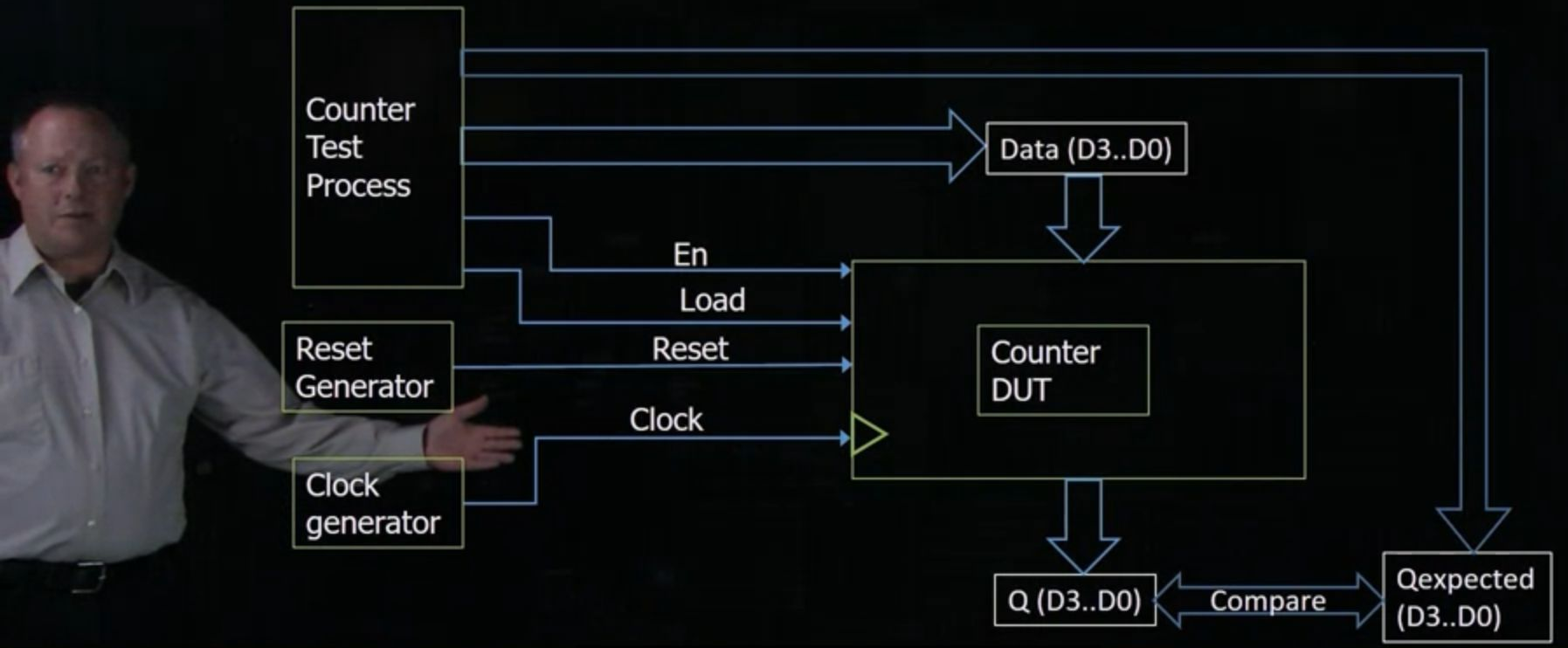
FPGA Design for Embedded Systems

Hardware Description Languages for Logic Design

University of Colorado
Boulder

Copyright © 2019 University of Colorado

Counter Test Bench

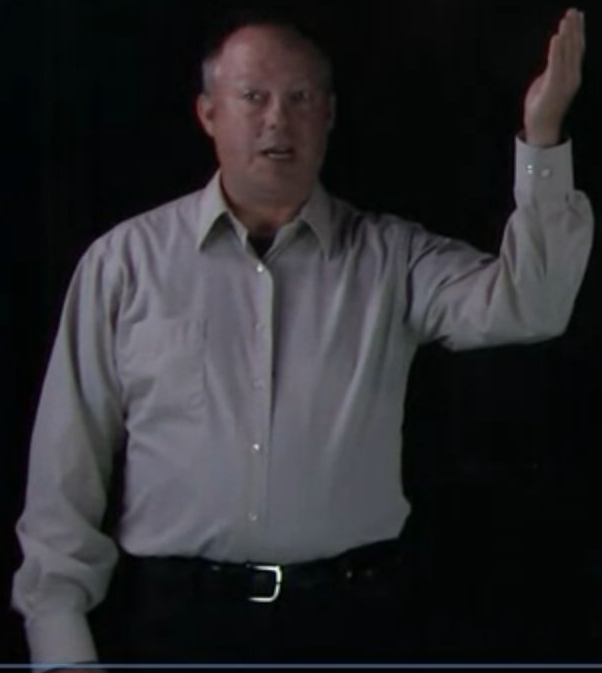


written our counter which
is our device under test,

University of Colorado
Boulder

Copyright © 2019 University of Colorado

Counter Test Bench



```
-- Testbench Entity : No port list
entity Counter_tb is  end entity Counter_tb;
-- Testbench Architecture
architecture Counter_arch of Counter_tb is
  component Counter port (
    d      : in std_logic_vector(3 downto 0);
    clk, reset, load, en : in std_logic;
    q      : out std_logic_vector(3 downto 0));
  end component Counter;

  constant delay : integer := 10; -- wait
  constant n     : integer := 4;  -- width counter
  constant T     : time    := 20 ns; -- clock period
  signal   clock : std_logic := '0';  -- clock generator
  signal   reset : std_logic := '0';  -- reset generator
```

Again, note that our entity is empty of

University of Colorado Boulder

Copyright © 2019 University of Colorado

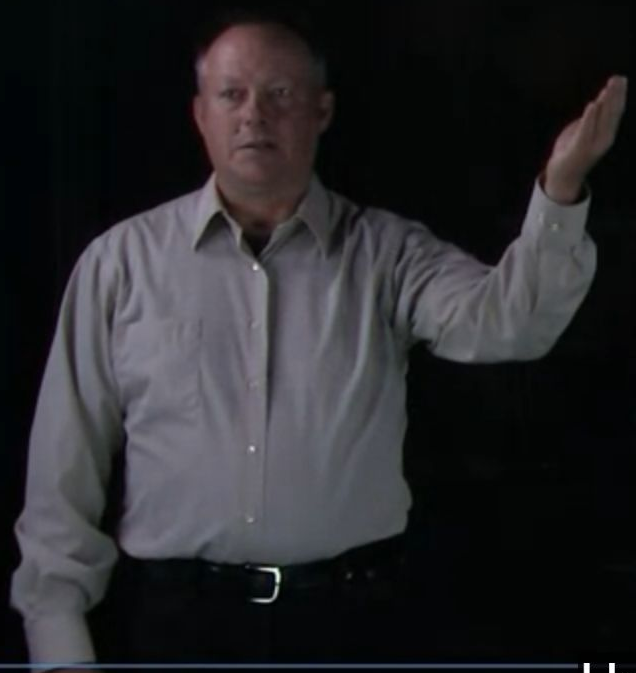
Counter Test Bench

```
-- Architecture
  signal  data_tb : std_logic_vector(n-1 downto 0) :=
    "0000";
  signal   load   : std_logic := '0';  -- stimulus
  signal   en     : std_logic := '0';  -- stimulus

  signal   q_tb   : std_logic_vector(3 downto 0);
                -- output
  signal   check  : std_logic_vector(n-1 downto 0) :=
    "0000";     -- compare to count
```

In our architecture we have the following data signal.

Share
University of Colorado Boulder
Copyright © 2019 University of Colorado
1:29 / 5:09

Counter Test Bench



```
begin
  -- DUT Instantiation
  DUT : Counter port map (
        d       => data_tb,
        clk     => clock,     reset   => reset,
        load    => load,      en      => en,
        q       => q_tb );

  clk_gen : process begin
    clock <= '0';
    wait for T/2; --  10 nsec of 0
    clock <= '1';
    wait for T/2; --  10 nsec of 1, for 20 nsec period
  end process;

  reset <= '1', '0' after 10 ns; --  10 nsec
```

Here we begin our architecture

University of Colorado Boulder

Copyright © 2019 University of Colorado

Press Esc to exit full screen

The clk_gen process presented here stops generating a clock signal after a few cycles due to the wait for T/2 statement.

