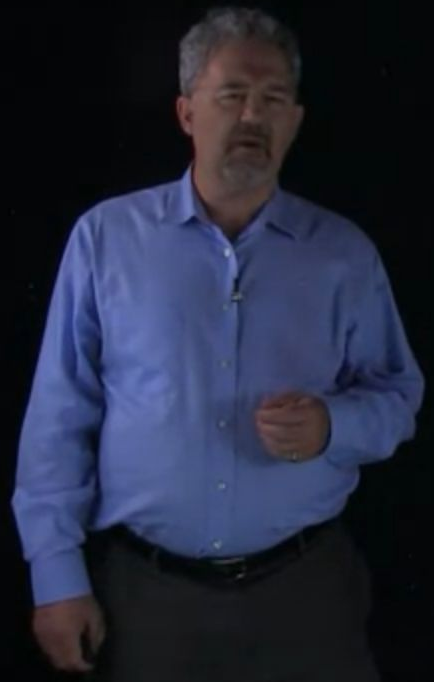
Verilog Modules, Port Modes and Data Types

Professor Tim Scherr

University of Colorado **Boulder**

Verilog Fundamentals

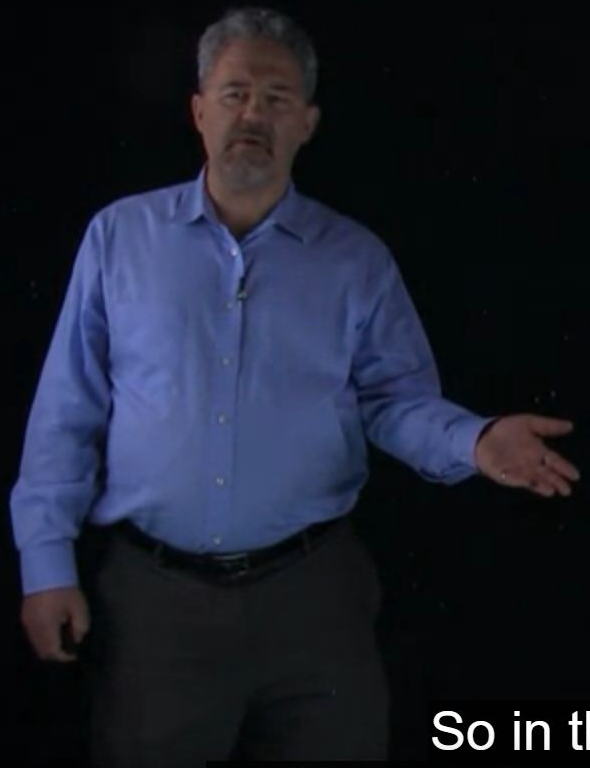


In this video, you will learn:

- Verilog Data Types: what they mean and how to use them
- The difference between Nets (wires) and Registers (storage)

Registers, which are the two main data types found in Verilog.

Verilog Fundamentals



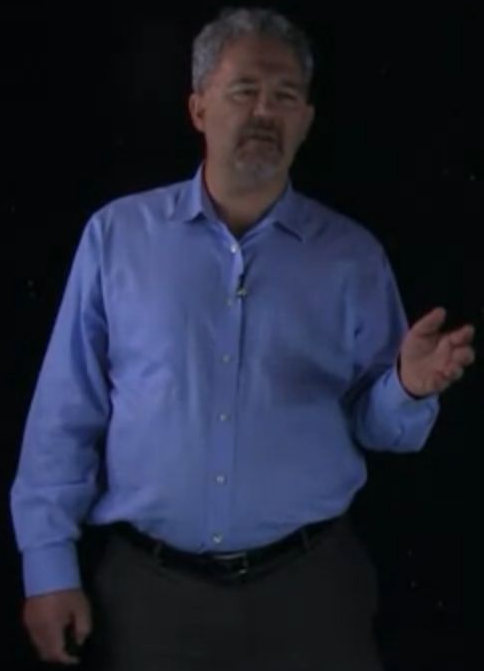
Ports

Input, Output, Inout - These keywords declare input, output and bidirectional ports of a **module** or **task**. Input and inout ports are of type **wire**. An output port can be configured to be of type **wire**, **reg**, **wand**, **wor** or **tri**. The default is wire.

```
module example(a, b, e, c)
   input a;  // An input, defaults to a wire
   output b, e;  // Outputs default to wire
   output [1:0] c;  /* 2-bit output, must be
                       declared */
                    c;     // c port declared as reg
```

So in this example, we have an input a and outputs b and

Verilog Fundamentals



Buses

Suppose we have this Verilog code:

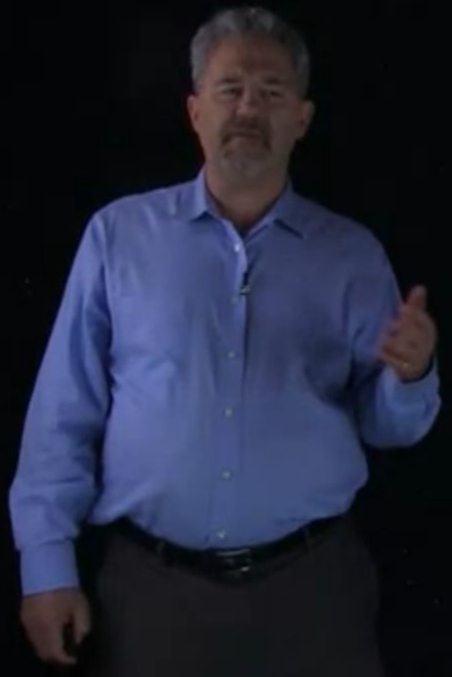
```
module invert(input [3:0] a, output [3:0] y);
  assign y = ~a;
endmodule
```

What circuit would this produce? We'd get a bank of inverters, one for every bit between the 4-bit bus a and the 4-bit bus y. In a, the bits from most significant to least are a[3], a[2], a[1] and a[0] and can be accessed individually using this nomenclature. This is the little-endian order, because the least significant bit has the smallest bit number. If it had been written a[0:3], this would be big-endian order, with the MSB accessed with the smallest bit number.

So this particular order that's used here is the little-endian

Hardware Description Languages," in *Digital*
Waltham, MA, Morgan Kaufman, 2016, pp. 178

Verilog Fundamentals



Data Types – Nets and Registers

Nets

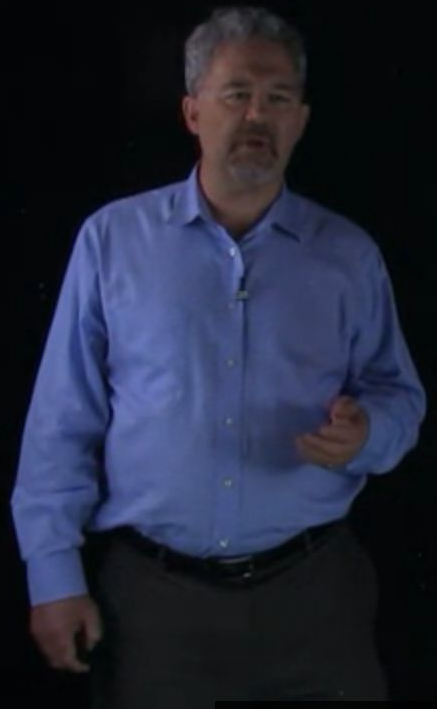
- Represent connections between hardware elements
- Must be driven continuously
- Used to wire up instantiations
- Include types wire, wor, tri and wand.

Registers

- Retain the last value assigned
- Often used to represent storage elements
- Includes types reg and integer

So then Registers are different from Nets.

