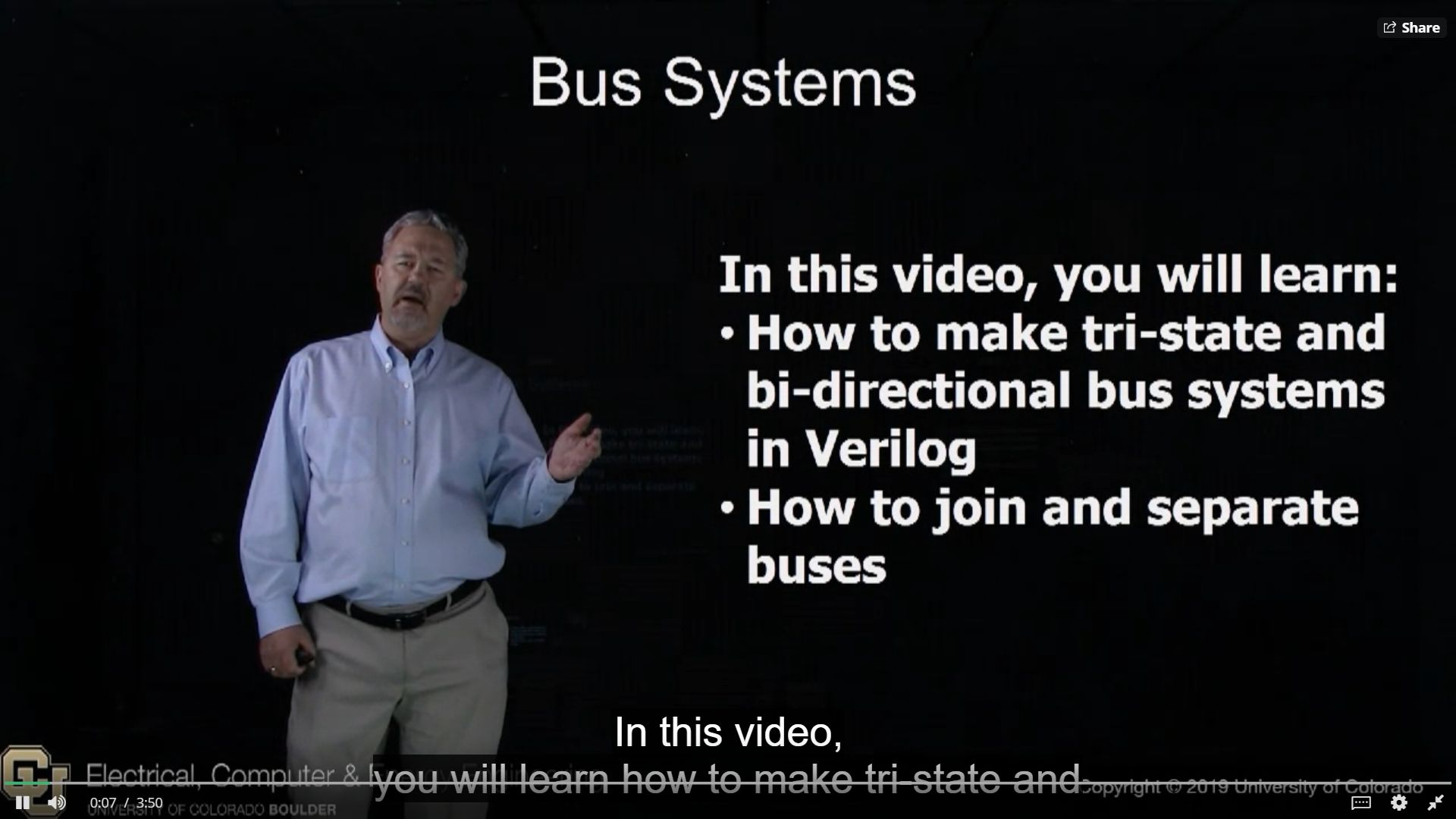
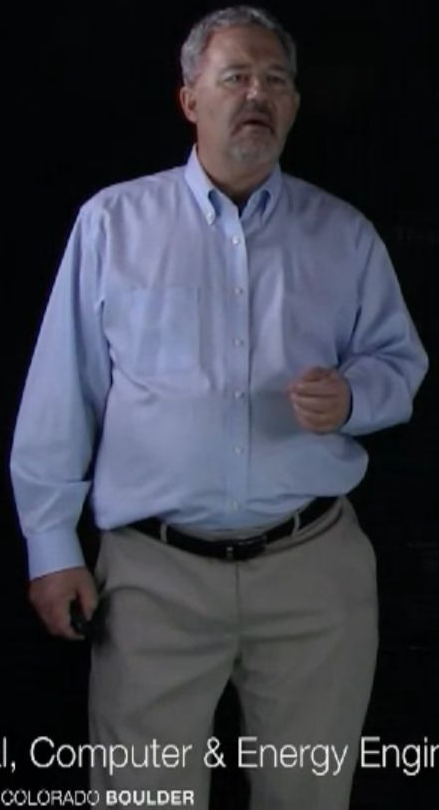
Buses and Tristate Buffers

Professor Tim Scherr

University of Colorado **Boulder**

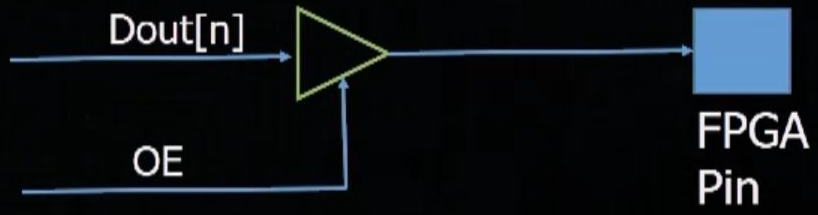

Share
Bus Systems
In this video, you will learn:
• How to make tri-state and bi-directional bus systems in Verilog
• How to join and separate buses
In this video,
you will learn how to make tri-state and
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
0:07 / 3:50
Copyright © 2019 University of Colorado

Tri-state Buses



External connections on FPGA pins are often in a group of related signals known as a bus.

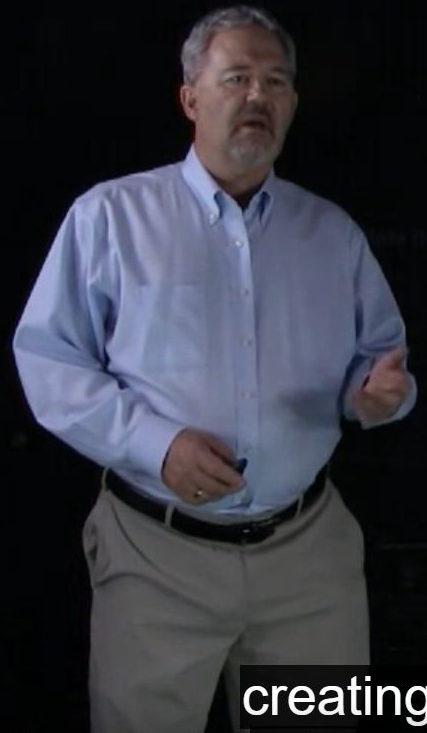
The I/O structures of FGPAs often allow the bus to be tri-stated, so that multiple drivers can be attached to the bus at the same time, with only one active.



Share

Press Esc to exit full screen

Tri-state Buses in Verilog



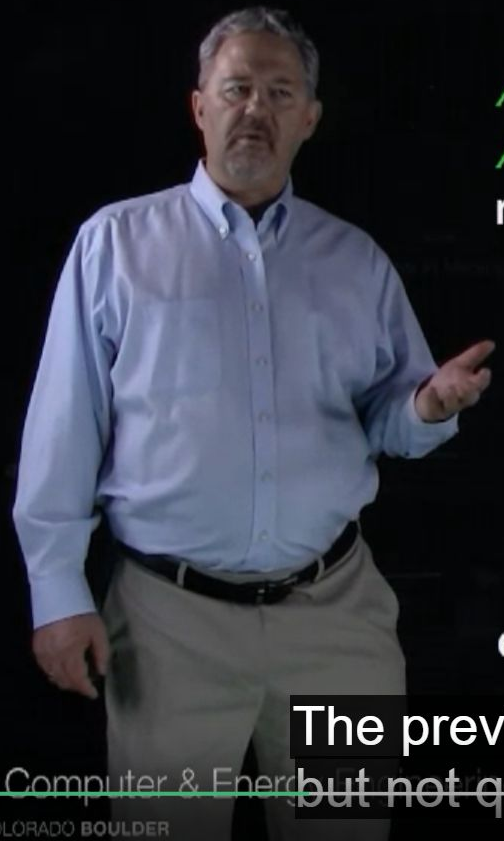
```
// Tri-state bus
//
module Tri (
  input wire [3:0] Dout,
  input wire OE,
  output wire [3:0] Pinout
  );
  assign Pinout = OE ? Dout : 4'bz;
endmodule
```

creating a tri-state buffer is most clear using the conditional statement here.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