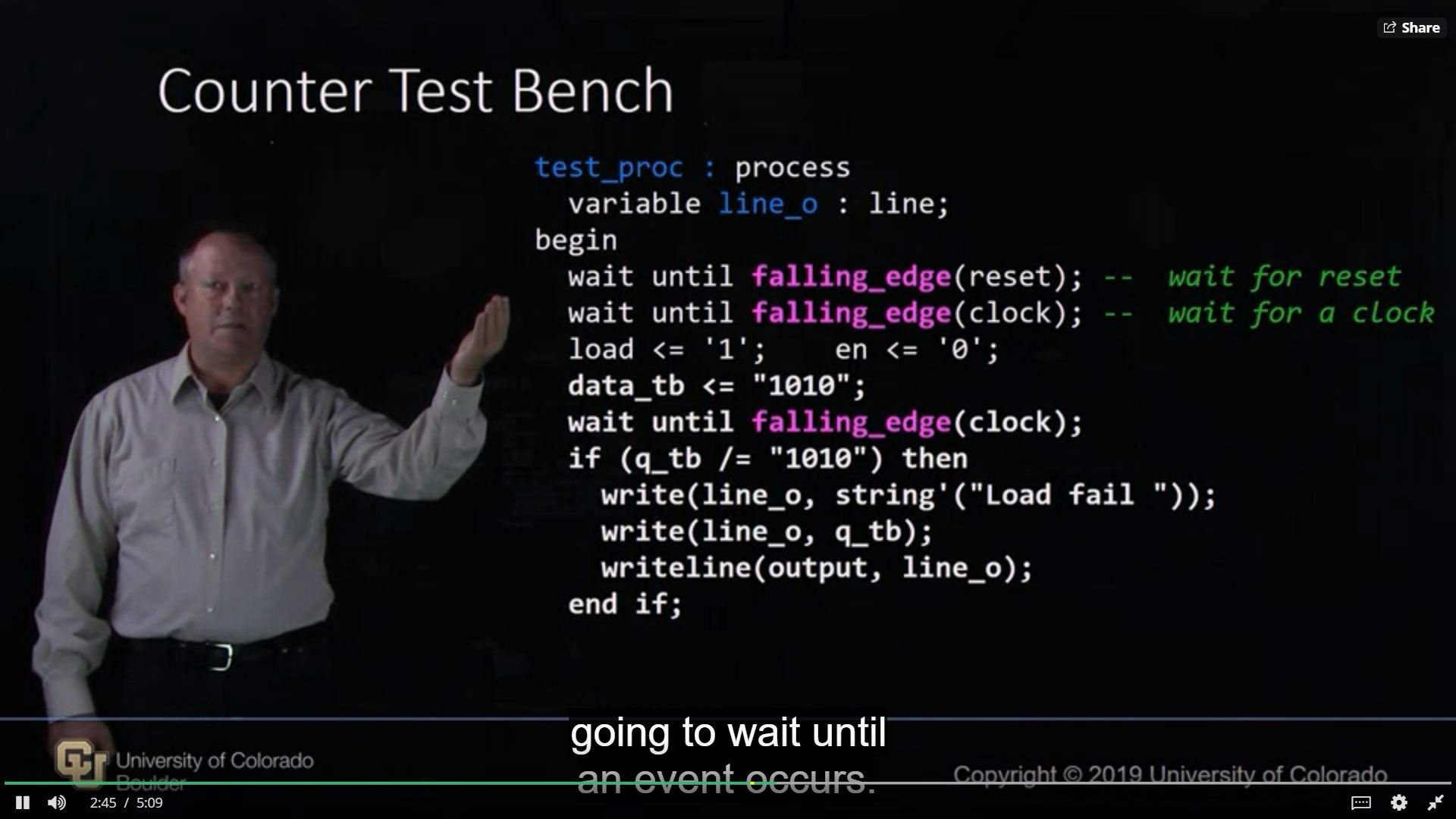
- True.
- False.

Correct
Correct. The clock process runs forever, independently and in parallel to other processes.

Continue

Copyright © 2015 University of Colorado

Counter Test Bench



```
test_proc : process
  variable line_o : line;
begin
  wait until falling_edge(reset); --  wait for reset
  wait until falling_edge(clock); --  wait for a clock
  load <= '1';    en <= '0';
  data_tb <= "1010";
  wait until falling_edge(clock);
  if (q_tb /= "1010") then
    write(line_o, string'("Load fail "));
    write(line_o, q_tb);
    writeline(output, line_o);
  end if;
```


going to wait until
an event occurs.

Copyright © 2019 University of Colorado

Share

Counter Test Bench

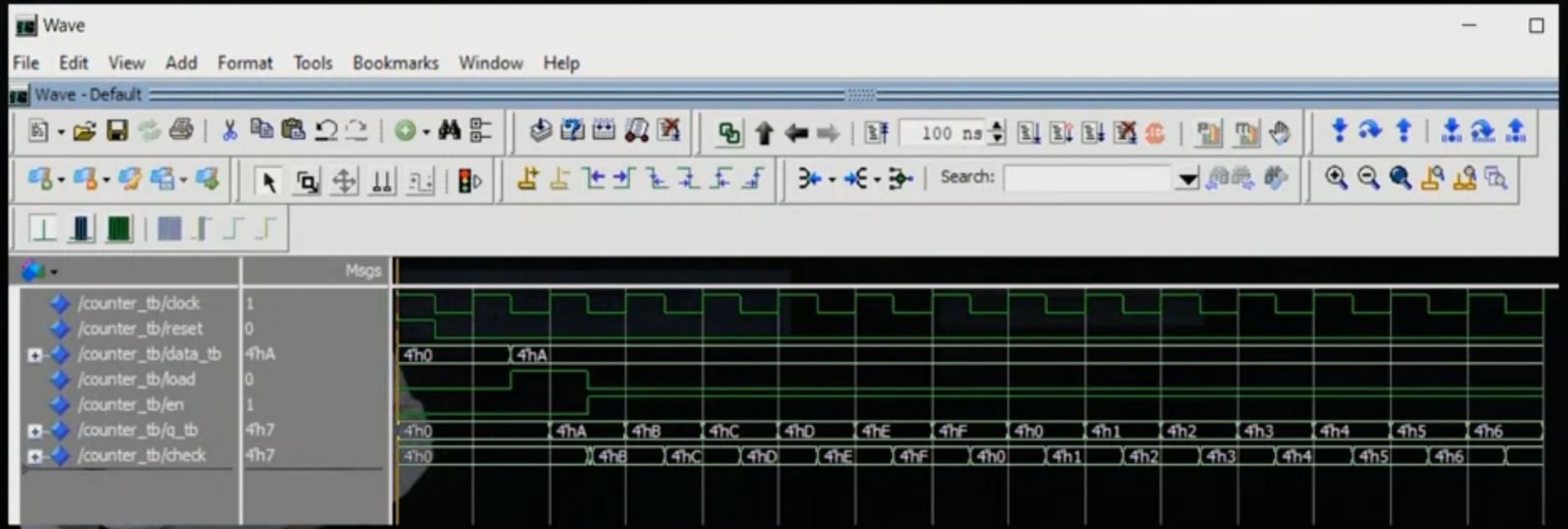
```
check <= "1010";
load  <= '0';
en    <= '1';


for i in 1 to 2**n loop
  check <= check + 1;
  wait until falling_edge(clock);
  if (q_tb /= check ) then
    report "count fail at time count" &
      time'image(now) & integer'image(i);
  end if;
end loop;
wait;
end process test_proc;
end architecture Counter_arch;
```

We continue in our process.

University of Colorado Boulder

Copyright © 2019 University of Colorado

Counter Test Bench Waves



As you can see clock is ticking along.

