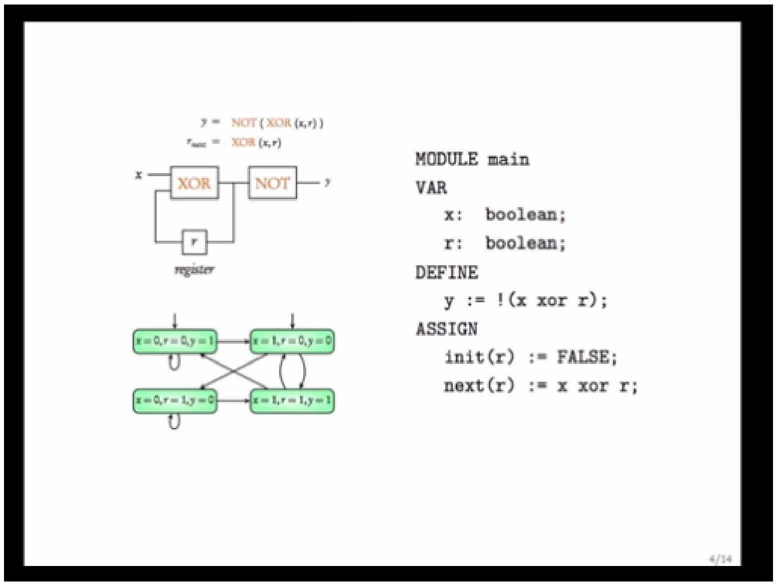
Model Checking

Prof. B. Srivathsan
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture - 09
Hardware Verification Using NuSMV

In the previous module, we saw how to model simple transition systems in NuSMV. In this module we will see how hardware circuits can be described in the tool NuSMV. There are 3 parts to this module, in the first part we will see a simple circuit, in the second and third parts we will see more interesting circuits.

(Refer Slide Time: 00:31)



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
DEFINE
   y := !(x xor r);
ASSIGN
   init(r) := FALSE;
   next(r) := x xor r;
```

Consider the circuit that we saw in unit 1, there are 2 inputs x and r, the output is determined as NOT of XOR of x comma r the next value of the register r; the next value of the input r is determined as XOR of x and r; the next value of x is non deterministic it could be anything this is clearly described in this transition system.

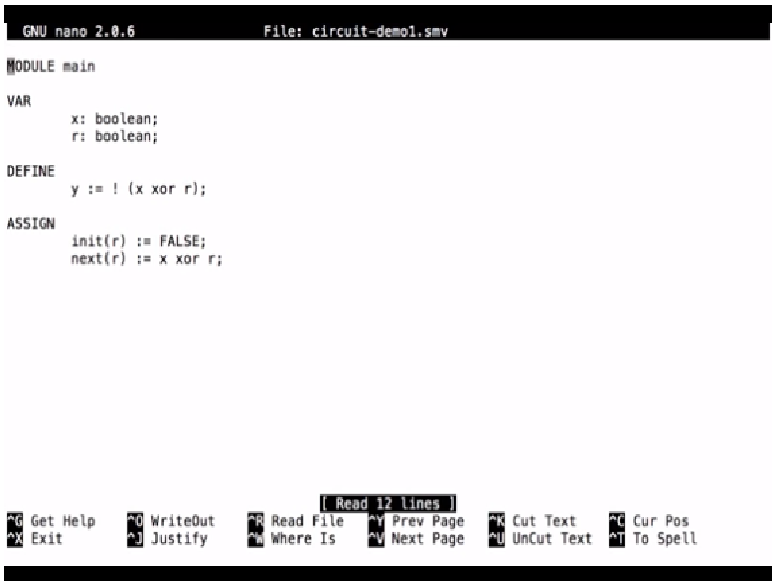
The initial value of r is assumed to be 0; the initial value of x could be either 0 or 1. Once the value of x and r are given the value of y gets determined this gives us 4 states. Let us now try to describe the circuit using NuSMV. As usual it starts with module main. There are 2 variables x and r both of them are of the boolean type these are the inputs. The initial value of the variable r is false.

There are 2 things now we need to define the next value of r and we need to assign the value y to be not of XOR of x comma r. Note that there is a difference between these 2 this says that once I had x and r the next value of r is XOR of x comma r. So, there is a unit delay for the next value, however once I get x and r immediately the value of y is not of XOR of x comma r for the value of y there is no delay.

How do we describe this? Here we say that next of r is x XOR r this is fine. What about y? We cannot say next of y is not of XOR of x comma r because once the value of x and r are given the value of is y is determined by the current values of x and r. To model such things there is a key word define this says that the value of y would depend on the current value of x and r as follows it is given as not of x XOR r.

We do not have to separately declare the variable y in the VAR block. We can directly write y to be not of x XOR r. This variable does not play a part in determining the number of states. The number of states is going to be determined by the variables here. Since x and r are boolean there are 2 choices for x, 2 choices for r. Hence, the number of states is going to be 4.

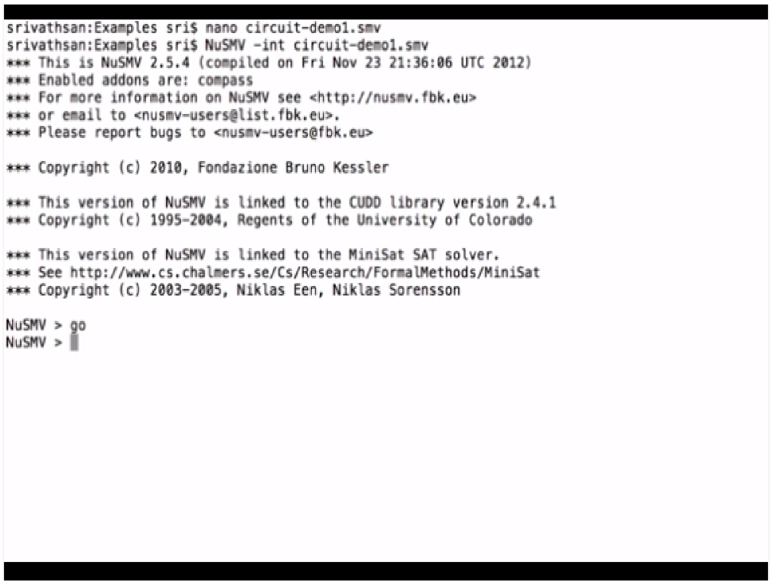
(Refer Slide Time: 03:53)



```
  GNU nano 2.0.6              File: circuit-demo1.smv

MODULE main

VAR
        x: boolean;
        r: boolean;

DEFINE
        y := ! (x xor r);

ASSIGN
        init(r) := FALSE;
        next(r) := x xor r;
```

Let us now see a demo of this, i have written the code in the file circuit hyphen demo1 dot smv.

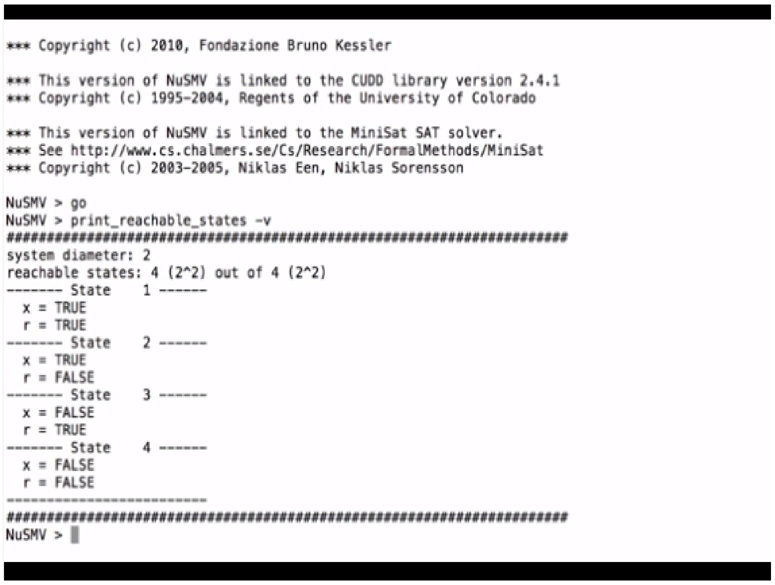
(Refer Slide Time: 04:33)



```
srivathsan:Examples sri$ nano circuit-demo1.smv
srivathsan:Examples sri$ NuSMV -int circuit-demo1.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV >
```

Let’s get into the interactive mode, you could directly give the name of the file in the command for the interactive mode. So, you can write NuSMV minus int circuit hyphen demo1 dot smv. If you remember we used to write read model, flatten hierarchy, encode variables and build model. So, however there seems to be simple command go which will do all these for us.

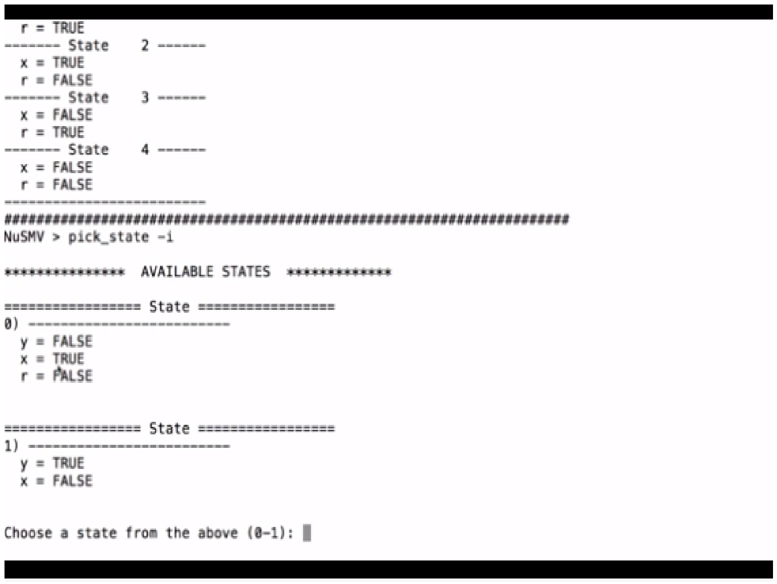
(Refer Slide Time: 04:48)



