String recognizer

Devesh Kumar, 16D070044

March 19, 2018

1 Overview of the experiment

The aim of this experiment is to write a VHDl code for an string recognizer which gives the output 1 if the string contains **Bomb, Gun, Terror, Knife**.

2 Design/algorithm

To implement the recognizer we use the finite states and the system should remenber the old state. For this we need memory so we use D flip flop. D flip-flop simply returns the previous value at next clock cycle. When a right value comes we will shift the state to next.

2.1 Gun

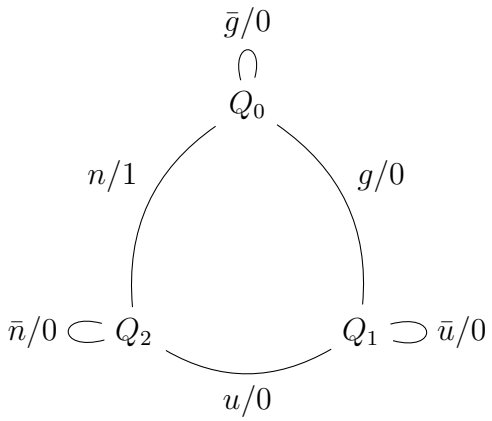


Figure 1: State Diagram gun

If g comes then we can move from first state to next similarly if u comes then we can further move to next state. If g does not come we willbe in that state only.

2.2 Bomb

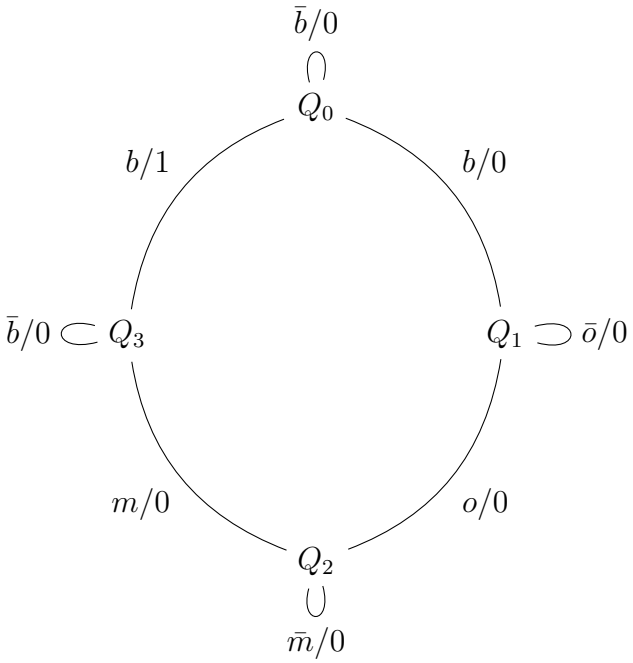


Figure 2: State Diagram of bomb

If b comes then we can move from first state to next similarly if o comes then we can further move to next state. If b does not come we willbe in that state only. The output will be one only if b comes and we are in bom state

2.3 knife

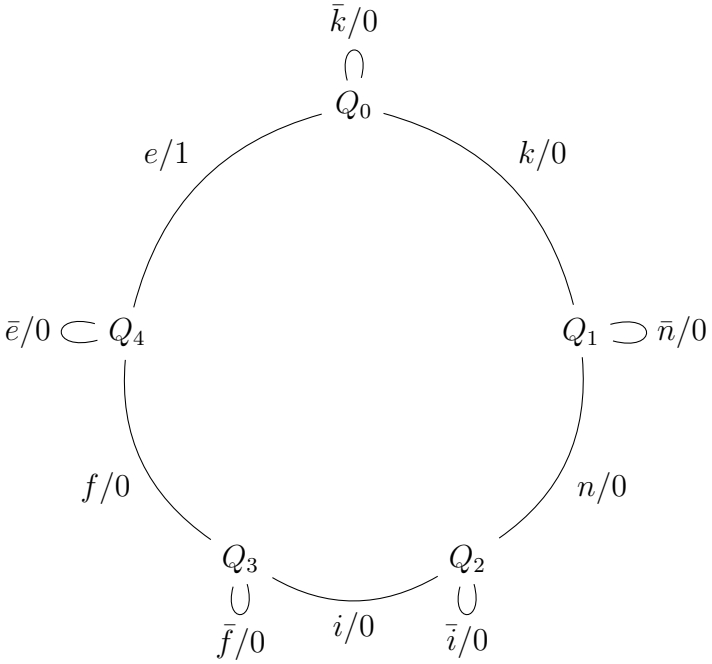


Figure 3: State Diagram of knife

If t comes then we can move from first state to next similarly if e comes then we can further move to next state. If t does not come we will be in that state only. The output will be one only if r comes and we are in terro state.