University of Colorado Boulder

Copyright © 2019 University of Colorado

Summary – Testbenches in VHDL II

In this video, you have learned:

- How to write testbenches for synchronous circuits
- How to use external signal generators to create stimulus for circuits

video we've learned how to write

University of Colorado Boulder

Copyright © 2019 University of Colorado

Press Esc to exit full screen

Share

FPGA Design for Embedded Systems

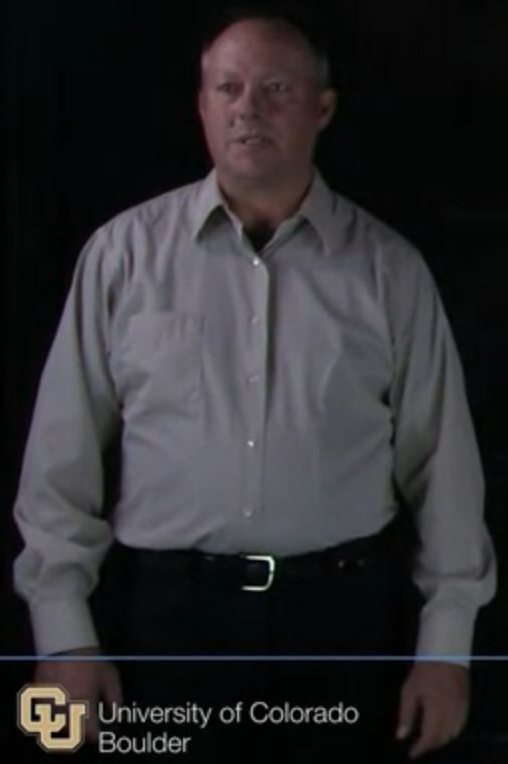
Hardware Description Languages for Logic Design

University of Colorado Boulder

Copyright © 2019 University of Colorado

0:01 / 7:39

Memory in VHDL



In this video you will learn :

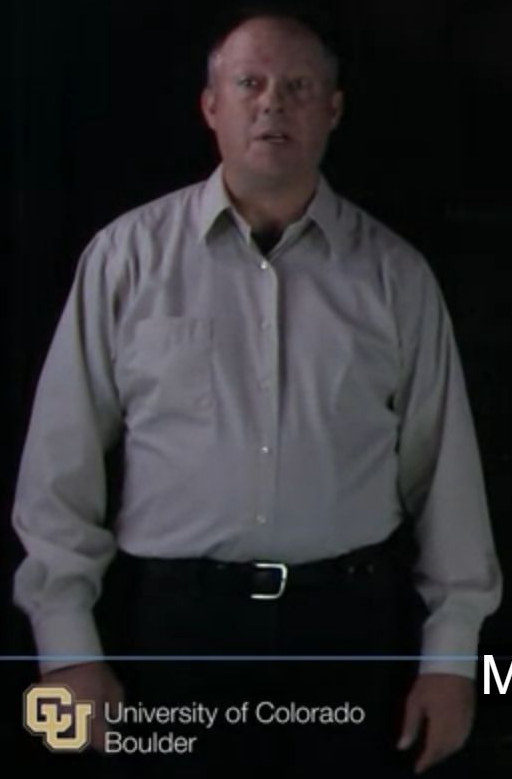
- How to create memory devices in VHDL
- How to initialize a memory using a file input
- How to create a simple Look Up Table

Hello, and welcome to FPGA design for embedded systems.

University of Colorado Boulder

yright © 2019 University of Colorado

Memory in VHDL



Memories are a common element in most digital systems.

Earlier in this course we described a register file circuit, in which individual registers were enabled for access by decoding an address.

We will extend this to include RAM and ROM memories.

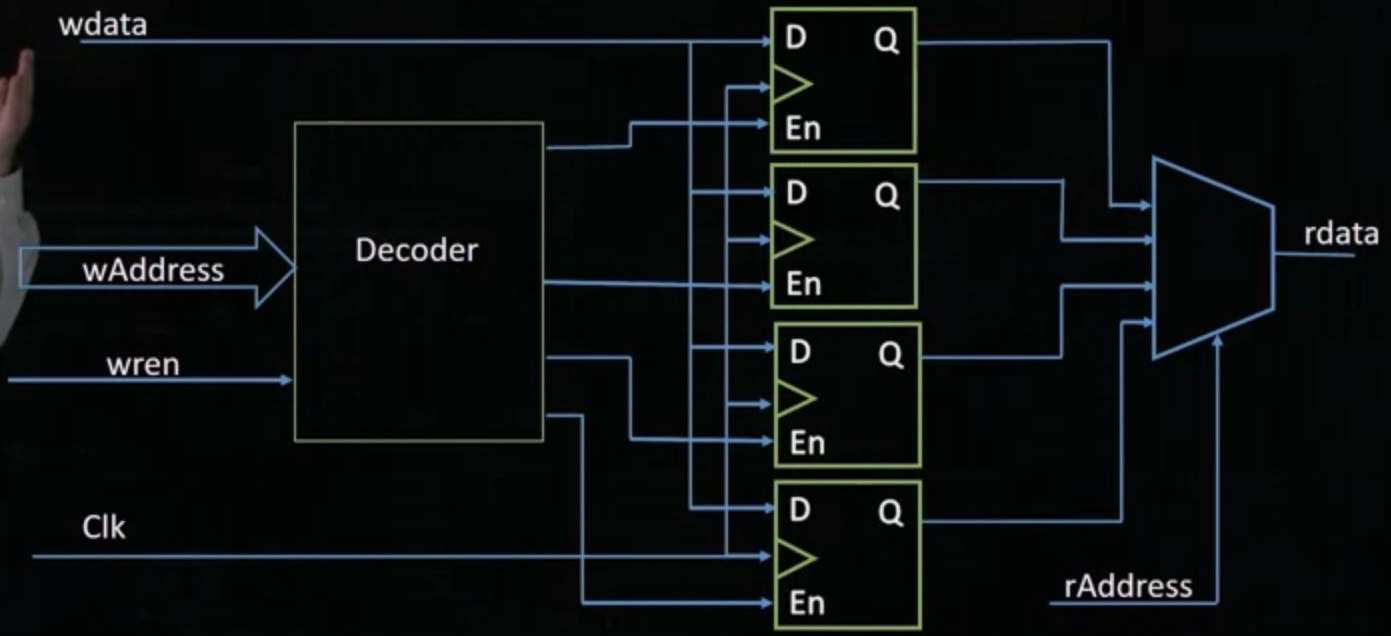
Memories are a common element in

University of Colorado Boulder

Copyright © 2019 University of Colorado

Synchronous Logic : Register File

Register Files are useful constructs that allow addressing of registers.
Here we assume each flip flop represents an n-bit register.

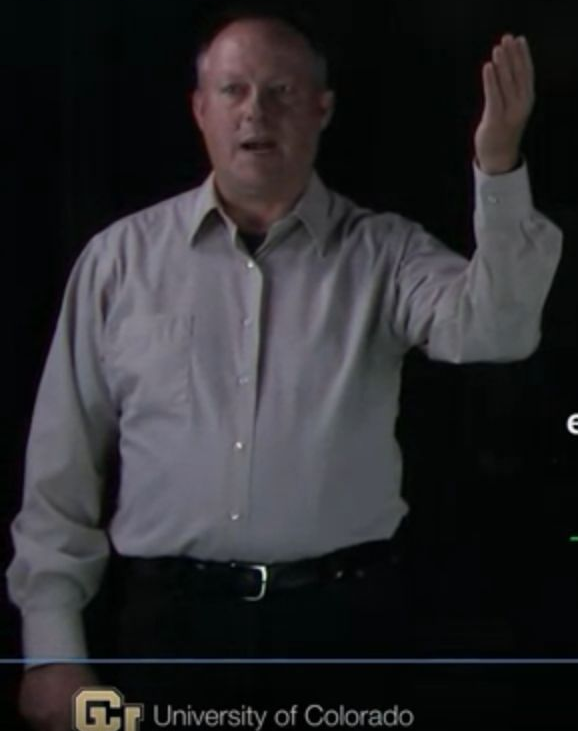


Register files are useful constructs that

University of Colorado Boulder

Copyright © 2019 University of Colorado

Dual Port RAM



```
-- Entity
entity DPRAM is
  generic (D_Width : integer := 8;
           A_Width : integer := 10 ); -- 2**10 = 1024
  port (
   clk, we      : in  std_logic;
   d            : in  std_logic_vector(D_Width-1 downto 0);
   w_add, r_add : in  std_logic_vector(A_Width-1 downto 0);
   q            : out std_logic_vector(D_Width-1 downto 0) );
end entity DPRAM;

-- Architecture, Next Slide
```

Here's our entity
for a dual port RAM.

