LAB REPORT of

GENERALISED STATE MACHINE GENERATOR

Submitted by:

Pavan Rai (4NM13LVS12)
Yogeesh Rao A (4NM13LVS18)

Class : M.Tech
Semester: II
Branch : VLSI & Embedded Systems Design
College : N.M.A.M.I.T. , Nitte

Submitted to:

Mahaveera K
Professor
Dept. of Electronics and Communication Engineering
N.M.A.M.I.T. , Nitte

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
N.M.A.M. INSTITUTE OF TECHNOLOGY , NITTE

Abstract:

Perl's process, file, and text manipulation facilities make it particularly well-suited for tasks involving automatic code generation, report filtering, net list patching, generating test vectors and controlling tools.

In our project we will write a Perl script to generate state machine RTL level HDL code, based on the input parameters given to the script.

This will show how scripts can be used for automatic code generation, making it easier for HDL programmers to code and reduce repetitive tasks.

Input:-

1. Design name
2. Input port names
3. Output port names
4. Clock signal's name and type
5. Reset signal's name, type, and resulting state
6. State transition table

Output:-

1. State machine in HDL (RTL-level Code Generated)

.

INDEX

1. A Brief description:-

A general structure of a state machine

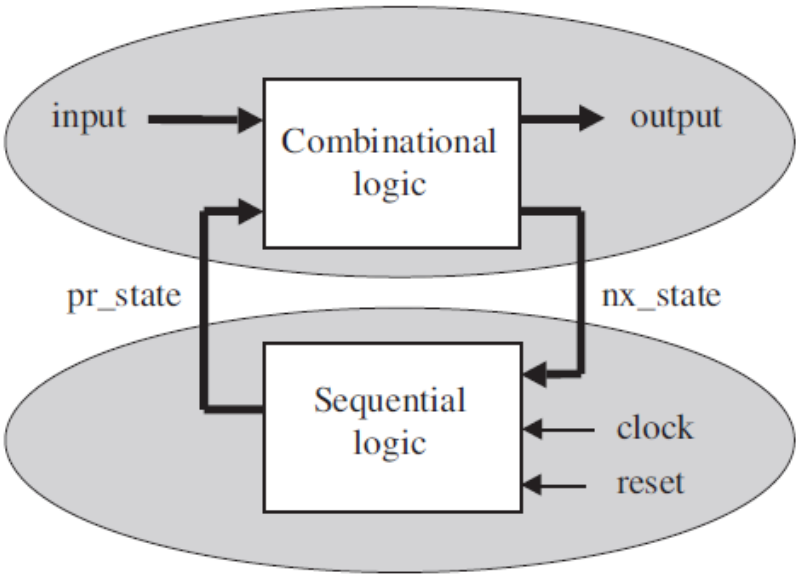


Figure 8.1
Mealy (Moore) state machine diagram.

Figure is taken from reference 1

Two scripts were written
1) State machine with no control signal
2) State machine with one control signal

1.1 State machine with no control signal

This script is best suited for RTL level VHDL code generation for counters. Our script can generate code for any kind of counter.

Example1
Example would be a BCD counter
Counting from zero to nine.

With only reset and clock signal.

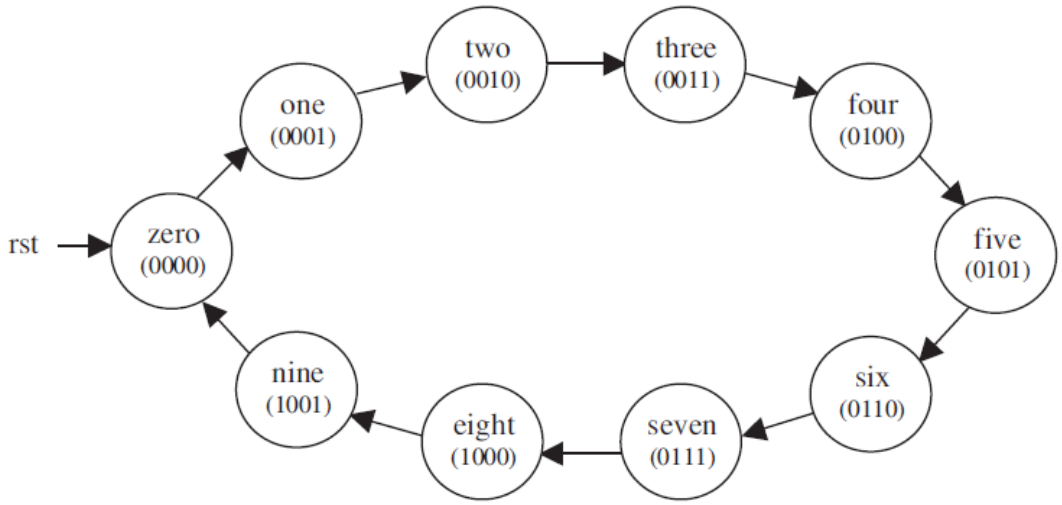


Figure 8.2
States diagram of example 8.1.