Verilog Fundamentals



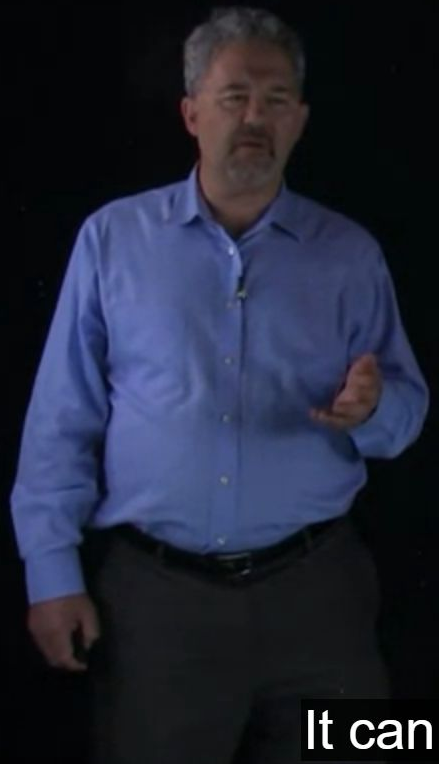
Data Types

Integers are general-purpose variables.

- For synthesis used mainly as loop indexes, parameters, and constants.
- Implicitly of type **reg,** however they store data as signed numbers whereas explicitly declared **reg** types store them as unsigned.
- If they hold numbers which are not defined at compile time, their size will default to 32-bits.
- If they hold constants, the synthesizer adjusts them to the minimum width needed at compilation.

the synthesizer will adjust them to the minimum width needed at compilation time.

Verilog Fundamentals



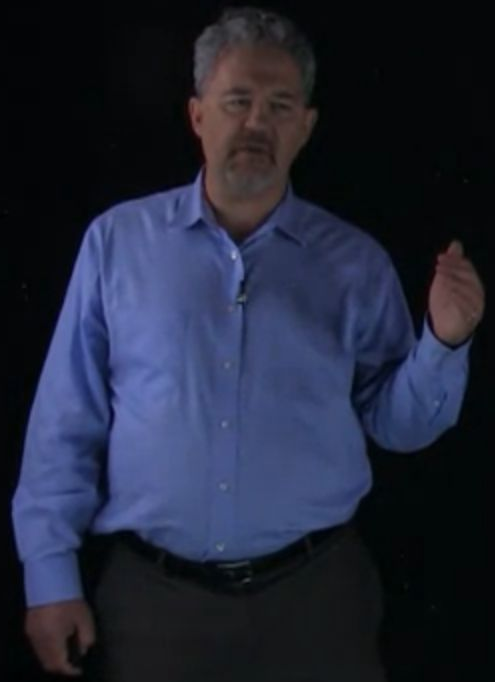
Data Types

Reg represents storage.

- Only reg type variable can be assigned to in an always block.
- Reg does not mean register. It can be modeled as a wire or as a storage cell depending on context. When used in combinational expressions in an always block, no storage is implemented. When used in sequential statements (begin/end, if, for, case, etc.), the a latch or FF will be created.

It can be modeled as a wire or

Verilog Fundamentals



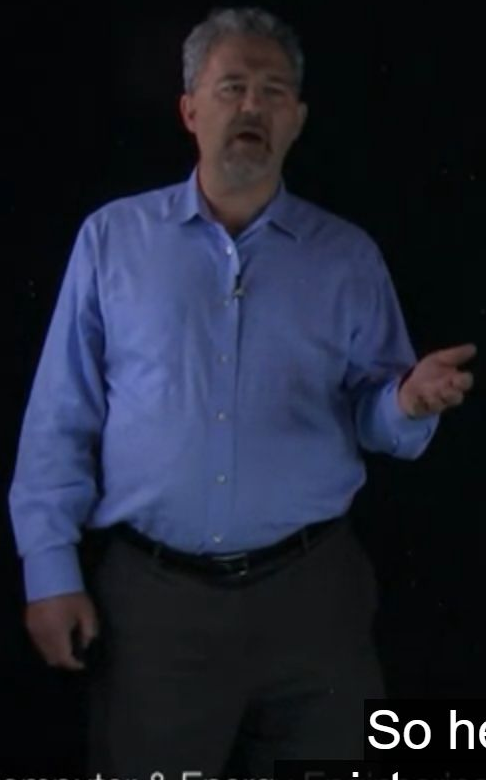
Port Rules

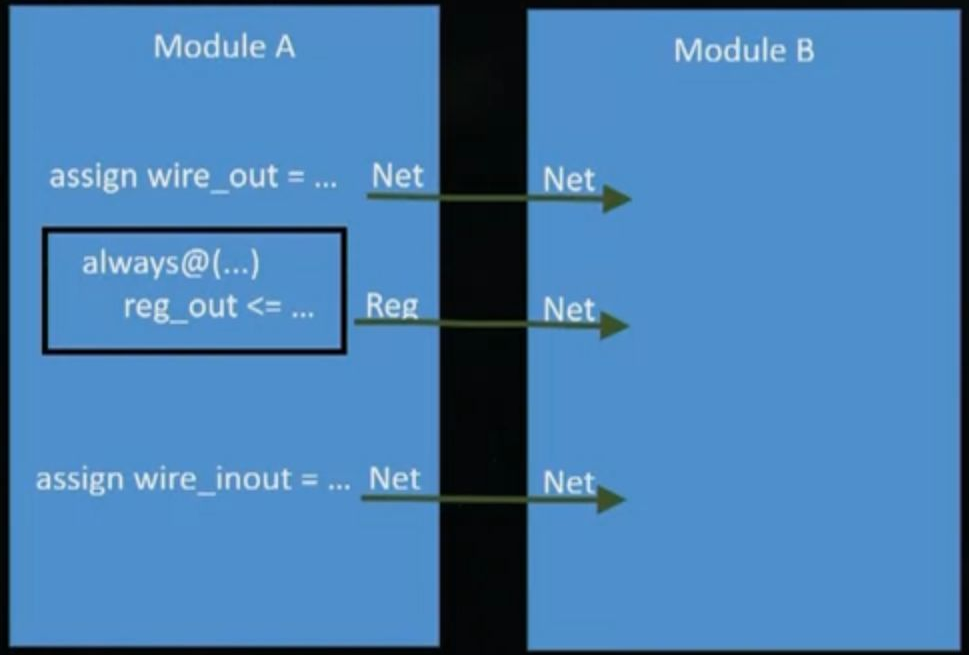
When connecting modules,
Inputs must be Nets (wire, etc.)
Outputs can be Nets or Registers
Inout must be Nets
The Left Hand Side (LHS) of **procedural assignment**s must be of a **Register** type.
For **continuous assignment**s outside of procedural blocks, LHS must be **Nets** (wires).
These rules pictured visually on the next slide.

For continuous assignments that are outside of procedural blocks,

Verilog Fundamentals

Port Rules

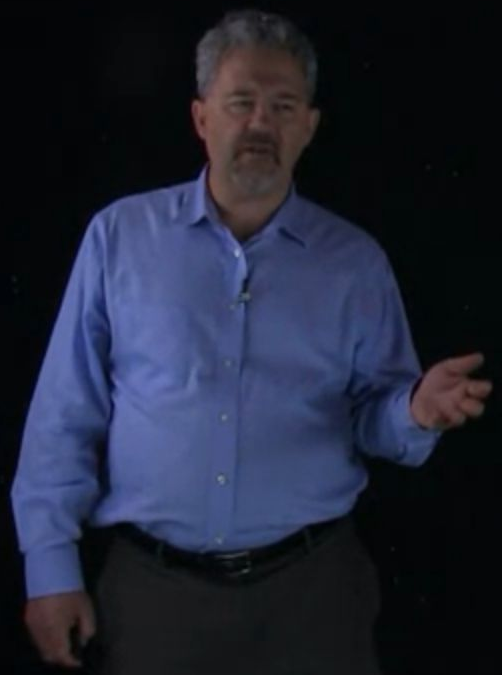




So here we kind of have
a picture where we see the wire

Verilog Fundamentals

Data Types



supply0 and **supply1** define wires tied to logic 0 (ground) and logic 1 (power), respectively.