2.4 Terror

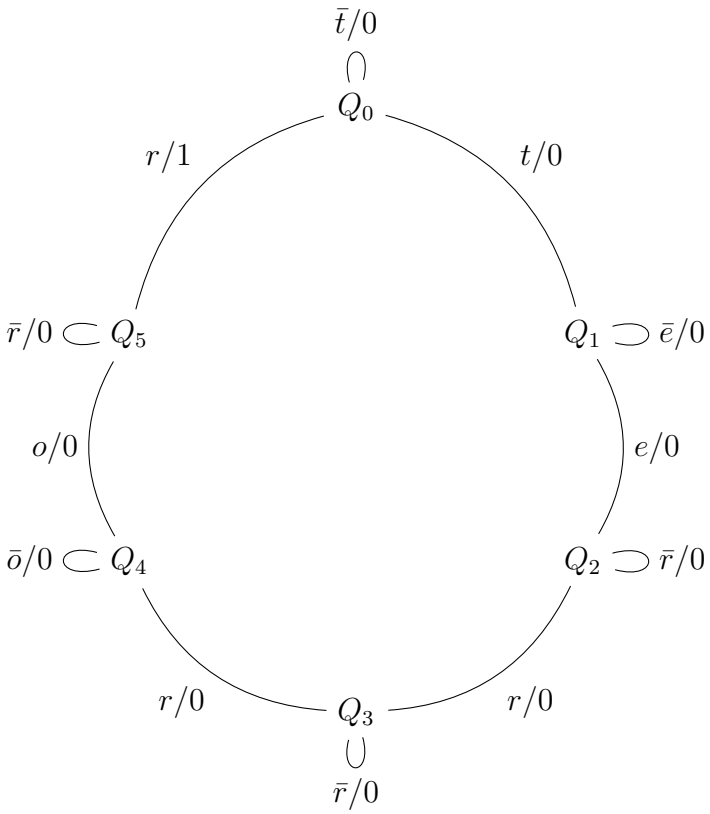


Figure 4: State Diagram of terror

3 VHDL code (well commented)

```
-------------------mux_bit---------------
use  std.standard.all; library ieee;
use ieee.std_logic_1164.all;
entity MUX_bit is
     port(
      c,d,Y : in std_logic;
                c0 : out std_logic
    );
                  end entity;
architecture behave of  MUX_bit is
signal y0: std_logic;
begin
y0 <= NOT(Y);
c0 <= ((c AND Y) OR ( d AND y0));
end architecture behave;
-------------mux---------------------
```

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

entity MUX is
     port(
      d0,d1,d2,d3,d4,f0,f1,f2,f3,f4 : in std_logic;
                y1 : in std_logic;
      g0,g1,g2,g3,g4 : out std_logic
    );
         end entity;
architecture behave of  MUX is
component  MUX_bit is
port(
      c,d,y : in std_logic;
                c0 : out std_logic
    );
         end component MUX_bit;

begin
X1: MUX_bit port map(c=>d0,d=>f0,y=>y1,c0=>g0);
X2: MUX_bit port map(c=>d1,d=>f1,y=>y1,c0=>g1);
X3: MUX_bit port map(c=>d2,d=>f2,y=>y1,c0=>g2);
X4: MUX_bit port map(c=>d3,d=>f3,y=>y1,c0=>g3);
X5: MUX_bit port map(c=>d4,d=>f4,y=>y1,c0=>g4);

end architecture behave;
```

3.1 Terror

To implement terror I used 6 states.If reset comes out to be one then i will set Q0 and q1,q2 to 0,0,0 respectively. I choose which letter I have to compare depending on which state I am at .For example if I am at initial state then i will compare the input with T. For this I used multiple mux. then I used comparater which compares the selected string with the input. If comparater output is one then I will move to the next state.

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

entity TERROR is
port (X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
clk,reset: in std_logic);
end entity TERROR ;
architecture behave of TERROR is
component D_FF is
port (D, CLK: in std_logic;  Q: out std_logic);
end component;
component MUX is
 port(
      d0,d1,d2,d3,d4,f0,f1,f2,f3,f4 : in std_logic;
          y1 : in std_logic;
```

```
      g0,g1,g2,g3,g4 : out std_logic
    );
         end component;
component  MUX_bit is
port(
      c,d,y : in std_logic;
                c0 : out std_logic
    );
         end component MUX_bit;
component comparator is
        port (
                a0,a1,a2,a3,a4,b0,b1,b2,b3,b4: in std_logic;
                 c: out std_logic
        );
end component comparator;
signal s10,s11,s12,s13,s14,s20,s21,s22,s23,s24 ,
       s30,s31,s32,s33,s34,s40,s41,s42,s43,s44,q0,q1,q2,
       rq1,rq0,rq2,nq0,nq1,nq2,stemp0,stemp1,stemp2,outq,tt,tl: std_logic;
        --000->null
        --001-> T
        --010-> TE
        --011-> TER
        --100-> TERR
        --101-> TERRO
begin
        --getting the value of rq0 from nqo
 filp0: D_FF port map ( d=>nq0, CLK=> clk, q=> rq0);
 filp1: D_FF port map ( d=>nq1, CLK=> clk, q=> rq1);
 filp2: D_FF port map ( d=>nq2, CLK=> clk, q=> rq2);
 --rq0,rq1,rq2 stores the value before the reset
 reset0: MUX_bit port map('0',rq0,reset,q0);
 reset1: MUX_bit port map('0',rq1,reset,q1);
 reset2: MUX_bit port map('0',rq2,reset,q2);
 mux_1:  mux port map('0','0','1','0','1', '1','0','1','0','0', q0, s10,s11,s12,s13,s14 );
 --e,t depending on q0 q1 q2 select the letter to be comparedand store it in sig3
 mux_2:  mux port map('1','0','0','1','0', '0','1','1','1','1', q0, s20,s21,s22,s23,s24 ); -
 mux_3:  mux port map('1','0','0','1','0', s10,s11,s12,s13,s14, q1, s30,s31,s32,s33,s34) ;
  -- r or(t,e depending on q0) depending on s1
 mux_4:  mux port map(s20,s21,s22,s23,s24, s30,s31,s32,s33,s34, q2, s40,s41,s42,s43,s44) ;
 -- if s2 is 1 then last two bits
 comp_1: comparator port map(s40,s41,s42,s43,s44 ,X4,X3,X2,X1,X0, outq);
 --compare input  s30 is te most significant
 w<= (not(q1) )and q0 and outq and q2;
stemp0<= not(q0);
stemp1 <= (q0 XOR q1) and (not q2); -- finding next value of state
tt <= (((not q2)and q1) and q0) ;
tl <=((q2 and (not q1))and (not q0));-- only 1 if 01
stemp2 <= tt or tl;
m1: MUX_bit port map  (stemp0,q0,outq, nq0);--updating value of nq0
m2: MUX_bit port map (stemp1,q1,outq,nq1);--updating value of nq1
```

```
m3: MUX_bit port map (stemp2,q2,outq,nq2);--updating value of nq2
end architecture behave;
```

