	1	NOR
1	0	AND
1	1	OR

2.2.5. Shift/Rotate Block

The shift/rotate block is composed of three 8-bit multiplexers to serve the purpose of choosing 4 inputs (shift left, shift right, rotate left, rotate right) to 1 output, using select bit S0 and S1. For an input of A7A6A5A4A3A2A1A0, then we will connect the input wire such that

Shift left: A6A5A4A3A2A1A00.

Shift right: 0A7A6A5A4A3A2A1.

Rotate left: A6A5A4A3A2A1A0A7.

Rotate right: A0A7A6A5A4A3A2A1.

Hence, we can select the desired output by applying the correct select bit S0 and S1.

2.3. Verilog simulation

2.3.1. ALU

When doing the Verilog simulation for ALU, the inputs were set as A = 00001111, B = 00000011. Fig 2.3.1 shows the codes of Verilog. Fig 2.3.2 shows the results of algorithm operations (Table 2.3.1) and Fig 2.3.3 shows the results of logic and shift/rotate operations (Table 2.3.2). One thing needs to be noticed is that the comparison results (larger, smaller, equal) are only valid upon the subtract operation. Thus, we will only consider this part of the results when the algorithm block is doing subtract operations. In this case, the outputs are A_larger =1, A_smaller =0, equal =0. The output cout only matters when doing add and increment operations.

```
//Verilog HDL for "ALU2", "ALU_optimized1" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module ALU_optimized1 ( A_larger, A_smaller, cout, equal, o0, o1, o2, o3, o4,
    o5, o6, o7, Vdd, gnd, A0, A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4,
    B5, B6, B7, S0, S1, S2, S3, S4, cin, clkin );

  input cin, clkin;
  input A0,A1,A2,A3,A4,A5,A6,A7,B0,B1,B2,B3,B4,B5,B6,B7,S0,S1,S2,S3,S4;
  output o0,o1,o2,o3,o4,o5,o6,o7,cout,A_smaller,A_larger,equal;
  inout Vdd, gnd;
   wire  sum0,sum1,sum2,sum3,sum4,sum5,sum6,sum7,cout1;
   wire  out0,out1,out2,out3,out4,out5,out6,out7;
   wire  out00,out11,out22,out33,out44,out55,out66,out77;
   wire  out000,out111,out222,out333,out444,out555,out666,out777;
   wire  o00, o11, o22, o33, o44, o55, o66, o77;
   wire  clk,clkp,clkP,clkPp,A_smaller1,A_larger1,equal1;

   algorithm_module g1 ( sum0, sum1, sum2, sum3, sum4, sum5, sum6, sum7, cout1,
Vdd, gnd, A0, A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4, B5, B6, B7, S0, S1, S2, cin);

   logic_8  g2 ( out0, out1, out2, out3, out4, out5, out6, out7, Vdd, gnd,
A0, A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4, B5, B6, B7, S0, S1);

   shift_rotate g3 ( out00, out11, out22, out33, out44, out55, out66, out77,
Vdd, gnd, A0, A1, A2, A3, A4, A5, A6, A7, S0, S1 );

   mux_8 g5 ( out000, out111, out222, out333, out444, out555, out666, out777, Vdd, gnd, out0, out1,
out2, out3, out4, out5, out6, out7, out00, out11, out22, out33, out44, out55, out66, out77, S3 );
   clk  g4 ( Vdd, clk, clkP, clkPp, clkp, gnd, clkin );
   comparator g6 ( A_larger1, A_smaller1, equal1, Vdd, gnd, sum0,sum1,sum2,sum3,sum4,sum5,sum6,sum7 );
   Dynamic_DFF g7 ( cout, Vdd, gnd, clk, clkP, clkPp, clkp, cout1);
   mux_8 g8 ( o00, o11, o22, o33, o44, o55, o66, o77, Vdd, gnd, sum0, sum1, sum2, sum3, sum4,
sum5, sum6, sum7, out000, out111, out222, out333, out444, out555, out666, out777, S4 );

   Dynamic_DFF_8bit g9 ( o0, o1, o2, o3, o4, o5, o6, o7, Vdd, gnd, clk, clkP, clkPp, clkp,
o00, o11, o22, o33, o44, o55, o66, o77);
   Dynamic_DFF g10 ( A_larger, Vdd, gnd, clk, clkP, clkPp, clkp, A_larger1);
   Dynamic_DFF g11 ( A_smaller, Vdd, gnd, clk, clkP, clkPp, clkp, A_smaller1);
   Dynamic_DFF g12 ( equal, Vdd, gnd, clk, clkP, clkPp, clkp, equal1);

endmodule
`endcelldefine
```

Fig 2.3.1 Codes of ALU's Verilog

Table 2.3.1 Results of algorithm operations

Operation	Add	Subtract	Increment	Decrement	1's complement	2's complement
S4	0	0	0	0	0	0
S2	0	0	1	1	0	0
S1	0	1	1	0	1	1
S0	0	0	0	0	1	1
cin	0	1	0	0	0	1
8-bit Output	00010010	00001100	00010000	00001110	11111100	11111101
cout	0	1	0	1	0	0

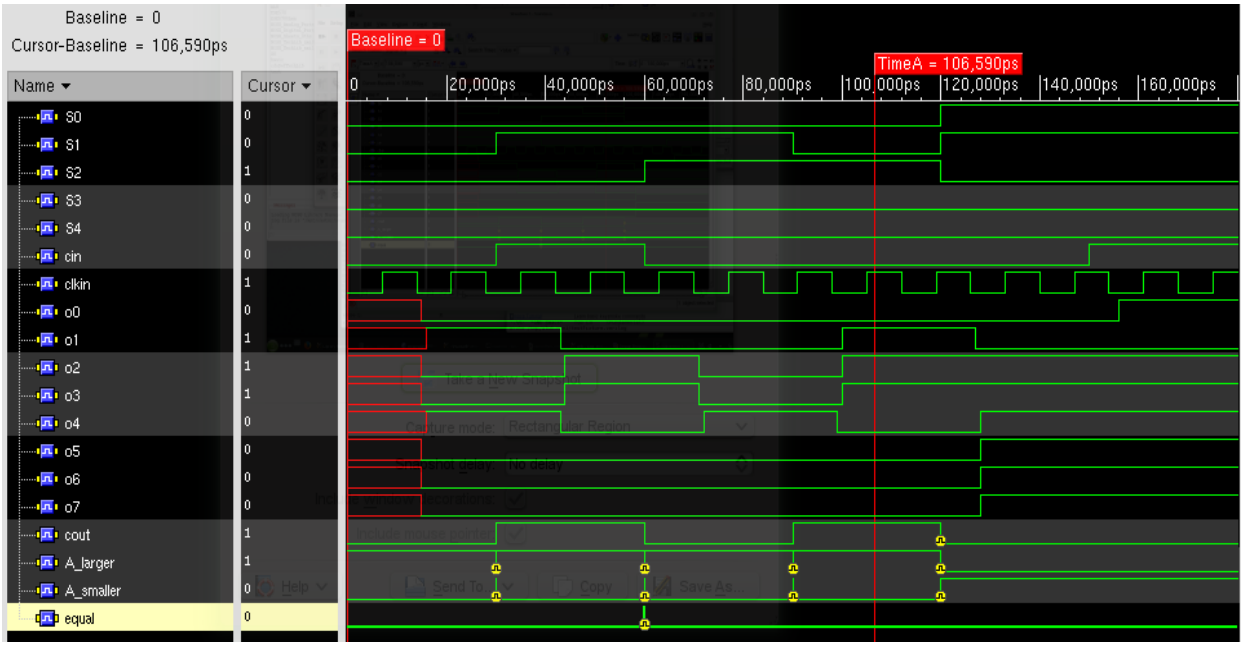


Fig 2.3.2 Results of Verilog simulation of algorithm operations

Table 2.3.2 Results of logic and shift/rotate operations

Operation	XOR	NOR	AND	OR	Shift left	Shift right	Rotate left	Rotate right
S4	1	1	1	1	1	1	1	1
S3	0	0	0	0	1	1	1	1
S1	0	0	1	1	0	0	1	1
S0	0	1	0	1	0	1	0	1
8-bit Output	0000 1100	1111 0000	0000 0011	0000 1111	0001 1110	0000 0111	0001 1110	1000 0111

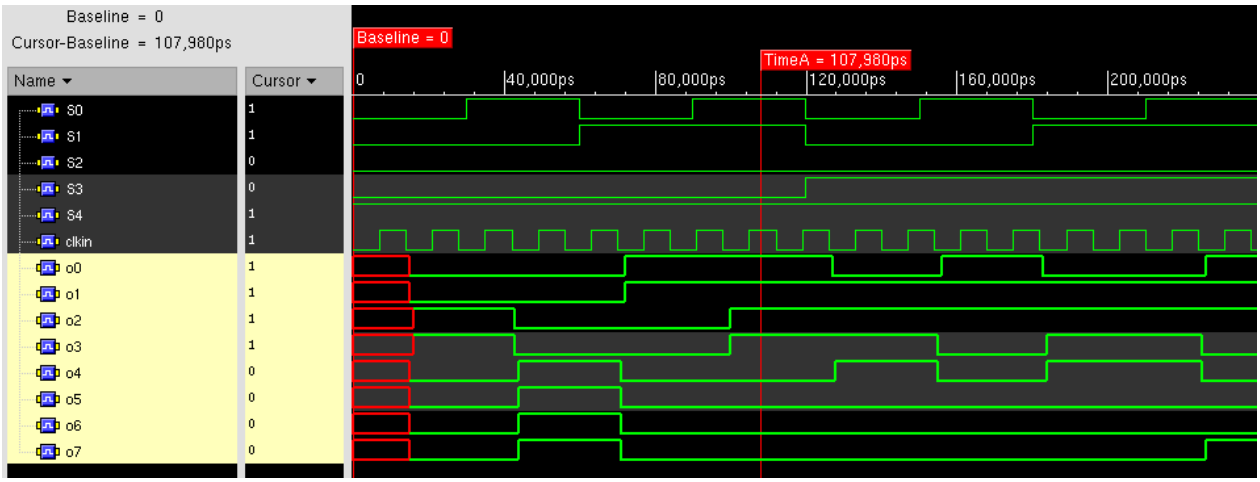


Fig 2.3.3 Results of Verilog simulation of logic and shift/rotate operations

2.3.2. 1-bit adder

The Verilog code for 1-bit adder is shown as Fig 2.3.4 and the results are shown in Fig 2.3.5. The in/out signals for this simulation are listed in Table 2.3.3.

Table 2.3.3 Verilog simulation of 1-bit adder

Period	1	2	3	4	5	6	7	8
A	0	0	0	0	1	1	1	1
B	0	0	1	1	1	1	0	0
cin	0	1	1	0	0	1	1	0
output	0	1	0	1	0	1	0	1
cout	0	0	1	0	1	1	1	0

```
//Verilog HDL for "ALU", "Adder_1bit" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module Adder_1bit ( cout, s, Vdd, gnd, a, b, cin );

  input a;
  inout gnd;
  inout Vdd;
  input cin;
  output s;
  output cout;
  input b;
  wire temp_s,and0,and1;

   xor (temp_s,a,b);
   xor (s,temp_s,cin);
   and (and0,temp_s,cin);
   and (and1,a,b);
   or  (cout,and0,and1);

endmodule
`endcelldefine
```

Fig 2.3.4 Verilog code of 1-bit adder

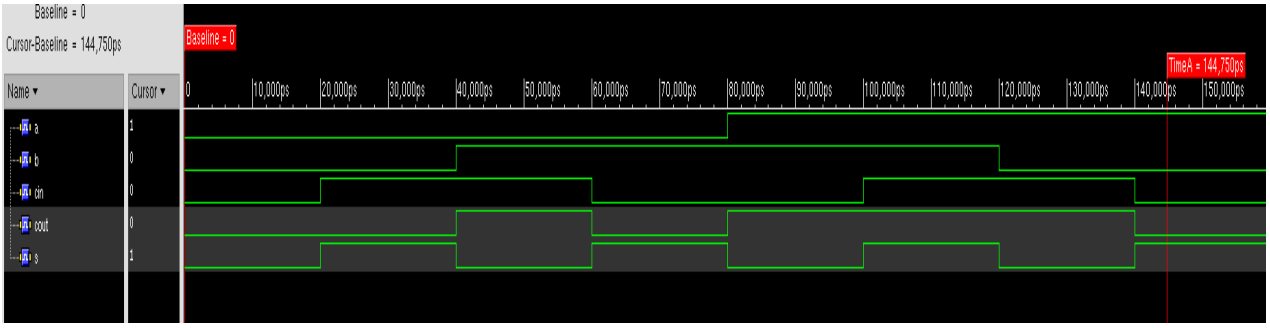


Fig 2.3.5 Verilog simulation of 1-bit adder

2.3.3. 8-bit adder

The results of Verilog code and simulation are shown in Fig 2.3.6 and Fig 2.3.7.

For A=01010000, B=01000101, cin=0, the output=10010101 with cout =0.

For A=10101000, B=10000101, cin=1, the output=00101110 with cout =1.

```
//Verilog HDL for "ALU", "Full_Adder_8bit" "functional"
//module Adder_1bit ( cout, s, Vdd, gnd, a, b, cin );
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module Full_Adder_8bit ( S0, S1, S2, S3, S4, S5, S6, S7, cout, Vdd, gnd, A0,
    A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4, B5, B6, B7, cin );

  input cin;
  input A0,A1,A2,A3,A4,A5,A6,A7;
  output S0,S1,S2,S3,S4,S5,S6,S7;
  input B0,B1,B2,B3,B4,B5,B6,B7;
  output cout;
  inout Vdd;
  inout gnd;
  wire C0,C1,C2,C3,C4,C5,C6;

   Adder_1bit fa0 (C0,S0,Vdd,gnd,A0,B0,cin);
   Adder_1bit fa1 (C1,S1,Vdd,gnd,A1,B1,C0);
   Adder_1bit fa2 (C2,S2,Vdd,gnd,A2,B2,C1);
   Adder_1bit fa3 (C3,S3,Vdd,gnd,A3,B3,C2);
   Adder_1bit fa4 (C4,S4,Vdd,gnd,A4,B4,C3);
   Adder_1bit fa5 (C5,S5,Vdd,gnd,A5,B5,C4);
   Adder_1bit fa6 (C6,S6,Vdd,gnd,A6,B6,C5);
   Adder_1bit fa7 (cout,S7,Vdd,gnd,A7,B7,C6);

endmodule
`endcelldefine
```

Fig 2.3.6 Verilog code of 8-bit full adder

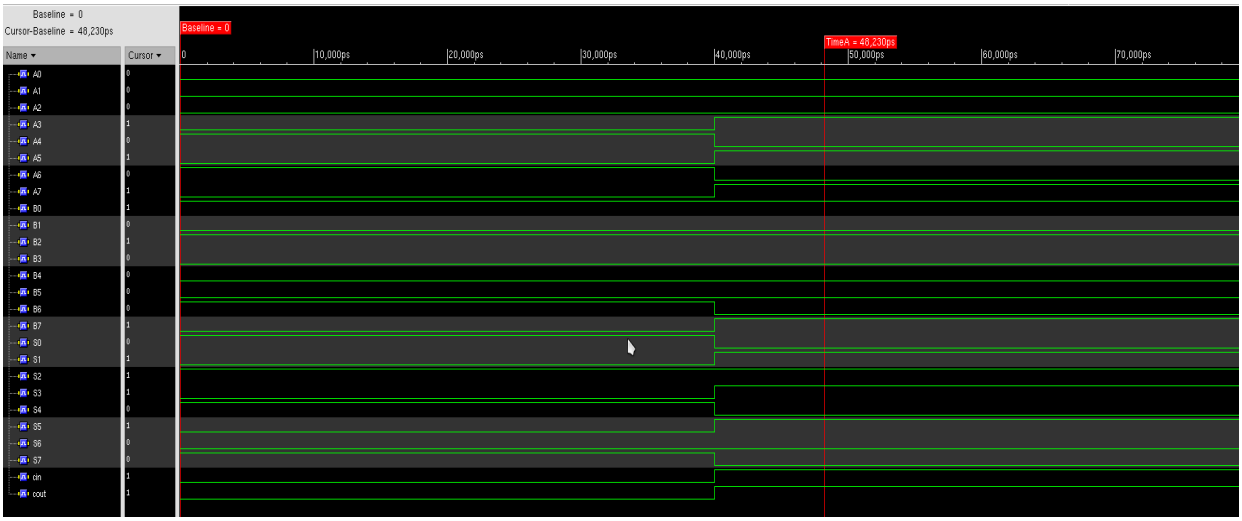


Fig 2.3.7 Verilog simulation of 8-bit full adder

2.3.4. Algorithm block

For A = 10001111, B = 00000011, the results of Verilog simulation are listed in Table 2.3.4. The Verilog code is shown in Fig 2.3.8 and the Verilog simulation result is shown in Fig 2.3.9.

Table 2.3.4 Verilog simulation of algorithm block

Operation	Add	Subtract	Increment	Decrement	1's complement	2's complement
SL_0	0	0	0	0	1	1
SL_1	0	1	1	0	1	1
SL_2	0	0	1	1	0	0
cin	0	1	0	0	0	1
Outpout	10010010	10001100	10010000	10001110	11111100	11111101
cout	0	1	0	1	0	0

```
//Verilog HDL for "ALU", "algorithm_module" "functional"

`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module algorithm_module ( S0, S1, S2, S3, S4, S5, S6, S7, cout, Vdd, gnd, A0,
    A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4, B5, B6, B7, SL_0, SL_1,
    SL_2, cin);

  input cin;
  input A0,A1,A2,A3,A4,A5,A6,A7;
  input B0,B1,B2,B3,B4,B5,B6,B7;
  output S0,S1,S2,S3,S4,S5,S6,S7;
  input SL_0,SL_1,SL_2;
  output cout;
  inout Vdd;
  inout gnd;
   wire outi0, outi1, outi2, outi3, outi4, outi5, outi6, outi7;
   wire outv0, outv1, outv2, outv3, outv4, outv5, outv6, outv7;
   wire outa0, outa1, outa2, outa3, outa4, outa5, outa6, outa7;
   wire outb0, outb1, outb2, outb3, outb4, outb5, outb6, outb7;
   wire outbv0, outbv1, outbv2, outbv3, outbv4, outbv5, outbv6, outbv7;

   mux_8 muxA ( outa0, outa1, outa2, outa3, outa4, outa5, outa6, outa7, Vdd, gnd, A0, A1, A2, A3, A4, A5, A6, A7, gnd,gnd,gnd,gnd,gnd,gnd,gnd,gnd, SL_0 );

   inv_8bit invert1  ( outi0, outi1, outi2, outi3, outi4, outi5, outi6, outi7, Vdd,gnd,B0,B1,B2,B3,B4,B5,B6,B7 );

   mux_8 muxB ( outb0, outb1, outb2, outb3, outb4, outb5, outb6, outb7, Vdd, gnd, B0, B1, B2, B3, B4, B5, B6, B7, outi0, outi1, outi2, outi3, outi4, outi5, outi6, outi7, SL_1 );

   mux_8 vg ( outv0, outv1, outv2, outv3, outv4, outv5, outv6, outv7, Vdd, gnd, Vdd,Vdd,Vdd,Vdd,Vdd,Vdd,Vdd,Vdd, Vdd,gnd,gnd,gnd,gnd,gnd,gnd,gnd, SL_1 );

   mux_8 bv ( outbv0, outbv1, outbv2, outbv3, outbv4, outbv5, outbv6, outbv7, Vdd, gnd, outb0, outb1, outb2, outb3, outb4, outb5, outb6, outb7, outv0, outv1, outv2, outv3, outv4, outv5, outv6, outv7, SL_2 );

   Full_Adder_8bit fa (S0,S1,S2,S3,S4,S5,S6,S7,cout,Vdd,gnd,outa0, outa1, outa2, outa3, outa4, outa5, outa6, outa7, outbv0, outbv1, outbv2, outbv3, outbv4, outbv5, outbv6, outbv7, cin);

endmodule
`endcelldefine
```

Fig 2.3.8 Verilog codes of algorithm block

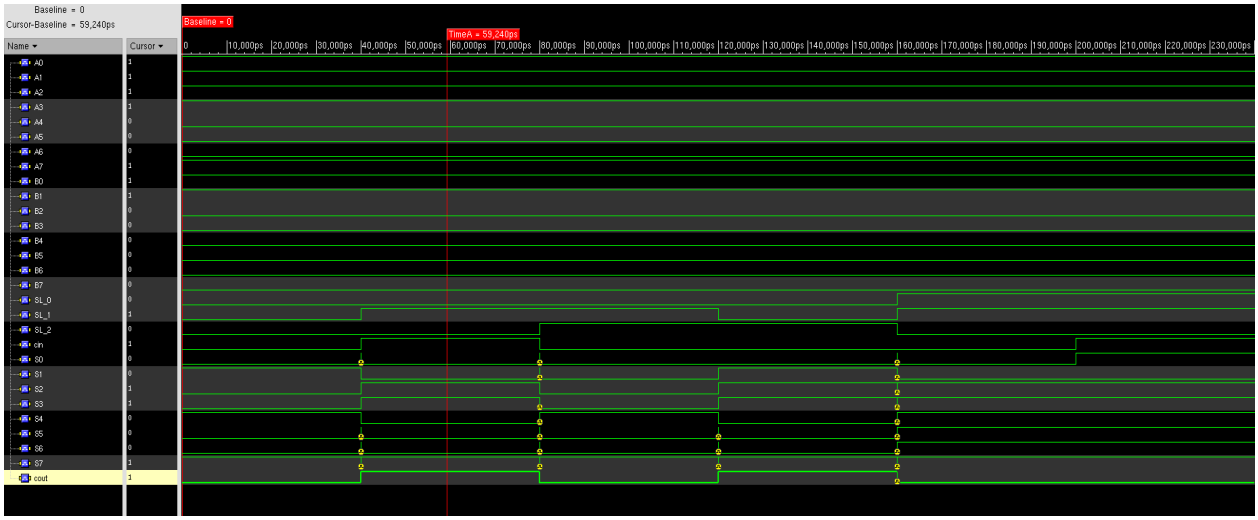


Fig 2.3.9 Verilog simulation of algorithm block

2.3.5. Comparator Block

The Verilog code and simulation is shown in Fig 2.3.10 and 2.3.11.

When the input is 00000000, A_larger = 0, A_smaller = 0, equal =1;

When the input is 00010000, A_larger = 1, A_smaller = 0, equal =0;

When the input is 10000000, A_larger = 0, A_smaller = 1, equal =0;

```
//Verilog HDL for "ALU", "comparator" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module comparator ( A_larger, A_smaller, equal, Vdd, gnd, in0, in1, in2, in3,
    in4, in5, in6, in7 );

  input in0,in1,in2,in3,in4,in5,in6,in7;
  inout gnd;
  inout Vdd;
  output equal;
  output A_smaller;
  output A_larger;
  wire  nor1,nor2,nor3,nor4,nor5,nor6,and1,and2;

   nor (nor1,in0,in1);
   nor (nor2,in2,in3);
   nor (nor3,in4,in5);
   nor (nor4,in6,in7);
   and (and1,nor1,nor2);
   and (and2,nor3,nor4);
   nand (nand1,and1,and2);
   nor (equal,nand1,gnd);
   nor (nor6,in7,gnd);
   not (A_smaller,nor6);
   xor (A_larger,equal,nor6);

endmodule
`endcelldefine
```

Fig 2.3.10 Verilog codes of comparator block

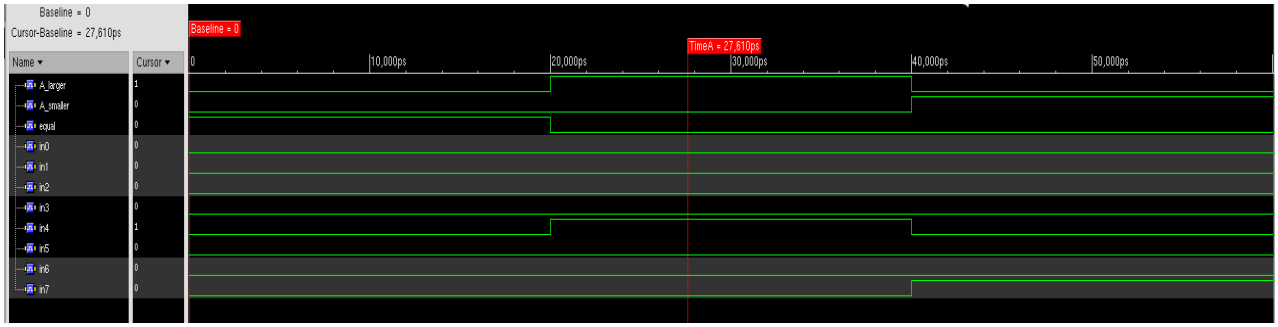


Fig 2.3.11 Verilog simulation of comparator block

2.3.6. 1-bit logic block

By setting A = 0 and B =1, the 1-bit logic gate can successfully carry out all the four logical operations, codes and simulations are shown in Fig 2.3.12 and Fig 2.3.13.

```
//Verilog HDL for "ALU", "logic" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module logic ( out, Vdd, gnd, A, B, s0, s1 );

  inout gnd;
  inout Vdd;
  input s0;
  input s1;
  input A;
  input B;
  output out;

  wire  xor1,nor1,and1,or1;
  wire w1,w2;

   xor(xor1, A, B);
   nor(nor1, A, B);
   and(and1, A, B);
   or(or1, A, B);

   mux n1 (w1, Vdd, gnd, xor1, nor1, s0);
   mux n2 (w2, Vdd, gnd, and1, or1, s0);
   mux n3 (out, Vdd, gnd, w1, w2, s1);


endmodule
`endcelldefine
```

Fig 2.3.12 Verilog code of 1-bit logic block

Fig 2.3.13 Verilog simulation of 1-bit logic block

2.3.7. 8-bit logic block

By setting A = 00001111 and B = 11000011, the results for the Verilog code and simulation is shown in Fig 2.3.14 and Fig 2.3.15. Four different logic operations were successfully carried out, with the results listed in Table 2.3.5.

Table 2.3.5 Verilog simulation of 8-bit logic gate

s1	s0	Logic	Output
0	0	XOR	11001100
0	1	NOR	00110000
1	0	AND	00000011
1	1	OR	11001111

```
//Verilog HDL for "ALU", "8bit_logic" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module logic_8  ( out0, out1, out2, out3, out4, out5, out6, out7,
    Vdd, gnd, A0, A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4,
    B5, B6, B7, s0, s1);

  input s1;  output out7;
  output out6;  input A0;
  output out1;  input A7;
  input A3;  input B5;
  input B1;  output out4;
  input A1;  input A6;
  input A4;  input B2;
  input B6;  inout Vdd;
  output out0;  input A2;
  input A5;  input B7;
  inout gnd;  input s0;
  output out3;  output out5;
  input B0;  input B4;
  output out2; input B3;

   logic n1 (out0, Vdd, gnd, A0, B0, s0, s1);
   logic n2 (out1, Vdd, gnd, A1, B1, s0, s1);
   logic n3 (out2, Vdd, gnd, A2, B2, s0, s1);
   logic n4 (out3, Vdd, gnd, A3, B3, s0, s1);
   logic n5 (out4, Vdd, gnd, A4, B4, s0, s1);
   logic n6 (out5, Vdd, gnd, A5, B5, s0, s1);
   logic n7 (out6, Vdd, gnd, A6, B6, s0, s1);
   logic n8 (out7, Vdd, gnd, A7, B7, s0, s1);

endmodule
`endcelldefine
```

Fig 2.3.14 Verilog code of 8-bit logic block

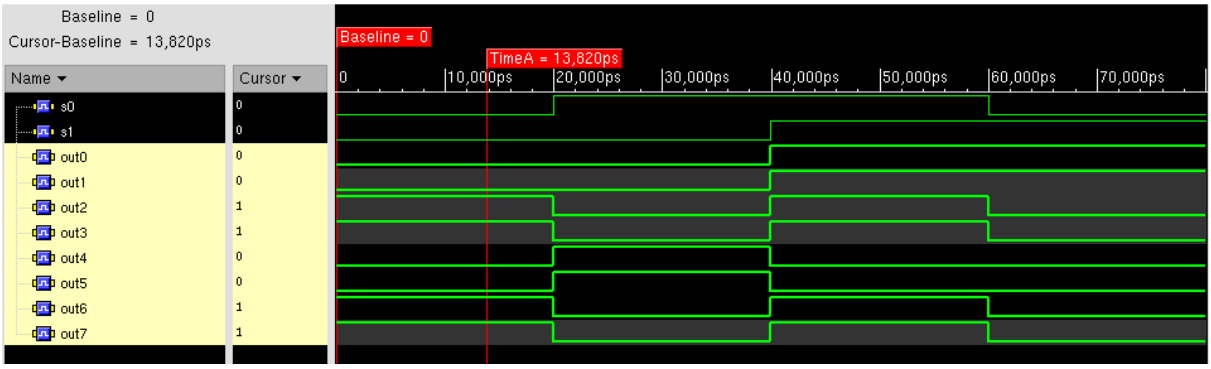


Fig 2.3.15 Verilog simulation of 8-bit logic block

2.3.8. Shift/Rotate Block

For the input=11001101, the shift/rotate block can operate under codes as listed in Table 2.3.6. And the Verilog code and simulations are shown in Fig 2.3.16 and Fig 2.3.17.

Table 2.3.6 Verilog simulation of shift/rotate block

shift_rotate	left_right	output function	output
0	0	shift left	10011010
0	1	shift right	01100110
1	0	rotate left	10011011
1	1	rotate right	11100110

```
//Verilog HDL for "ALU", "shift_rotate" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps


module shift_rotate ( o0,o1,o2,o3,o4,o5,o6,o7, Vdd, gnd,
    in0, in1, in2, in3, in4, in5, in6, in7, left_right, shift_rotate );

  input shift_rotate;
  input in0,in1,in2,in3,in4,in5,in6,in7;
  inout gnd;
  inout Vdd;
  output o0,o1,o2,o3,o4,o5,o6,o7;
  input left_right;
  wire a,b,c,d,e,f,g,h,aa,bb,cc,dd,ee,ff,gg,hh;

   mux_8 m1 (a,b,c,d,e,f,g,h,Vdd,gnd,gnd,in0,in1,in2,in3,in4,in5,in6,in7,in0,in1,  in2,in3,in4,in5,in6,shift_rotate);

   mux_8 m2 (aa,bb,cc,dd,ee,ff,gg,hh,Vdd,gnd,in1,in2,in3,in4,in5,in6,in7,gnd,in1,in2,in3,in4,in5,in6,in7,in0,shift_rotate);

   mux_8 m3 (o0,o1,o2,o3,o4,o5,o6,o7,Vdd,gnd,a,b,c,d,e,f,g,h,aa,bb,cc,dd,ee,ff,gg,hh,left_right);


endmodule
`endcelldefine
```

Fig 2.3.16 Verilog simulation of shift/rotate block

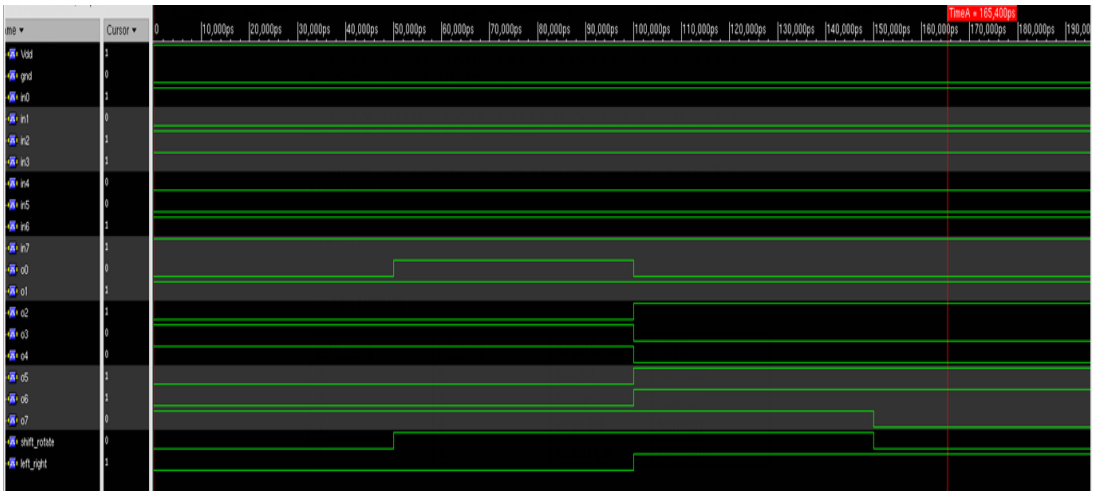


Fig 2.3.17 Verilog simulation of shift/rotate block

2.4. Transistor level schematics and simulations

2.4.1. 1-bit adder

The schematic and symbol of 1-bit adder are shown in Fig 2.4.1 and Fig 2.4.2.

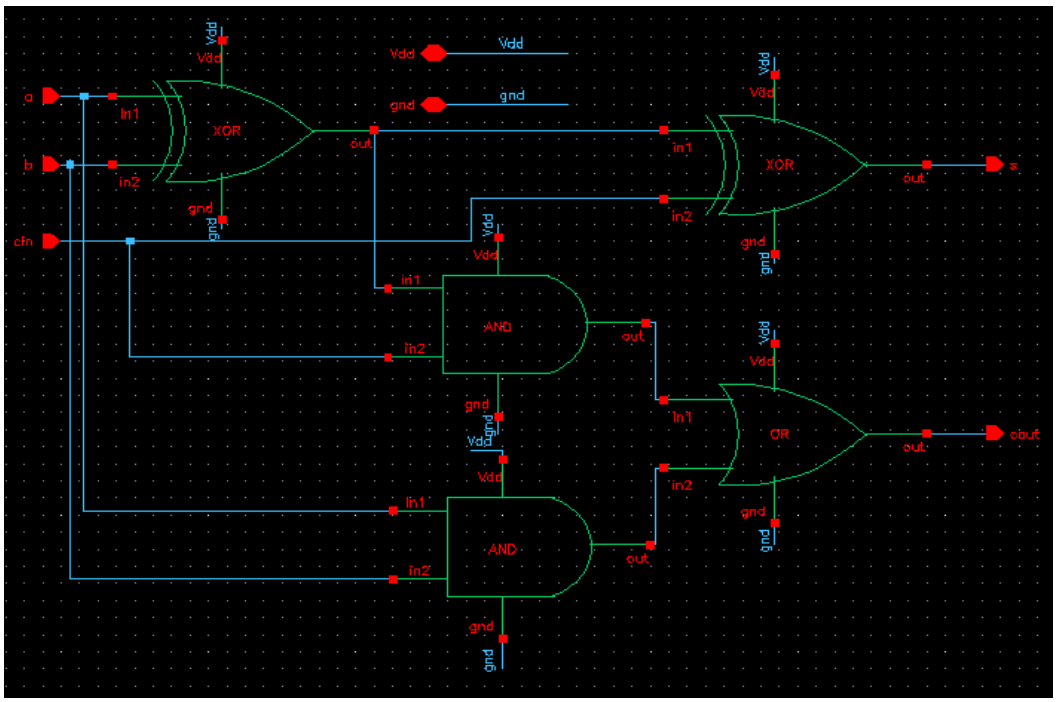


Fig 2.4.1 Schematic of 1-bit adder

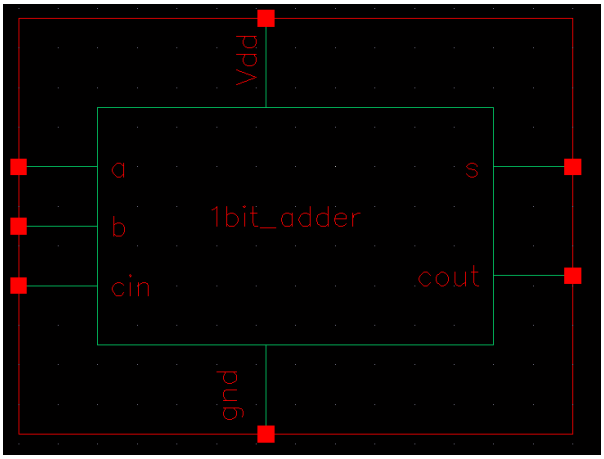


Fig 2.4.2 Symbol of 1-bit adder

2.4.2. 8-bit full adder

The schematic and symbol of 8-bit full adder are shown in Fig 2.4.3.