Share
University of Colorado Boulder
Copyright © 2019 University of Colorado
0:50 / 7:39

Share

Press Esc to exit full screen

Dual Port RAM

```
architecture DPR_Arch of DPRAM is -- Architecture
    type ram_type is array (0 to 2**A_Width-1) of
        std_logic_vector (D_Width-1 downto 0);

impure function read_file(txt_file : in string) return ram_type is
  file ram_file : text open read_mode is txt_file;
  variable  txt_line  : line;
  variable  txt_bit   : bit_vector(D_Width-1 downto 0);
  variable  txt_ram   : ram_type;
  begin for i in ram_type'range loop
    readline(ram_file, txt_line);
    read(txt_line, txt_bit);
    txt_ram(i)  :=  to_stdlogicvector(txt_bit);
  end loop;   return txt_ram;
end function;
```

the power of address with minus

University of Colorado Boulder

https://www.fpga4student.com/2018/08/how-to-read-image-in-vhdl.html

Copyright © 2019 University of Colorado

1:53 / 7:39

Dual Port RAM

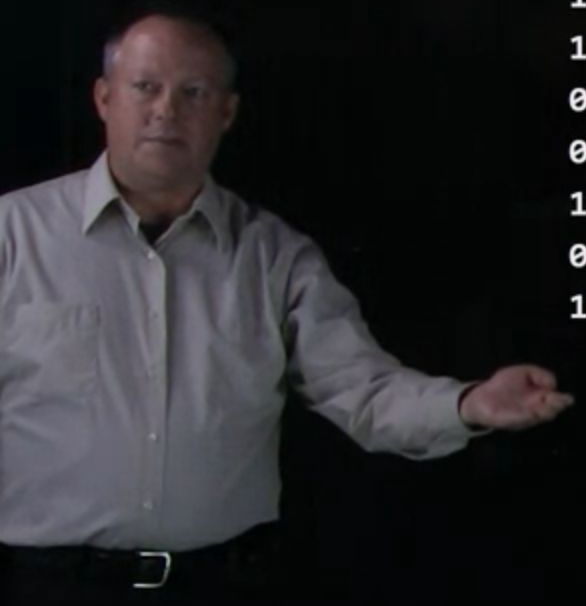
```
-- Architecture
    signal ram : ram_type :=  read_file("initialRAM.txt");
       -- Read the ram text file from the function
    signal data_reg : std_logic_vector (D_Width-1 downto 0);
  begin ram_proc : process (clk)
    begin
     if    (rising_edge(clk))      then
       if (we='1')                 then
          ram(to_integer(unsigned(w_add)))  <= d;
          data_reg <= ram(to_integer(unsigned(r_add))) ;
       end if;
     end if;
     q <= data_reg ;
  end process ram_proc;
end architecture DPR_Arch;
```

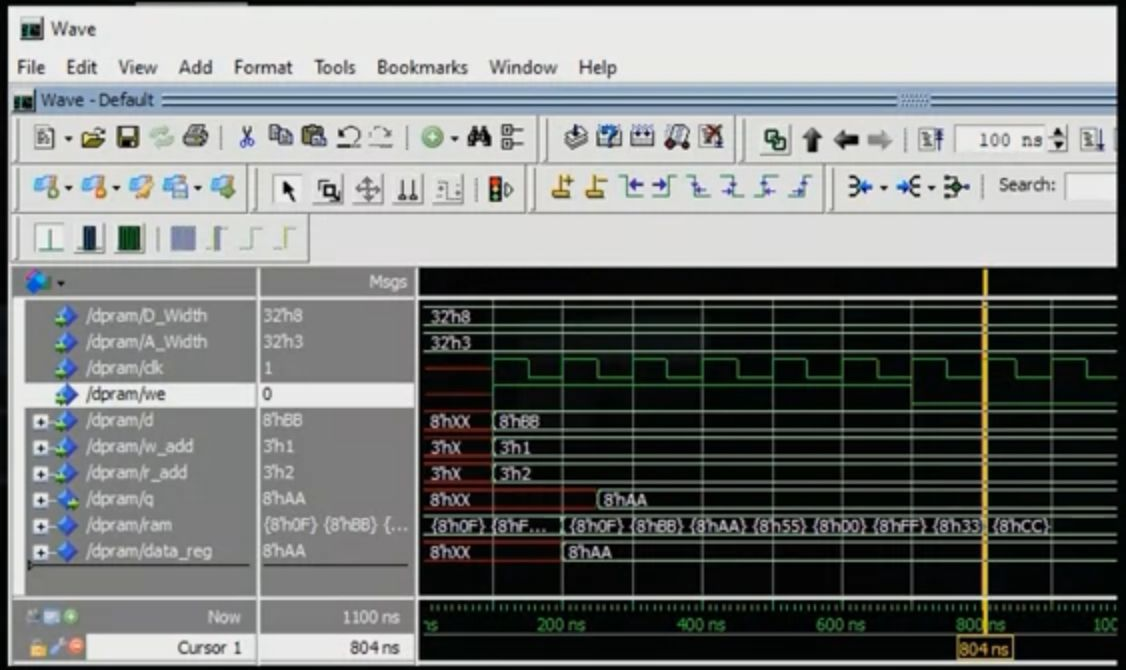
and we're going to set that
equal to our function,

Copyright © 2019 University of Colorado

Dual Port RAM

```
-- Short RAM Text File : address 0 to 7, 8 bits
00001111
11110000
10101010
01010101
00000000
11111111
00110011
11001100
```



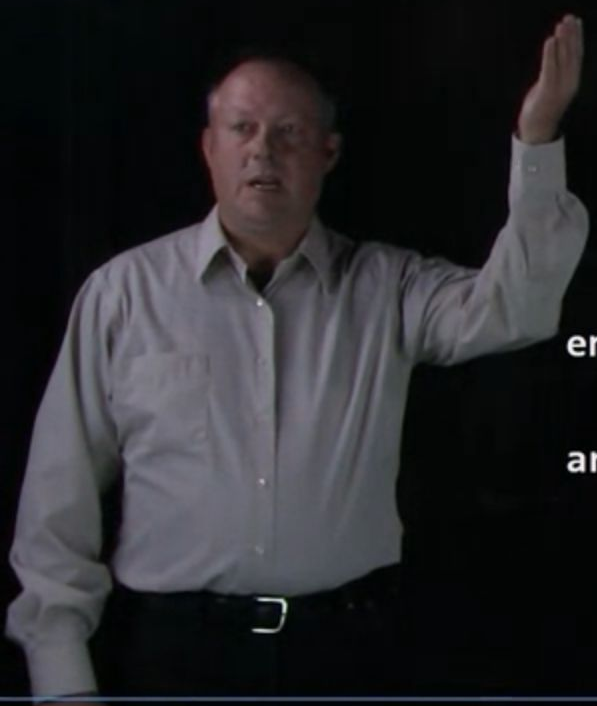


Here's an example with a short text file.