3.2 Gun

To implement Gun I used 3 states.If reset comes out to be one then i will set Q0 and q1 to 0,0 respectively. I choose which letter I have to compare depending on which state I am at .For example if I am at initial state then I will compare the input with G. for this I used multiple mux. then I used comparater which compares the selected string with the input. if comparater output is one then I will move to the next state.

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

entity GUN is
port (X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
clk,reset: in std_logic);
end entity GUN ;
architecture behave of GUN is
component D_FF is
port (D, CLK: in std_logic;  Q: out std_logic);
end component;
component MUX is
 port(
      d0,d1,d2,d3,d4,f0,f1,f2,f3,f4 : in std_logic;
                y1 : in std_logic;
      g0,g1,g2,g3,g4 : out std_logic
    );
         end component;
component  MUX_bit is
port(
      c,d,y : in std_logic;
                c0 : out std_logic
    );
         end component MUX_bit;
component comparator is
        port (
                a0,a1,a2,a3,a4,b0,b1,b2,b3,b4: in std_logic;
                 c: out std_logic

        );
end component comparator;
signal s10,s11,s12,s13,s14,s20,s21,s22,s23,s24,
stemp0,stemp1,rq0,rq1 ,q0,q1,nq0,nq1,outq: std_logic;
        --00->null
        --01-> g
        --10-> gu
begin
 filp0: D_FF port map ( d=>nq0, CLK=> clk, q=> rq0);--getting the value of rq0 from nqo
 filp1: D_FF port map ( d=>nq1, CLK=> clk, q=> rq1);
 reset0: MUX_bit port map('0',rq0,reset,q0);
```

```
 reset1: MUX_bit port map('0',rq1,reset,q1);

mux_1:  mux port map('1','0','1','0','1',  '0','0','1','1','1',  q0, s10,s11,s12,s13,s14 );
   --u g --depending on s0 s1 select the letter to be comparedand store it in sig3
mux_2:  mux port map('0','1','1','1','0',  s10,s11,s12,s13,s14,  q1, s20,s21,s22,s23,s24 );
com: comparator port map (s20,s21,s22,s23,s24 ,X4,X3,X2,X1,X0, outq);
--compare input  s30 is te most significant
w<= not(q0) and q1 and outq;
stemp0<= (not(q1)) and (not(q0));--adding 1 to q1q0 but  adding 1 to 10 should result in 00
stemp1 <= (not(q1)) and q0;  -- only 1 if 01
m1: MUX_bit port map (stemp0,q0,outq, nq0);--updating value of nq0
m2: MUX_bit port map (stemp1,q1,outq,nq1);--updating value of nq1