```
*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV > print_reachable_states -v
######################################################################
system diameter: 2
reachable states: 4 (2^2) out of 4 (2^2)
------- State    1 ------
  x = TRUE
  r = TRUE
------- State    2 ------
  x = TRUE
  r = FALSE
------- State    3 ------
  x = FALSE
  r = TRUE
------- State    4 ------
  x = FALSE
  r = FALSE
-------------------------
######################################################################
NuSMV >
```

Let us now print the set of reachable states, print reachable states minus v. There are 4 states given by x equal to true, r equal to true, x equal to true r equal to false, x equal to false r equal to true, x equal to false r equal to false. As you see this the states are determined only by the variables in the VAR block.

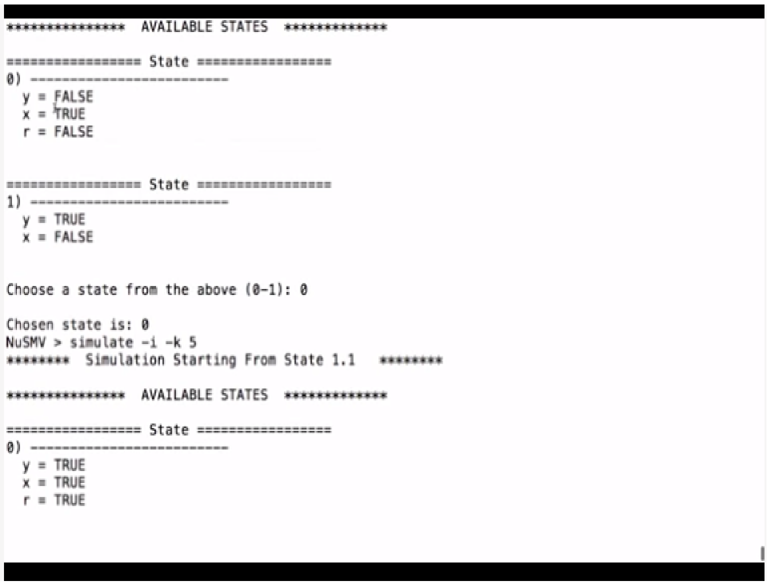
(Refer Slide Time: 05:22)



```
  r = TRUE
------- State    2 ------
  x = TRUE
  r = FALSE
------- State    3 ------
  x = FALSE
  r = TRUE
------- State    4 ------
  x = FALSE
  r = FALSE
-------------------------
#######################################################################
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = FALSE
  x = TRUE
  r = FALSE


================= State =================
1) -------------------------
  y = TRUE
  x = FALSE


Choose a state from the above (0-1): 
```

Let us now simulate this transition system we pick the initial state rather and initial state. There are 2 states one with by x equal to true or false the other one has r to be false and x to be false as well. The value of y gets determined given the values of the variables x and r. So, y was not of x r of x comma r, x r of true and false is true, not of true is false here r is false. So, XOR of false and false is false, not of false is true this is as we expected. Let me take this state and simulate for state 5 steps.

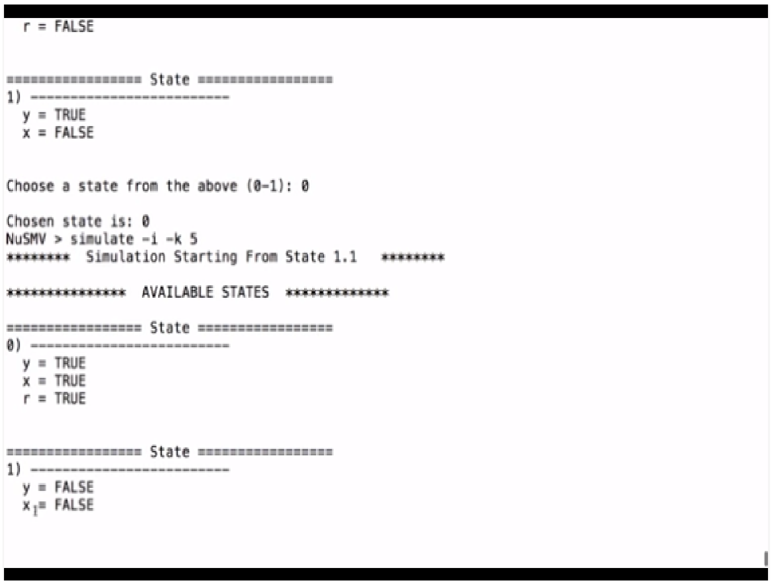
(Refer Slide Time: 06:08)



```
***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = FALSE
  x = TRUE
  r = FALSE


================= State =================
1) -------------------------
  y = TRUE
  x = FALSE


Choose a state from the above (0-1): 0

Chosen state is: 0
NuSMV > simulate -i -k 5
********  Simulation Starting From State 1.1   ********

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = TRUE
  x = TRUE
  r = TRUE
```

So, we started with the state x equal to true r equal to false and y is equal to false. The next value of r should be XOR of true and false which is true.

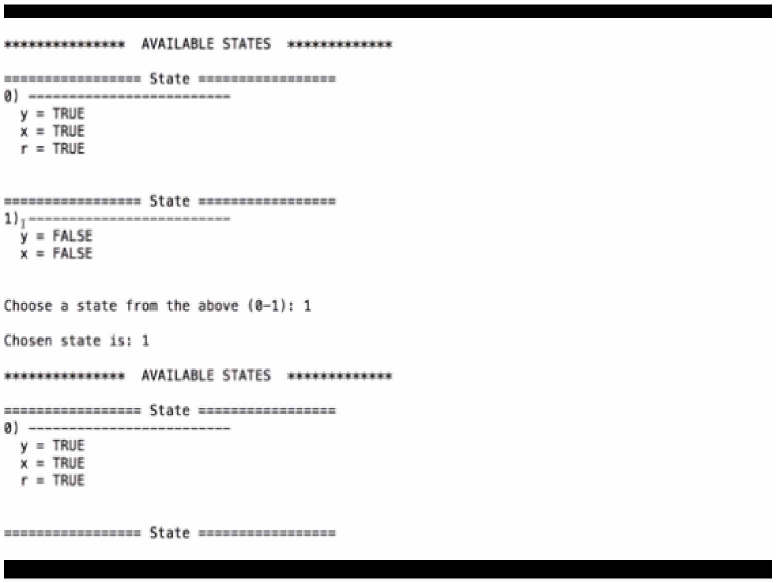
(Refer Slide Time: 06:28)



```
  r = FALSE


================= State =================
1) -------------------------
  y = TRUE
  x = FALSE


Choose a state from the above (0-1): 0

Chosen state is: 0
NuSMV > simulate -i -k 5
********  Simulation Starting From State 1.1   ********

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = TRUE
  x = TRUE
  r = TRUE


================= State =================
1) -------------------------
  y = FALSE
  x = FALSE
```

If you see in the available states the value of r is true so if r is not given here you have to look here. So, r is true however x could be either true or false so that's why get 2 states.

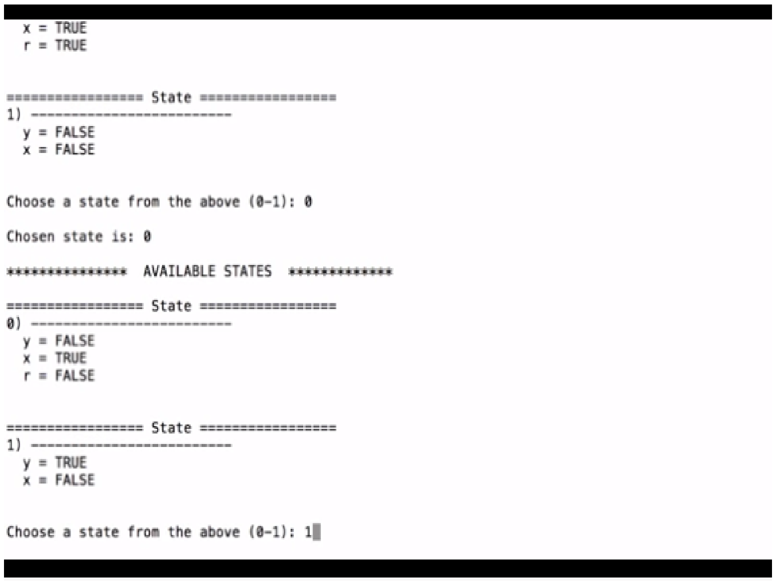
(Refer Slide Time: 06:44)



```
***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = TRUE
  x = TRUE
  r = TRUE


================= State =================
1) -------------------------
  y = FALSE
  x = FALSE


Choose a state from the above (0-1): 1

Chosen state is: 1

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = TRUE
  x = TRUE
  r = TRUE


================= State =================
```

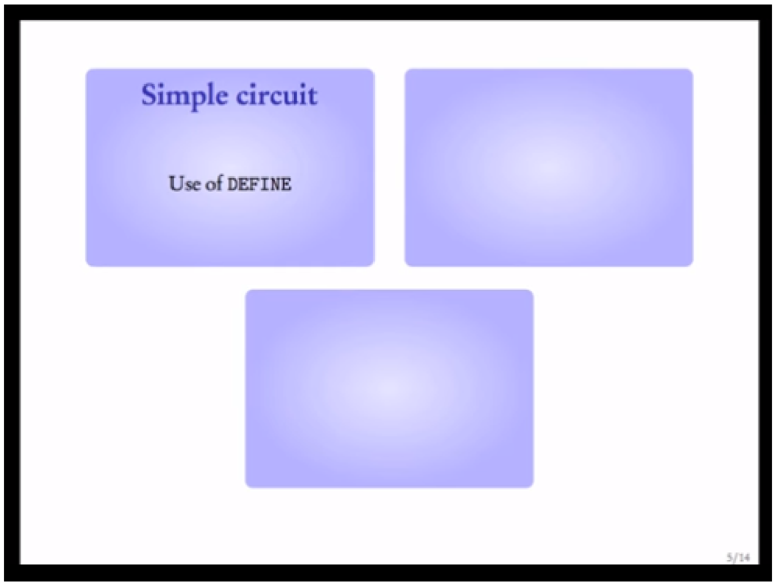
Let me choose 1, here the value of r is true, XOR of true and false is true so r is going to be true here and x could be either true or false so you can continue doing this.

(Refer Slide Time: 06:59)



```
  x = TRUE
  r = TRUE


================= State =================
1) -------------------------
  y = FALSE
  x = FALSE


Choose a state from the above (0-1): 0

Chosen state is: 0

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  y = FALSE
  x = TRUE
  r = FALSE


================= State =================
1) -------------------------
  y = TRUE
  x = FALSE


Choose a state from the above (0-1): 1
```

(Refer Slide Time: 07:05)



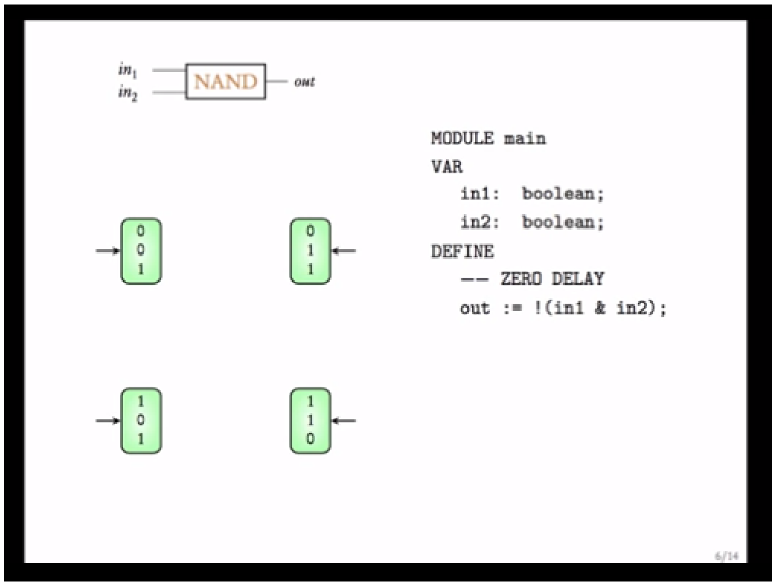
Simple circuit

Use of DEFINE

5/14

Each time we have 2 successor transitions that was an example of simple circuit. What you need to remember is the use of the keyword define, define is useful whenever the output value is determined by the current values of the inputs.

(Refer Slide Time: 08:51)



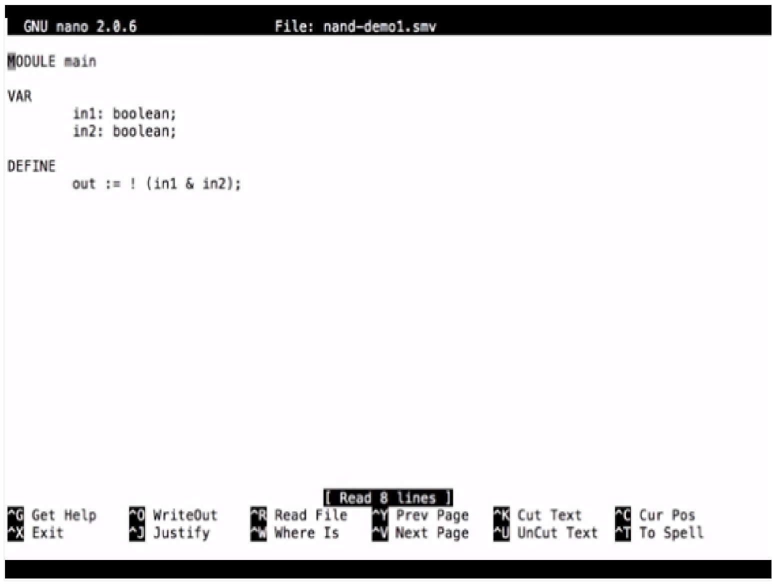
```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