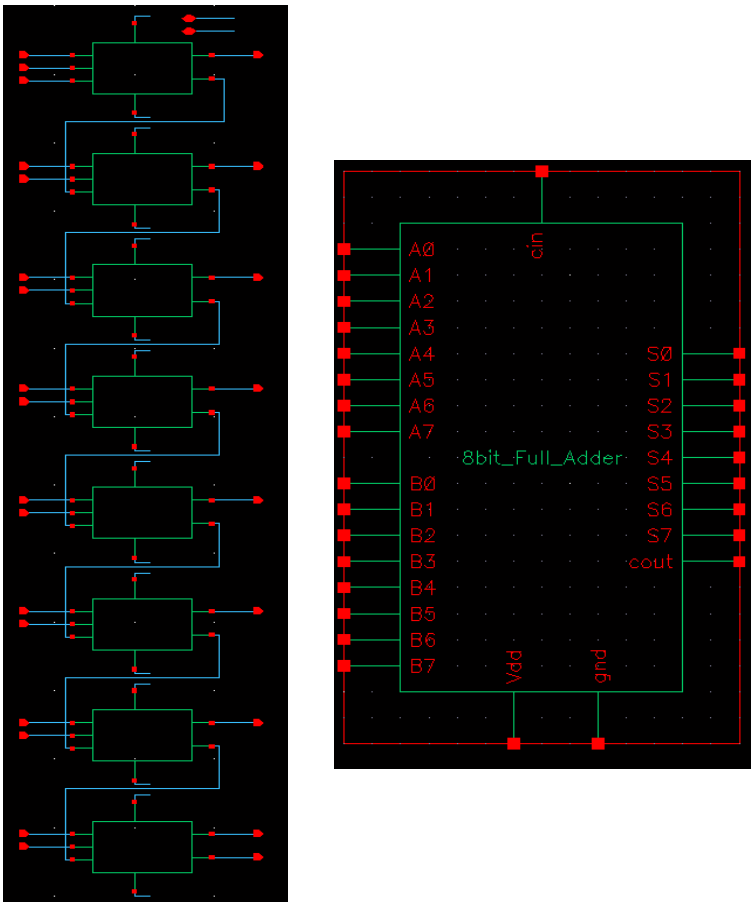


Fig 2.4.3 Schematic and symbol of 8-bit full adder

Fig 2.4.4 gives the test bench of the 8-bit full adder. In order to do the simulation, set the input A as 11111111, input B oscillating from 00000000 to 00000001. Then the output should be oscillating from 11111111(with cout=0) to 00000000 (with cout =1), as shown in Fig 2.4.5.

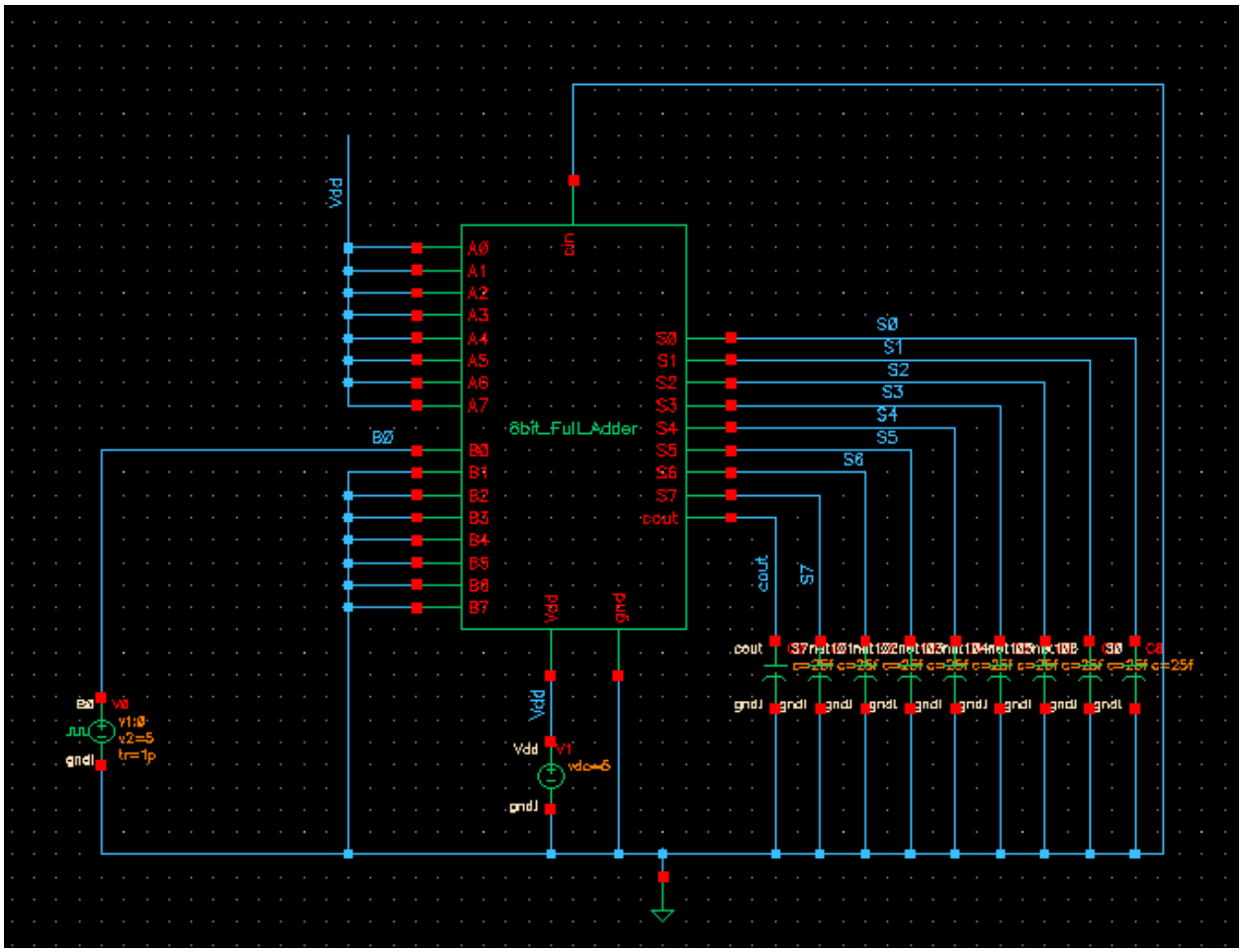


Fig 2.4.4 Test bench of 8-bit full adder

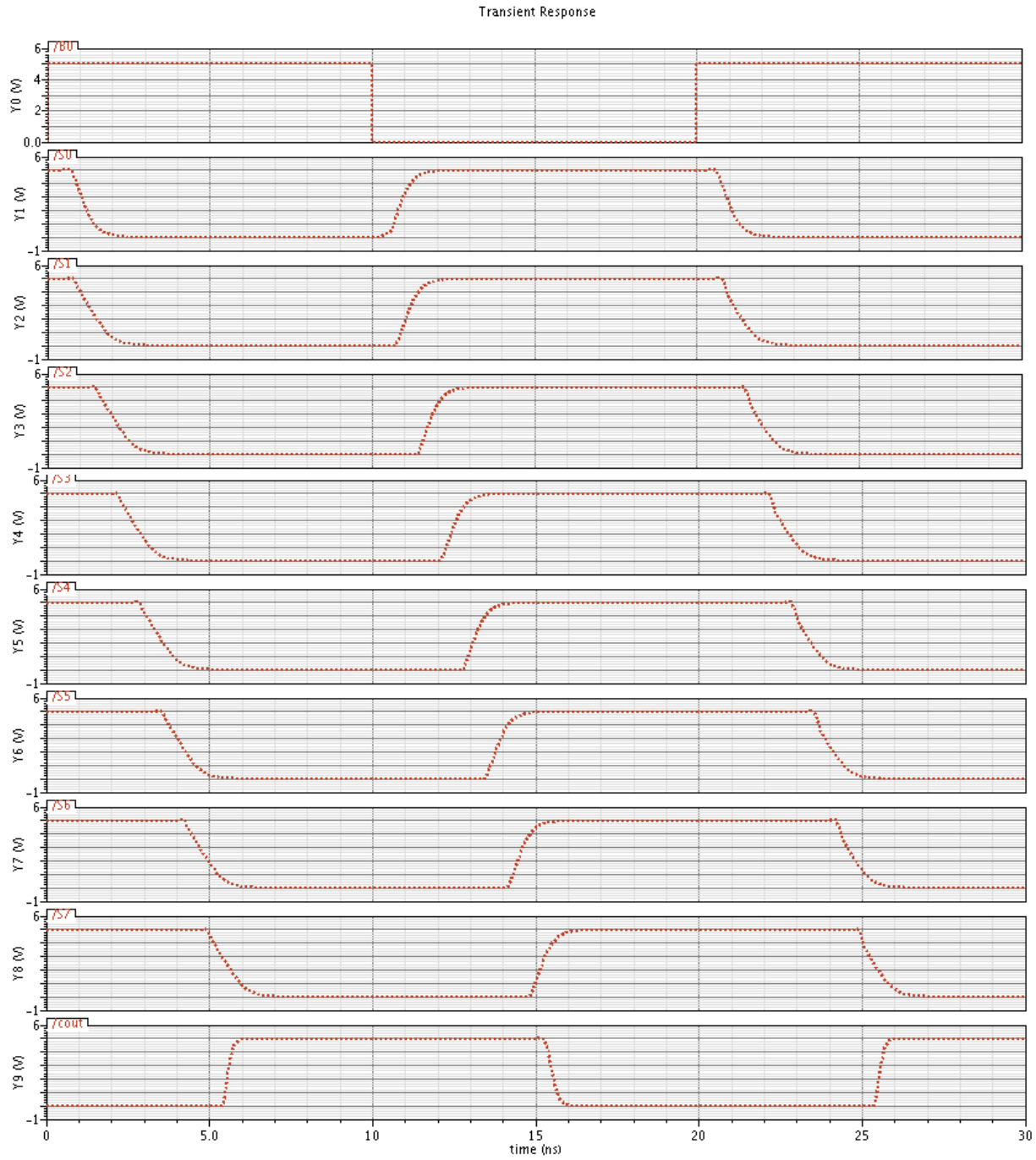


Fig 2.4.5 Simulation of 8-bit full adder

2.4.3. Algorithm block

The schematic and symbol of algorithm block are shown in Fig 2.4.6 and Fig 2.4.7.

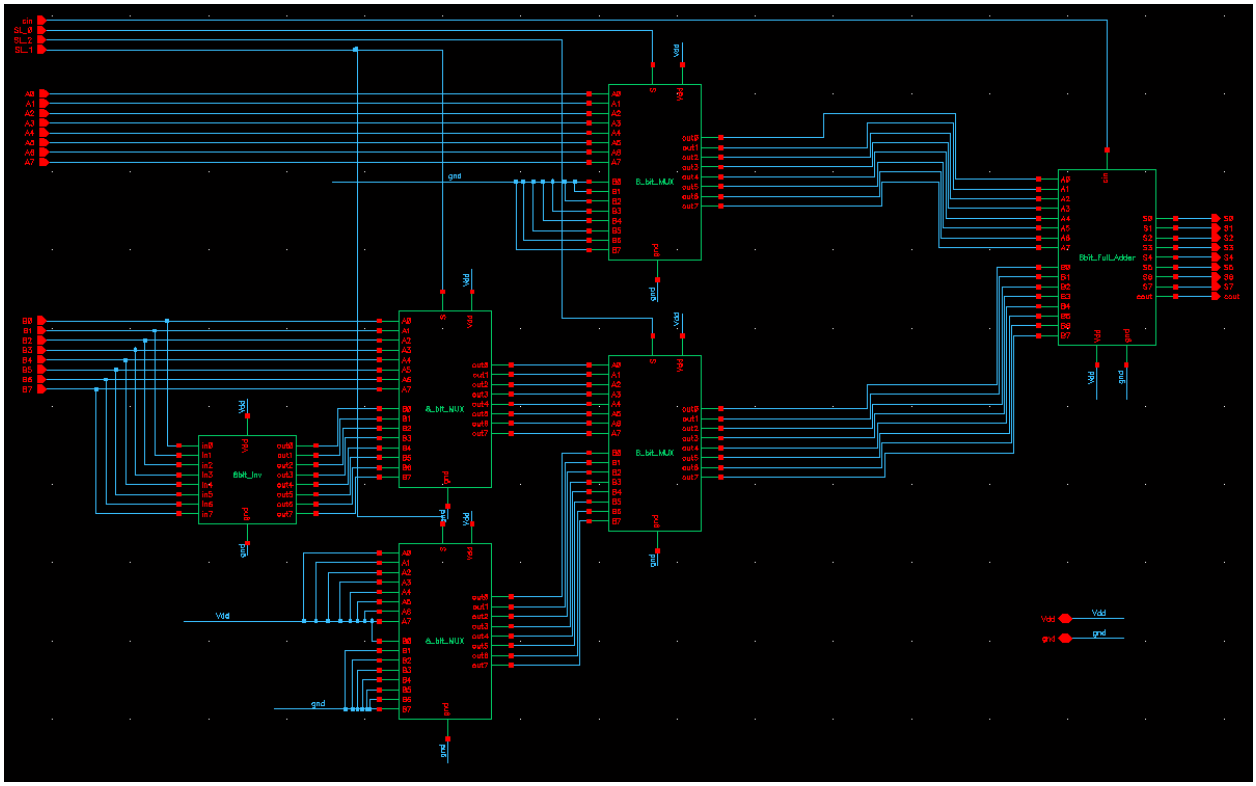


Fig 2.4.6 Schematic of algorithm block

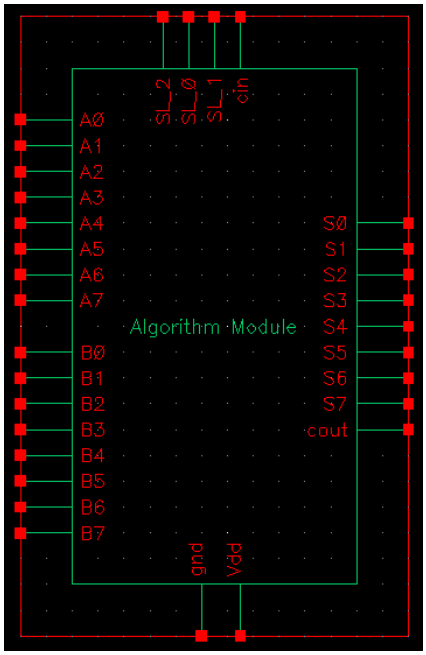


Fig 2.4.7 Symbol of algorithm block

Fig 2.4.8 gives the test bench of the algorithm block. In order to make the output synchronous, one 8-bit dynamic DFF and one 1-bit dynamic DFF is used in the test bench. Inputs A and B both are 00000001. The simulation results of add/increment/1's complement/2's complement operations are shown in Fig 2.4.9. The simulation results of subtract/decrement operations are shown in Fig 2.4.10. The signals and outputs of simulation of add/increment/1's complement/2's complement operations are listed in Table 2.4.1. The signals and outputs of simulation of subtract/decrement operations are listed in Table 2.4.2.

Table 2.4.1 Signals/outputs of add/increment/1's complement/2's complement operations

S2	S1	S0	cin	output function	output
0	0	0	0	add	00000010
1	1	0	0	increment	00000010
0	1	1	0	1's complement	11111110
0	1	1	1	2's complement	11111111

Table 2.4.2 Signals/outputs of subtract/decrement operations

S2	S1	S0	cin	output function	output
0	1	0	1	subtract	00000000
1	0	0	0	decrement	00000000

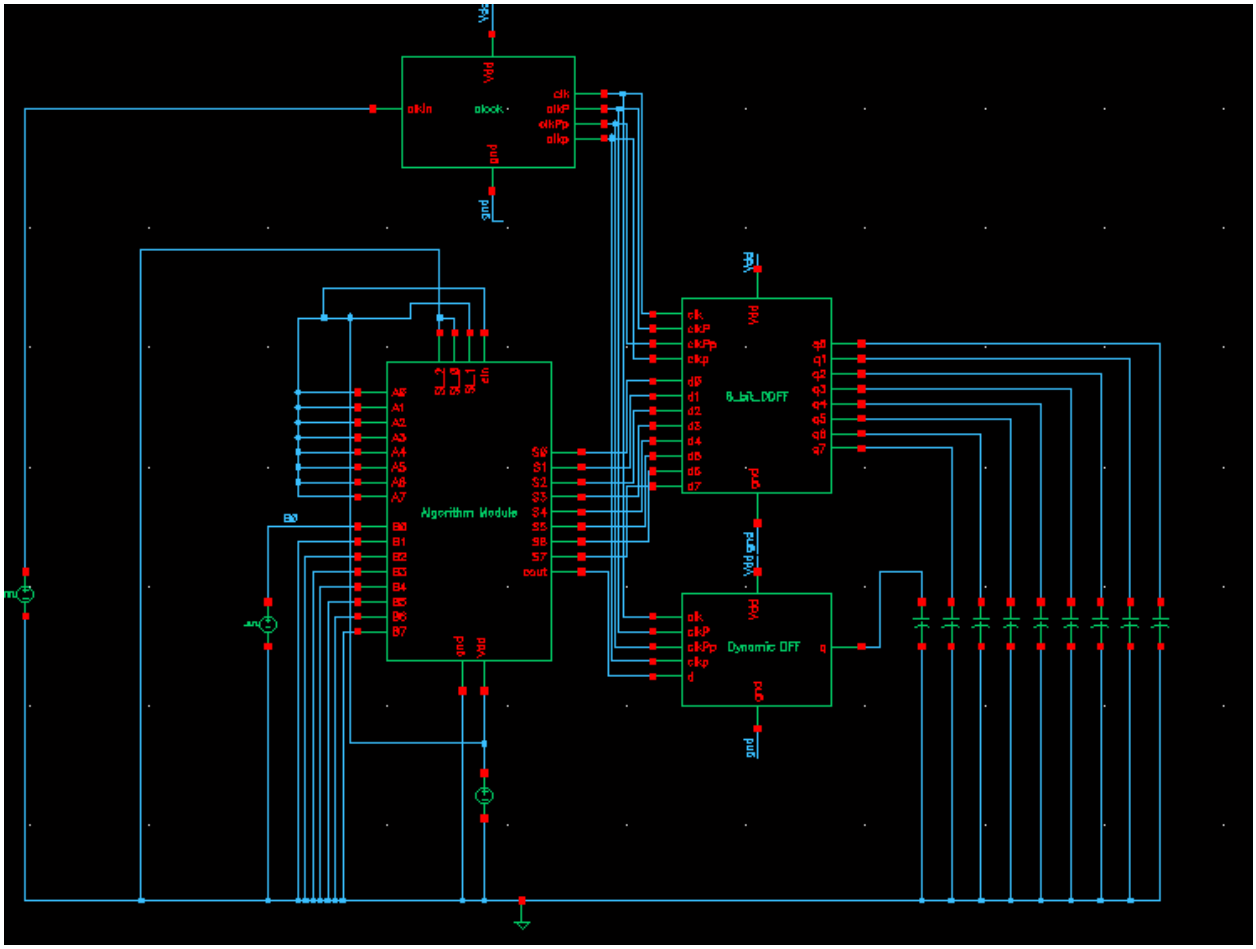


Fig 2.4.8 Test bench of algorithm block

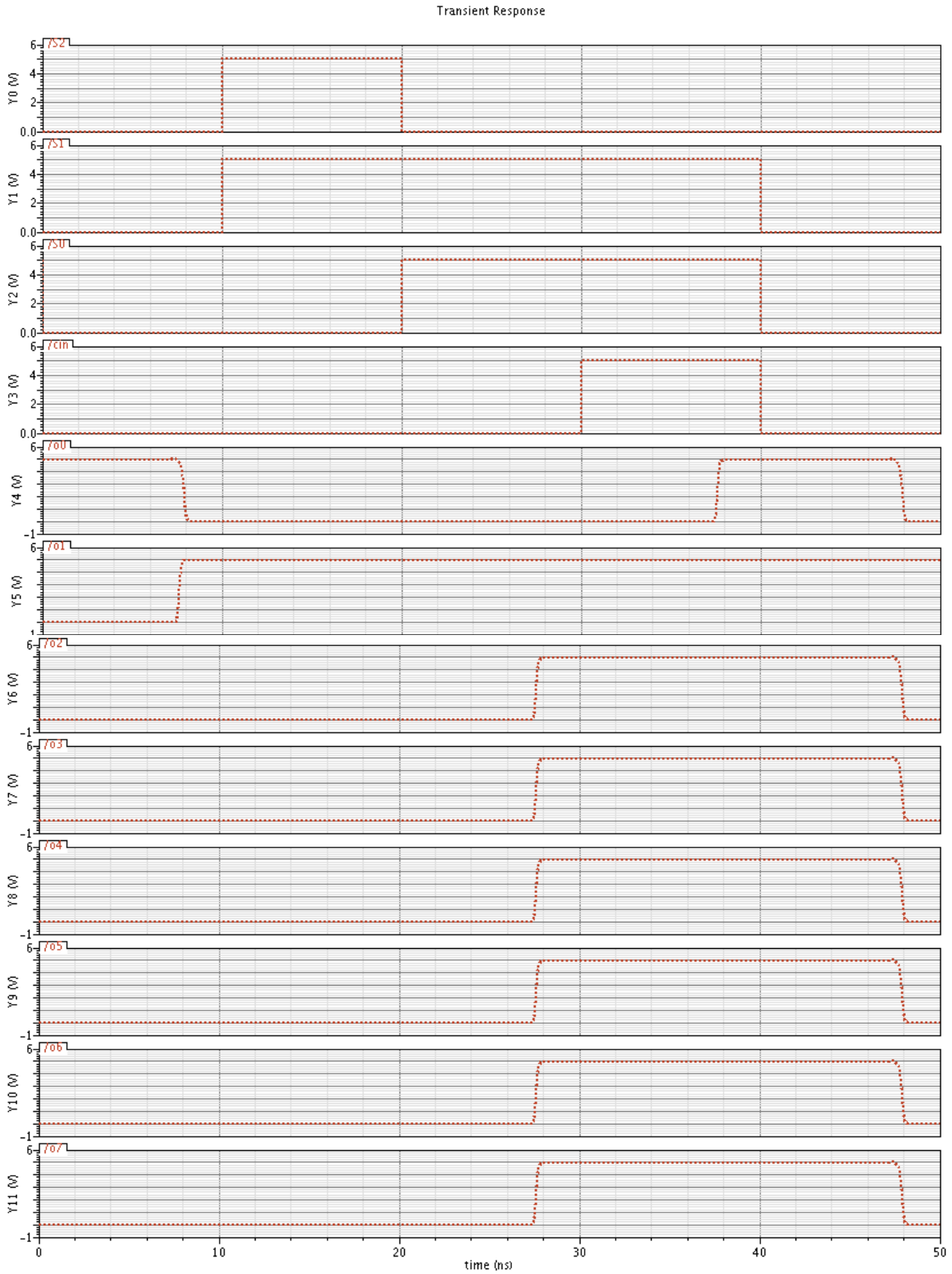


Fig 2.4.9 Simulation of algorithm block: add/increment/1's complement/2's complement

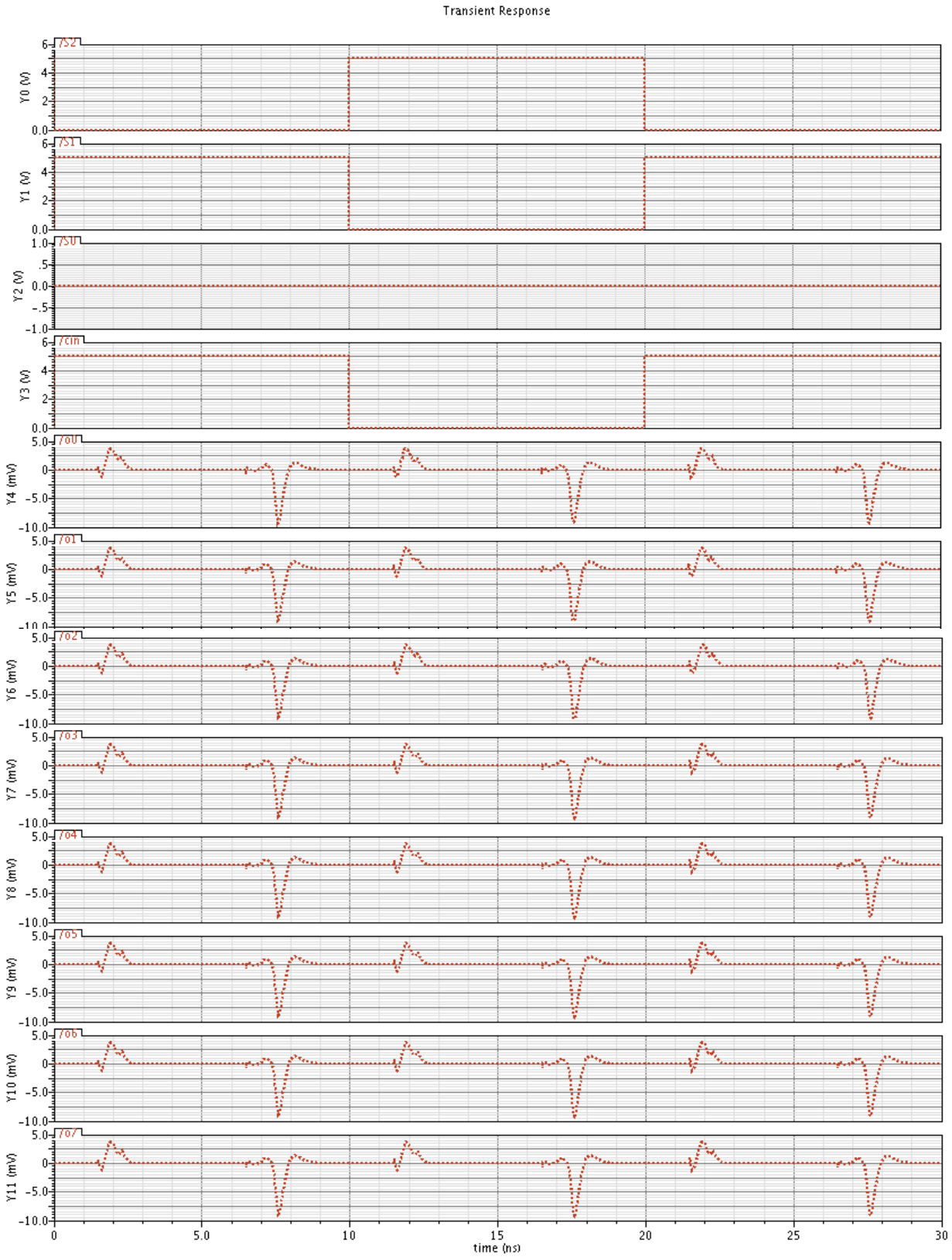


Fig 2.4.10 Simulation of algorithm block: subtract/decrement

2.4.4. Comparator block

The schematic and symbol of comparator block are shown in Fig 2.4.11 and Fig 2.4.12.

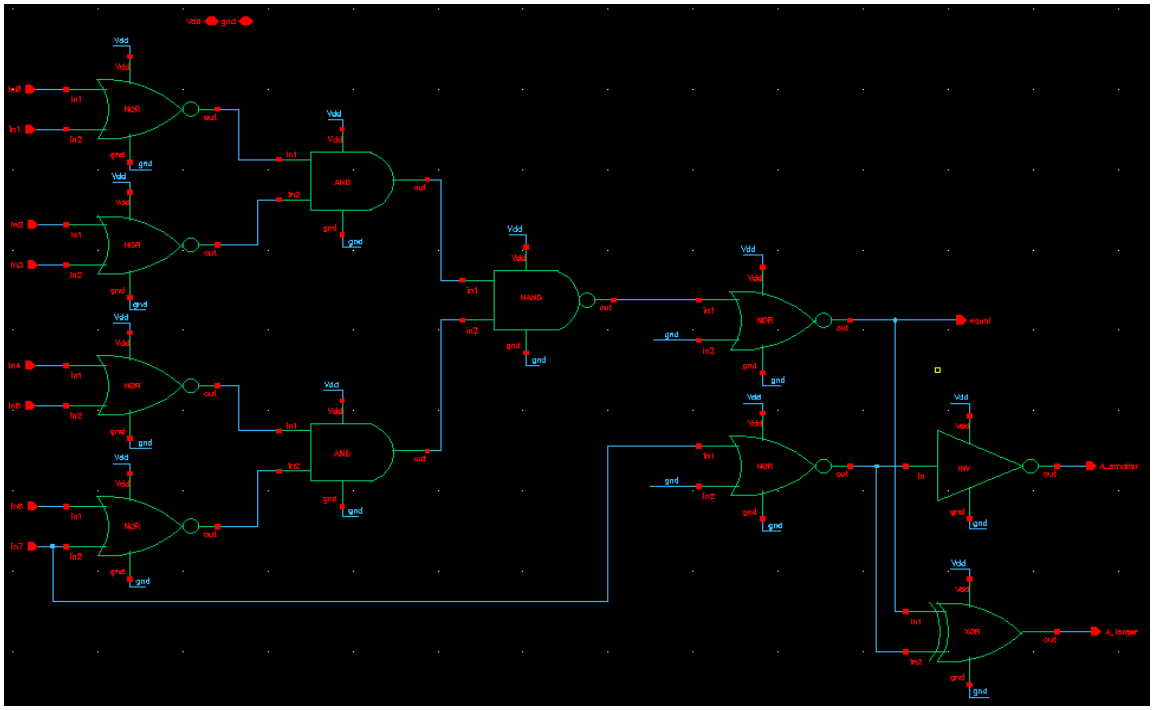


Fig 2.4.11 Schematic of comparator block

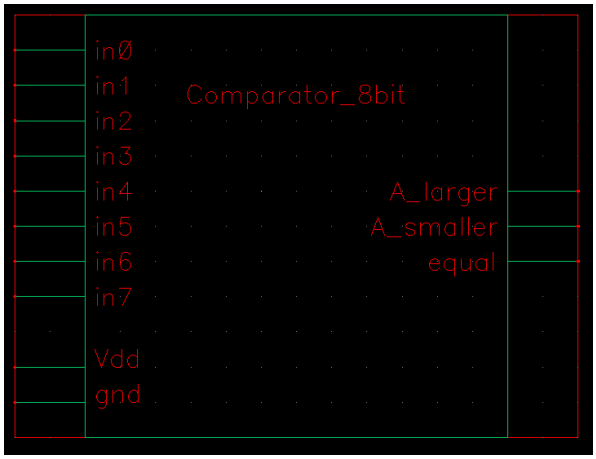


Fig 2.4.12 Symbol of comparator block

Fig 2.4.13 gives the test bench of the comparator block, which is linked after the algorithm block. In order to make the output synchronous, three 1-bit dynamic DFF is used in the test bench. The simulation result is shown in Fig 2.4.14. Inputs A and B are varying at different frequency.

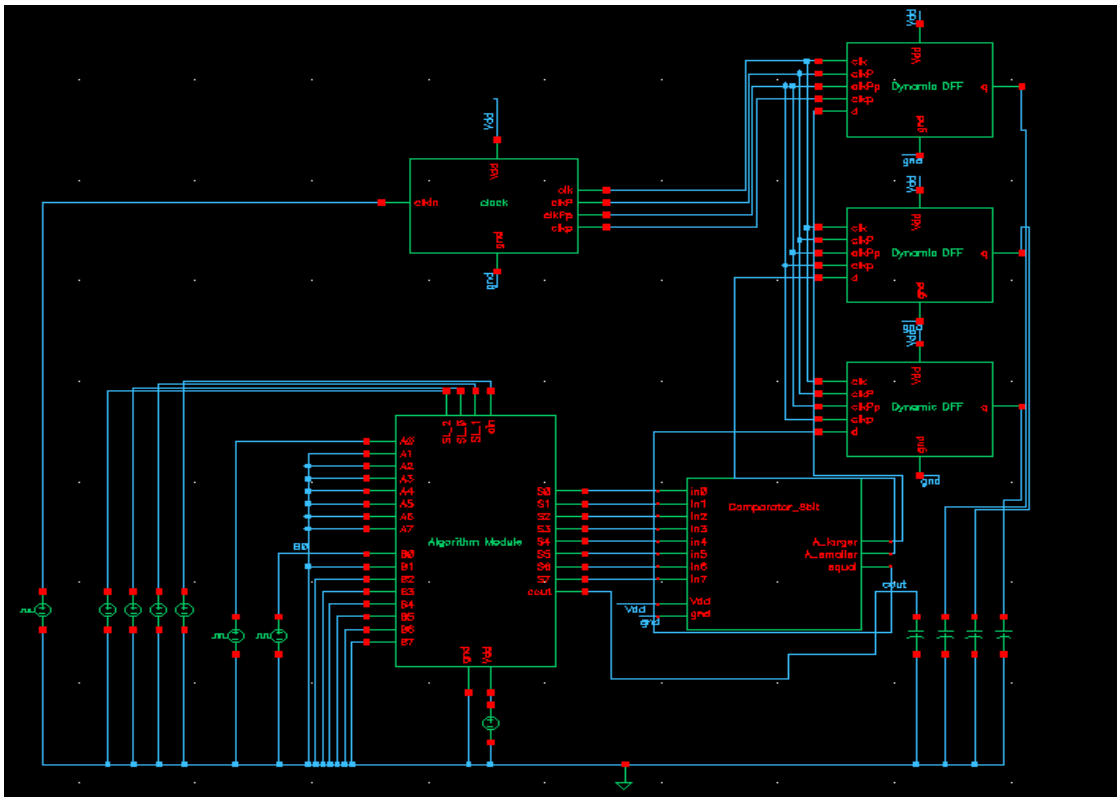


Fig 2.4.13 Test bench of comparator block

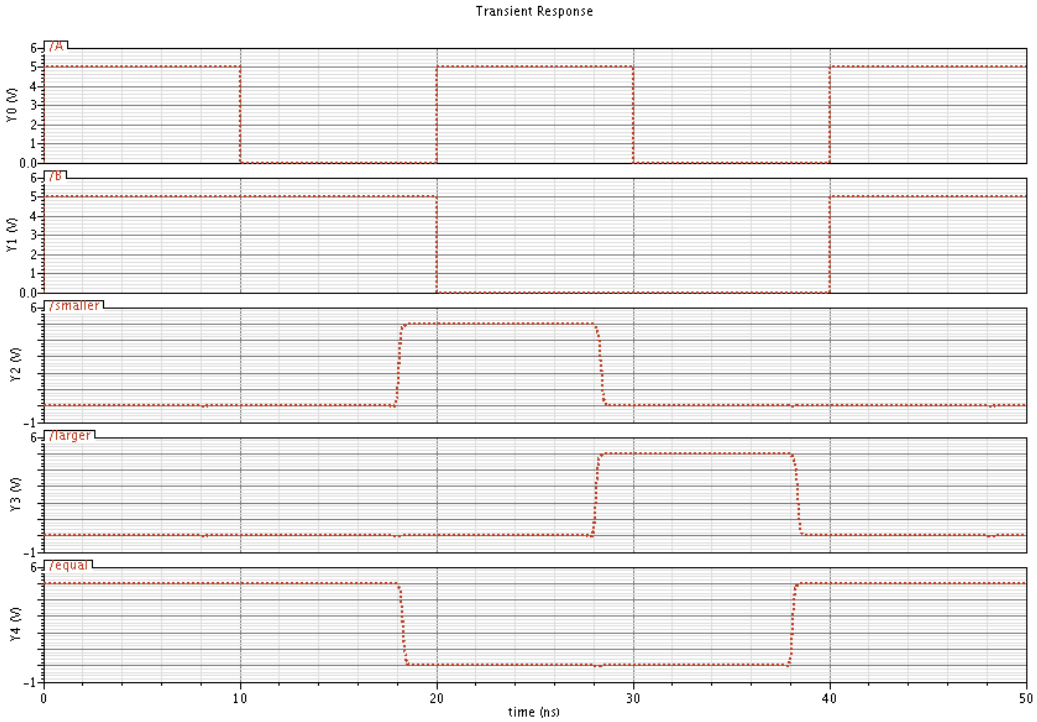


Fig 2.4.14 Simulation of comparator block

2.4.5. 1-bit Logic block

The schematic and symbol of 1-bit logic block are shown in Fig 2.4.15 and Fig 2.4.16.