0:47 / 3:50

Tri-state Buses in Verilog for Simulation

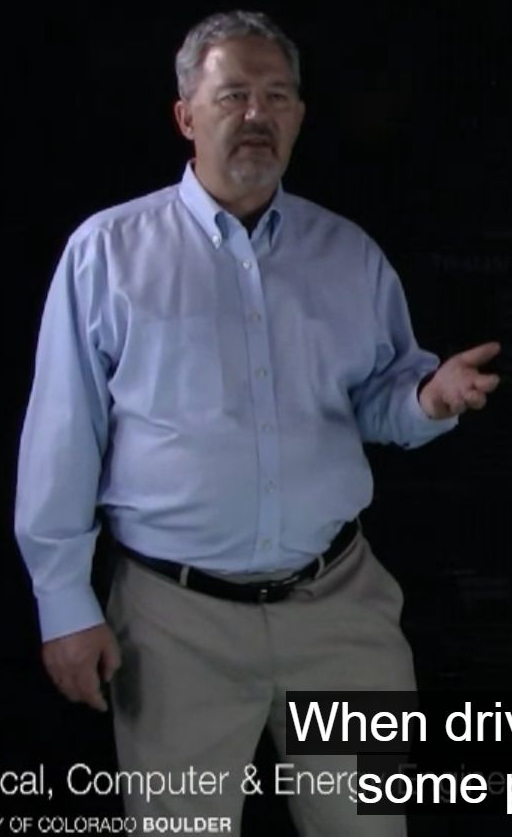


```
// Tri-state bus
//
module Tri (
  input wire [3:0] Dout,
  input wire OE,
  output wire [3:0] Pinout
  );
  assign Pinout = (OE == 1) ? Dout :
                  (OE == 0) ? 4'bz : 4'bx;

endmodule
```

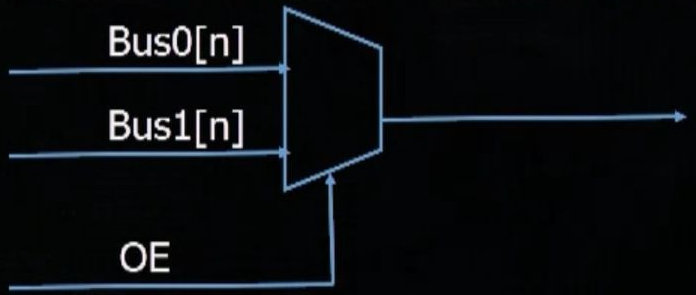
The previous code was compact,
but not quite rigorous enough for

Tri-state Buses



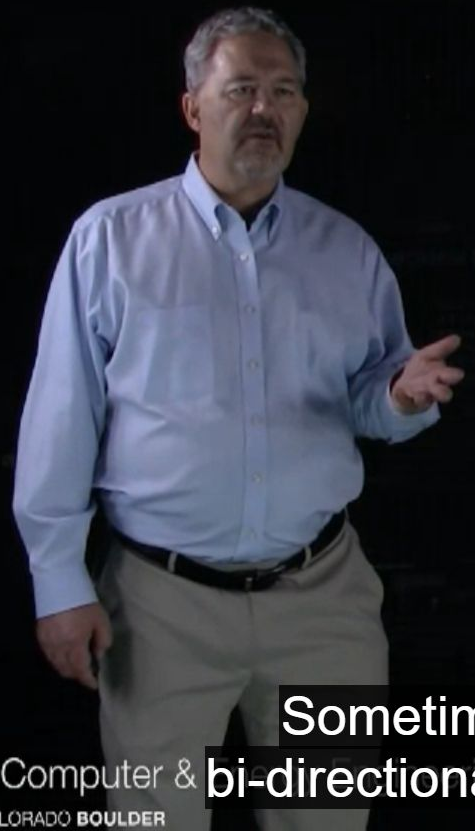
When driving external tri-stated buses, some protocol should be used to assure that only one drive is active on the bus at a time.

Internal Tri-state buses can be defined, but typically are not implemented by FPGA tools as enabled buffers but rather as multiplexers.

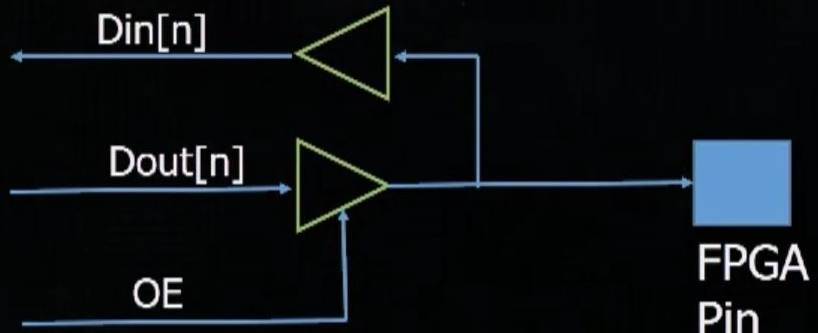


When driving external tri-stated buses, some protocol should be used to

Bi-directional Buses

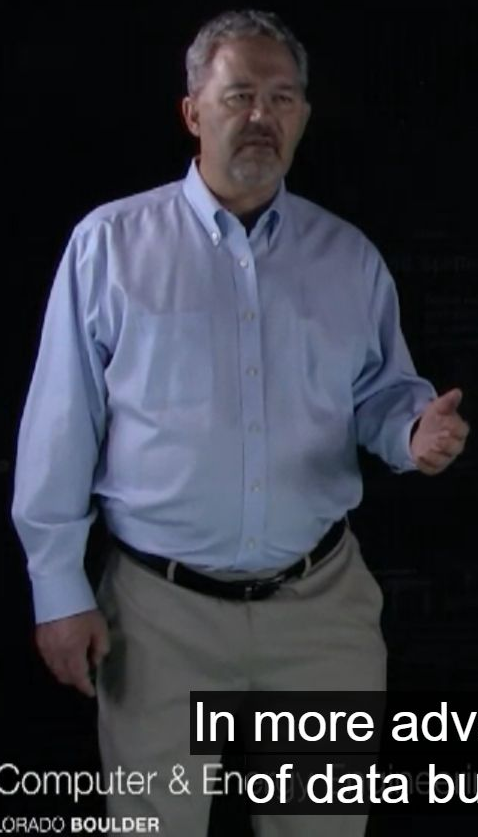


The I/O structure of the FPGA will also allow us to create Bi-directional buses, in which the external pin can be treated as either an input or an output, depending on the state of the enable signal.

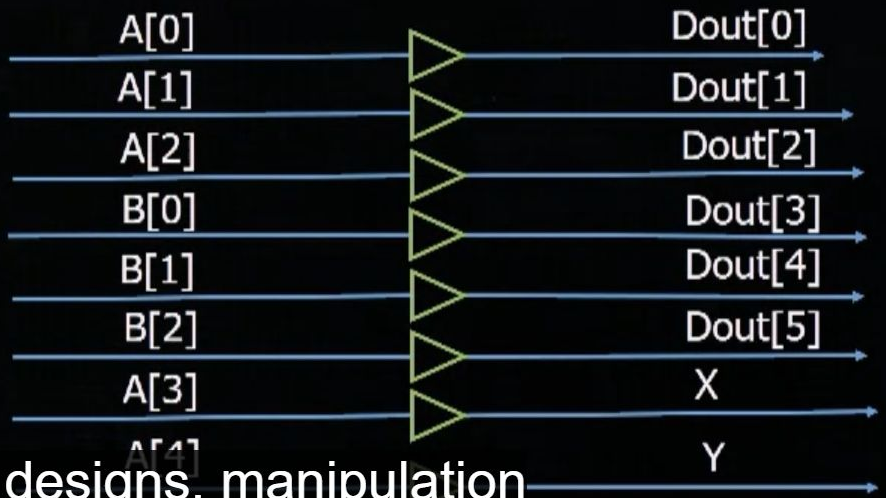


Sometimes the internal buses are bi-directional which allows them to drive

Joining and Splitting Buses

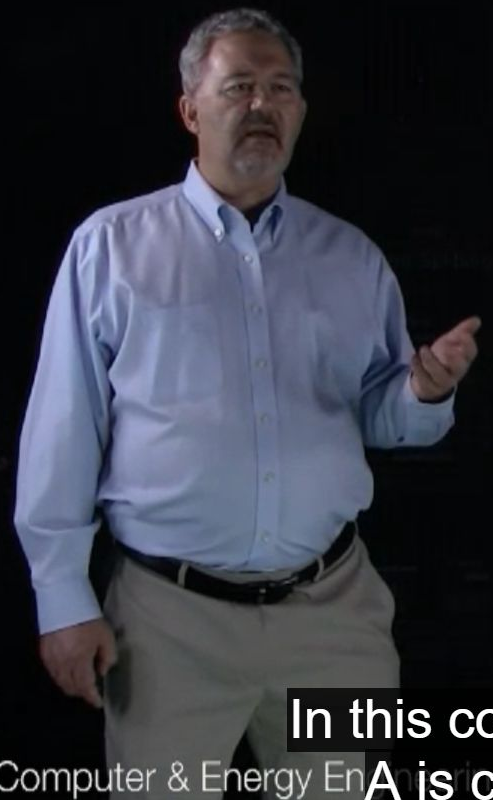


Verilog includes the concatenation and replication operators which allows buses to be combined. Splitting buses can be done using indexing.