end architecture behave;
```

3.3 Bomb

To implement Bomb I used 4 states.If reset comes out to be one then i will set Q0 and q1 to 0,0 respectively. I choose which letter I have to compare depending on which state I am at .For example if I am at initial state then i will compare the input with b. For this I used multiple mux. then I used comparater which compares the selected string with the input. If comparater output is one then I will move to the next state.

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

entity BOMB is
port (X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
clk,reset: in std_logic);
end entity BOMB ;
architecture behave of BOMB is
component D_FF is
port (D, CLK: in std_logic;  Q: out std_logic);
end component;
component MUX is
 port(
      d0,d1,d2,d3,d4,f0,f1,f2,f3,f4 : in std_logic;
                y1 : in std_logic;
      g0,g1,g2,g3,g4 : out std_logic
    );
         end component;
component  MUX_bit is
port(
      c,d,y : in std_logic;
                c0 : out std_logic
    );
         end component MUX_bit;
component comparator is
        port (
                a0,a1,a2,a3,a4,b0,b1,b2,b3,b4: in std_logic;
                 c: out std_logic
```

```
        );
end component comparator;
signal s10,s11,s12,s13,s14,s20,
s21,s22,s23,s24,s30,s31,s32,s33,s34,
rq0,rq1 ,stemp0,stemp1,q0,q1,nq0,nq1,outq: std_logic;
begin
        --00->null
        --01-> b
        --10-> bo
        --11-> bom

 reset0: MUX_bit port map('0',rq0,reset,q0);
 reset1: MUX_bit port map('0',rq1,reset,q1);

mux_1:  MUX port map( '0','1','1','1','1',  '0','0','0','1','0', q0, s10,s11,s12,s13,s14 );
   --o, b--depending on s0 s1 select the letter to be comparedand store it in sig3
mux_2:  MUX port map( '0','0','0','1','0' ,'0','1','1','0','1', q0, s20,s21,s22,s23,s24 );
   -- b m
 mux_3:  MUX port map(s20,s21,s22,s23,s24, s10,s11,s12,s13,s14, q1, s30,s31,s32,s33,s34) ;
comp: comparator port map (s30,s31,s32,s33,s34 , X4,X3,X2,X1,X0, outq);
--compare input  s30 is te most significant
w<= q0 and q1 and outq;
stemp0<=not q0;--adding 1 to q1q0
stemp1 <= q0 xor q1;
m1: MUX_bit port map (stemp0,q0,outq, nq0);
--updating value of nq0
m2: MUX_bit port map (stemp1,q1,outq,nq1);
--updating value of nq1
 filp0: D_FF port map ( d=>nq0, CLK=> clk, q=> rq0);
 --getting the value of rq0 from nqo
 filp1: D_FF port map ( d=>nq1, CLK=> clk, q=> rq1);
end architecture behave;
```

3.4 Knife

To implement knife I used 5 states.If reset comes out to be one then i will set Q0 and q1 q2to 0,0,0 respectively. I choose which letter I have to compare depending on which state I am at .For example if I am at initial state then i will compare the input with k. For this I used multiple mux. then I used comparater which compares the selected string with the input. If comparater output is one then I will move to the next state.

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

entity KNIFE is
port (X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
clk,reset: in std_logic);
end entity KNIFE ;
architecture behave of KNIFE is
```

```
component D_FF is
port (D, CLK: in std_logic;  Q: out std_logic);
end component;
component MUX is
 port(
      d0,d1,d2,d3,d4,f0,f1,f2,f3,f4 : in std_logic;
                y1 : in std_logic;
      g0,g1,g2,g3,g4 : out std_logic
    );
         end component;
component  MUX_bit is
port(
      c,d,y : in std_logic;
                c0 : out std_logic
    );
         end component MUX_bit;
component comparator is
        port (
                a0,a1,a2,a3,a4,b0,b1,b2,b3,b4: in std_logic;
                 c: out std_logic

        );
end component comparator;
signal s10,s11,s12,s13,s14,s20,s21,s22,s23,s24 ,
       s30,s31,s32,s33,s34,s40,s41,s42,s43,s44,
       q0,q1,q2,rq0,rq2,rq1,nq0,nq1,nq2,
       stemp0,stemp1,stemp2,outq: std_logic;
        --000->null
        --001-> K
        --010-> KN
        --011-> KNI
        --100-> KNIF
begin
 filp0: D_FF port map ( d=>nq0, CLK=> clk, q=> rq0);
 --getting the value of rq0 from nqo
 filp1: D_FF port map ( d=>nq1, CLK=> clk, q=> rq1);
 filp2: D_FF port map ( d=>nq2, CLK=> clk, q=> rq2);
 reset0: MUX_bit port map('0',rq0,reset,q0);
 reset1: MUX_bit port map('0',rq1,reset,q1);
 reset2: MUX_bit port map('0',rq2,reset,q2);
 mux_1:  mux port map('0','1','1','1','0',  '0','1','0','1','1',  q0, s10,s11,s12,s13,s14 );
 --n,k--depending on q0 q1 q2 select the letter to be comparedand store it in sig3
 mux_2:  mux port map('0','0','1','1','0',  '0','1','0','0','1',  q0, s20,s21,s22,s23,s24 );
 --f, i
 mux_3:  mux port map(s20,s21,s22,s23,s24, s10,s11,s12,s13,s14, q1, s30,s31,s32,s33,s34) ;
 mux_4:  mux port map('0','0','1','0','1', s30,s31,s32,s33,s34, q2, s40,s41,s42,s43,s44) ;
 --e,s3
 com: comparator port map (s40,s41,s42,s43,s44 ,X4,X3,X2,X1,X0, outq);
 --compare input  s30 is te most significant
w<= ((not(q0) and (not(q1)))and q2 )and outq;
```