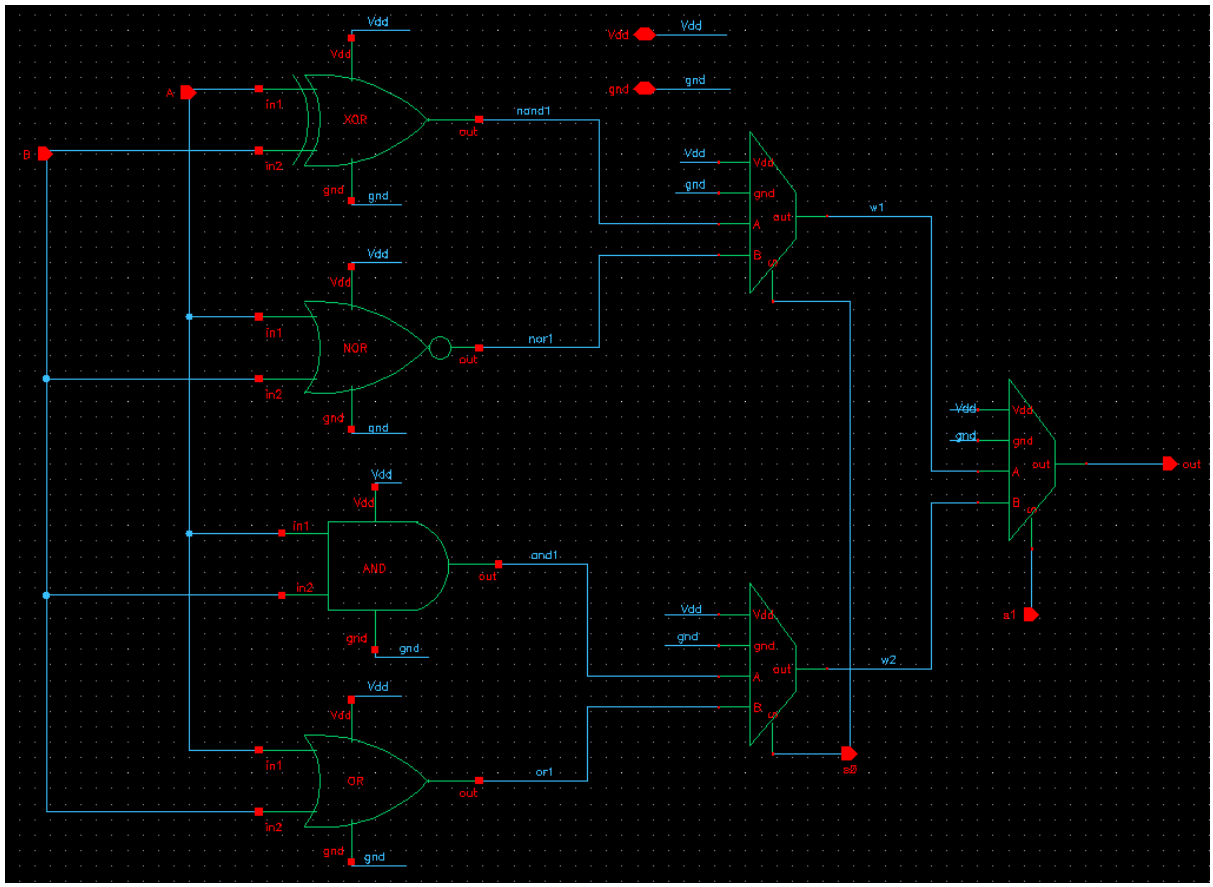


Fig 2.4.15 Schematic of 1-bit logic block

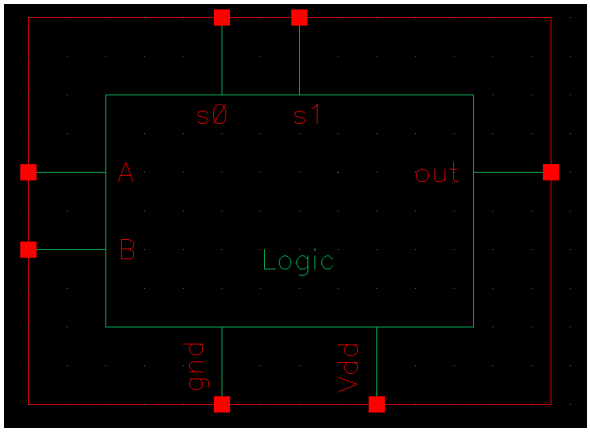


Fig 2.4.16 Symbol of 1-bit logic block

2.4.6. 8-bit Logic block

The schematic and symbol of 8-bit logic block are shown in Fig 2.4.17.

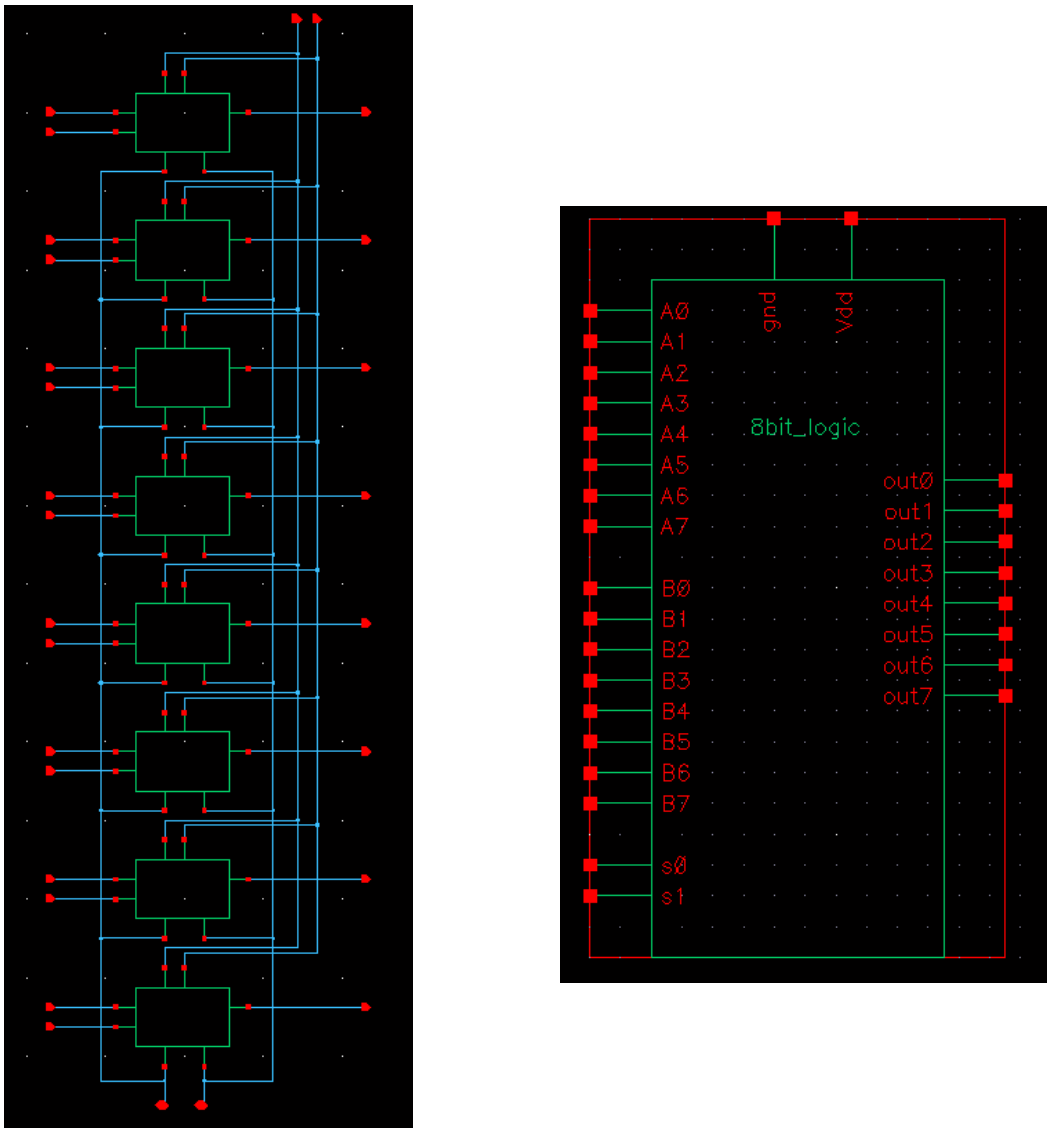


Fig 2.4.17 Schematic and symbol of 8-bit logic block

The test bench is shown in Fig 2.4.18 and the simulation results in Fig 2.4.19. In order to make the output synchronous, one 8-bit dynamic DFF is used in the test bench. The inputs here are A = 11110000, B = 11111111. The signals and outputs of the simulation are listed in Table 2.4.3.

Table 2.4.3 Signals/outputs of logic operations

S1	S0	Logic	Output
0	0	XOR	00001111
0	1	NOR	00000000
1	0	AND	11110000
1	1	OR	11111111

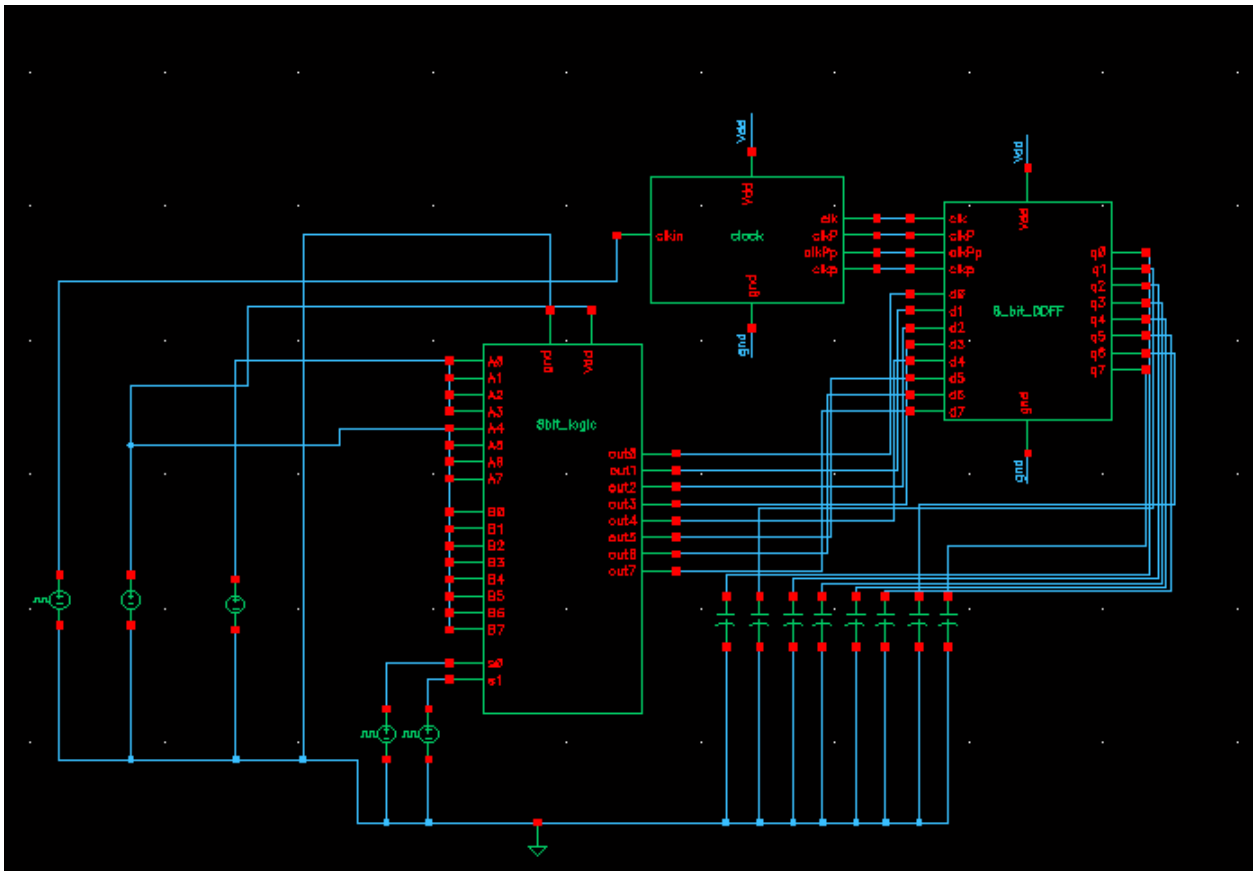


Fig 2.4.18 Test bench of 8-bit logic block

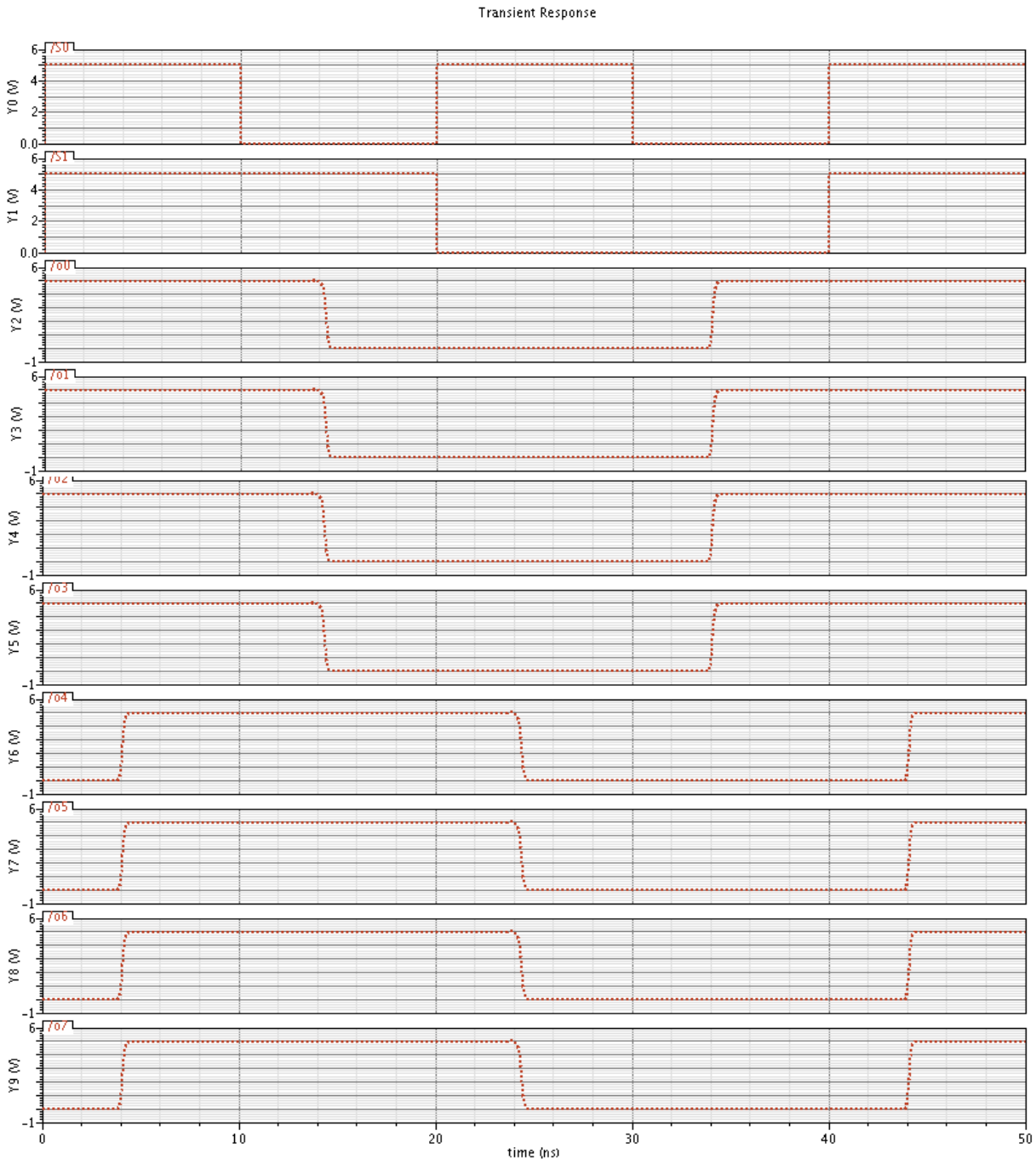


Fig 2.4.19 Simulation of 8-bit logic block

2.4.7. Shift/rotate block

The schematic and symbol of shift/rotate block are shown in Fig 2.4.20 and Fig 2.4.21.

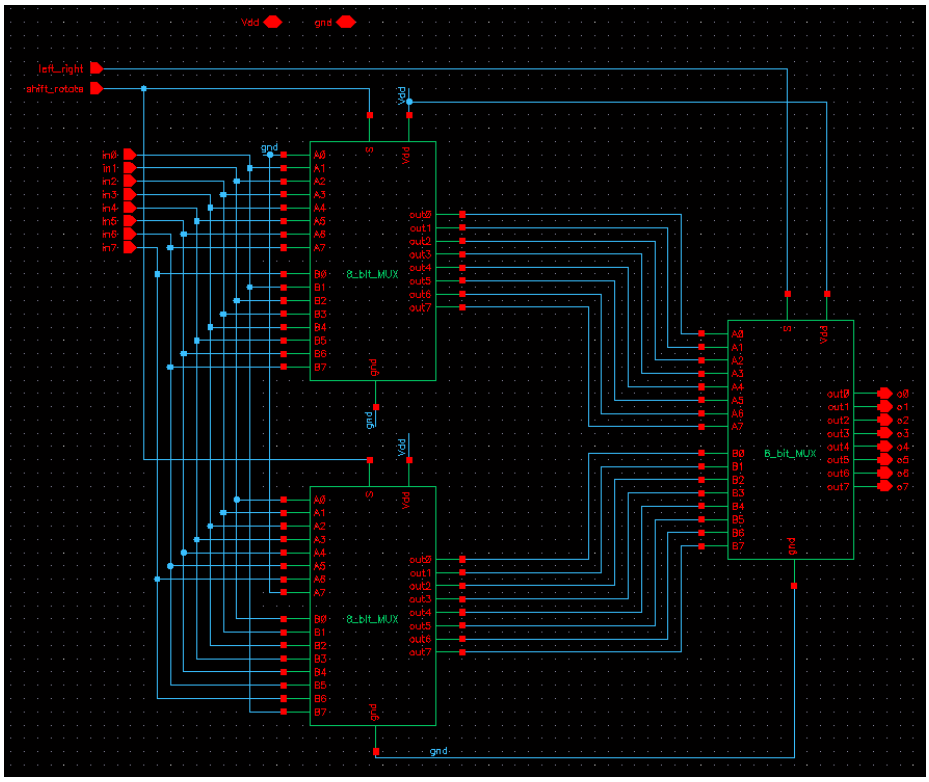


Fig 2.4.20 Schematic of shift/rotate block

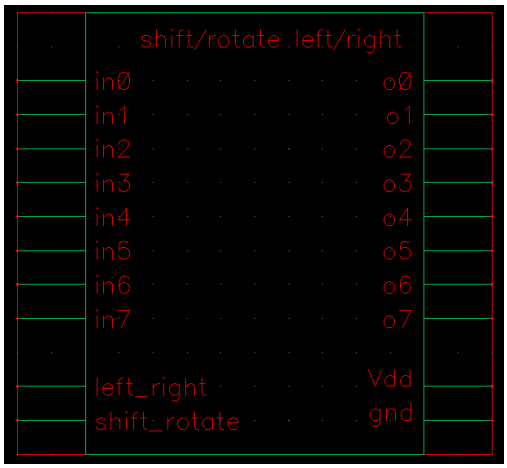


Fig 2.4.21 Symbol of shift/rotate block

The test bench is shown in Fig 2.4.22 and the simulation results in Fig 2.4.23. In order to make the output synchronous, one 8-bit dynamic DFF is used in the test bench. The input A = 10000111. The signals and outputs of the simulation are listed in Table 2.4.4.

Table 2.4.4 Signals/outputs of shift/rotate operations

S1	S0	output function	output
0	0	shift left	00001110
0	1	shift right	01000011
1	0	rotate left	00001111
1	1	rotate right	11000011

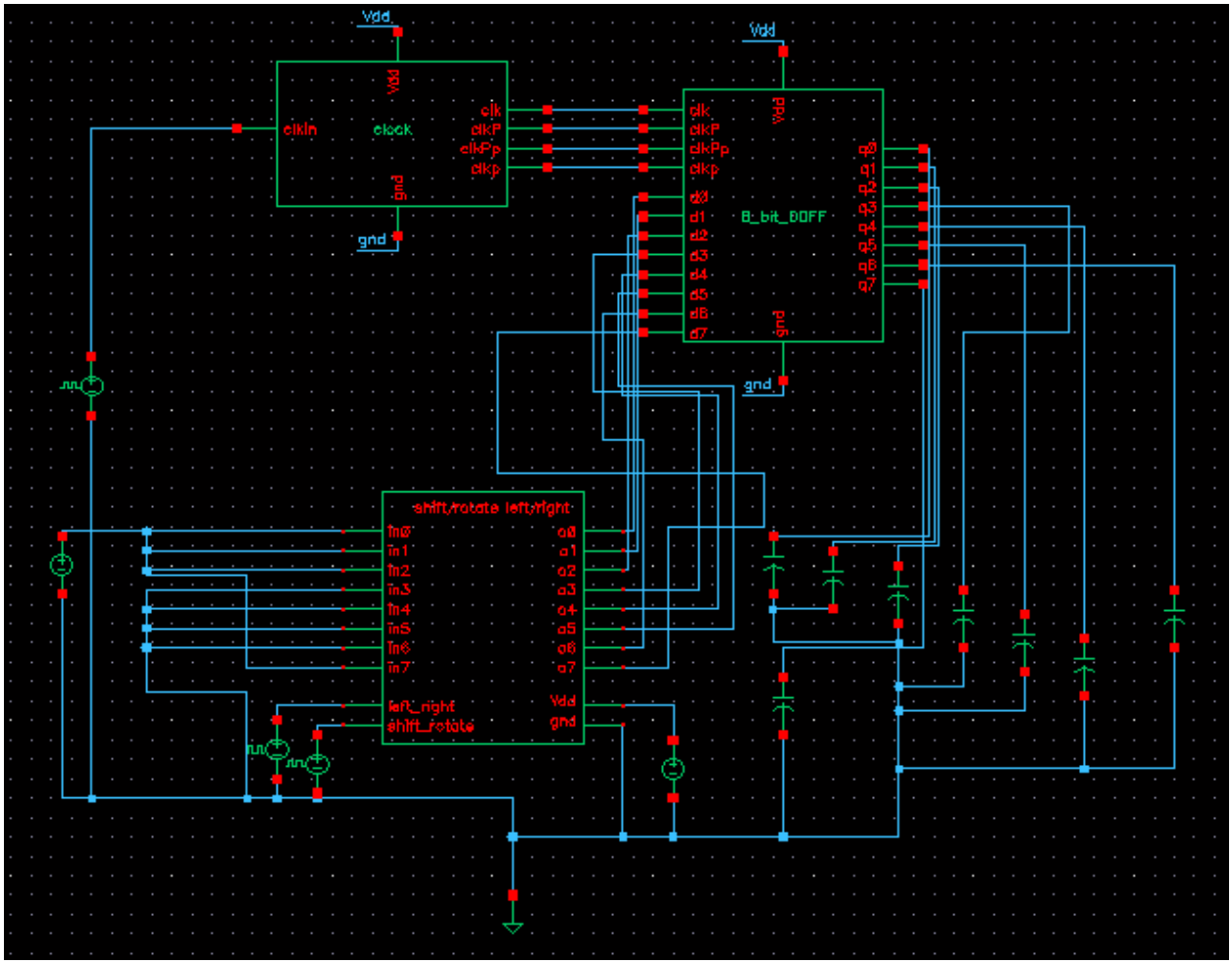


Fig 2.4.22 Test bench of shift/rotate block

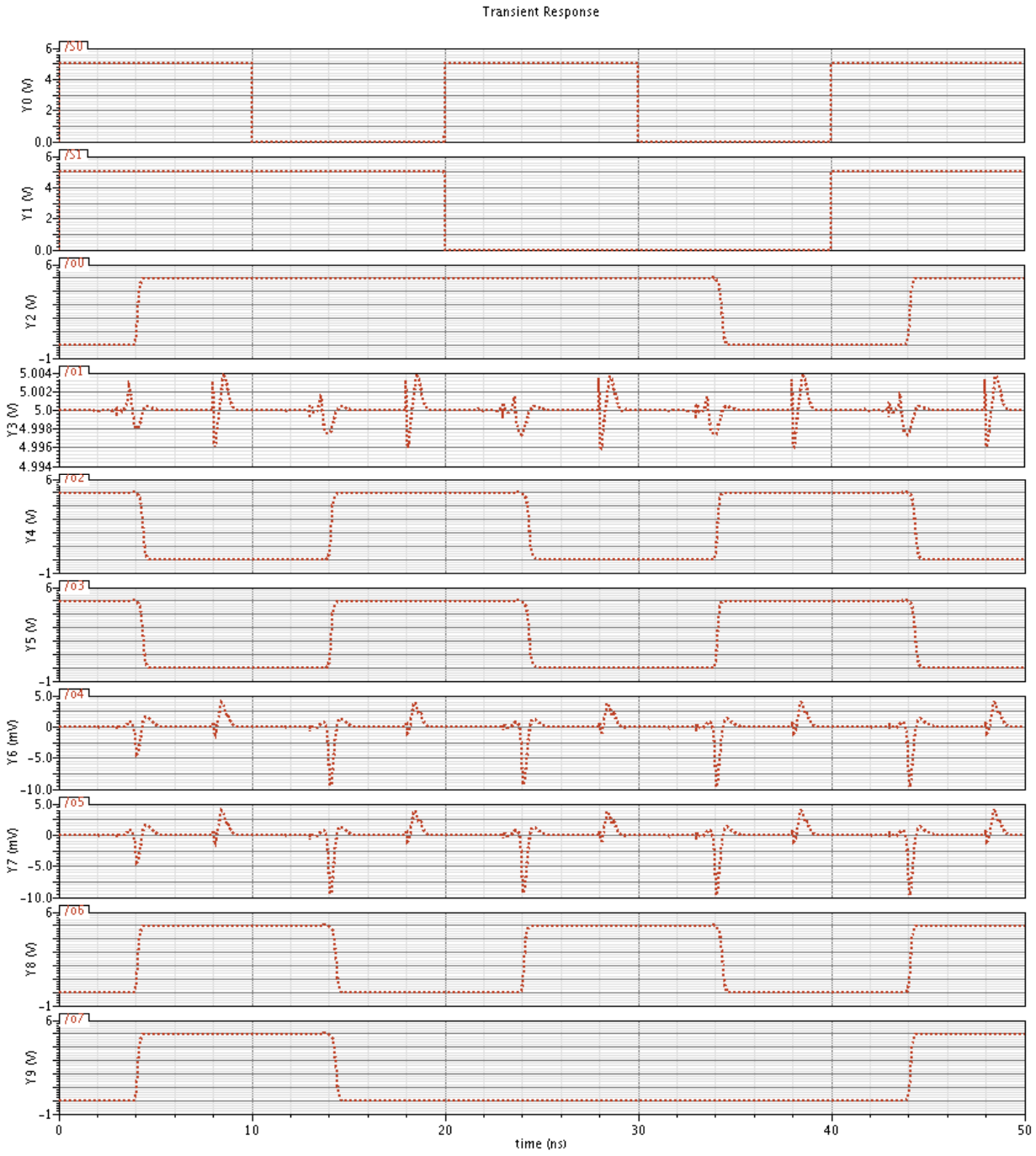


Fig 2.4.23 Simulation of shift/rotate block

3. Layout

3.1. Layout of all blocks and full chip

Fig 3.1.1 gives the layout of ALU and Fig 3.1.2 is the design rule check. The whole chip area is 330×352 um^2.

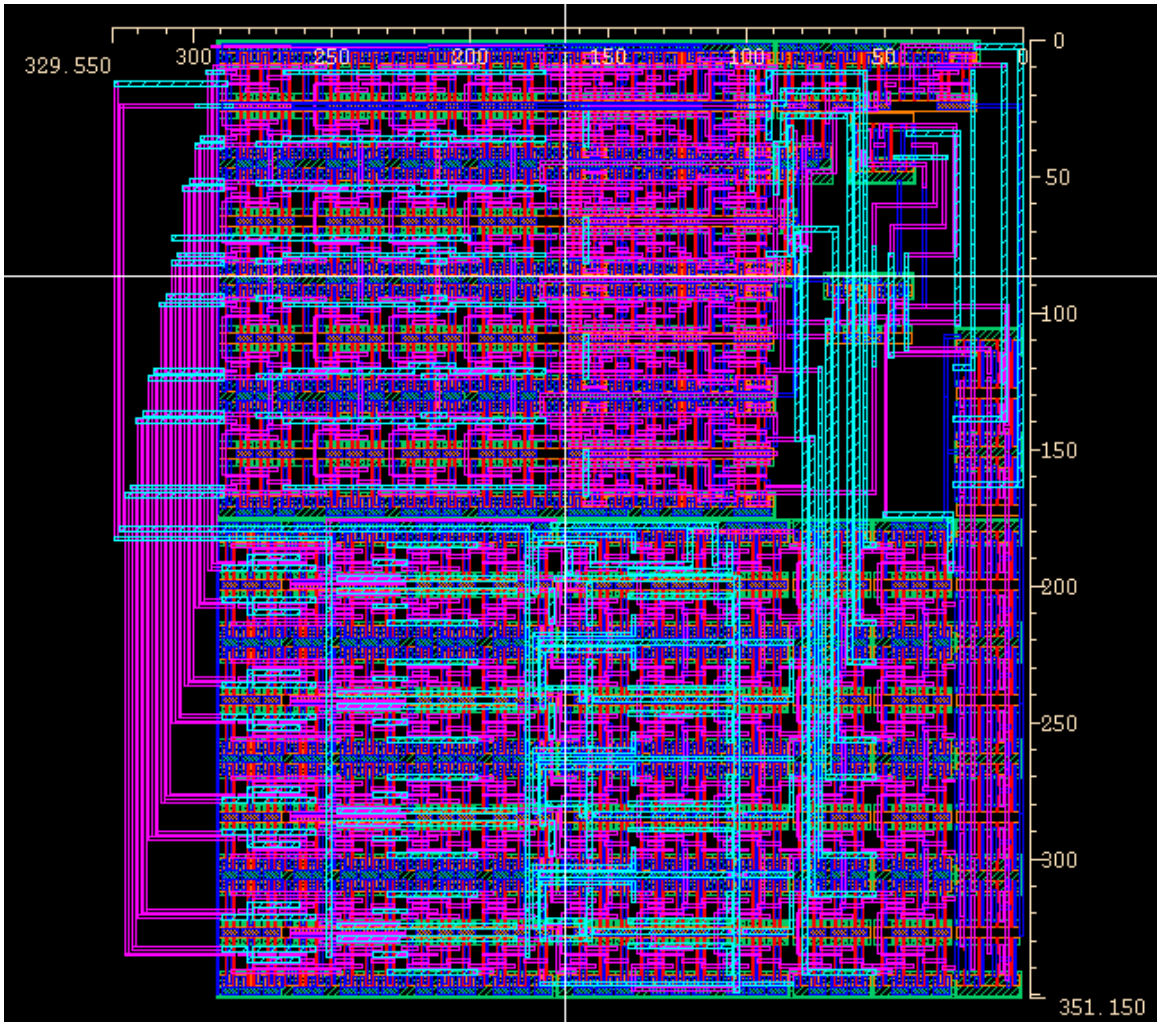


Fig 3.1.1 Layout of ALU

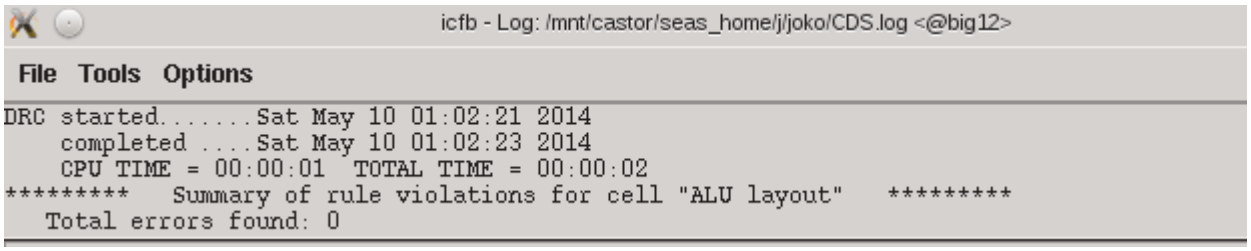


Fig 3.1.2 DRC of ALU

Fig 3.1.3 gives the layout of algorithm block and Fig 3.1.4 is the design rule check.