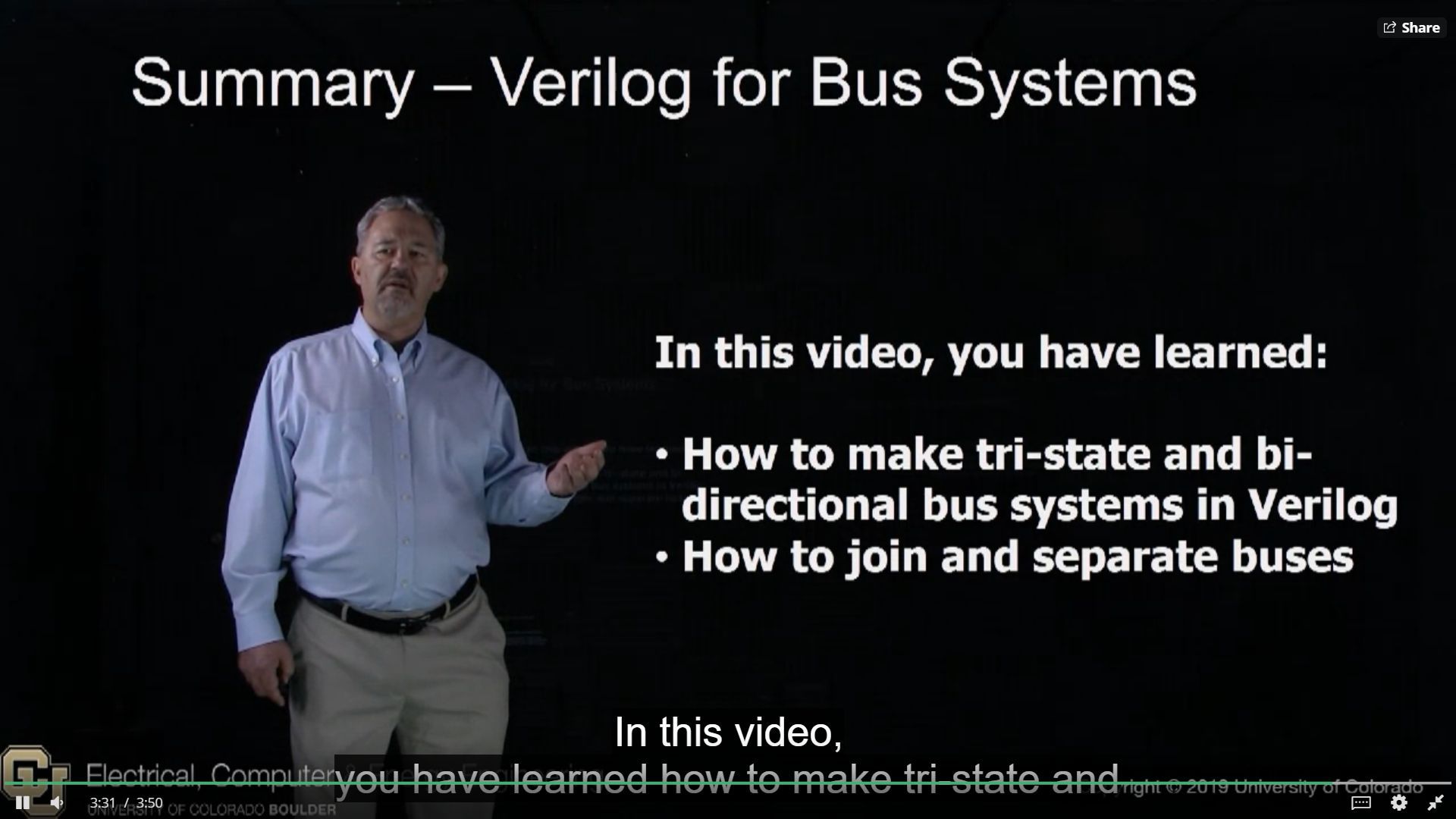
In more advanced designs, manipulation of data buses is very often required.

Joining and Splitting Buses



```
// Manipulating Buses
//
module BusMe (
   input wire [4:0] A,
   input wire [2:0] B,
   output wire X,Y,
   output wire [5:0] Dout
   );
   assign Dout = {B,A[2:0]};
   assign X = A[3];
   assign Y = A[4];
endmodule
```

In this code, we see that the bus A is combined with bus B,


Share
Summary – Verilog for Bus Systems
In this video, you have learned:
• How to make tri-state and bi-directional bus systems in Verilog
• How to join and separate buses
In this video,
you have learned how to make tri-state and
Electrical, Computer
UNIVERSITY OF COLORADO BOULDER
3:31 / 3:50
right © 2019 University of Colorado

References

[1] V. Angelov. (2009). *Introduction to Verilog.* [Online]. Available: https://www.physi.uni-heidelberg.de/~angelov/VHDL/VHDL_SS09_Teil10.pdf

Be Boulder.

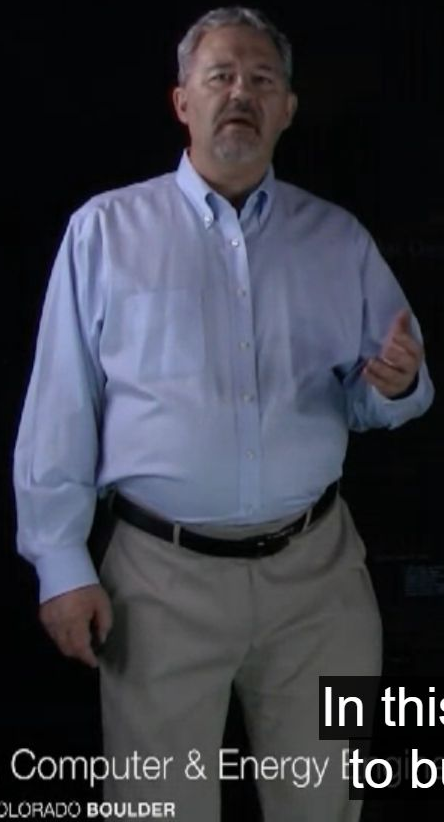
University of Colorado **Boulder**

Modular Design in Verilog

Professor Tim Scherr

University of Colorado **Boulder**

Modular Design in Verilog



In this video, you will learn:

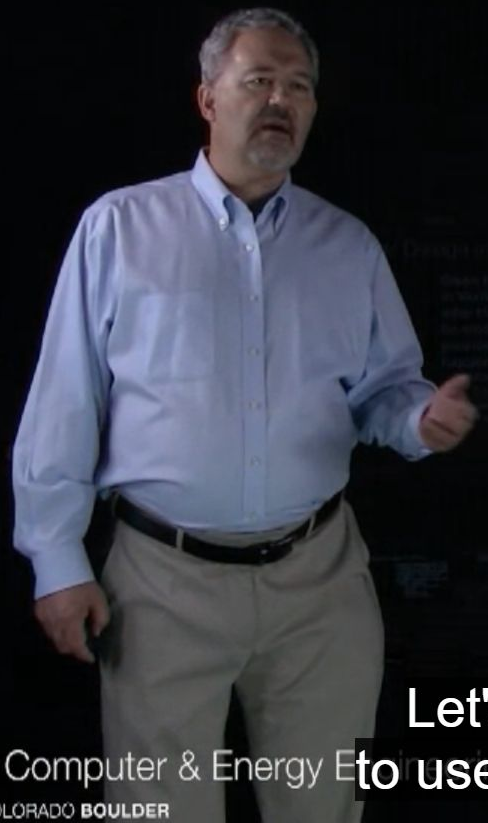
- **How to build bigger designs using modular design techniques**
- **Use of loops in Verilog**
- **How to use for … Generate to make copies of circuits**

In this video, you will learn how to build bigger designs using

Electrical, Computer & Energy E

UNIVERSITY OF COLORADO **BOULDER**

Modular Design in Verilog

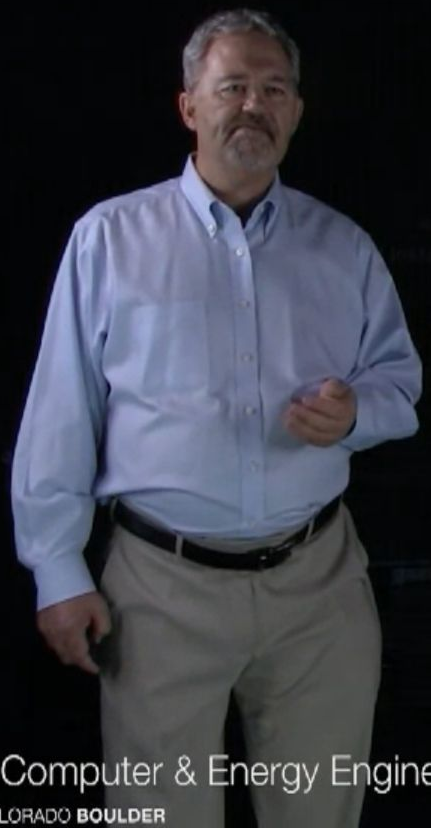


Given the principal design entity in Verilog is a module, one could infer that Verilog designs should be modular. Many tools are provided in Verilog to make this happen, including

- **Component Instantiation**
- **Looping**
- **Generate Blocks**
- **Tasks and Functions**

Let's take a look at how to use some of these today.