Figure is taken from reference [1]

1.2 State machine with one control signal:-

This script little bit more generalized and can generate RTL level VHDL code for state machines with one control signal given by the programmer.

Example2

A generalized example is given below

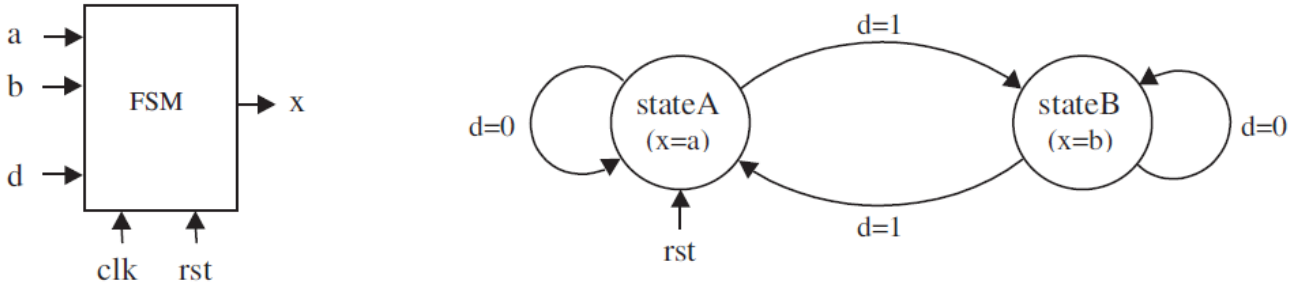


Figure 8.4
State machine of example 8.1.

Figure is taken from reference [1]

Example3
A specific example of sequence detector

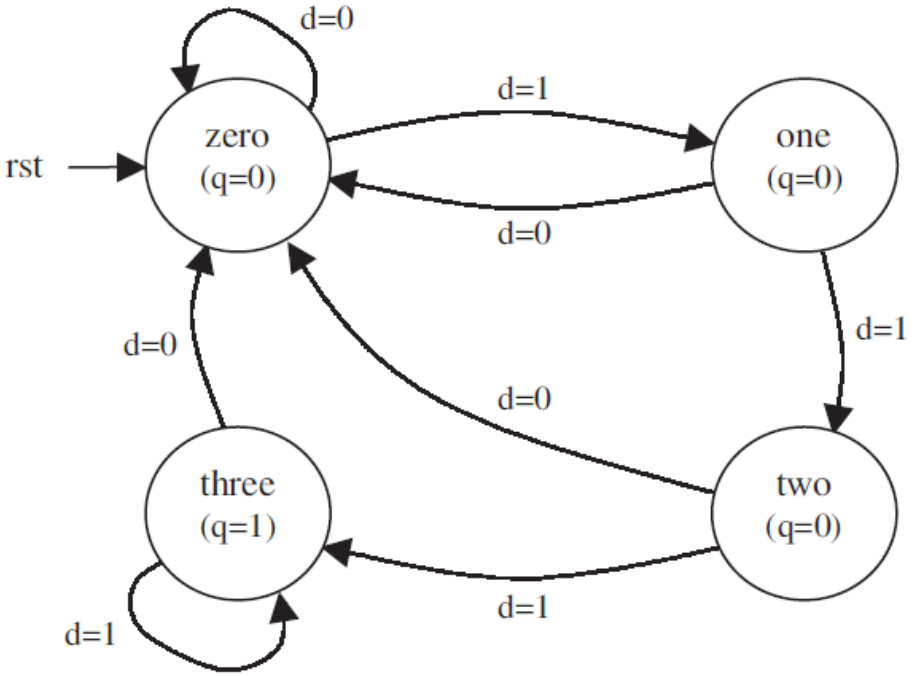


Figure 8.8
States diagram for example 8.4.

Figure is taken from reference [1]

It detects sequence “111”

2. The coding part:-

2.1 Software's used:-

1) For editing purpose we have used gVIM editor[2] reader can use any other editor of his choice
2) Interpreter Active Perl [3]