Time is a 64-bit quantity that can be used in conjunction with the $time system task to hold simulation time. Time is not supported for synthesis and hence is used only for simulation purposes.

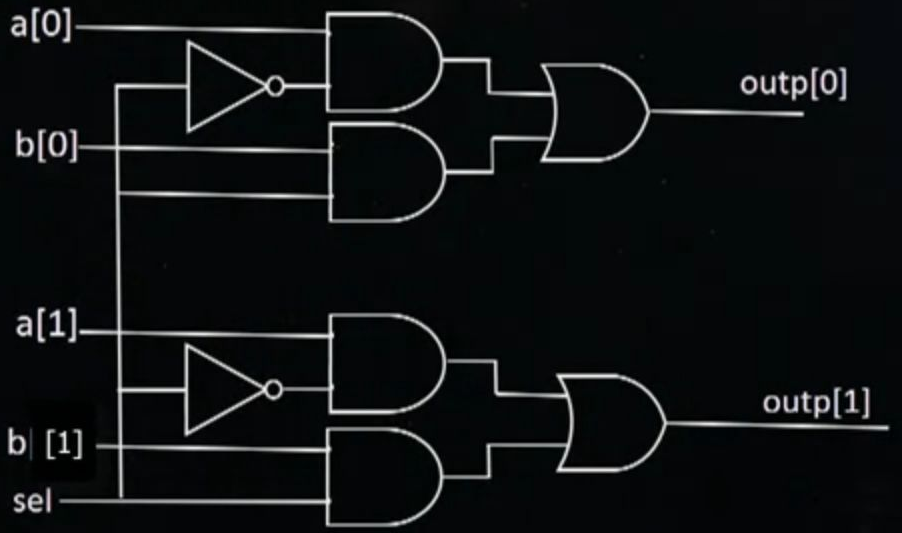
Parameters allows constants like word length to be defined symbolically in one place. This makes it easy to change the word length later, by changing only the parameter.

This makes it easy to change the word length later across the entire

Verilog Fundamentals

Parameters and Buses

Assume we wanted to create this circuit:

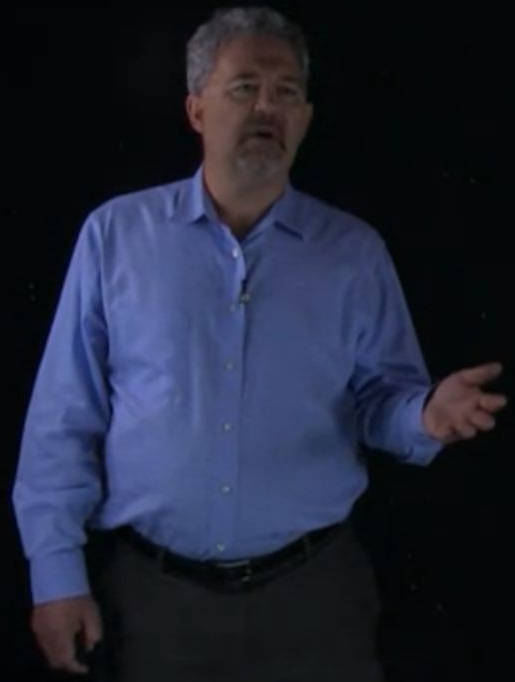


How would this be coded in Verilog?

So now, knowing what you know about data types and vectors and

Verilog Fundamentals

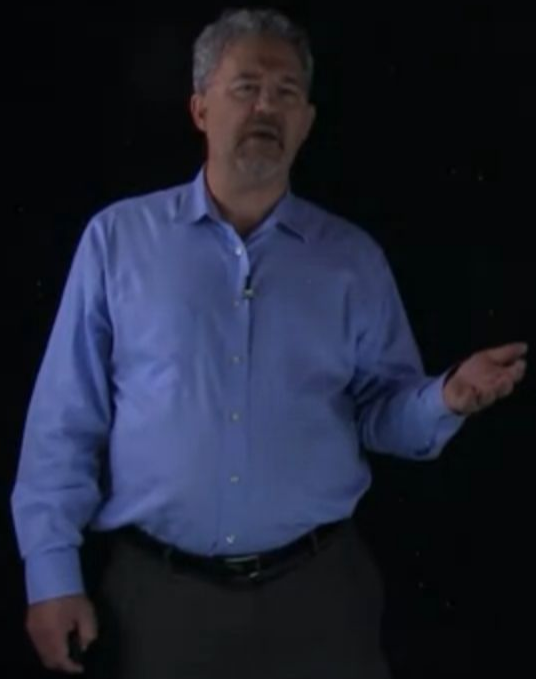
Parameters and Buses



```
module bus_mux(a, b, sel, outp);
   parameter n = 2;
   input  [n-1:0] a;
   input [n-1:0] b;
   input sel;
   output outp;
   wire [n-1:0] sel_bus;
assign sel_bus = {n{sel}};  // replicates
                                        2 times
assign outp = (~sel_bus & a) | (sel_bus & b);

endmodule
```

The parameter n = 2 would be n equals 8, 16, 32 or whatever.

Summary – Verilog Data Types



In this video, you have learned:

- Verilog Data Types: what they mean and how to use them
- How to use Parameters and make buses
- The difference between Nets (wires) and Registers (storage)

what they mean, how to use them,
how to use parameters.

Be Boulder.

University of Colorado **Boulder**

Verilog Structure

Professor Tim Scherr

University of Colorado **Boulder**

Verilog Structure

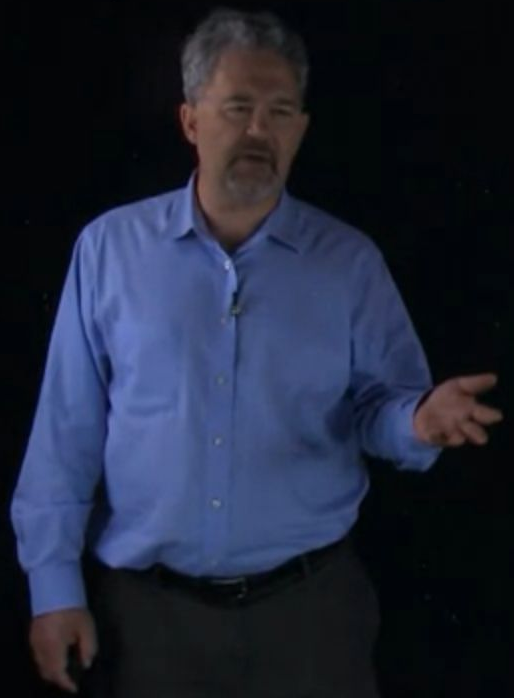
In this video, you will learn:

- How to use modules as building blocks for Verilog circuit design descriptions
- How to include an instance of another module within a module to build hierarchical designs
- 2 ways to instantiate a module, and which one is better.