Component Instantiation in Verilog



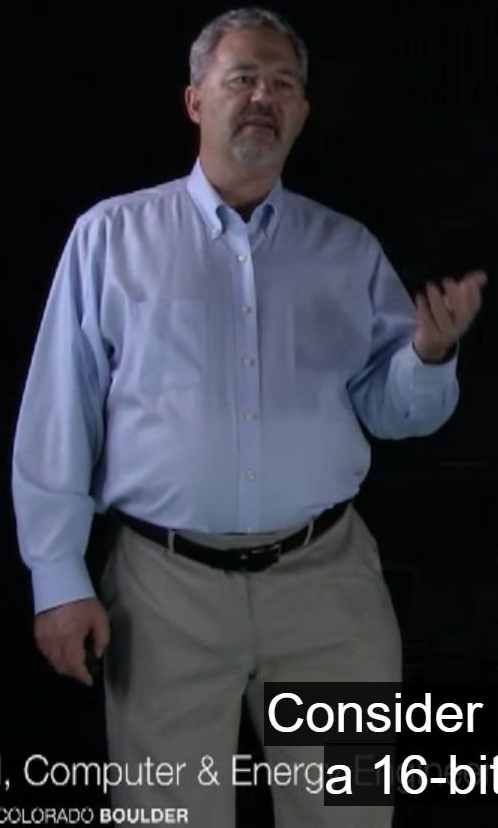
A modular design in Verilog at the top level oftentimes consists only of component instantiations, the fundamental way to build hierarchy in a design.

```
module module_name_top  (port connection
list)
  instance_name_1 (port connection list),
  instance_name_2 (port connection list),

  .......
  instance_name_n (port connection list);

endmodule
```

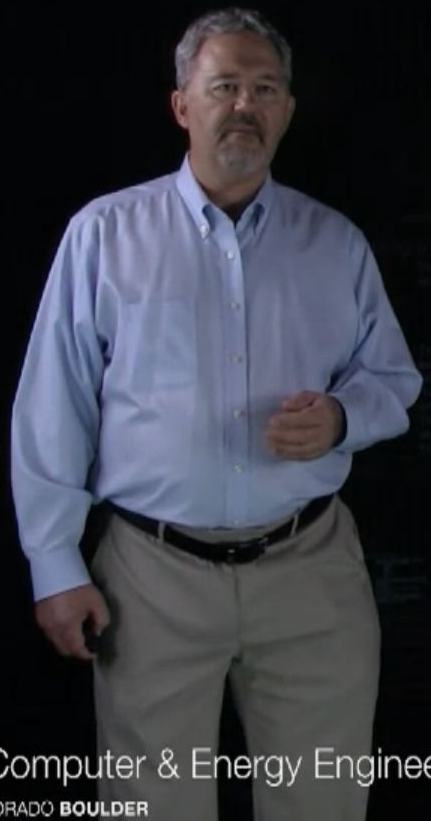
Component Instantiation in Verilog



```
//  16-bit Adder
//   built with 4 instantiations of
//   4-bit Adders
module Add16_top  (
  input [15:0] A,
  input [15:0] B,
  input Cin,
  output Cout,
  output [15:0] Sum
);
  wire Cin2, Cin3, Cin4; //intermediate Cin
  add4 add4_1 (.Data1(A[3:0]), .Data2(B[3:0]),
.Cin(Cin), .Cout(Cin2), .Sum(Sum[3:0]));
  add4 add4_2 (.Data1(A[7:4]), .Data2(B[7:4]),
.Cin(Cin2), .Cout(Cin3), .Sum(Sum[7:4]));
  add4 add4_3 (.Data1(A[11:8]), .Data2(B[11:8]),
.Cin(Cin3), .Cout(Cin4), .Sum(Sum[11:8]));
  add4 add4_4 (.Data1(A[15:12]), .Data2(B[15:12]),
.Cin(Cin4), .Cout(Cout), .Sum(Sum[15:12]));
```

Consider this example which creates a 16-bit adder using nothing but

Looping in Verilog



There are several looping constructs within Verilog, including

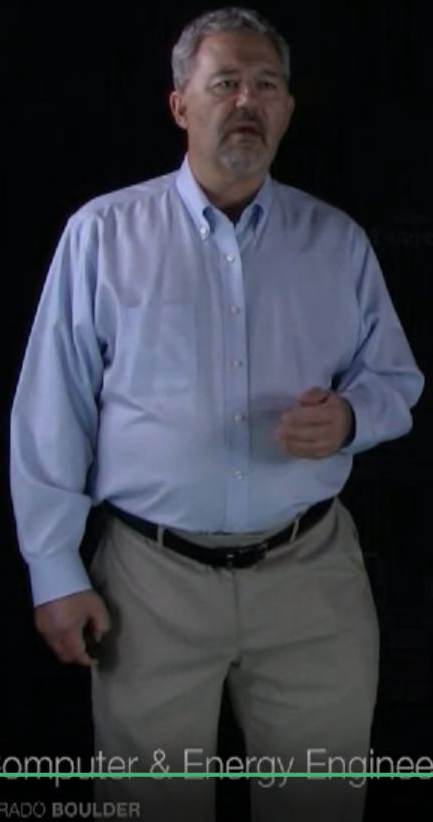
- **repeat**
- **while**
- **forever**
- **for**

The for loop statement is written and behaves just like it does in C, as is the while loop.

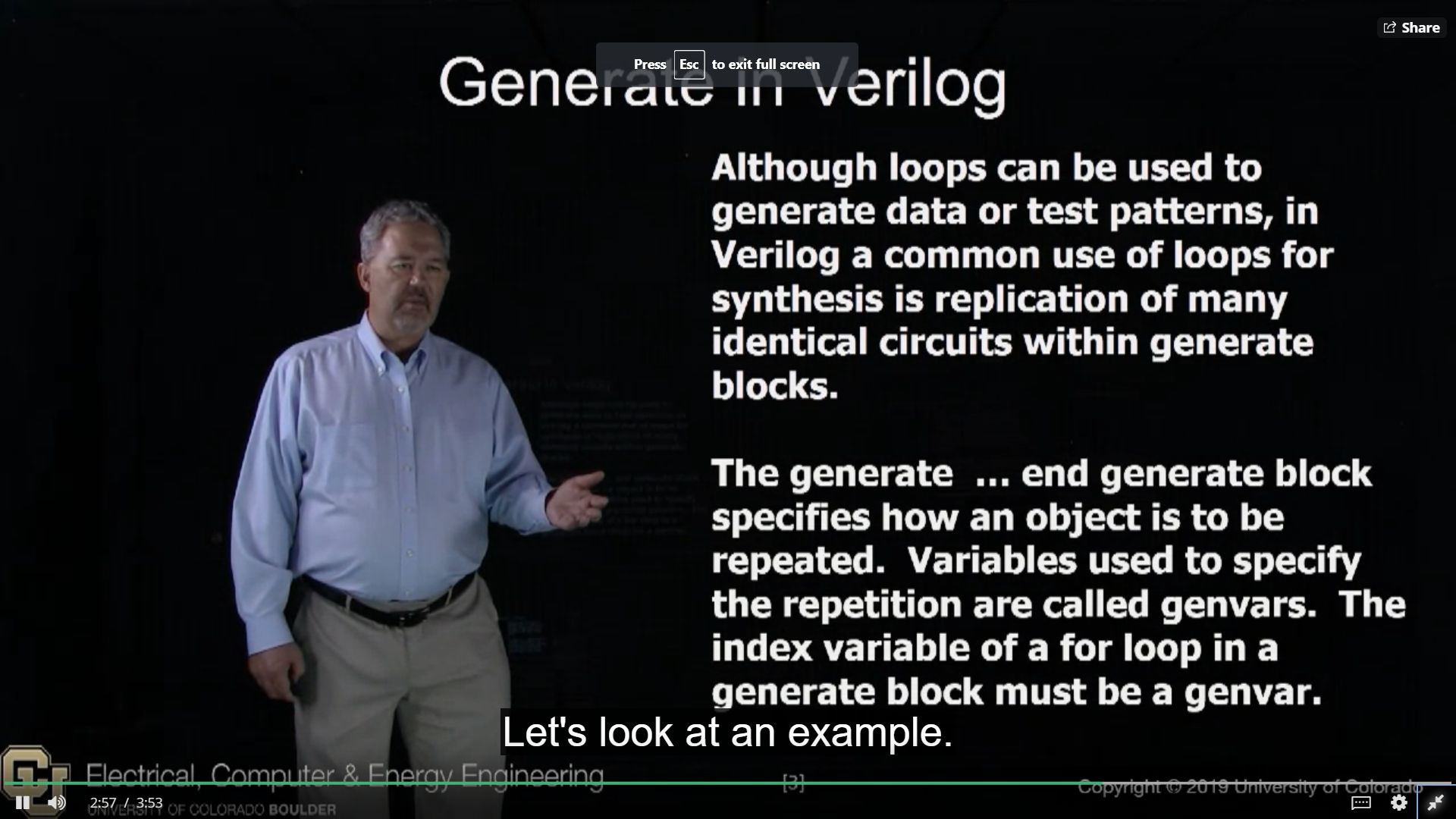
The forever loop is explicitly an infinite loop.

