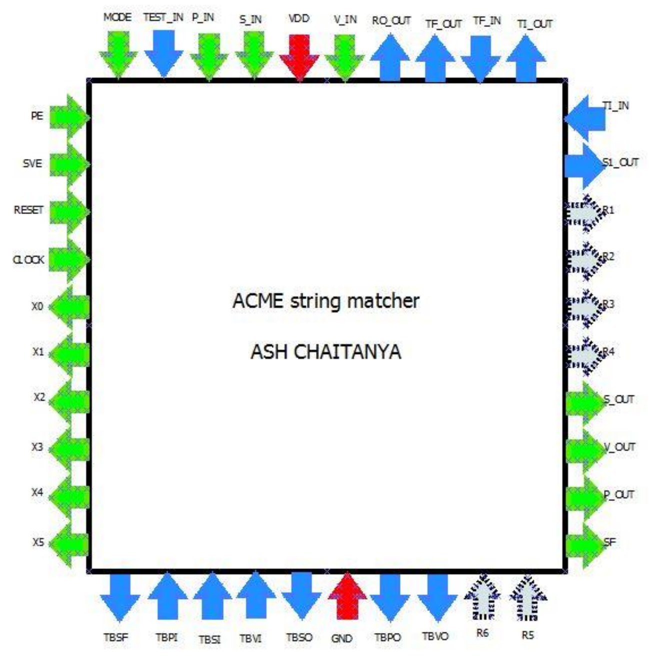


ACME STRING MATCHER

By Sai Chaitanya and Asrith Reddy

ABSTRACT

A chip which compares a substring with the mother string and produces the location of the match if match is success.

INTRODUCTION TO VLSI DESIGN- SEMESTER PROJECT 2015

Project: String Matching

Team

Asrith Reddy Singa Reddy

M08893106
singaray@mail.uc.edu
Ph: +15133060706

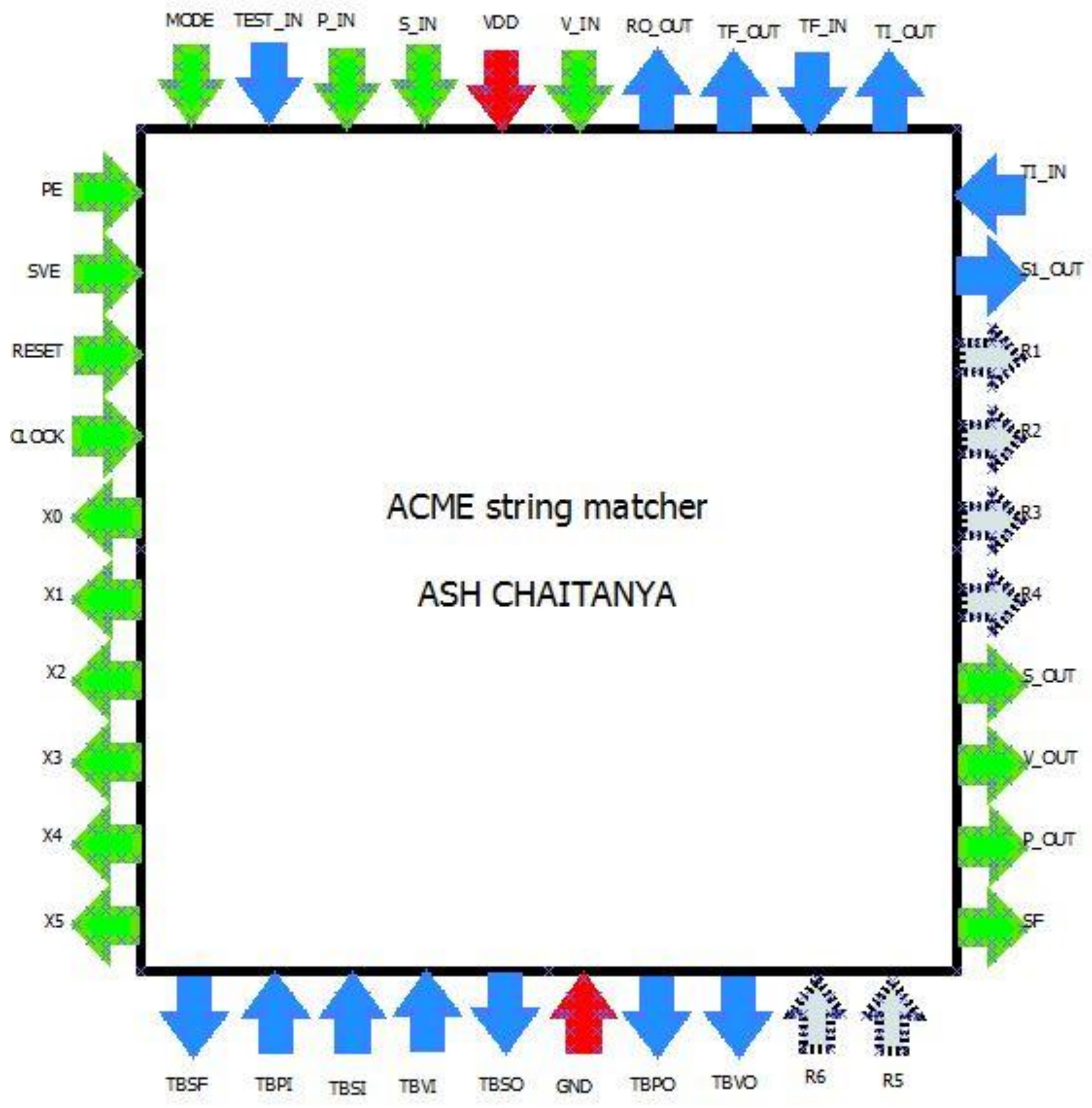
Sai Chaitanya Nandipati
Coordinator
M08905912
nadipsa@mail.uc.edu
Ph: +15133069558

Chip name	:	**ACME**
Chip process	:	AMI C5 0.5 n-well CMOS
Design Rules	:	MOSIS SCMOS SUBM
		λ= 0.3 microns (feature size 0.6)
Chip Dimensions	:	1.5 mm x 1.5 mm (5000λ x 5000λ)
No of pins	:	40
No of bitslices	:	54

Path to CIF file: nandipsa/project/padframefinal/chipfile2.cif

CONTENTS

Pin Out Diagram for the Chip



GREEN- NORMAL PINS
BLUE- TEST PINS

User guide

Pin description

PIN NO.	PIN NOTATION	PIN DESCRIPTION
1	MODE	Test/Normal
2	TEST_IN	Test Input
3	P_IN	Main String Input
4	S_IN	Sub String Input
5	VDD	Source Voltage
6	V_IN	Valid String Input
7	RO_OUT	Ring Oscillator Output
8	TF_OUT	Test Flip flop Input
9	TF_IN	Test Flip Flop Output
10	TI_OUT	Test Inverter Input
11	TI_IN	Test Inverter Input
12	S1_OUT	First Row Substring Output
13	R1	Unused
14	R2	Unused
15	R3	Unused
16	R4	Unused
17	S_OUT	Sub String Output
18	V_OUT	Valid Output
19	P_OUT	Main String Output
20	SF	Success/Fail
21	R5	Unused
22	R6	Unused
23	TBVO	Test Bit slice Valid Out
24	TBPO	Test Bit slice Main Out
25	GND	GROUND
26	TBSO	Test Bit slice Substring OUT
27	TBVI	Test Bit slice Valid IN
28	TBSI	Test Bit slice Substring IN
29	TBMI	Test bit slice Main IN
30	TBSF	Test bit slice Success Fail
31	X5	INDEX 5
32	X4	INDEX 4
33	X3	INDEX 3
34	X2	INDEX 2
35	X1	INDEX 1
36	X0	INDEX 0
37	CLOCK	CLOCK INPUT

38	RESET	RESET
39	SVE	SUBSTRING VALID ENABLE
40	PE	MAIN STRING ENABLE

Working of the chip

Function of the chip:

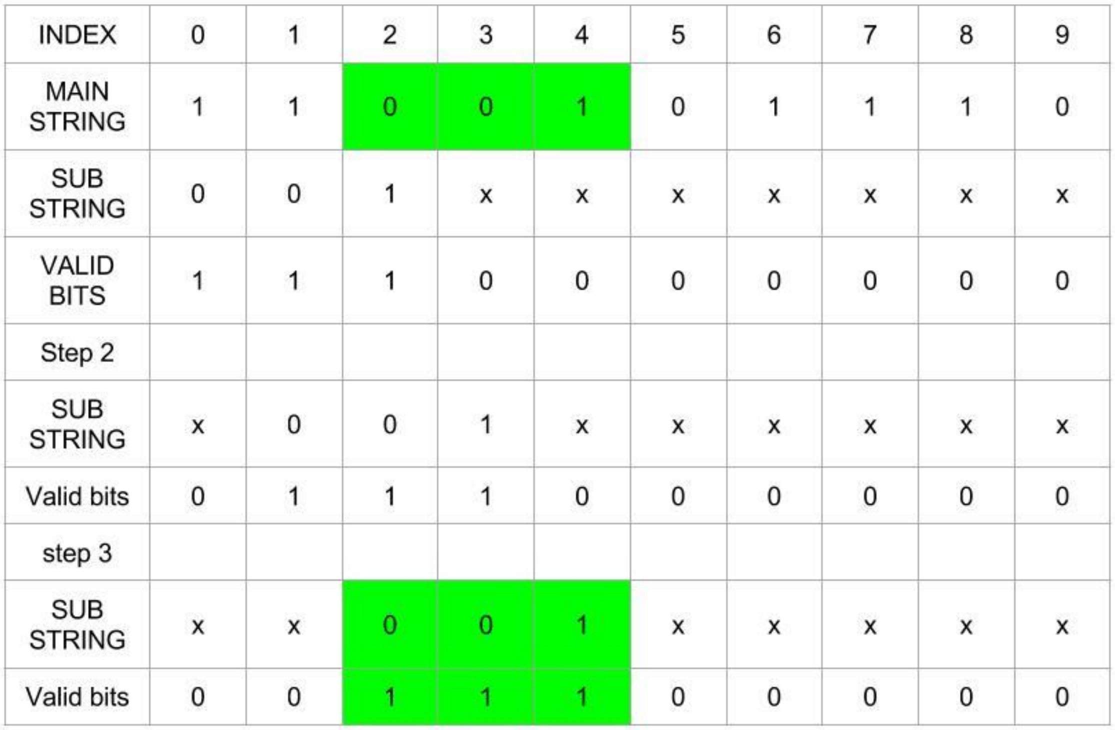
- The chip is a hardware implementation of a string matcher.

Modes of operation:

1) The Chip has two modes of functioning- Normal and Test mode.
    a. Normal mode is when the chip functions as a string matcher
    b. Test mode is when the chip can be tested by enabling all the flip flops to form a scan chain
2) *Normal Mode usage*:
    a. Set the MODE pin to 1 which makes it function in the normal mode.
    b. The entire chip is RESET once before start of the operation. Then the reset is set to 1. Remember this is a negative reset. The reset pin stays high as long as the operation continues.
    c. The main string enable (PE) is made high. Once the main string enable is high, main string is loaded serially over N clock cycles.
    d. In the next clock cycle, PE is turned OFF and SVE (substring enable) is turned ON.
    e. Substring and valid bits (which are 1's) are loaded serially over next L clock cycles. After loading is completed SVE is turned OFF.
    f. Number of valid bits indicate the length of sub-string.
    g. After successfully loading of main string, substring and valid bits string comparison starts.
    h. Success/fail pin is asserted based on the result of comparison.
    Success/fail=1 then it's a match (given sub-string is present in the main string).
    Success/fail=0 then it's a fail (given sub-string is not present in the main string).
    i. Based on the success/fail result counter is incremented and match position is calculated by counter output which is output index.

Example:

MISMATCH -> Shift the substring and valid bits



INDEX	0	1	2	3	4	5	6	7	8	9
MAIN STRING	1	1	0	0	1	0	1	1	1	0
SUB STRING	0	0	1	x	x	x	x	x	x	x
VALID BITS	1	1	1	0	0	0	0	0	0	0
Step 2										
SUB STRING	x	0	0	1	x	x	x	x	x	x
Valid bits	0	1	1	1	0	0	0	0	0	0
step 3										
SUB STRING	x	x	0	0	1	x	x	x	x	x
Valid bits	0	0	1	1	1	0	0	0	0	0

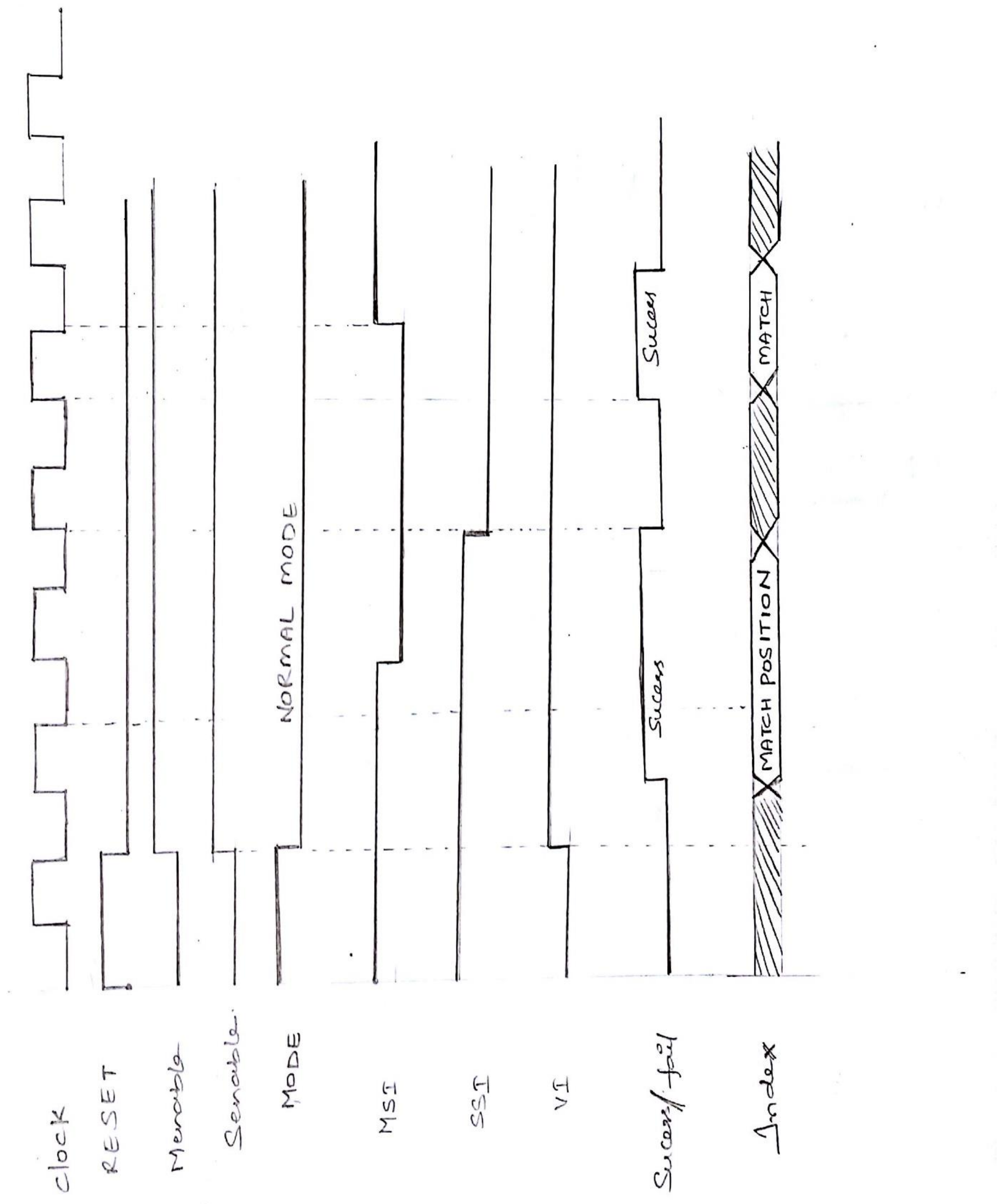
Index match at location 2.

3) *Test mode usage:*
   a. Set the MODE pin to 0 which makes the chip to function in the test mode.
   b. In the test mode all the flip flops are connected such that they form a serial in serial out shift register (scan chain).
   c. The test mode can be verified by sending in the data through test-input (TI) and observing test outputs on the chip.
   d. In the test mode PE (main string enable) is set to ‘1’ and SVE (sub-string enable) is automatically set to ‘1’ based on the MODE input.
   e. Data sent serially through the test input is received through test output pins after passing through all the flip flops in the scan chain.
   f. If the data is received successfully then all the flip flops in the design are working perfectly.
   g. Separate test inverter, D-flip flop and bit-slice are connected in the pad frame, which are accessible through pins on chip. They can be tested separately for basic functionality.

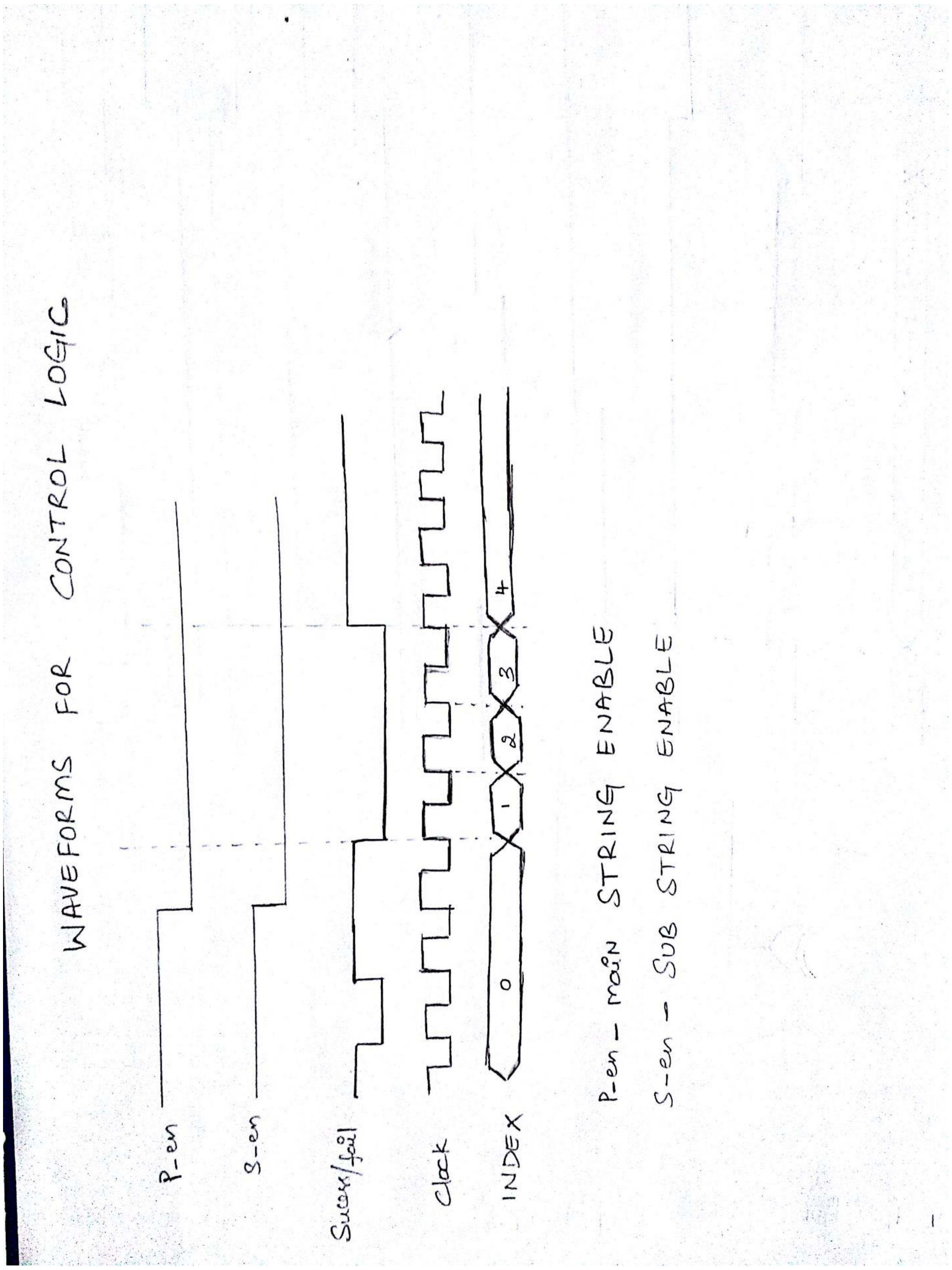
Assumptions:

- PE and SVE are user controlled.
- Number of 1’s in the valid string defines the length of the sub-string.

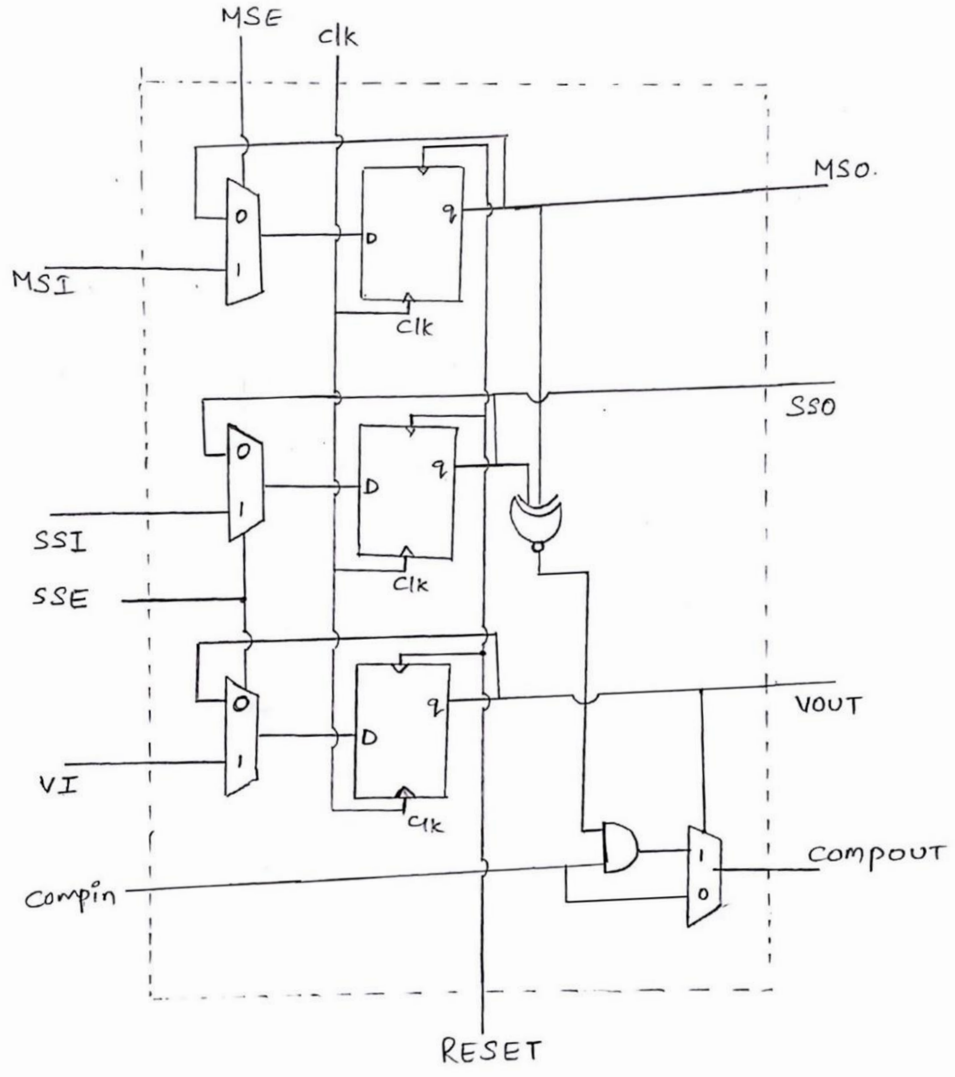
User level timing diagrams:



Counter timing diagrams:



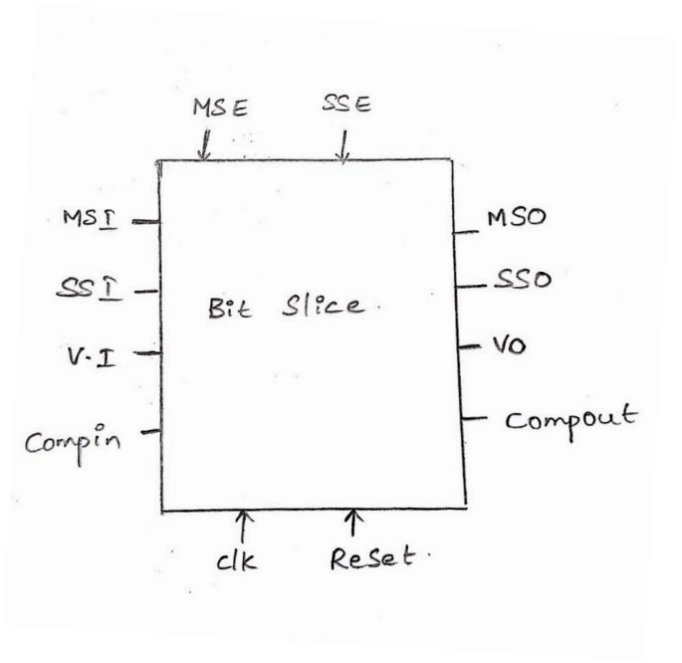
Bit slice:



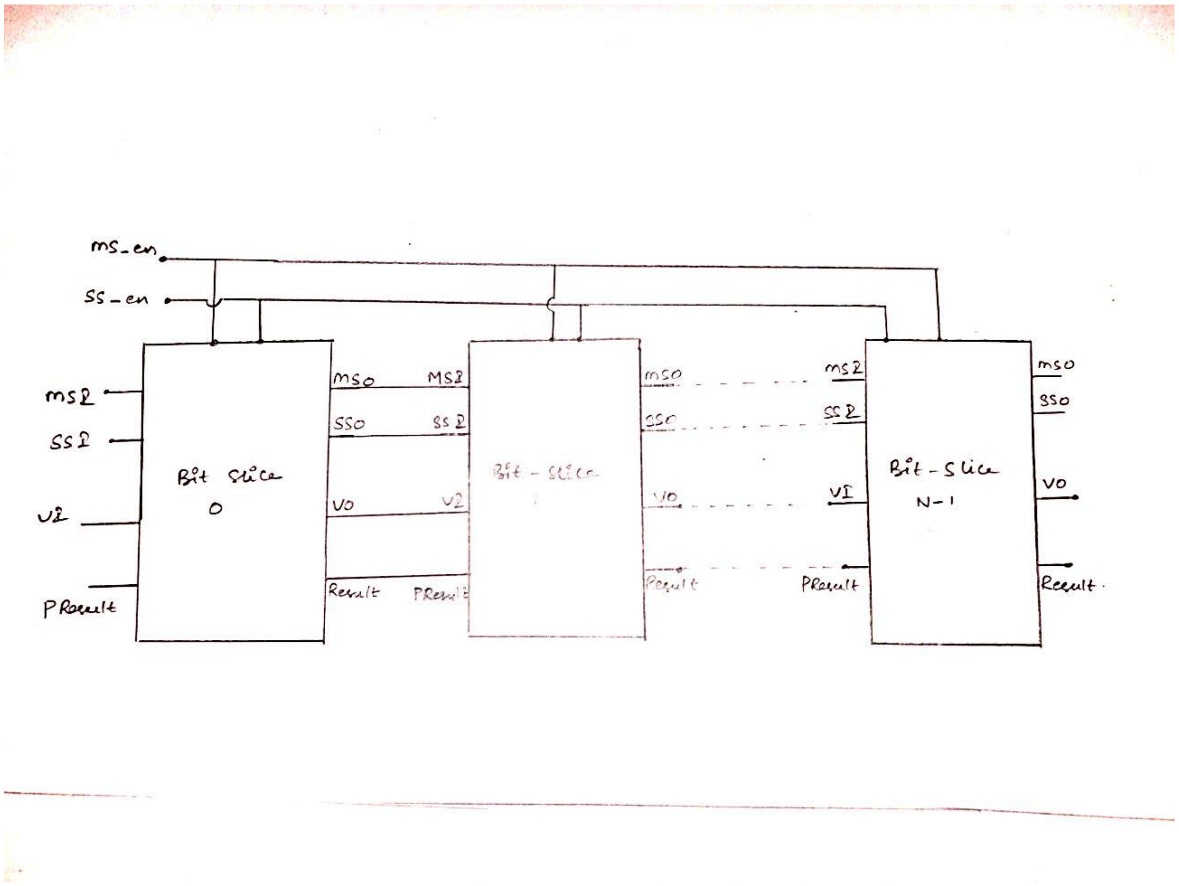
MSE – main String enable
SSE – Sub-String enable.
MSI – main String input
SSI – Sub String input.
VI – Valid input.
Compin – previous Stage Comparator input.
Compout – Comparision Output.