and two ways to
instantiate a module,

Verilog Structure

Modules



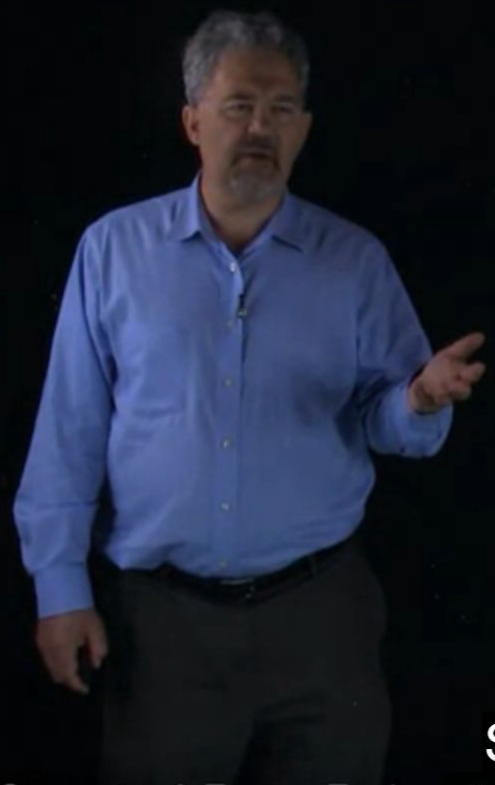
Modules are the basic building blocks in Verilog. A module definition always starts with the keyword module, followed by the **module name**, **port list**, **port declarations**, and **parameters**. This is followed in any order by:

- Variable definitions (local scope)
- Dataflow statements
- Instantiation of lower modules
- Behavioral Blocks
- Tasks or Functions

Followed by the keyword endmodule (no semicolon)

which isn't going to have a semicolon after it.

Verilog Structure



module module_name (port_list);

I. Port declarations

II. Variable definitions

III. Parameters

IV. Data Flow statements (assign …)

V. Module Instantiations

VI. Behavioral Blocks (begin…end)

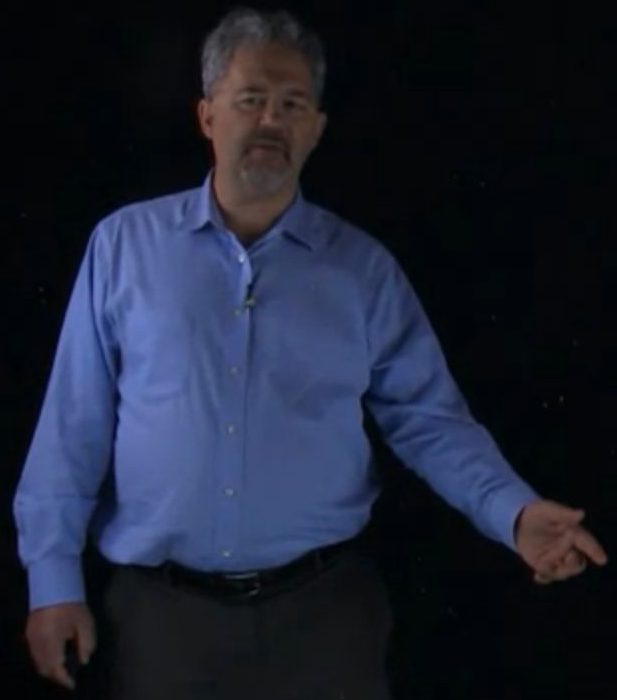
VII. Tasks or Functions

VII. Timing Specifications

endmodule

So if you need a picture of a module,

Verilog Structure



Module Instantiation

Modules can be instantiated by using ordered port lists, or by port names which don't need to follow the module order. For example, here are 2 ways to instantiate a full adder:

```
reg [3:0] A, B;   // top level signals in caps
reg C_IN;
wire [3:0] SUM;
wire C_OUT;
fulladd4 faordered(SUM, C_OUT, A, B,
C_IN);
fulladd4 fa_name(.sum(SUM),
                 UT),.a(A),.b(B), .cin(C_IN));
```

you see a couple of instantiations both

Verilog Structure

Making Bigger Circuits from Smaller

Recall the Verilog code for a 2-input mux:

```
module mux2(a, b, sel, y);
  input [3:0] a, b;
  input sel;
  output [3:0] y;

  assign y = sel ? b : a;
endmodule
```

How can we use this module to build a mux4?

larger circuits out of
smaller building blocks.

Verilog Structure

Making Bigger Circuits from Smaller

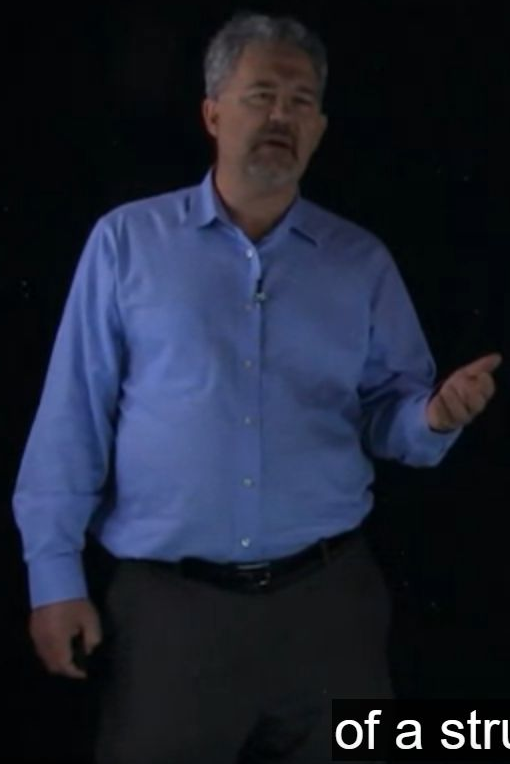
We can casacade the 2-input mux several times to make a 4-input mux :

```
module mux4(a, b, sel, y);
   input [3:0] a, b;
   input sel;
   output [3:0] y;

   wire[3:0] low, high;
   mux2  lowmux(a, b, s[0], low);
   mux2  highmux(c, d, s[0], high);
   mux2  outmux(low, high, s[1], y);
endmodule
```

We could have also made this by nesting conditional statements

would be scalars by default,

Verilog Structure



module module_name (port_list);

I. Port declarations and Parameters

II. Variable definitions

III. Parameters

IV. Data Flow statements (assign ...)

V. Module Instantiations

VI. Behavioral Blocks (begin...end)

VII. Tasks or Functions

VII. Timing Specifications

endmodule

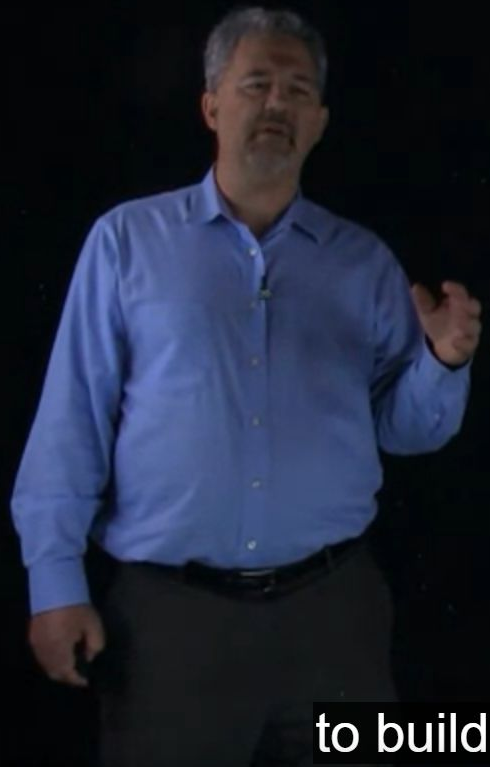
of a structure of a module.