6/14

Let us now look at more examples of circuits, this is a simple NAND with two inputs. Let us now describes this very simple circuit using NuSMV. Module main there are 2 variables in1 and in2 we have assumed that there is a 0 delay here, that is out is determined as not of in1 and in2 immediately with 0 delay. So, we use the define keyword what is the transition system representing this circuit?

There are 4 states determined by the values of the inputs and this is in1, this is in2, this is out. Out is NAND of 0, 0 which is 1 here NAND of 1 and 0, AND of 1 and 0 is 0 and not of 1 sorry not of 0 is 1. The only place where out is 0 is here where AND of 1 and 1 is 1 and not of 1 is 0. We have not assigned any initial states so all 4 of them are going to be initial.

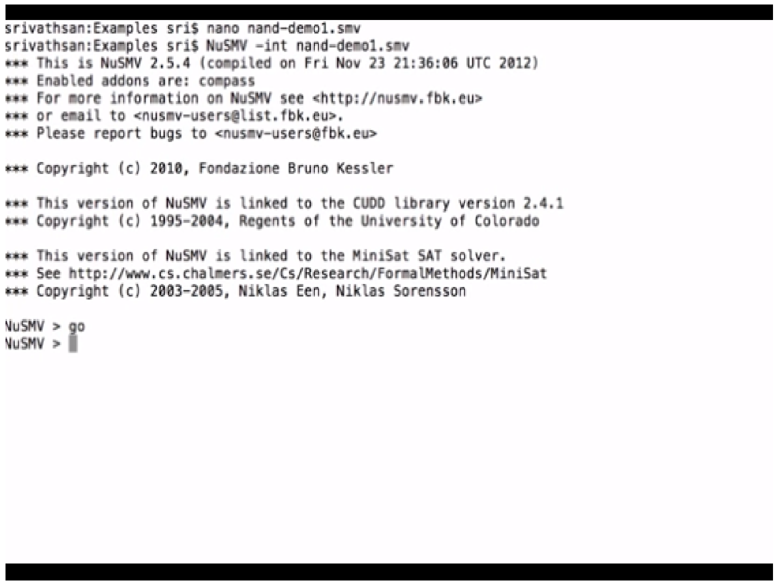
(Refer Slide Time: 08:54)



```
  GNU nano 2.0.6                 File: nand-demo1.smv

MODULE main

VAR
        in1: boolean;
        in2: boolean;

DEFINE
        out := ! (in1 & in2);
```

```
                                        [ Read 8 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

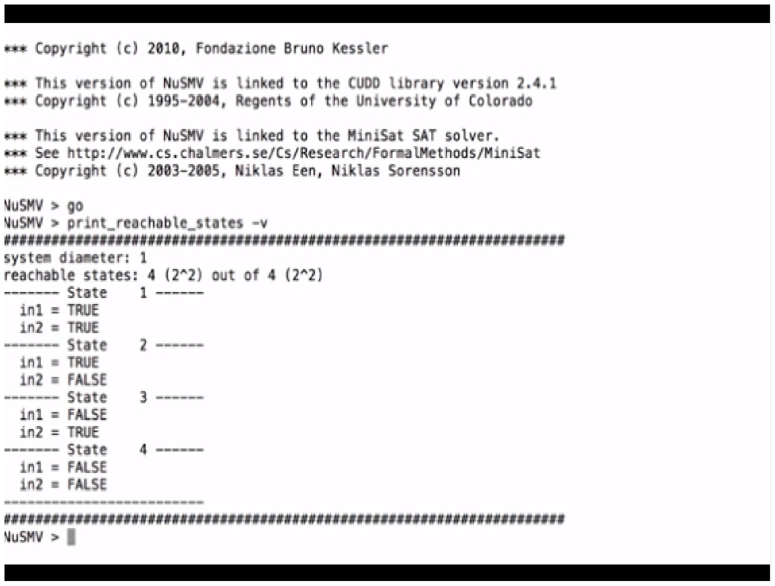
(Refer Slide Time: 09:10)



```
srivathsan:Examples sri$ nano nand-demo1.smv
srivathsan:Examples sri$ NuSMV -int nand-demo1.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV >
```

Let us simultaneously see a demo of this example in NuSMV. Here is the code lets us start running it.

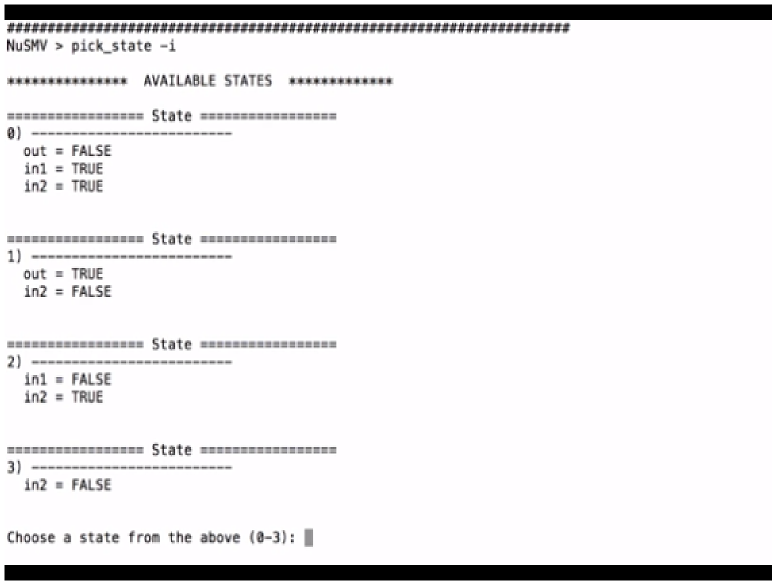
(Refer Slide Time: 09:20)



```
*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV > print_reachable_states -v
######################################################################
system diameter: 1
reachable states: 4 (2^2) out of 4 (2^2)
------- State    1 ------
  in1 = TRUE
  in2 = TRUE
------- State    2 ------
  in1 = TRUE
  in2 = FALSE
------- State    3 ------
  in1 = FALSE
  in2 = TRUE
------- State    4 ------
  in1 = FALSE
  in2 = FALSE
-------------------------
######################################################################
NuSMV >
```

Let’s first get the set of reachable states there are 4 states depending on the values of inputs in1 and in2.

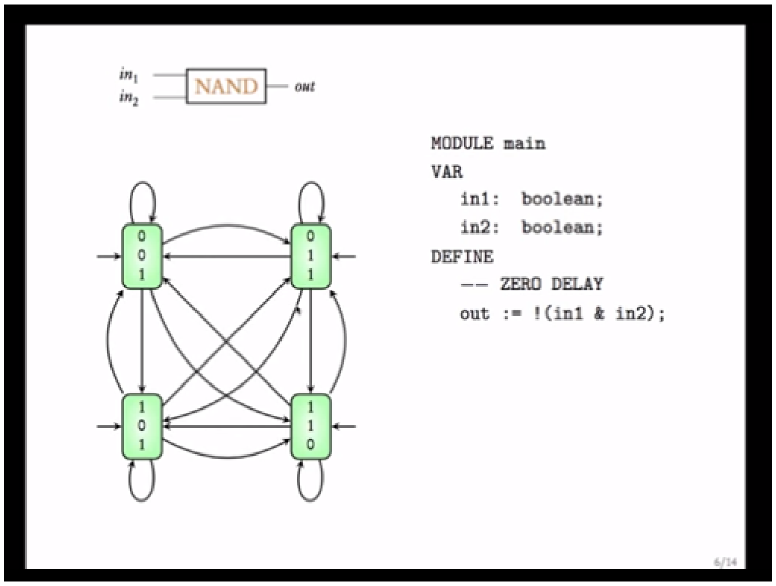
(Refer Slide Time: 09:33)



```
######################################################################
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  out = FALSE
  in1 = TRUE
  in2 = TRUE


================= State =================
1) -------------------------
  out = TRUE
  in2 = FALSE


================= State =================
2) -------------------------
  in1 = FALSE
  in2 = TRUE


================= State =================
3) -------------------------
  in2 = FALSE


Choose a state from the above (0-3):
```

What are the initial states? Since we have not defined what the initial state is any of them could be the initial state this is what we have seen here.

(Refer Slide Time: 09:42)

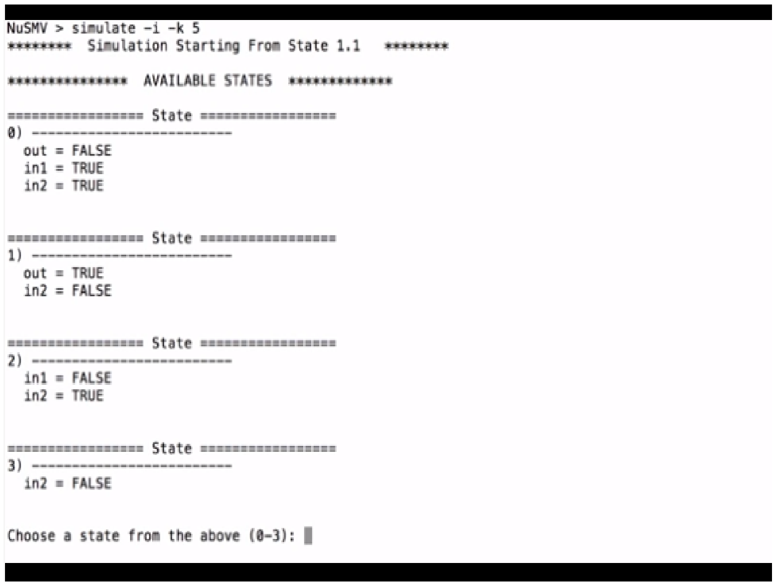


```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

6/14

What are the transitions? We have not defined anything here, the assigned block is completely empty. So, each state can move to any of the 4 states there are all possible transitions. Let us now see this in NuSMV.