The repeat statement executes a statement or block of statements a fixed number of times.

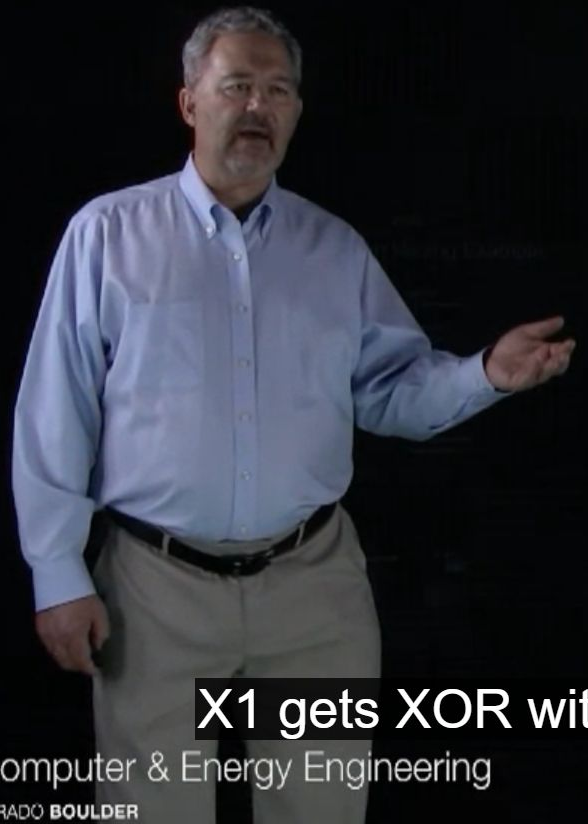
Looping in Verilog Example



```
// And scalar g with vector A
//
Module scalarAnd
 #(parameter N = 4);
  (input g,
   input [N-1:0] a,
   output [N-1:0] y);
  reg [N-1:0] tmp, y;
  integer i;   //loop index, not
signal
  always @(a or g)
  begin
    for(i=0; i<N; i=i+1)
      begin
        tmp[i] = a[i] & g;
      end
    y = tmp;
  end
endmodule
```


Share
Press Esc to exit full screen
Generate in Verilog
Although loops can be used to generate data or test patterns, in Verilog a common use of loops for synthesis is replication of many identical circuits within generate blocks.
The generate ... end generate block specifies how an object is to be repeated. Variables used to specify the repetition are called genvars. The index variable of a for loop in a generate block must be a genvar.
Let's look at an example.
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
[3]
Copyright © 2019 University of Colorado
2:57 / 3:53

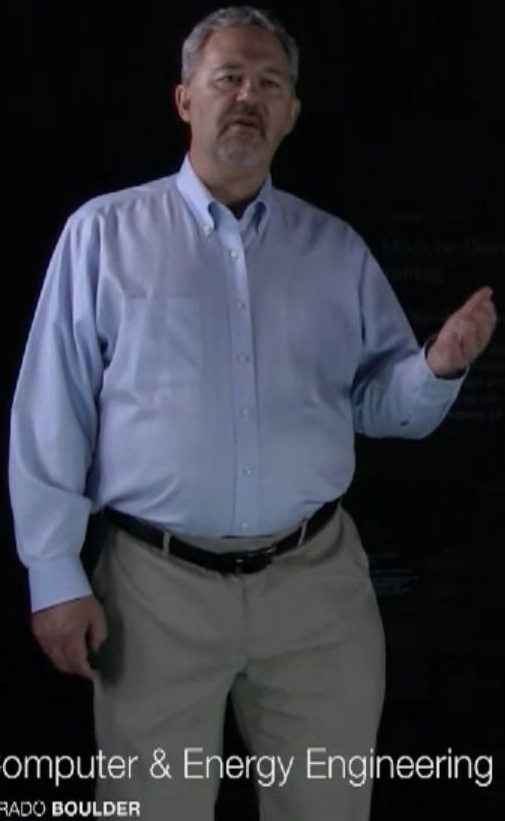
Generate in Verilog Example



```
//  Generate n XOR gates
//
module XorGen
 #(parameter width = 4,
             delay = 10)
  (output [1:width] xout,
   input [1:width] xin1, xin2
  );
  generate
    genvar i;
    for (i=1; i<=width; i=i+1)
    begin
      assign #delay
        xout[i] = xin1[i] ^ xin2[i];
    end
  endgenerate
endmodule
```

X1 gets XOR with X2 producing X out.

Summary – Modular Design in Verilog



In this video, you have learned:

- **How to build bigger designs using modular design techniques**
- **Use of loops in Verilog**
- **How to use for ... Generate to make copies of circuits**

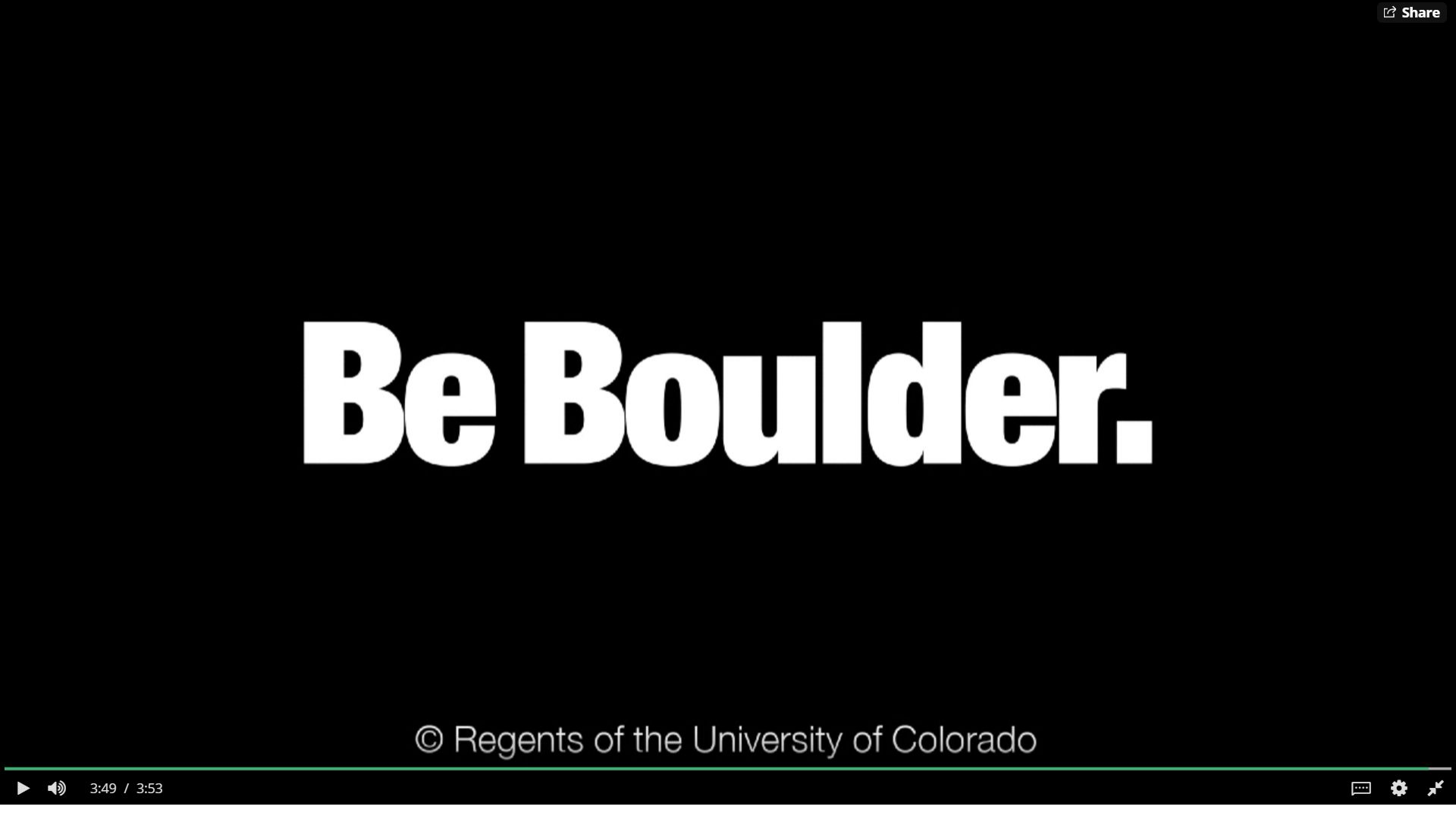
In this video,

References

[1] P. Nyasulu and J. Knight, (2003/Oct/01), *Introduction to Verilog* pdf. [Online]. Available: https://b-ok.org/ireader/3040760

[2] V. Angelov. (2009). *Introduction to Verilog.* [Online]. Available: https://www.physi.uni-heidelberg.de/~angelov/VHDL/VHDL_SS09_Teil10.pdf

[3] D. Thomas and P. Moorby, "Module Hierarchy", in The Verilog Hardware Description Language, 5th ed. Norwell, MA: Kluwer Academic Pub., 2002, ch. 5, pp. 143-153