University of Colorado Boulder

Copyright © 2019 University of Colorado

ROM Memory



```
entity ROM is                              -- Entity
  generic (D_Width : integer := 8;
           A_Width : integer := 3 ); -- 2**3 = 8 address
  port (
   clk        : in  std_logic;
   addr       : in  std_logic_vector(A_Width-1 downto 0);
   data       : out std_logic_vector(D_Width-1 downto 0) );
end entity ROM;

architecture ROm_Arch of ROM is -- Architecture
  signal rom_d, data_reg : std_logic_vector
         (D_Width-1 downto 0);
  signal addr_sel : std_logic_vector (2 downto 0);
```

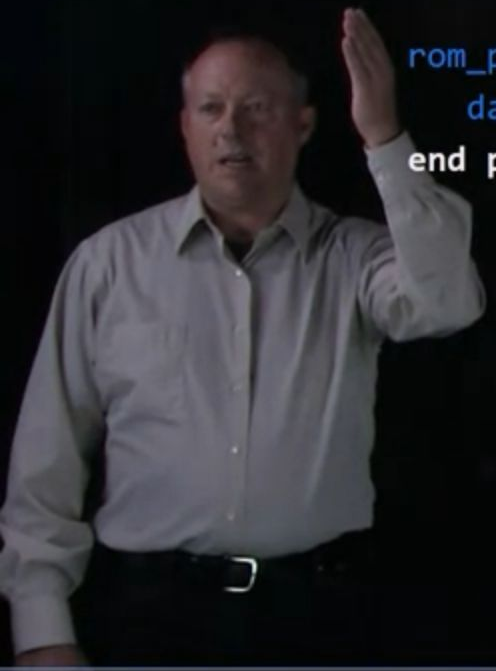
Next, we will look at generating a ROM memory.

University of Colorado Boulder

Copyright © 2019 University of Colorado

ROM Memory

```
begin
    addr_sel <= addr;

    rom_proc : process (clk)  begin
       data_reg <= rom_d;
    end process rom_proc;
```



We're going to begin our architecture,

https://www.fpga4student.com/2

Copyright © 2019 University of Colorado

ROM Memory

```
lookup_proc : process  begin -- Lookup Table
  case(addr_sel) is
    when "000" => rom_d <= "10000000";  when "100" => rom_d <= "00000000";
    when "001" => rom_d <= "10101010";  when "101" => rom_d <= "10011001";
    when "010" => rom_d <= "01010101";  when "110" => rom_d <= "10000001";
    when "011" => rom_d <= "10000011";  when "111" => rom_d <= "11110000";
    when others => rom_d <= "00000000"; -- +700 cases possible, X, U
  end case;
end process lookup_proc;

data <= data_reg;

end architecture ROM;
```

Here we have a lookup process.

https://www.fpga4student.com/2018/08/how-to-read-image-in-vhdl.html

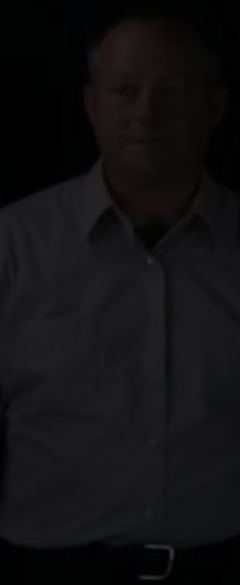
Copyright © 2019 University of Colorado

The when others statement covers the case when rom_d is unknown or "UUUUUUUU"?

- True.
- False.

Correct
Correct, the when others statement covers when addr_sel is "UUU" or any X, U, W, L, H, - condition that is not '1' or '0' combinations.

Continue

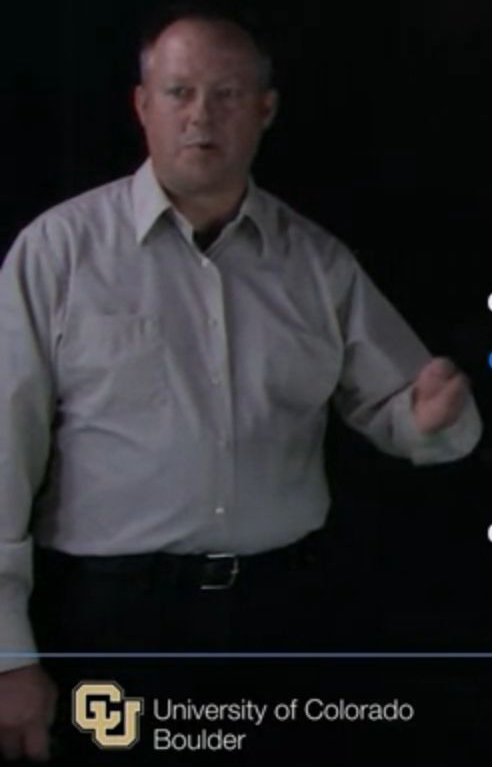


Boulder

end our architecture.

Copyright © 2019 University of Colorado

Binary Encoding



```
--   ... Insert States An90 to An315 here
when An315 =>  --   Last State, others states
   if    (MoveCW  = '1') then NextState <= An0;
   elsif (MoveCCW = '1') then NextState <= An270;
   else                       NextState <= An315;
when others =>                NextState <= An0;
end case;
end process comb_proc;
clk_proc : process (clk, reset)  begin
  if (reset = '1') then CurrentState <= PhyPosition;
  elsif (rising_edge(clk)) then CurrentState <= NextState;
end process clk_proc;
```

a clock process to take us through the state.

University of Colorado Boulder

Copyright © 2019 University of Colorado

ROM Memory : Wave Addr=h3

our data values at
x Address three.

Share
University of Colorado Boulder
Copyright © 2019 University of Colorado

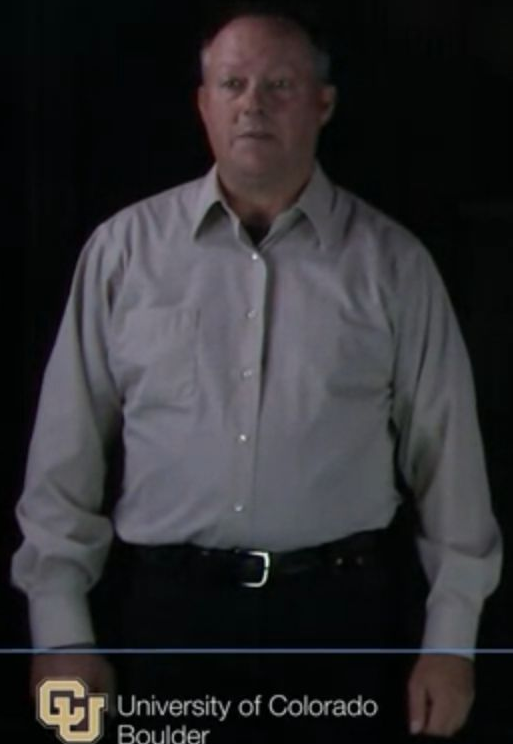
ROM Memory

```
lookup_proc : process  begin -- Lookup Table
  case(addr_sel) is
    when "000" => rom_d <= "10000000";  when "100" => rom_d <= "00000000";
    when "001" => rom_d <= "10101010";  when "101" => rom_d <= "10011001";
    when "010" => rom_d <= "01010101";  when "110" => rom_d <= "10000001";
    when "011" => rom_d <= "10000011";  when "111" => rom_d <= "11110000";
    when others => rom_d <= "00000000"; -- +700 cases possible, X, U
  end case;
end process lookup_proc;

data <= data_reg;

end architecture ROM;
```

When we look back at
our address 0-1-1,

Share
University of Colorado Boulder
https://www.fpga4student.com
Copyright © 2019 University of Colorado
7:17 / 7:39

Summary – RAM and ROM Memory in VHDL



In this video, you have learned:

- How to create RAM and ROM memory devices
- How to initialize a memory using an external text file as input
- How to create a Look Up Table