2.2 Source code:-

2.2.1 State machine with no control signal

Script:-

```
#-------------------------------------------------------------------------------------------
#
# State machine generator:-
#                   *no-control signal
#                   *can be used for generating RTL code for counter
#                   *with flexibility of N no states and N number of output bits
#-------------------------------------------------------------------------------------------
#
#             Authors:Pavan Rai, Yogish
#             subject:Application Lab
#             course :M.tech
#             Branch:VLSI and Embedded system design
#             Institution:NMAMIT,Nitte
#             Rev 0.1, may 10 , 2014
#
#-------------------------------------------------------------------------------------------
#              input parameter:
#              *no of output bits(counter bit length)
#              *no of states in state machine
#              *values of output at different states
#
#              output from script:
#              *RTL code generated
#              *which can simulated using any VHDL simulator
#-------------------------------------------------------------------------------------------

#!/usr/bin/perl

my $file = $ARGV[0];

# check to see if the user entered a file name.
die "syntax: [perl] vhdl_mod.pl new_file.vhd\n" if ( $file eq "" );

# check to make sure that the file doesn't exist.
die "Oops! A file called '$file' already exists.\n" if -e $file;
```

```
#number of bits to represent output bit
printf "----------enter the number of bits in output----------\n";
our $range = <STDIN>;
chop($range);

#number of states in the state machine
print "----------enter the number of states in the state machine ----------\n";
my $no_states = <STDIN>;
chop($no_states);

#enter the different state values
print "----------enter different state values----------\n";
our @array;
for ( my $count_val = 0 ; $count_val < $no_states ; $count_val++ ) {
   $array[$count_val] = <STDIN>;
   chop( $array[$count_val] );
}

open( our $inF, ">", $file );

# Strip the .vhd from the file name and use for the module name:
$file =~ s/\.vhd$//;

# Make Date int MM/DD/YYYY
my $year  = 0;
my $month = 0;
my $day   = 0;
( $day, $month, $year ) = (localtime)[ 3, 4, 5 ];

# Grab username from PC:
my $author = "$^O user";
if ( $^O =~ /mswin/i ) {
   $author = $ENV{USERNAME} if defined $ENV{USERNAME};
}
else {
   $author = getlogin();
}

#Module Template:
printf( $inF
"--------------------------------------------------------------------------------\n"
);
printf( $inF
     "--                                          Revision: 1.1 \n"
);
printf( $inF
"--                                        Date: %02d/%02d/%04d \n",
   $month + 1, $day, $year + 1900 );
printf( $inF
"--------------------------------------------------------------------------------\n"
);
printf( $inF "--\n" );
printf( $inF "--   File name :  $file.vhd\n" );
printf( $inF "--   Title     :  adder\n" );
printf( $inF "--   Module    :  $file\n" );
```

```
printf( $inF "--   Author    :  $author\n" );
printf( $inF "--   Purpose   :  task project\n" );
printf( $inF "--\n" );
printf( $inF "--   Roadmap   :\n" );
printf( $inF
"---------------------------------------------------------------------------\n"
);
printf( $inF "--   Modification History :\n" );
printf( $inF "--\tDate\t\tAuthor\t\tRevision\tComments\n" );
printf( $inF "--\t%02d/%02d/%04d\t$author\tRev A\t\tCreation\n",
   $month + 1, $day, $year + 1900 );
printf( $inF
"---------------------------------------------------------------------------\n"
);
printf( $inF "\n" );
printf( $inF "Library IEEE;\n" );
printf( $inF "use IEEE.STD_LOGIC_1164.all;\n" );
printf( $inF "use IEEE.std_logic_unsigned.all;\n" );
printf( $inF "use IEEE.std_logic_arith.all;\n" );
printf( $inF "use IEEE.Numeric_STD.all;\n" );
printf( $inF "\n" );
printf( $inF "\n" );
printf( $inF "\n" );
printf( $inF
     "-- Declare module entity. Declare module inputs, inouts, and outputs.\n"
);
printf( $inF "entity $file is\n" );
printf( $inF "\tport\t(\n" );
printf( $inF "\t\t-- *** Inputs ***\n" );
printf( $inF
     "\t\treset,clock : in std_logic ;          --  reset and clock \n" );
my $range1;
$range1 = $range - 1;
printf( $inF
"\t\tcount : out  std_logic_vector($range1 downto 0)              -- output \n"
);
printf( $inF "); \n" );
printf( $inF "end $file;\n" );
printf( $inF "\n" );
printf( $inF "-- Begin module architecture/code.\n" );
printf( $inF "architecture behave of $file is\n" );
printf( $inF "\n" );
printf( $inF "-- N/A\n" );
printf( $inF "-- general signals\n" );
printf( $inF "\n" );
printf( $inF "\n" );

#printing the states.....state0,state1,state2,state3.....................
printf( $inF "\ttype state is ( " );
for ( my $i = 0 ; $i < $no_states ; $i++ ) {
   printf( $inF "state$i" );
   if ( $i < ( $no_states - 1 ) ) {
      printf( $inF "," );
   }
}
printf( $inF ");\n" );
```

```
printf( $inF
"\tsignal  pr_state ,nx_state:state ;          -- signal to store temprory variable \n"
);
printf( $inF "\n" );
printf( $inF "begin\n" );
printf( $inF "\n" );

#printing the first process part here
printf( $inF "-- Insert Processes and code here.\n" );
printf( $inF "\tprocess(reset,clock)  \n" );
printf( $inF "begin\n" );
printf( $inF "\tif (reset='1') then  \n" );
printf( $inF "\t\tpr_state<=state0; \n" );
printf( $inF "\telsif (clock'event and clock='1') then \n" );
printf( $inF "\t\t pr_state<=nx_state; \n" );
printf( $inF "\tend if; \n" );
printf( $inF "end process;\n" );

#printing the second process part here
printf( $inF "\tprocess(pr_state)\n" );
printf( $inF "begin\n" );
printf( $inF "\t\tcase pr_state is \n" );

#printing the different cases when  statements repeatively using for loop
for ( my $count_case = 0 ; $count_case < $no_states ; $count_case++ ) {
   printf( $inF "\t\twhen state$count_case => \n" );
   printf( $inF "\t\t\tcount<=\"$array[$count_case]\";\n" );
   my $k = ( $count_case + 1 ) % ($no_states);
   printf( $inF "\t\t\tnx_state<=state$k;\n" );
}
printf( $inF "end case;\n" );
printf( $inF "end process; \n" );
printf( $inF "\n" );
printf( $inF "end behave; -- architecture\n" );

#closing the file after completing the printing task
close(inF);

print("\nThe script has finished successfully! You can now use $file.vhd.\n\n");
```