MSO – main String Output
SSO – Sub String Output.
VOUT – Valid output
clk – clock
Reset – Reset.

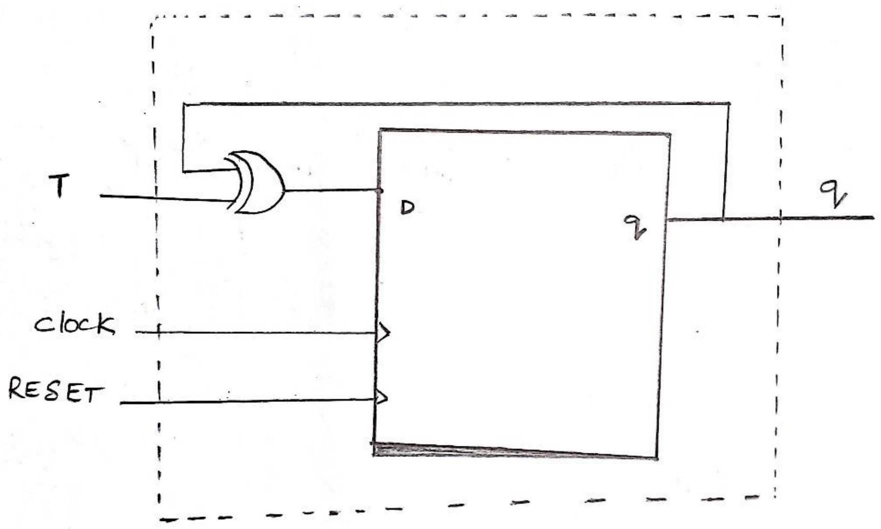
Bit slice pin out diagram:



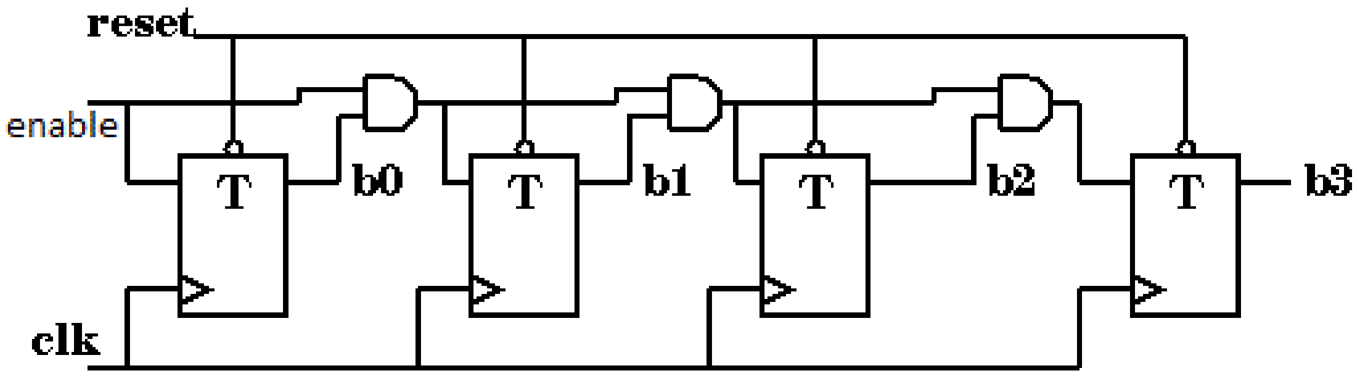
Bit slice connection:



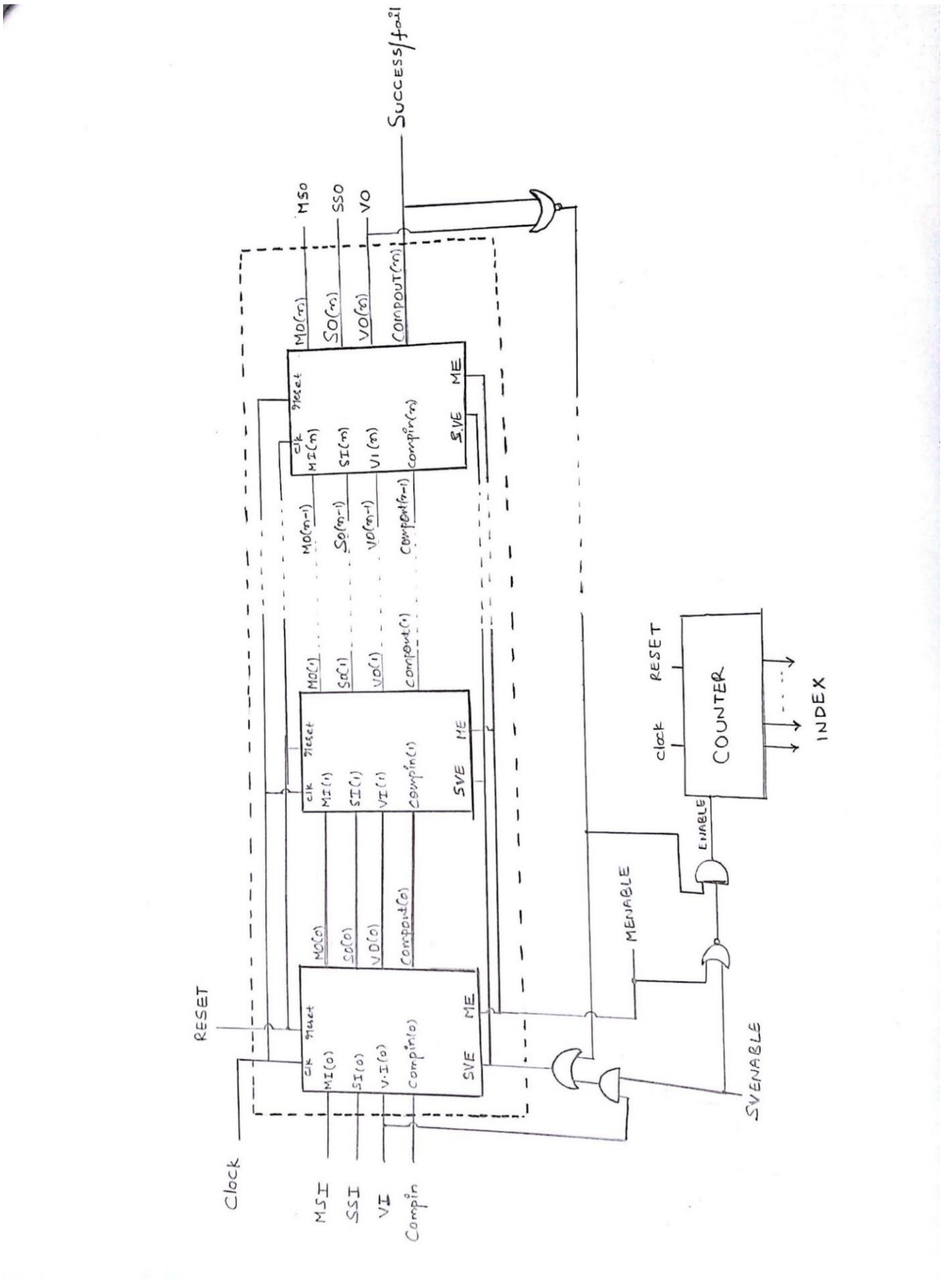
T flip flop from d flip flop:



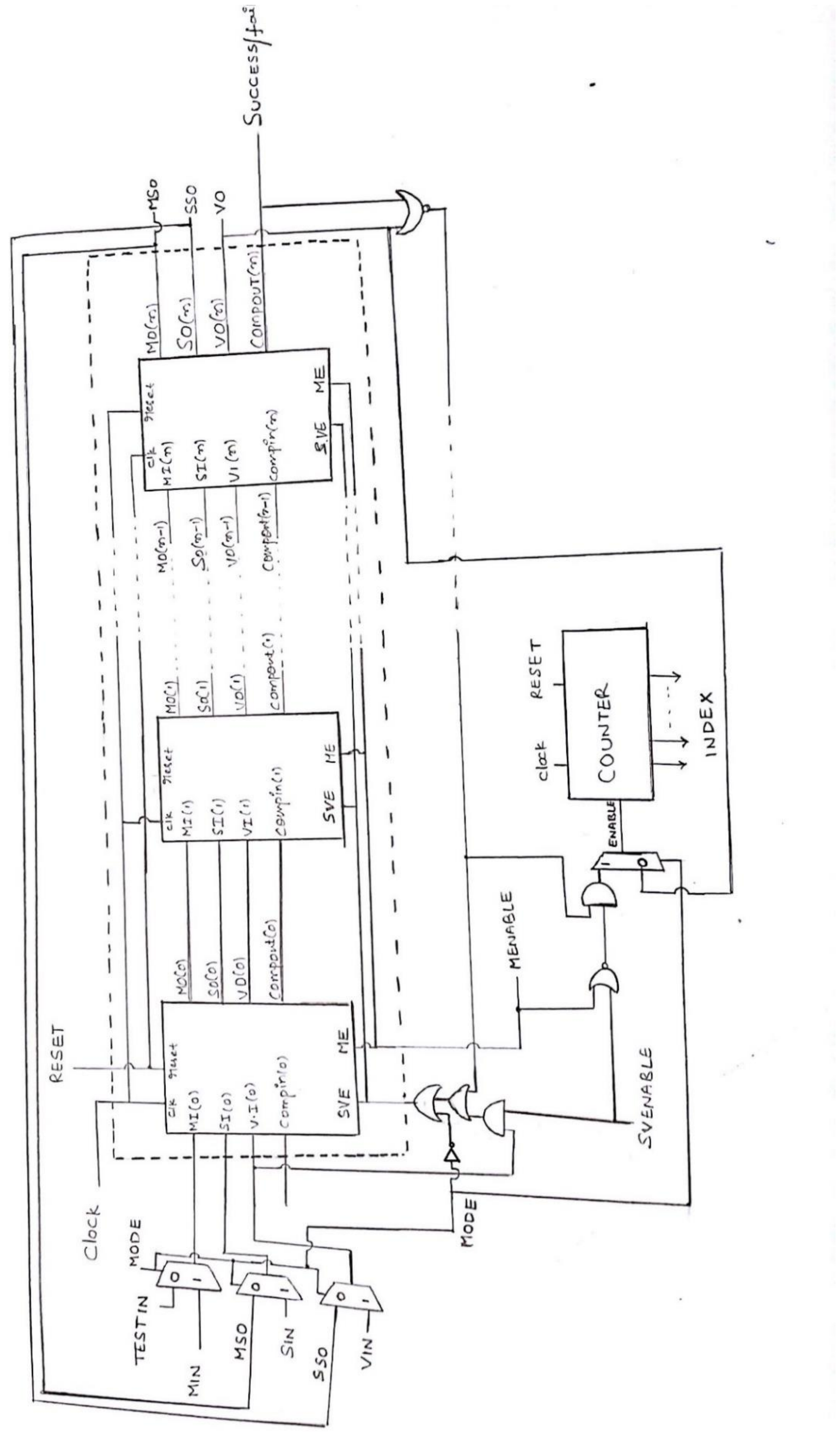
Up counter:



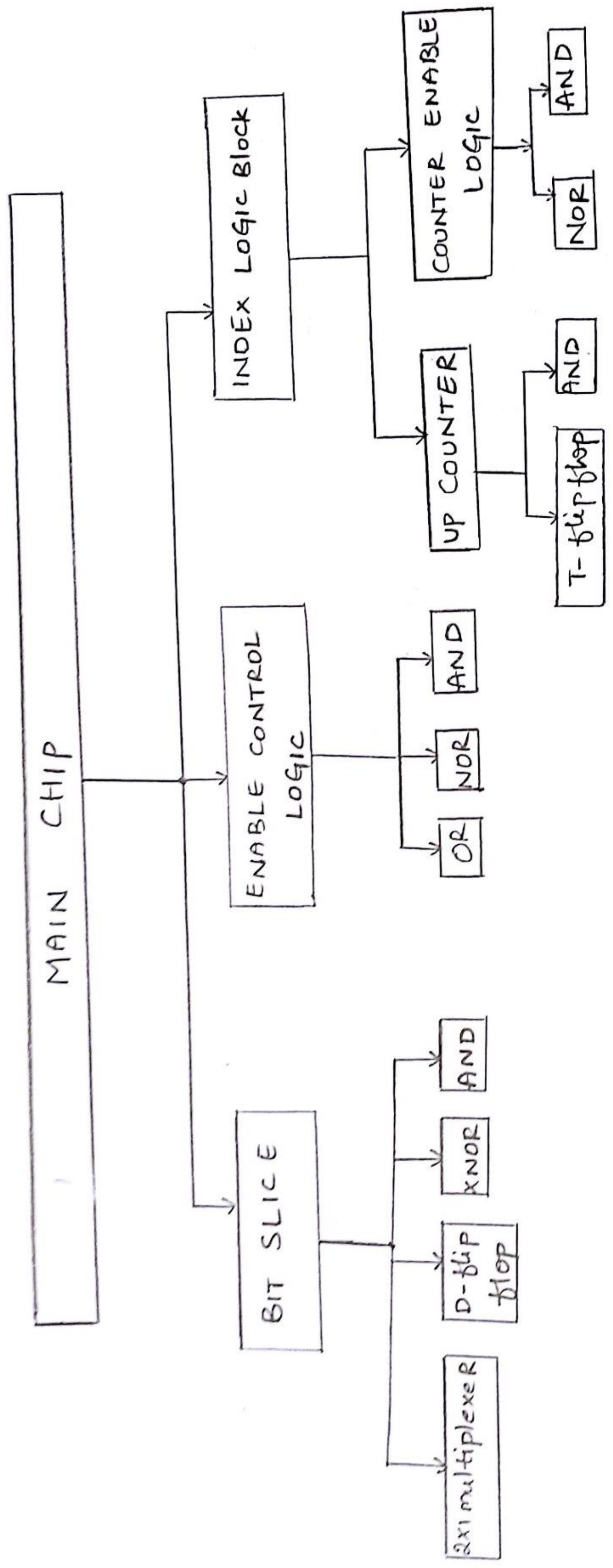
Final n-bit slice:



Final N-bit slice with test logic:



Top level hierarchy:



Basic gates used in the design:

Or, and, not, Xor, Xnor, Nor

Flip flops: D-flip flop and T- flip flop (designed using D-flip flop)

Counter: synchronous up counter

Multiplexer 2 to 1

TEST MODE AND STRATEGY

Scan chain

- The chip can be used in test mode (mode 0) apart from the normal mode (mode 1). In the test mode the all the flip flops in the chip form a shift register there by providing a scan chain.
- In test mode PE is set to ‘1’.
- The test engineer can access the test mode by setting the MODE pin to 0.
- The Reset pin when low resets the flip flops with output data 0. When high, the flip flops continue to operate in the normal mode.
- In the test mode, the data from the P_OUT of the final bit slice is fed as input to the S_IN of the first bit slice.
- The S_OUT from the final bit slice is then fed as input to the V_IN of first bit slice. The V_OUT of the final bit slice which is the 54th bit slice follows the input from the P_IN.
- The exact replication of the test input after 54x3 (162) clock cycles at the output pin V_OUT shows that the flip flops are working correctly.

Pins used

MODE (pin 1)	0
TEST_IN (pin 2)	Test vector
P_OUT (pin 19)	Output follows test vector after 54 clock cycles
S_OUT (pin 17)	Output follows test vector after 108 clock cycles
V_OUT(pin 18)	Output follows test vector after 162 clock cycles

Counter

- During the test mode, there are other flip flops in the 6 bit counter which are not included in the scan chain. These flip flops as usual keep counting from 0 to 2^6-1 in the test mode based on the test input. The state of the counter can be tapped at the index pins X0 to X5, thus verifying the correctness of the flip flops present in the counter.

MODE (pin 1)	0
Counter outputs X0, X1,X2,X3,X4, X5	The outputs keep counting from 0 to 2^6 -1 and repeats from 0 again as long as the mode is set to 0

SINGLE BITSLICE

- There is a single bit slice in the chip which can be used independently from the entire chip.
- The bit slice inputs are TBPI, TBSI, TBVI which are the main string, substring and valid inputs and the outputs are the TBPO, TBSO, TBVO which are the respective outputs.

TBPI pin 29	Main string input
TBSI pin 28	Sub string input
TBVI pin 27	Valid Input
TBPO pin 24	Main string output
TBSO pin 26	Substring output
TBVO pin 23	Valid output
TBSF pin 30	Success or fail output

- The output is observed at the success or fail pin (pin 30).

TEST INVERTER

- The test inverter is placed with the input at the pin no 11
- The output is placed at the pin no 10.

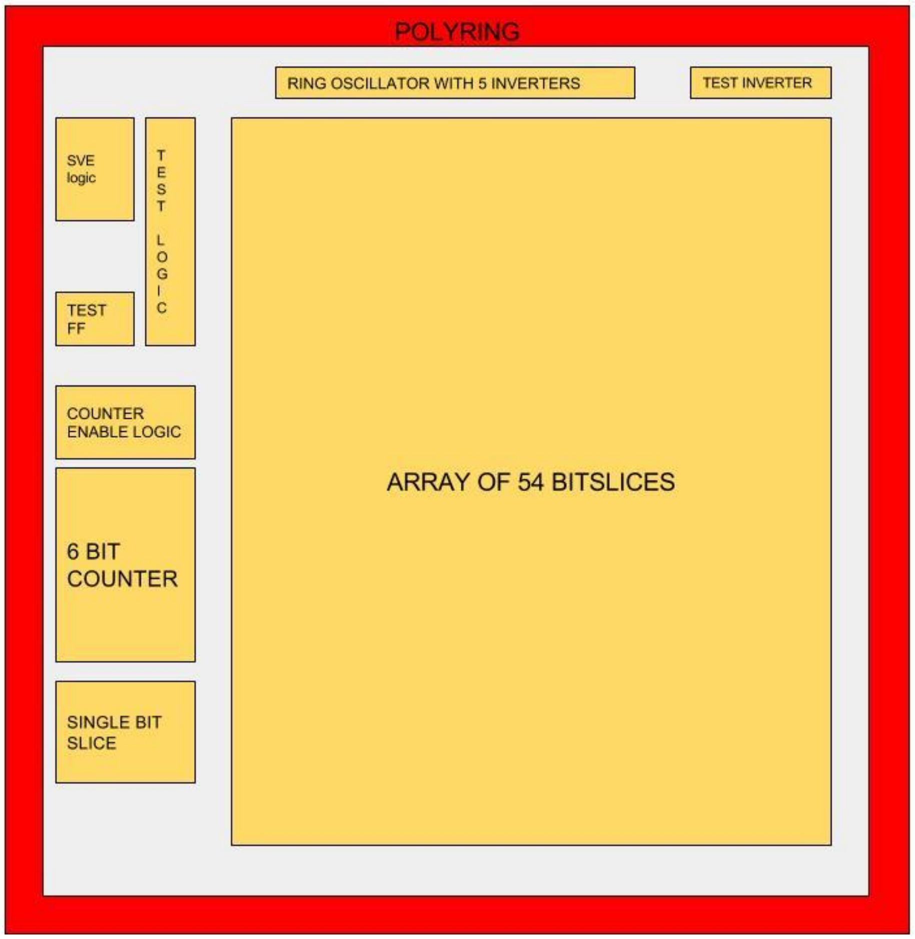
TEST FLIP FLOP

- The Test flip flop is a positive edge triggered D flip flop with the input at the pin no. 9
- The flip flop data output is given at the pin no. 8.

Ring Oscillator

- The ring oscillator with five inverter gates is given for testing at the pins with the output pin at pin no 7.

Architecture:



Main architecture consists of 9 segments

1. Array of 54 bit slices
2. Logic for sub string and valid bits enable
3. 6 bit counter
4. Test logic
5. Enable logic for the counter
6. Test bit slice
7. Test flip flop
8. Test inverter
9. Test ring oscillator

- Sub string and valid bits should be shifted only when there is a mismatch so to make this possible sub string and valid bits enable logic is used.
- Test logic is used to make all the flip flops in the chip to form a scan chain
- Counter is used to find out match location index.
- Counter should start functioning only after main string enable and sub string, valid bit enable is turned OFF. So, counter enable logic is used to make this possible.
- Separate bit slice, Inverter, D-flip flop, Ring oscillator are included in the architecture which can be accessible through external pins for testing purpose.

Major design alternatives and Decisions:

1. Having a sequential flow between bit slices. In this design output of one bit slices triggers the next slice and control logic simultaneously. If there is a match in the first bit slice then next bit slice is evaluated else control logic will be triggered to shift the sub-string.

   Disadvantages: Increase in time delay, low through put, complex control circuitry.

2. Having control logic completely outside the bit slices. In this design bit slice consists of two d-flip flops and a bit comparator (XNOR), and rest all control logic for shifting, counting is placed outside the bit slices.

   Disadvantages: wastage of area, routing complexity.

Final design decision: All the bit slices execute in parallel. If there is any mismatch using control logic sub string is shifted accordingly. Most of the control logic is included in the bit-slice itself. As most of the control circuitry is included in the bit slice available area is used effectively. By just placing slices adjacent to each other we can make automatic connections between them using array command in magic which simplifies routing.

Advantages: Effective usage of available area to maximize N, increase in through put compared to other design alternatives, routing problems are reduced.

Changes from first report to final report: In the start we thought of using counters to measure the length of main string and substring. In the later designs counters are removed to maximize the N and to make a simple design. In present design length of substring is measured using valid bits. Control logic is modified based on the requirement as we progressed in the project.