```
stemp0<= (not(q0) )and (not(q2));--adding 1 to q2q1q0
stemp1 <= (not q2) and (q0 XOR q1);
stemp2 <= (not(q2) and q0) and q1;-- only 1 if 01
m0: MUX_bit port map (stemp0,q0,outq,nq0);--updating value of nq0;
m1: MUX_bit port map (stemp1,q1,outq,nq1);--updating value of nq1
m2: MUX_bit port map (stemp2,q2,outq,nq2);--updating value of nq2
end architecture behave;
```

3.5 String Recogniser

To implement the main program I passed the inputs to all the four components ie bomb,terror, knife,gun. The final output will be one of one or more of the components have output 1 .

```
library ieee;
use ieee.std_logic_1164.all;
entity stringRecognizer is
port (reset, clk: in std_logic;
          X4,X3,X2,X1,X0: in std_logic;
          W: out std_logic
 );
end entity stringRecognizer;
architecture behave  of stringRecognizer is
component BOMB is
        port (
                X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
      clk,reset: in std_logic
        );
end component BOMB;

component TERROR is
        port (
                X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
      clk,reset: in std_logic
        );
end component TERROR ;
component KNIFE is
        port (
                X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
      clk,reset: in std_logic
        );
end component KNIFE;
component GUN is
        port (
                X4,X3,X2,X1,X0: in std_logic; W: out std_logic;
      clk,reset: in std_logic
        );
end component GUN;
signal a,b,c,d :std_logic;
begin
        aa: BOMB port map ( X4,X3,X2,X1,X0,a,clk,reset);
```

```
        K: KNIFE port map ( X4,X3,X2,X1,X0,b,clk,reset);
        T: TERROR port map (X4,X3,X2,X1,X0,c,clk,reset );
        G: GUN port map ( X4,X3,X2,X1,X0,d,clk,reset);
        w<= a OR b or c or d;
end architecture behave ;
```

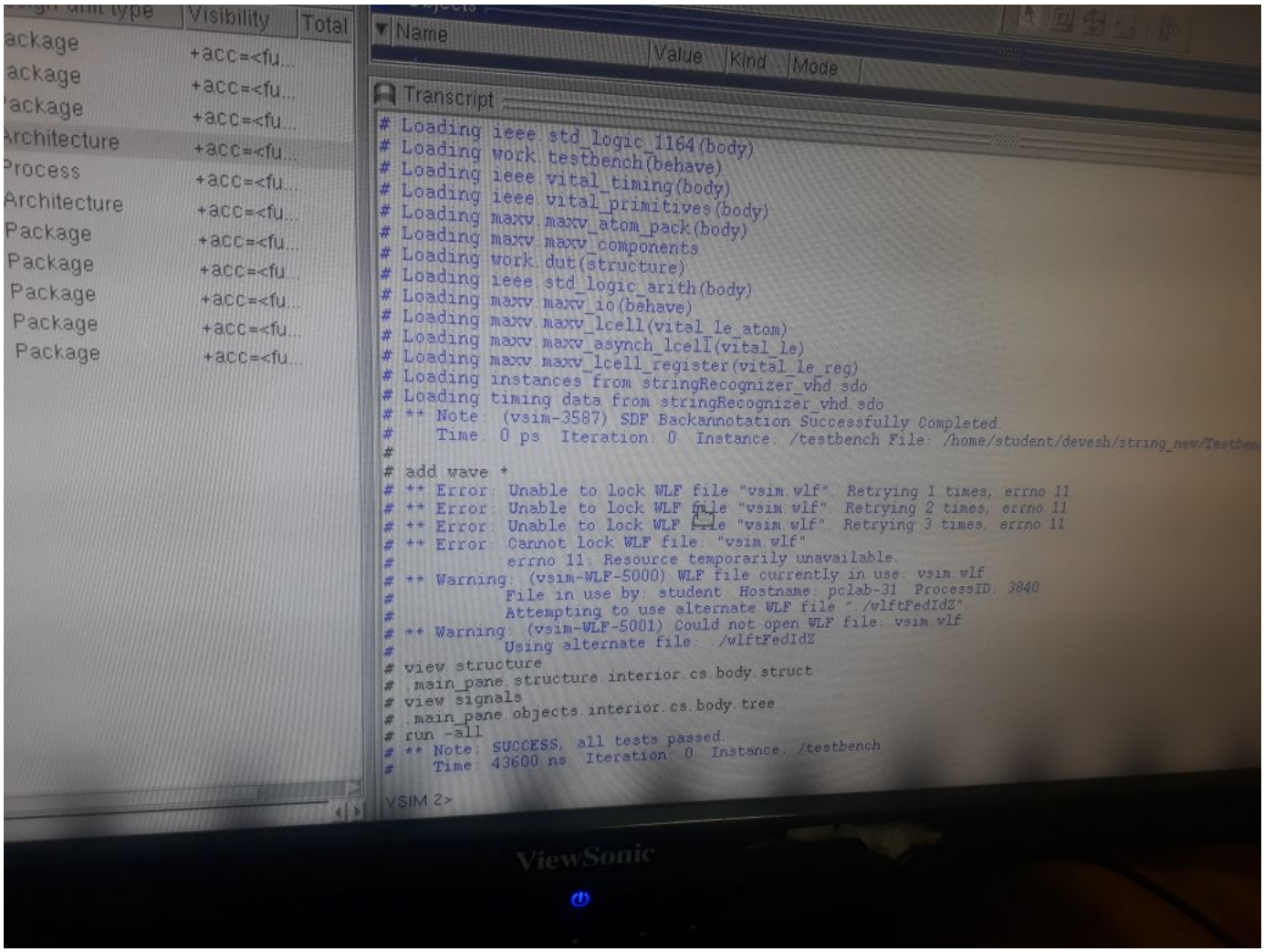
4 Test bench