Output for Example1:-

Terminal view:-

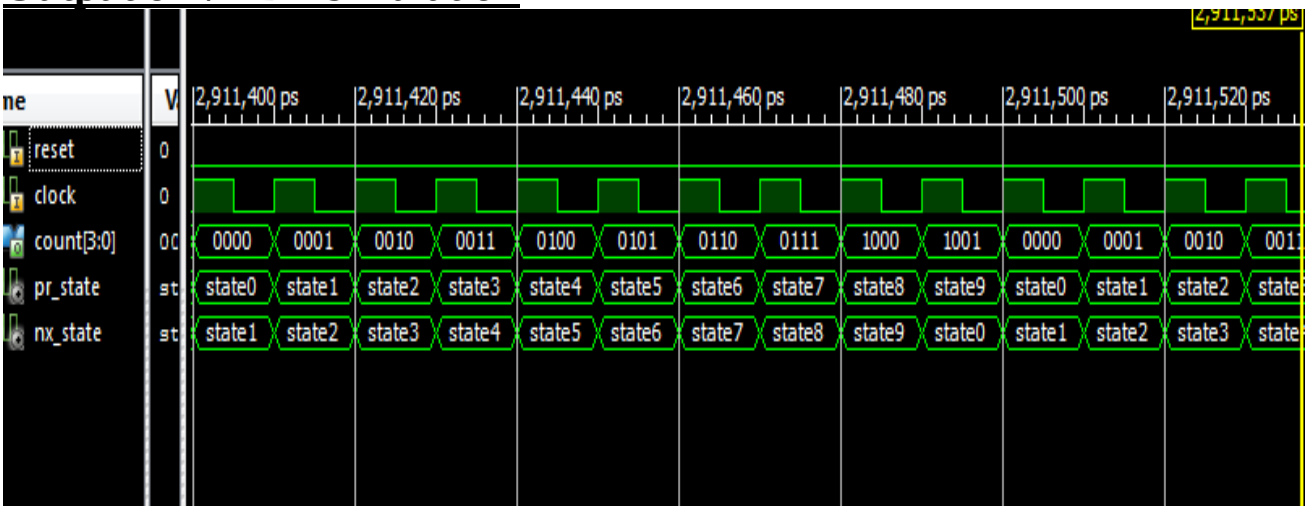
```
D:\bigperl>perl state2.pl BCDcounter.vhd
----------enter the number of bits in output----------
4
----------enter the number of states in the state machine ----------
10
----------enter different state values----------
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001

The script has finished successfully! You can now use BCDcounter.vhd.
```

```
--------------------------------------------------------------------------------
--                                  Revision: 1.1
--                                  Date: 05/11/2014
--------------------------------------------------------------------------------
--
--  File name :  BCDcounter.vhd
--  Title     :  adder
--  Module    :  BCDcounter
--  Author    :  PaVaN
--  Purpose   :  task project
--
--  Roadmap   :
--------------------------------------------------------------------------------
--  Modification History :
--      Date            Author          Revision    Comments
--      05/11/2014      PaVaN   Rev A               Creation
--------------------------------------------------------------------------------

Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.Numeric_STD.all;



-- Declare module entity. Declare module inputs, inouts, and outputs.
entity BCDcounter is
        port    (
                -- *** Inputs ***
                reset,clock : in std_logic ;        --  reset and clock
                count : out  std_logic_vector(3 downto 0)           -- output
);
end BCDcounter;
```

```
-- Begin module architecture/code.
architecture behave of BCDcounter is

-- N/A
-- general signals


        type state is ( state0,state1,state2,state3,state4,state5,state6,state7,state8,state9);
        signal  pr_state ,nx_state:state ;          -- signal to store temprory variable

begin

-- Insert Processes and code here.
        process(reset,clock)
begin
        if (reset='1') then
                pr_state<=state0;
        elsif (clock'event and clock='1') then
                 pr_state<=nx_state;
        end if;
end process;
        process(pr_state)
begin
                case pr_state is
                when state0 =>
                        count<="0000";
                        nx_state<=state1;
                when state1 =>
                        count<="0001";
                        nx_state<=state2;
                when state2 =>
                        count<="0010";
                        nx_state<=state3;
                when state3 =>
                        count<="0011";
                        nx_state<=state4;
                when state4 =>
                        count<="0100";
                        nx_state<=state5;
                when state5 =>
                        count<="0101";
                        nx_state<=state6;
                when state6 =>
                        count<="0110";
                        nx_state<=state7;
                when state7 =>
                        count<="0111";
                        nx_state<=state8;
                when state8 =>
                        count<="1000";
                        nx_state<=state9;
                when state9 =>
                        count<="1001";
                        nx_state<=state0;
end case;
end process;
```