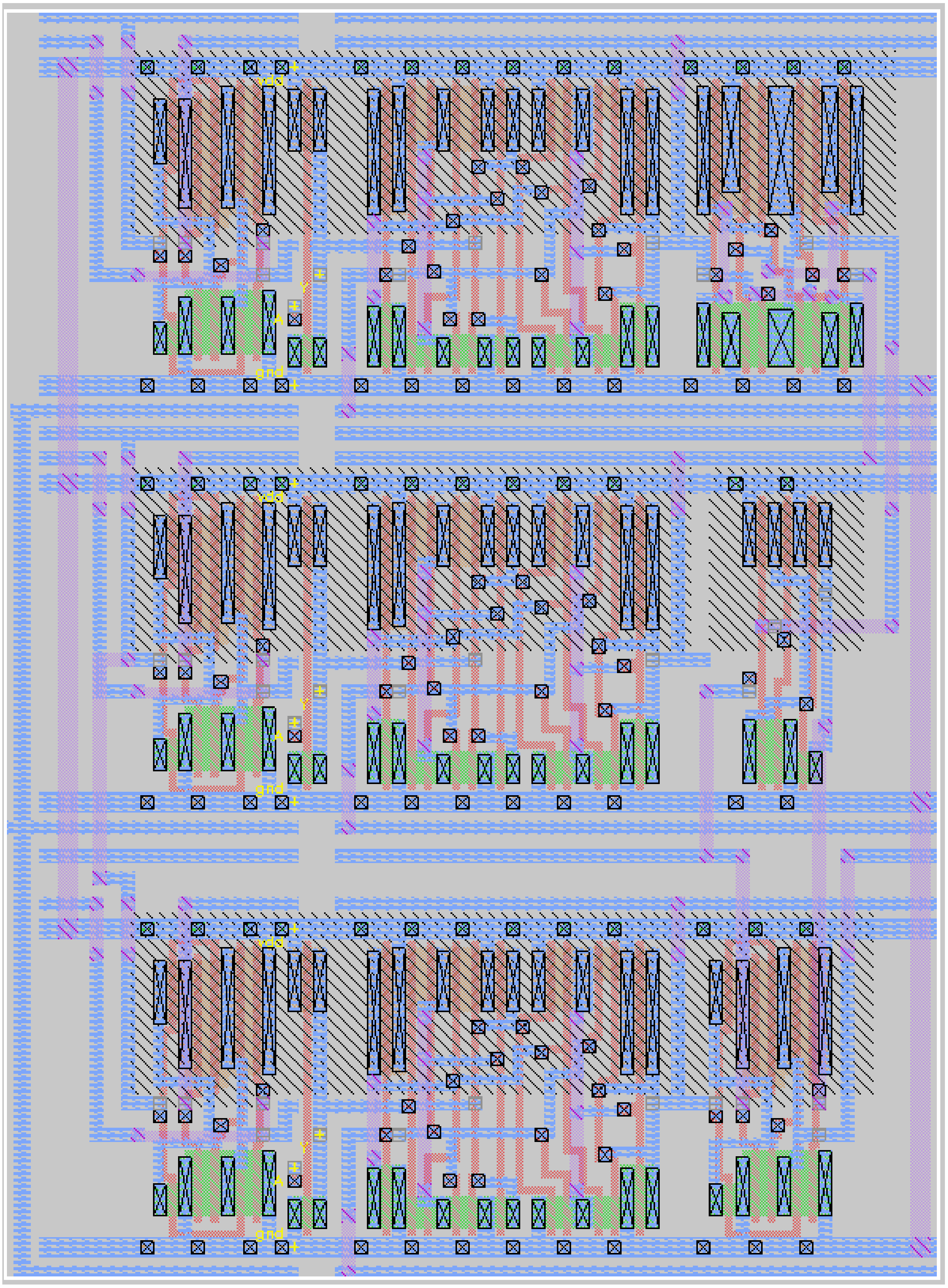
Design decisions in Magic:

Major considerations:

- Stacking bit-slices either vertically or horizontally results in wastage of area so we stacked bit slices in a mixed fashion (combination of horizontal and vertical stacking).
- The control logic is placed at the corners. Even though it increases routing complexity we chose this method to achieve primary goal of maximizing N.
- Array of bit-slices are flipped up-down to vertically stack two of horizontally stacked array of bit-slices.
- Array of bit-slices are flipped right-left to minimize inter sliced routing.
- Maximum N that we achieved is 54.

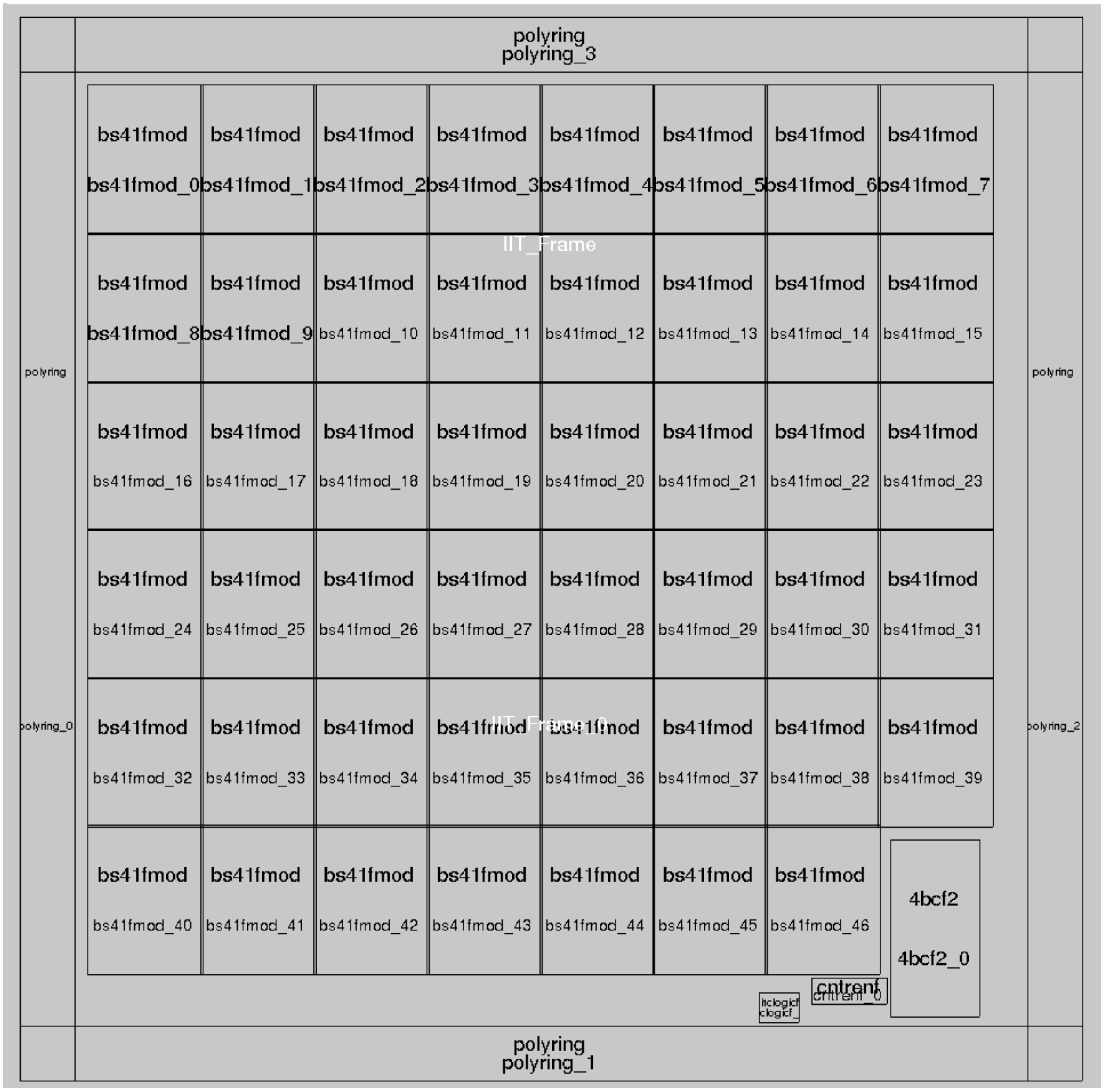
The bit slice design was modified to increase the N of the chip. With new design we have increased the value of N from 48 to 54.

OLD BIT SLICE LAYOUT



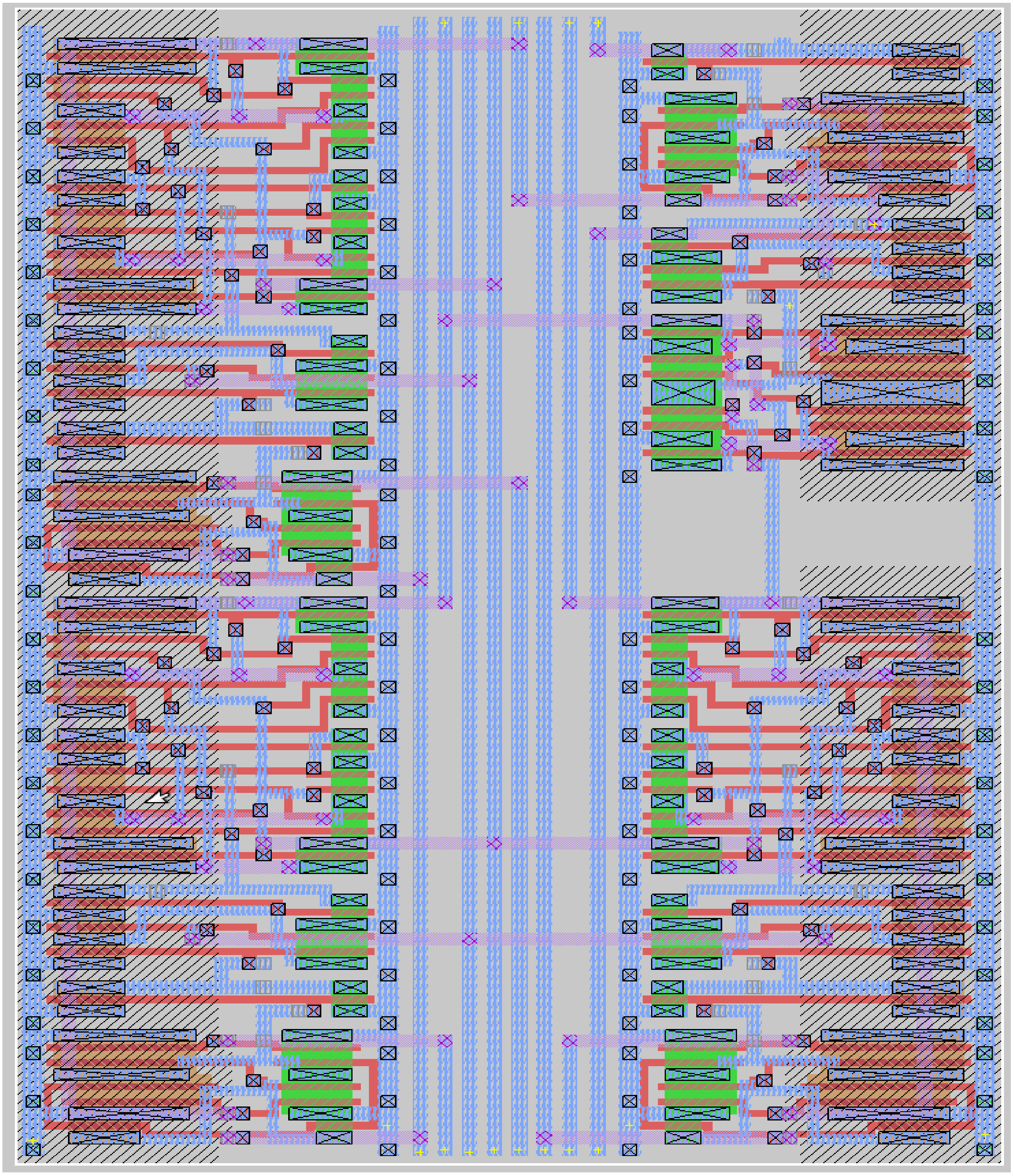
FLOORPLAN OF THE CHIP AND ROUTING:

The pads are eliminated from the diagram for clarity.



LAYOUTS IN MAGIC and SIMULATION RESULTS

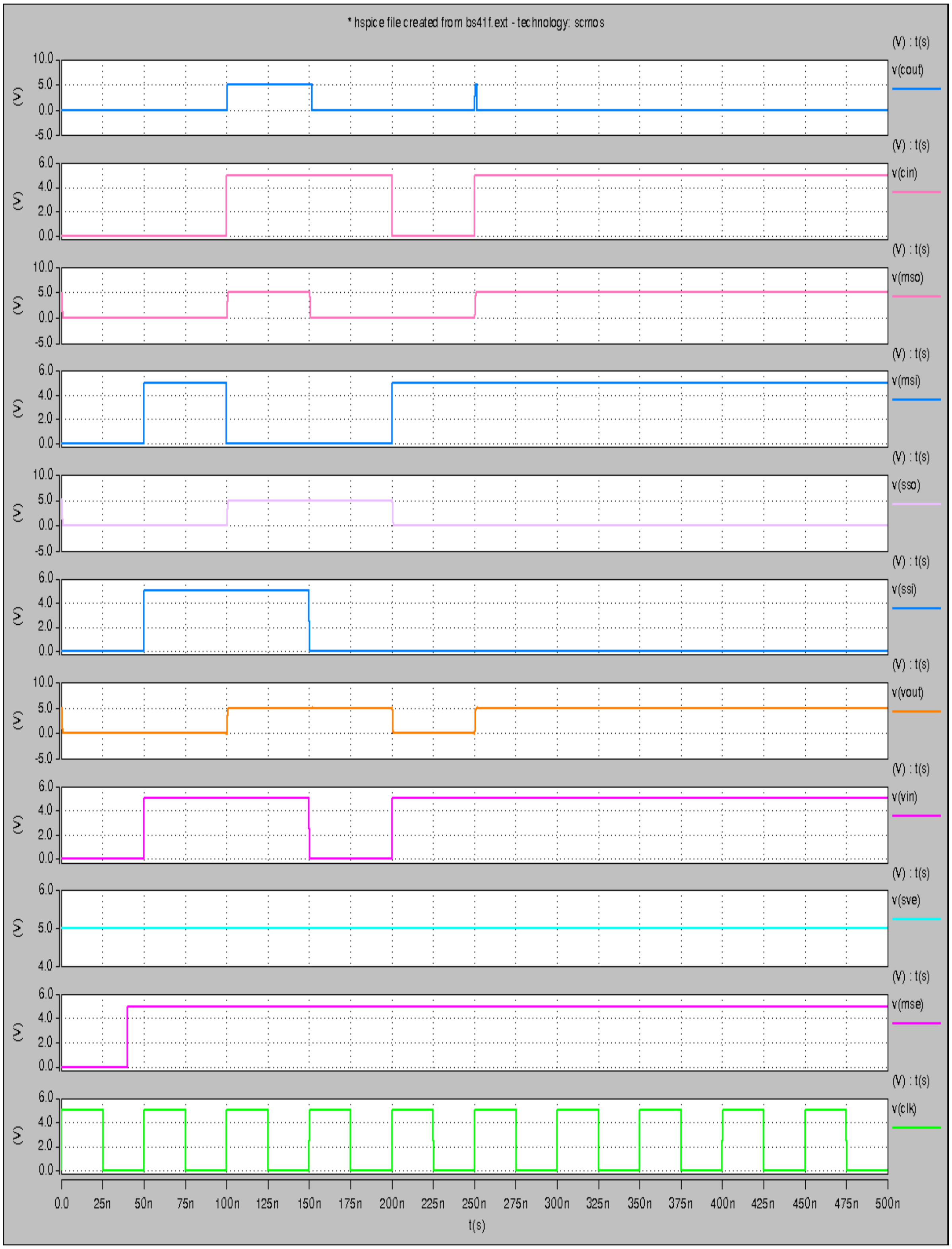
BIT SLICE



INPUTS: RESET, PIN, SIN, VIN, PEN, SEN, CIN, COUT, CLK

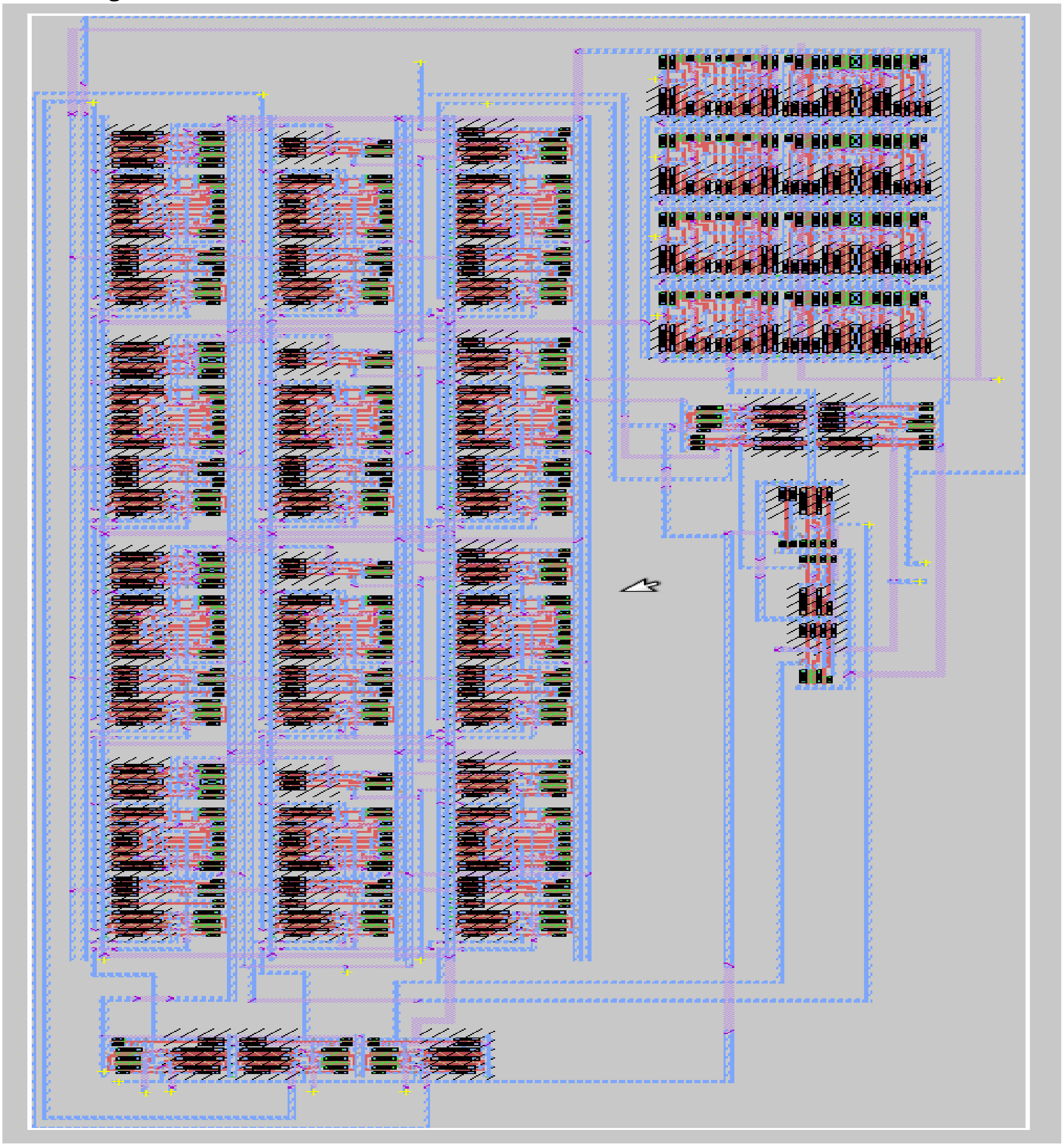
OUTPUTS: POUT, SOUT, VOUT, COUT

SPICE WAVE FORMS:

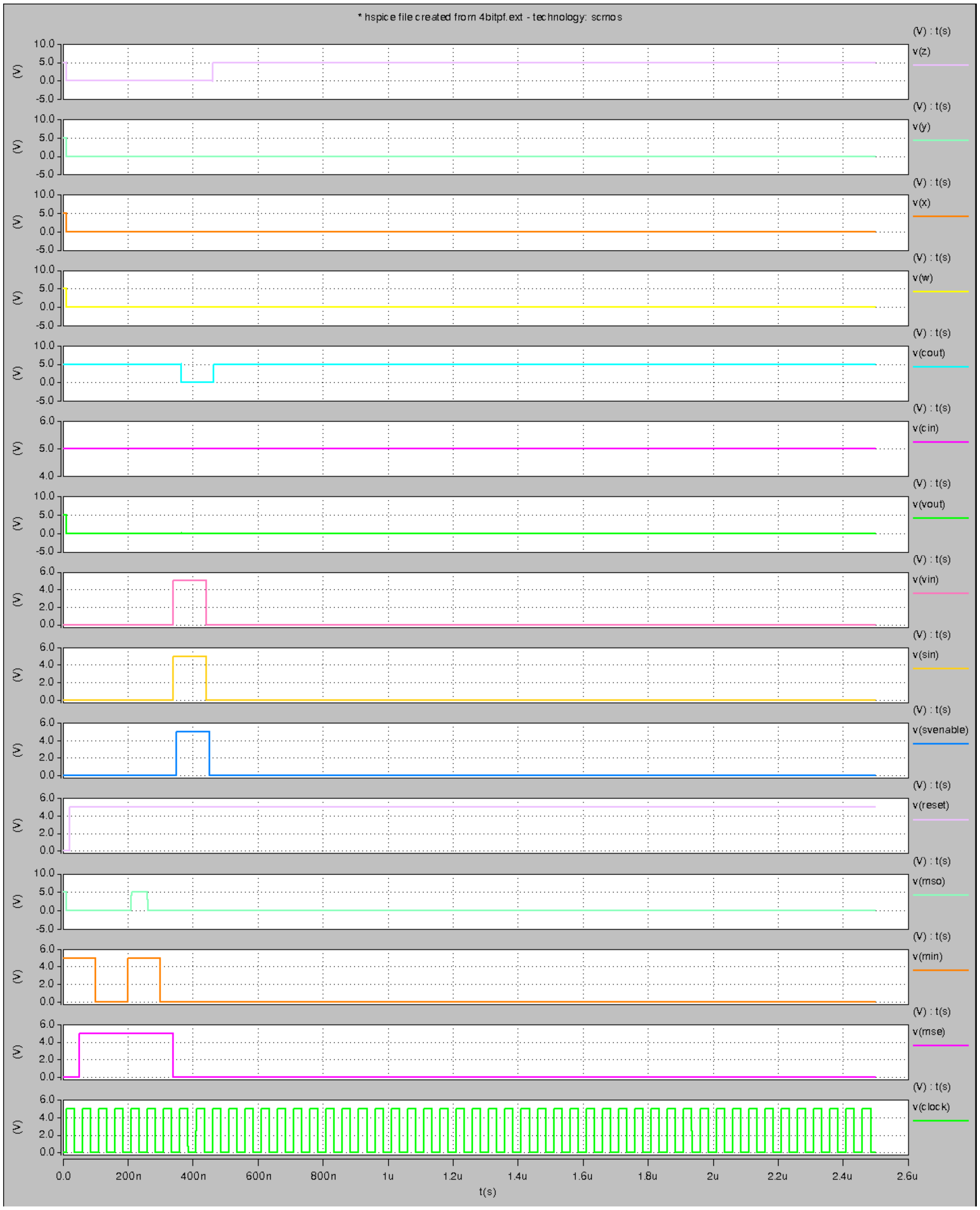


The COUT which is the result of the comparison, is one only when the cin is 1, valid is 1 along with the matching inputs. The substring, main string and the valid outputs are following their respective inputs after one clock cycle.

4 bit string matcher:



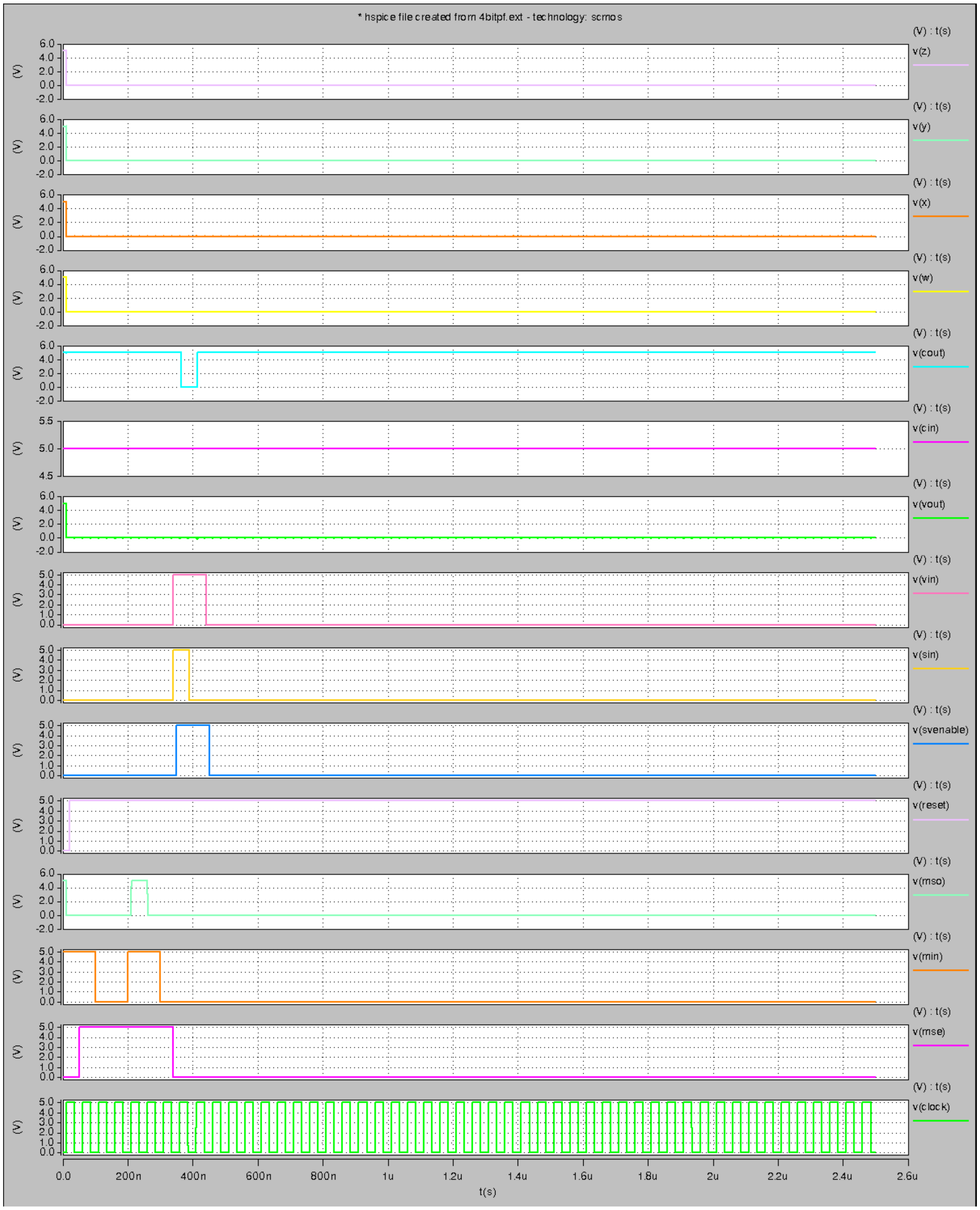
Spice simulation results:



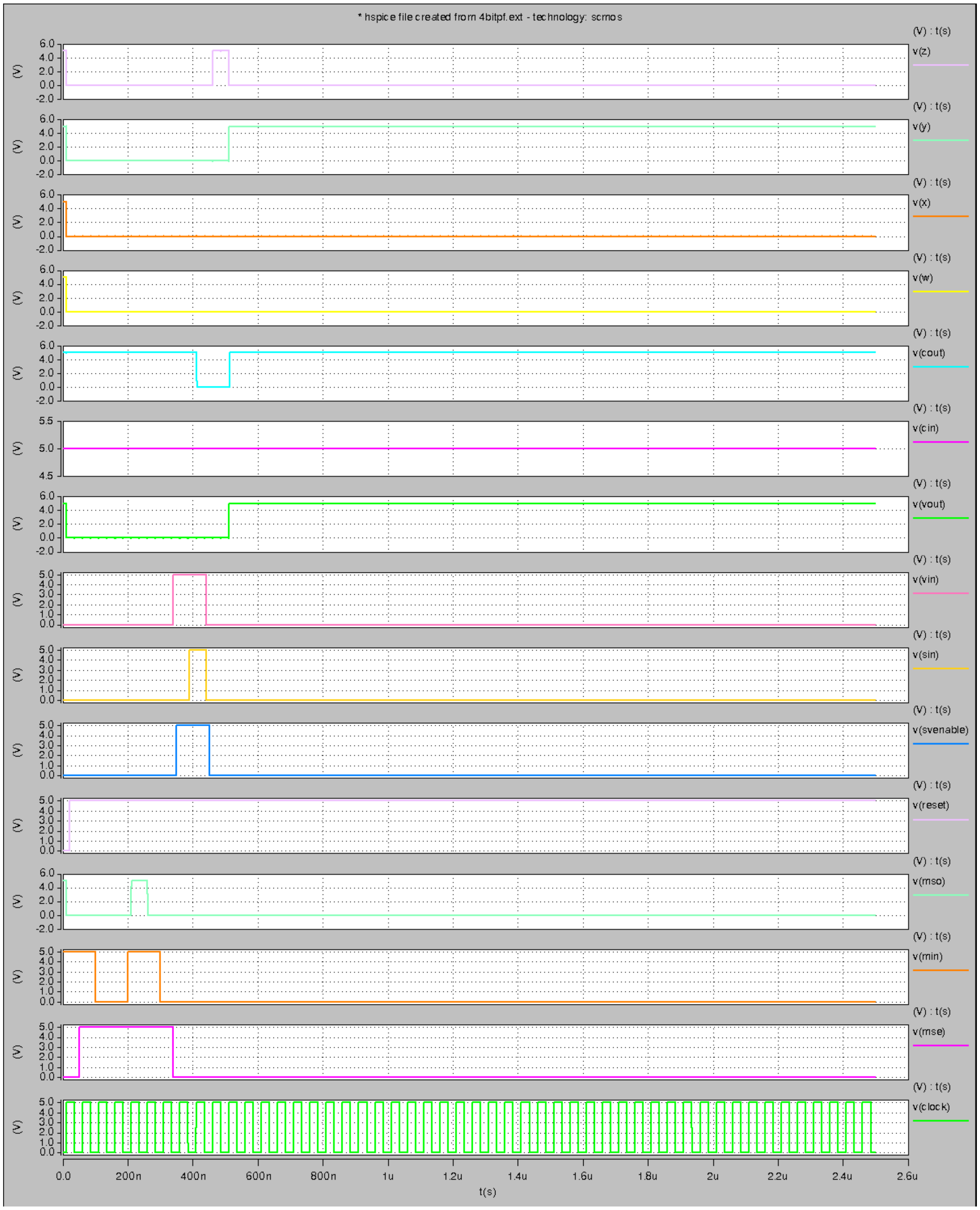
Comments: The test vector is given such that the match occurs at the first position.