```
 library std;
use std.textio.all;

library std;
use std.standard.all;

library ieee;
use ieee.std_logic_1164.all;

entity Testbench is
end entity;
architecture Behave of Testbench is

  ----------------------------------------------------------------
  --  edit the following lines to set the number of i/o's of your
  --  DUT.
  ----------------------------------------------------------------
  constant number_of_inputs  : integer := 7;  -- # input bits to your design.
  constant number_of_outputs : integer := 1;  -- # output bits from your design.

  -- component port widths..
  component DUT is
   port(input_vector: in std_logic_vector(number_of_inputs-1  downto 0);
        output_vector: out std_logic_vector(number_of_outputs-1 downto 0));
  end component;

  -- end editing.
  ----------------------------------------------------------------
  ----------------------------------------------------------------

  signal input_vector  : std_logic_vector(number_of_inputs-1 downto 0);
  signal output_vector : std_logic_vector(number_of_outputs-1 downto 0);

  -- create a constrained string outof
  function to_string(x: string) return string is
      variable ret_val: string(1 to x'length);
      alias lx : string (1 to x'length) is x;
  begin
      ret_val := lx;
      return(ret_val);
  end to_string;
```

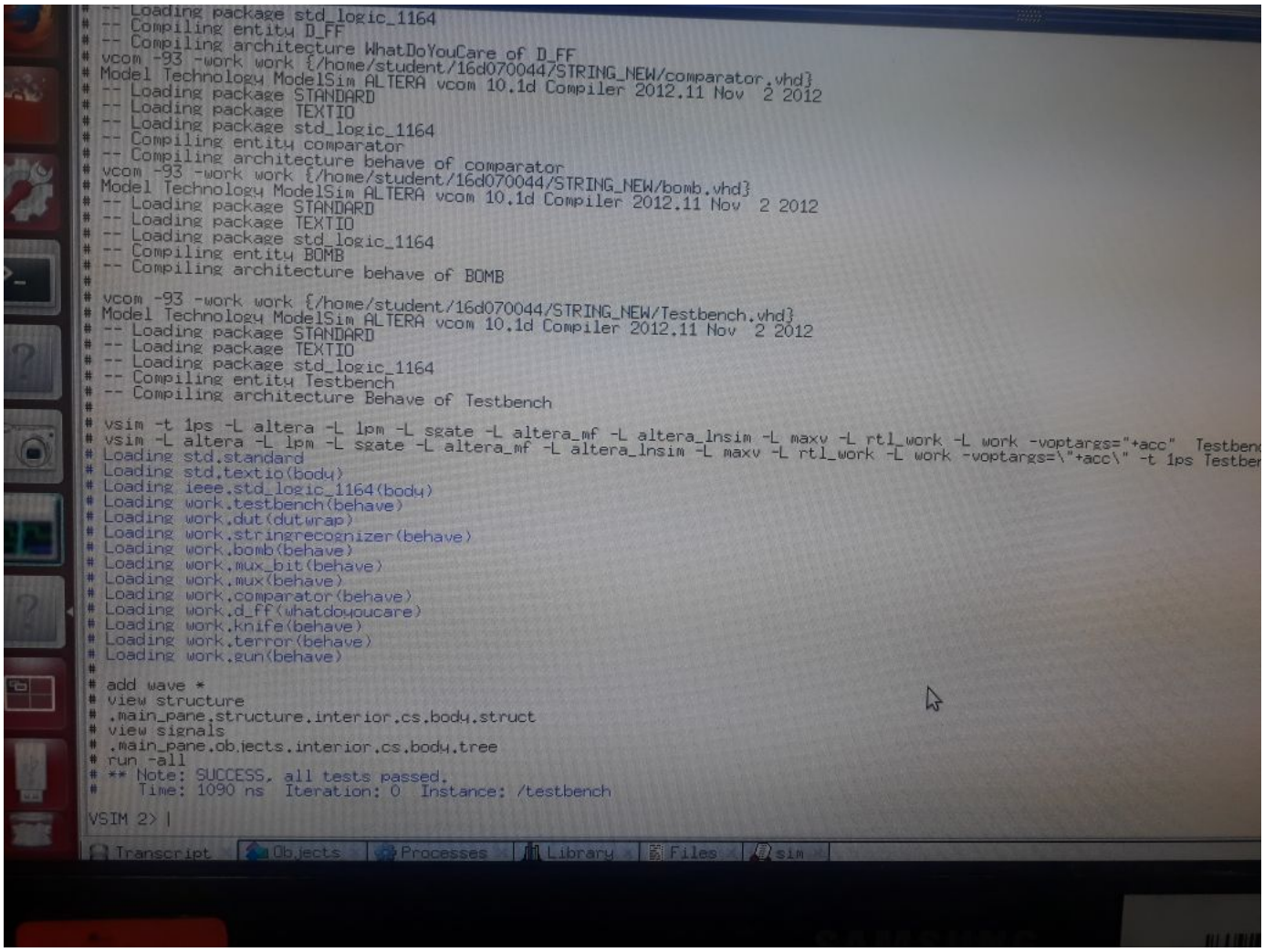
```
begin
  process
    variable err_flag : boolean := false;
    File INFILE: text open read_mode is
    "/home/student/16d070044/STRING_NEW/string_recognizer_test_files/tracefile1.txt";
    FILE OUTFILE: text  open write_mode is
    "/home/student/16d070044/STRING_NEW/string_recognizer_test_files/OUTPUT11.txt";