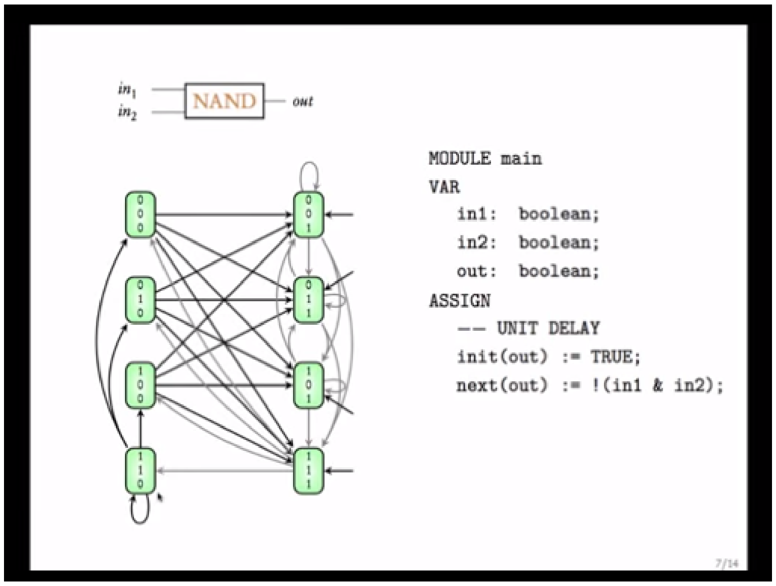
(Refer Slide Time: 10:13)



```
NuSMV > simulate -i -k 5
********  Simulation Starting From State 1.1   ********

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  out = FALSE
  in1 = TRUE
  in2 = TRUE


================= State =================
1) -------------------------
  out = TRUE
  in2 = FALSE


================= State =================
2) -------------------------
  in1 = FALSE
  in2 = TRUE


================= State =================
3) -------------------------
  in2 = FALSE


Choose a state from the above (0-3):
```

We pick say state 0 and now simulate the 5 steps you will get all possible options all possible 4 states as available no matter whatever what state you choose. This clearly shows that the transition system corresponding to this code is this. Let us now slightly complicate the code.

(Refer Slide Time: 12:09)



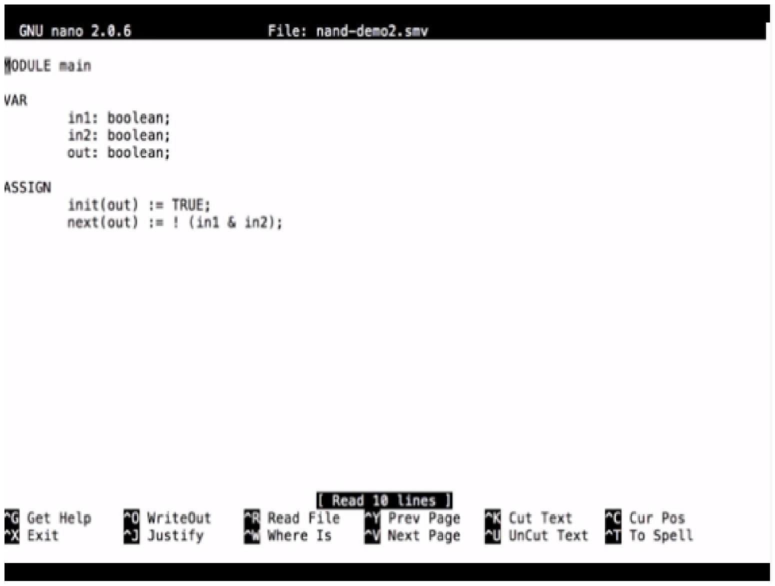
```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

Instead of assuming that the output comes within a 0 delay I am now assuming that the output comes after a unit delay. So, we define out in the VAR block we give some initial value for the output and say that once I have my inputs in1 and in2, the next value of the output is NAND of in1 and in2.

If a time t equal to 0 i have 1 and 0 as inputs, at time t equal to 1 the output will be NAND of 1 and 0 which is 1. The output at time t equal to 2 would be determined by the values of inputs at time t equals to 1 and so on. Let us now start building the transition system, since there are 3 variables in the VAR block can you guess the number of states in the transition system. It would be 2 times, 2 times to this eight states.

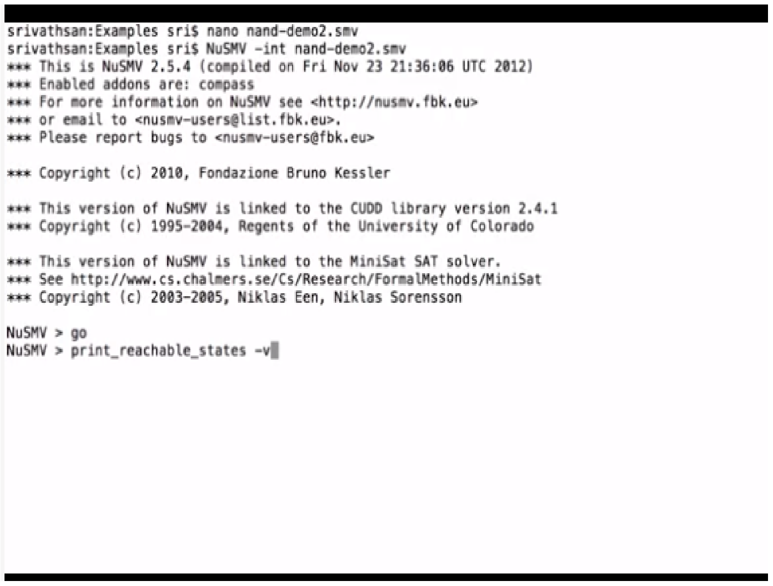
Since we have said that the initial value of out is true there are 4 initial states the once where out is 1.

(Refer Slide Time: 12:12)



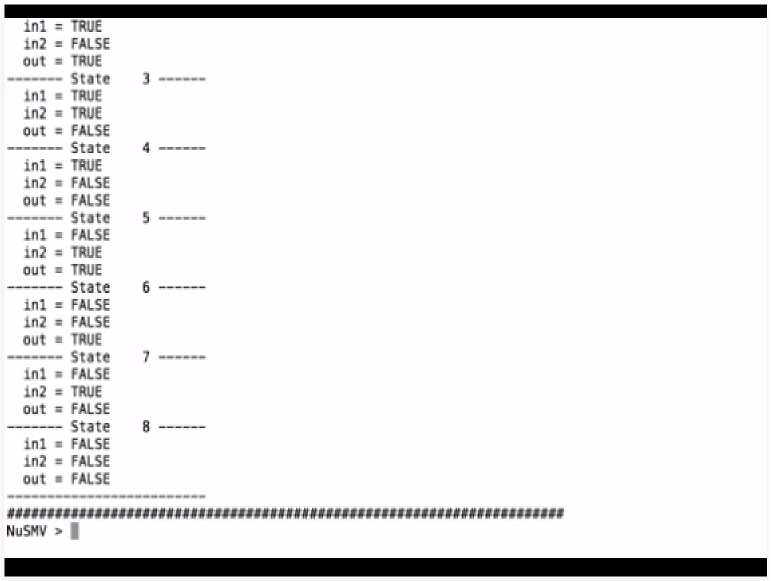
```
GNU nano 2.0.6                File: nand-demo2.smv

MODULE main

VAR
        in1: boolean;
        in2: boolean;
        out: boolean;

ASSIGN
        init(out) := TRUE;
        next(out) := ! (in1 & in2);
```

(Refer Slide Time: 12:27)



```
srivathsan:Examples sri$ nano nand-demo2.smv
srivathsan:Examples sri$ NuSMV -int nand-demo2.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV > print_reachable_states -v
```

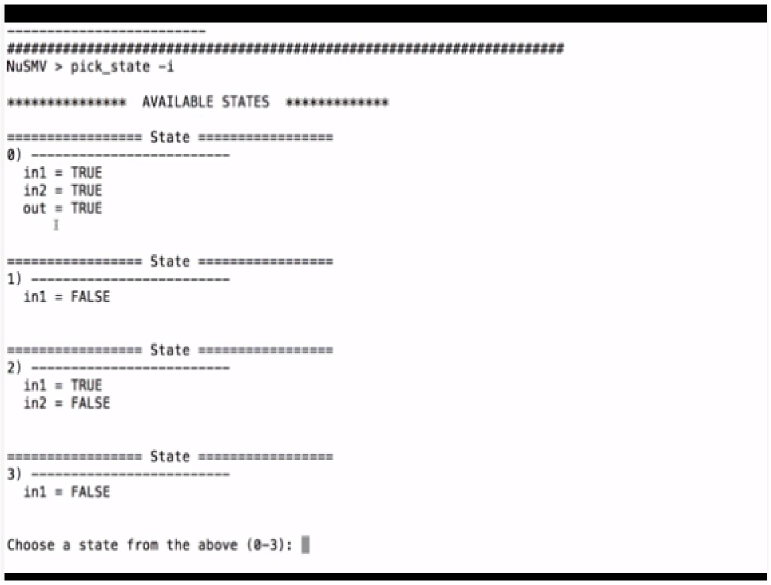
(Refer Slide Time: 12:28)



```
  in1 = TRUE
  in2 = FALSE
  out = TRUE
------- State    3 ------
  in1 = TRUE
  in2 = TRUE
  out = FALSE
------- State    4 ------
  in1 = TRUE
  in2 = FALSE
  out = FALSE
------- State    5 ------
  in1 = FALSE
  in2 = TRUE
  out = TRUE
------- State    6 ------
  in1 = FALSE
  in2 = FALSE
  out = TRUE
------- State    7 ------
  in1 = FALSE
  in2 = TRUE
  out = FALSE
------- State    8 ------
  in1 = FALSE
  in2 = FALSE
  out = FALSE
-------------------------
##########################################################################
NuSMV >
```

Let us now check this in the tool, here is the code. Let us start with printing the reachable states, see there are 8 states depending on the values of the variables in1 in2 and out.

(Refer Slide Time: 12:45)



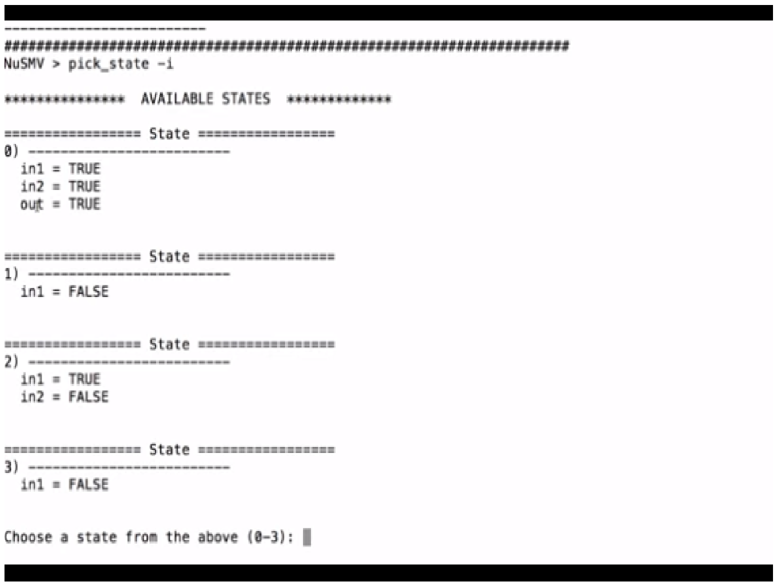
```
-------------------------
##########################################################################
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  in1 = TRUE
  in2 = TRUE
  out = TRUE