Share
Be Boulder.
© Regents of the University of Colorado
3:49 / 3:53

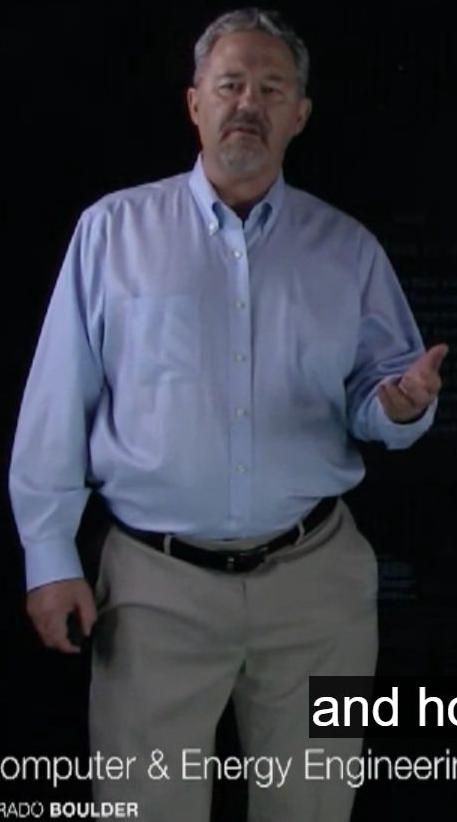
University of Colorado **Boulder**

Testbenches in Verilog

Professor Tim Scherr

University of Colorado **Boulder**

Testbenches in Verilog I

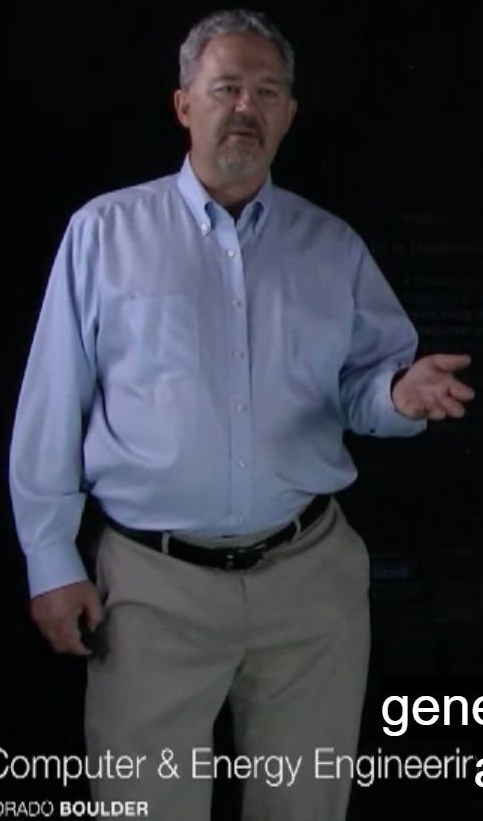


In this video, you will learn:

- **The concept of using a Verilog program, called a testbench, to test another Verilog program (your code)**
- **How to write simple testbenches**
- **How to use loops to generate stimulus**
- **How to use assertions to determine and report test results**

and how to use assertions to

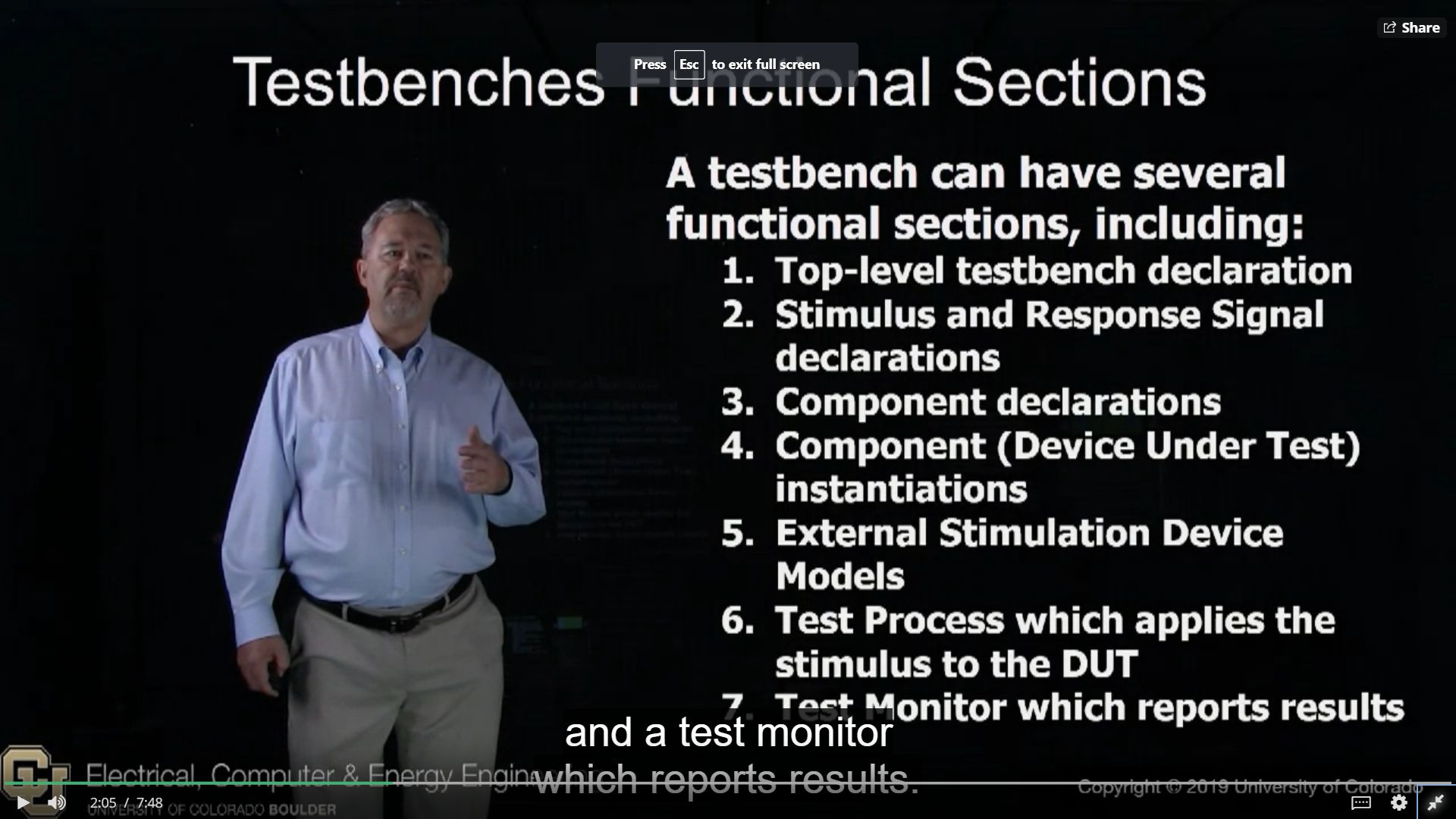
What is a testbench?



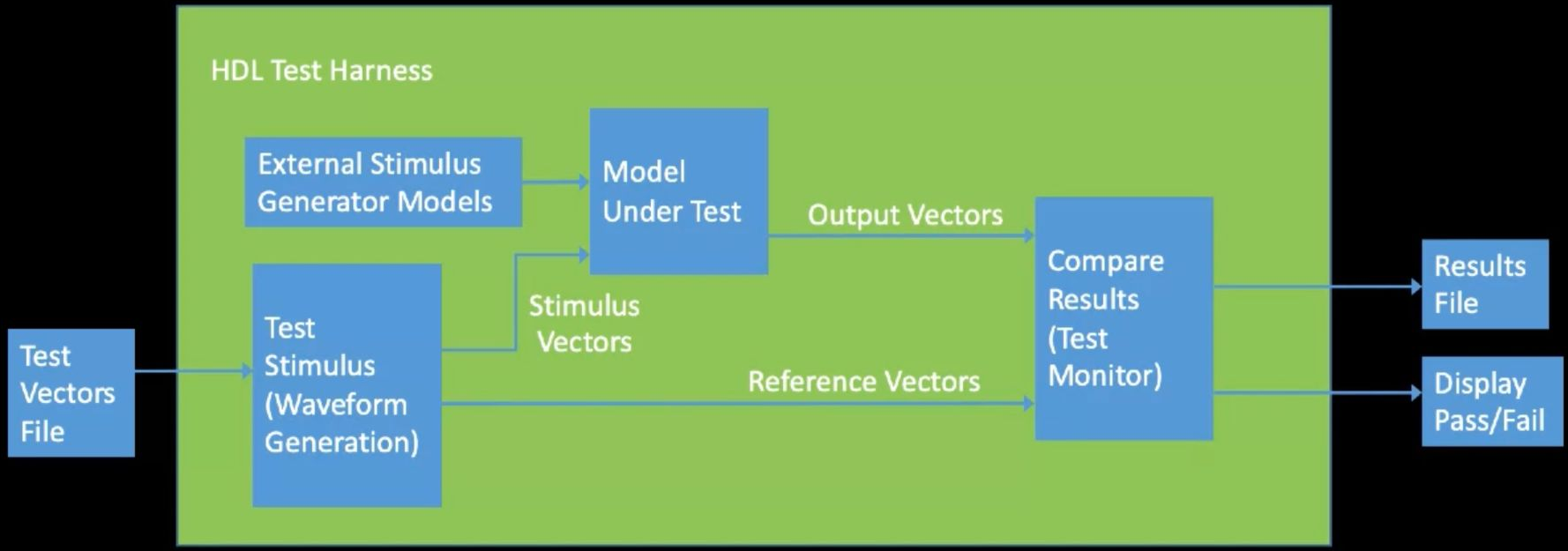
A testbench is a program written in any language for the purposes of **exercising and verifying** the **functional correctness** of the hardware model as coded.