end behave; -- architecture

Output of VHDL simulation



2.2.2 State machine with one control signal

```
#-------------------------------------------------------------------------------------------
#
# State machine generator:-
#                   *one-control signal
#                   *can be used for generating RTL code for sequece generator or any other purposewith one control
signal
#                   *with flexibility of N no states ,N number of output bits(or it can be input),
#-------------------------------------------------------------------------------------------
#
#               Authors:Pavan Rai, Yogish
#               subject:Application Lab
#               course :M.tech
#               Branch:VLSI and Embedded system design
#               Institution:NMAMIT,Nitte
#               Rev 0.1, may 10 , 2014
#
#-------------------------------------------------------------------------------------------
#               input parameter:
#               *name of VHDL file to be generted given as command line argument
#               *no of input signals including one control signal
#               *control signal and the input signal
#               *no of states in state machine
#               *values of output at different states
#
#               output from script:
#               *RTL code generated
#               *which can simulated using any VHDL simulator
```

```
#-----------------------------------------------------------------------------------------------

#!/usr/bin/perl

my $file = $ARGV[0];

# check to see if the user entered a file name.
die "syntax: [perl] vhdl_mod.pl new_file.vhd\n" if ( $file eq "" );

# check to make sure that the file doesn't exist.
die "Oops! A file called '$file' already exists.\n" if -e $file;

#number of inputs
print
"\n----------enter the number of input signal including one control signal---------\n----------excluding clock and reset--
--------\n";
my $num_ip = <STDIN>;

#inputsitself
print "\n----------enter the inputs and control signal----------\n";
our @control_ip;
for ( my $count_ip = 0 ; $count_ip < $num_ip ; $count_ip++ ) {
   $control_ip[$count_ip] = <STDIN>;
   chop( $control_ip[$count_ip] );
}

print "-----------enter the number of states in the statemachine----------\n";
my $n = <STDIN>;
chop($n);
print "-----------enter different state values----------\n";
our @array;
for ( my $j = 0 ; $j < $n ; $j++ ) {
   $array[$j] = <STDIN>;
   chop( $array[$j] );
}

print "----------enter state jumpings based on conditions ----------\n";
print "----------input format -----------\n";
print "----------if(control==1)---------\n";
print "----------true outcome -------\n";
print "----------else ----------\n";
print "----------false outcome-------\n";

our @conditions;
for ( my $count_cond = 0 ; $count_cond < ( 2 * $n ) ; $count_cond++ ) {
   $conditions[$count_cond] = <STDIN>;
   chop( $conditions[$count_cond] );
}
```

```
open( our $inF, ">", $file );

# Strip the .v from the file name and use for the module name:
$file =~ s/\.vhd$//;

# Make Date int MM/DD/YYYY
my $year  = 0;
my $month = 0;
my $day   = 0;
( $day, $month, $year ) = (localtime)[ 3, 4, 5 ];

# Grab username from PC:
my $author = "$^O user";
if ( $^O =~ /mswin/i ) {
   $author = $ENV{USERNAME} if defined $ENV{USERNAME};
}
else {
   $author = getlogin();
}

#Module Template:
printf( $inF
"--------------------------------------------------------------------------------\n"
);
printf( $inF
     "--                                          Revision: 1.1 \n"
);
printf( $inF
"--                                         Date: %02d/%02d/%04d \n",
   $month + 1, $day, $year + 1900 );
printf( $inF
"--------------------------------------------------------------------------------\n"