================= State =================
1) -------------------------
  in1 = FALSE


================= State =================
2) -------------------------
  in1 = TRUE
  in2 = FALSE


================= State =================
3) -------------------------
  in1 = FALSE


Choose a state from the above (0-3):
```

What are the initial states? There are 4 initial states all of them have the value of the variable out to be true, if you remember i said if the value of a variable is not written here then we have to look at the previous state. So, here this is a state where out is true, this is the state with out equal to true this is yet again a state with out equal to true.

We have 4 initial states, what are the transitions? The transition function says that the next value of output is given by not of in1 and in2, the next value of in1 and in2 is nondeterministic. Let's start from this state, the next value of output would be NAND of 1 and 1, which is 0.

There are 4 possible transitions because you just need the value of out to be 0 in1 and in2 could be any of the possibilities. Now, consider this state here the value of in1 and in2 is 0 and 0 AND of this is 0, NAND of this is 1 this would go to states where out is 1. Similarly, for these 2 states the successors are the once where the out is 1. Here it's the same the only difference is here in the state where the value of in1 and in2 are both 1 so AND of this is 1, NAND of this is going to be 0, so this goes to all these states.

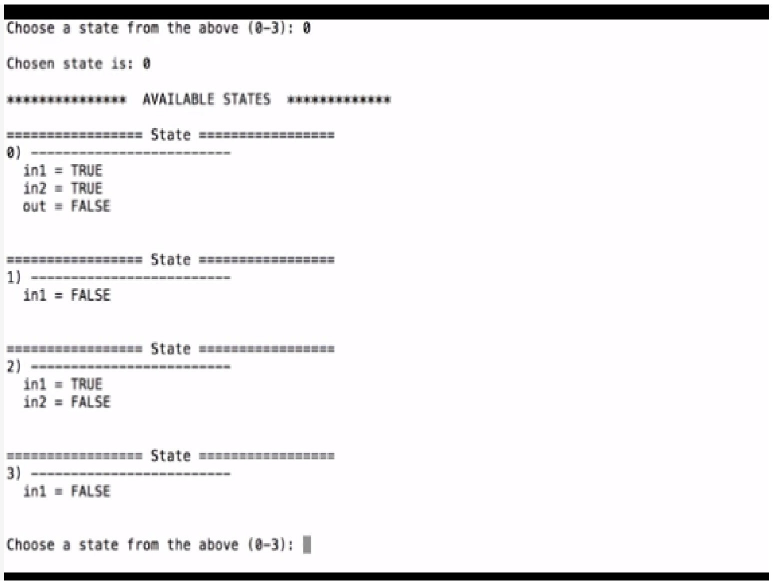
(Refer Slide Time: 14:38)



```
-------------------------
########################################################################
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  in1 = TRUE
  in2 = TRUE
  out = TRUE


================= State =================
1) -------------------------
  in1 = FALSE


================= State =================
2) -------------------------
  in1 = TRUE
  in2 = FALSE


================= State =================
3) -------------------------
  in1 = FALSE


Choose a state from the above (0-3):
```

Let us now check this, let us choose the state where in1, in2 and out are all true. There were 4 possible transitions each of them would have out to be false. Let us choose this state both in1 and in2 are true. So the successors should be the same set of states let us check it, Yes indeed so.

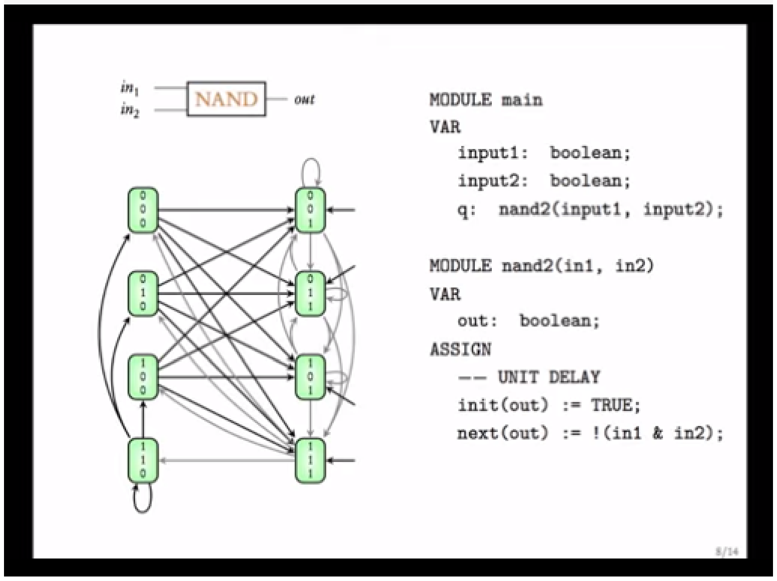
(Refer Slide Time: 15:09)



```
Choose a state from the above (0-3): 0

Chosen state is: 0

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  in1 = TRUE
  in2 = TRUE
  out = FALSE


================= State =================
1) -------------------------
  in1 = FALSE


================= State =================
2) -------------------------
  in1 = TRUE
  in2 = FALSE


================= State =================
3) -------------------------
  in1 = FALSE


Choose a state from the above (0-3): 
```

If i take 2 then there will be 4 successors where the value of output is true. This is the case so you can keep checking this.

(Refer Slide Time: 15:32)

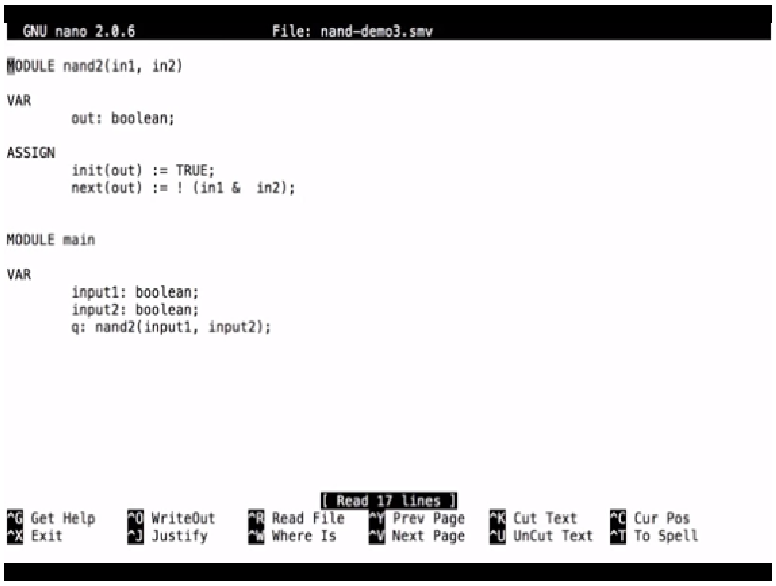


```
MODULE main
VAR
   input1:  boolean;
   input2:  boolean;
   q:  nand2(input1, input2);

MODULE nand2(in1, in2)
VAR
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

Let me now give another way of writing the same code. We are going to define a new module with the name NAND this is possible in NuSMV. We define a module NAND with 2 inputs in1, in2. It has 1 output under the VAR block there is a variable out which is boolean the next value of out is given by the statement how do we use this module in the main.

In the main you define 2 variables input1 input2 and you can call the module nand2 of input1, input2. Here the states are going to be determined by the values of input1, input2 and q, q has the variable out.

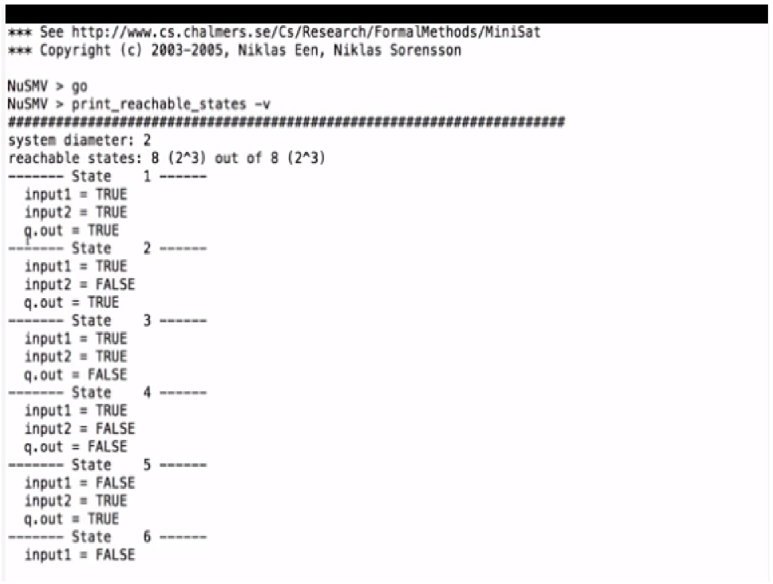
(Refer Slide Time: 16:39)



```
  GNU nano 2.0.6              File: nand-demo3.smv

MODULE nand2(in1, in2)

VAR
        out: boolean;

ASSIGN
        init(out) := TRUE;
        next(out) := ! (in1 &  in2);


MODULE main

VAR
        input1: boolean;
        input2: boolean;
        q: nand2(input1, input2);







                                    [ Read 17 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Let us see how NuSMV treats this code. Here is the code where NAND is defined as a separate module the main module calls nand2.

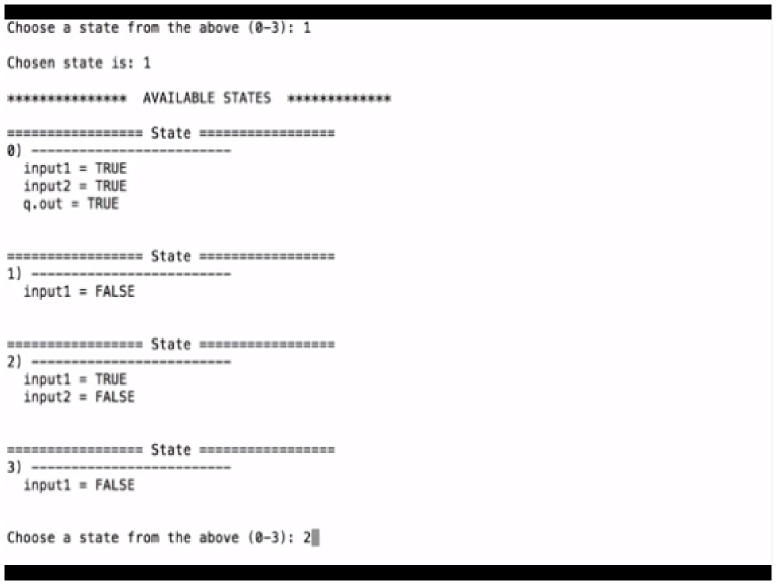
(Refer Slide Time: 17:05)