Example: Multiply-Accumulator

```
`timescale 1 ns/ 100 ps  // time resolution/precis for sim
module mult_acc (              // module and name
  input [7:0] data1, data2,       // I. port declarations
  input clk, aclr,                 // I. port declarations
  output reg [15:0] mac_out ); // I. port declarations
  wire [15:0] mult_out, add_out; // II. variable definition
  parameter mult_size = 8;       // III. parameter
  assign add_out = mult_out + mac_out;  // IV.dataflow
  multiplr #(.width_in(mult_size))    // V. instantiation
    u1 (.in_a(data1), .in_b(data2), .mult_out(mult_out));
  always @ (posedge clk, posedge aclr) begin
  if (aclr)                                // VI. behavioral block
    mac_out <= 16'h0000;
  else
    mac_out <= adder_out;
  end                                      // end behavioral block
```

in the always behavioral block that's listed here.

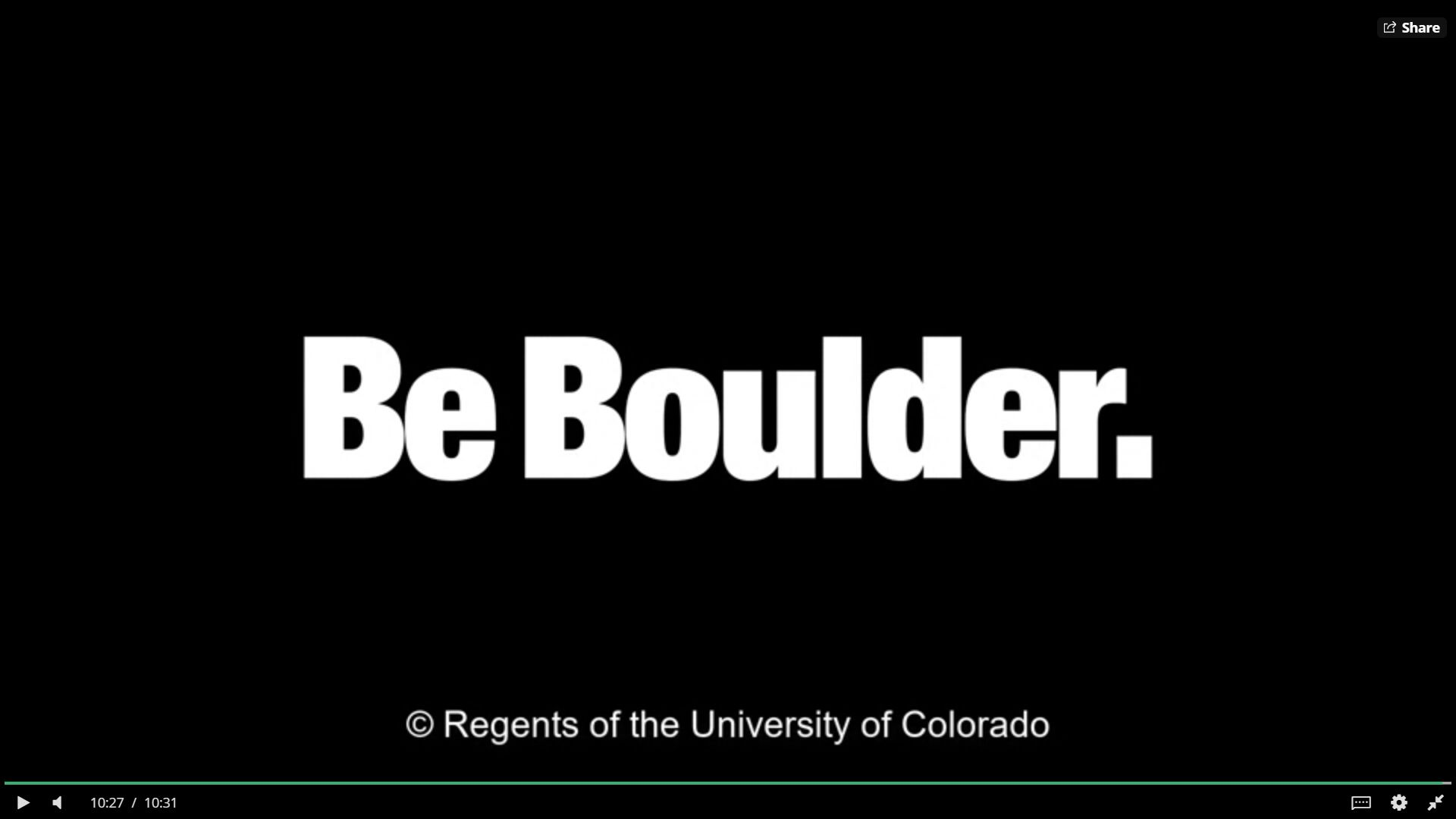
Summary – Verilog Structure



In this video, you have learned:

- **How to use modules as building blocks for Verilog circuit design descriptions by instantiating, or including an instance of the module**
- **2 ways to instantiate a module, and which one is better.**
- **How to include an instance of another module within a module to build hierarchical designs**

to build a hierarchical design.


Share
Be Boulder.
© Regents of the University of Colorado
10:27 / 10:31

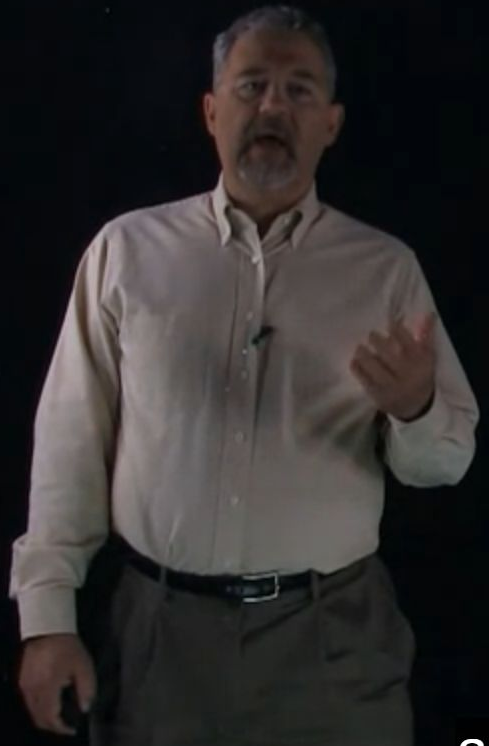
University of Colorado **Boulder**

Testing with Modelsim

Professor Tim Scherr

University of Colorado **Boulder**

Verilog Evaluation



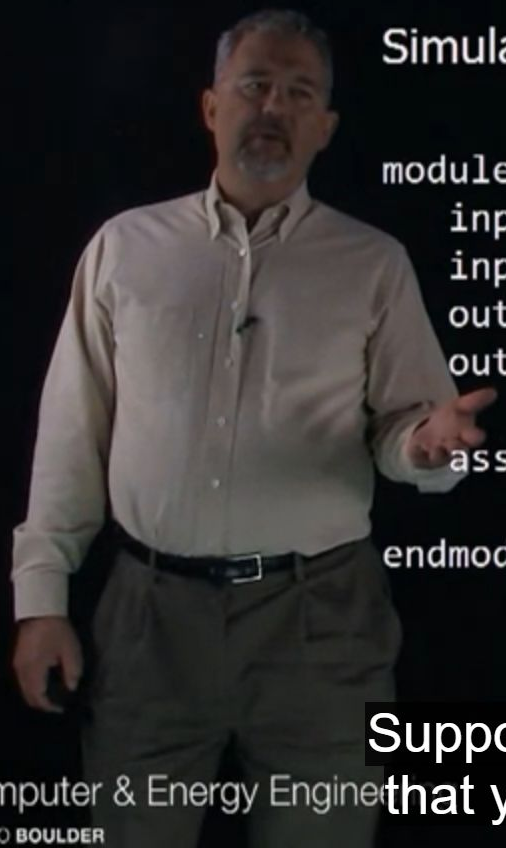
In this video, you will learn:

- **How to download an HDL Simulator, ModelSim from Mentor Graphics**
- **How to install ModelSim on your PC**

and how to install
ModelSim on your PC.