Main string: 1011

Substring: 01. Match at 1st position. The counter output can be seen as 0001.



Comments: The test vector is given such that it's a match at the 0th position.
Main string 1011
Substring 10   The counter output can be seen as 0000

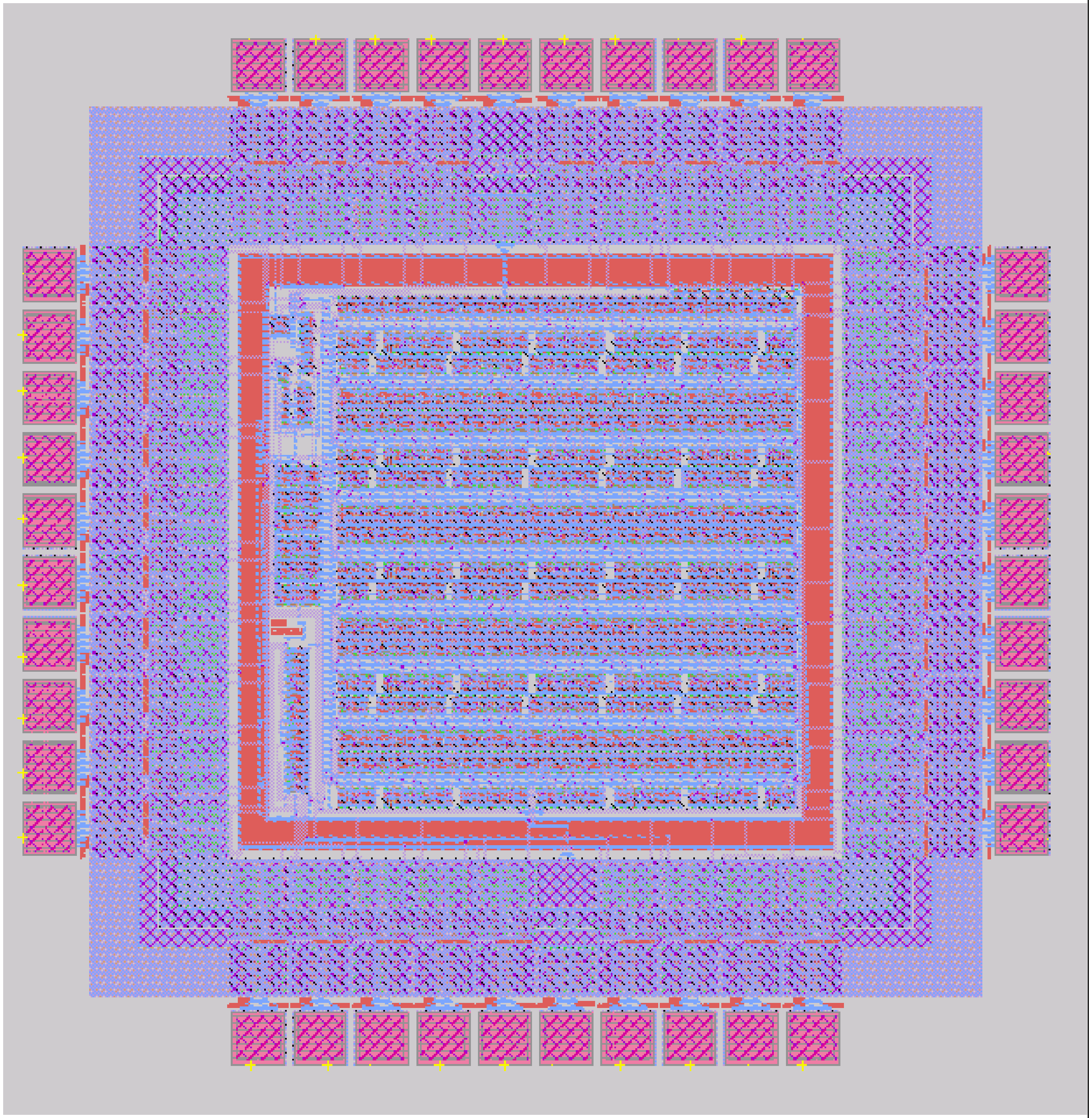


Comments:

The main string is given as 1011.

The substring is given as 11 so match occurs at the position 2. The Counter can be seen as 0010

Final pad frame:



Simulation and Testing strategy

The simulations are done such that there are matches at various positions in the main string.

This is done to thoroughly verify the working of the chip

Some cases are:

- The match occurs at the immediate first position.
- The match occurs at the last position.
- The match occurs at the middle position.
- The match doesn't occur.

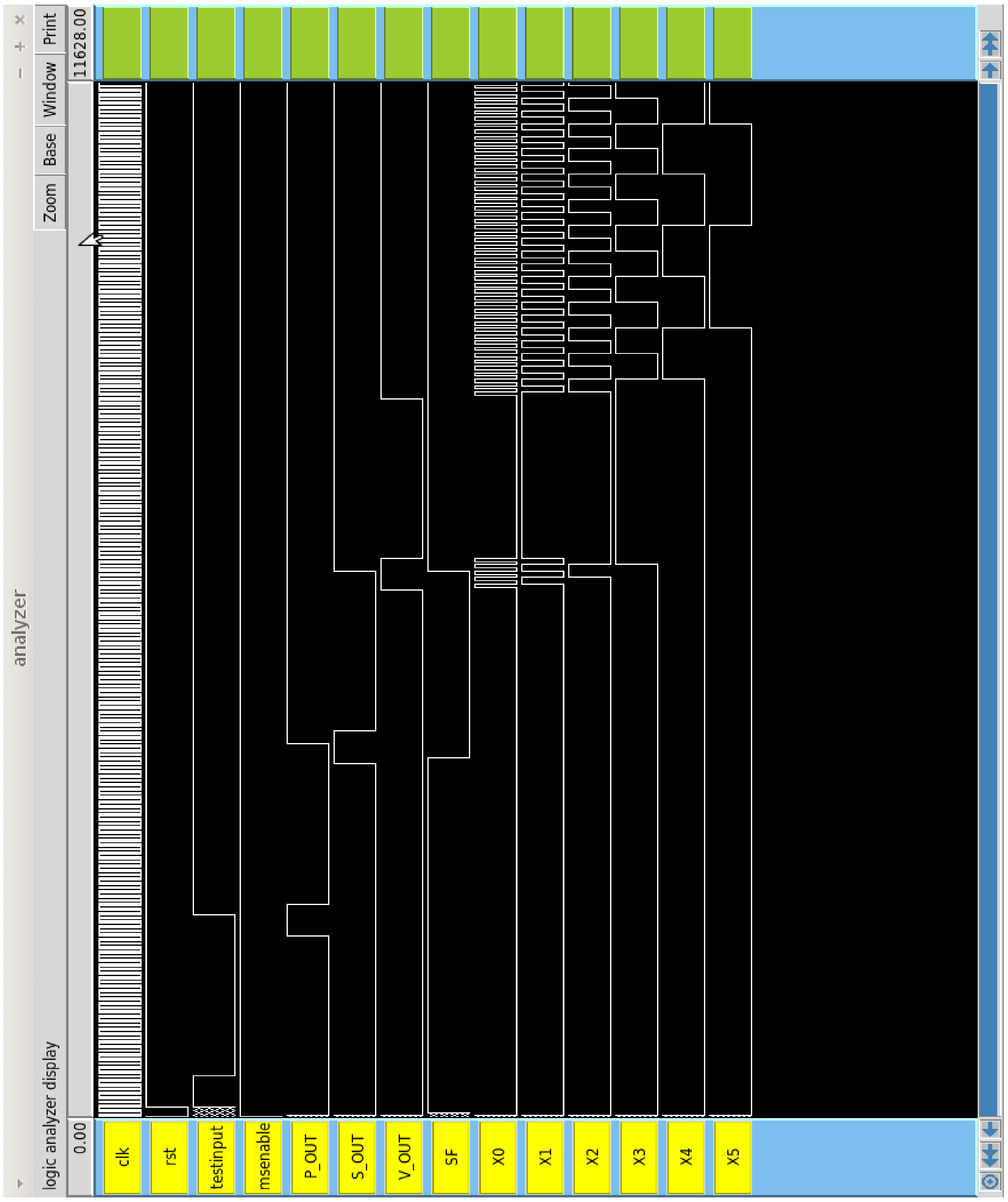
Critical Path:

The critical path for our design occurs when there is a match in all the previous slices as well as match occurs at the last slice.
In that case, the match from the previous 53 slices must reflect and be transmitted to the final multiplexer of the slice through 53 AND gates.
This gives the worst case delay and is the critical path for our design.

IRSIM command file for test mode:

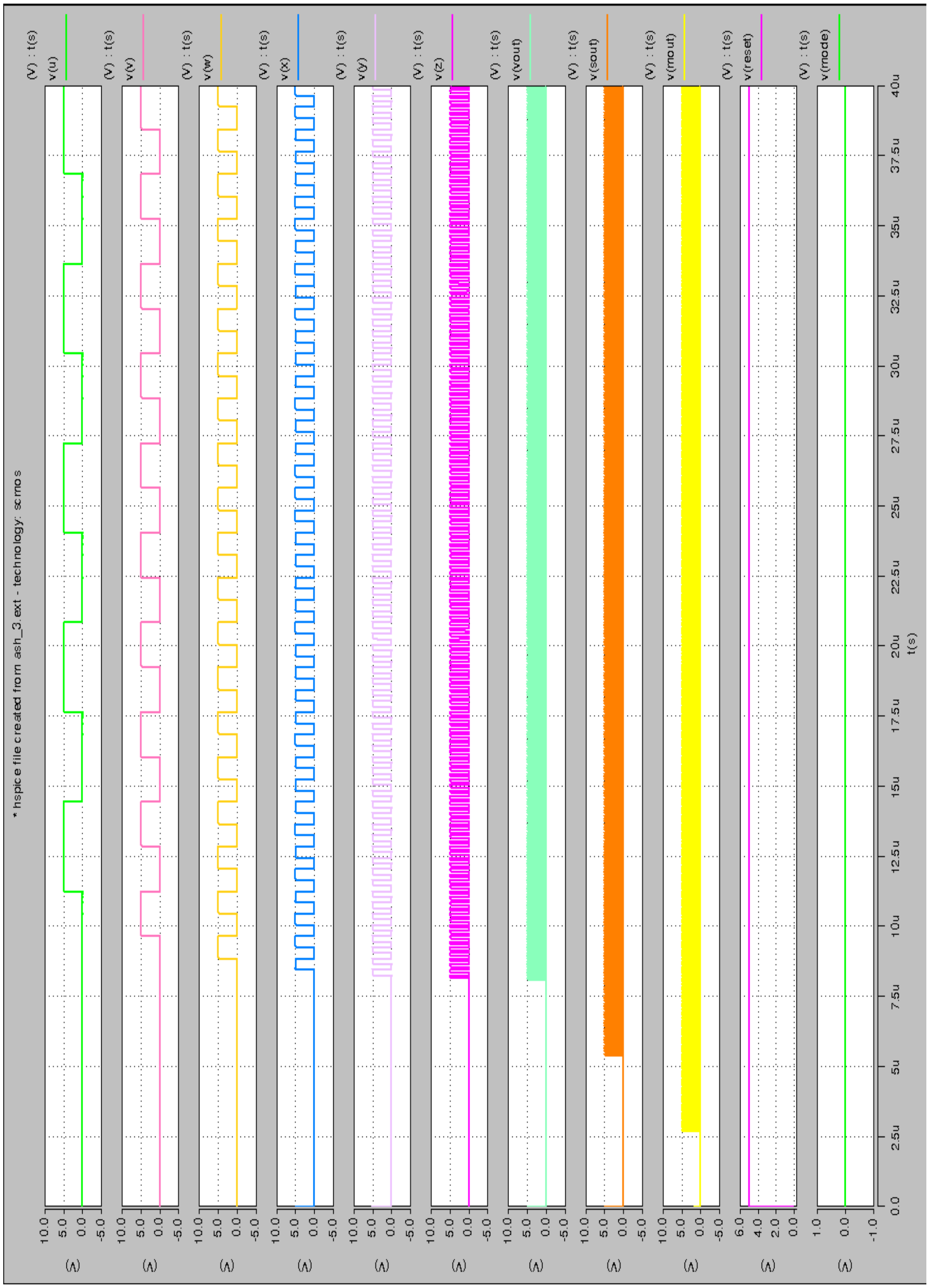
```
stepsize 20n
vector clk CLOCK
vector rst RESET
vector msenable PE
vector svenable SVE
vector modpin MODE
vector testinput TEST_IN
vector main P_IN
vector sub S_IN
vector validinput V_IN


analyzer clk rst testinput msenable P_OUT S_OUT V_OUT SF X0 X1 X2 X3 X4 X5

clock clk 0 1
setvector modpin 0

setvector msenable 1

setvector rst 0
c 3

setvector rst 1

setvector testinput 1
c 10

setvector testinput 0
c 50

setvector testinput 1
c 10

setvector testinput 1
c 250
```

Simulation results:



It can be seen on the waveforms that test input is passed through all the flip flops in the scan chain. Main string, sub string and valid output of the final bit slice are the replica of test input. After passing through all the flip flops test input triggers the counter.

Spice test mode:



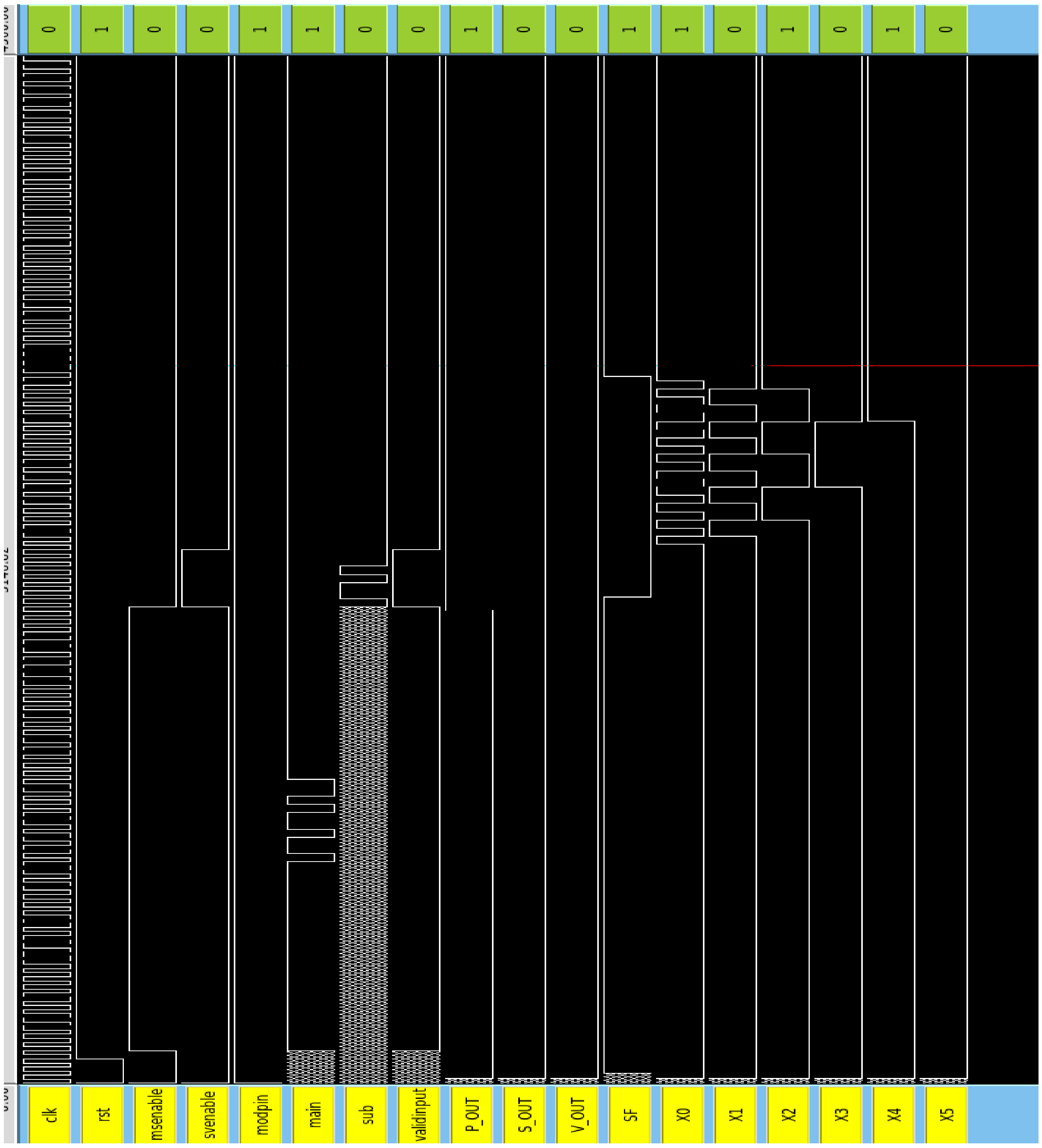
Comments: The mode is set to 0 and the input is given as a clock which can be seen to be followed by the main string, substring and valid outputs.

IRSIM command file: normal mode

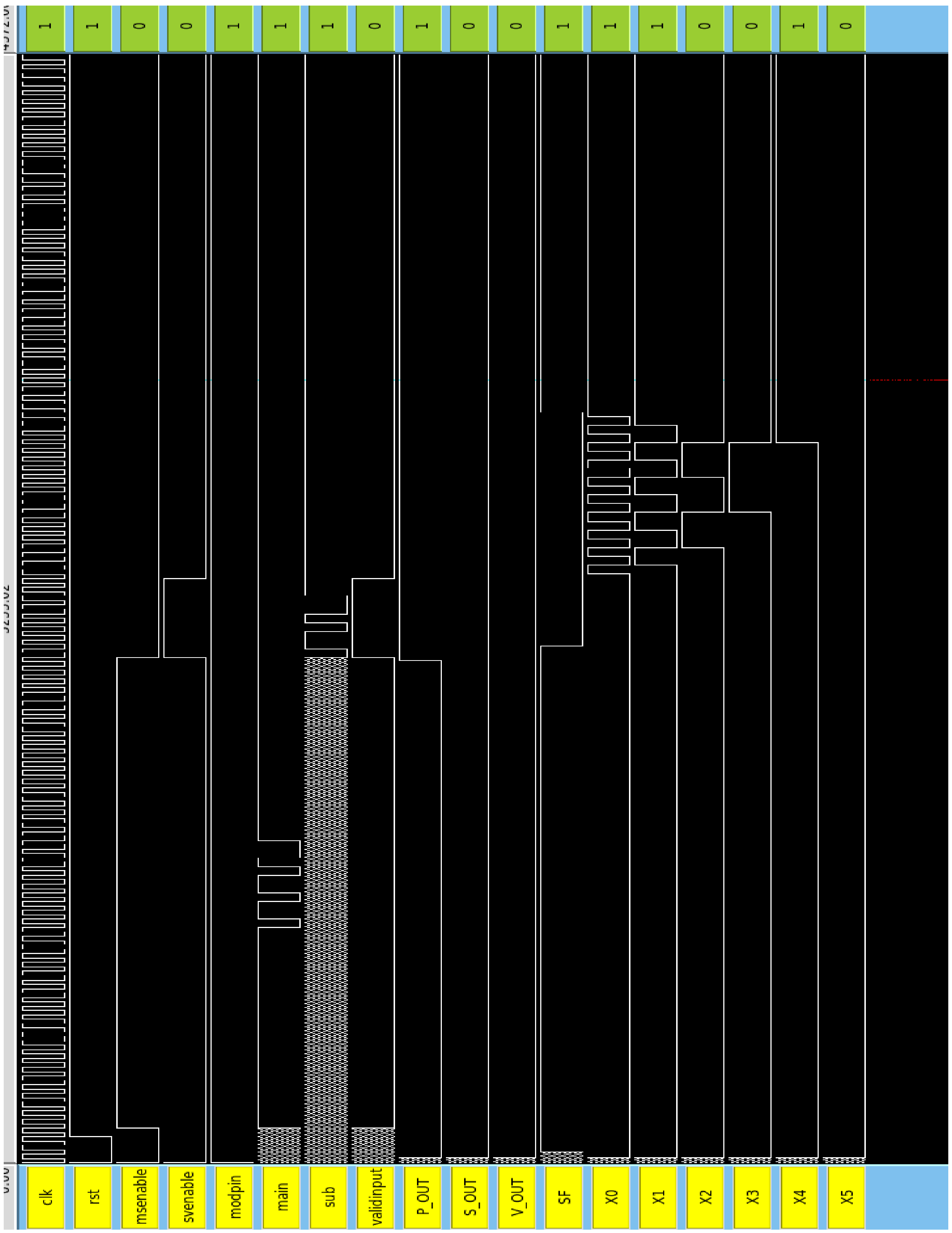
```
vector clk CLOCK
vector rst RESET
vector msenable PE
vector svenable SVE
vector modpin MODE
vector testinput TEST_IN
vector main P_IN
vector sub S_IN
vector validinput V_IN

analyzer clk rst msenable svenable modpin main sub validinput P_OUT S_OUT V_OUT SF X0
X1 X2 X3 X4 X5

clock clk 0 1
setvector modpin 1
setvector msenable 0
setvector svenable 0
setvector rst 0
c 3
setvector rst 1
c 1
setvector validinput 0
setvector msenable 1
setvector svenable 0
setvector main 0
c 1
setvector main 1
c 1
setvector validinput 1
setvector msenable 0
setvector svenable 1
setvector sub 0
c 1
setvector validinput 0
setvector msenable 0
setvector svenable 0
c 60
```