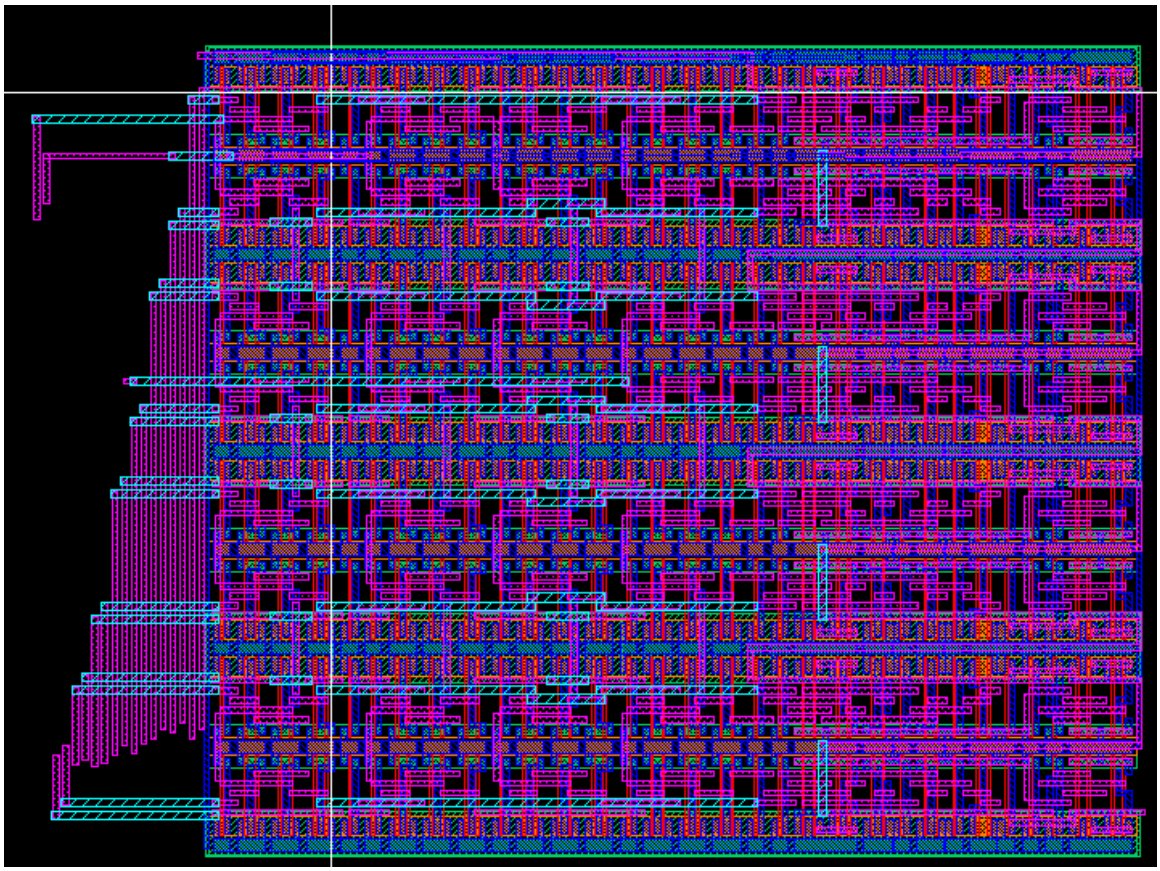


Fig 3.1.3 Layout of algorithm block

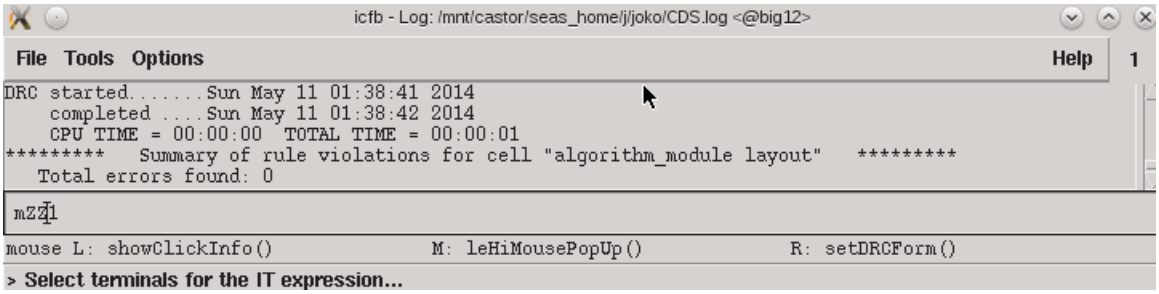


Fig 3.1.4 DRC of algorithm block

Fig 3.1.5 gives the layout of 1-bit adder and Fig 3.1.6 is the design rule check.

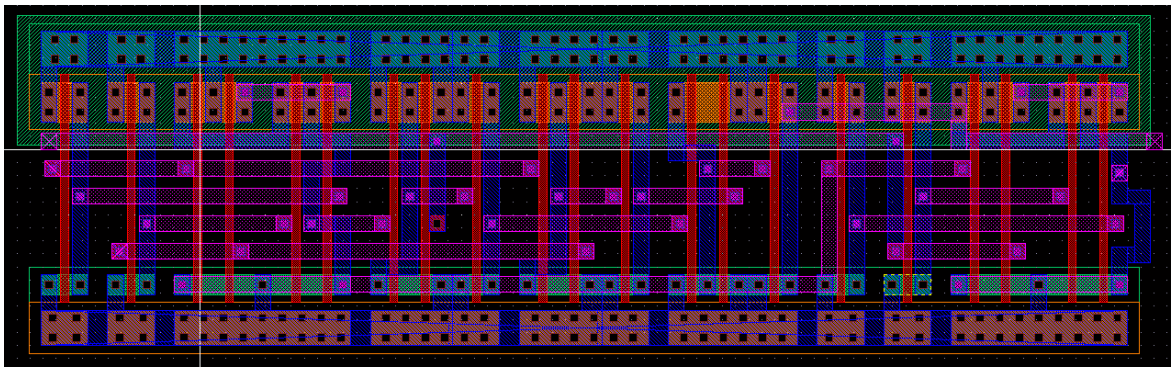


Fig 3.1.5 Layout of 1-bit adder

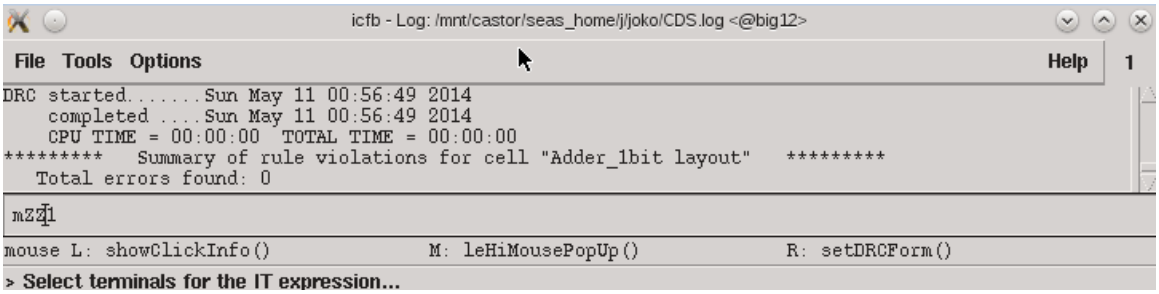


Fig 3.1.6 DRC of 1-bit adder

Fig 3.1.7 gives the layout of 8-bit full adder and Fig 3.1.8 is the design rule check.

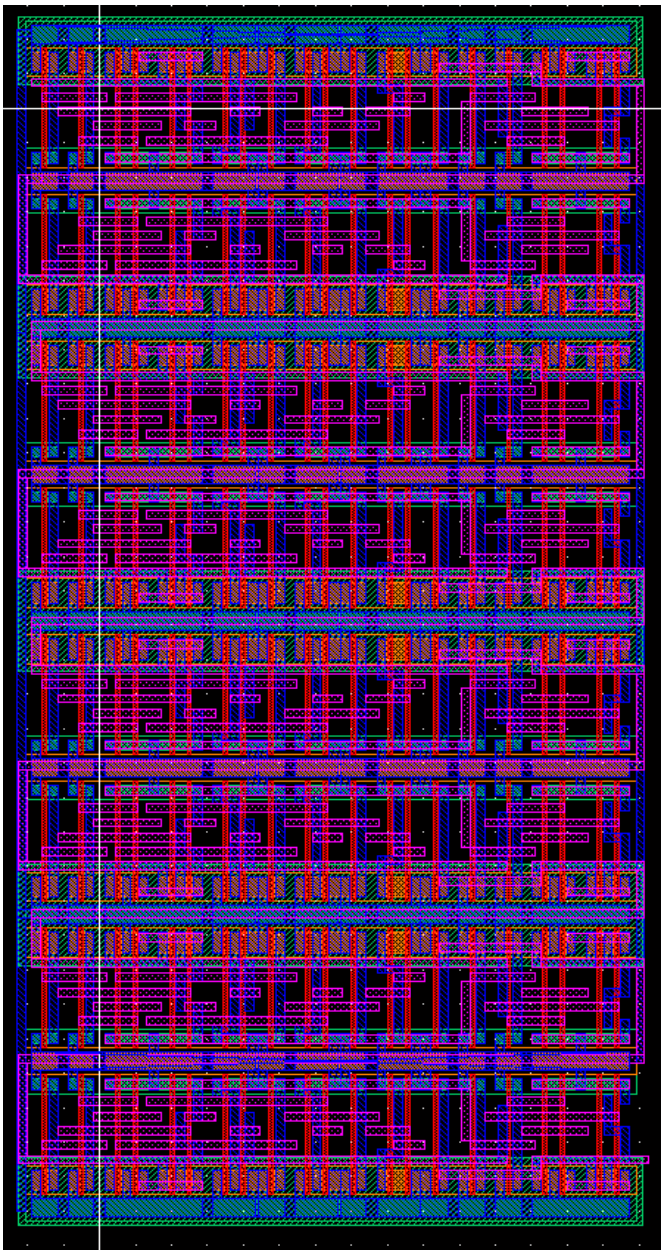


Fig 3.1.7 Schematic and layout of 8-bit full adder

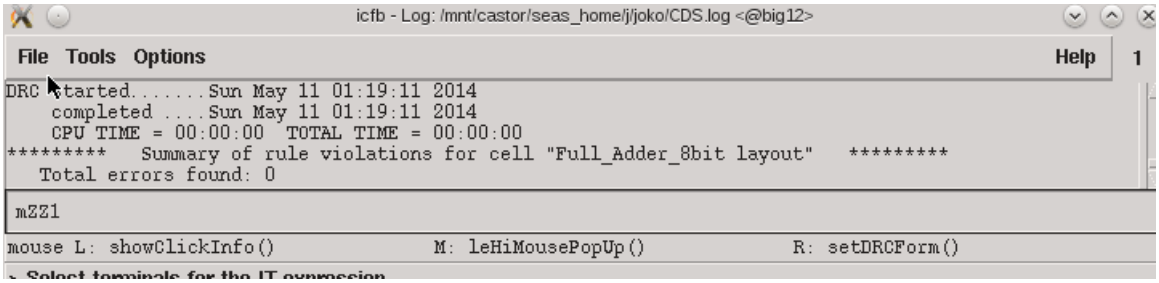


Fig 3.1.8 DRC of 8-bit full adder

Fig 3.1.9 gives the layout of comparator block and Fig 3.1.10 is the design rule check.

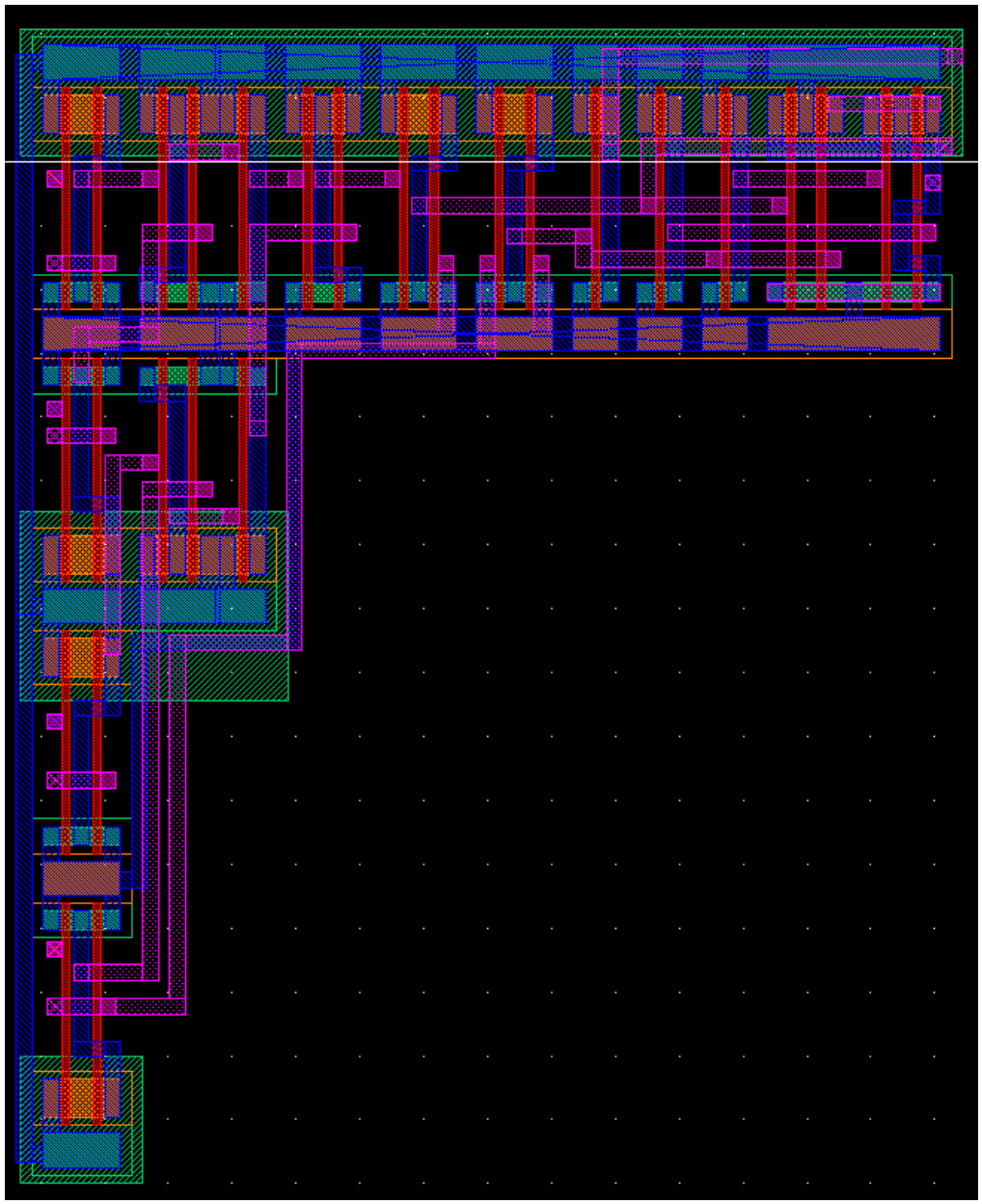


Fig 3.1.9 Layout of comparator block

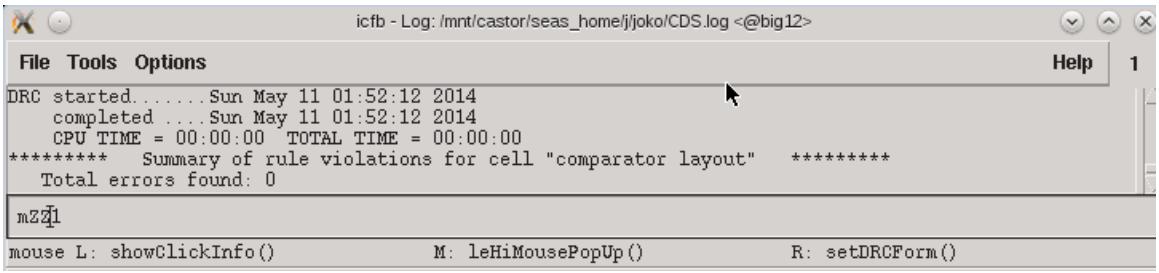


Fig 3.1.10 DRC of comparator block

Fig 3.1.11 gives the layout of 1-bit logic block and Fig 3.1.12 is the design rule check.

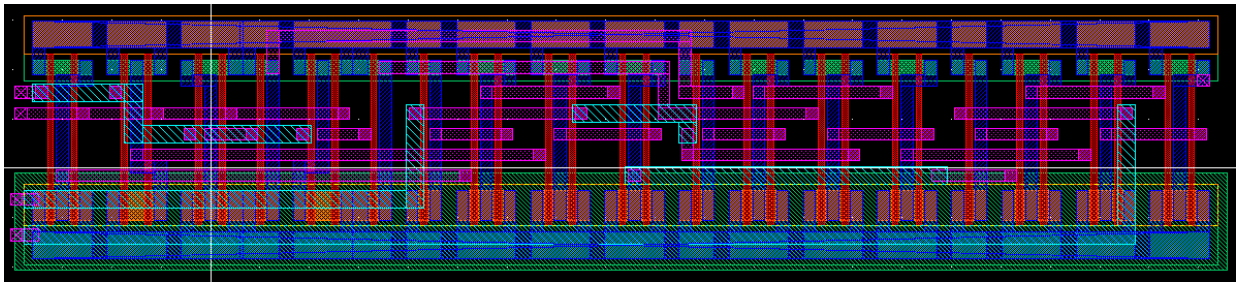


Fig 3.1.11 Layout of 1-bit logic block

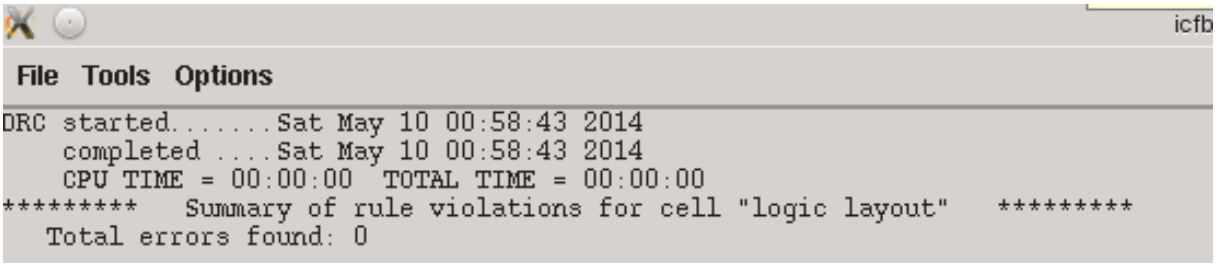


Fig 3.1.12 DRC of 1-bit logic block

Fig 3.1.13 gives the layout of 8-bit logic block and Fig 3.1.14 is the design rule check.

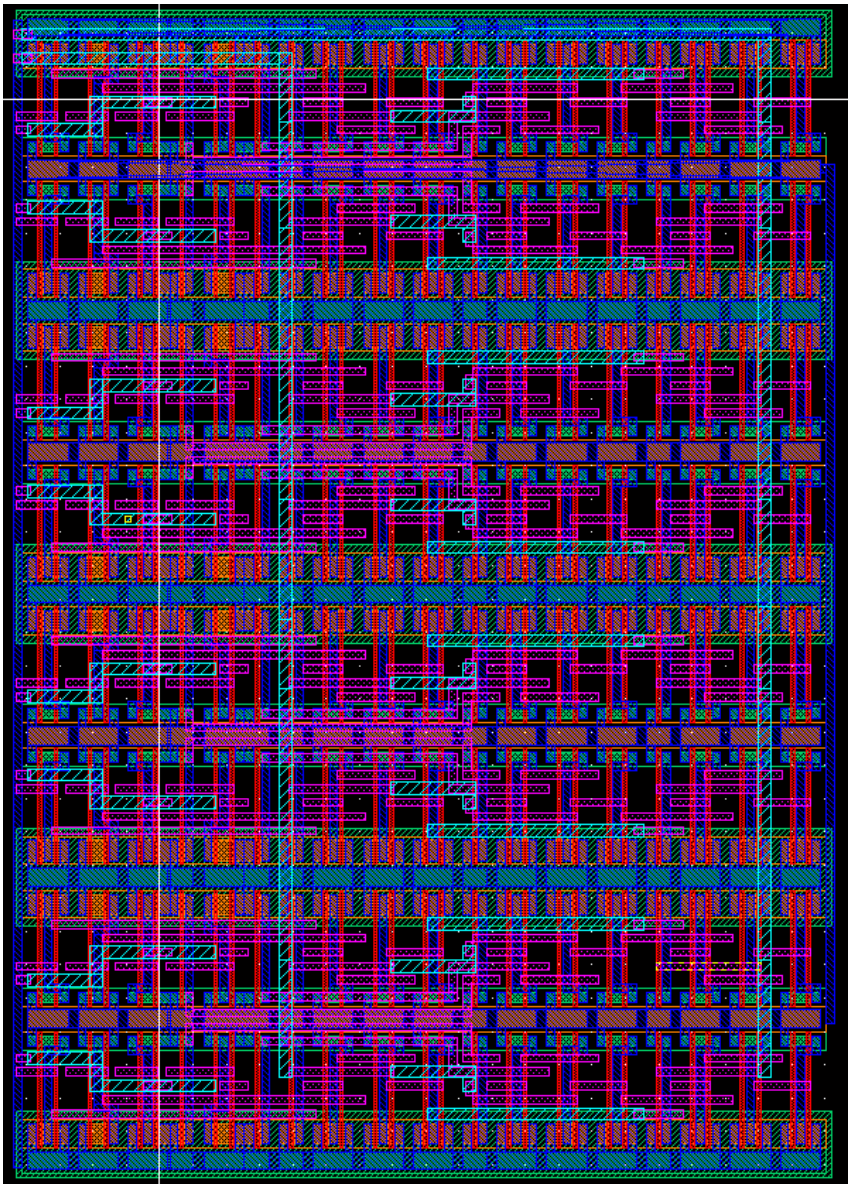


Fig 3.1.13 Layout of 8-bit logic block

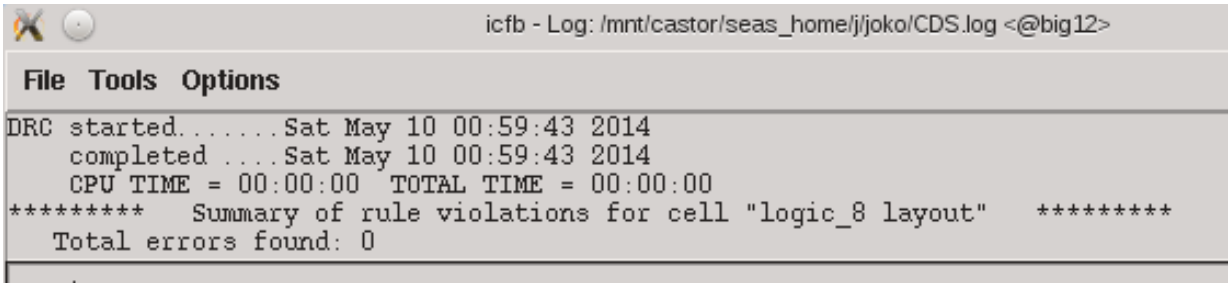


Fig 3.1.14 DRC of 8-bit logic block

Fig 3.1.15 gives the layout of shift/rotate block and Fig 3.1.16 is the design rule check.

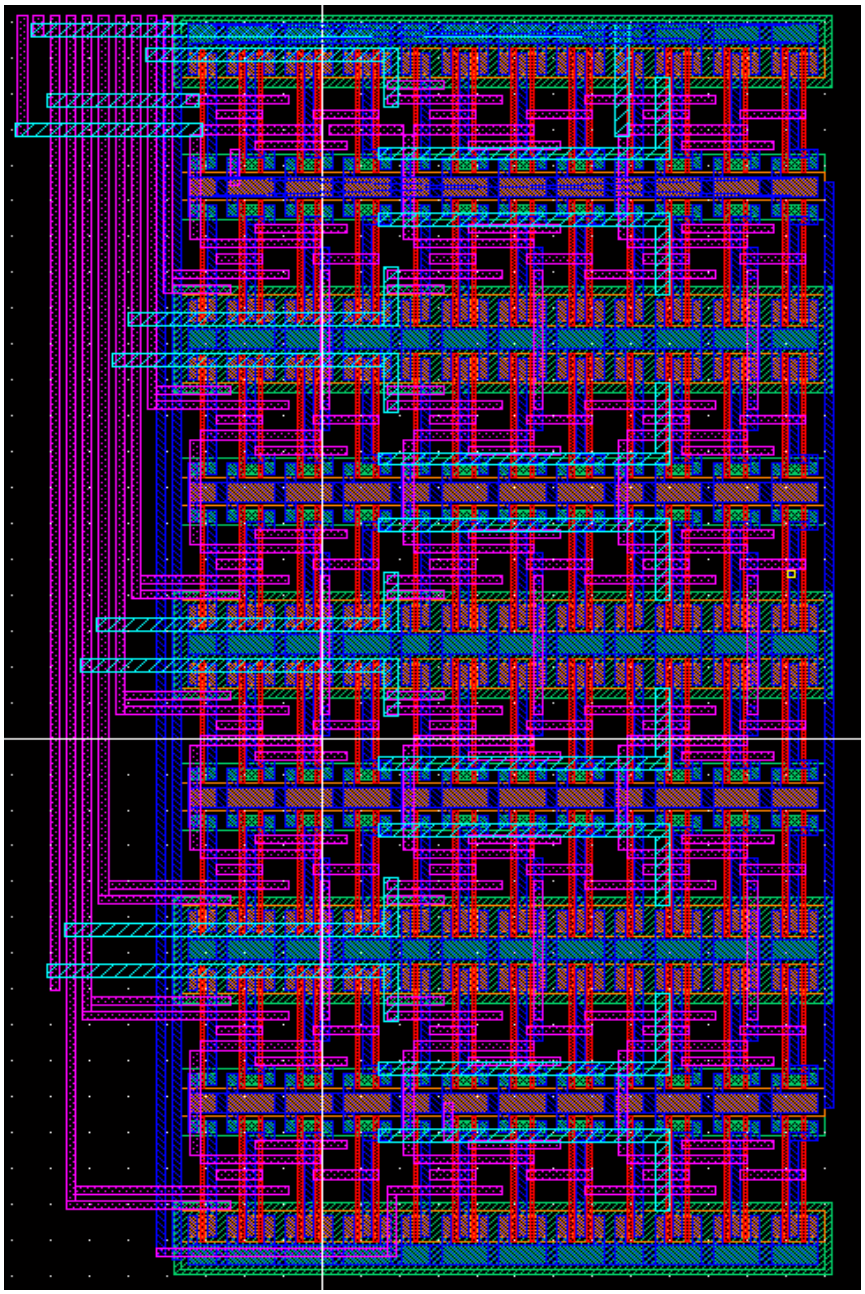


Fig 3.1.15 Layout of shift/rotate block

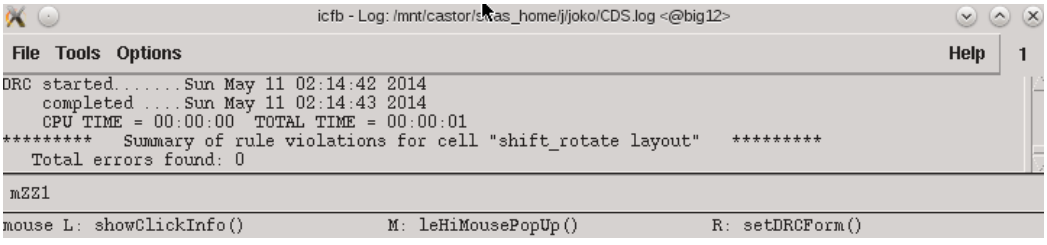


Fig 3.1.16 DRC of shift/rotate block

3.2. Extraction and layout vs. schematic (LVS)

Fig 3.2.1 gives the extracted view of ALU and Fig 3.2.2 is the result of layout vs. schematic. All the blocks and gates have done the LVS check in this project. Here we only include the ALU level LVS, which implies that all the sub-blocks and gates have passed the LVS check.

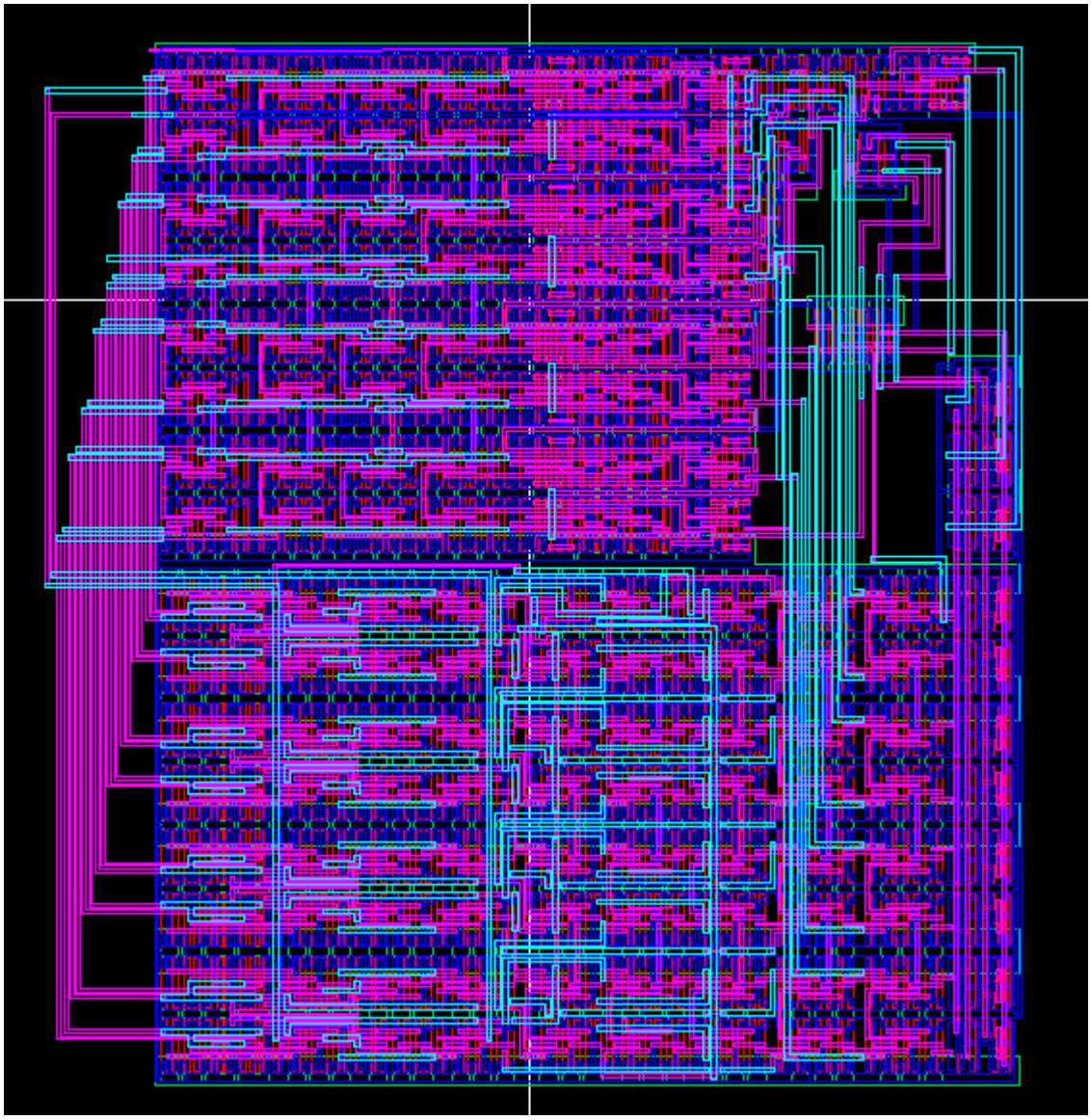


Fig 3.2.1 Extracted view of ALU

```
The net-lists match.

                              layout  schematic
                                 instances
        un-matched               0       0
        rewired                  0       0
        size errors              0       0
        pruned                   0       0
        active                   2052    2052
        total                    2052    2052

                                   nets
        un-matched               0       0
        merged                   0       0
        pruned                   0       0
        active                   1068    1068
        total                    1068    1068

                                 terminals
        un-matched               0       0
        matched but
        different type           0       0
        total                    37      37
```

Fig 3.2.2 Result of layout vs. schematic of ALU

Fig 3.2.3 gives the extracted view of algorithm block.

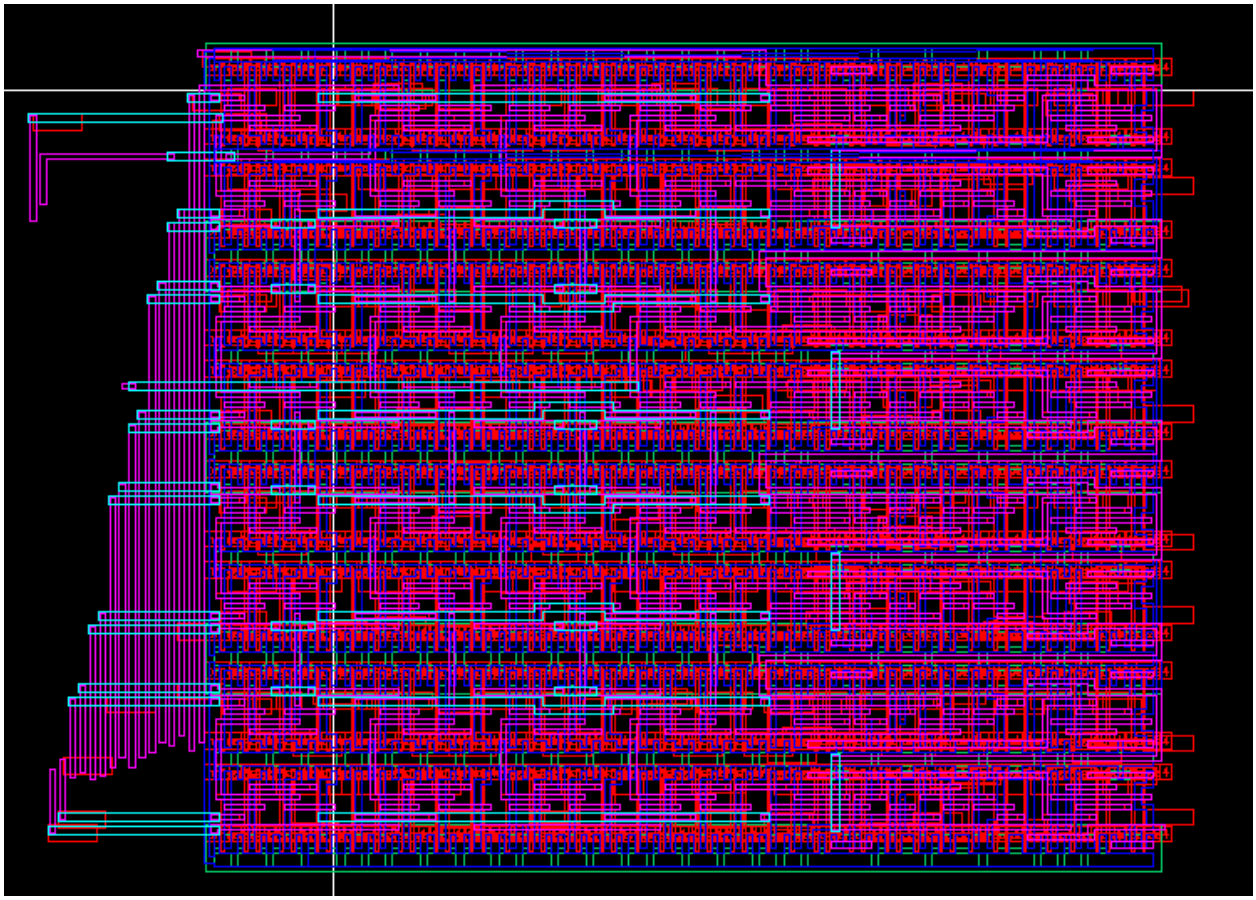


Fig 3.2.3 Extracted view of algorithm block

Fig 3.2.4 gives the extracted view of comparator block.

Fig 3.2.4 Extracted view of comparator block

Fig 3.2.5 gives the extracted view of logic block.