Also known as a test fixture or test harness.
A powerful tool for auto-generating test stimulus and test results

generating test stimulus
and test results,

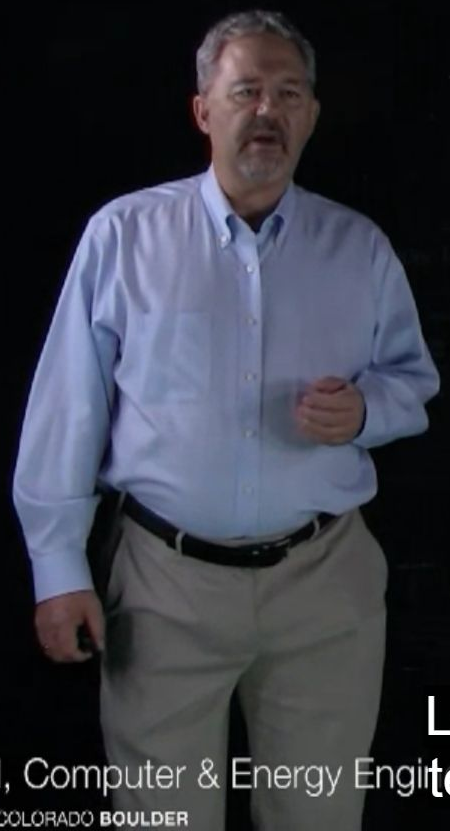

Share
Press Esc to exit full screen
Testbenches Functional Sections
A testbench can have several functional sections, including:
1. Top-level testbench declaration
2. Stimulus and Response Signal declarations
3. Component declarations
4. Component (Device Under Test) instantiations
5. External Stimulation Device Models
6. Test Process which applies the stimulus to the DUT
7. Test Monitor which reports results
and a test monitor
which reports results.
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
2:05 / 7:48
Copyright © 2019 University of Colorado

Testbench Structure



Here's a picture of the structure of a testbench.

Adder Test Bench

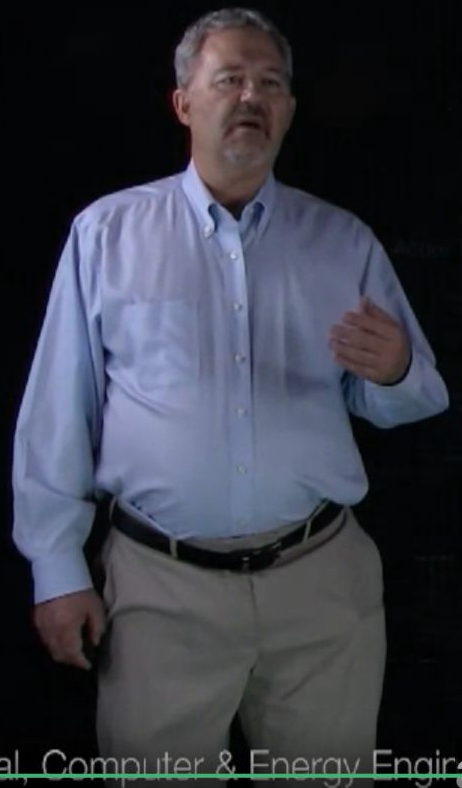


```
`timescale 1 ns / 1 ps  // set timescale to
                        //nanoseconds, ps precision
module Adder_tb(); //  no sensitivity list!
// signal declarations
reg [3:0] a_tb, b_tb;  // data input stimulus
reg Cin;               // data input stimulus
wire [3:0] y_tb;       // data output response
wire Co_tb;            // data output response
reg [3:0] expected;    // expected sum result

// DUT instantiation
add4 DUT(.A(a_tb), .B(b_tb), .Cin(Cin),
.Sum(y_tb), .Cout(Co_tb));
```

Let's construct a simple testbench for an adder.

Adder Test Bench



```
// Adder Testbench, continued
//Test stimulus generation
initial
  begin
    #0 a_tb=2; b_tb=2; Cin=0; expected=4;
    #10 a_tb=15; b_tb=0; Cin=1; expected=0;
    #10 a_tb=2; b_tb=4; Cin=1; expected=7;
    #10 $stop;
  end
// Test Results
initial
  $monitor("time=%d, a=%b, b=%b, Cin=%b, sum=%b,
cout=%b, expected sum=%b",
   $time, a_tb, b_tb, Cin, y_tb, Co_tb,
expected);
```

Continuing with the adder testbench code,

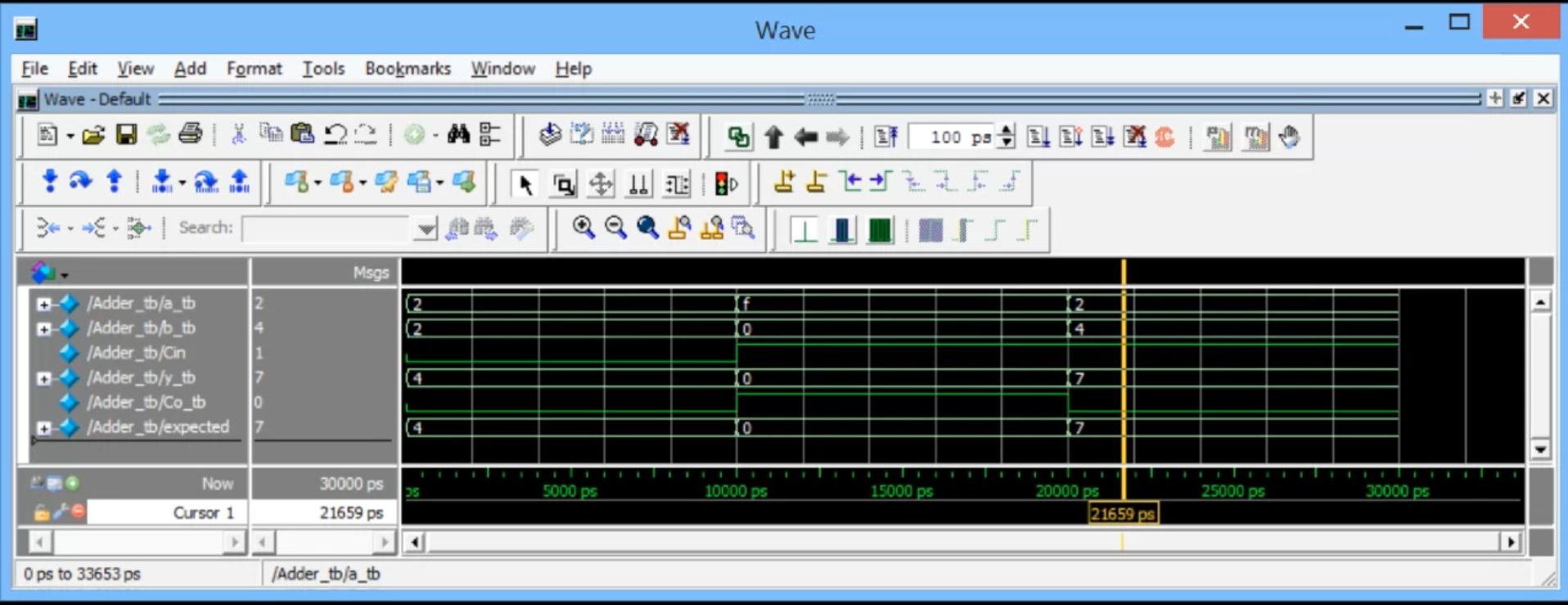
Adder Test Bench Output

```
# Loading presynth.Adder_tb
# Loading presynth.add4
# time=                    0, a=0010, b=0010, Cin=0, sum=0100, cout=0, expected sum=0100                    0
# time=                   10, a=1111, b=0000, Cin=1, sum=0000, cout=1, expected sum=0000                   10
# time=                   20, a=0010, b=0100, Cin=1, sum=0111, cout=0, expected sum=0111                   20
# ** Note: $stop    : C:/Microsemi_Prj/2019/C2M4V7/C2M4V7/stimulus/AdderMonitor_tb.v(66)
#    Time: 30 ns  Iteration: 0  Instance: /Adder_tb
# Break in Module Adder_tb at C:/Microsemi_Prj/2019/C2M4V7/C2M4V7/stimulus/AdderMonitor_tb.v line 66

VSIM 2>
```

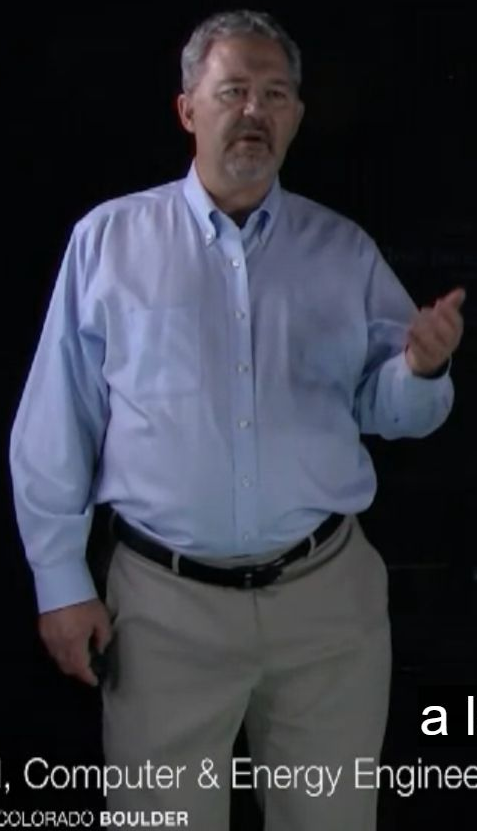
which will look like this
for a functional simulation.