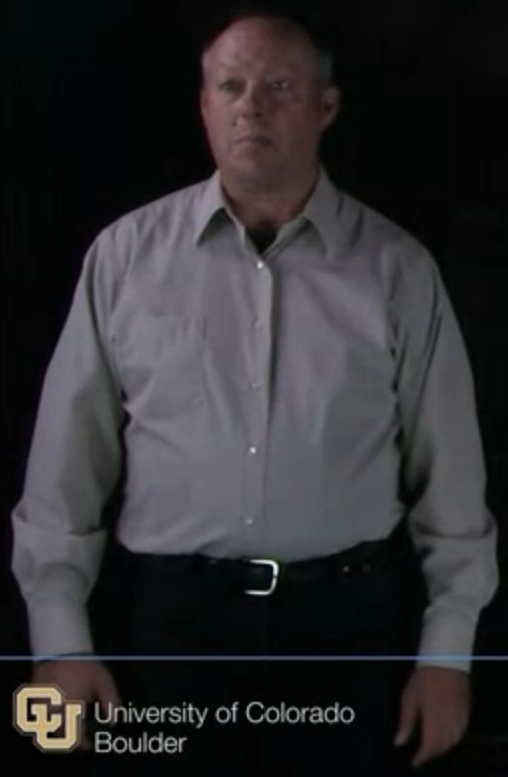
a RAM and ROM memory device,

University of Colorado Boulder

Copyright © 2019 University of Colorado


Share
Press Esc to exit full screen
FPGA Design for Embedded Systems
Hardware Description Languages for Logic Design
University of Colorado Boulder
Hello, and welcome to FPGA design for embedded systems.
right © 2019 University of Colorado
0:02 / 8:02

VHDL Finite State Machines



In this video you will learn :

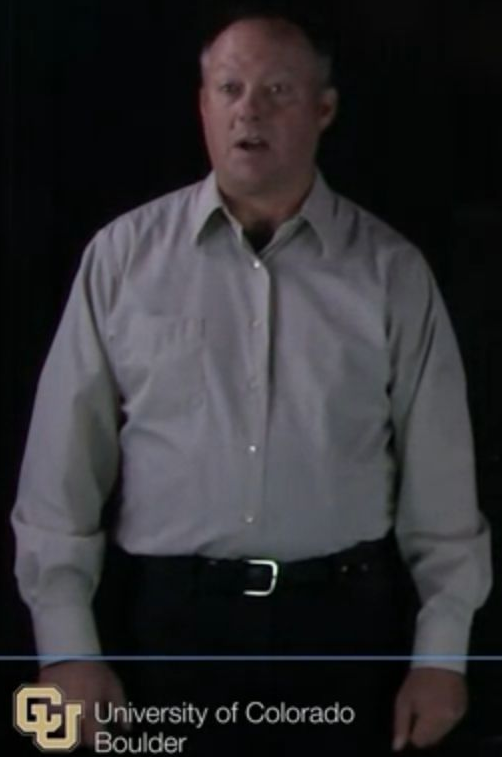
- The rationale for using Finite State Machines
- How to create Finite State Machines
- Criteria for determining which state encoding formats to use

Hello, and welcome to FPGA design for embedded systems.

University of Colorado Boulder

Copyright © 2019 University of Colorado

Finite State Machines



Finite State Machines (FSM) are a very important part of digital design

(and software design, too) !

The state machine concept provides a highly reliable, maintainable, and methodical way to design circuits that perform a sequence of operations with great predictability.

State machines are always in a known state. Good designs make use of FSMs.

University of Colorado Boulder

digital design and
software design too.

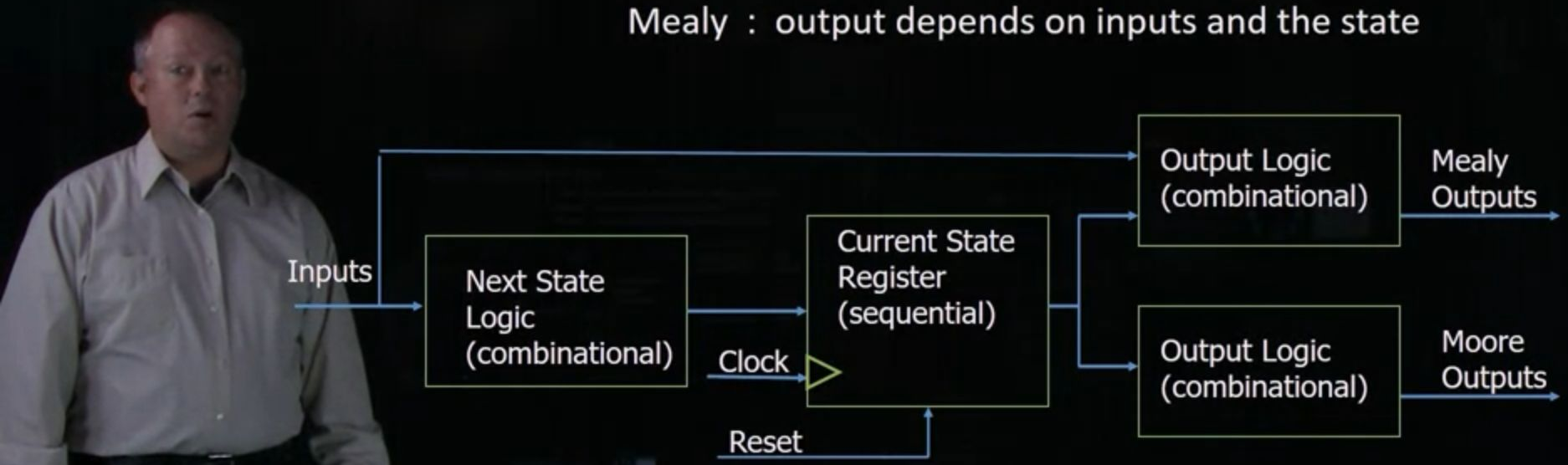
Copyright © 2019 University of Colorado

Finite State Machines

FSMs are categorized into 2 types : Moore or Mealy

Moore : output depends on the state

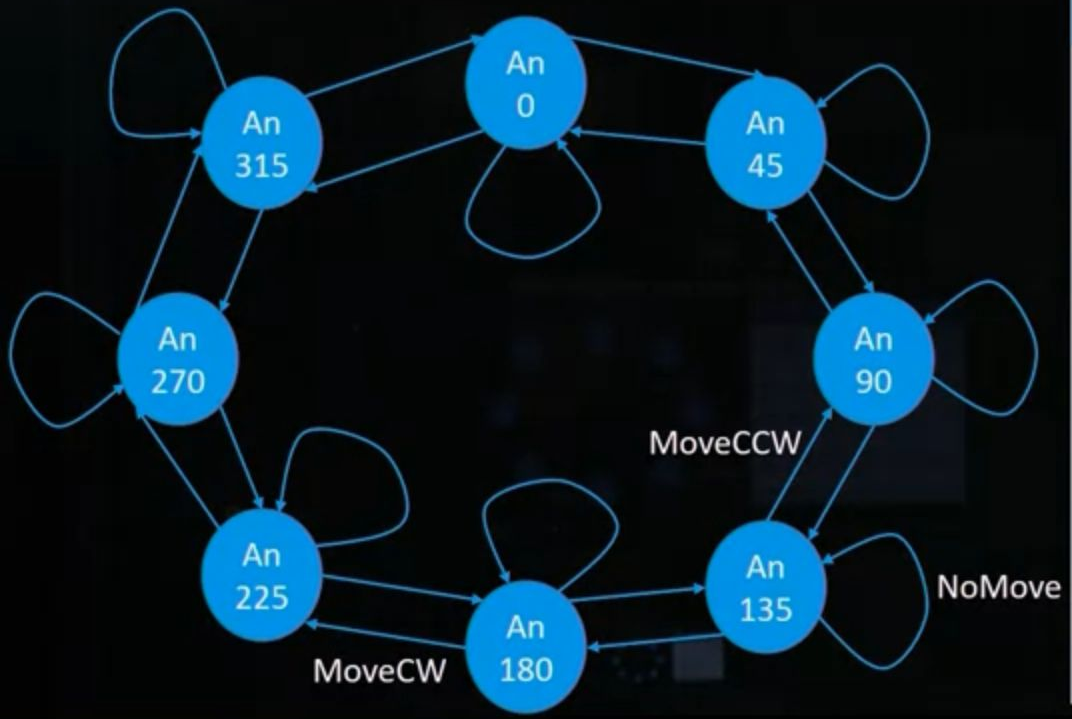
Mealy : output depends on inputs and the state



two types; moore or mealy.