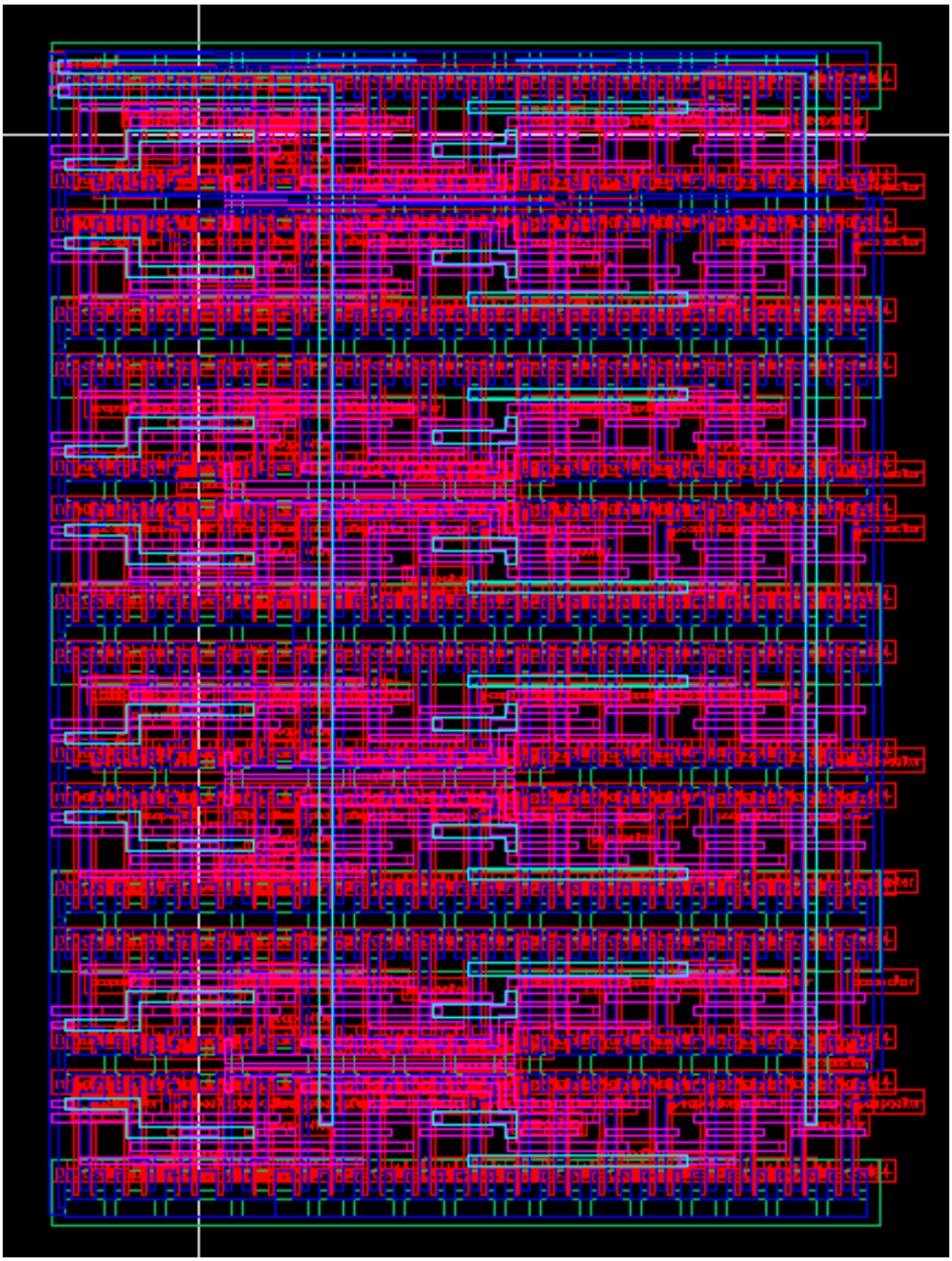


Fig 3.2.5 Extracted view of logic block

Fig 3.2.6 gives the extracted view of shift/rotate block.

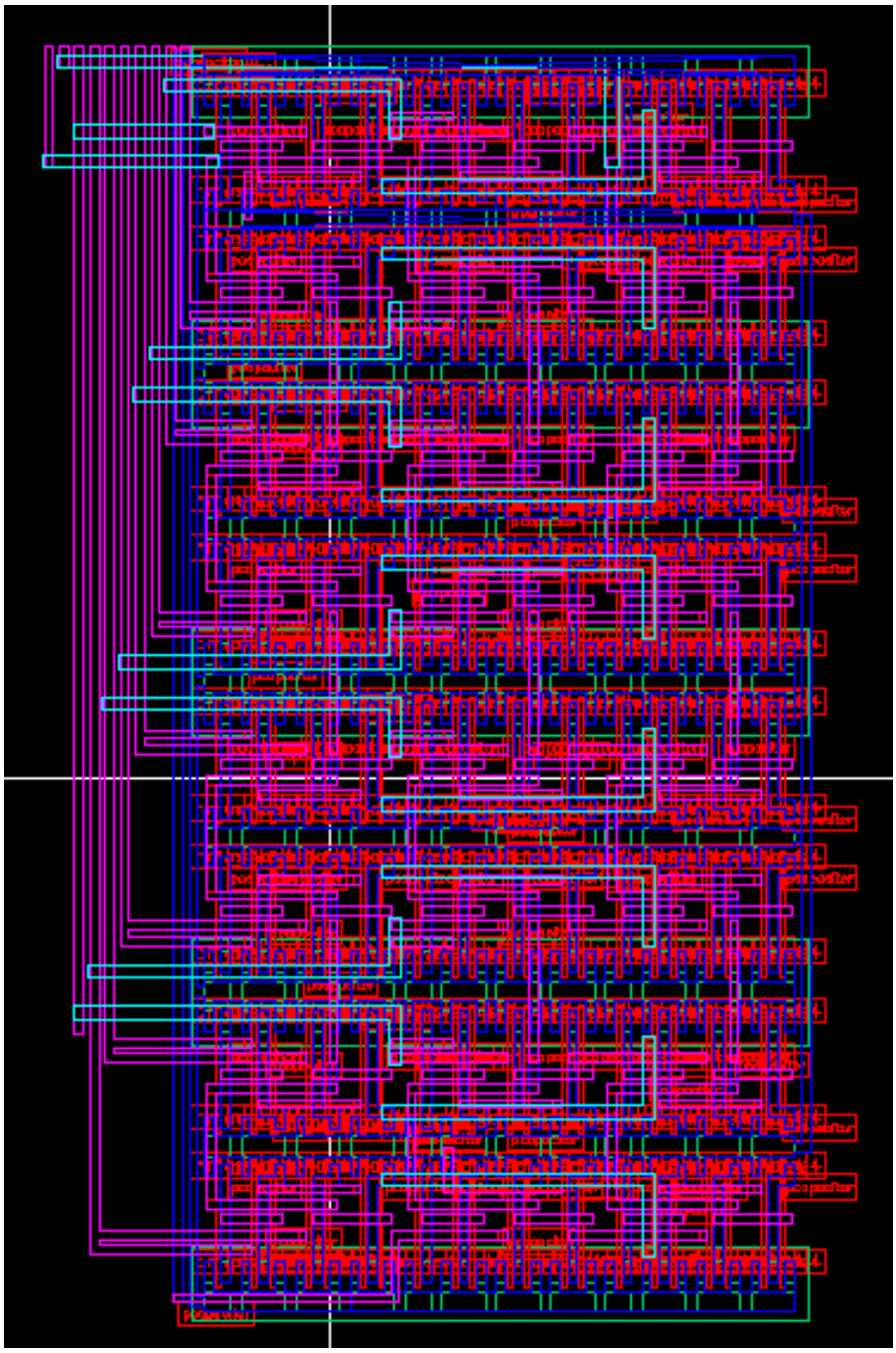


Fig 3.2.6 Extracted view of shift/rotate block

4. Post Layout Simulation

The test bench for the ALU chip is shown in Fig 4.1 with a clock signal cycle of 10 ns. As stated before, DFF gates are used before the final output. We did several tests, all ALU chip level operations have almost the same delay time of around 8 ns which means our ALU can perfectly run at 100MHz. Representative tests are showed in the report.

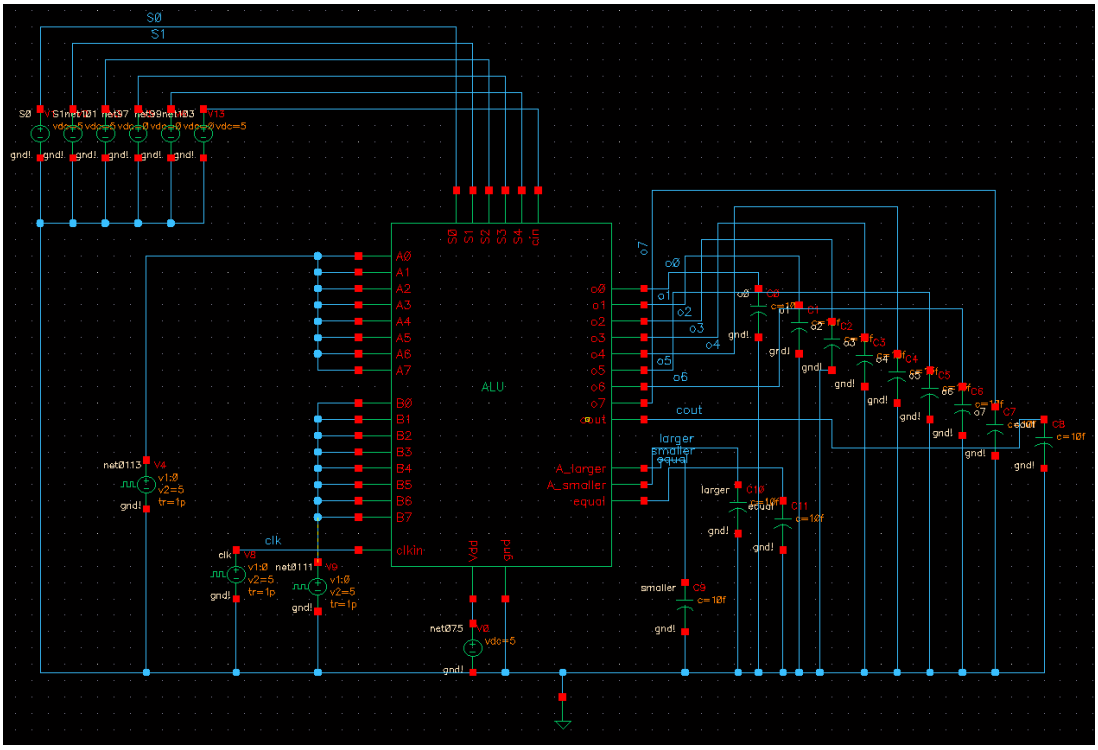


Fig 4.1 Test Bench for ALU

4.1. ALU adder

The test we did on adding operation and the correct result we should get is shown in table 4.1. and the PLS is shown in Fig 4.2.

Table 4.1 Test of ALU's adding operation

	A	B	o	Cout
Time 1	11111111	00000001	00000000	1
Time 2	00000000	00000000	00000000	0

The rise-delay for ALU level add operation is about 7.8 ns, while fall-delay is about 8.1 ns.

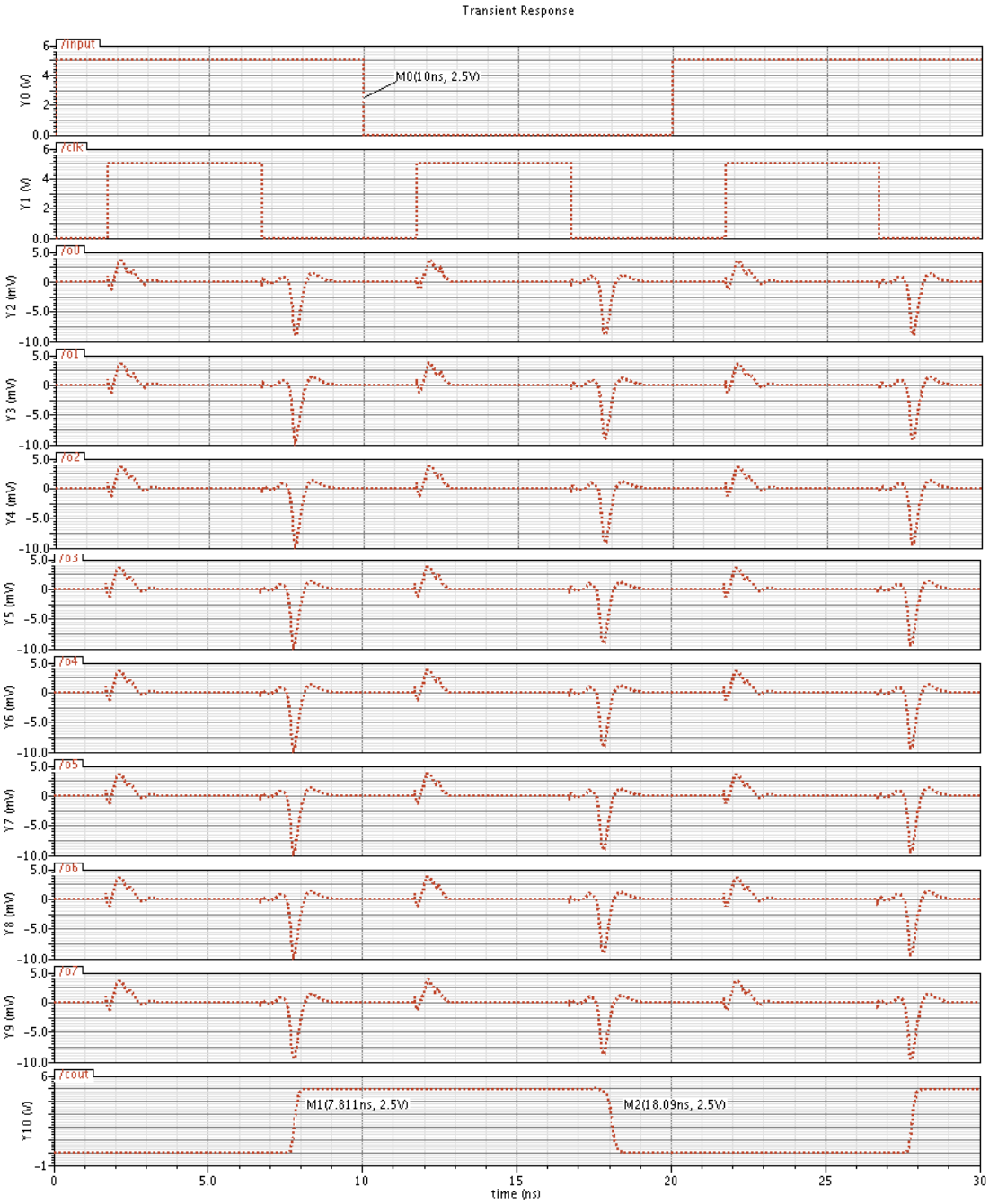


Fig 4.2 PLS for ALU level add operation

4.2. ALU subtract

The test we did on subtract operation and the correct result we should get is shown in table 4.2 and the PLS is shown in Fig 4.3. The rise-delay for ALU level subtract operation is about 7.8 ns, while fall-delay is about 8.1 ns.

Table 4.2 Test of ALU's subtract operation

	A	B	o
Time 1	11111111	00000001	11111110
Time 2	00000000	00000000	00000000

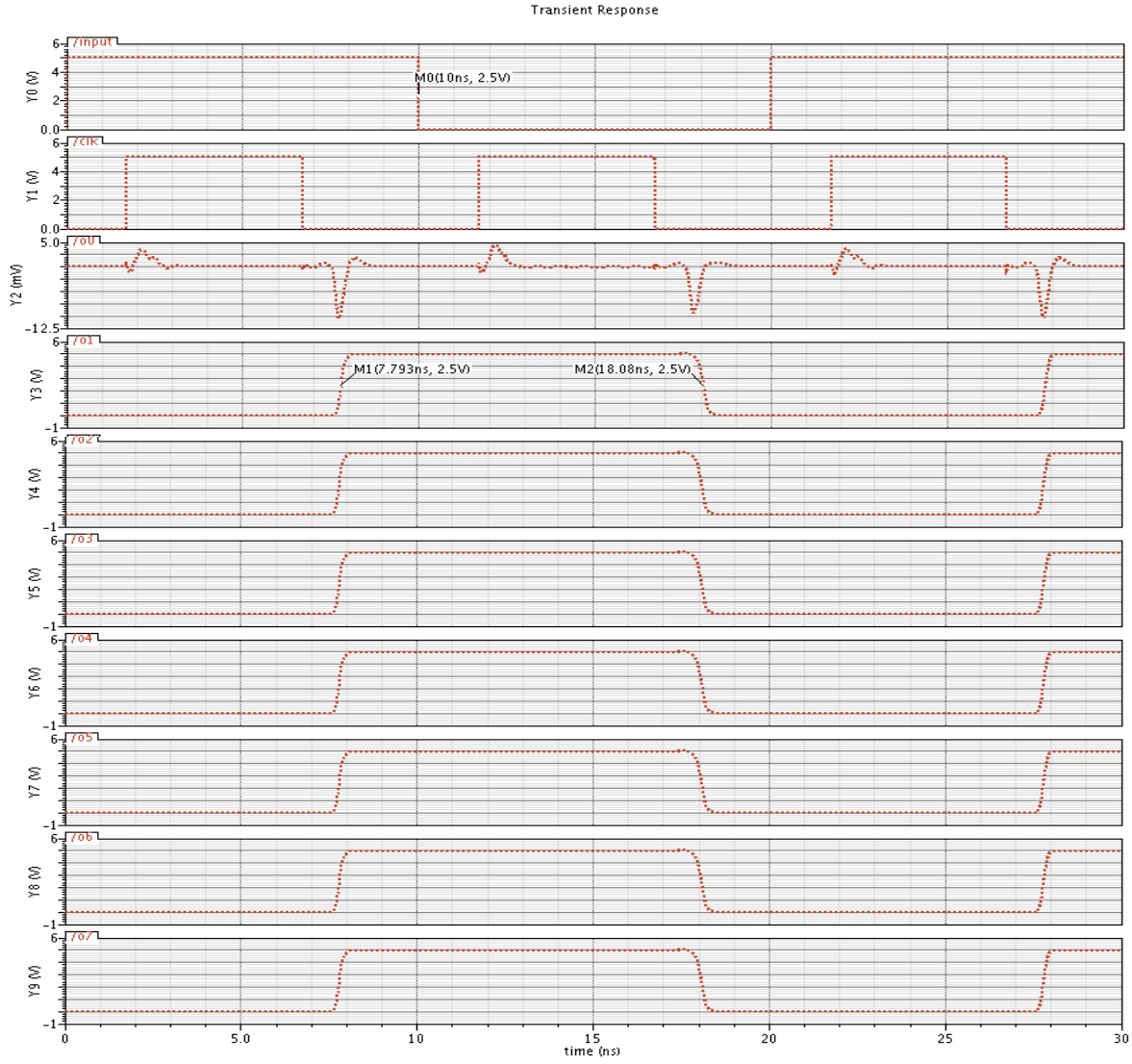


Fig 4.3 PLS for ALU level subtract operation

4.3. Comparator

In this test, we set A with all bits as input1, and set B with all bits as input2. So when input 1 is larger, the "larger" should show 1 with others showing 0. When input2 is larger, the "smaller" should show 1 with others showing 0. When input1 is equal to input2, the "equal" should show 1 with others showing 0. PLS is shown in Fig 4.4.

The rise-delay for ALU level comparator is about 8.5 ns, while fall-delay is about 8.8 ns.

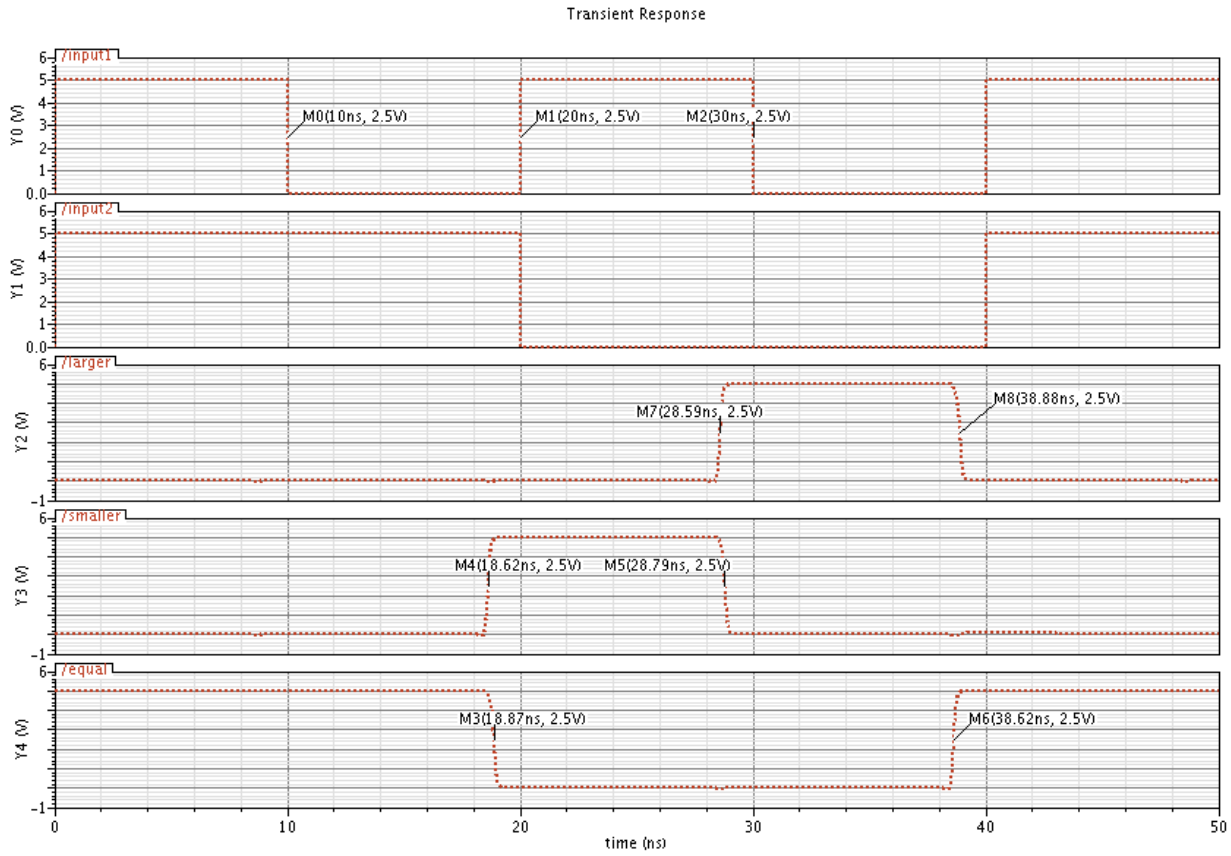


Fig 4.4 PLS for ALU level comparator operation

4.4. Increment

The test we did on Increment operation and the correct result we should get is shown in table 4.3 and the PLS is shown in Fig 4.5. The rise-delay for ALU level increment is about 7.8 ns, while fall-delay is about 8.1 ns.

Table 4.3 Test of ALU's increment operation

	A	o	Cout
Time 1	11111111	00000000	1
Time 2	00000000	00000001	0

Fig 4.5 PLS for ALU level increment operation

4.5. Decrement

The test we did on decrement operation and the correct result we should get is shown in table 4.4. and the PLS is shown in Fig 4.6. The rise-delay for ALU level decrement is about 7.8 ns, while fall-delay is about 8.1 ns.

Table 4.4 Test of ALU's decrement operation

	A	o
Time 1	11111111	11111110
Time 2	00000001	00000000

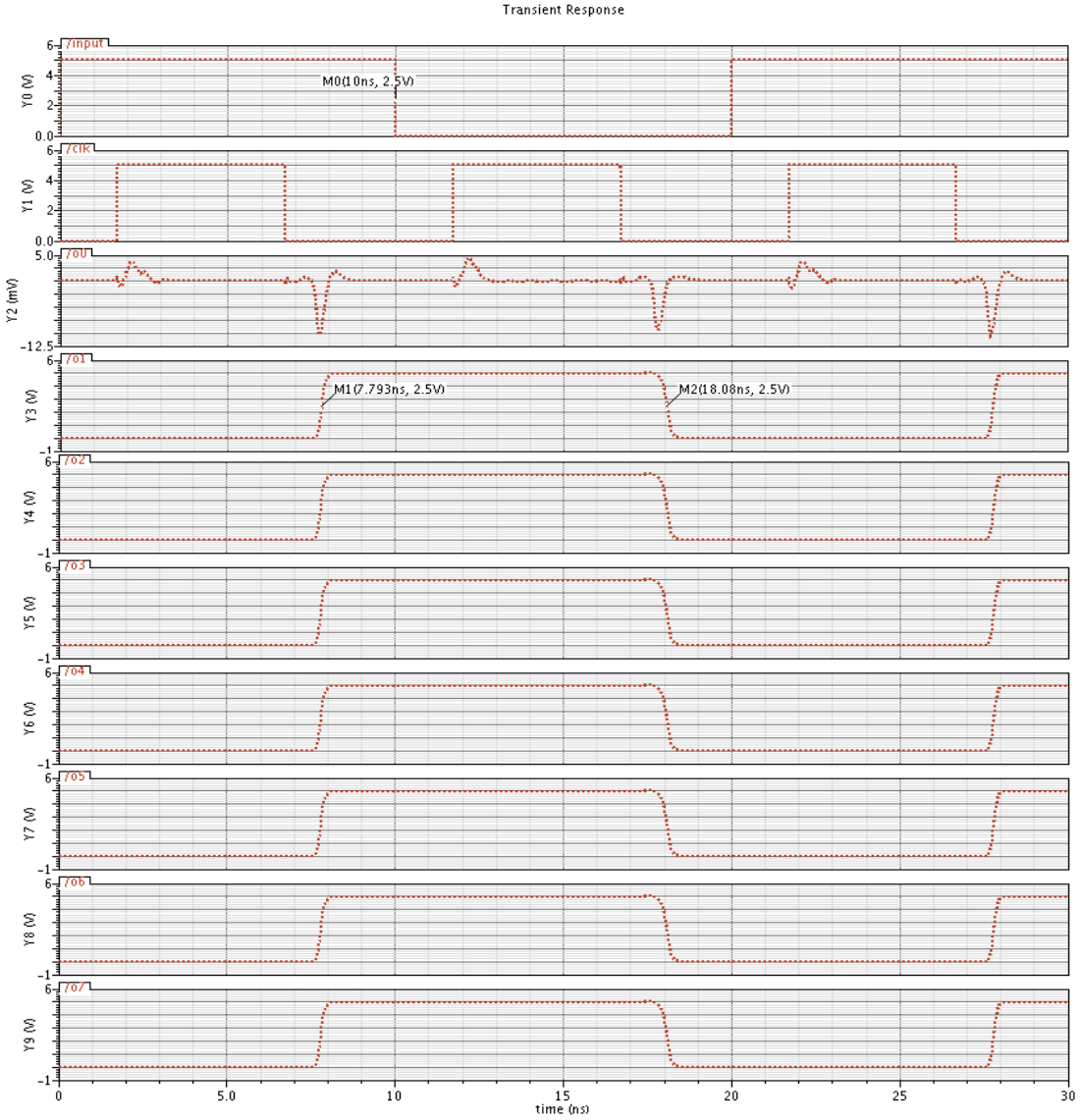


Fig 4.6 PLS for ALU level decrement operation

4.6. 1's complement

The rise-delay for ALU level decrement is about 7.8 ns, while fall-delay is about 8.1 ns. The input for the 1's complement test is oscillating from 00000000 to 00000001, so the output is oscillating from 11111111 to 11111110, as shown in Fig 4.7.

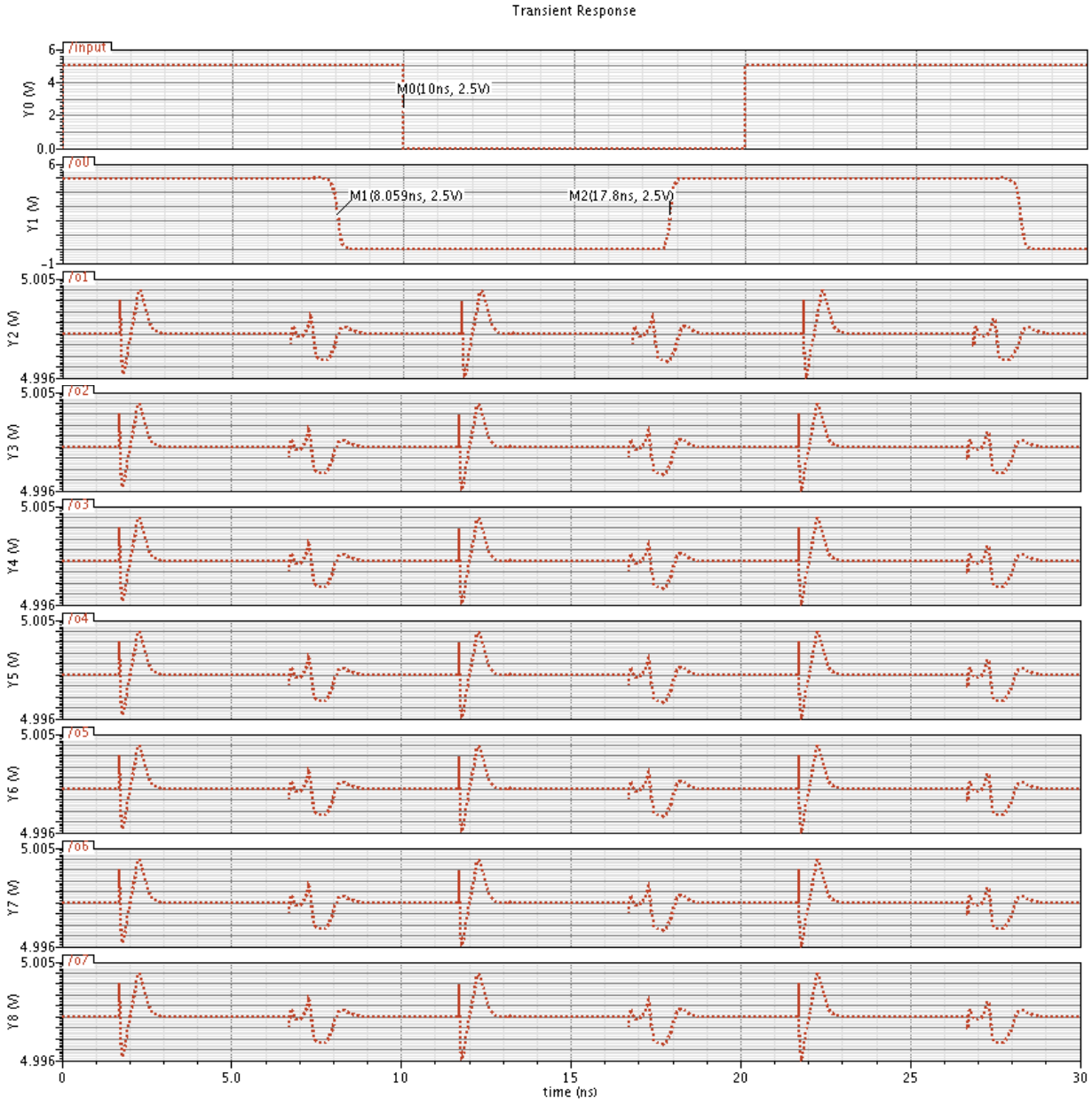


Fig 4.7 PLS for ALU level 1's complement operation

4.7. 2's complement

The rise-delay for ALU level decrement is about 7.8 ns, while fall-delay is about 8.1 ns. The input for the 1's complement test is oscillating from 00000001 to 00000011, so the output is oscillating from 11111111 to 11111101, as shown in Fig 4.8.

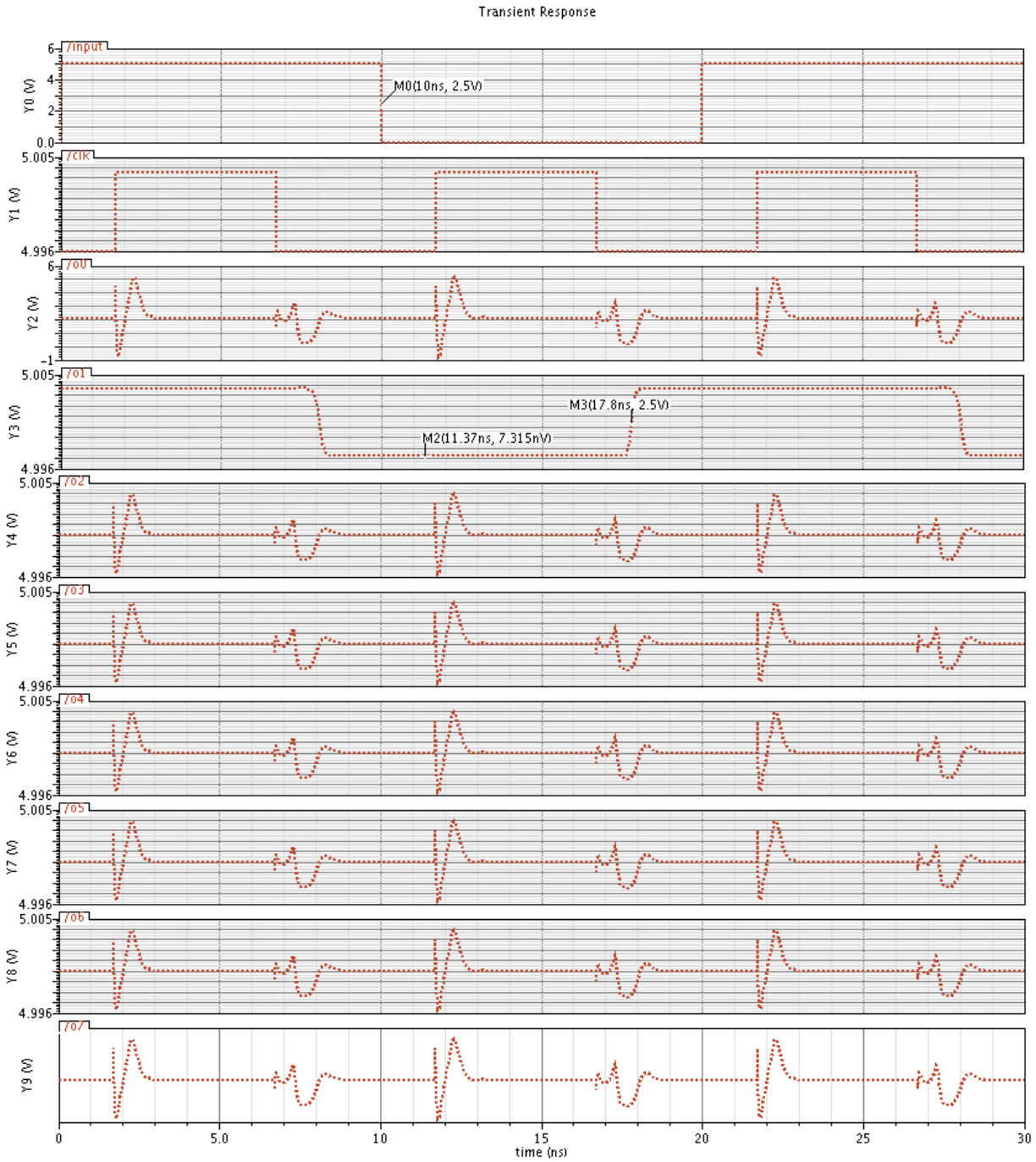


Fig 4.8 PLS for ALU level 2's complement operation

4.8. Logic

In this test, we set A as steady 11110000 and set B as steady 11111111. We are changing S0 and S1 with time. The correct result should show is shown in table 4.5 and the PLS is shown in Fig. 4.9. The rise-delay for ALU level logic operation is about 7.8 ns, while fall-delay is about 8.1 ns.