Electrical, Computer & Energy Engir
UNIVERSITY OF COLORADO **BOULDER**

Verilog Evaluation

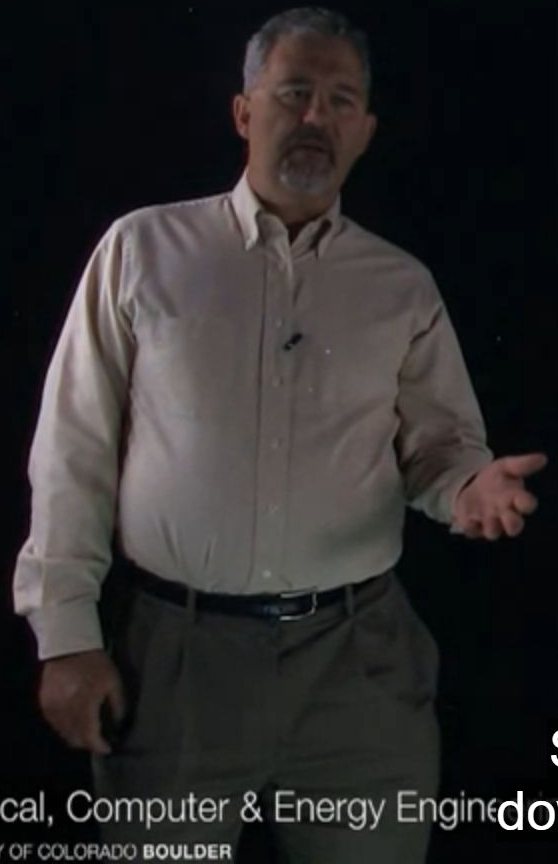


Simulation Example: 4-bit Adder

```
module add4 (                  // module and name
   input [3:0] Data1, Data2, // I. port declarations
   input Cin,
   output Cout,
   output wire [3:0] Sum );

   assign {Cout, Sum} = Data1 + Data2 + Cin;

endmodule
```

Suppose for example
that you had written

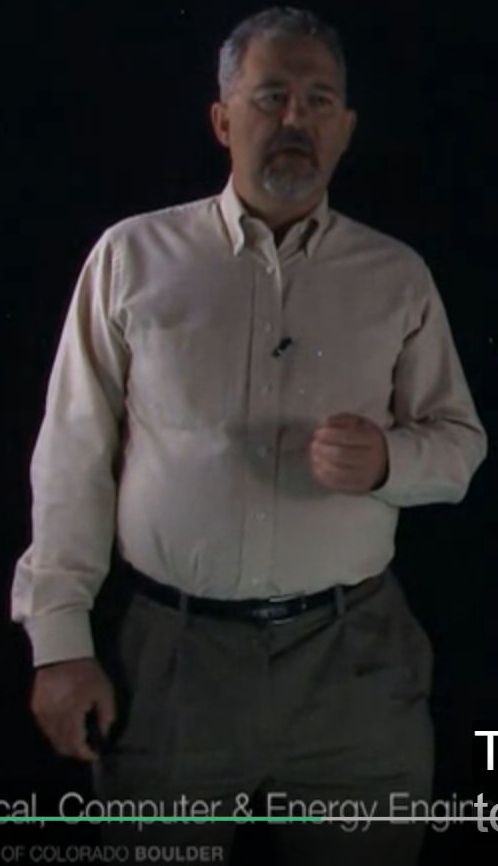
Downloading ModelSim



Go to https://www.mentor.com/company/higher_ed/modelsim-student-edition.

So here's how to download ModelSim.

Downloading ModelSim Alternative

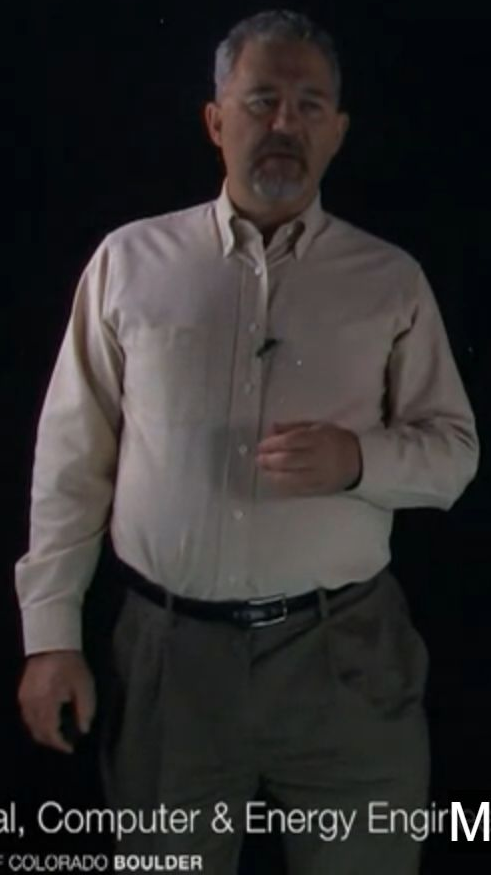


If you took the Introduction to FPGA Design Course, Course 1 of this series, you already have the ModelSim-Altera version of ModelSim installed.
If you did not take Course 1 this is still a good alternative.

Follow the directions in ALTERA QUARTUS DOWNLOAD AND INSTALLATION.docx to install this version of ModelSim.

There is an alternative to the student version.