Adder Test Bench Output



the testbench simulation automatically.

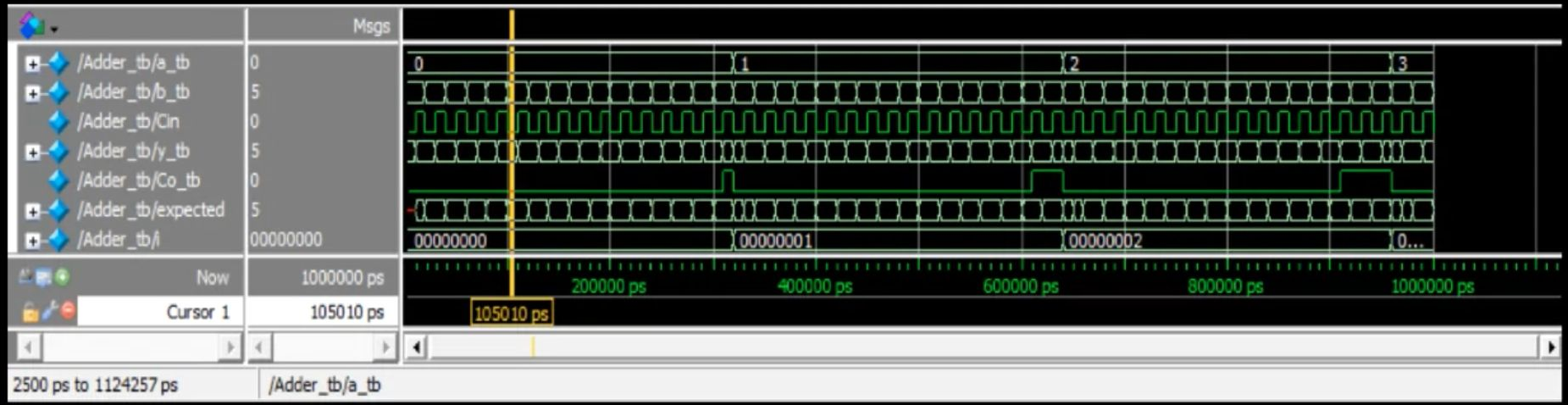
Adder Test Bench with Loops



```
integer i, j, k;
//Adder Testbench, continued
//Test stimulus generation
initial
  begin // loop over number of a inputs
        // possible
    for(i = 0; i<16; i = i+1) begin
      a_tb <= i;
      for(j=0; j<16; j = j+1) begin
        b_tb <= j;
        for (k=0; k<2; k=k+1) begin
          Cin <= k;
          #(10);
          expected <= a_tb + b_tb + Cin;
        end
      end
    end
  end
```

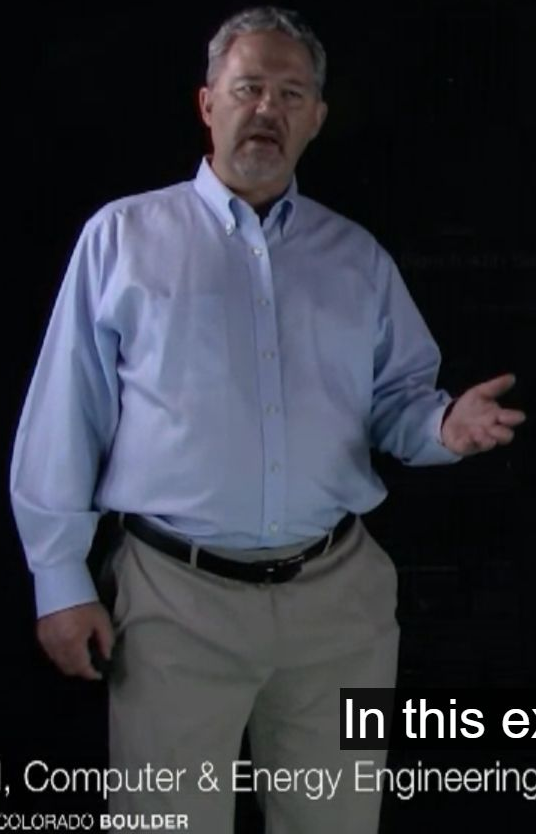
a loop to iterate through

Adder Test Bench Output



such a testbench will look like after simulation.

Adder Test Bench with Self-checking



From the previous slide, Replace

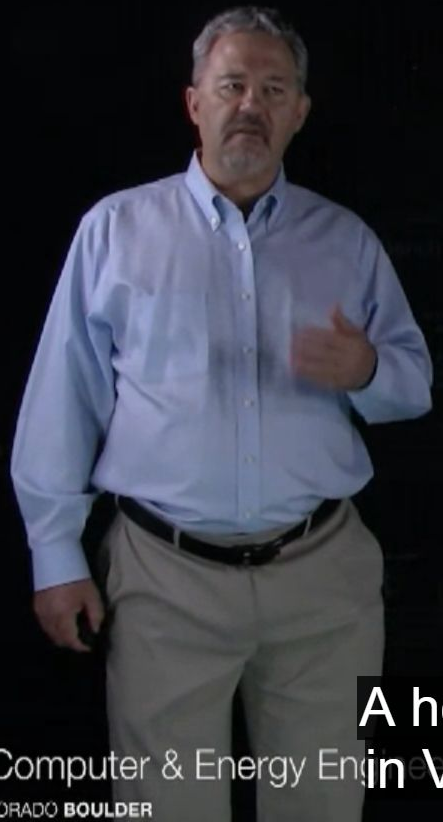
```
expected <= a_tb + b_tb + Cin;
```

with

```
if (y_tb !== a_tb + b_tb + Cin)
  begin
    $display("Error - sum is wrong");
    $stop
  end
```

In this example, any mismatch

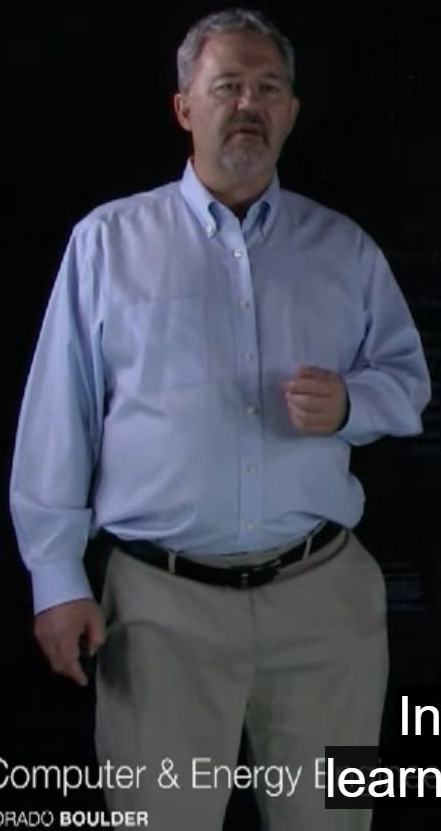
Adder Test Bench with Assertions



```
assert_label:
  assert (y_tb == a_tb + b_tb + Cin)
    begin
      $display("Test passed");
    end
    else
    begin
      $error("Error - sum is wrong");
      $stop;
     end
```

A helpful feature available
in VHDL and also system

Summary – Testbenches in Verilog I



In this video, you have learned:

- **The concept of using a Verilog program, called a testbench, to test another Verilog program**
- **How to write simple testbenches**
- **How to use loops to generate stimulus**
- **How to use assertions to determine and report test results**

In this video, you have learned the concept of using

References

[1] D. Smith, "Test Harnesses" in *HDL Chip Design, A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog,* Madison, AL, Doone Publications, 1996, ch. 11, pp. 323-344.

[2] J. Wawrzynek, (2007, Oct 17). *Verilog Tutorial.* [Online]. Available: www-inst.eecs.berkeley.edu/~cs61c/resources/verilog.pdf

[3] J. Lee, "Test Benches and Test Management" in *Verilog Quickstart, A Practical Guide to Simulation and Synthesis in Verilog*, 3rd ed. Norwell, MA, Kluwer Academic, 2002, ch. 18, p. 243.

Be Boulder.

University of Colorado **Boulder**