Table 4.5 Test of ALU's logic operation

S1	S0	Logic	Output
0	0	XOR	00001111
0	1	NOR	00000000
1	0	AND	11110000
1	1	OR	11111111

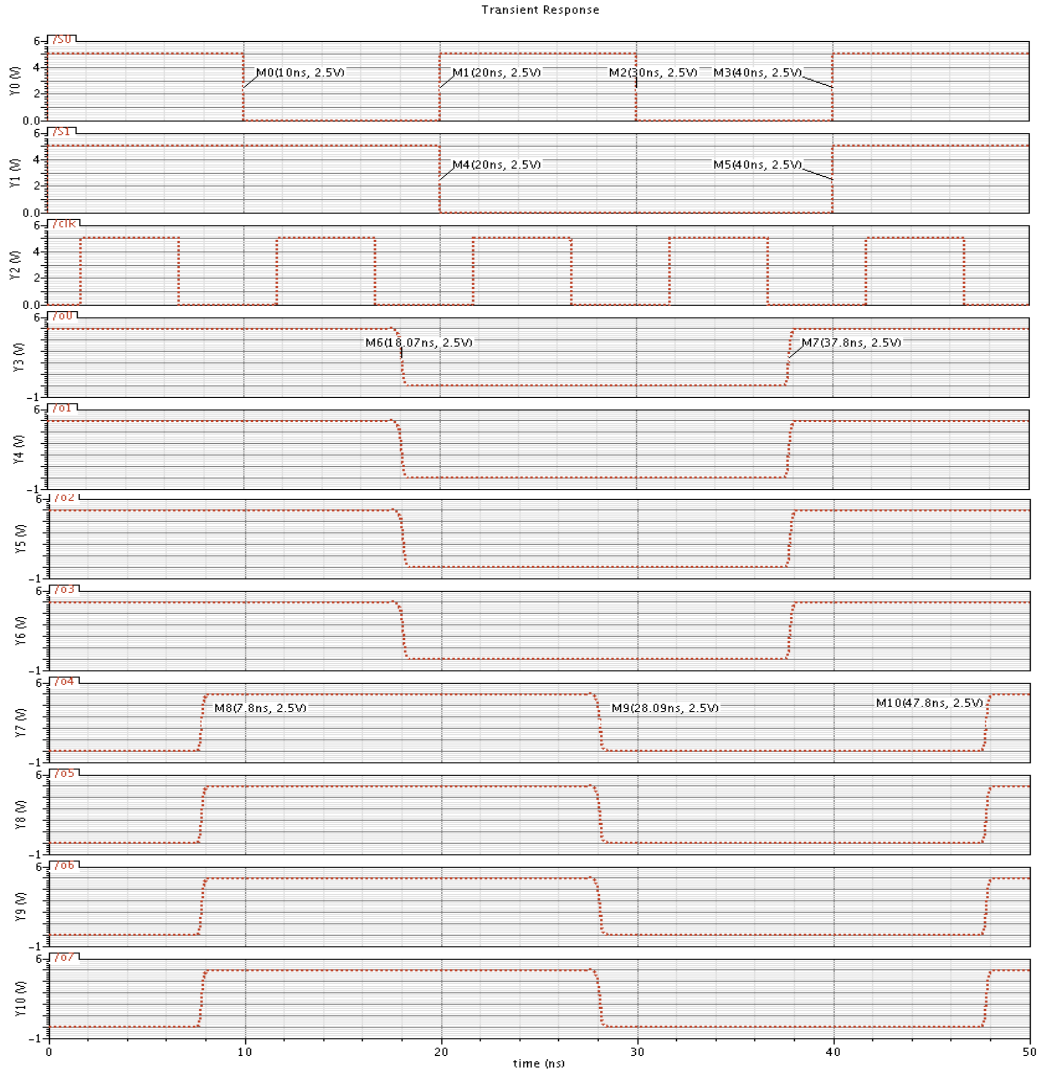


Fig 4.9 PLS for ALU level logic operation

4.9. Shift/Rotate

In this test, we set A as steady 10000111. We are changing S0 and S1 with time. The correct result should show is shown in table 4.6 and the PLS is shown in Fig. 4.10. The rise-delay for ALU level shift/rotate is about 7.8 ns, while fall-delay is about 8.1 ns.

Table 4.6 Test of ALU's logic operation

S1	S0	output function	output
0	0	shift left	00001110
0	1	shift right	01000011
1	0	rotate left	00001111
1	1	rotate right	11000011

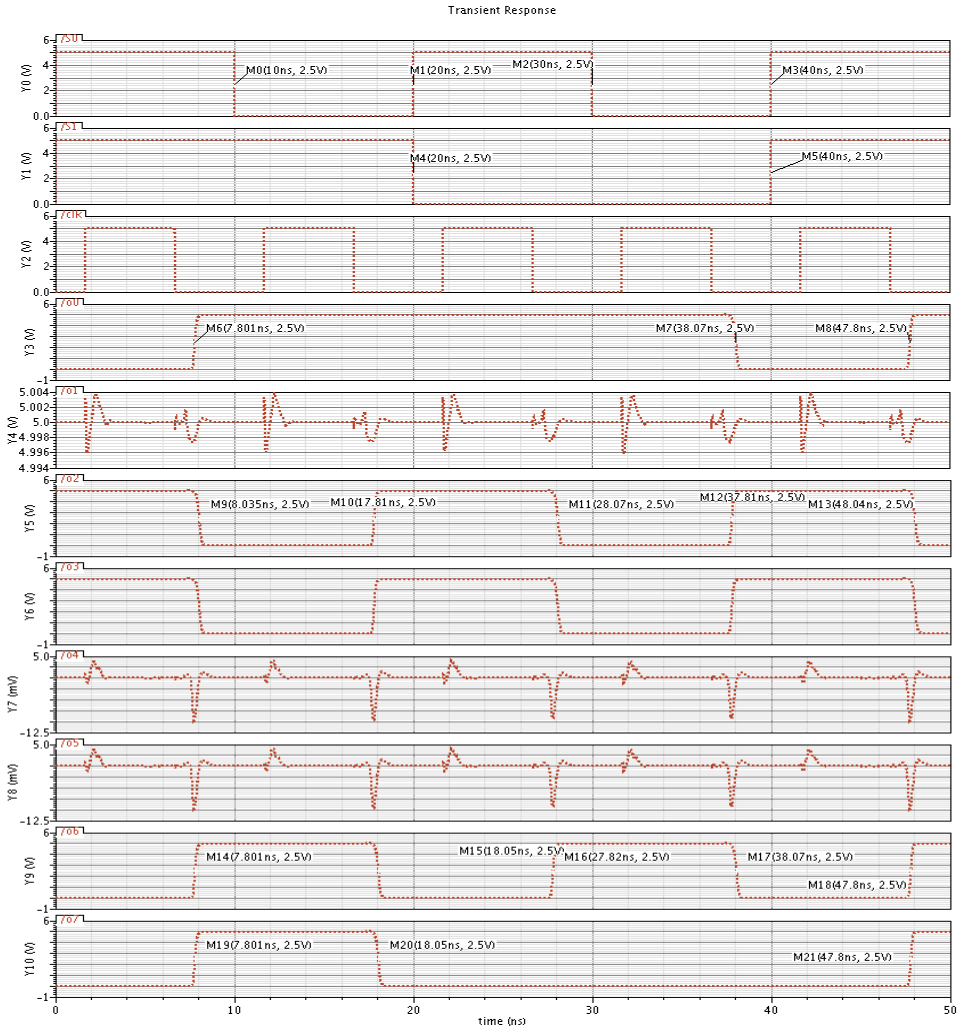


Fig 4.10 PLS for ALU level shift/rotate operation

5. Power consumption and noise margin

Noise margin means the minimum input voltage that can be considered as “1” and the maximum input voltage that can be considered as “0”. In orfer to achieve this, we set the input as A=11111111 and B=00000001 and select adding operation. We did the dc sweep for B0 (least significant bit of input B). As shown in Fig 5.1, the noise margin for “0” is 2.59V, and the noise margin for “1” is 2.71V.

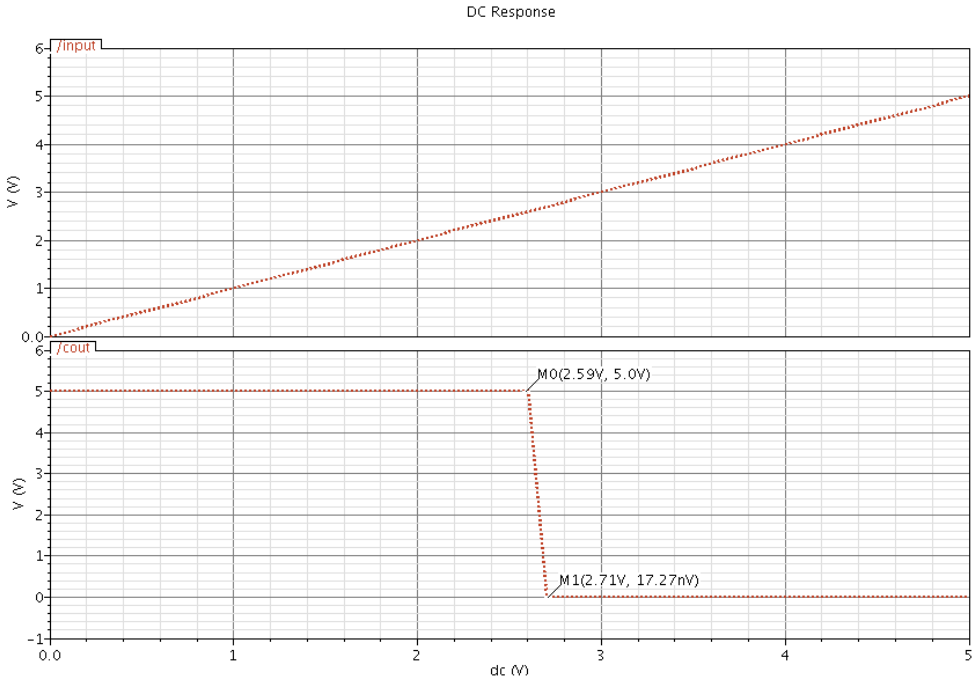


Fig 5.1 Noise margin

We also measured the maximum power consumption of all 4 blocks. By setting input A as 11111111, input B as 00000000. Also we set Cin operating at 100MHz. Using the calculator in cadence, we got the power consumption in all 4 blocks. The power consumption is 2.672 mW for algorithm block (Fig 5.2), 566.7 uW for comparator (Fig 5.3), 5.375 nW for logic block (Fig 5.4), and 2.795 nW for shift/rotate (Fig 5.5).

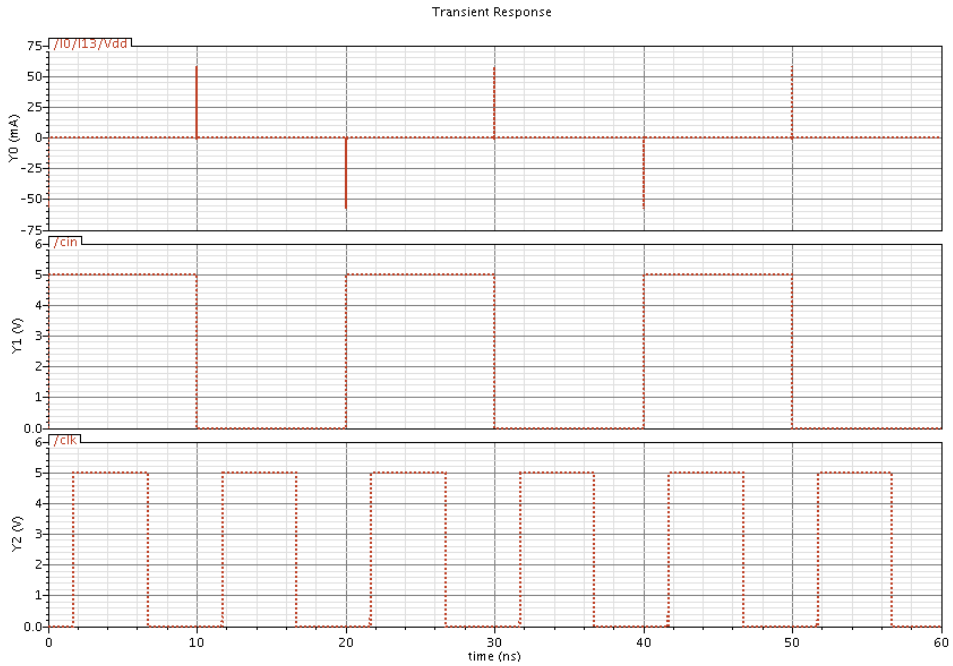


Fig 5.2 Power consumption of algorithm block

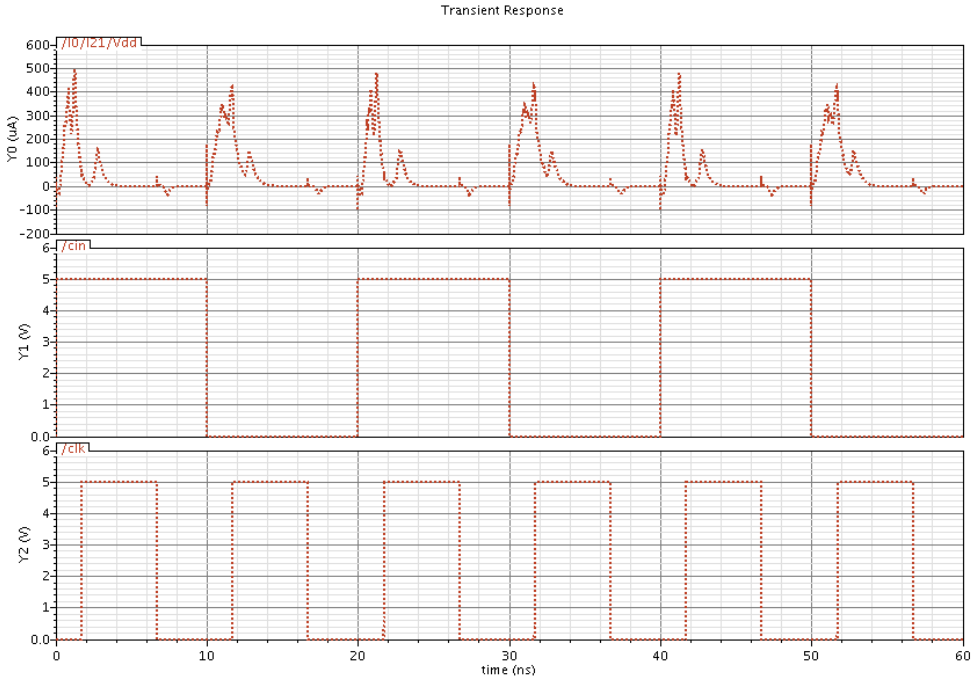


Fig 5.3 Power consumption of comparator block

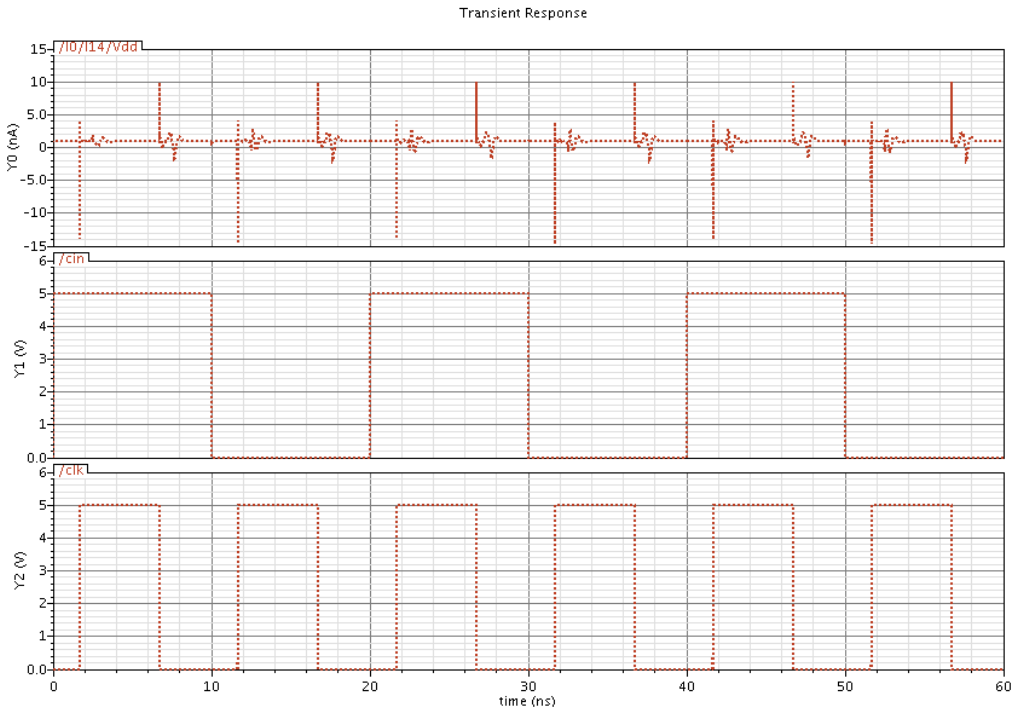


Fig 5.4 Power consumption of logic block

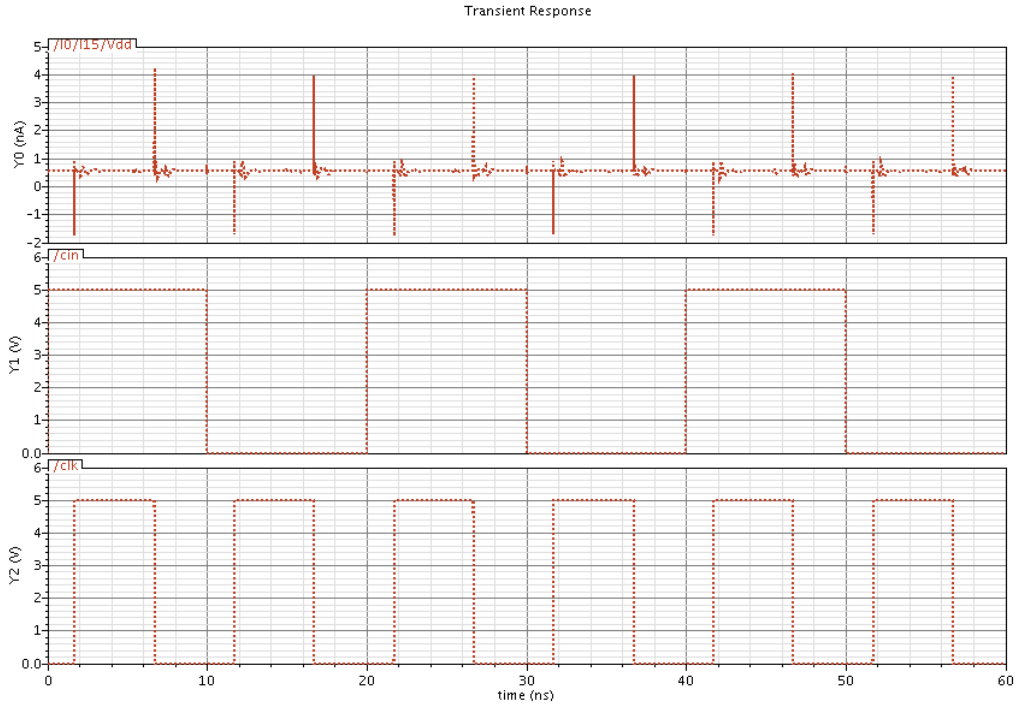


Fig 5.5 Power consumption of shift/rotate block

6. Conclusions

We successfully achieved an 8-bit ALU with a delay of 8 ns. The ALU has 4 blocks: algorithm block, comparator block, logic block, shift and rotate block. The ALU could achieve the following functions: add, subtract, increment, decrement, 1's complement, 2's complement, comparing, NOR, AND, OR, XOR, left shift, right shift, left rotate and right rotate. The chip performances are as follow:

- Die area: 330 × 352 um^2.
- Power dissipation: 2.672 mW for algorithm block, 566.7 uW for comparator, 5.375 nW for logic block， and 2.795 nW for shift/rotate block.
- Maximum clock frequency: slightly larger than 100MHz
- Noise margin: 2.59V for "0", 2.71V for "1".

There is a trade-off between the transistor size and delay of the whole system. In this project we use Wn/Ln = 1.5/0.6 um, Wp/Lp = 3/0.6 um. If the widths of the transistors are increased, the delay time of the system can be decreased.

7. Reference

[1] WIKI: http://en.wikipedia.org/wiki/Arithmetic_logic_unit

[2] http://whatis.techtarget.com/definition/arithmetic-logic-unit-ALU

Appendix:

Schematic, symbol, Verilog and layout (PLS) for basic gates

1. 1-bit Inverter

a) Schematic, symbol and layout are shown in Fig A.1.1

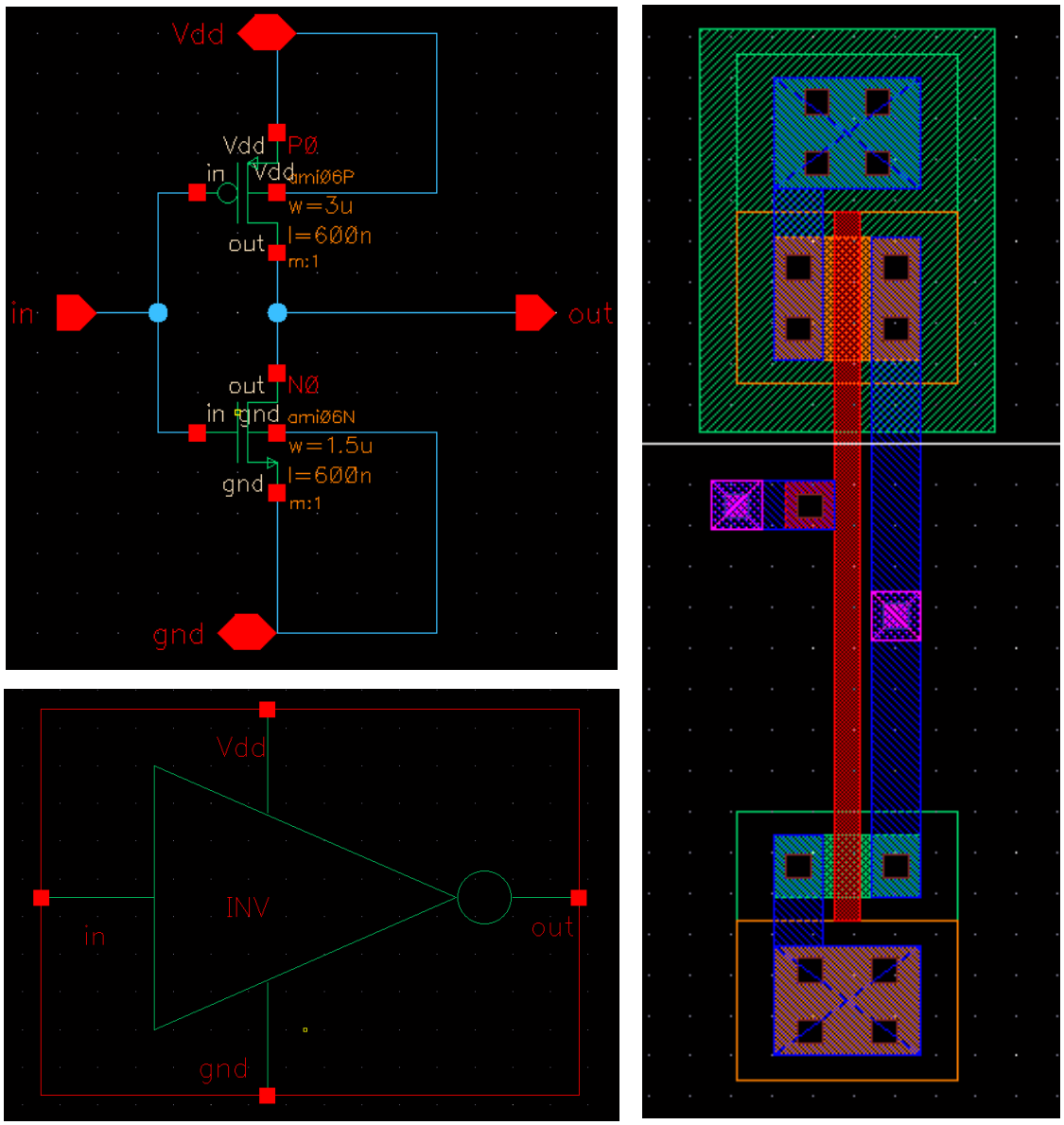


Fig A.1.1, Schematic, symbol and layout of 1-bit inverter

b) Verilog code and simulation are shown in Fig A.1.2 and Fig A.1.3

```
//Verilog HDL for "ALU", "inv" "functional"

`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module inv ( out, Vdd, gnd, in );

  inout gnd;
  inout Vdd;
  input in;
  output out;

  not (out,Vdd, gnd, in);

endmodule

`endcelldefine
```

Fig A.1.2 Verilog code of 1-bit inverter

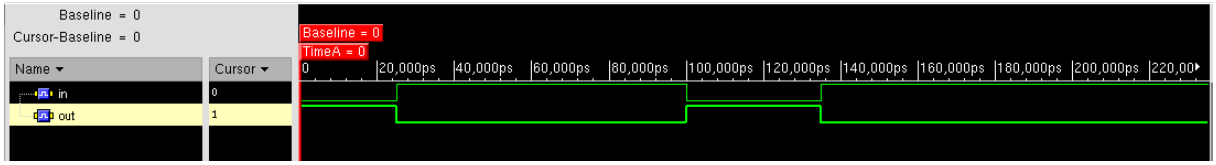


Fig A.1.3 Verilog simulation of 1-bit inverter

c) PLS of 1-bit inverter is shown in Fig A.1.4

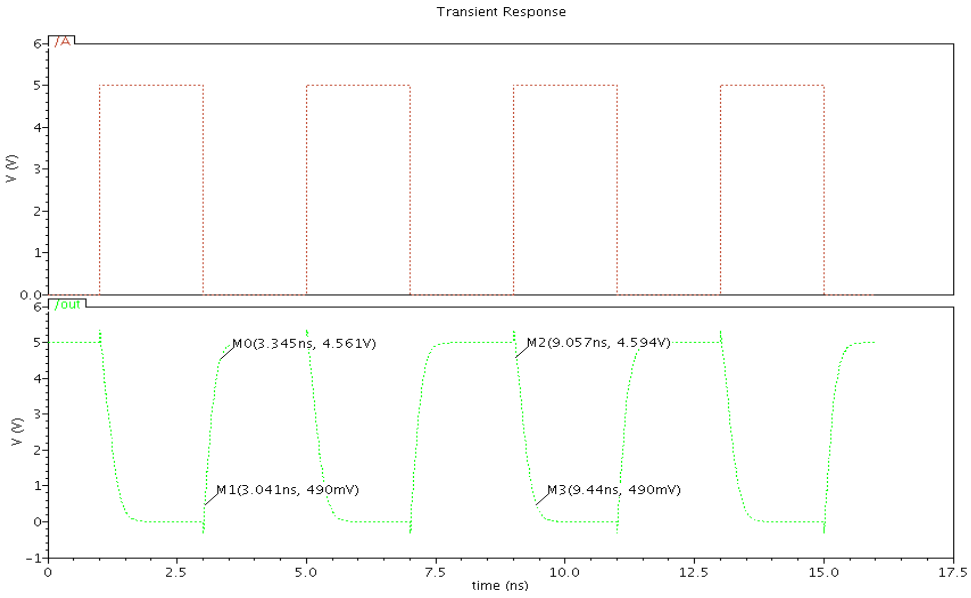


Fig A.1.4 PLS of 1-bit inverter

2. 1-bit NAND gate

a) Schematic, symbol and layout are shown in Fig A.2.1

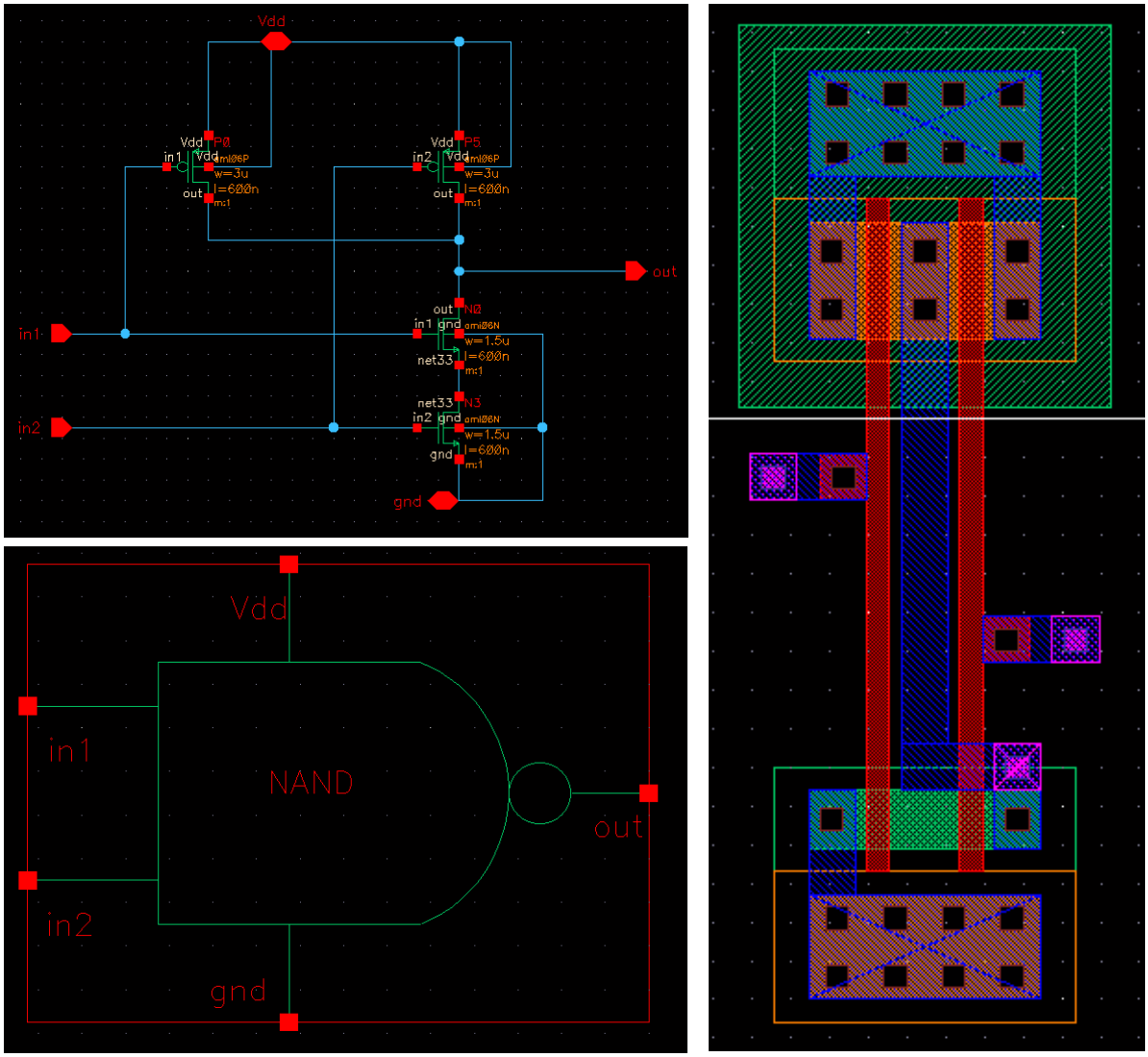


Fig A.2.1, Schematic, symbol and layout of 1-bit NAND

b) Verilog code and simulation are shown in Fig A.2.2 and Fig A.2.3

```
//Verilog HDL for "ALU", "nand" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module \nand_  ( out, Vdd, gnd, in1, in2 );

  inout gnd;
  inout Vdd;
  input in2;
  input in1;
  output out;

   nand (strong1, strong0)#1(out, in1, in2);

endmodule
`endcelldefine
```

Fig A.2.2 Verilog code of 1-bit NAND

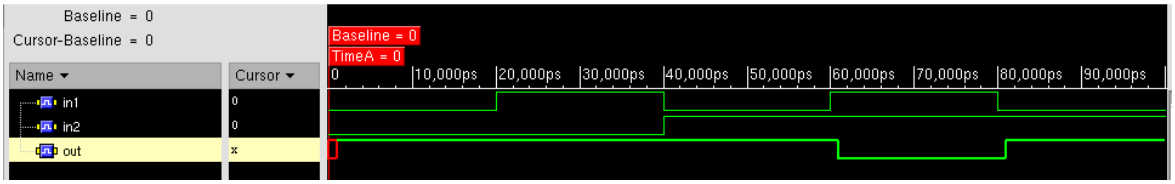


Fig A.2.3 Verilog simulation of 1-bit NAND

c) PLS is shown in Fig A.2.4

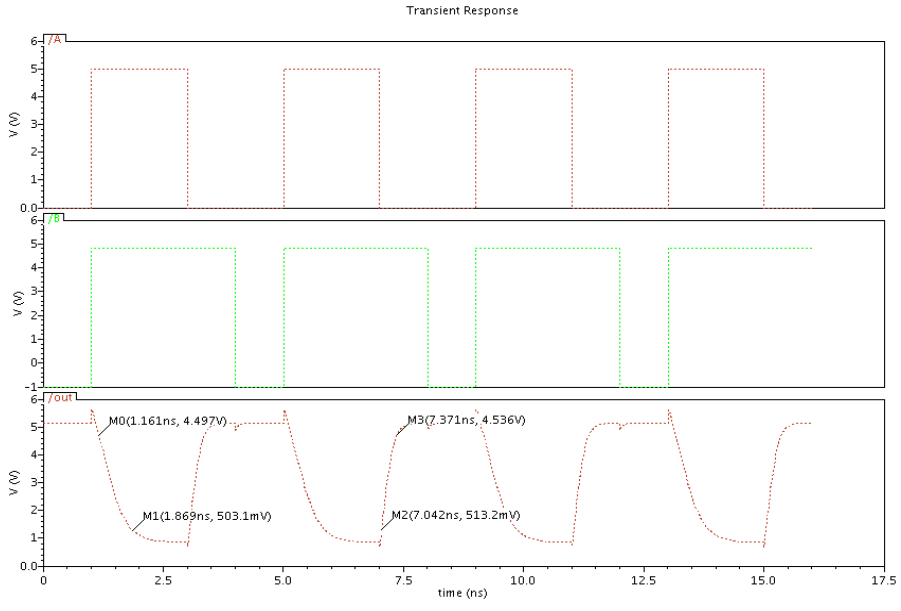


Fig A.2.4 PLS of 1-bit NAND

3. 1-bit NOR gate

a) Schematic, symbol and layout are shown in Fig A.3.1

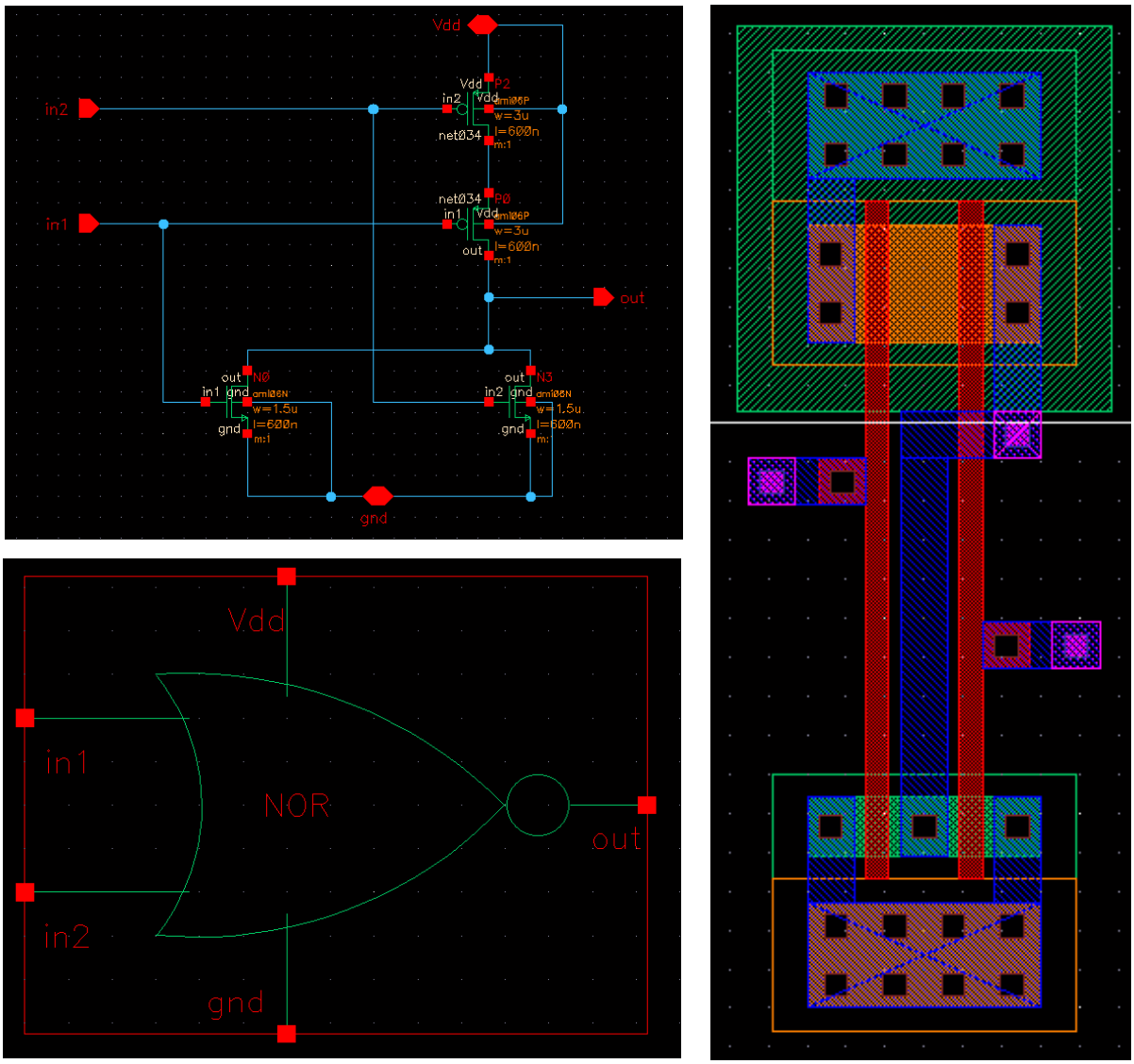


Fig A.3.1, Schematic, symbol and layout of 1-bit NOR

b) Verilog code and simulation are shown in Fig A.3.2 and Fig. A.3.3.

```
//Verilog HDL for "ALU", "nor" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module \nor_ ( out, Vdd, gnd, in1, in2 );

  inout gnd;
  inout Vdd;
  input in2;
  input in1;
  output out;

  nor(strong1, strong0) #1(out, in1, in2);

endmodule
`endcelldefine
```

Fig A.3.2 Verilog code of 1-bit NOR

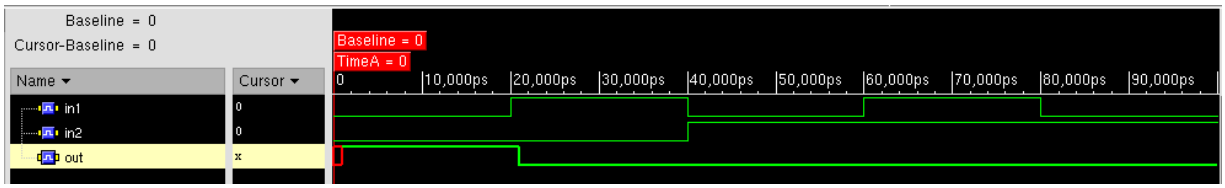


Fig A.3.3 Verilog simulation of 1-bit NOR

c) PLS is shown in Fig A.3.4.