Summary – Verilog Evaluation

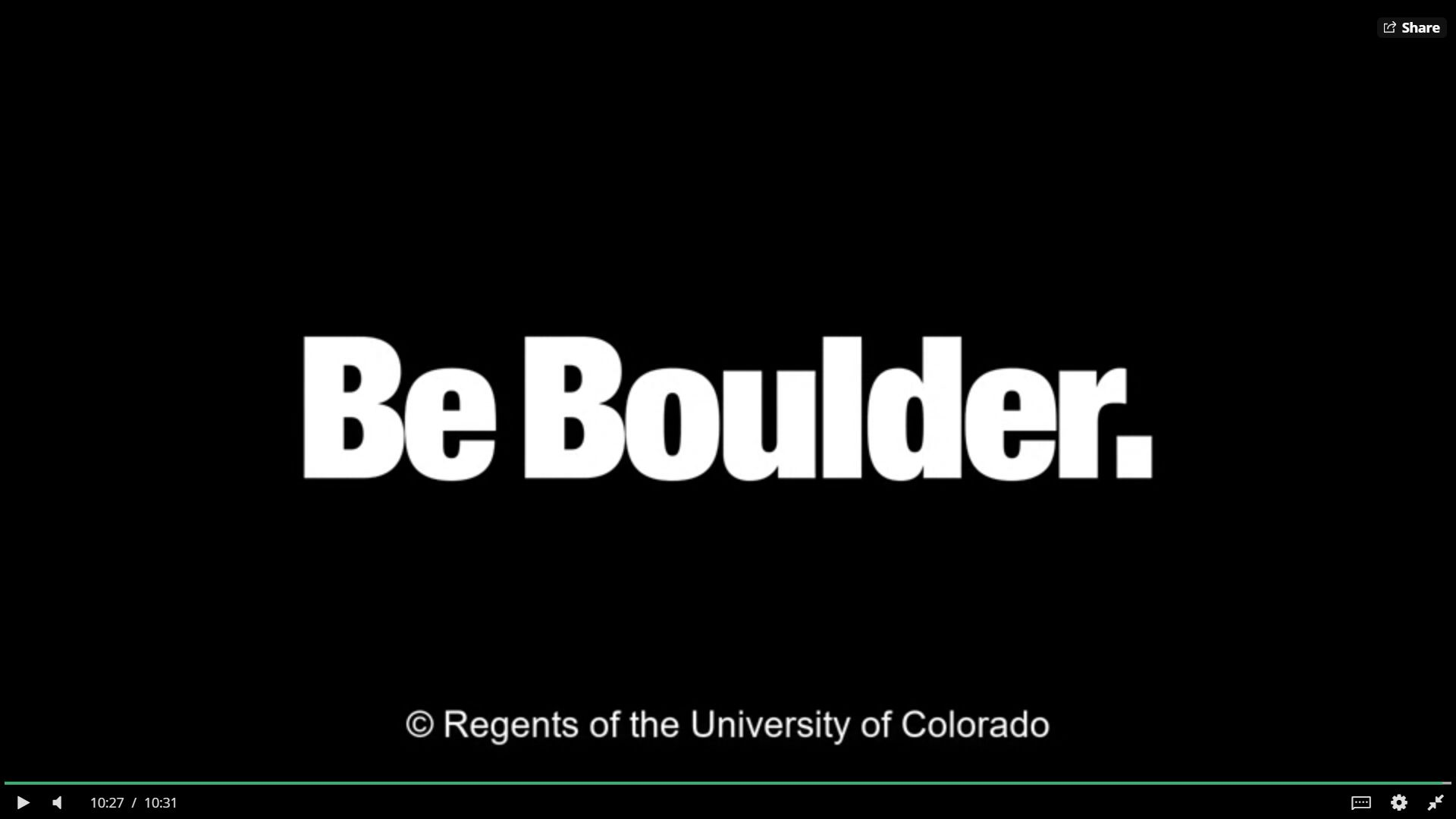


In this video, you have learned:

- **How to download an HDL Simulator, ModelSim from Mentor Graphics**
- **How to install ModelSim on your PC**

and how to install
ModelSim on your PC.

Electrical, Computer & Energy Engir
UNIVERSITY OF COLORADO **BOULDER**


Share
Be Boulder.
© Regents of the University of Colorado
10:27 / 10:31

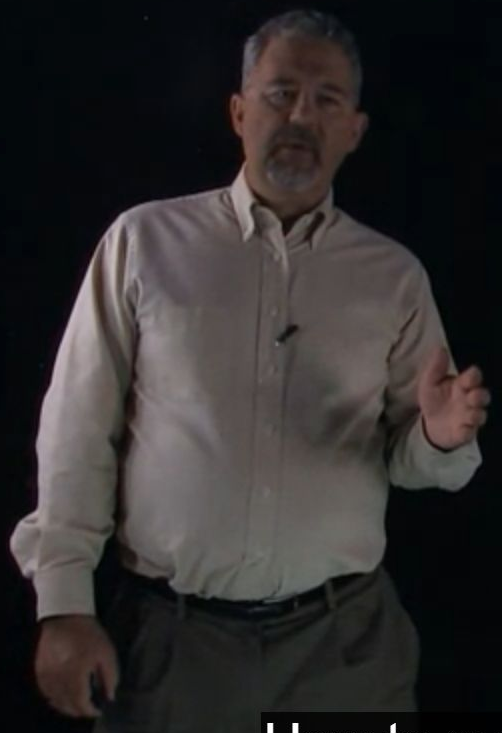
University of Colorado **Boulder**

Verilog Evaluation

Professor Tim Scherr

University of Colorado **Boulder**

Verilog Evaluation



In this video, you will learn:

- **How to test the code your write using an HDL simulator.**
- **How to evaluate the correctness of your design using waveform analysis**
- **How to control the simulator to examine the code and its output.**

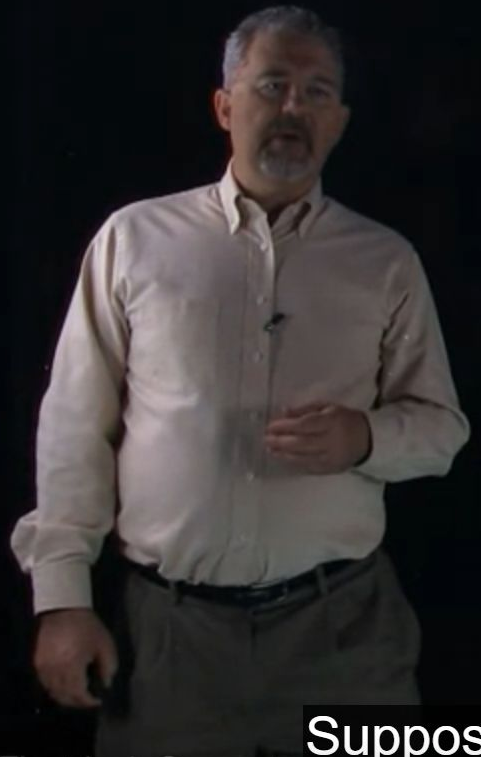
How to evaluate the correctness of your design using waveform analysis.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

Share

Press Esc to exit full screen

Verilog Evaluation



Simulation Example: 4-bit Adder

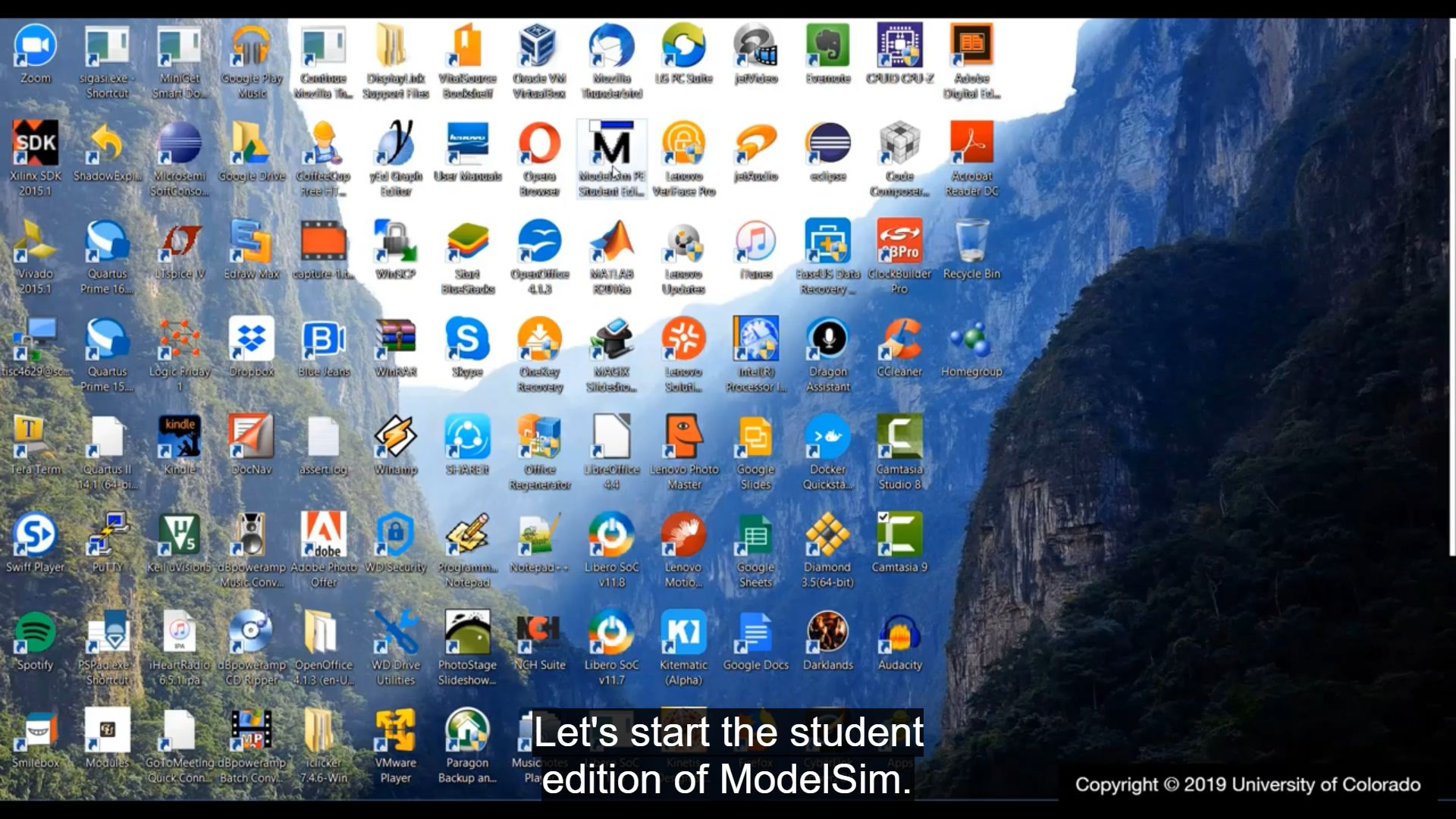
```
module add4 (                  // module and name
   input [3:0] Data1, Data2, // I. port declarations
   input Cin,
   output Cout,
   output wire [3:0] Sum );

   assign {Cout, Sum} = Data1 + Data2 + Cin;

endmodule
```