Irsim simulation results

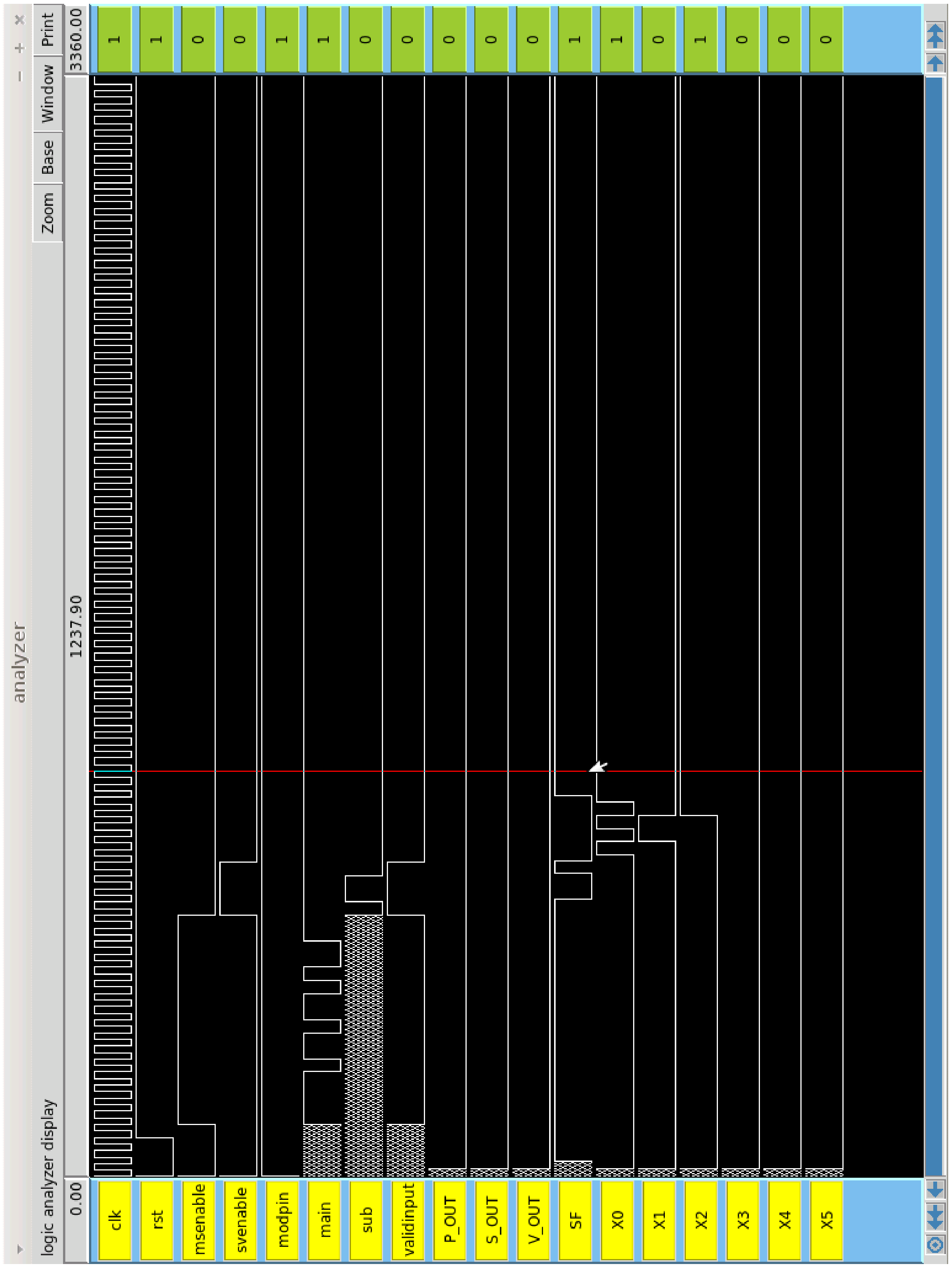
Index match at 010101



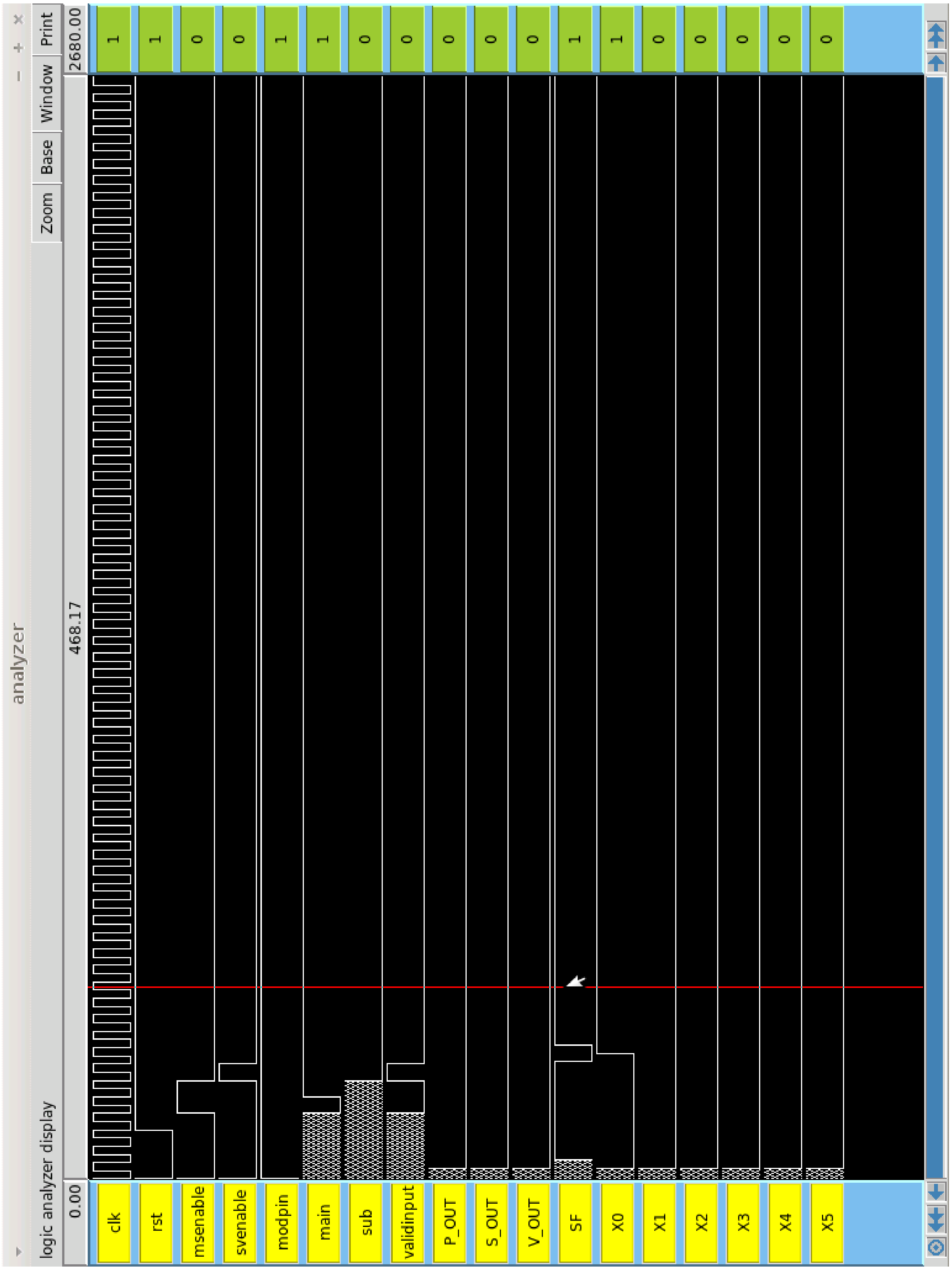
Mismatch condition



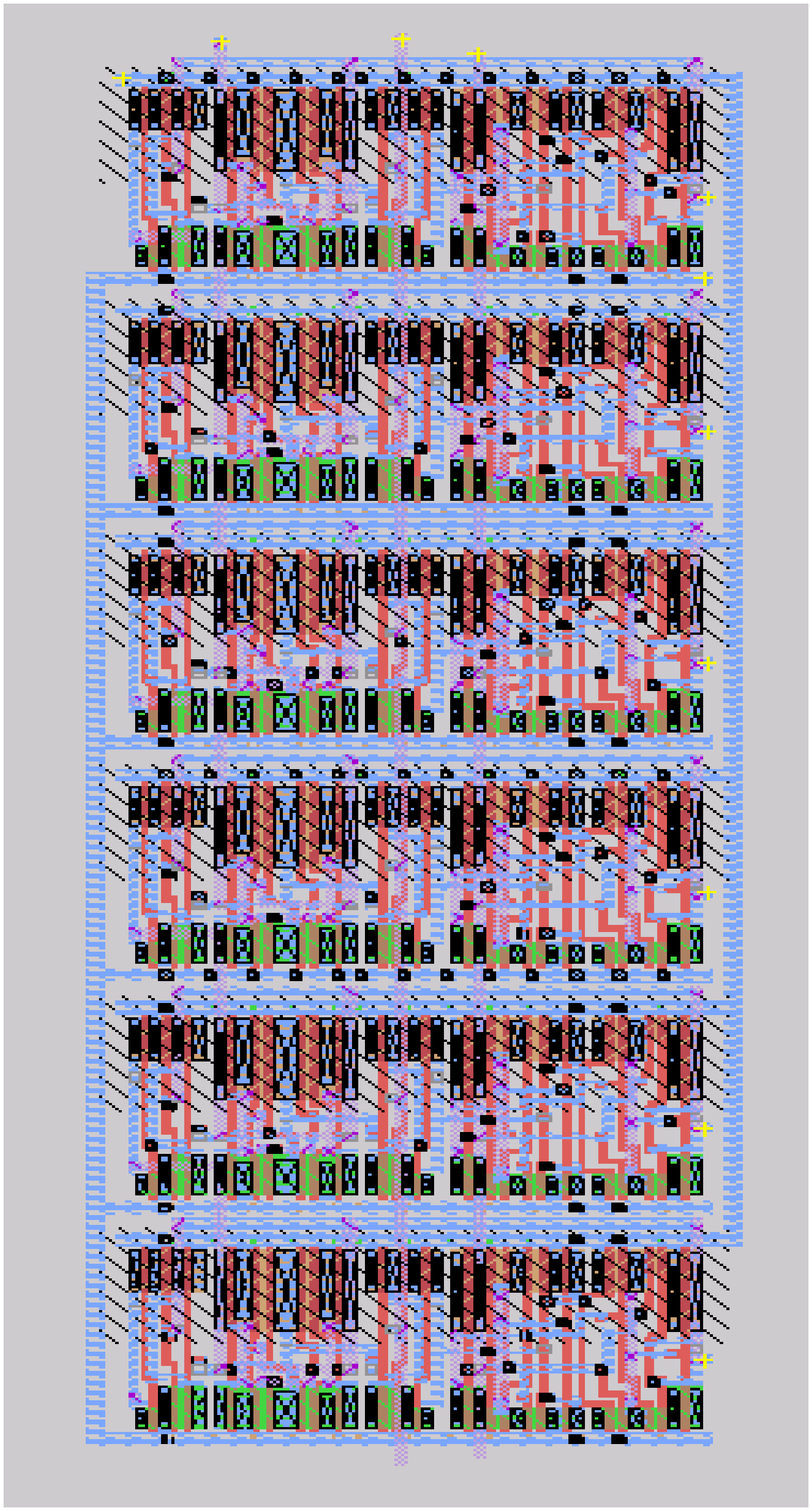
Index match at 000101



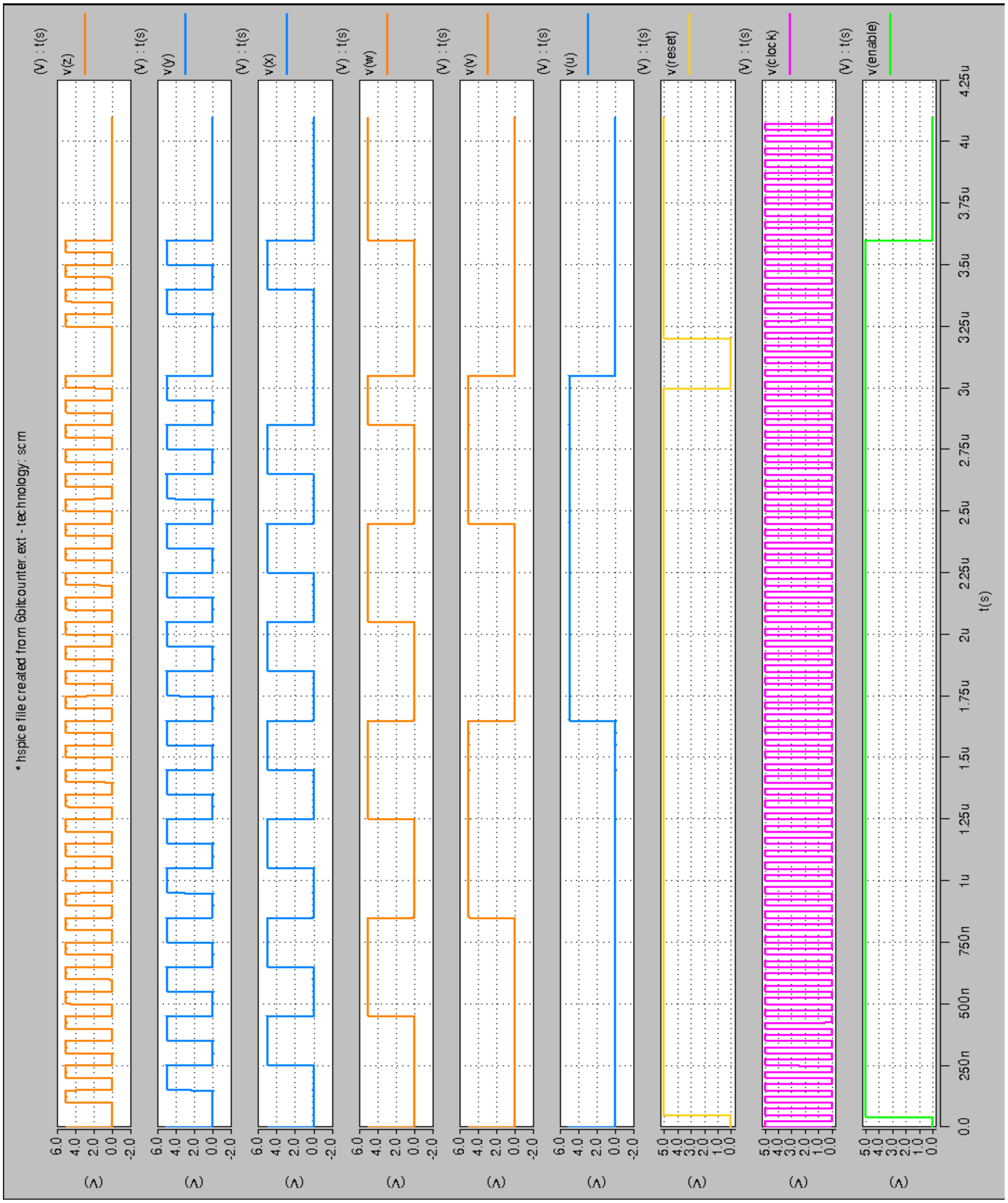
Index match at 000001



Counter:

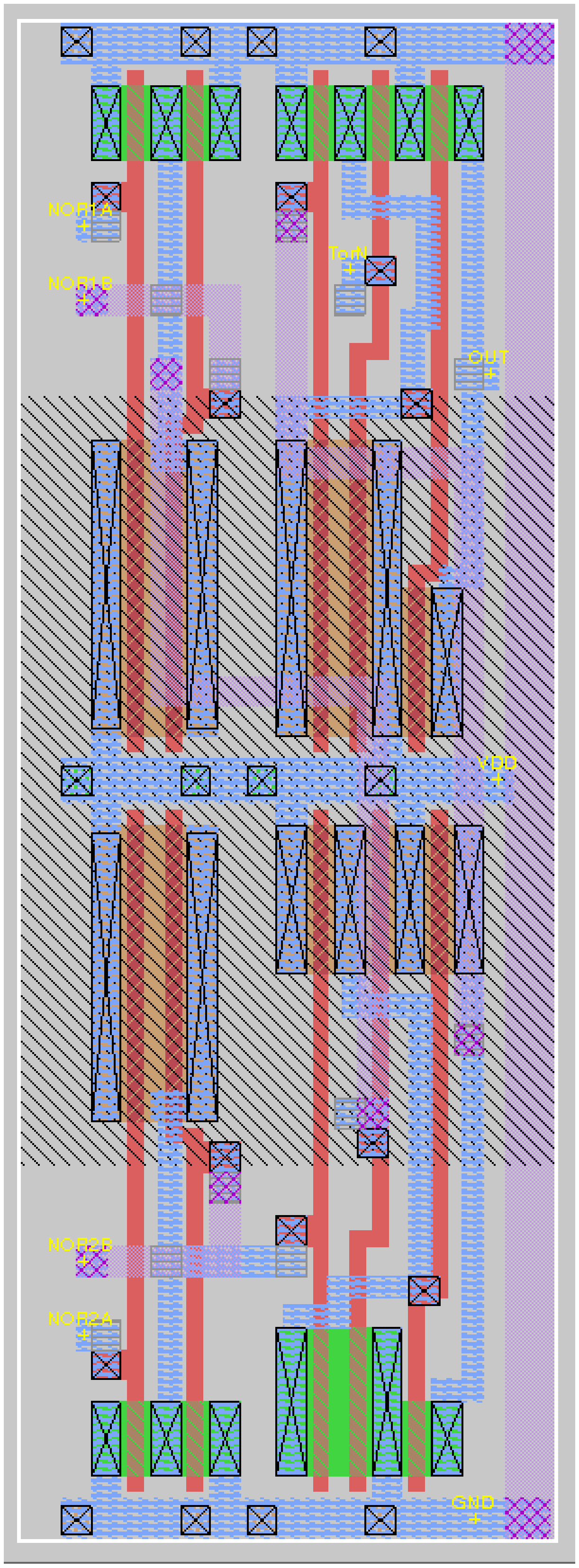


Spice waveforms:

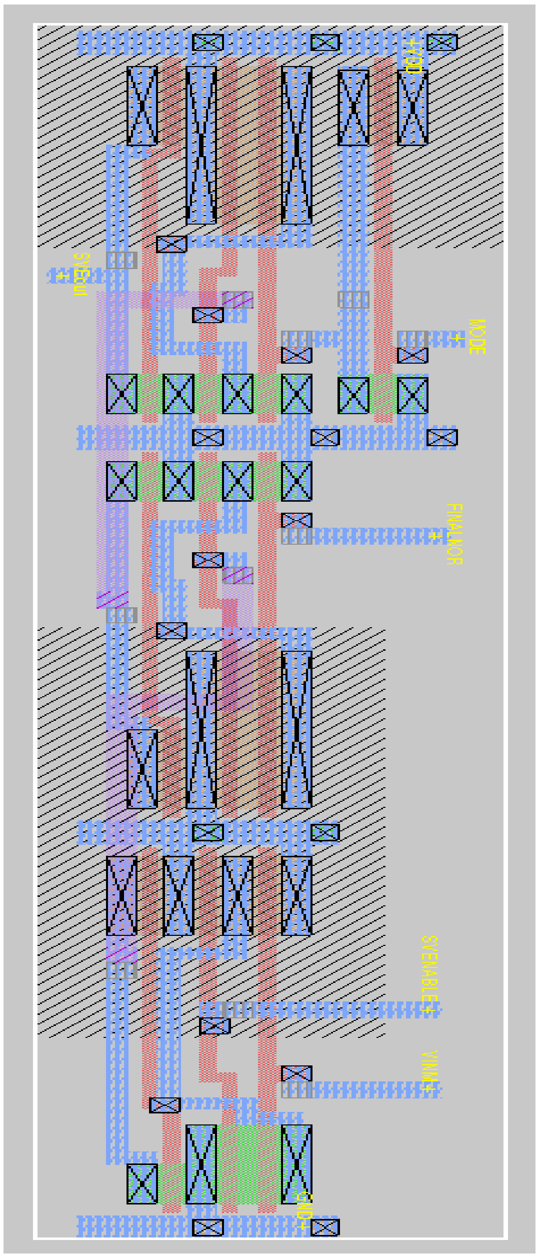


U is the MSB and Z is the LSB.

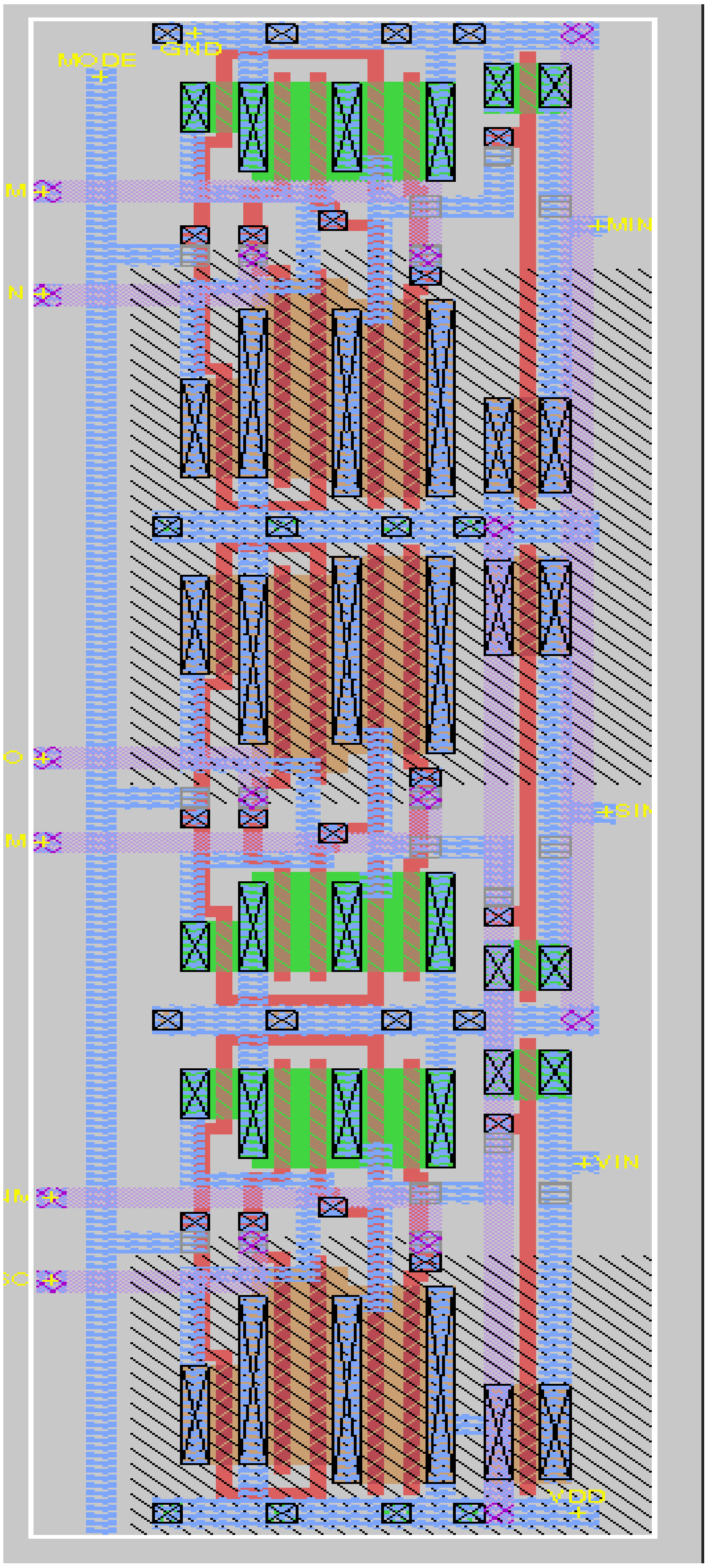
Enable logic for counter



Logic for the SV Enable



Logic for the TEST/NORMAL mode



SPICE SIMULATION DELAYS FOR THE GATES USED

AND	187.54ps
OR	198.75 ps
NOT	74.35 ps
XOR	226.08 ps
XNOR	258.34 ps
NOR	96.47 ps
MUX	184.07 ps
DFF	373.45 ps

VHDL MODELS WITH BACK ANNOTATED TIMING AND SIMULATIONS

VHDL code for Bit slice:

```
library ieee;
use ieee.std_logic_1164.all;
--entity declaration
entity bitslice1 is
   port( p_en,sv_en,p_in ,s_in,v_in,clk,reset,compin: in std_logic;
            p_out,s_out,v_out,comp_out : out std_logic);
end bitslice1;
--working
architecture struct1 of bitslice1 is
--1dff
component dff
port (
d,clk,reset: in std_logic;
q : out std_logic);
end component;
--2xnorgate
component xnorgate
port(p,q:in std_logic;
      r:out std_logic);
end component;
--3mux
component mux
port ( a : in  std_logic;
       b : in  std_logic;
       s : in  std_logic;
       z : out std_logic  );
end component;
--4and
component andgate
   port( a, b : in std_logic;
            f : out std_logic);
end component;
--5not
component notgate
   port( m : in std_logic;
            n : out std_logic);
end component;
--6or
component orgate
port ( a,b:in std_logic;
  c:out std_logic);
```