```
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
NuSMV > print_reachable_states -v
######################################################################
system diameter: 2
reachable states: 8 (2^3) out of 8 (2^3)
------- State    1 ------
  input1 = TRUE
  input2 = TRUE
  q.out = TRUE
------- State    2 ------
  input1 = TRUE
  input2 = FALSE
  q.out = TRUE
------- State    3 ------
  input1 = TRUE
  input2 = TRUE
  q.out = FALSE
------- State    4 ------
  input1 = TRUE
  input2 = FALSE
  q.out = FALSE
------- State    5 ------
  input1 = FALSE
  input2 = TRUE
  q.out = TRUE
------- State    6 ------
  input1 = FALSE
```

Let us first look at the set of states there are eight states. Now, you see instead of in1, in2, out you have input1, input2 and q dot out. This is going to be the same as before there is no change except that we have the new variable names input1 input2 and q dot out.

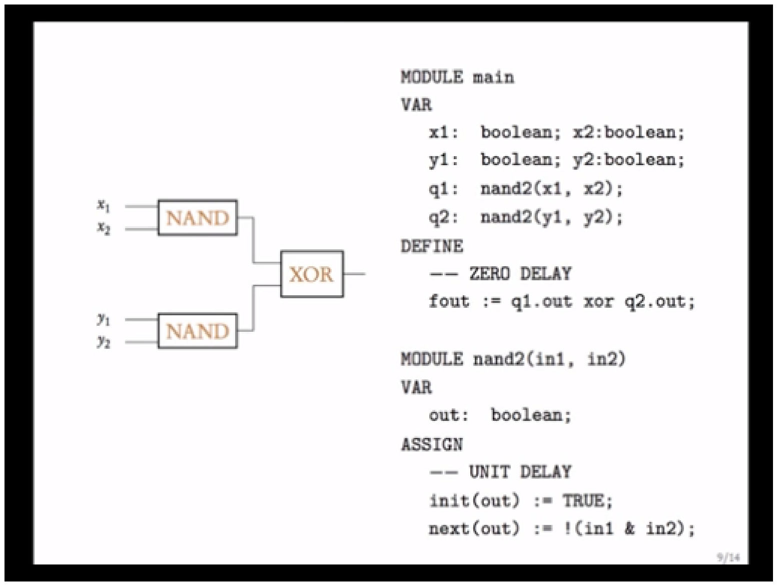
(Refer Slide Time: 17:30)



```
Choose a state from the above (0-3): 1

Chosen state is: 1

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  input1 = TRUE
  input2 = TRUE
  q.out = TRUE


================= State =================
1) -------------------------
  input1 = FALSE


================= State =================
2) -------------------------
  input1 = TRUE
  input2 = FALSE


================= State =================
3) -------------------------
  input1 = FALSE


Choose a state from the above (0-3): 2
```

Let me quickly simulate the system each time we should be getting 4 possible choices. Yes, this is how it is, if both the inputs are true then we should go to states where q dot out should be false indeed so. If 1 of them is true we should go to the states where q dot out should be true, yes indeed so. We have seen how to use modules in NuSMV code.

(Refer Slide Time: 18:12)



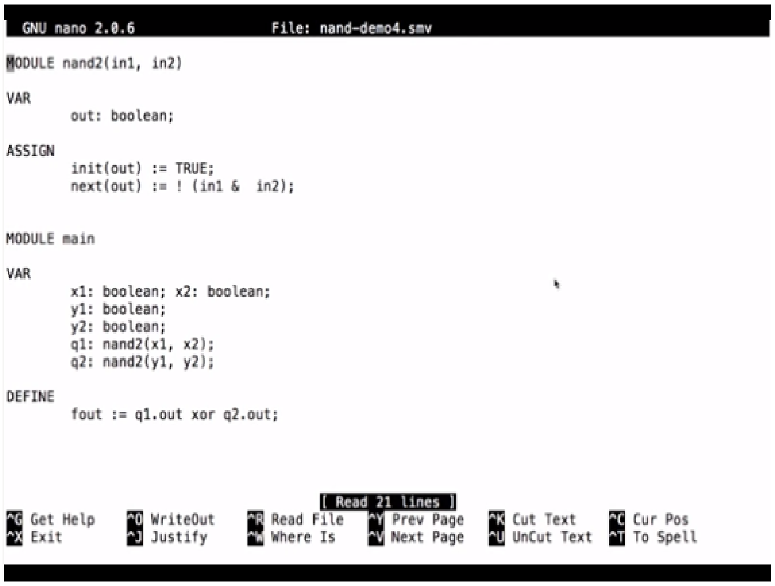
```
MODULE main
VAR
   x1:  boolean; x2:boolean;
   y1:  boolean; y2:boolean;
   q1:  nand2(x1, x2);
   q2:  nand2(y1, y2);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;

MODULE nand2(in1, in2)
VAR
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

9/14

Modules become more useful when we reuse components; for example in this circuit there are 2 NAND gates connected to an XOR. Here, we will reuse the module for NAND 2 times we define 4 variables x1, x2, y1, y2 and we will define 2 variables q1 and q1 which are of the nand2 type.

They take us input x1 and x2, q2 takes as input y1 and y2. We have assumed that there is a 0 delay for this gate, f out is going to be the output of q1 and the output of q2 which is written as q1 dot out XOR q2 dot out this shows how useful modules can be.

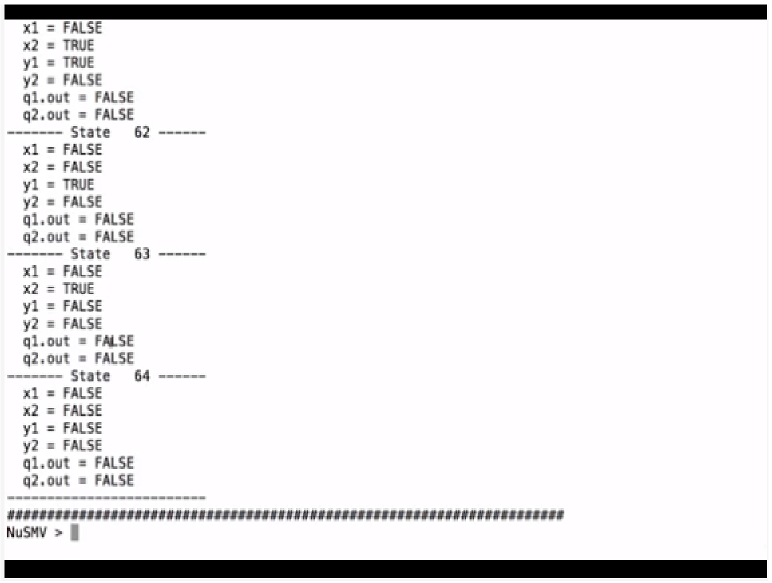
(Refer Slide Time: 19:14)



```
  GNU nano 2.0.6                File: nand-demo4.smv

MODULE nand2(in1, in2)

VAR
        out: boolean;

ASSIGN
        init(out) := TRUE;
        next(out) := ! (in1 &  in2);


MODULE main

VAR
        x1: boolean; x2: boolean;
        y1: boolean;
        y2: boolean;
        q1: nand2(x1, x2);
        q2: nand2(y1, y2);

DEFINE
        fout := q1.out xor q2.out;



                               [ Read 21 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Here, is the code where we have used the module nand2 twice. We will do the same things as before.

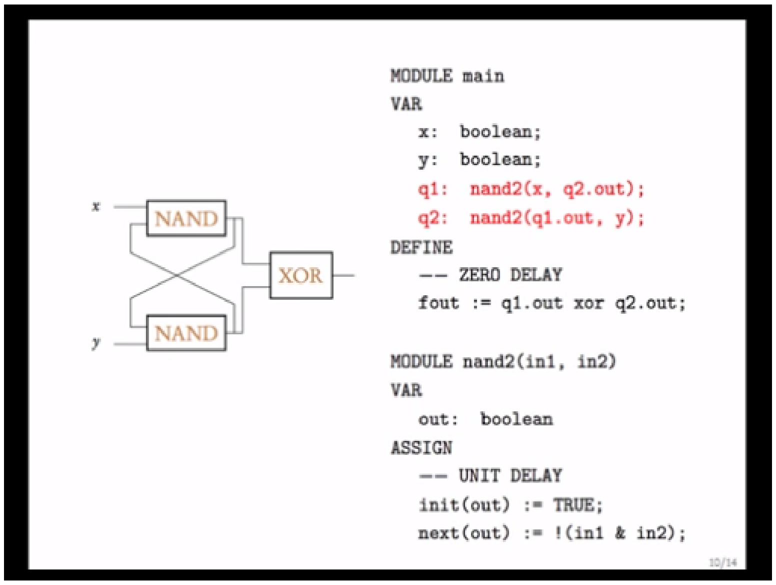
(Refer Slide Time: 19:49)



```
  x1 = FALSE
  x2 = TRUE
  y1 = TRUE
  y2 = FALSE
  q1.out = FALSE
  q2.out = FALSE
------- State   62 ------
  x1 = FALSE
  x2 = FALSE
  y1 = TRUE
  y2 = FALSE
  q1.out = FALSE
  q2.out = FALSE
------- State   63 ------
  x1 = FALSE
  x2 = TRUE
  y1 = FALSE
  y2 = FALSE
  q1.out = FALSE
  q2.out = FALSE
------- State   64 ------
  x1 = FALSE
  x2 = FALSE
  y1 = FALSE
  y2 = FALSE
  q1.out = FALSE
  q2.out = FALSE
-------------------------
########################################################################
NuSMV >
```

There are lot of states now, Why? because there are more variables. The variable are x1, x2, y1, y2, q1 dot out, q2 dot out. So, there are 2 times, 2 times, 2 times, 2 times, 2 times, to 2 power six states which is 64.

(Refer Slide Time: 20:06)



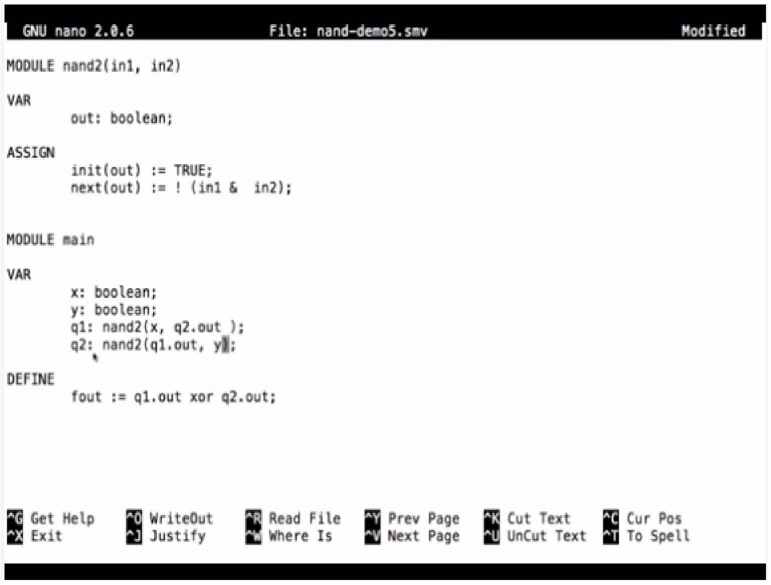
```
MODULE main
VAR
   x:  boolean;
   y:  boolean;
   q1:  nand2(x, q2.out);
   q2:  nand2(q1.out, y);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;

MODULE nand2(in1, in2)
VAR
   out:  boolean
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

10/14

Here is an another place where modules become very useful. In this circuit the output of the first NAND gate becomes the input of the second, and the output of the second becomes the input of the first this is a hierarchical design. How do we write this? x there are 2 inputs x and y which are of the boolean type. There is a variable of type nand2 which takes as input x and q2 dot out.