Suppose for example, you had written this Verilog code to implement a 4-bit adder.

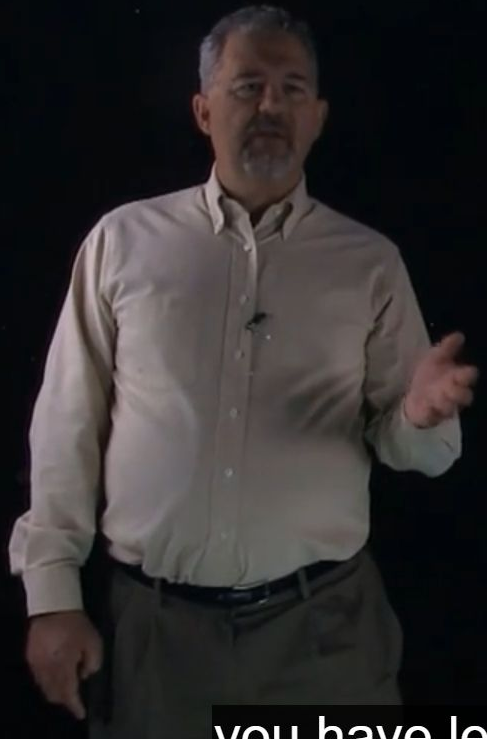
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

0:27 / 11:24


Share
Using ModelSim
Come, Simulate with me!
Let's simulate the Add4.v code in ModelSim. Start ModelSim now and follow the steps as we simulate this code together.
Start ModelSim now and follow the steps
as we simulate this code together.
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
1:22 / 11:24



Summary – Verilog Evaluation



In this video, you have learned:

- **How to test the code your write using an HDL simulator, ModelSim.**
- **How to evaluate the correctness of your design using waveform analysis.**
- **How to control the simulator to examine the code and its output. In this case, a 4-bit adder using different number combinations to get good test coverage.**

you have learned how to test the code
you write using HDL Simulator ModelSim.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

Be Boulder.

University of Colorado **Boulder**

FPGA Design for Embedded Systems

Hardware Description Languages for Logic Design

University of Colorado Boulder

Welcome to the FPGA design for embedded systems,

Submitting Verilog Programming Assignments

Professor Tim Scherr

University of Colorado Boulder

submit your Verilog programming assignments

Submitting Verilog Programming Assignments

1. Write your Verilog code for the programming assignment as directed by the Application Assignment document.
2. Download the *_tb.vp testbench file from the Coursera Platform corresponding to programming assignment for each program. For example, for AAC2M3P1, download AAC2M3P1_tb.vp
3. Place your .v code file, the file vector.out, and the testbench file in a ModelSim project directory.
4. Run the testbench in Modelsim. You should see a myvector.out output file generated.
5. Upload the myvector.out file and your .v file for this programming assignment to the Coursera Platform.

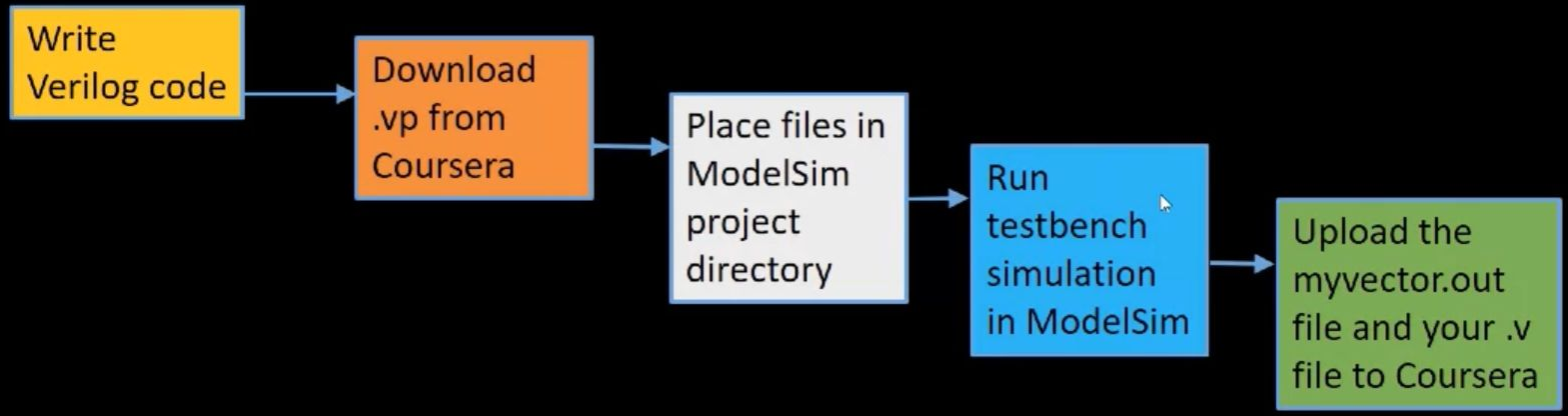
vector.out file and your.v file,

CORRECTION

You only need to submit your myvector.out file to complete the programming assignment. This is the file that will be autograded. Save your .v file on your local drive for your own future reference.

Continue

Submitting Verilog Programming Assignments



you'll run the test bench simulation in ModelSim,

Again, remember that you only need to submit your myvector.out file to complete the programming assignment.

Continue

CORRECTION

These files are NOT located in "Course Resources". All .v, vector.out, and _tb.vp files are in a .zip file in the reading items "Files for Week 3 Programming Assignments" and "Files for Week 4 Programming Assignments". Download the .zip file for the assignment you're working on to access the correct files.

Continue

Summary

In this video you have learned how to submit your Verilog Programming assignments to the Coursera platform.

This involves a 5 step process which may seem complicated at first, but should be easy to do after you have reviewed this video and submitted your first Verilog programming assignment.

but should be easy to do
after you have reviewed

Be Boulder.

University of Colorado **Boulder**