    ------------------------------------------------
    -- edit the next two lines to customize
    ------------------------------------------------
    variable input_vector_var: bit_vector (number_of_inputs-1 downto 0);
    variable output_vector_var: bit_vector (number_of_outputs-1 downto 0);
    variable output_mask_var: bit_vector (number_of_outputs-1 downto 0);
    variable output_comp_var: bit_vector (number_of_outputs-1 downto 0);
    constant ZZZZ : bit_vector(number_of_outputs-1 downto 0) := (others => '0');
    ------------------------------------------------

    variable INPUT_LINE: Line;
    variable OUTPUT_LINE: Line;
    variable LINE_COUNT: integer := 0;


  begin
    while not endfile(INFILE) loop
          -- will read a new line every 5ns, apply input,
          -- wait for 1 ns for circuit to settle.
          -- read output.


          LINE_COUNT := LINE_COUNT + 1;


          -- read input at current time.
          readLine (INFILE, INPUT_LINE);
          read (INPUT_LINE, input_vector_var);
          read (INPUT_LINE, output_vector_var);
          read (INPUT_LINE, output_mask_var);

          -- apply input.
          input_vector <= to_stdlogicvector(input_vector_var);

          -- wait for the circuit to settle
          wait for 1 ns;

          -- check output.
          output_comp_var := (output_mask_var and (to_bitvector(output_vector) xor output_vec
          if (output_comp_var  /= ZZZZ) then
             write(OUTPUT_LINE,to_string("ERROR: line "));
```

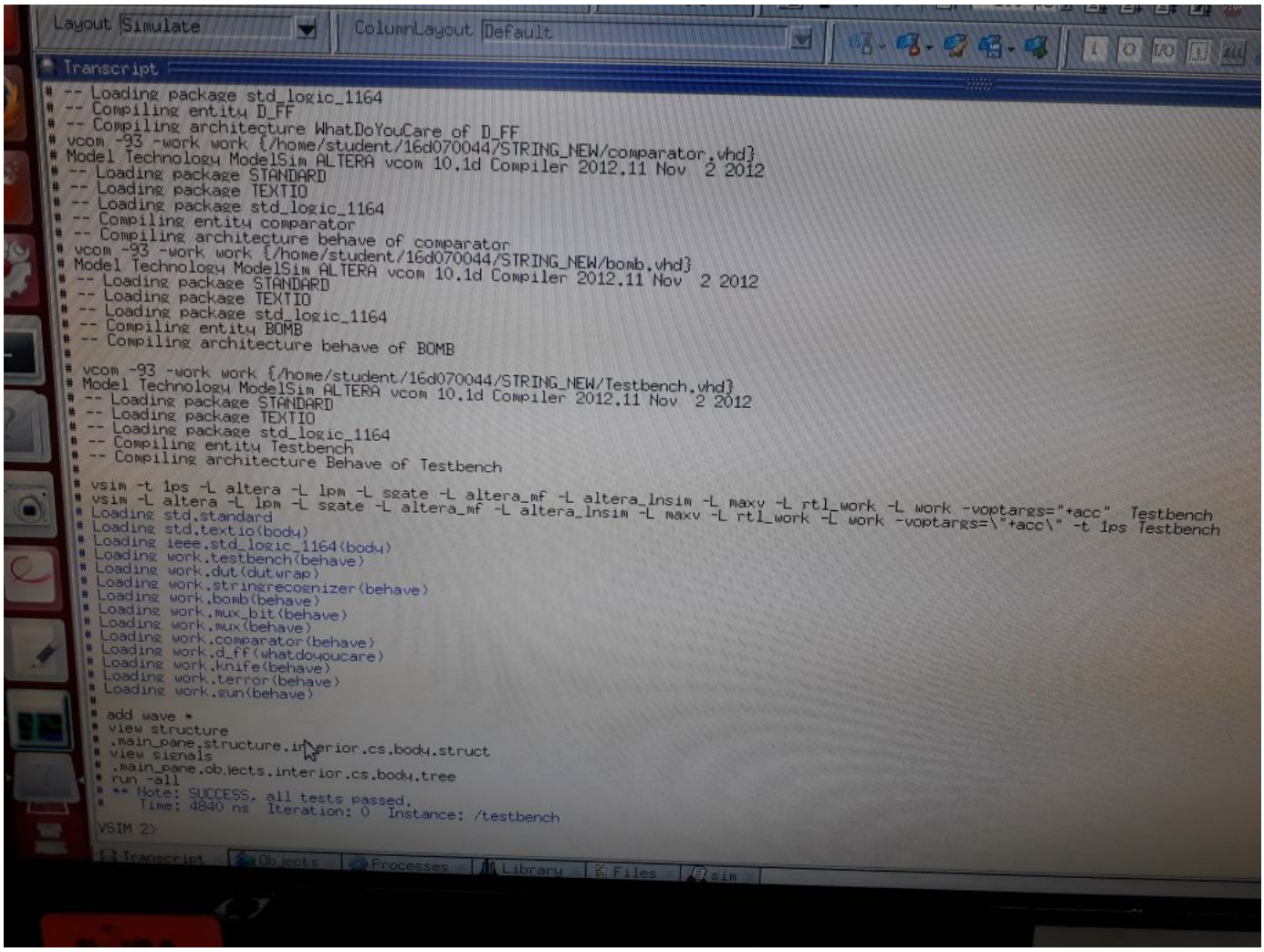
```
            write(OUTPUT_LINE, LINE_COUNT);
            writeline(OUTFILE, OUTPUT_LINE);
            err_flag := true;
        end if;