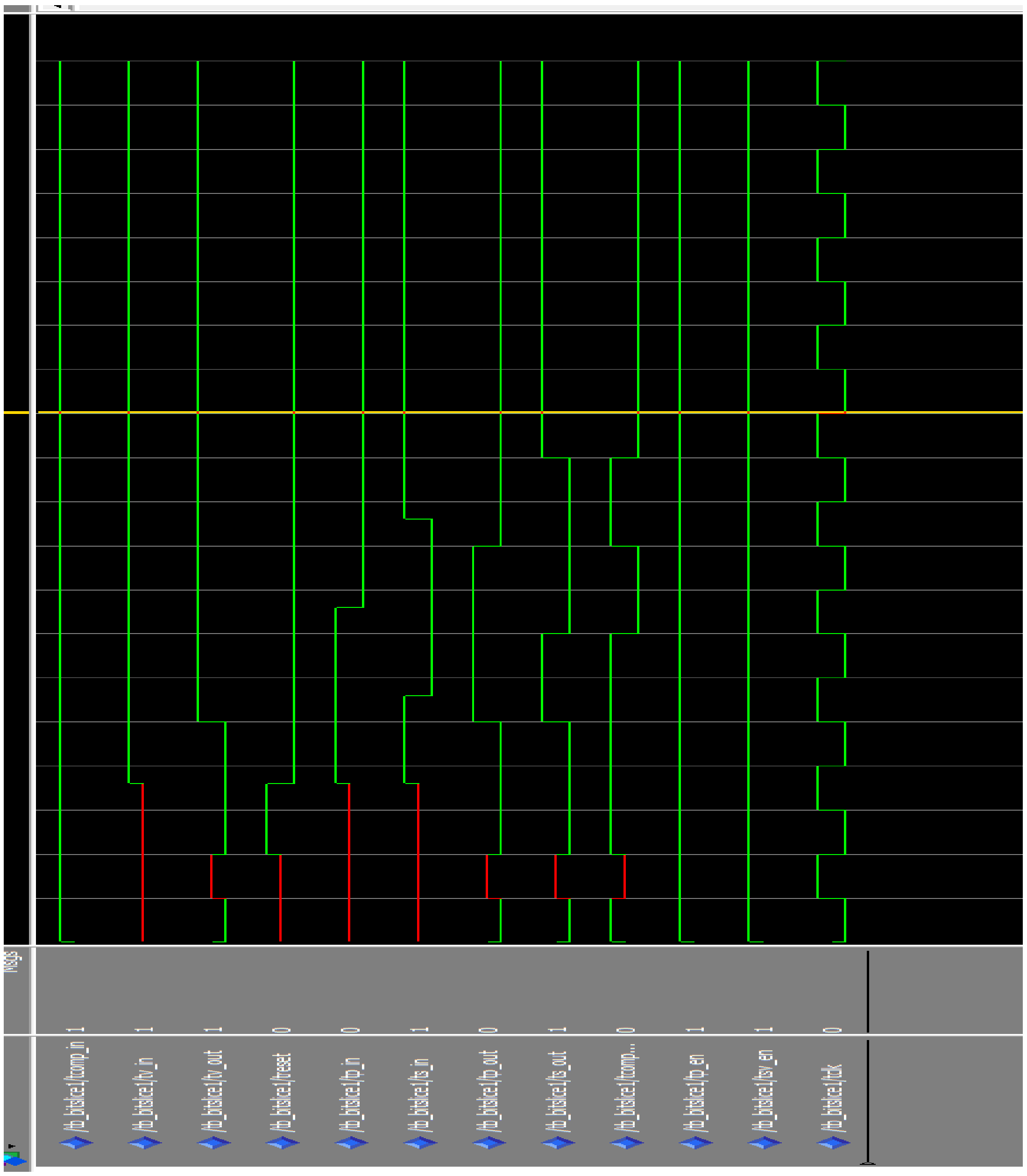
```
end component;
------signals----------
signal spz,ssz,sorout,svz,spq,ssq,svq,sr,sand,scompz: std_logic;
--component declaration ends
begin
-------------BIT SLICE1------
-----mux
muxp1: mux port map (spq,p_in,p_en,spz);
muxs1: mux port map (ssq,s_in,sv_en,ssz);
muxv1: mux port map (svq,v_in,sv_en,svz);
--flipflops
dffp1: dff port map (spz,clk,reset,spq);
dffs1: dff port map (ssz,clk,reset,ssq);
dffv1: dff port map (svz,clk,reset,svq);
-----------------complogic------------------
xnor1: xnorgate port map (spq,ssq,sr);
andv1: andgate port map (sr,compin,sand);
muxcomp1: mux port map (compin,sand,svq,scompz);
----output signals------
p_out<=spq;
s_out<=ssq;
v_out<=svq;
comp_out<=scompz;
end struct1;
```

Test Bench for Bit slice:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity tb_bitslice1 is
end tb_bitslice1;
architecture testbitslice1 of tb_bitslice1 is
component bitslice1 is
   port( p_en,sv_en,p_in ,s_in,v_in,clk,reset,compin: in std_logic;
            p_out,s_out,v_out,comp_out : out std_logic);
end component;
signal tp_in ,ts_in, tp_out,ts_out,tv_out,tcomp_out : std_logic;
signal tp_en,tsv_en,tv_in,tcomp_in: std_logic ;
signal tclk, treset        : std_logic;
begin
   U1 : bitslice1
 port map (tp_en,tsv_en,tp_in ,ts_in,tv_in,tclk,treset,tcomp_in,tp_out,ts_out,tv_out,tcomp_out);

  process
```

```
       begin
          tclk <= '0';
          wait for 50 ns;
          tclk <= '1';
          wait for 50 ns;
end process;

process
       begin
          tclk <= '0';
          wait for 50 ns;
          tclk <= '1';
          wait for 50 ns;
end process;
process
begin
          tp_en<='1';
          tsv_en<='1';
          tcomp_in<='1';

         wait for 100 ns;
          treset<='1';
          wait for 80 ns;---180 ns
          tv_in<='1';
          tp_in<='1';
          ts_in<='1';
          wait for 100 ns;
          tp_in<='1';
          ts_in<='0';
          wait for 100 ns;
          tp_in<='0';
          ts_in<='0';
          wait for 100 ns;
          tp_in<='0';
          ts_in<='1';
          wait for 100 ns;

wait;
end process;
end testbitslice1;
```

Wave forms for Bit slice:



VHDL code for N-bit (With test mode):

```
library ieee;
use ieee.std_logic_1164.all;
--entity declaration
entity testmode is
generic (n:integer:=3);
port( np_en,np_in ,ns_in,nv_in,nclk,nreset,svenable: in std_logic;
        np_out,ns_out,nv_out,ncomp_out : out std_logic;
         index:out std_logic_vector (5 downto 0);
         testin,mode:in std_logic);
end testmode;
architecture structn of testmode is
--outside logic---
--1and gate--
component andgate
   port( a, b : in std_logic;
        f : out std_logic);
end component;
--2notgate--
component notgate
   port( m : in std_logic;
        n : out std_logic);
end component;
--3orgate--
component orgate
port(
  a,b:in std_logic;
  c:out std_logic);
end component;
--4norgate--
component norgate
   port( x, y : in std_logic;
        z : out std_logic);
end component;
--bitslicecomponent--
component bitslice1
   port( p_en,sv_en,p_in ,s_in,v_in,clk,reset,compin : in std_logic;
        p_out,s_out,v_out,comp_out : out std_logic);
end component;
--up counter
component gencounter
generic (n:integer:=6);
port (
enable,clk,reset: in std_logic;
q : out std_logic_vector ( n-1 downto 0));
end component ;
-----mux
component mux is
port ( a : in  std_logic;
       b : in  std_logic;
       s : in  std_logic;
       z : out std_logic
     );
end component;
signal snp_in ,sns_in,snv_in,sncomp_in:std_logic_vector(n downto 0);
signal  ssandout,ssorout,sssorout:std_logic;
signal ssn_resin:std_logic;
------counter signal---
signal snoroutput,sandoutput:std_logic;
signal counteroutput: std_logic_vector (5 downto 0);
------------test mode signals--------
signal tp_in,ts_in,tv_in,tenable,notmode: std_logic;
--codebegins--
begin
----------------test mode logic----------
mux1: mux port map (testin,np_in,mode,tp_in);
mux2: mux port map (snp_in(n),ns_in,mode,ts_in);
mux3: mux port map (sns_in(n),nv_in,mode,tv_in);
mux4: mux port map (snv_in(n),sandoutput,mode,tenable);
notgatetest: notgate port map (mode,notmode);
-----------svenable logic---------------
n_andgate: andgate port map (svenable,tv_in,ssandout);
n_orgate: orgate port map (ssandout,ssn_resin,sssorout);
orgatetest: orgate port map ( notmode,sssorout,ssorout);
n_norgate: norgate port map (snv_in(n),sncomp_in(n),ssn_resin);
------------main inputs--------
snp_in(0)<=tp_in;
sns_in(0)<=ts_in;
snv_in(0)<=tv_in;
sncomp_in(0)<='1';
------generate stamnt starts----------------
nbit: for i in 0 to n-1 generate
begin
n_bitslice: bitslice1 port map ( np_en,ssorout,snp_in(i),sns_in(i),snv_in(i),nclk,nreset,
sncomp_in(i),snp_in(i+1),sns_in(i+1),snv_in(i+1),sncomp_in(i+1));
end generate nbit;
----------generate ends---------
----main output signals------
```

```
np_out<=snp_in(n);
ns_out<=sns_in(n);
nv_out<=snv_in(n);
ncomp_out<=sncomp_in(n);
-------final counter logic--------
norn: norgate port map (np_en,svenable,snoroutput);
andgaten: andgate port map (snoroutput,ssn_resin,sandoutput);
counter :   gencounter   port map (tenable,nclk,nreset,counteroutput);
index<=counteroutput;
end structn;
```

Test bench: (normal mode) (mode: 1)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity testchip is
end testchip;

architecture tb of testchip is

 component testmode
generic (n:integer:=64);
port( np_en,np_in ,ns_in,nv_in,nclk,nreset,svenable: in std_logic;
            np_out,ns_out,nv_out,ncomp_out : out std_logic;
             index:out std_logic_vector (5 downto 0);
             testin,mode:in std_logic);
end component;

signal tnp_in ,tns_in, tnp_out,tns_out,tnv_out,sucess_fail,tsvenable: std_logic;
signal tnp_en,tnv_in: std_logic ;
signal tnclk, tnreset: std_logic;
signal tindex: std_logic_vector (5 downto 0);
signal stestin,smode: std_logic;
begin

   U1 : testmode
 port map (tnp_en,tnp_in ,tns_in,tnv_in,tnclk,tnreset,tsvenable,tnp_out,tns_out,tnv_out,sucess_fail,
            tindex,stestin,smode);

  process
      begin
```

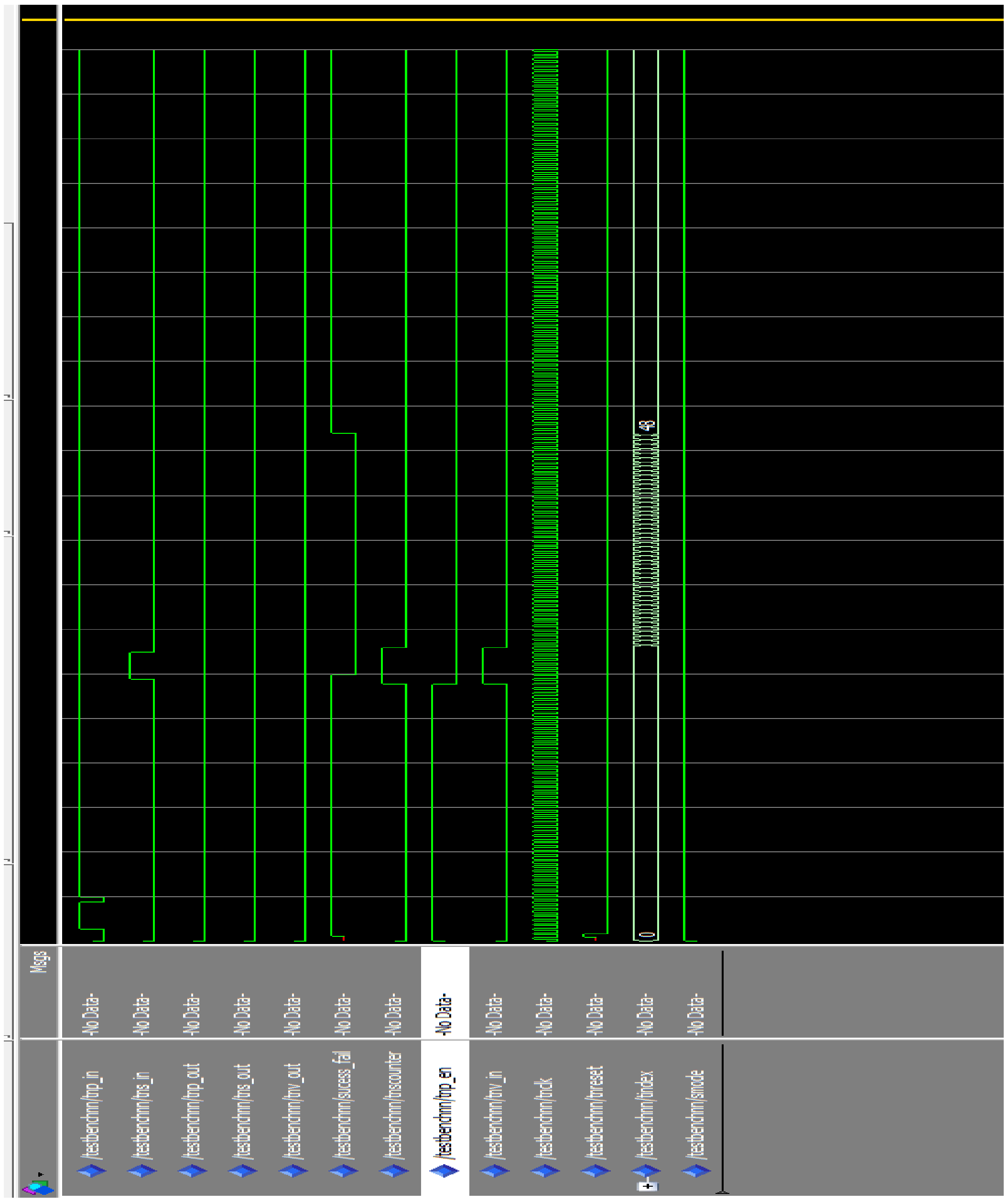
```
          tnclk <= '0';
          wait for 5 ns;
          tnclk <= '1';
          wait for 5 ns;
end process;
process
begin
          smode<='1';
          tnp_in<='0';
          tns_in<='0';
          tnv_in<='0';
          tnp_en<='1';
        tsvenable<='0';


          wait for 10 ns;
          tnreset<='1';
          wait for 8 ns;--180 ns
          --------p string input starts------
          tnreset<='0';
          tnp_en<='1';
--1-----------------------------------
           tnp_in<='0';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns

           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns
          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='0';
           wait for 10 ns; ---951 ns
 ----------------------------------------
--2-----------------------------------
           tnp_in<='1';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns

           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns

          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='1';
           wait for 10 ns; ---951 ns
 ----------------------------------------
--3-----------------------------------
           tnp_in<='1';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns
           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns

          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='1';
           wait for 10 ns; ---951 ns
 ----------------------------------------
--4------------------------------------
           tnp_in<='1';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns

           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns

          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='1';
           wait for 10 ns; ---951 ns
 ----------------------------------------
--5-----------------------------------
           tnp_in<='1';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns

           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns

          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='1';
           wait for 10 ns; ---951 ns
 ----------------------------------------
--6-----------------------------------
           tnp_in<='1';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns

           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
```

```
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns

          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='1';
           wait for 10 ns; ---951 ns
 ---------------------------------------
--7----------------------------------
           tnp_in<='1';
          wait for 10 ns;--280 ns(last bit)

          tnp_in<='1';
          wait for 10 ns;---380 ns

          tnp_in<='1';
          wait for 10 ns;---480 ns

           tnp_in<='1';
          wait for 10 ns;---580 ns

          tnp_in<='1';
          wait for 10 ns;---680 ns

          tnp_in<='1';
          wait for 10 ns;---780 ns

          tnp_in<='1';
          wait for 10 ns;---880 ns

           tnp_in<='1';
           wait for 10 ns; ---951 ns
 ---------------------------------------

-------------p string ends----------------
tnv_in<='1';
           tnp_en<='0'; tsvenable<='1';
```

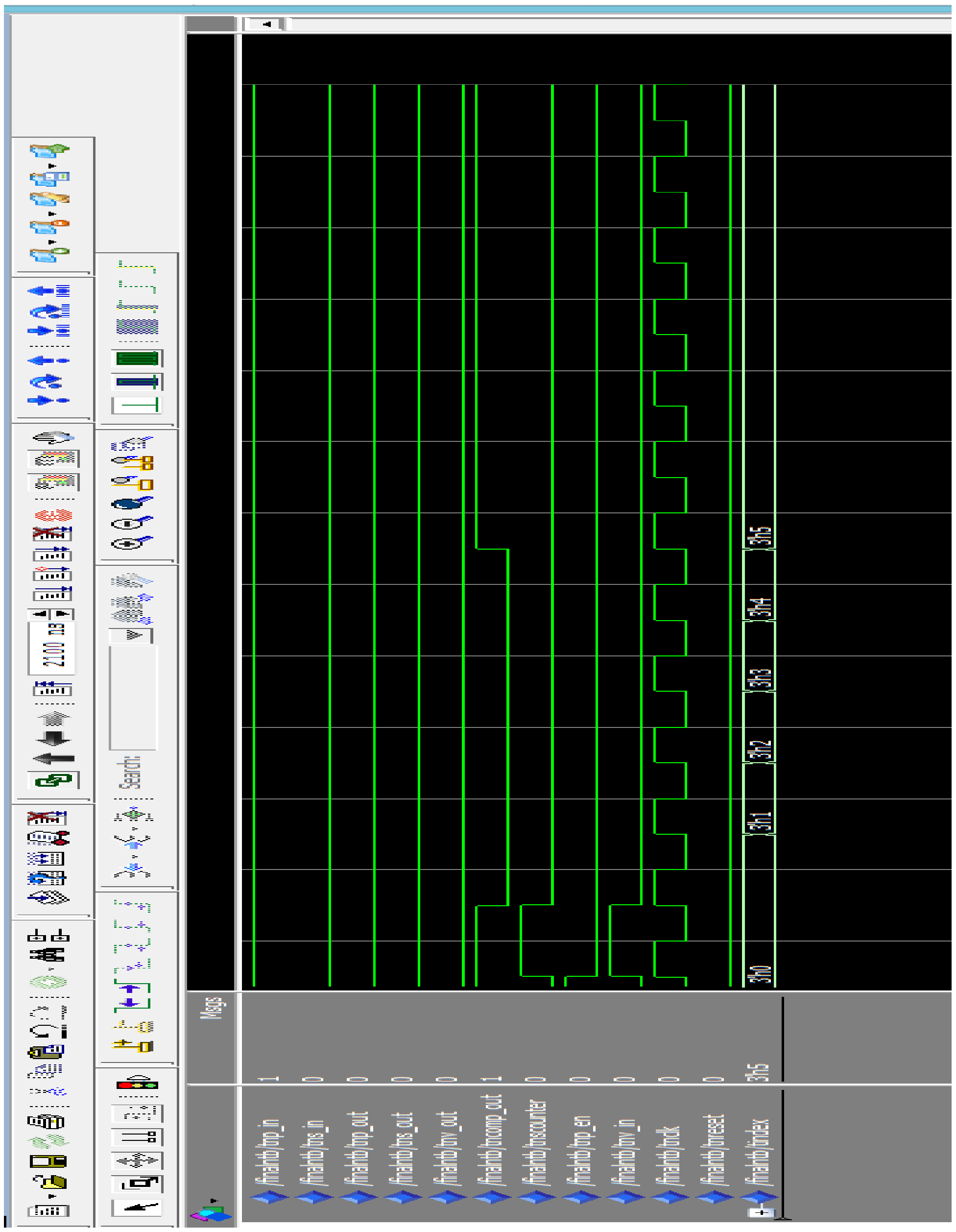
```
----------sub string starts--------------------
--1-----------------------------------
           tns_in<='0';
          wait for 10 ns;--280 ns(last bit)

          tns_in<='1';
          wait for 10 ns;---380 ns

          tns_in<='1';
          wait for 10 ns;---480 ns

           tns_in<='1';
          wait for 10 ns;---580 ns

          tns_in<='1';
          wait for 10 ns;---680 ns

          tns_in<='1';
          wait for 10 ns;---780 ns

          tns_in<='1';
          wait for 10 ns;---880 ns

           tns_in<='0';
           wait for 10 ns; ---951 ns
 ------------substring ends----------------------------
--------------------------------------
           tnv_in<='0';
            tsvenable<='0';
--------------------------------------

wait;
end process;
end tb;
```

Simulation results: (normal mode)

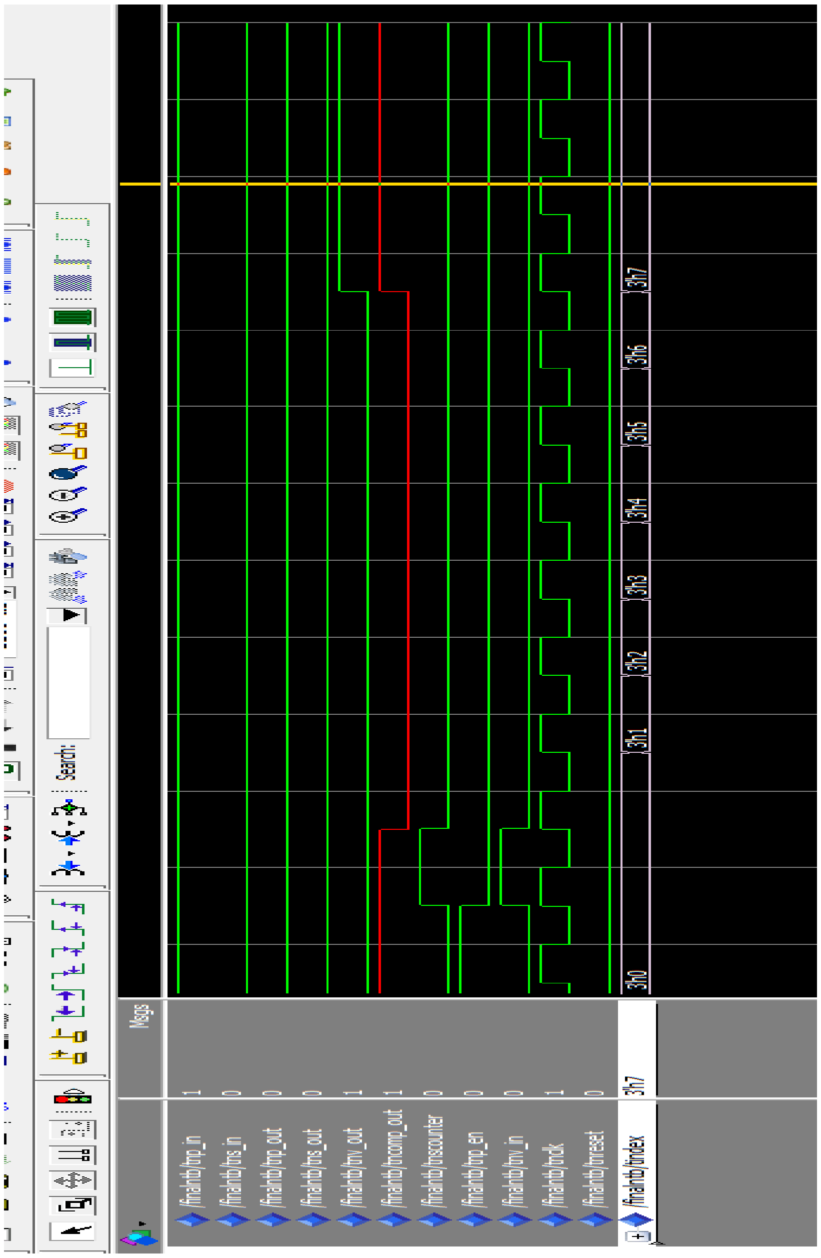
Index match at 48th location.



Main string: 1111100 substring: 0   match at index: 5

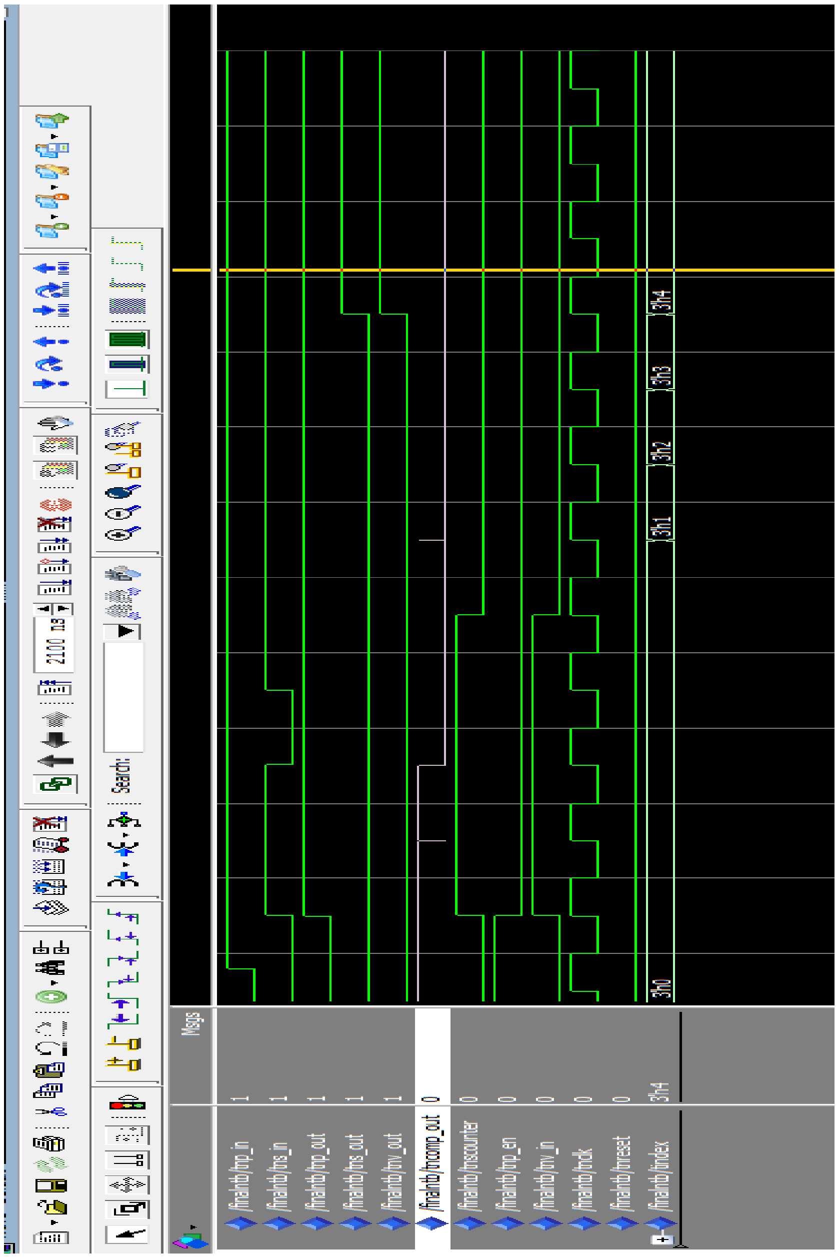


Main string: 11111110 substring: 0 match at index: 7

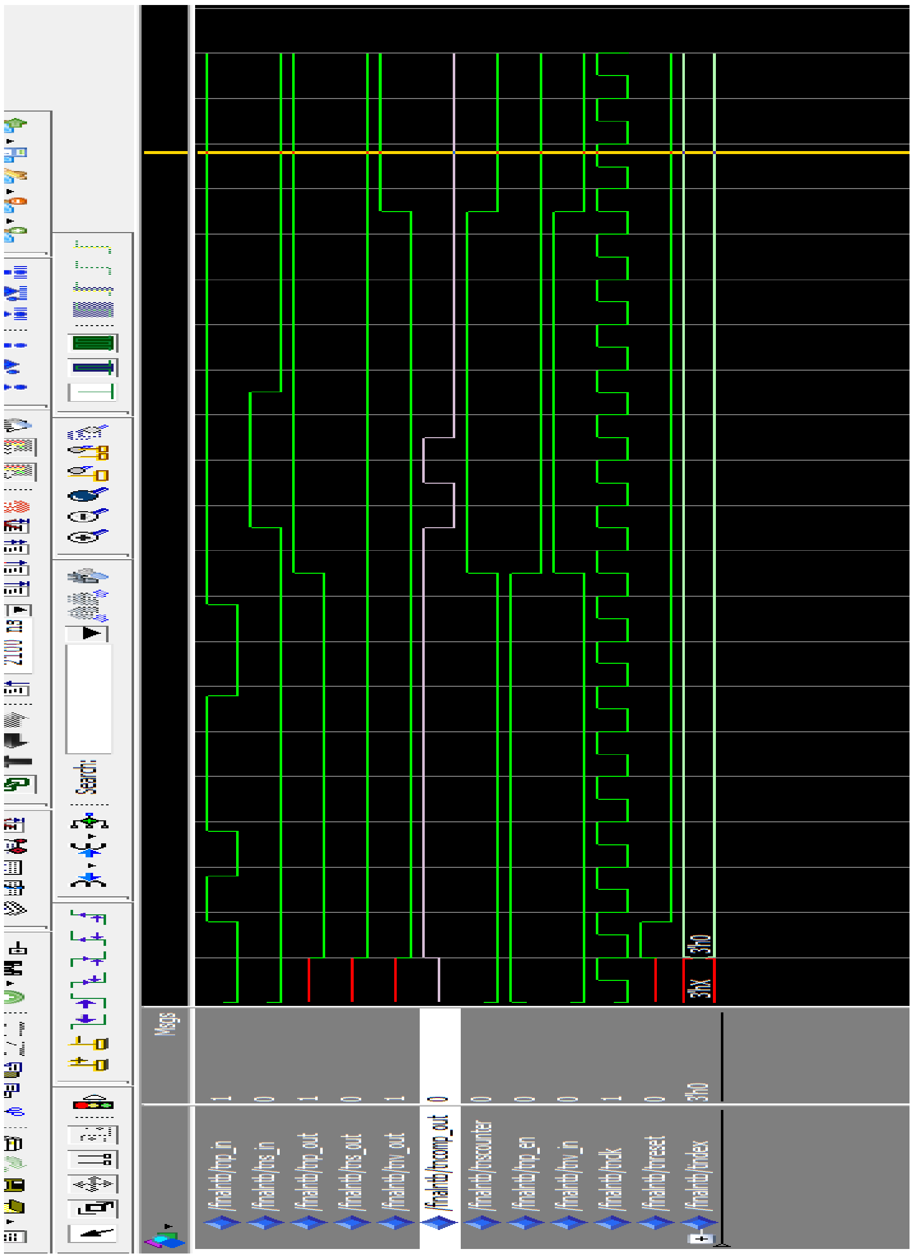


```
/finalntb/mp_in      1
/finalntb/ms_in      0
/finalntb/mp_out     0
/finalntb/ms_out     0
/finalntb/mv_out     1
/finalntb/mcomp_out  1
/finalntb/mscounter  0
/finalntb/mp_en      0
/finalntb/mv_in      0
/finalntb/mclk       1
/finalntb/mreset     0
/finalntb/index      3'h7
```