);
printf( $inF "--\n" );
printf( $inF "--   File name :  $file.vhd\n" );
printf( $inF "--   Title     :  state machine\n" );
printf( $inF "--   Module    :  $file\n" );
printf( $inF "--   Author    :  $author\n" );
printf( $inF "--   Purpose   :  task project\n" );
printf( $inF "--\n" );
printf( $inF "--   Roadmap   :\n" );
printf( $inF
"--------------------------------------------------------------------------------\n"
);
printf( $inF "--   Modification History :\n" );
printf( $inF "--\tDate\t\tAuthor\t\tRevision\tComments\n" );
printf( $inF "--\t%02d/%02d/%04d\t$author\tRev A\t\tCreation\n",
   $month + 1, $day, $year + 1900 );
printf( $inF
```

```
"--------------------------------------------------------------------------------\n"
);
printf( $inF "\n" );
printf( $inF "Library IEEE;\n" );
printf( $inF "use IEEE.STD_LOGIC_1164.all;\n" );
printf( $inF "use IEEE.std_logic_unsigned.all;\n" );
printf( $inF "use IEEE.std_logic_arith.all;\n" );
printf( $inF "use IEEE.Numeric_STD.all;\n" );
printf( $inF "\n" );
printf( $inF "\n" );
printf( $inF "\n" );
printf( $inF
     "-- Declare module entity. Declare module inputs, inouts, and outputs.\n"
);
printf( $inF "entity $file is\n" );
printf( $inF "\tport\t(\n" );
printf( $inF "\t\t-- *** Inputs ***\n" );
printf( $inF "\t\t" );

for ( my $count_print = 0 ; $count_print < $num_ip ; $count_print++ ) {
   printf( $inF "$control_ip[$count_print]" );
   if ( $count_print < ($num_ip) ) {
      printf( $inF "," );
   }
}
printf( $inF "reset,clock : in std_logic ;          --  reset and clock \n" );
printf( $inF "\t\toutput : out  std_logic          -- output \n" );
printf( $inF "); \n" );
printf( $inF "end $file;\n" );
printf( $inF "\n" );
printf( $inF "-- Begin module architecture/code.\n" );
printf( $inF "architecture behave of $file is\n" );
printf( $inF "\n" );

printf( $inF "-- N/A\n" );
printf( $inF "-- general signals\n" );
printf( $inF "\n" );
printf( $inF "\n" );

printf( $inF "\ttype state is ( " );

for ( my $i = 0 ; $i < $n ; $i++ ) {
   printf( $inF "state$i" );
   if ( $i < ( $n - 1 ) ) {
      printf( $inF "," );
   }
}

printf( $inF ");\n" );
```

```
printf( $inF
"\tsignal  pr_state ,nx_state:state ;      -- signal to store temprory variable \n"
);
printf( $inF "\n" );
printf( $inF "begin\n" );
printf( $inF "\n" );
printf( $inF "-- Insert Processes and code here.\n" );
printf( $inF "\tprocess(reset,clock)  \n" );
printf( $inF "begin\n" );
printf( $inF "\tif (reset='1') then  \n" );
printf( $inF "\t\tpr_state<=state0; \n" );
printf( $inF "\telsif (clock'event and clock='1') then \n" );
printf( $inF "\t\t pr_state<=nx_state; \n" );
printf( $inF "\tend if; \n" );
printf( $inF "end process;\n" );
printf( $inF "\tprocess(" );

for ( my $count_ip1 = 0 ; $count_ip1 < $num_ip ; $count_ip1++ ) {
   printf( $inF "$control_ip[$count_ip1]," );
}
printf( $inF "pr_state)\n" );
printf( $inF "begin\n" );
printf( $inF "\t\tcase pr_state is \n" );
our $temp = 0;
for ( my $i = 0 ; $i < $n ; $i++ ) {
   printf( $inF "\t\twhen state$i => \n" );

   if ( $array[$i] =~ /[a-z]+/i ) {
      printf( $inF "\t\t\toutput<=$array[$i];\n" );
   }
   else {
      printf( $inF "\t\t\toutput<='$array[$i]';\n" );
   }

   printf( $inF "\t if ($control_ip[$num_ip-1]='1') then \n" );
   printf( $inF "\t\t\tnx_state<=state$conditions[$temp];\n" );
   printf( $inF "\t else\n" );
   $temp = ( $temp + 1 );
   printf( $inF "\t\t\tnx_state<=state$conditions[$temp];\n" );
   printf( $inF "\tend if;\n" );
   $temp = $temp + 1;
}

printf( $inF "end case;\n" );
printf( $inF "end process; \n" );
printf( $inF "\n" );
printf( $inF "end behave; -- architecture\n" );
close(inF);
print("\nThe script has finished successfully! You can now use $file.vhd.\n\n");
```

exit;