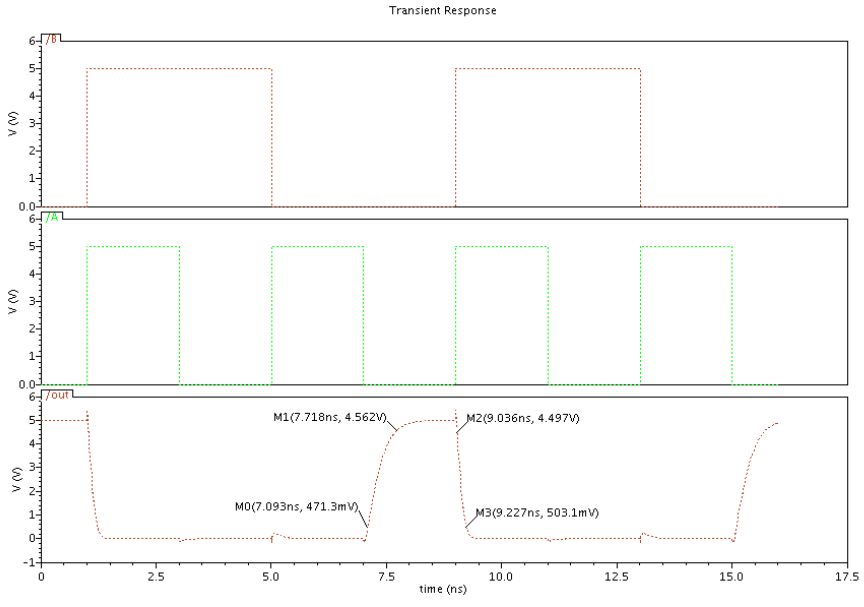


Fig A.3.4 PLS of 1-bit NOR

4. 1-bit AND gate

a) Schematic, symbol and layout are shown in Fig A.4.1.

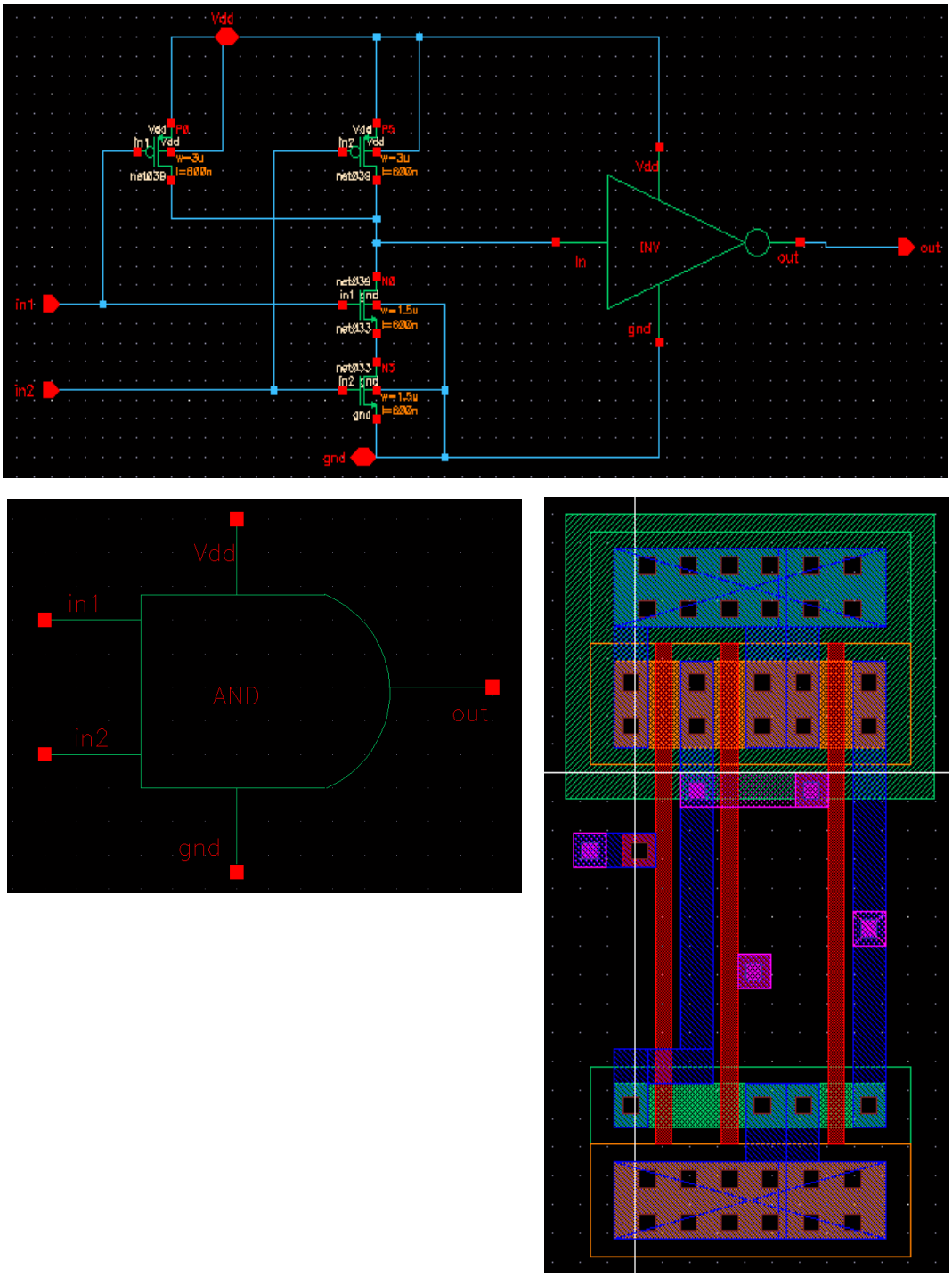


Fig A.4.1, Schematic, symbol and layout of 1-bit AND

b) Verilog code and simulation are shown in Fig A.4.2 and A.4.3

```
//Verilog HDL for "ALU", "and" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module \and_  ( out, gnd, in1, in2, Vdd );

  inout gnd;
  inout Vdd;
  input in2;
  input in1;
  output out;

   and(strong1, strong0)#1(out, in1, in2);

endmodule
`endcelldefine
```

Fig A.4.2 Verilog code of 1-bit AND

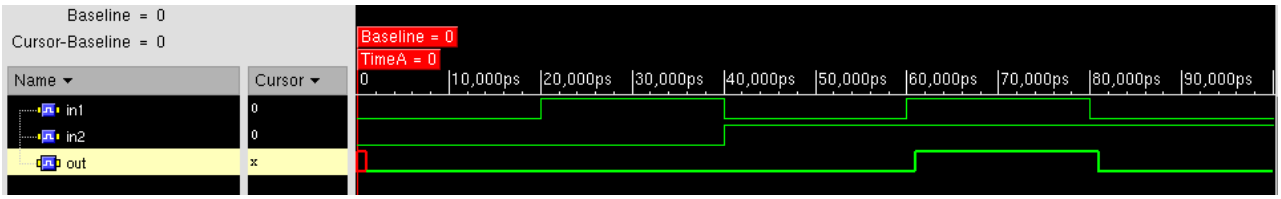


Fig A.4.3 Verilog simulation of 1-bit AND

c) PLS is shown in Fig A.4.4

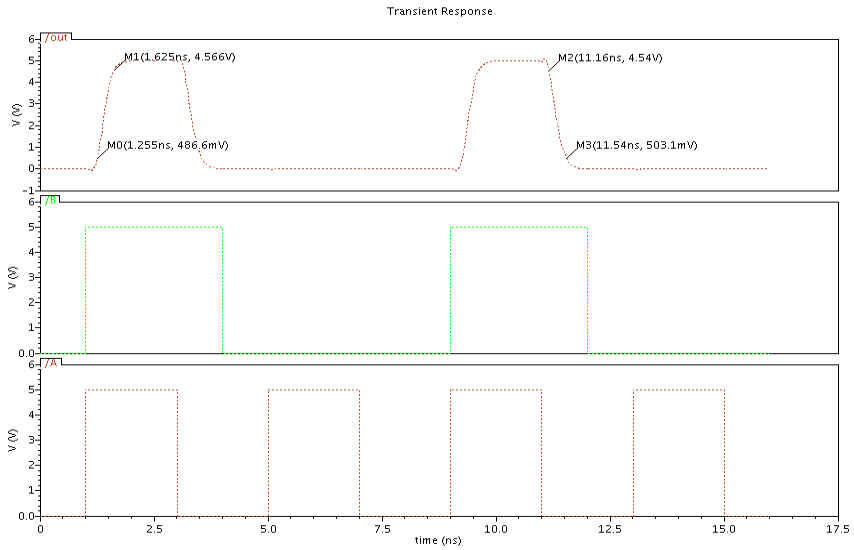


Fig A.4.4 PLS of 1-bit AND

5. 1-bit OR gate

a) Schematic, symbol and layout are shown in Fig A.5.1.

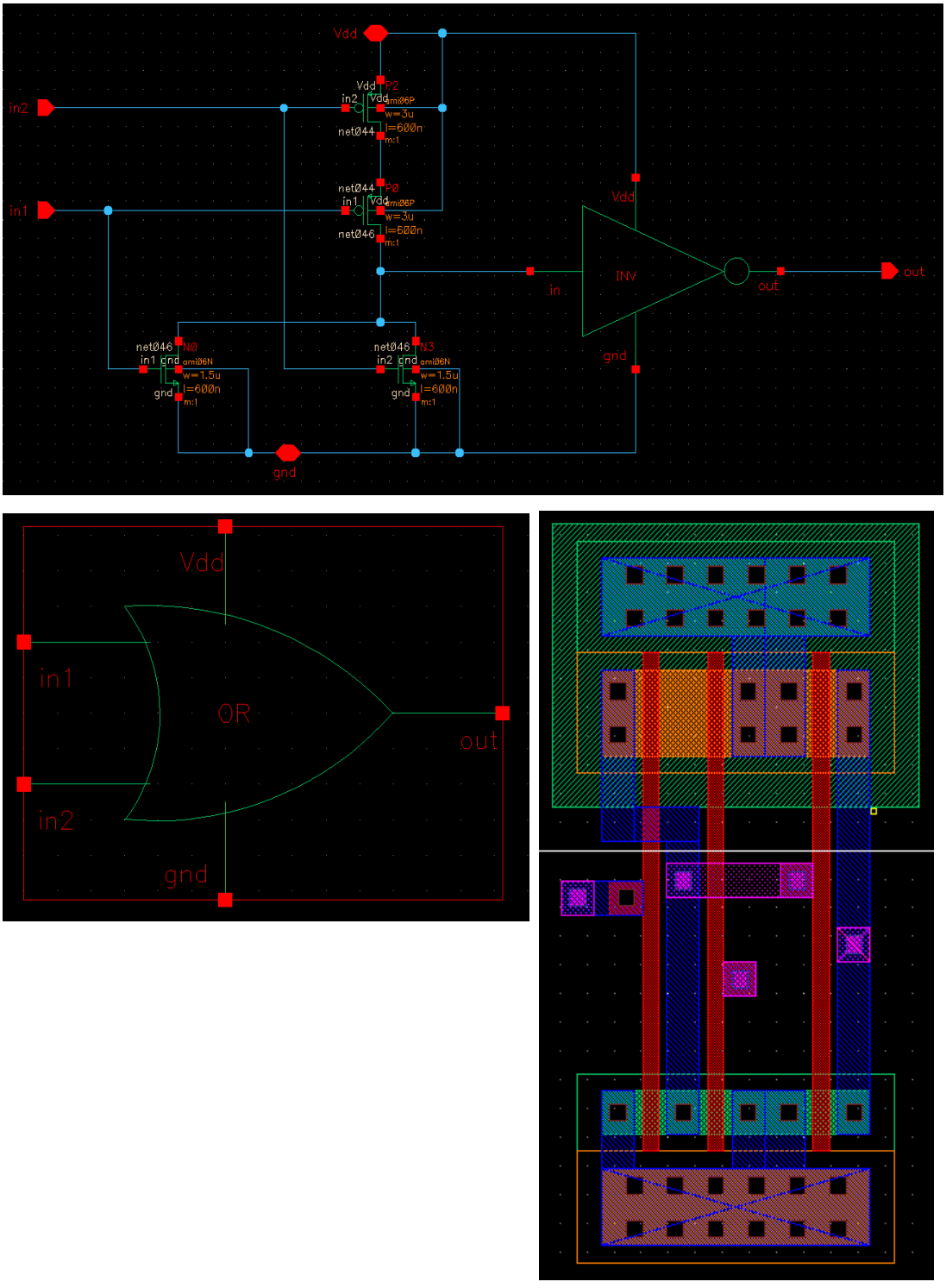


Fig A.5.1, Schematic, symbol and layout of 1-bit OR

b) Verilog code and simulation are shown in Fig A.5.2 and A.5.3.

```
//Verilog HDL for "ALU", "or" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module \or_  ( out, Vdd, gnd, in1, in2 );

  inout gnd;
  inout Vdd;
  input in2;
  input in1;
  output out;

   or(strong1, strong0)#1(out, in1, in2);

endmodule
`endcelldefine
```

Fig A.5.2 Verilog code of 1-bit OR

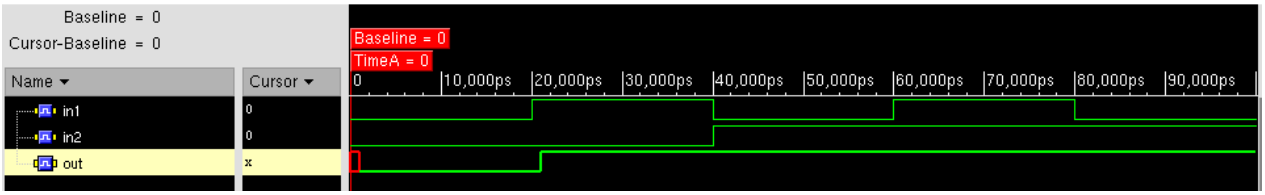


Fig A.5.3 Verilog simulation of 1-bit OR

c) PLS is shown in Fig A.5.4.

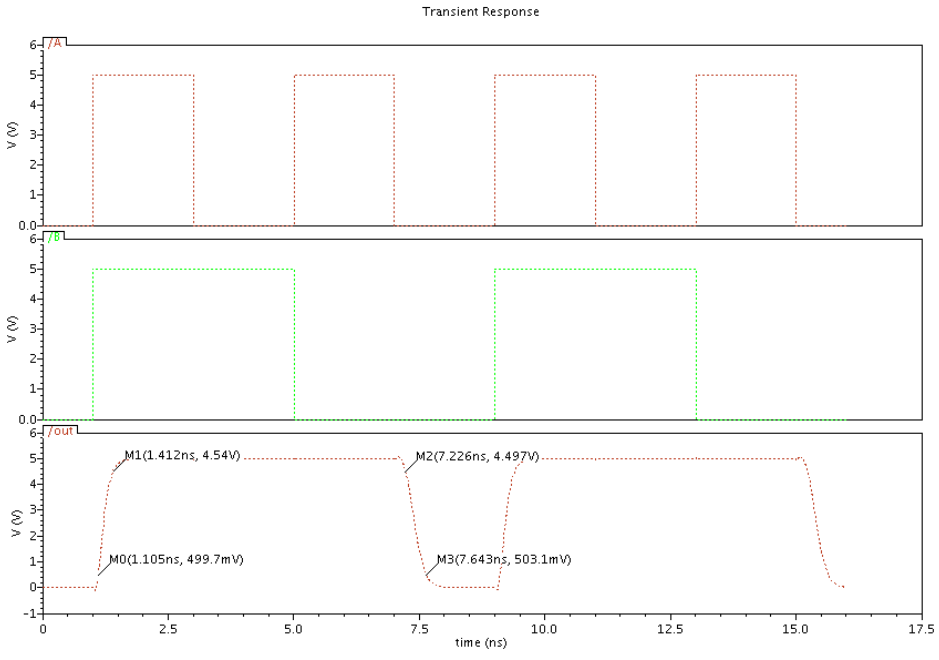


Fig A.5.4 PLS simulation of 1-bit OR

6. 1-bit XOR gate

a) Schematic, symbol and layout are shown in Fig A.6.1.

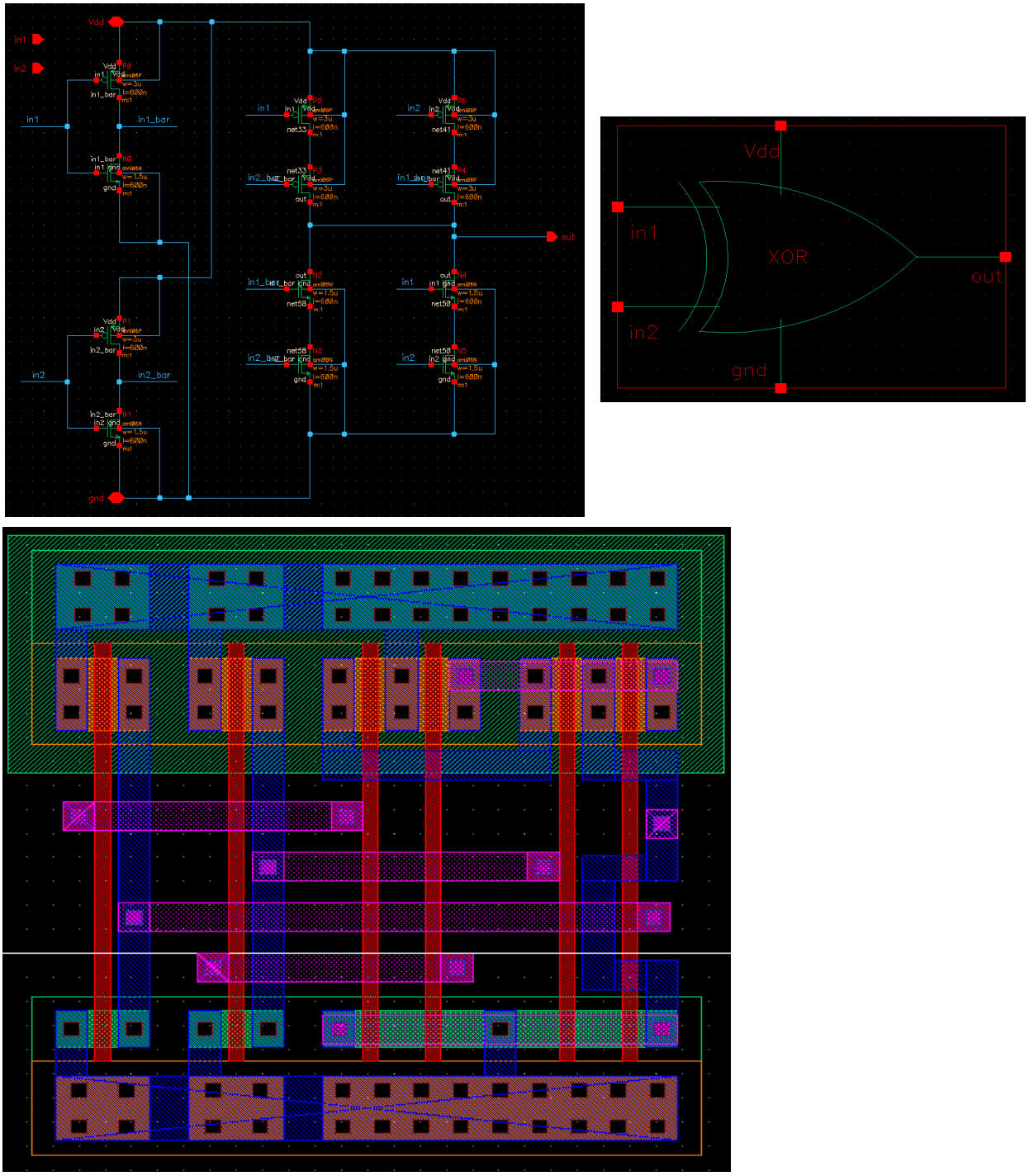


Fig A.6.1, Schematic, symbol and layout of 1-bit XOR

b) Verilog code and simulation are shown in Fig A.6.2 and Fig A.6.3.

```
//Verilog HDL for "ALU", "xor" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module \xor_  ( out, gnd, in1, in2, Vdd );

  inout gnd;
  inout Vdd;
  input in2;
  input in1;
  output out;

   xor(strong1, strong0)#1(out, in1, in2);

endmodule
`endcelldefine
```

Fig A.6.2 Verilog code of 1-bit XOR

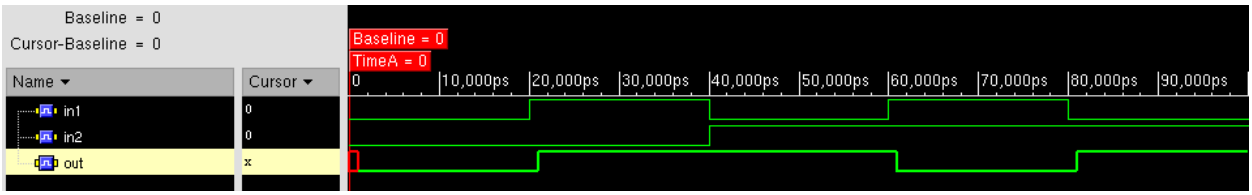


Fig A.6.3 Verilog simulation of 1-bit XOR

c) PLS is shown in Fig A.6.4.

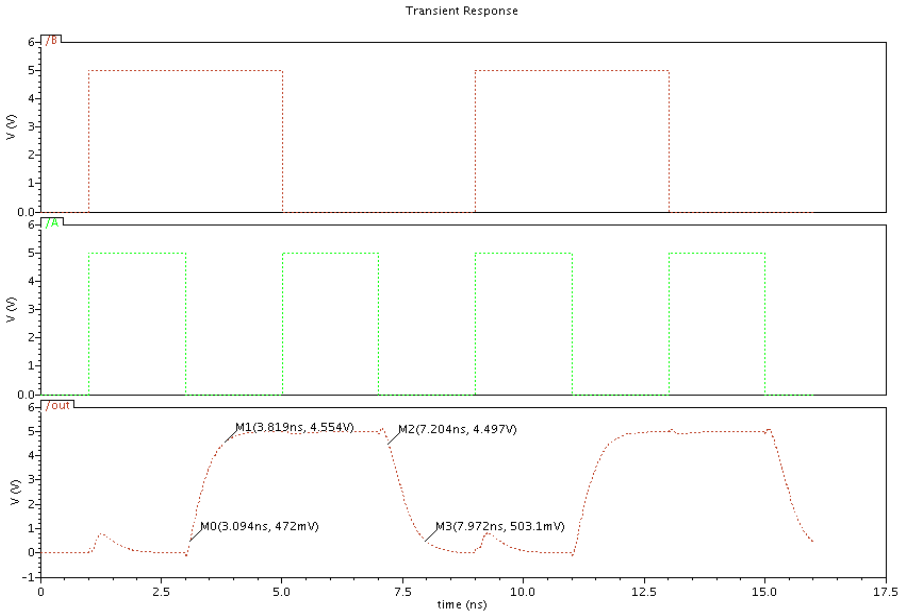


Fig A.6.4 PLS of 1-bit XOR

7. 1-bit MUX gate

a) Schematic, symbol and layout are shown in Fig A.7.1.

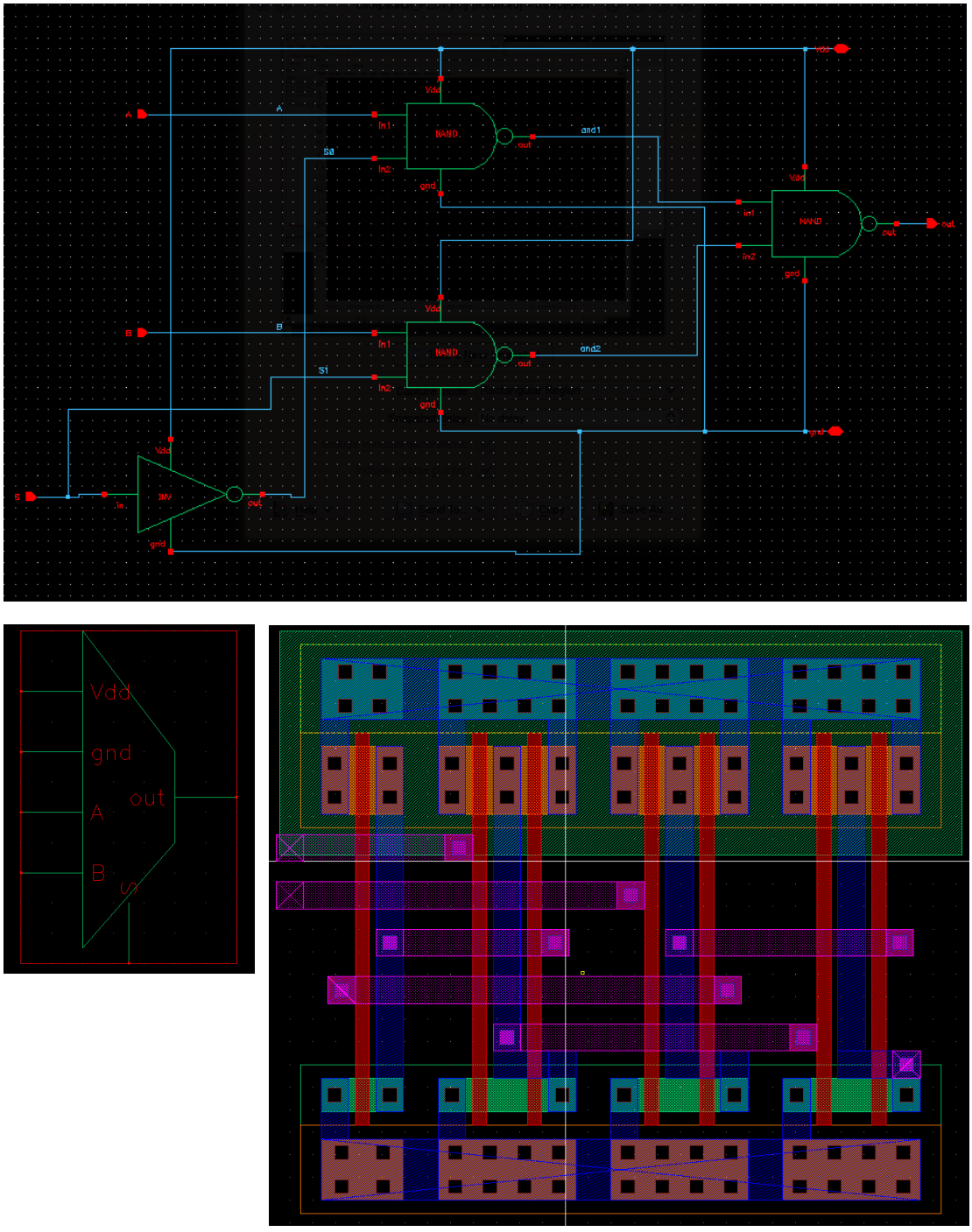
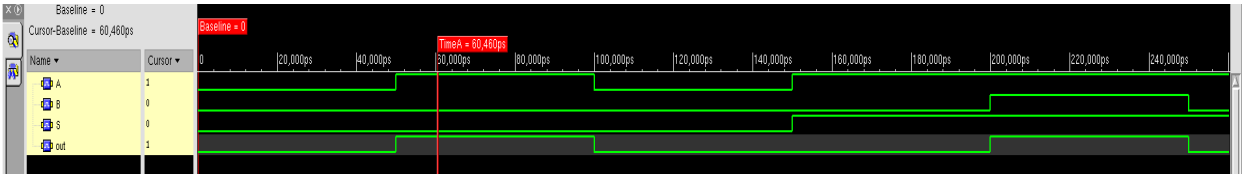


Fig A.7.1, Schematic, symbol and layout of 1-bit MUX

b) Verilog code and simulation A.7.2 and A.7.3

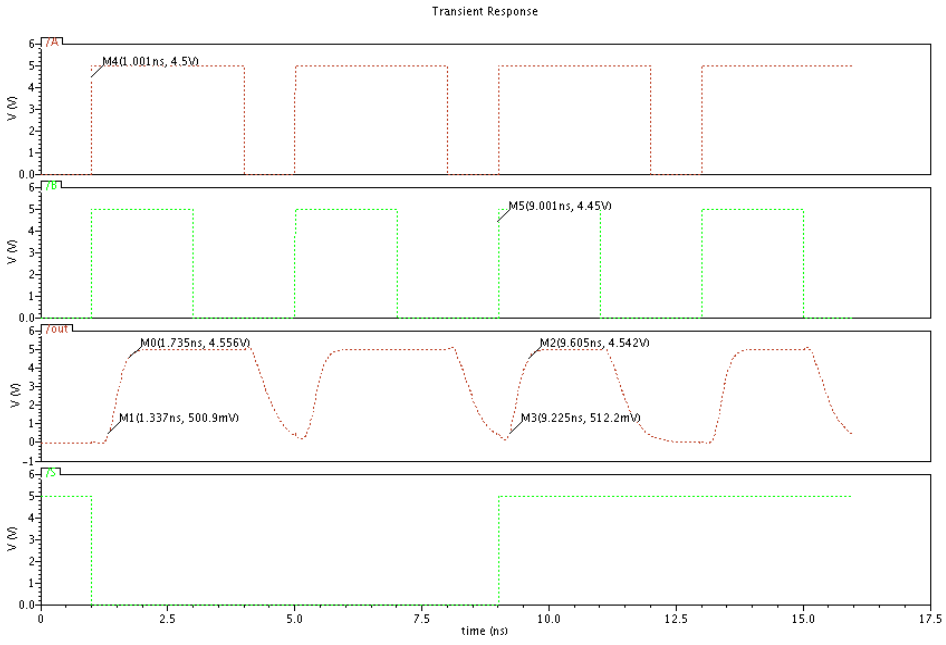
```
//Verilog HDL for "ALU", "mux" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module mux ( out, Vdd, gnd, A, B, S );

  inout gnd, Vdd;
  input S,A,B;
  output out;
   wire  Sb,p,q;

   not  (Sb,S);
   nand (p,A,Sb);
   nand (q,B,S);
   nand (out,p,q);

endmodule
`endcelldefine
```

A.7.2 Verilog code of 1bit MUX



A.7.3 Verilog simulation of 1bit MUX

c) PLS is shown in Fig A.7.4.



A.7.4 PLS simulation of 1bit MUX

8. 8-bit MUX

a) Schematic, symbol and layout are shown in Fig A.8.1.