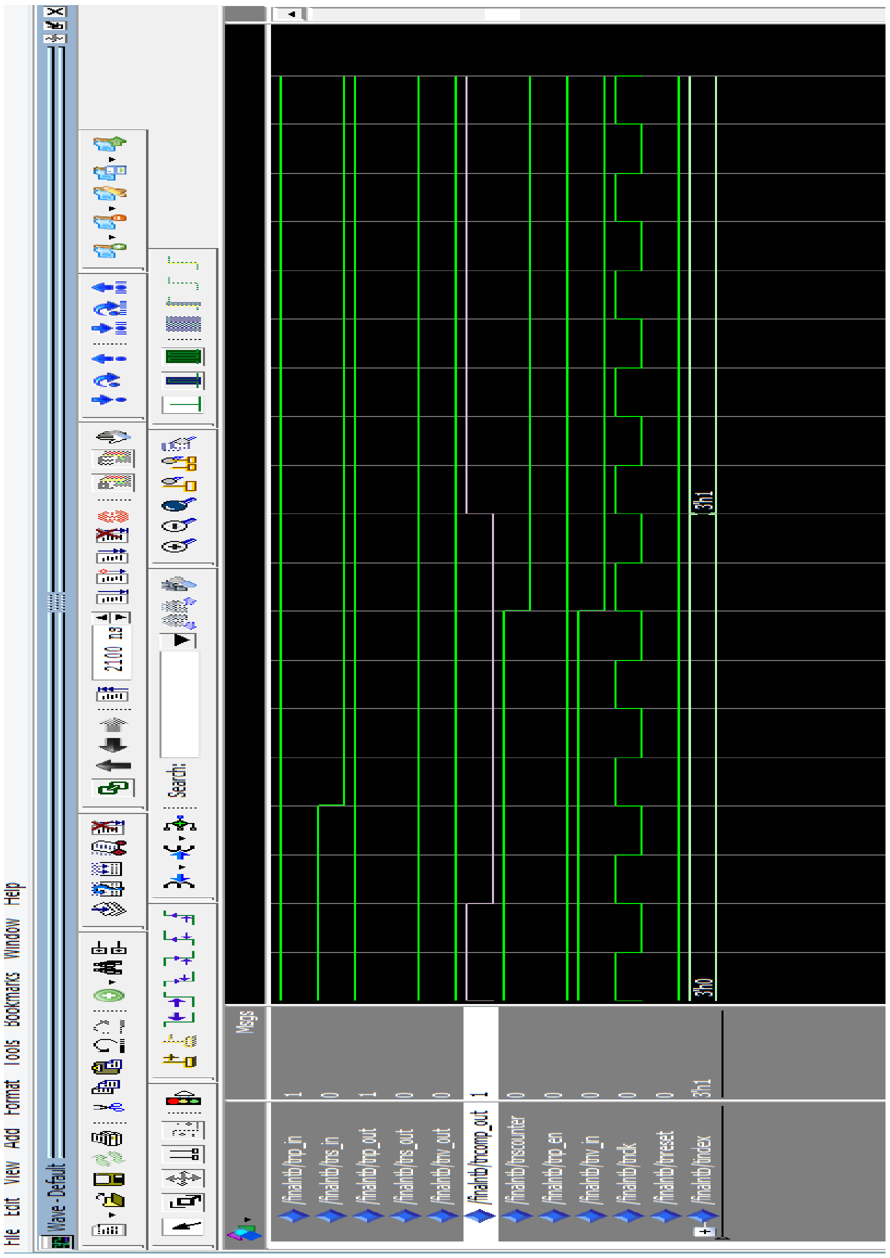
Main string: 10011101   substring: 1011   Mismatch



Main string: 10011101 substring: 00001110 Mismatch

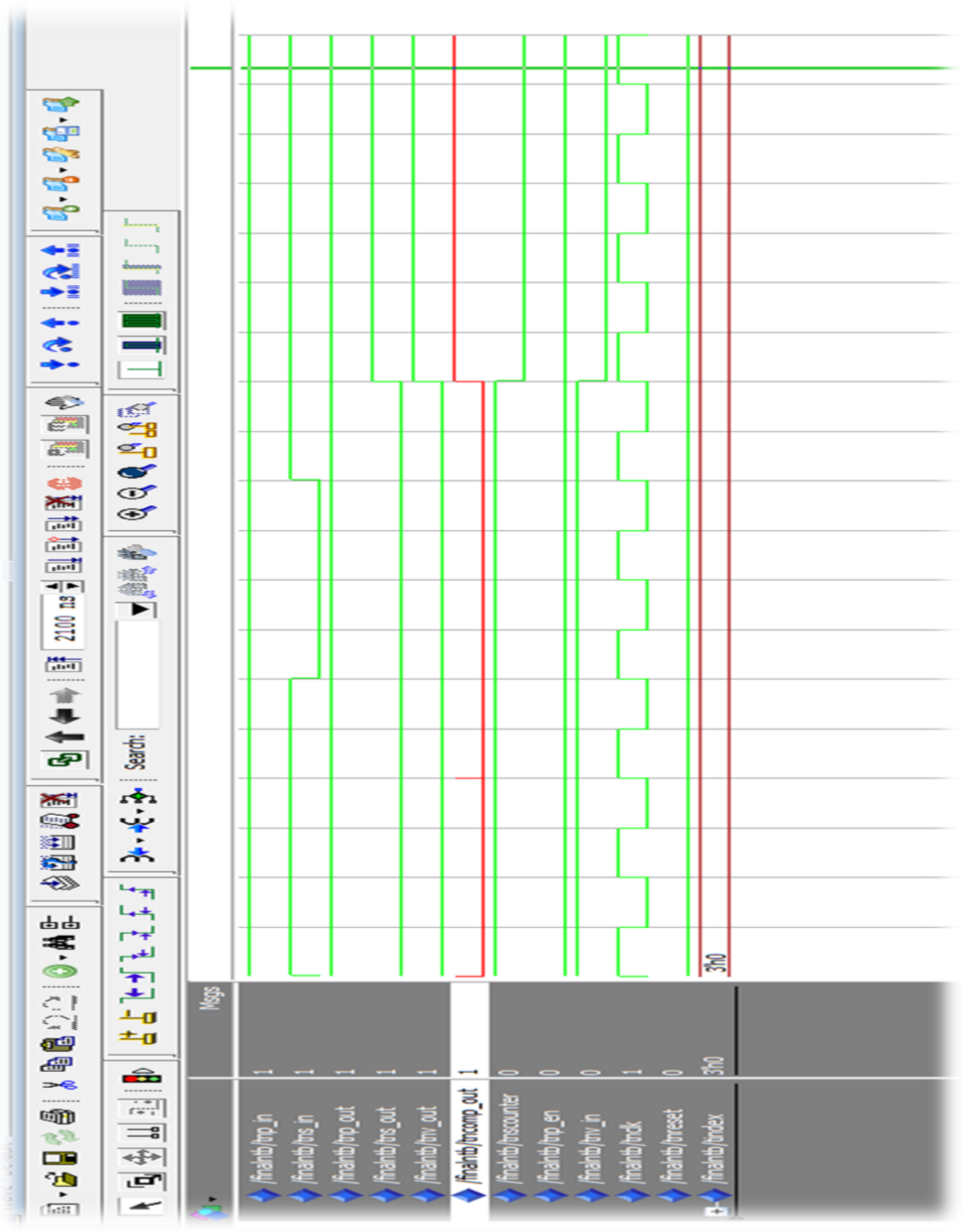


Main string: 10011101   substring: 001110   match at index 1



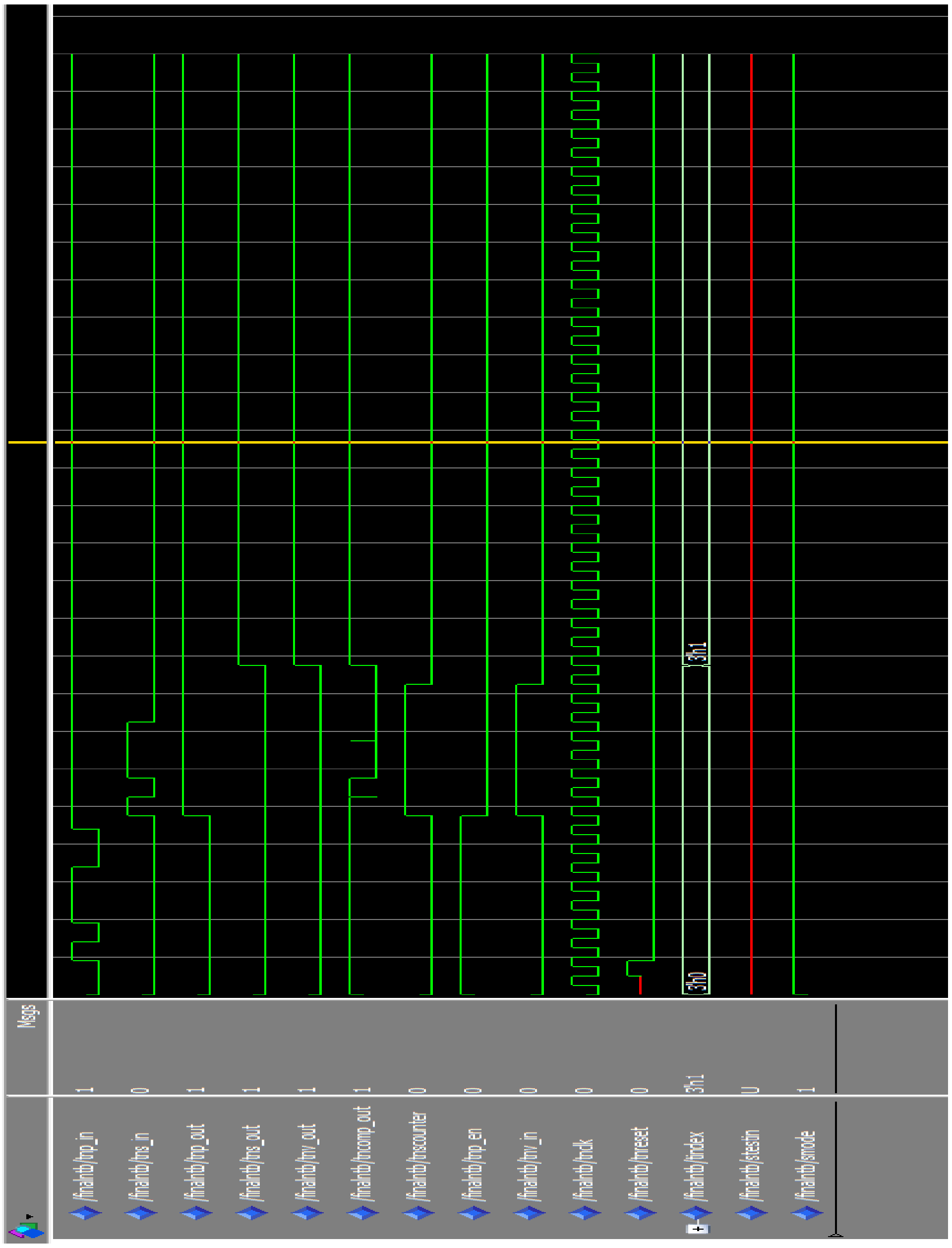
```
/finaltb/fmp_in        1
/finaltb/fms_in        0
/finaltb/fmp_out       1
/finaltb/fms_out       0
/finaltb/fmv_out       0
/finaltb/fmcomp_out    1
/finaltb/fmscounter    0
/finaltb/fmp_en        0
/finaltb/fmv_in        0
/finaltb/fmclk         0
/finaltb/fmreset       0
/finaltb/findex        3'h1
```

Main string: 10011101 substring: 10011101 match at index 0

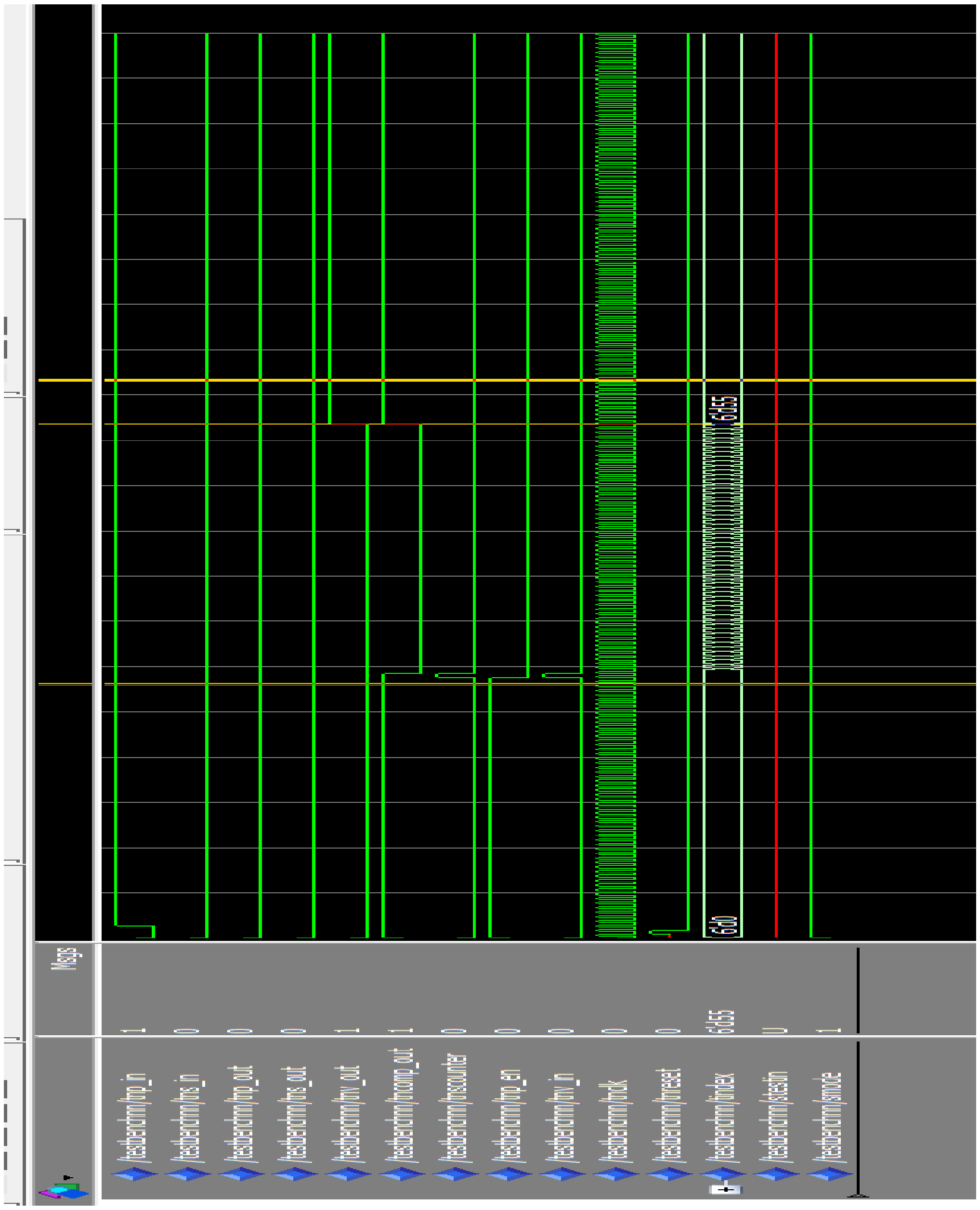


```
Msgs
/finalmtb/tnp_in        1
/finalmtb/tns_in        1
/finalmtb/tnp_out       1
/finalmtb/tns_out       1
/finalmtb/tnv_out       1
/finalmtb/tncomp_out    1
/finalmtb/tnscounter    0
/finalmtb/tnp_en        0
/finalmtb/tnv_in        0
/finalmtb/tnclk         1
/finalmtb/tnreset       0
/finalmtb/tnindex       3'h0
```

Main string: 10011101   substring: 0011101   match at index 1



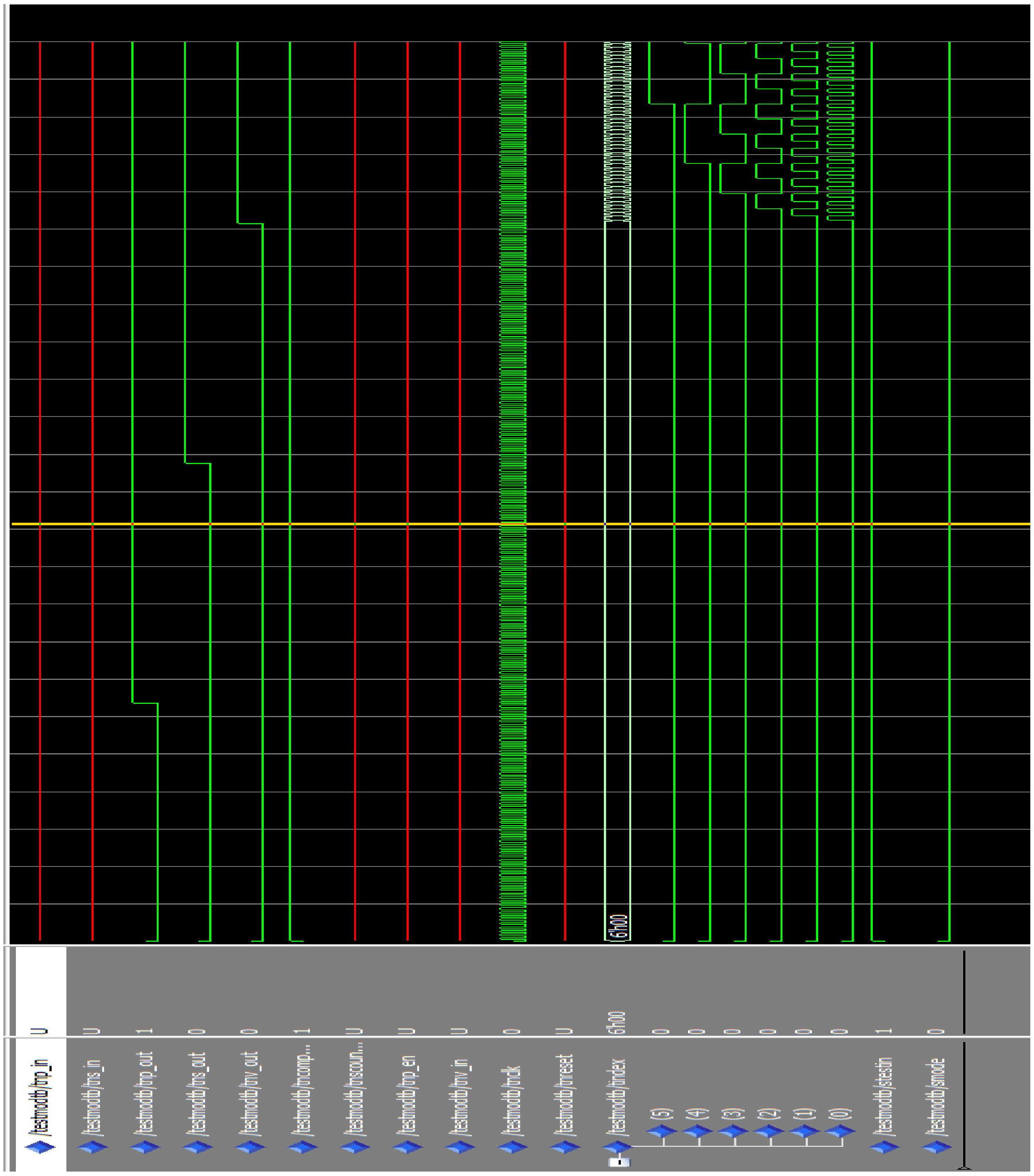
Index match at 55th position



Test bench (test mode) mode: 0

```
library ieee;
use ieee.std_logic_1164.all;
entity testmodtb is
end testmodtb;
architecture tb of testmodtb is
component testmode is
generic (n:integer:=64);
port( np_en,np_in ,ns_in,nv_in,nclk,nreset,svenable: in std_logic;
            np_out,ns_out,nv_out,ncomp_out : out std_logic;
             index:out std_logic_vector (5 downto 0);
             testin,mode:in std_logic);
end component;
signal tnp_in ,tns_in, tnp_out,tns_out,tnv_out,tncomp_out,tnsvenable: std_logic;
signal tnp_en,tnv_in: std_logic ;
signal tnclk, tnreset: std_logic;
signal tindex: std_logic_vector (5 downto 0);
signal stestin,smode: std_logic;
begin
   U1 : testmode
 port map (tnp_en,tnp_in ,tns_in,tnv_in,tnclk,tnreset,tnsvenable,tnp_out,tns_out,tnv_out,tncomp_out,
            tindex,stestin,smode);
  process
       begin
          tnclk <= '0';
          wait for 50 ns;
          tnclk <= '1';
          wait for 50 ns;
end process;
process
begin
smode<='0';
stestin<='1';
wait;
end process;
end tb;
```

Wave forms: (test mode)