Output for Example2:-

Terminal view:-

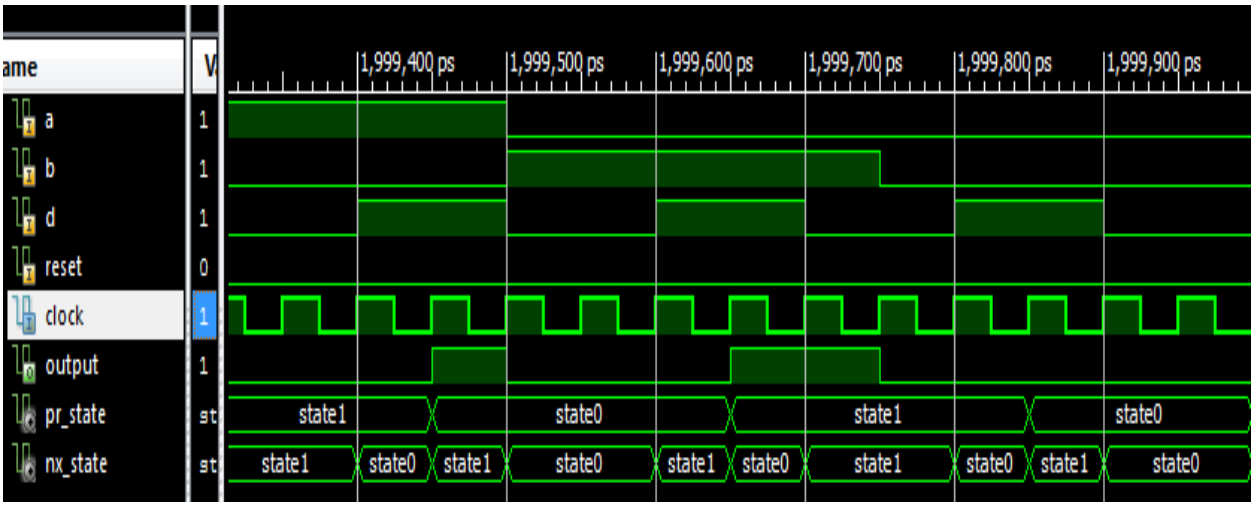
```
D:\bigperl>perl state3.pl fsm1.vhd

----------enter the number of input signal including one control signal-------

----------excluding clock and reset----------
3

----------enter the inputs and control signal----------
a
b
d
----------enter the number of states in the statemachine----------
2
----------enter different state values----------
a
b
----------enter state jumpings based on conditions ----------
----------input format ----------
----------if(control==1)----------
----------true outcome --------
------------else ----------
----------false outcome-------
1
0
0
1

The script has finished successfully! You can now use fsm1.vhd.
```

```
--------------------------------------------------------------------------------
--                                Revision: 1.1
--                                Date: 05/11/2014
--------------------------------------------------------------------------------
--
--  File name :  fsm1.vhd
--  Title     : state machine
--  Module    : fsm1
--  Author    : PaVaN
--  Purpose   : task project
--
--  Roadmap   :
--------------------------------------------------------------------------------
--  Modification History :
--        Date              Author          Revision          Comments
--        05/11/2014        PaVaN  Rev A              Creation
--------------------------------------------------------------------------------

Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;
```

```
use IEEE.std_logic_arith.all;
use IEEE.Numeric_STD.all;



-- Declare module entity. Declare module inputs, inouts, and outputs.
entity fsm1 is
        port    (
                -- *** Inputs ***
                a,b,d,reset,clock : in std_logic ;       --  reset and clock
                output : out  std_logic         -- output
);
end fsm1;

-- Begin module architecture/code.
architecture behave of fsm1 is

-- N/A
-- general signals


        type state is ( state0,state1);
        signal  pr_state ,nx_state:state ;     -- signal to store temprory variable

begin

-- Insert Processes and code here.
        process(reset,clock)
begin
        if (reset='1') then
                pr_state<=state0;
        elsif (clock'event and clock='1') then
                 pr_state<=nx_state;
        end if;
end process;
        process(a,b,d,pr_state)
begin
                case pr_state is
                when state0 =>
                        output<=a;
         if (d='1') then
                        nx_state<=state1;
         else
                        nx_state<=state0;
        end if;
                when state1 =>
                        output<=b;
         if (d='1') then
```

```
                    nx_state<=state0;
        else
                    nx_state<=state1;
        end if;
end case;
end process;

end behave; -- architecture
```