Copyright © 2019 University of Colorado

State Diagram and State Table



Current State	Next State	Next State	Next State	Output	Output
	Input = Move CW	Input = Move CCW	Input = NoMove	Desired Position	PosError
An0	An45	An315	An0	Current State	DesPos - PhyPos
An45	An90	An0	An45	..	..
An90	An135	An45	An90	..	..
An135	An180	An90	An135	..	..
An180	An225	An135	An180	..	..
An225	An270	An180	An225	..	..
An270	An315	An225	An270	..	..
An315	An0	An270	An315	..	..

the angles as we proceed through

Copyright © 2019 University of Colorado

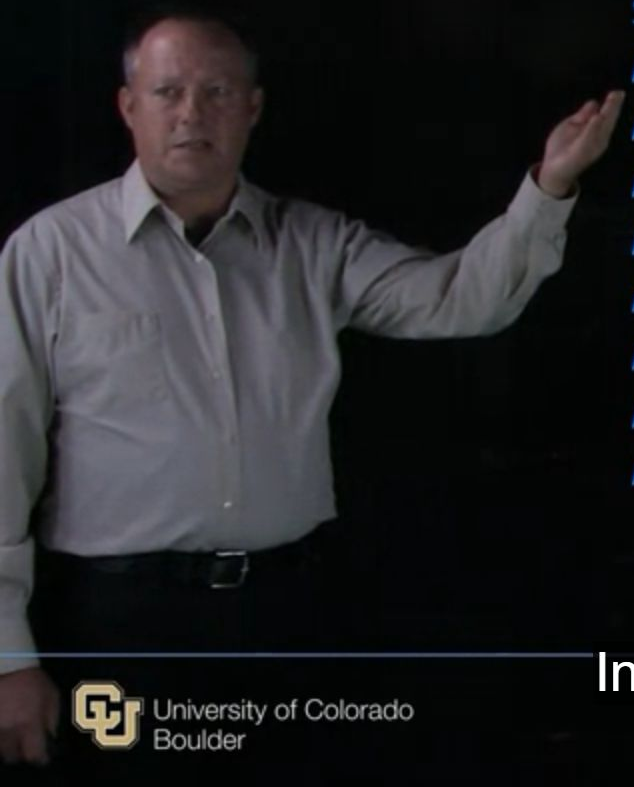
State Encoding Types

No.	Binary	Gray	Johnson	One-Hot
0	000	000	0000	00000001
1	001	001	0001	00000010
2	010	011	0011	00000100
3	011	010	0111	00001000
4	100	110	1111	00010000
5	101	111	1110	00100000
6	110	101	1100	01000000
7	111	100	1000	10000000

binary, Gray,
Johnson, and one-hot.

Copyright © 2019 University of Colorado

Binary Encoding

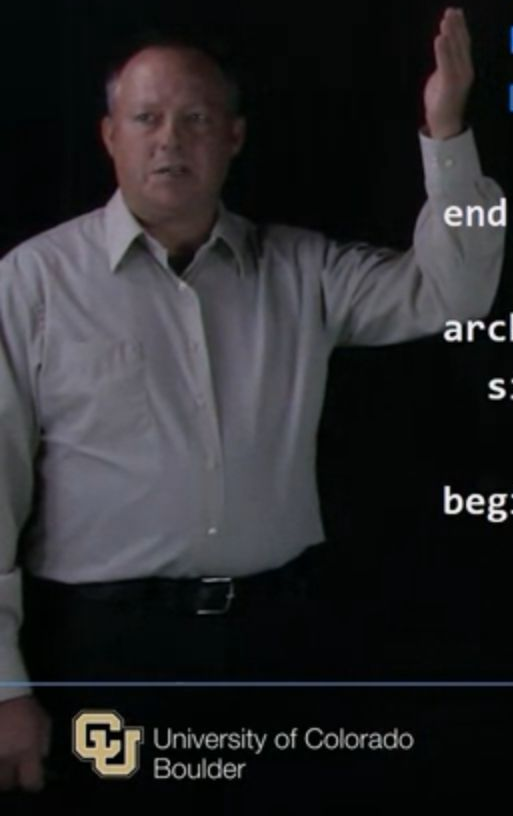


```
entity AngleFSM is                    -- Entity
  generic (
   S_Width : integer := 3;    -- State Width
   An0   : std_logic_vector(3 downto 0) := "000";
   An45  : std_logic_vector(3 downto 0) := "001";
   An90  : std_logic_vector(3 downto 0) := "010";
   An135 : std_logic_vector(3 downto 0) := "011";
   An180 : std_logic_vector(3 downto 0) := “100";
   An225 : std_logic_vector(3 downto 0) := “101";
   An270 : std_logic_vector(3 downto 0) := “110";
   An315 : std_logic_vector(3 downto 0) := “111"   );
```

In VHDL we can create a table for our numerated design.

Copyright © 2019 University of Colorado

Binary Encoding



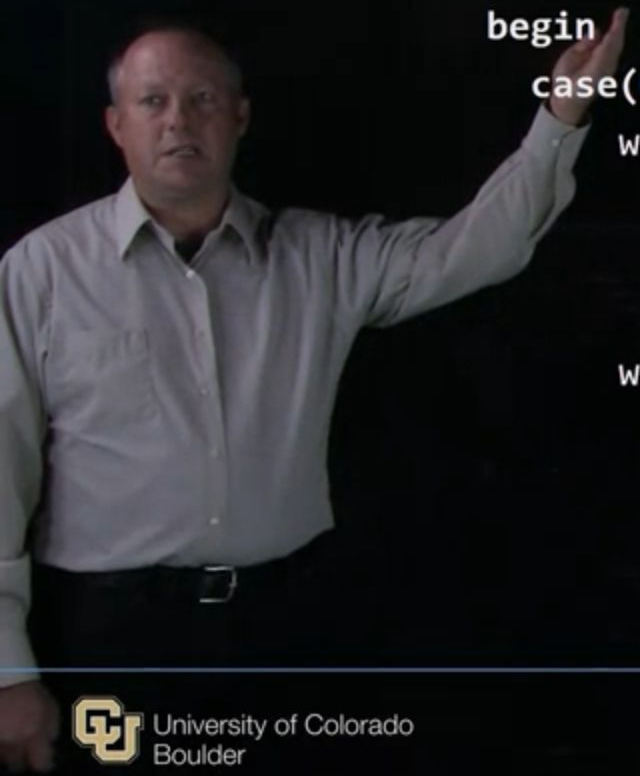
```
    port (
     clk, reset , MoveCw, MoveCCW      : in  std_logic;
     PhyPosition   : in  std_logic_vector(S_Width-1 downto 0);
     DesPosition, PosError   : out
                          std_logic_vector(S_Width-1 downto 0));
end entity AngleFSM;

architecture FSM_Arch of AngleFSM is -- Architecture
  signal CurrentState, NextState :
                          std_logic_vector (S_Width-1 downto 0);
begin
```

Here, we have our port declaration,

University of Colorado Boulder

Copyright © 2019 University of Colorado

Binary Encoding

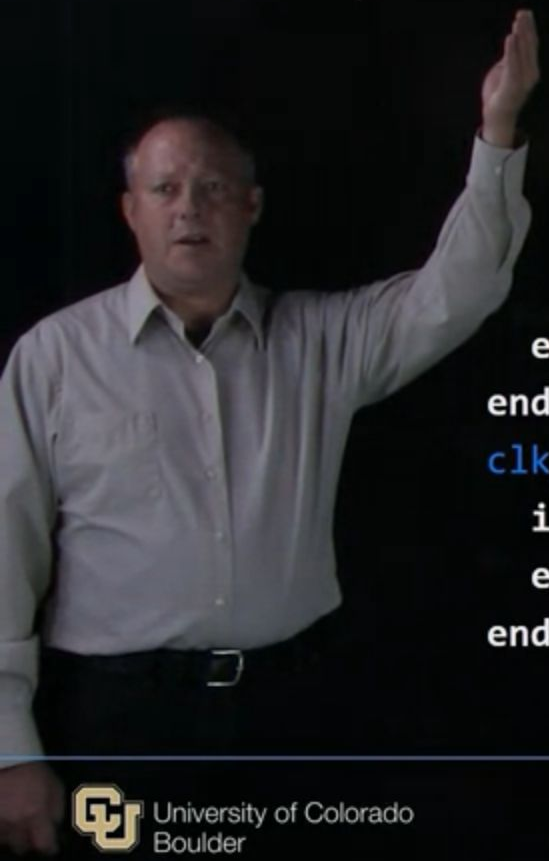


```
comb_proc : process (MoveCw, MoveCCW, PhyPosition,
                     CurrentState)
begin
  case(CurrentState) is
    when An0 =>
      if    (MoveCW  = '1') then NextState <= An45;
      elsif (MoveCCW = '1') then NextState <= An315;
      else                       NextState <= An0;
    when An45 =>
      if    (MoveCW  = '1') then NextState <= An90;
      elsif (MoveCCW = '1') then NextState <= An0;
      else                       NextState <= An45;
```