        write(OUTPUT_LINE, to_bitvector(input_vector));
        write(OUTPUT_LINE, to_string(" "));
        write(OUTPUT_LINE, to_bitvector(output_vector));
        writeline(OUTFILE, OUTPUT_LINE);

        -- advance time by 4 ns.
        wait for 4 ns;
    end loop;

    assert (err_flag) report "SUCCESS, all tests passed." severity note;
    assert (not err_flag) report "FAILURE, some tests failed." severity error;

    wait;
  end process;

  dut_instance: DUT
        port map(input_vector => input_vector, output_vector =>output_vector );

end Behave;
```

5 Simulation results: RTL viewer, waveforms if readable, simulation report after running test bench



gatelevel


-- Loading package std_logic_1164
-- Compiling entity D_FF
-- Compiling architecture WhatDoYouCare of D_FF
vcom -93 -work work {/home/student/16d070044/STRING_NEW/comparator.vhd}
Model Technology ModelSim ALTERA vcom 10.1d Compiler 2012.11 Nov 2 2012
-- Loading package STANDARD
-- Loading package TEXTIO
-- Loading package std_logic_1164
-- Compiling entity comparator
-- Compiling architecture behave of comparator
vcom -93 -work work {/home/student/16d070044/STRING_NEW/bomb.vhd}
Model Technology ModelSim ALTERA vcom 10.1d Compiler 2012.11 Nov 2 2012
-- Loading package STANDARD
-- Loading package TEXTIO
-- Loading package std_logic_1164
-- Compiling entity BOMB
-- Compiling architecture behave of BOMB
vcom -93 -work work {/home/student/16d070044/STRING_NEW/Testbench.vhd}
Model Technology ModelSim ALTERA vcom 10.1d Compiler 2012.11 Nov 2 2012
-- Loading package STANDARD
-- Loading package TEXTIO
-- Loading package std_logic_1164
-- Compiling entity Testbench
-- Compiling architecture Behave of Testbench
vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L maxv -L rtl_work -L work -voptargs="+acc" Testben
vsim -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L maxv -L rtl_work -L work -voptargs=\"+acc\" -t 1ps Testbe
Loading std.standard
Loading std.textio(body)
Loading ieee.std_logic_1164(body)
Loading work.testbench(behave)
Loading work.dut(dutwrap)
Loading work.stringrecognizer(behave)
Loading work.bomb(behave)
Loading work.mux_bit(behave)
Loading work.mux(behave)
Loading work.comparator(behave)
Loading work.d_ff(whatdoyoucare)
Loading work.knife(behave)
Loading work.terror(behave)
Loading work.gun(behave)
add wave *
view structure
.main_pane.structure.interior.cs.body.struct
view signals
.main_pane.objects.interior.cs.body.tree
run -all
** Note: SUCCESS, all tests passed.
Time: 1090 ns Iteration: 0 Instance: /testbench
VSIM 2>
Transcript
Objects
Processes
Library
Files
sim



We can see that all the test cases were passed.

6 Scan chain result