There is an another NAND variable q2 which takes as input q1 dot out and y. Let us now try to simulate the transition system of this circuit.

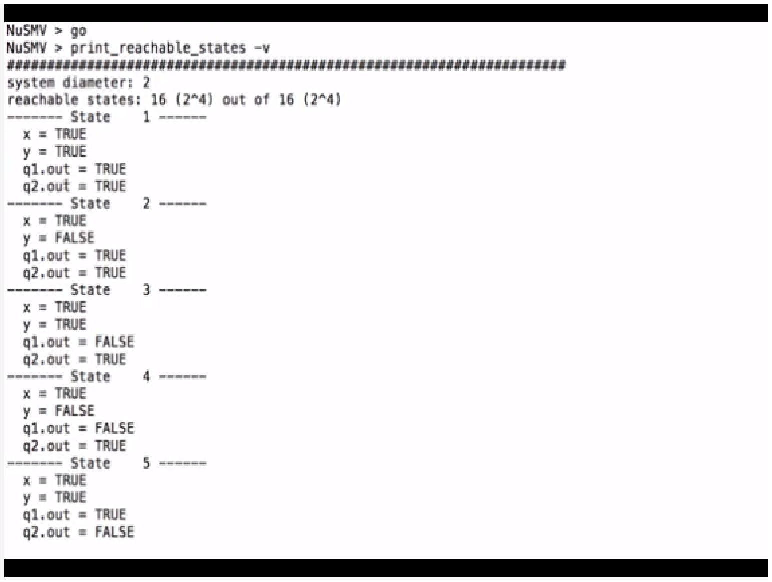
(Refer Slide Time: 21:07)



```
GNU nano 2.0.6                File: nand-demo5.smv                        Modified

MODULE nand2(in1, in2)

VAR
        out: boolean;

ASSIGN
        init(out) := TRUE;
        next(out) := ! (in1 &  in2);


MODULE main

VAR
        x: boolean;
        y: boolean;
        q1: nand2(x, q2.out );
        q2: nand2(q1.out, y);

DEFINE
        fout := q1.out xor q2.out;
```

Here is the code there are 2 nand2 type variables 1 uses inputs x and q2 dot out and the other uses inputs q1 dot and y.

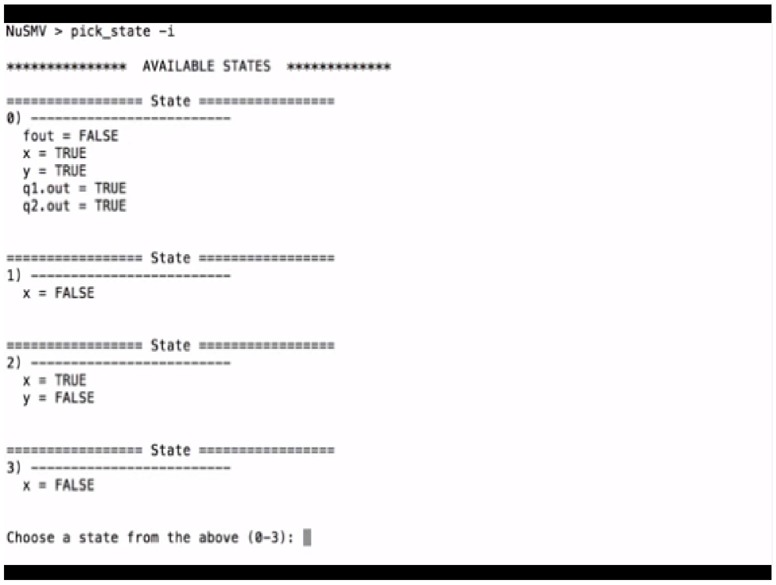
(Refer Slide Time: 21: 35)



```
NuSMV > go
NuSMV > print_reachable_states -v
######################################################################
system diameter: 2
reachable states: 16 (2^4) out of 16 (2^4)
------- State    1 ------
  x = TRUE
  y = TRUE
  q1.out = TRUE
  q2.out = TRUE
------- State    2 ------
  x = TRUE
  y = FALSE
  q1.out = TRUE
  q2.out = TRUE
------- State    3 ------
  x = TRUE
  y = TRUE
  q1.out = FALSE
  q2.out = TRUE
------- State    4 ------
  x = TRUE
  y = FALSE
  q1.out = FALSE
  q2.out = TRUE
------- State    5 ------
  x = TRUE
  y = TRUE
  q1.out = TRUE
  q2.out = FALSE
```

The reachable states are 16 in number because there were only 4 variables x, y, q1 dot out and q2 dot out.

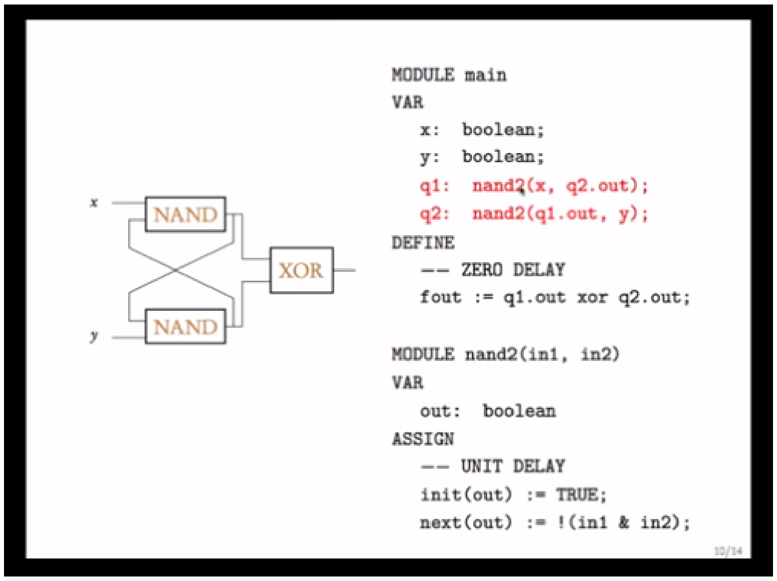
(Refer Slide Time: 21:50)



```
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  fout = FALSE
  x = TRUE
  y = TRUE
  q1.out = TRUE
  q2.out = TRUE


================= State =================
1) -------------------------
  x = FALSE


================= State =================
2) -------------------------
  x = TRUE
  y = FALSE


================= State =================
3) -------------------------
  x = FALSE

Choose a state from the above (0-3):
```

What is the initial state? there are 4 initial states, firstly both q1 dot out and q2 dot out are true initially.

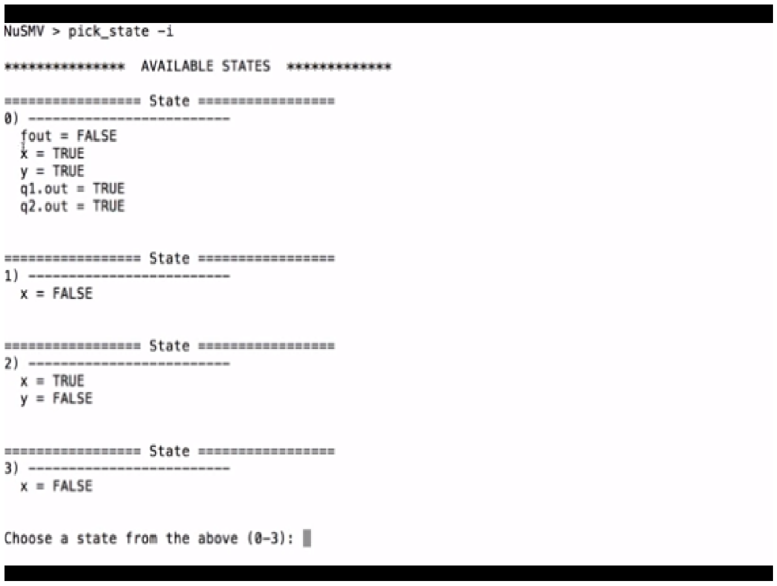
(Refer Slide Time: 22:03)



```
MODULE main
VAR
   x:  boolean;
   y:  boolean;
   q1:  nand2(x, q2.out);
   q2:  nand2(q1.out, y);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;

MODULE nand2(in1, in2)
VAR
   out:  boolean
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

That is how we have defined q1 is of the nand 2 type it has a variable out of boolean type which is initially true so both q1 dot out and q2 dot out would be true.

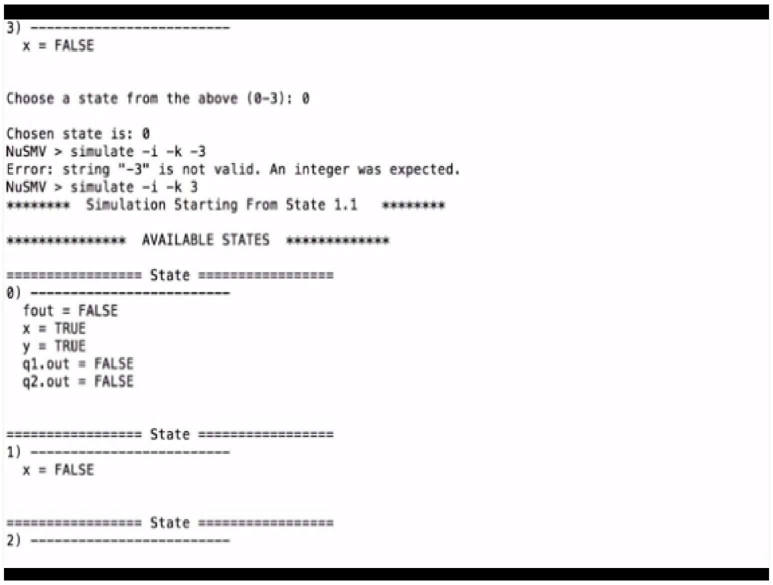
(Refer Slide Time: 22:17)



```
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  fout = FALSE
  x = TRUE
  y = TRUE
  q1.out = TRUE
  q2.out = TRUE


================= State =================
1) -------------------------
  x = FALSE


================= State =================
2) -------------------------
  x = TRUE
  y = FALSE


================= State =================
3) -------------------------
  x = FALSE