we're looking at our sensitivity list

Copyright © 2019 University of Colorado

Binary Encoding



```
--   ... Insert States An90 to An315 here
when An315 =>  --   Last State, others states
   if    (MoveCW  = '1') then NextState <= An0;
   elsif (MoveCCW = '1') then NextState <= An270;
   else                       NextState <= An315;
when others =>                NextState <= An0;
end case;
end process comb_proc;
clk_proc : process (clk, reset)  begin
  if (reset = '1') then CurrentState <= PhyPosition;
  elsif (rising_edge(clk)) then CurrentState <= NextState;
end process clk_proc;
```

angle 90 through angle 315,

Copyright © 2019 University of Colorado

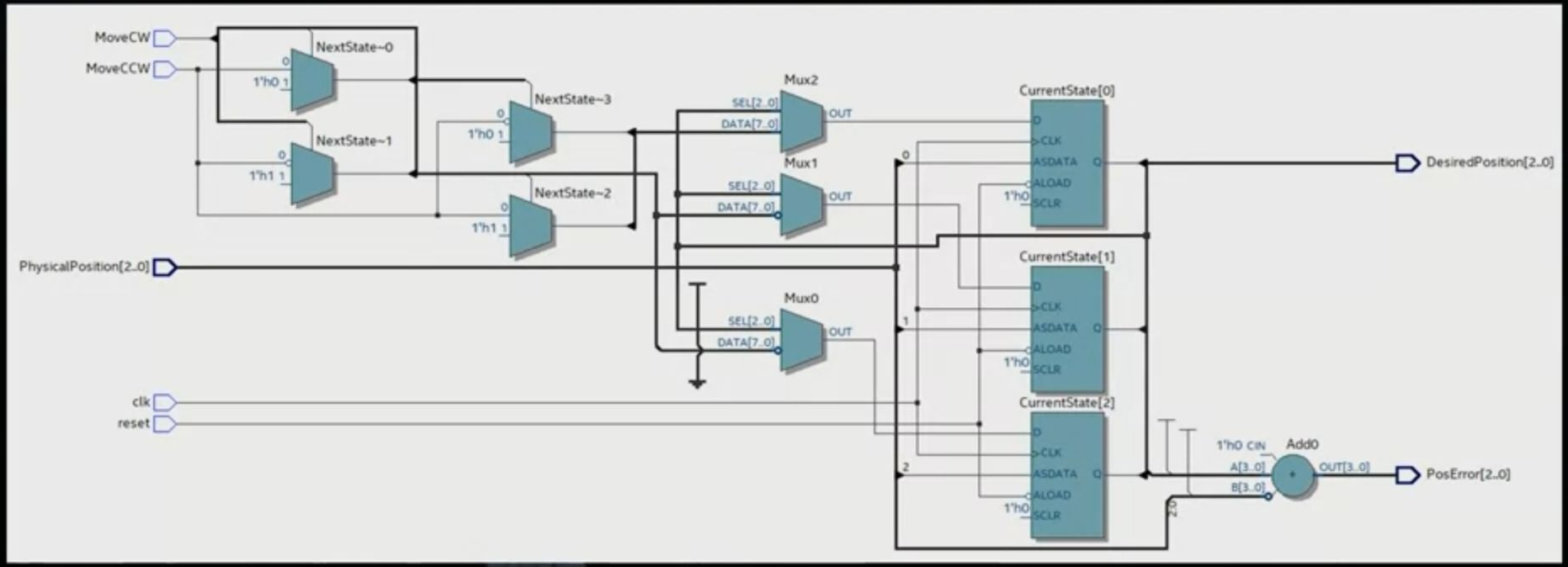
Binary Encoding

```
-- Output Logic
-- Moore Output
DesPosition <= CurrentState;
-- Mealy Output
PosError <= DesPosition - PhyPosition;

end architecture FSM_Arch;
```

we can set our moore output.

Share
University of Colorado Boulder
Copyright © 2019 University of Colorado
6:31 / 8:02

Binary Encoding - Schematic

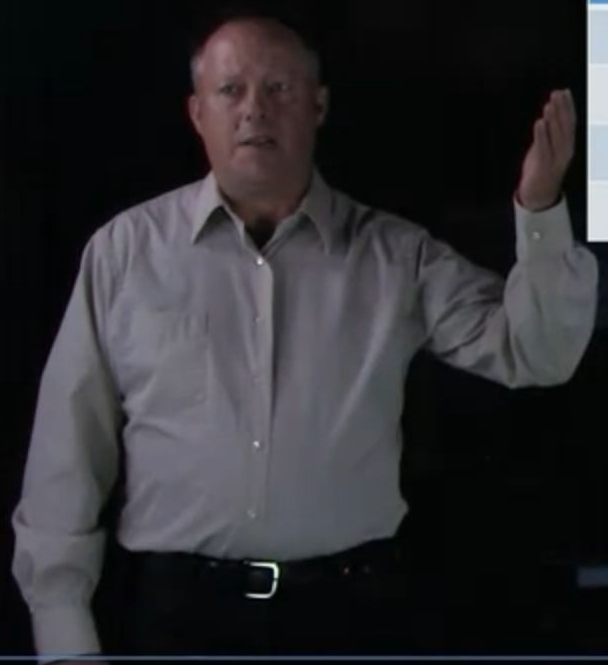


implementation in a MAX 10 FPGA.

Copyright © 2019 University of Colorado

State Encoding - Compare Cell Usage

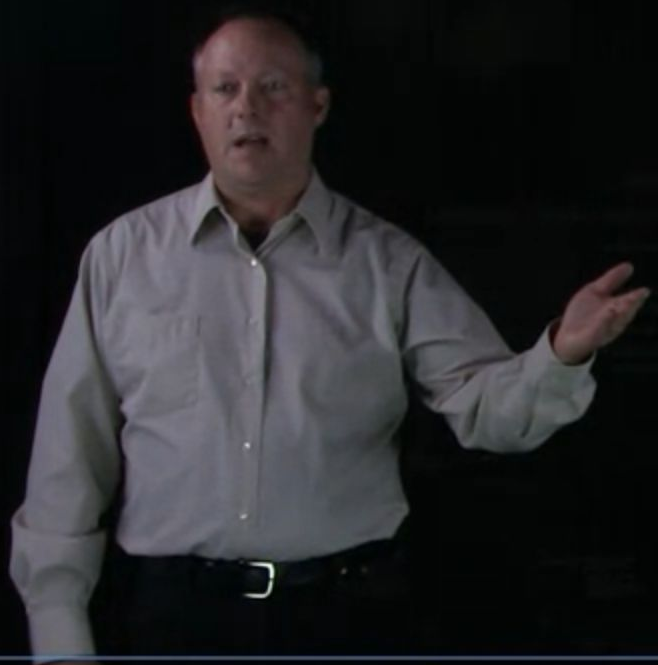
Encoding	Registers (FFs)	Total Logic Cells FSM	Output Encoding Conversion Logic	Total Logic Cells
Binary	3	17	0	17
Gray	3	20	9	29
Johnson	4	36	9	51
One-Hot	8	107	33	146



we find that when
we encode binary,

Copyright © 2019 University of Colorado

Summary – Finite State Machines in VHDL



In this video, you have learned:

- The rationale for use of Finite State Machines
- How to create Finite State Machines
- Some criteria for determining which state encoding formats to use

for the use of finite state machines,

Copyright © 2019 University of Colorado