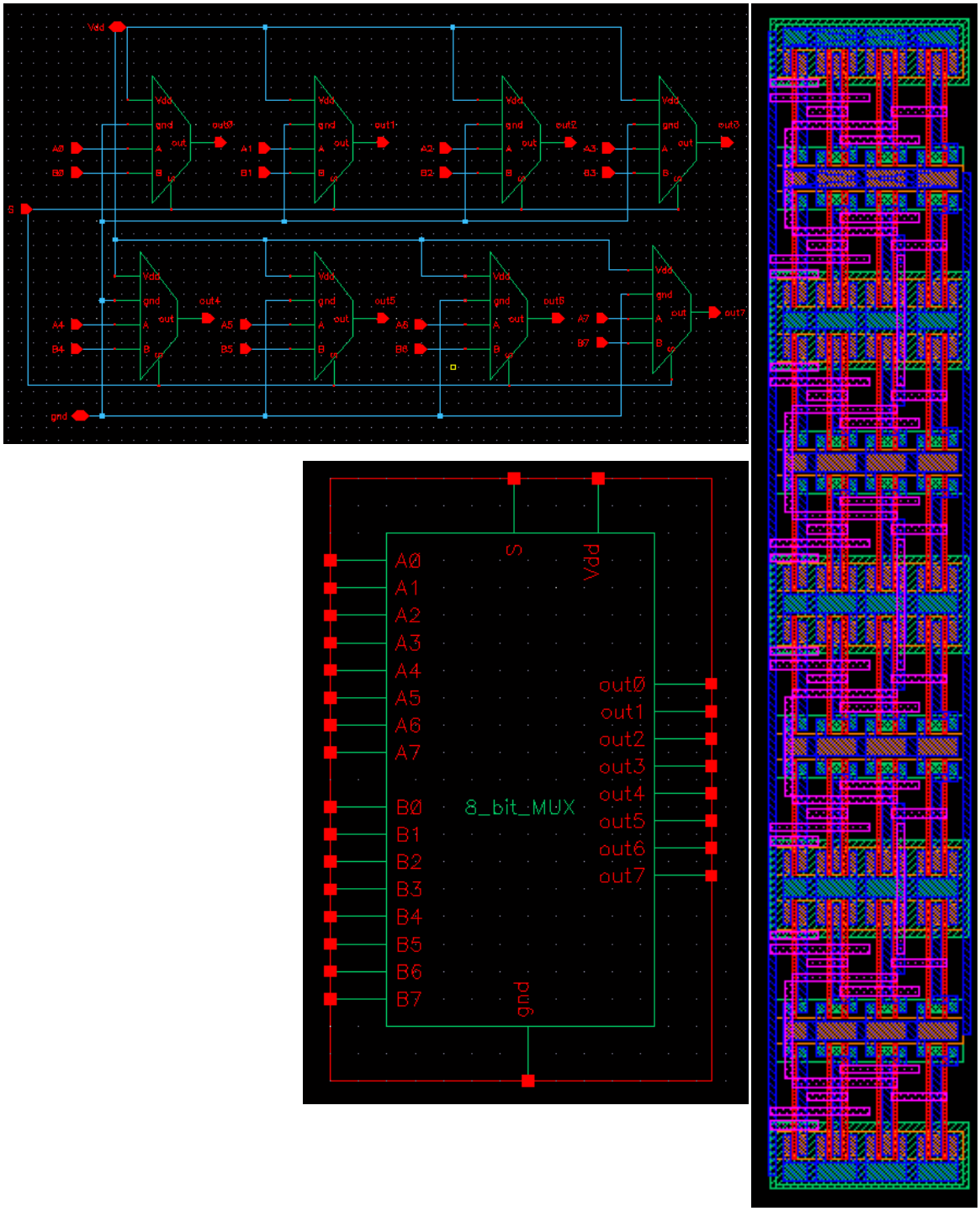


Fig A.8.1, Schematic, symbol and layout of 8-bit MUX

b) Verilog code and simulation are shown in Fig A.8.2 and Fig A.8.3.

```
//Verilog HDL for "ALU", "mux_8" "functional"

`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module mux_8 ( out0, out1, out2, out3, out4, out5, out6, out7, Vdd, gnd, A0,
    A1, A2, A3, A4, A5, A6, A7, B0, B1, B2, B3, B4, B5, B6, B7, S );

  output out7,out6,out5,out4,out3,out2,out1,out0;
  input B7,B6,B5,B4,B3,B2,B1,B0;
  input A0,A1,A2,A3,A4,A5,A6,A7;
  input S;
  inout gnd;
  inout Vdd;

   wire  Sb,p0,p1,p2,p3,p4,p5,p6,p7,q0,q1,q2,q3,q4,q5,q6,q7;

    mux n1 (out0,Vdd, gnd, A0,B0,S);
    mux n2 (out1,Vdd, gnd, A1,B1,S);
    mux n3 (out2,Vdd, gnd, A2,B2,S);
    mux n4 (out3,Vdd, gnd, A3,B3,S);
    mux n5 (out4,Vdd, gnd, A4,B4,S);
    mux n6 (out5,Vdd, gnd, A5,B5,S);
    mux n7 (out6,Vdd, gnd, A6,B6,S);
    mux n8 (out7,Vdd, gnd, A7,B7,S);

endmodule
`endcelldefine
```

Fig A.8.2 Verilog code of 8-bit MUX

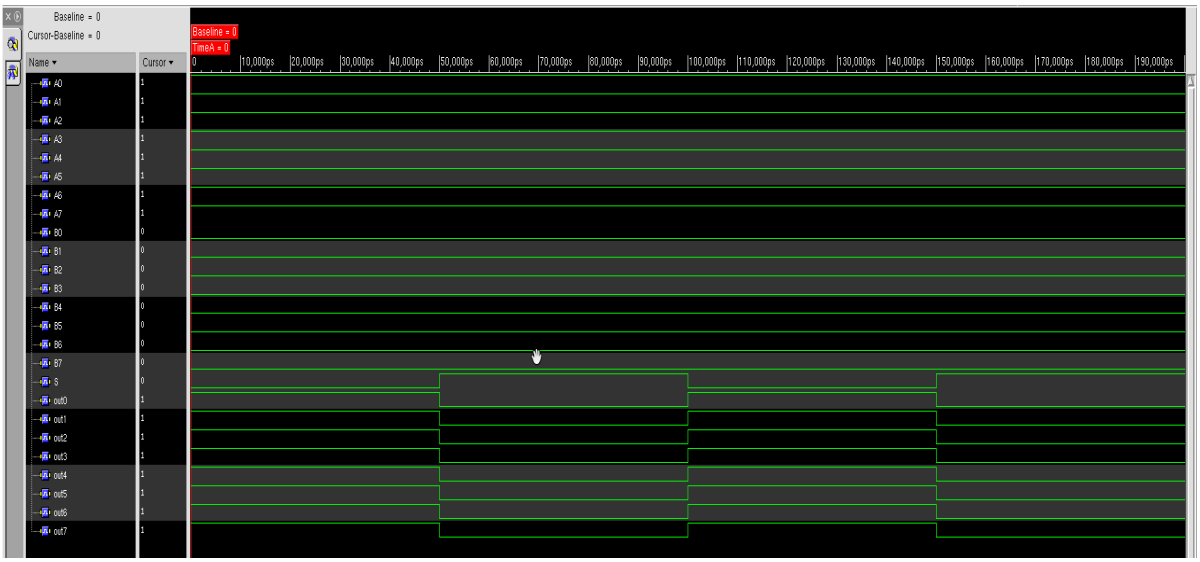


Fig A.8.3 Verilog simulation of 8-bit MUX

9. 1-bit Dynamic DFF

a) Schematic, symbol and layout are shown in Fig A.9.1, Fig A.9.2, Fig A.9.3.

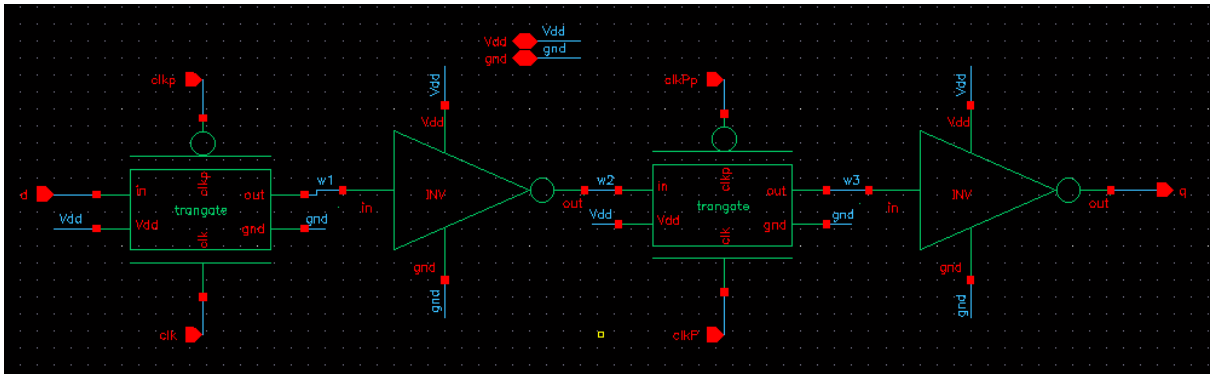


Fig A.9.1 Schematic of 1-bit Dynamic DFF

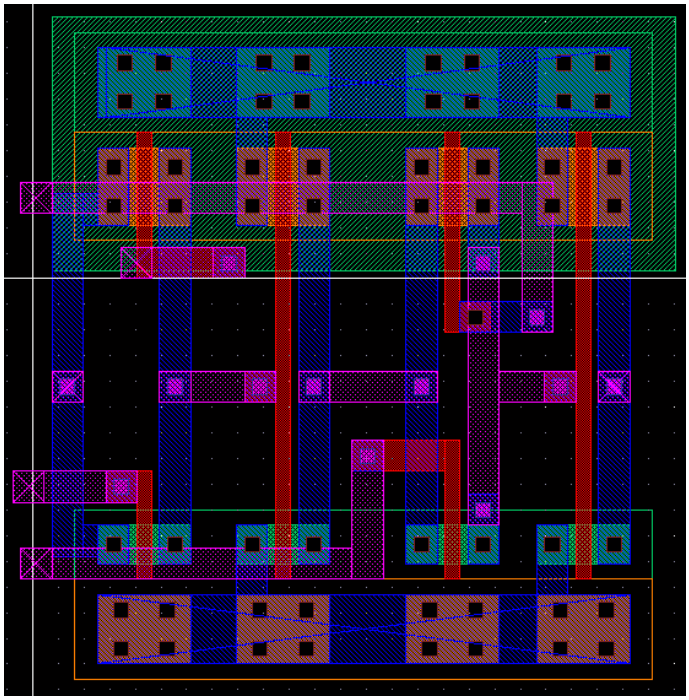


Fig A.9.2 layout of 1-bit Dynamic DFF

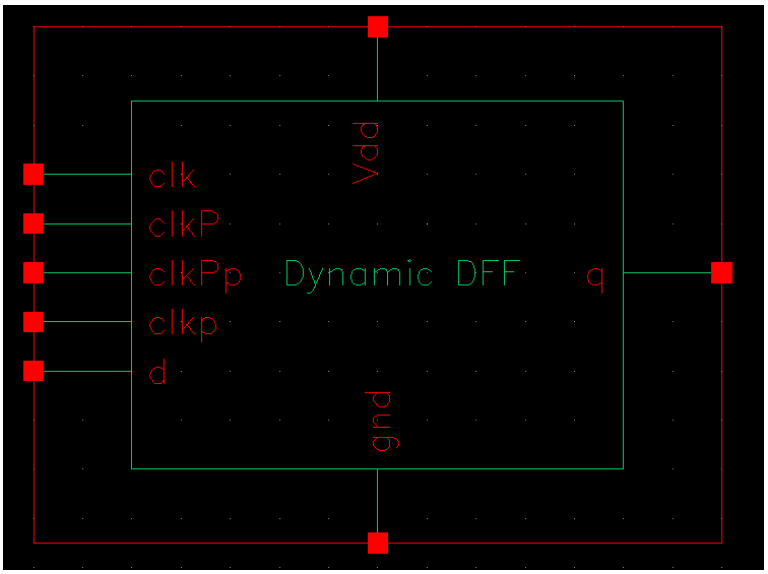


Fig A.9.3 symbol of 1-bit Dynamic DFF

b) PLS is shown in Fig A.9.4.

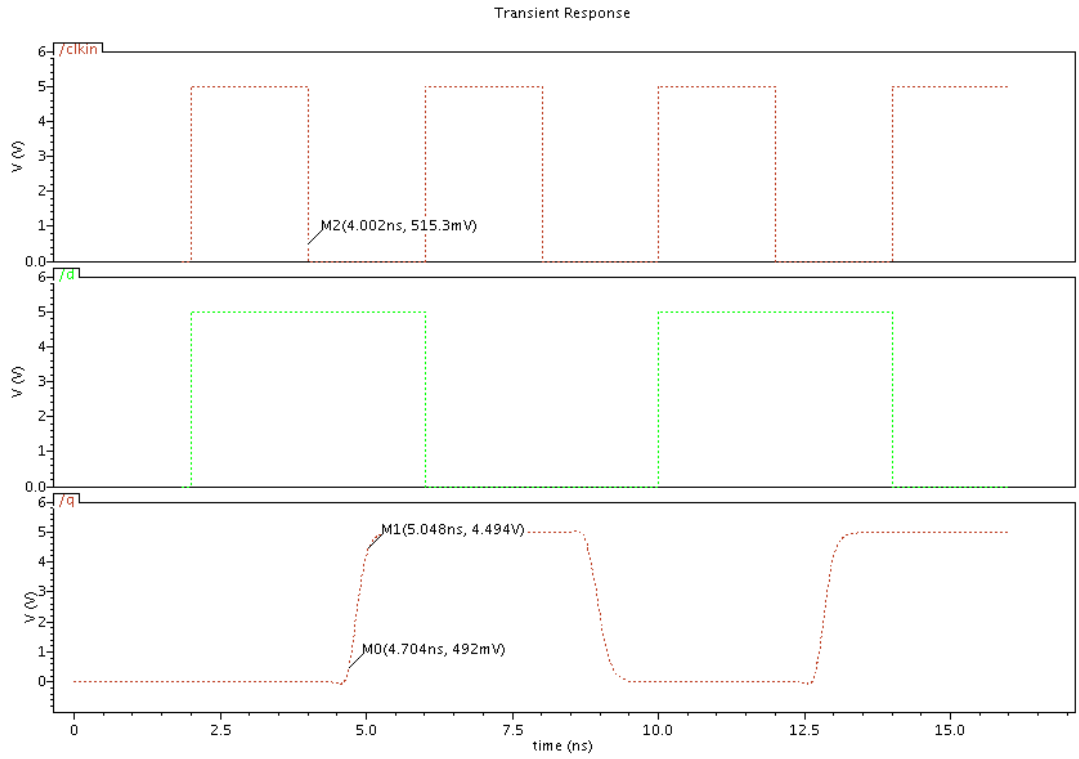


Fig A.9.4 PLS of 1-bit Dynamic DFF

10. 8-bit Dynamic DFF

a) Schematic, symbol and layout are shown in Fig A.10.1

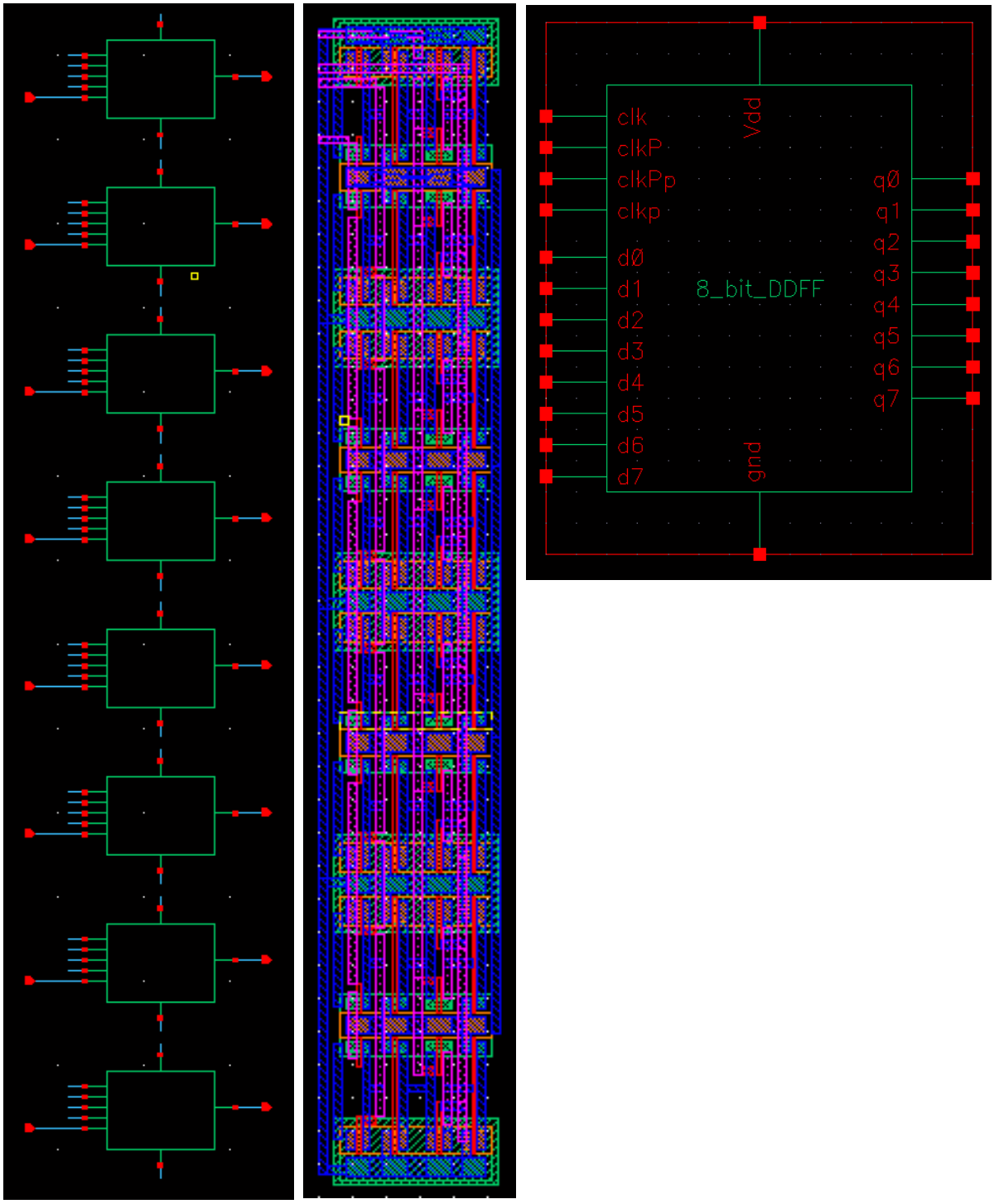


Fig A.10.1, Schematic, symbol and layout of 8-bit Dynamic DFF

b) Verilog code and simulation are shown in Fig A.10.2 and A.10.3.

```
//Verilog HDL for "ALU", "Dynamic_DFF_8bit" "functional"

`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module Dynamic_DFF_8bit ( q0, q1, q2, q3, q4, q5, q6, q7, Vdd, gnd, clk, clkP,
    clkPp, clkp, d0, d1, d2, d3, d4, d5, d6, d7 );

  output q0, q1, q2, q3, q4, q5, q6, q7;
  input d0, d1, d2, d3, d4, d5, d6, d7;
  inout gnd;
  inout Vdd;
  input clk, clkp, clkP, clkPp;

   Dynamic_DFF n0 ( q0, Vdd, gnd, clk, clkP, clkPp, clkp, d0);
   Dynamic_DFF n1 ( q1, Vdd, gnd, clk, clkP, clkPp, clkp, d1);
   Dynamic_DFF n2 ( q2, Vdd, gnd, clk, clkP, clkPp, clkp, d2);
   Dynamic_DFF n3 ( q3, Vdd, gnd, clk, clkP, clkPp, clkp, d3);
   Dynamic_DFF n4 ( q4, Vdd, gnd, clk, clkP, clkPp, clkp, d4);
   Dynamic_DFF n5 ( q5, Vdd, gnd, clk, clkP, clkPp, clkp, d5);
   Dynamic_DFF n6 ( q6, Vdd, gnd, clk, clkP, clkPp, clkp, d6);
   Dynamic_DFF n7 ( q7, Vdd, gnd, clk, clkP, clkPp, clkp, d7);

endmodule
`endcelldefine
```

Fig A.10.2, Verilog code of 8-bit Dynamic DFF

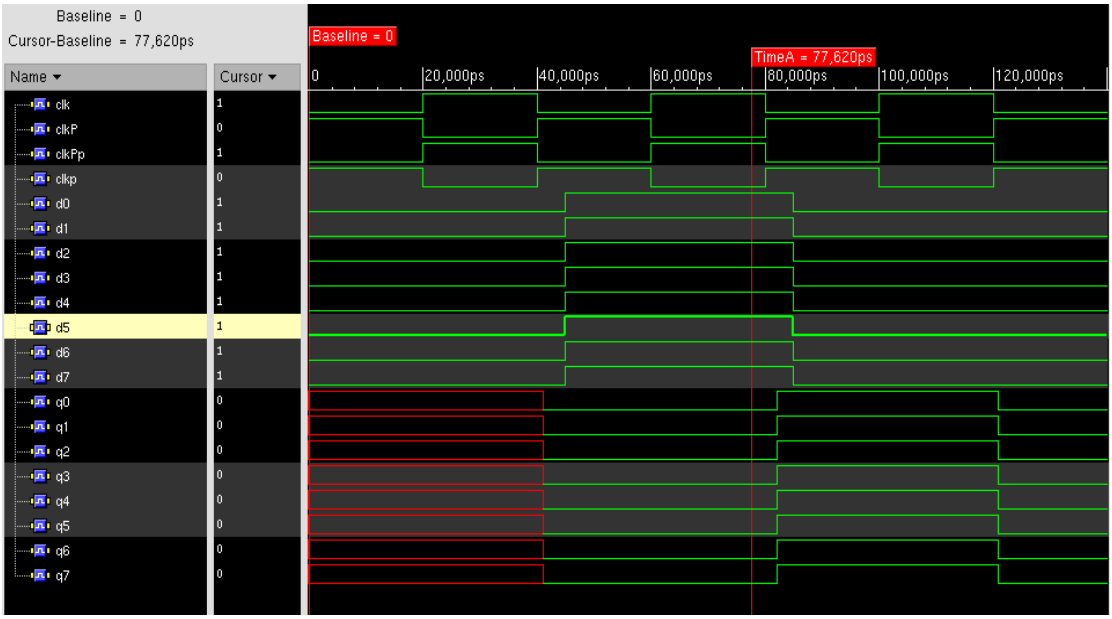


Fig A.10.2, Verilog simulation of 8-bit Dynamic DFF

11. 2-phase clock

a) Schematic, symbol and layout are shown in Fig A.11.1, Fig 11.2 and Fig 11.3

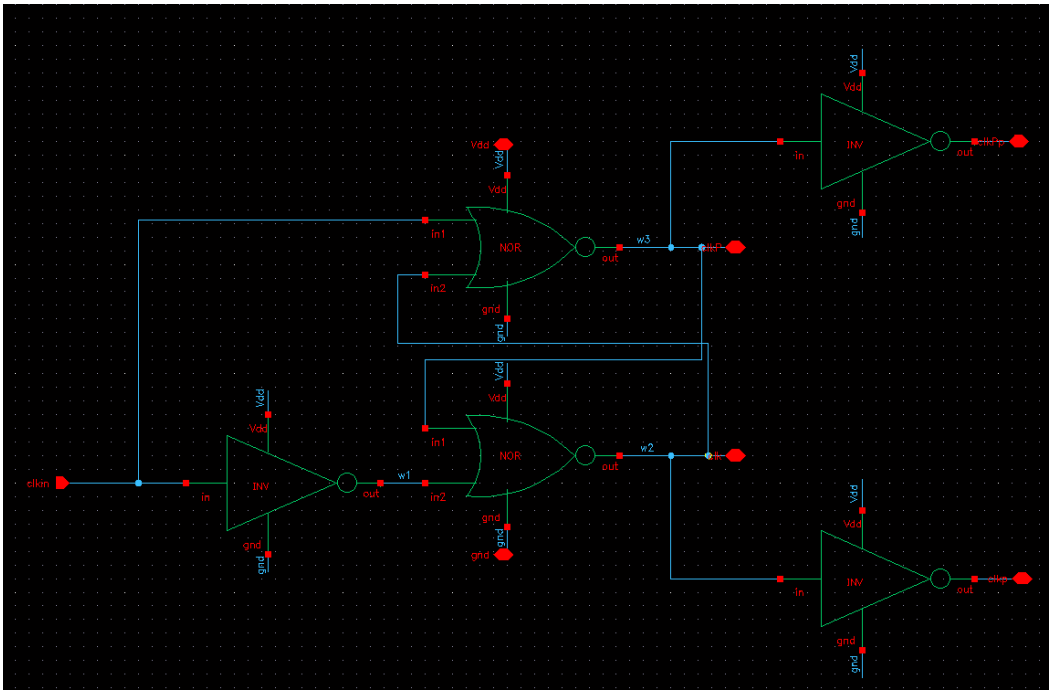


Fig A.11.1, Schematic of 2-phase clock

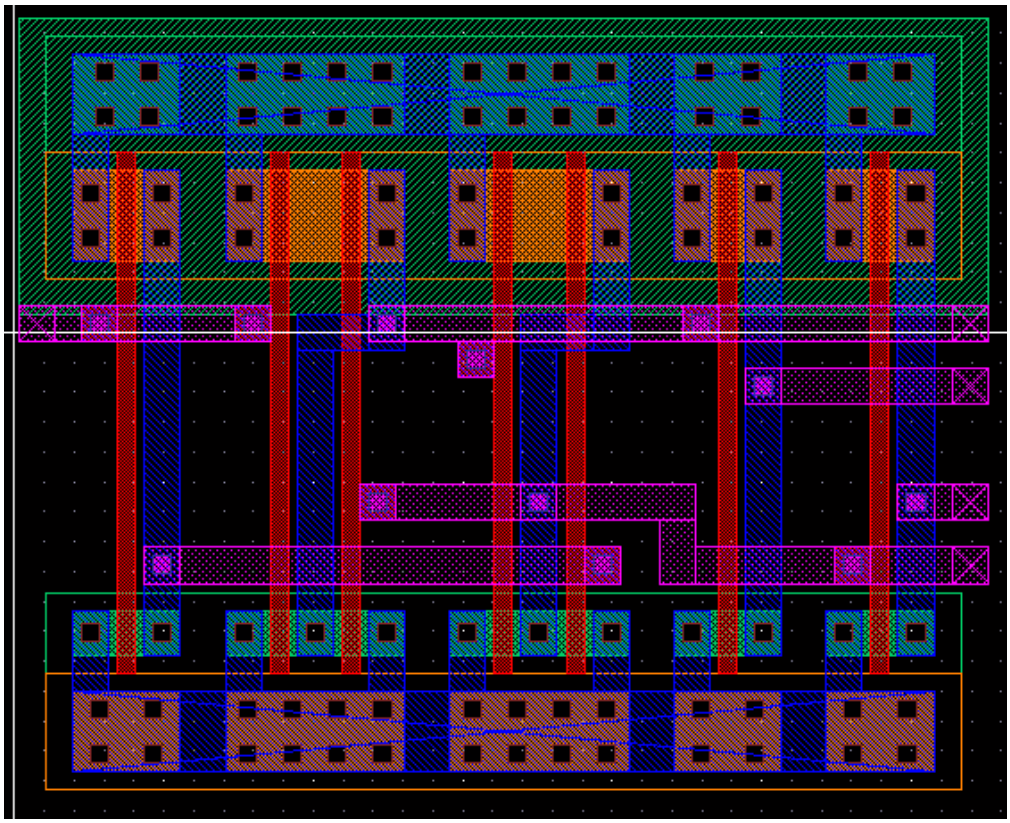


Fig A.11.2 layout of 2-phase clock

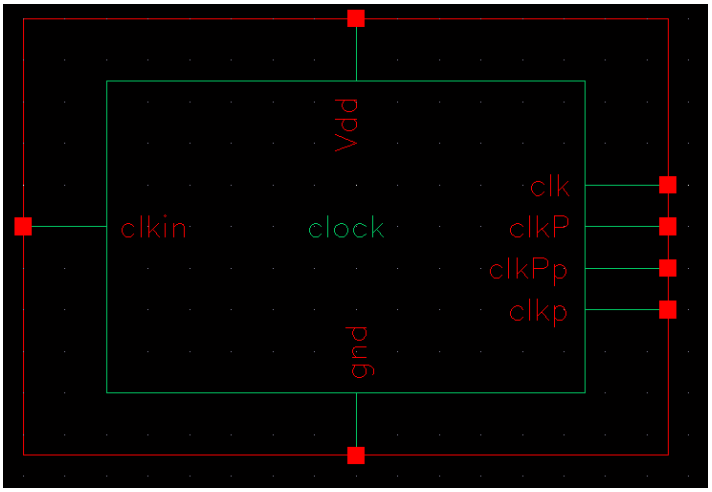


Fig A.11.3 Symbol of 2-phase clock

b) Verilog code and simulation are shown in Fig 11.4 and Fig 11.5

```
//Verilog HDL for "ALU", "2_phase_clk" "functional"
`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module clk  ( Vdd, clk, clkP, clkPp, clkp, gnd, clkin );

  inout gnd;
  inout Vdd;
  inout clk;
  input clkin;
  inout clkp;
  inout clkPp;
  inout clkP;
   wire w1;


   not n0 (w1, clkin);
   nor n1 (clk, clkP, w1);
   nor n2 (clkP, clk, clkin);
   not n3 (clkPp, clkP);
   not n4 (clkp, clk);


endmodule
`endcelldefine
```

Fig A.11.4 Verilog code of 2-phase clock

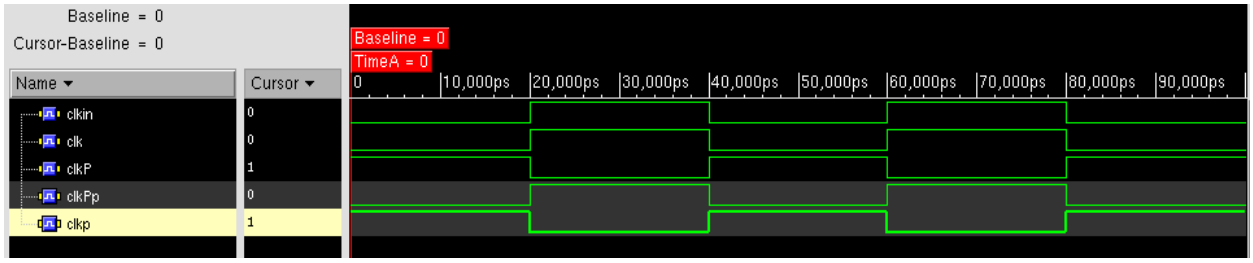


Fig A.11.5 Verilog simulation of 2-phase clock

12. 8-bit Inverter

a) Schematic, symbol and layout are shown in Fig A.12.1

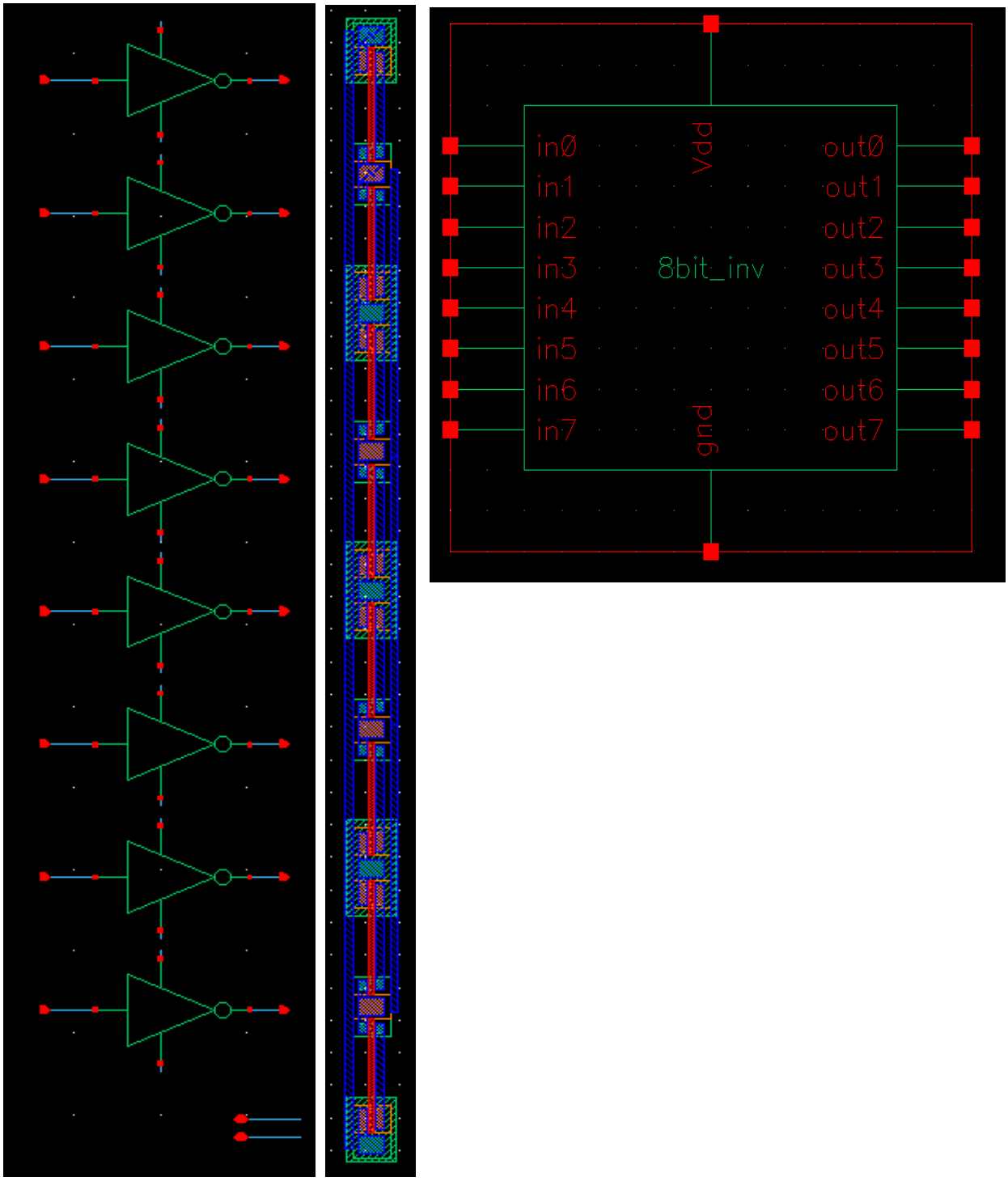


Fig A.12.1, Schematic, symbol and layout of 8-bit Inverter

b) Verilog code and simulation are shown in Fig A.12.2 and Fig A.12.3.

```
//Verilog HDL for "ALU", "inv_8bit" "funvtional"

`resetall
`celldefine
`delay_mode_path
`timescale 1ns/10ps

module inv_8bit ( out0, out1, out2, out3, out4, out5, out6, out7, Vdd, gnd,
    in0, in1, in2, in3, in4, in5, in6, in7 );

  input in0, in1, in2, in3, in4, in5, in6, in7;
  inout gnd;
  inout Vdd;
  output out0, out1, out2, out3, out4, out5, out6, out7;

   not (out0, in0);
   not (out1, in1);
   not (out2, in2);
   not (out3, in3);
   not (out4, in4);
   not (out5, in5);
   not (out6, in6);
   not (out7, in7);

endmodule
`endcelldefine
```

Fig A.12.2, Verilog code of 8-bit Inverter

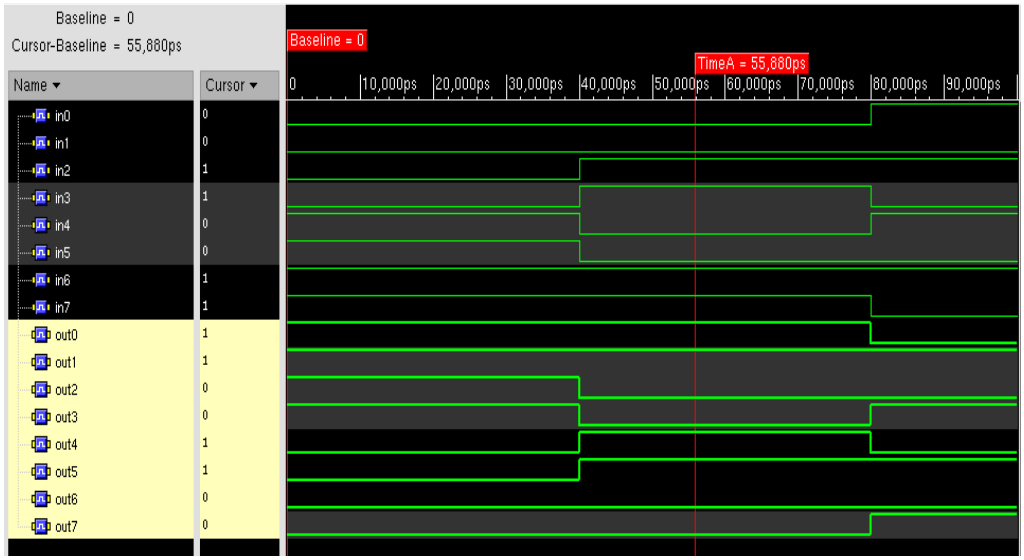


Fig A.12.2, Verilog simulation of 8-bit Inverter