Choose a state from the above (0-3):
```

Lets us take the first state where x and y are true and both q1 dot out and q2 dot out are true. The next state would be determined as follows, the next of out should be not of the 2 inputs so the next of q1 dot out would be not of x and q2 dot out.

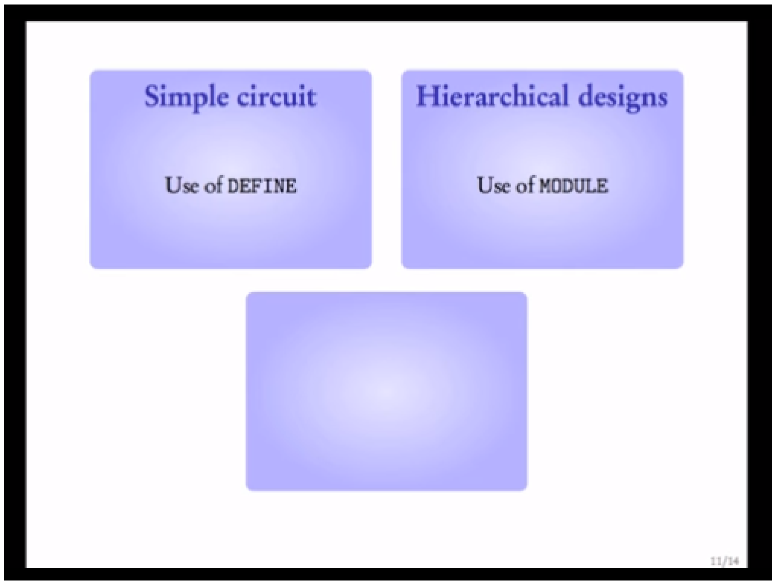
(Refer Slide Time: 22:47)



```
3) -------------------------
  x = FALSE


Choose a state from the above (0-3): 0

Chosen state is: 0
NuSMV > simulate -i -k -3
Error: string "-3" is not valid. An integer was expected.
NuSMV > simulate -i -k 3
********  Simulation Starting From State 1.1   ********

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  fout = FALSE
  x = TRUE
  y = TRUE
  q1.out = FALSE
  q2.out = FALSE


================= State =================
1) -------------------------
  x = FALSE


================= State =================
2) -------------------------
```

Let's see we have taken this the next of q1 dot out should be NAND of true and true which is false. Let's try to stimulate and check this what do you expect? In the next state q1 dot out should be false that’s how it is. Similarly, the next of q2 dot should be NAND of y and q1 dot out which is false as well, x and y have no restrictions so there are 4 states all of them have q1 dot out and q2 dot out to be false. You can continue to check the transition relation.

(Refer Slide Time: 23:55)



Simple circuit

Use of DEFINE

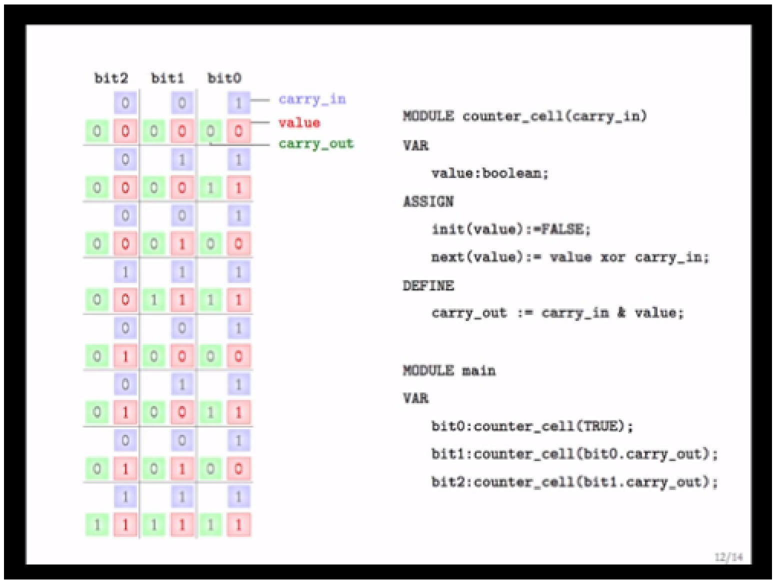
Hierarchical designs

Use of MODULE

11/14

We have seen that NuSMV code can be broken down into modules these modules can be reused conveniently.

(Refer Slide Time: 30:08)



```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;

MODULE main
VAR
    bit0:counter_cell(TRUE);
    bit1:counter_cell(bit0.carry_out);
    bit2:counter_cell(bit1.carry_out);
```

12/14

Let us now go to the next example, here is the NuSMV code of a 3 bit counter our task now is to understand this code. There is a module counter cell which takes as input a variable carry in. It has a variable value which is of boolean type. In the module main there are 3 instantiations of module counter cell the first 1 bit0 has a carry in of true always bit1 instantiates counter cell with a carry out of bit0, bit2 instantiates counter cell with the carry out of bit1.

We will slowly try to understand this code by simulating this. What are the variables in the main module bit0, bit1, bit2; bit0 is of the counter cell type. What are its variables? It has variables value, carryout and it takes in a carry in. In bit0 carry in is true initially value of bit0 is false and carry out is determined by the define statement. So, the value of carryout is given by the current values of carry in and value which is carry in and value 1 and 0 is 0.

What about bit1? bit1 is instantiated with bit0 dot carryout. So, bit1 dot carry in is 0 which is the same as this, bit1 dot value is false, bit1 dot carryout is bit1 dot carry in and bit1 dot value which is 0 and 0 which is 0. What about bit2? bit2 is instantiated with bit1 dot carryout, bit1 dot carryout is 0. So, bit2 dot carry in is 0, bit2 dot value is 0 ,bit2 dot carryout is carry in and value which is 0. This is going to be the initial state of the transition system determined by this code initially these are the values.

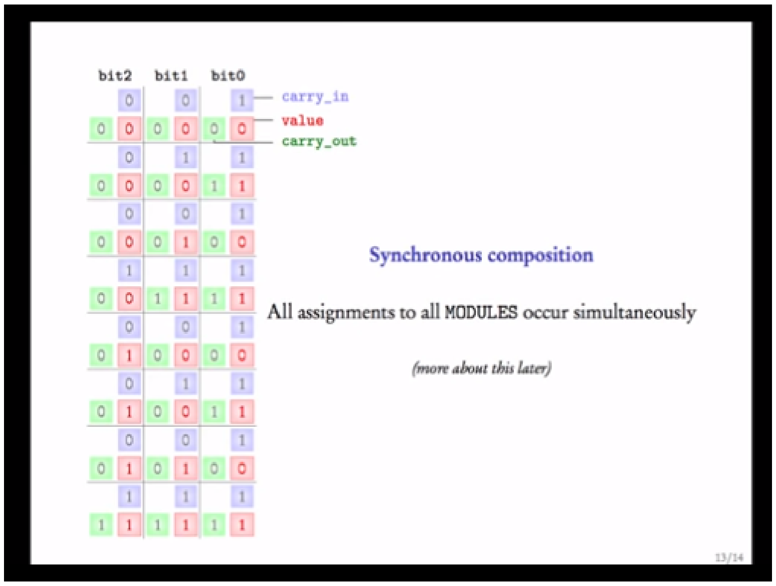
The next set of values will be determined by the next assignment next of bit0 dot value is going to be bit0 dot value x or bit0 dot carry in, which is 1 XOR is 0 1, bit0 dot carry in is always 1 because we have instantiated bit0 with counter cell of true. With these values bit0 dot carryout becomes 1 because 1 and 1 is 1, bit1 dot value is going to be 0 XOR 0 which is 0, bit1 dot carry in is the same as bit0 dot carryout, bit1 dot carryout is AND of 1 and 0 is 0.

Let us come to bit2, bit2 dot value is 0 XOR 0 which is 0, bit2 dot carry in is the same as bit1 dot carryout, bit2 dot carryout is 0. You can see the values here from 0,0,0 we came to 0,0,1 so there has been an addition of 1. What is the next step? In the next step bit0 dot value would be XOR of 1 and 1 which is 0, bit0 dot carry in always 1.

With these values bit0 dot carryout is 0, bit1 dot value is XOR of 1 and 0 which is 1, bit1 dot carry in is the same as bit0 dot carryout which is 0 and bit1 dot carryout is a AND of these 2 which is 0. There is no change here XOR of 0,0 is 0, bit1 dot carryout is 0 and AND of these 2 is 0.

Similarly, if you compute the next step you will see that from 0 1 0 the values go to 0 1 1 and then it goes to 1 0 0, 1 0 1, 1 1 1 sorry 1 1 0, 1 1 1 and back to 0 0 0 so this code is simulating a 3 bit counter. Try to simulate this code a NuSMV as an exercise.

(Refer Slide Time: 32:03)



Synchronous composition

All assignments to all MODULES occur simultaneously

(more about this later)

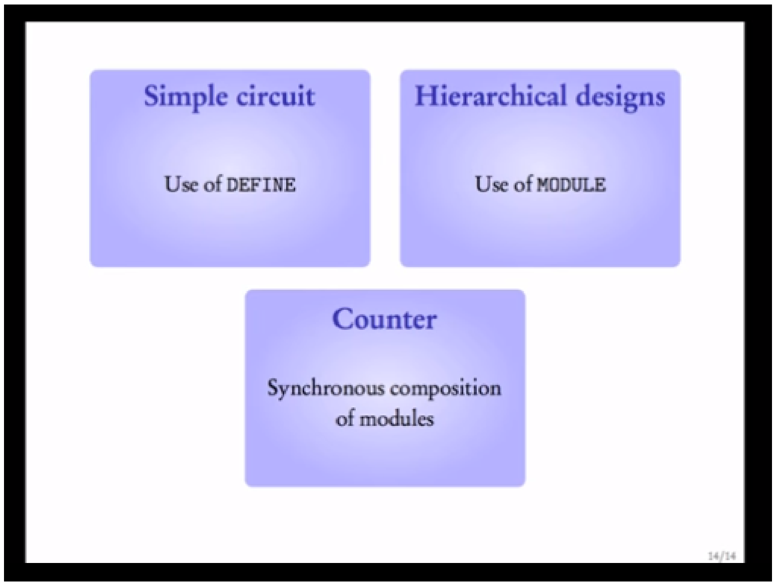
13/14

This is 1 more example where the use of modules is interesting. I would like to point out an observation about this example in particular about the use of modules in this example. This was the initial state, the said state is determined by the values of the variables these are the variables in each of the modules.

The next state is determined by evaluating the next assignment in each of these modules. Each of these modules move forward in the next state this is how the next state is determined. This kind of scenario where all the modules take the next transition in 1 step is said to be synchronous composition. All assignments to all modules occur simultaneously, all assignments to all modules will occur in the next step.

We will later to see another form of composition called synchronous composition where we can force only 1 of the module to take the next transition. However, here we have defined bit0, bit1 and bit2 to be variables of type determined by a module definition. In such a case all of them will change in the next step and we will get a synchronous composition. We will see more about this later in more detail.

(Refer Slide Time: 32:07)



Simple circuit

Use of `DEFINE`

Hierarchical designs

Use of `MODULE`

Counter

Synchronous composition of modules

14/14

Let me now summarize what we have seen in this module, we started with the definition of a simple circuit in NuSMV we saw the use of the keyword define. Then we moved on to hierarchical designs of circuits where the use of the module keyword was important. We finally saw an example of a 3 bit counter where we could explain the synchronous composition of modules.