Output of VHDL simulation



Output for Example3:-
Terminal view:-

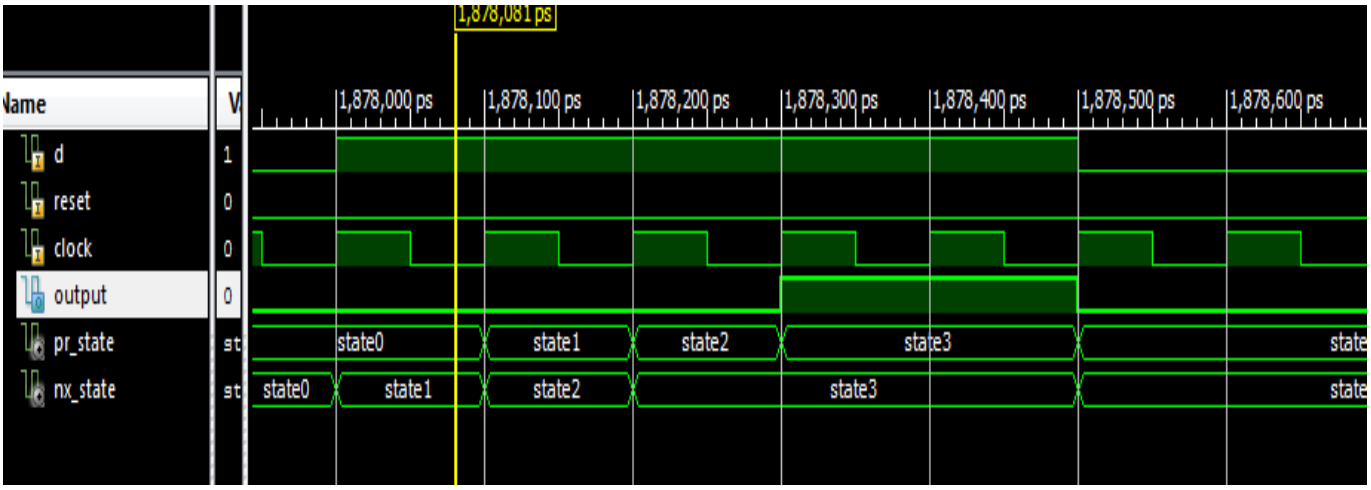
```
D:\bigperl>perl state3.pl fsm3.vhd

----------enter the number of input signal including one control signal--

----------excluding clock and reset----------
1

----------enter the inputs and control signal----------
d
----------enter the number of states in the statemachine----------
4
----------enter different state values----------
0
0
0
1
----------enter state jumpings based on conditions ----------
----------input format ----------
----------if(control==1)----------
----------true outcome -------
----------else ----------
----------false outcome-------
1
0
2
0
3
0
3
0

The script has finished successfully! You can now use fsm3.vhd.
```

```
--------------------------------------------------------------------------------
--                                  Revision: 1.1
--                                  Date: 05/11/2014
--------------------------------------------------------------------------------
--
--  File name :  fsm3.vhd
--  Title     :  state machine
--  Module    :  fsm3
--  Author    :  PaVaN
--  Purpose   :  task project
--
--  Roadmap   :
--------------------------------------------------------------------------------
--  Modification History :
--        Date              Author          Revision Comments
--        05/11/2014        PaVaN   Rev A            Creation
--------------------------------------------------------------------------------

Library IEEE;
```

```
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.Numeric_STD.all;



-- Declare module entity. Declare module inputs, inouts, and outputs.
entity fsm3 is
        port    (
                -- *** Inputs ***
                d,reset,clock : in std_logic ;          --  reset and clock
                output : out  std_logic          -- output
);
end fsm3;

-- Begin module architecture/code.
architecture behave of fsm3 is

-- N/A
-- general signals


        type state is ( state0,state1,state2,state3);
        signal  pr_state ,nx_state:state ;      -- signal to store temprory variable

begin

-- Insert Processes and code here.
        process(reset,clock)
begin
        if (reset='1') then
                pr_state<=state0;
        elsif (clock'event and clock='1') then
                 pr_state<=nx_state;
        end if;
end process;
        process(d,pr_state)
begin
                case pr_state is
                when state0 =>
                        output<='0';
         if (d='1') then
                        nx_state<=state1;
         else
                        nx_state<=state0;
        end if;
                when state1 =>
                        output<='0';
```

```
            if (d='1') then
                        nx_state<=state2;
            else
                        nx_state<=state0;
            end if;
                  when state2 =>
                        output<='0';
            if (d='1') then
                        nx_state<=state3;
            else
                        nx_state<=state0;
            end if;
                  when state3 =>
                        output<='1';
            if (d='1') then
                        nx_state<=state3;
            else
                        nx_state<=state0;
            end if;
end case;
end process;

end behave; -- architecture
```

Output of VHDL simulation



3)Reference:-

1.Circuit Design With VHDL,Volnei A.Pedroni,MIT Press

2. http://www.vim.org/download.php

3. http://www.activestate.com/activeperl

4. http://www.perl.org/learn.html

5.Beginning Perl by Simon Cozens, Peter Wainwright. 700 pages. Wrox Press Inc. (May 25, 2000).(Highly recommended)

6.http://www.epic-ide.org/

7. http://opencores.org/project,perlilog,overview

8. http://embedded.eecs.berkeley.edu/Alumni/pinhong/scriptEDA/

9.http://embedded.eecs.berkeley.edu/Alumni/pinhong/scriptEDA/seda_final.pdf (recommended)

10. http://www.vlsiip.com/perlindex.html