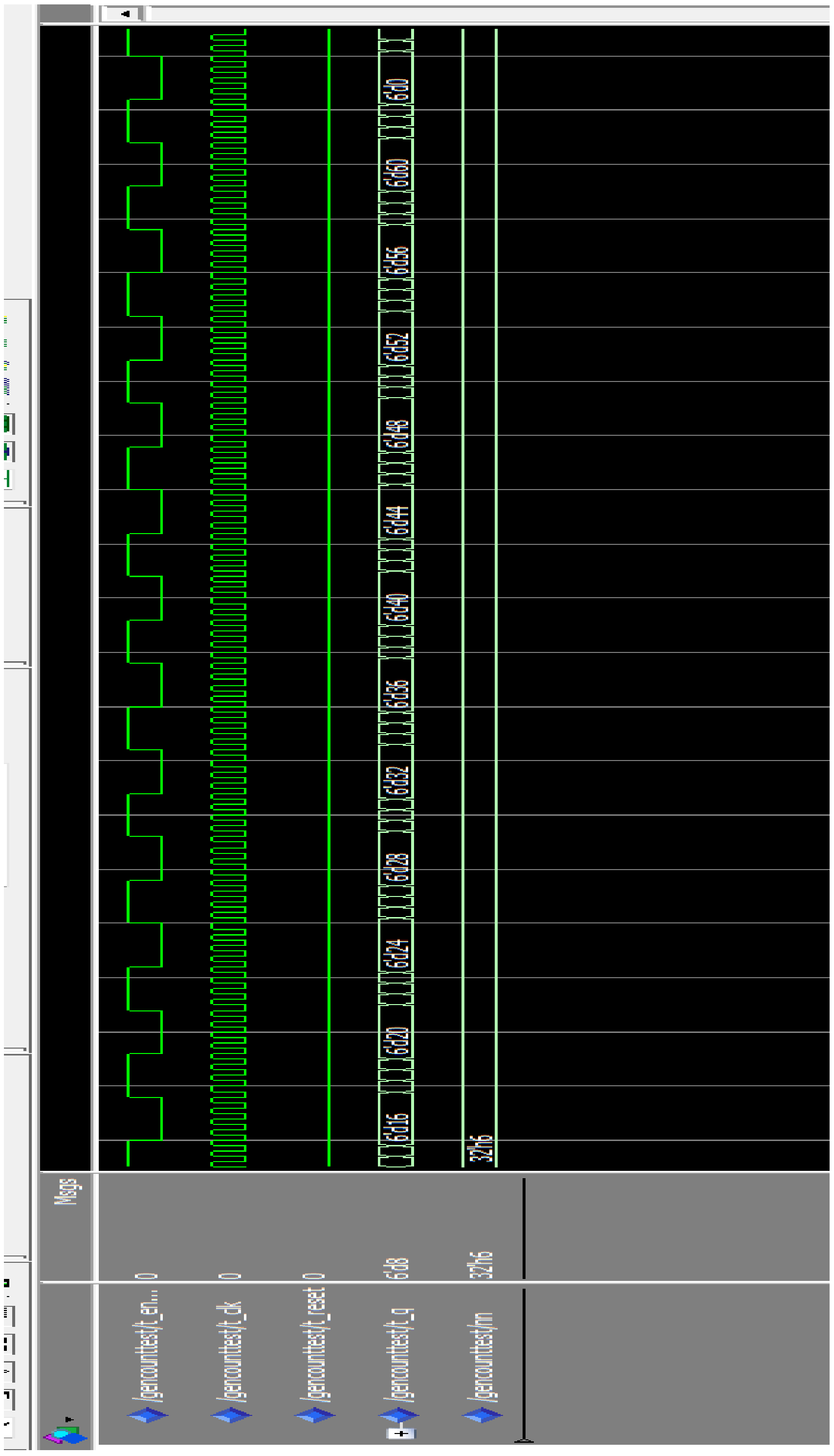
VHDL code for up-counter:

```
library ieee;
use ieee.std_logic_1164.all;

entity gencounter is
generic (n:integer:=6);
port (
enable,clk,reset: in std_logic;
q : out std_logic_vector ( n-1 downto 0));
end gencounter ;

architecture structural of gencounter is

--andgate
component andgate
   port( a, b : in std_logic;
            f : out std_logic);
end component;
--tff

component tff
port (t,clk,reset: in std_logic;
tq : out std_logic);
end component;
signal sq: std_logic_vector (n-1 downto 0);
signal sa: std_logic_vector (n downto 0);
begin
gencounter1: for i in 0 to n-1 generate
begin
sa(0)<=enable;
tff1: tff port map(sa(i),clk,reset,sq(i));
andgate1: andgate port map (sa(i),sq(i),sa(i+1));
end generate gencounter1;
q<=sq;
end structural;
```

Test bench for Up-counter:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity gencounttest is
end gencounttest;

architecture test of gencounttest is

component gencounter
generic (n:integer:=6);
port (
enable,clk,reset: in std_logic;
q : out std_logic_vector ( n-1 downto 0));
end component ;

constant nn:integer:=6;

signal t_enable, t_clk , t_reset      : std_logic := '0';
signal t_q : std_logic_vector (nn-1 downto 0);


begin

counter12 : gencounter generic map(nn)
            port map (t_enable, t_clk, t_reset, t_q);


          t_clk <= not t_clk after 10 ns;
          --t_enable <='1';
          t_enable <= not t_enable after 80 ns;

  end test;
```

Wave forms (up counter):



VHDL code for T-flip flop:

```
library ieee;
use ieee.std_logic_1164.all;
entity tff is
port (
t,clk,reset: in std_logic;
tq : out std_logic);
end tff;
architecture struct of tff is
---component dff
component dff
port (
d,clk,reset: in std_logic;
q : out std_logic);
end component;
--component xor
component xorgate
port(l,m:in std_logic;
     n:out std_logic);
end component;
signal xorout,dffout: std_logic;
begin
dff1: dff port map (xorout,clk,reset,dffout);
xorgate1: xorgate port map (t,dffout,xorout);
tq<=dffout;
end struct;
```

Test bench (T-ff):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity tff_test is
end tff_test;
architecture testdff of tff_test is
component tff is
port (t,clk,reset: in std_logic;
tq : out std_logic);
end component;
signal t_clk, t_reset, t_d   : std_logic := '0';
signal t_q  : std_logic:='0';
begin
   tff1 : tff port map (t_d, t_clk, t_reset, t_q);
      t_clk <= not t_clk after 10 ns;
      t_d<= not t_d after 20 ns;
 end testdff;
```

VHDL code for D- flip flop:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity dff is
  port(
     d,clk,reset : in STD_LOGIC;
     q : out STD_LOGIC
     );
end dff;
architecture d_flip_flop_arc of dff is
  signal dout_i : STD_LOGIC := '0';
begin
      dff : process (clk,reset) is
begin
    if (reset='1') then
        dout_i <= '0';
    elsif (rising_edge (clk)) then
        dout_i <= d after 373.45 ps;
```

```
   end if;
end process dff;
q <= dout_i;
end d_flip_flop_arc;
```

Test bench for D flip flop:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity tb_dff is
end tb_dff;
architecture testdff of tb_dff is
component dff is
port (
d,clk,reset: in std_logic;
q : out std_logic);
end component;
signal t_clk, t_reset , t_d      : std_logic := '0';
signal t_q          : std_logic:='0';
begin
  U1 : dff port map (t_d, t_clk, t_reset, t_q);

        t_clk <= not t_clk after 10 ns;
        t_d<= not t_d after 20 ns;

 end testdff;
```

AND gate VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;

--entity declaration
entity andgate is
```

```
   port( a, b : in std_logic;
            f : out std_logic);
end andgate;

--working
architecture func of andgate is
begin
  f <= a and b after 187.5 ps;
end func;
```

Test bench:

```
library ieee;
use ieee.std_logic_1164.all;

--ENTITY DECLARATION: no inputs, no outputs
entity andGate_tb is
end andGate_tb;

architecture tb1 of andGate_tb is
   --pass xnorGate entity to the testbench as component
   component andgate
   port (a,b:in std_logic;
      f:out std_logic);
end component;

   signal  ta, tb, tf : std_logic;
begin
   --map the testbench signals to the ports of the xnorGate
   mapping: andgate port map(ta, tb, tf);

   process

   begin
      --TEST 1
      ta <= '0';
      tb <= '0';
      wait for 15 ns;

      --TEST 2
      ta <= '0';
      tb <= '1';
      wait for 15 ns;

      --TEST 3
      ta <= '1';
      tb <= '1';
      wait for 15 ns;

--TEST 4
      ta <= '1';
      tb <= '0';
      wait for 15 ns;

wait;
end process;
end tb1;
```

Mux vhdl code:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port ( a : in  std_logic;
       b : in  std_logic;
       s : in  std_logic;
       z : out std_logic
     );
end mux;
architecture behav of mux is
begin

  mux_p: process(a,b,s)
  begin
    if(s='0') then
      z <= a after 184.07 ps;
    else
      z <= b after 184.07 ps;
    end if;
  end process mux_p;

end behav;
```

Test bench:

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
entity mux_tb is
end mux_tb;
architecture beh of mux_tb is

  component mux
    port ( a : in  std_logic;
           b : in  std_logic;
           s : in  std_logic;
           z : out std_logic);
  end component mux;
-- signal and variable declaration
  signal ta : std_logic;
  signal tb : std_logic;
  signal ts : std_logic;
  signal tz : std_logic;
begin

  mux_instance: mux
  port map (
   ta,tb,ts,tz);

tb1: process
  begin
    ta <= '0';
    tb <= '0';
    ts <= '0';
    wait for 100 ns;

    ta <= '0';
    tb <= '0';
    ts <= '1';
    wait for 100 ns;
    ta <= '0';
    tb <= '1';
    ts <= '0';
    wait for 100 ns;
    ta <= '0';
    tb <= '1';
    ts <= '1';
    wait for 100 ns;
    ta <= '1';
    tb <= '0';
    ts <= '0';
    wait for 100 ns;
    ta <= '1';
    tb <= '0';
    ts <= '1';
    wait for 100 ns;
    ta <= '1';
    tb <= '1';
    ts <= '0';
    wait for 100 ns;
    ta <= '1';
    tb <= '1';
    ts <= '1';
    wait for 100 ns;

    wait;
  end process tb1;

end  beh;
```

Not gate VHDL code :

```
library ieee;
use ieee.std_logic_1164.all;

--entity declaration
entity notgate is
   port( m : in std_logic;
            n : out std_logic);
end notgate;

--working
architecture func of notgate is
begin
```

```
  n <= not m after 74.35 ps;
end func;
```

Test bench:

```
library ieee;
use ieee.std_logic_1164.all;

--ENTITY DECLARATION: no inputs, no outputs
entity notGate_tb is
end notGate_tb;

-- Describe how to test the XNOR Gate
architecture tb of notGate_tb is
  --pass xnorGate entity to the testbench as component
  component notgate
  port (m:in std_logic;
    n:out std_logic);
end component;

  signal  tm,tn : std_logic;
begin
  --map the testbench signals to the ports of the xnorGate
  mapping: notgate port map(tm, tn);

  process

  begin
    --TEST 1
    tm <= '0';

    wait for 15 ns;

    --TEST 2
    tm <= '1';

    wait for 15 ns;

    --TEST 3
    tm <= '0';
```

```
      wait for 15 ns;

--TEST 4
      tm <= '1';

      wait for 15 ns;

wait;
end process;
end tb;
```

OR gate VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;


entity orgate is
port(
  a,b:in std_logic;
  c:out std_logic
);
end entity orgate;

architecture orgate_behav of orgate is
begin

orgate_process:process(a,b)
begin

c<= a or b after 188.75 ps;

end process orgate_process;

end orgate_behav;
```

Test bench (or) :

```
library ieee;
use ieee.std_logic_1164.all;

--ENTITY DECLARATION: no inputs, no outputs
```

```
entity orGate_tb is
end orGate_tb;

-- Describe how to test the XNOR Gate
architecture tb1 of orGate_tb is
   --pass xnorGate entity to the testbench as component
   component orgate
   port (a,b:in std_logic;
      c:out std_logic);
end component;
signal  ta, tb, tc : std_logic;
begin
   --map the testbench signals to the ports of the xnorGate
   mapping: orgate port map(ta, tb, tc);
 process
        begin
      --TEST 1
      ta <= '0';
      tb <= '0';
      wait for 15 ns;

      --TEST 2
      ta <= '0';
      tb <= '1';
      wait for 15 ns;

      --TEST 3
      ta <= '1';
      tb <= '1';
      wait for 15 ns;

--TEST 4
      ta <= '1';
      tb <= '0';
      wait for 15 ns;

wait;
end process;
end tb1;
```

Xnor gate VHDL code:

```
library ieee;
use ieee.std_logic_1164.all;

entity xnorgate is
port(p,q:in std_logic;
     r:out std_logic);
end entity xnorgate;

architecture behav of xnorgate is
begin

xnorgate_process:process (p,q)
begin

r<= p xnor q after 258.34 ps;

end process xnorgate_process;
end behav;
```

Test bench (Xnor):

```
library ieee;
use ieee.std_logic_1164.all;

--ENTITY DECLARATION: no inputs, no outputs
entity xnorGate_tb is
end xnorGate_tb;

-- Describe how to test the XNOR Gate
architecture tb of xnorGate_tb is
  --pass xnorGate entity to the testbench as component
  component xnorgate
  port (p,q:in std_logic;
     r:out std_logic);
end component;

  signal  tp, tq, tr : std_logic;
begin
  --map the testbench signals to the ports of the xnorGate
```

```
   mapping: xnorgate port map(tp, tq, tr);

   process

   begin
     --TEST 1
     tp <= '0';
     tq <= '0';
     wait for 15 ns;

     --TEST 2
     tp <= '0';
     tq <= '1';
     wait for 15 ns;

     --TEST 3
     tp <= '1';
     tq <= '1';
     wait for 15 ns;

  --TEST 4
     tp <= '1';
     tq <= '0';
     wait for 15 ns;

  wait;
  end process;
  end tb;
```

VHDL code for XOR gate:

```
library ieee;
use ieee.std_logic_1164.all;

entity xorgate is
port(l,m:in std_logic;
     n:out std_logic);
end entity xorgate;

architecture behav of xorgate is
begin
```

```
process (l,m)
begin
n<=l xor m after 226.08 ps;
end process;
end behav;
```

Test bench (xor):

```
library ieee;
use ieee.std_logic_1164.all;

--ENTITY DECLARATION: no inputs, no outputs
entity xorGate_tb is
end xorGate_tb;

-- Describe how to test the XNOR Gate
architecture tb1 of xorGate_tb is
  --pass xnorGate entity to the testbench as component
  component xorgate
  port (l,m:in std_logic;
    n:out std_logic);
end component;

  signal  ta, tb, tf : std_logic;
begin
  --map the testbench signals to the ports of the xnorGate
  mapping: xorgate port map(ta, tb, tf);

  process

  begin
    --TEST 1
    ta <= '0';
    tb <= '0';
    wait for 15 ns;

    --TEST 2
    ta <= '0';
    tb <= '1';
    wait for 15 ns;
    --TEST 3
```

```
      ta <= '1';
      tb <= '1';
      wait for 15 ns;
       --TEST 4
      ta <= '1';
      tb <= '0';
      wait for 15 ns;
      wait;
end process;
end tb1;
```

VHDL code for NOR gate:

```
library ieee;
use ieee.std_logic_1164.all;
--entity declaration
entity norgate is
   port( x, y : in std_logic;
            z : out std_logic);
end norgate;
--working
architecture funcnor of norgate is
begin
  z <= x nor y after 96.47 ps ;
end funcnor;
```

Test bench (nor gate):

```
library ieee;
use ieee.std_logic_1164.all;--ENTITY DECLARATION: no inputs, no outputs
entity norGate_tb is
end norGate_tb;
      -- Describe how to test the NOR Gate
architecture tb1 of norGate_tb is
   --pass norGate entity to the testbench as component
   component norgate
   port (x,y:in std_logic;
      z:out std_logic);
end component;
signal  ta, tb, tf : std_logic;
begin
   --map the testbench signals to the ports of the xnorGate
   mapping: norgate port map(ta, tb, tf);
```

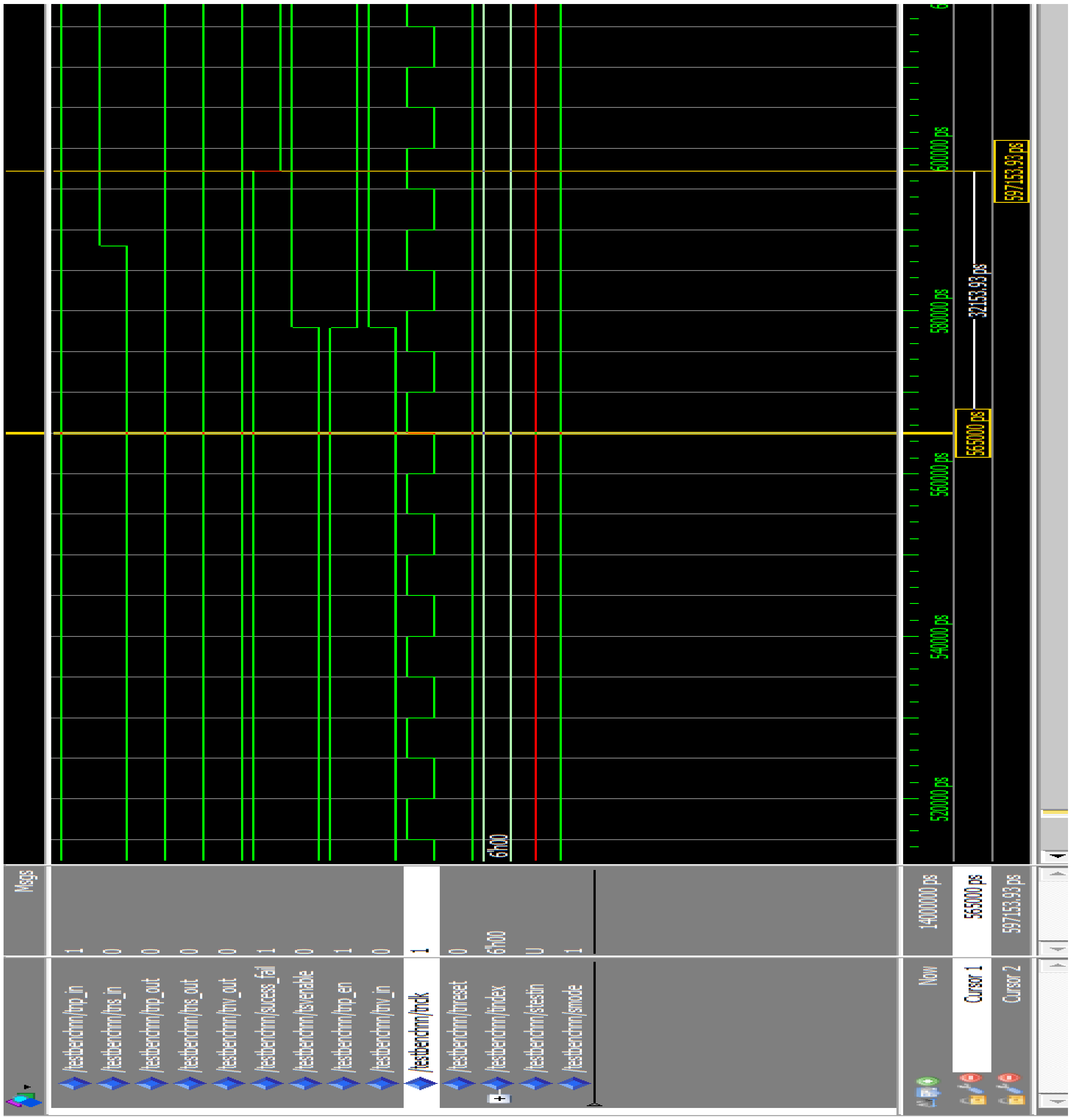
```
process
     begin
      --TEST 1
      ta <= '0';
      tb <= '0';
      wait for 15 ns;
     --TEST 2
      ta <= '0';
      tb <= '1';
      wait for 15 ns;

      --TEST 3
      ta <= '1';
      tb <= '1';
      wait for 15 ns;

--TEST 4
      ta <= '1';
      tb <= '0';
      wait for 15 ns;
wait;
end process;
end tb1;
```

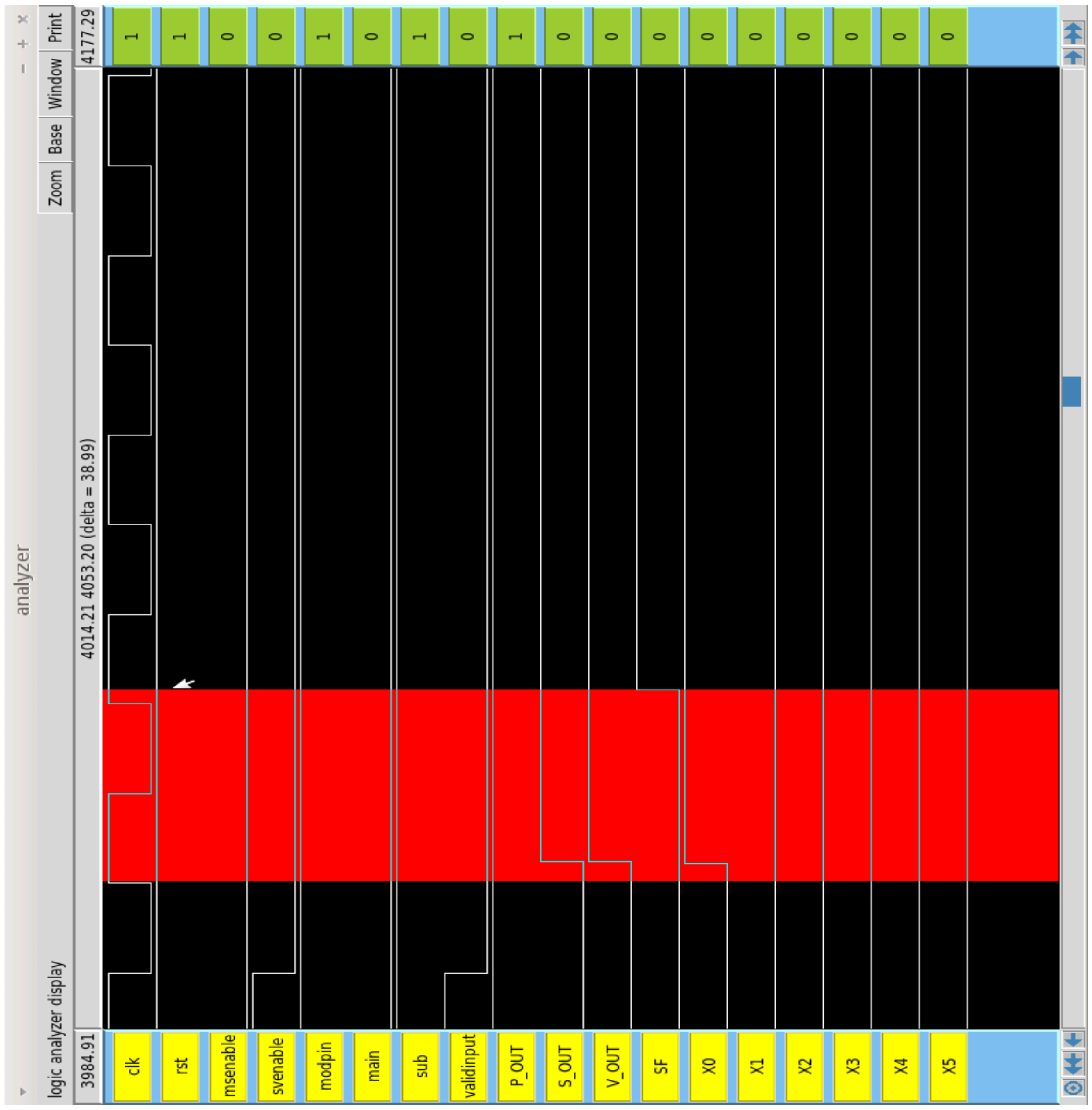
THROUGHPUT, CLOCK FREQUENCY and OVERALL TIMING

- The initial output becomes ready at the output after the N+L clock cycles once all the inputs are loaded.
- LATENCY: N+L clock cycles (best case).
- LATENCY: 2N clock cycles (worst case).
- Throughput: 1/ (N+L) (best case).
- Throughput: 1/ (2N) (worst case).
- Circuit runs perfectly with a clock cycle of 36 ns which gives the clock frequency to 27.7 MHz
- Giving a margin of 10% which is 3.6 ns, the optimal clock frequency is set to 25.25 MHz

COMPARISION OF IRSIM AND VHDL SIMULATIONS:

The worst delay for the VHDL simulations have come around 32.15 ns. The IRSIM simulations have worst case delay of 38.8ns. This is because there are no contact and wire capacitances in VHDL. So, delay in VHDL is less compared to spice.




analyzer
logic analyzer display
Zoom
Base
Window
Print
3984.91
4014.21 4053.20 (delta = 38.99)
4177.29
clk 1
rst 1
msenable 0
svenable 0
modpin 1
main 0
sub 1
validinput 0
P_OUT 1
S_OUT 0
V_OUT 0
SF 0
X0 0
X1 0
X2 0
X3 0
X4 0
X5 0

Time line till report 2 followed by timeline for final report

DATE	Asrith Reddy	Sai Chaitanya
• Team met and discussed the further proceeding and drawbacks of the first report. • Other communication was done using emails and phone calls. We used online storage to share the documents.		
26th Oct	Started working on components required for single bit slice for MAGIC	Started working on components for N-bit VHDL code
27th Oct	Started working on the MAGIC Bit Slice layout	Started working on the N-bit VHDL
28th Oct	Finished designing layout of single bit slice and submitted the magic layout with extracted files to Chaitanya by evening	Finished the N-bit VHDL code along with the control logic and submitted the code to Asrith by evening
29th Oct	Forced the values and simulated to verify the code. Worked on the VHDL test bench for the code provided by Chaitanya	Verified the correctness of the bit slice. Started working on the extraction of individual components from MAGIC layouts and calculating the delays.
30th Oct	Worked on the test bench for the code and simulated and verified the correctness of the code without delays. Received the Delays tabulated in the night	Started working on the extraction of individual components and calculating the delays. Tabulated delays were submitted to Asrith by night.

31st Oct	Worked on simulating test bench with the back annotated timing delays given by Chaitanya the previous day. Submitted the waveforms to Chaitanya.	Verification of the simulation waveforms submitted by Asrith. Came out to be correct. Started designing the Control logic components in MAGIC
1st November	Received the control logic layouts from Chaitanya by noon. Extracted them, verified the spice simulations. Started layout of two slices and verified the circuit one again from MAGIC extraction using spice simulation and waveforms.	Finished working on control logic components and submitted the layouts to asrith by noon. Started working on the layout diagrams for documentation
2nd November	• Meeting in the morning and discussed on the status of the project. • Discussed on the pin placements revision done in first report. • Planned on the placement and routing for the chip in the pad frame • Discussed on the clock tree and placement of buffers and control logic. • Test strategy revised	
	Each worked individually by laying out and estimating the throughput rate, timing of the circuit and the estimated clock frequency.	
3rd November	Met in the morning and reviewed the both types of placement and routing and decided on the better strategy.	
4th November Deadline in the midnight	Started working on the documentation. Each of us individually documented what we did in separate documents. We verified the documentation done by the other person. Final documentation merging and aesthetics were done by Asrith Reddy and Chaitanya took the responsibility of printing out the Document.	

TIMELINE FOR THE FINAL REPORT		
6th November	Started designing the control logic for the circuit	Started simulating the circuits by connecting the bit slices
9th November	Control logic is designed and submitted	Simulation of the 4 bit slice is done successfully.
10th November	Discussed the strategy for planning and layout	
11th November	Placed the N-bit outside the pad frame separately.	
12th November	Simulated the N-bit without the pad frame.	
13th November	Started layout inside the pad frame.	
14th November	The pad frame connections are made with the extracted sim files simulated.	

15th November	Documentation of the final report was started with work shared according to the work done.
16th November	Deadline at midnight. Final touchups to the documentation.