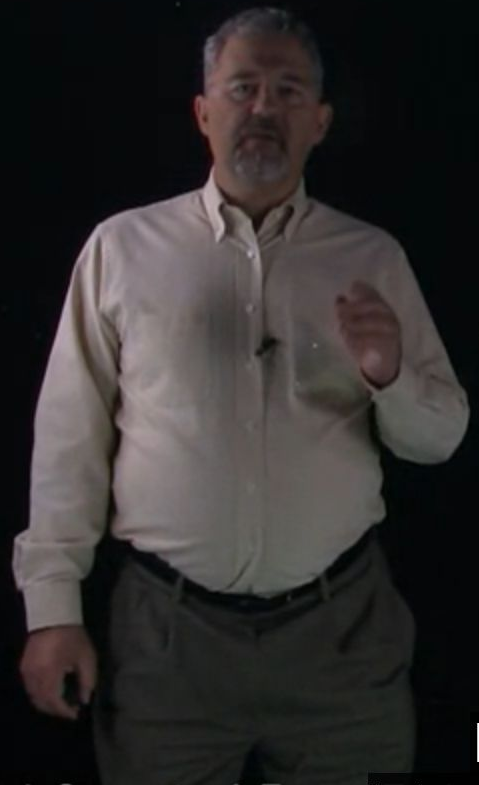
Introduction to Verilog

Professor Tim Scherr

University of Colorado **Boulder**

An Introduction to Verilog



Verilog is one of the most popular, if not the most popular, languages for logic design.

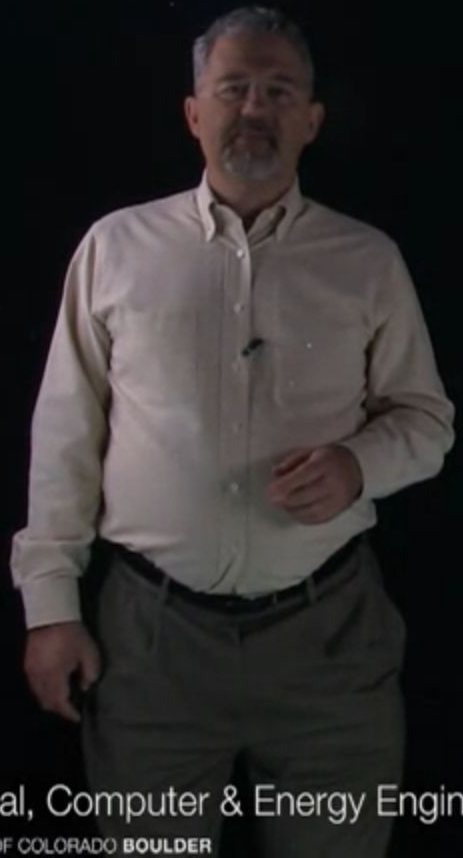
In this Module we will introduce the Verilog language so that you can begin to design logic circuits with it.

In this module, we will introduce the Verilog language

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

Copyright © 2019 University of Colorado

Why learn Verilog?



It's fun.

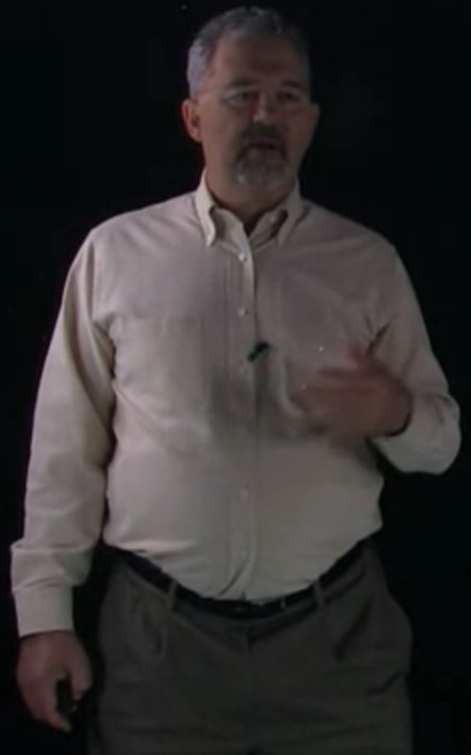
You can make a living doing this.

It's better than writing code in VHDL.

So let's learn Verilog
for fun and profit.

Copyright © 2019 University of Colorado

How can you learn Verilog?



Like any other language…

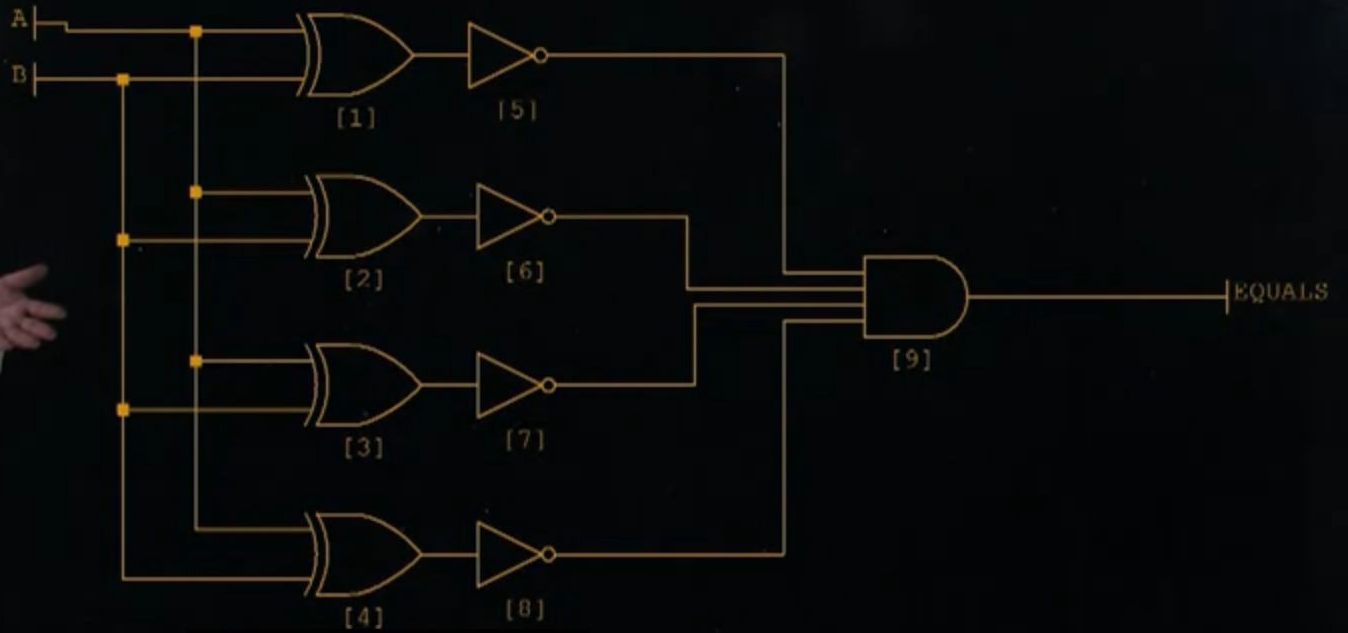
If you were going to learn Spanish…

- Learn key phrases, practice them
- Learn Grammar and Syntax
- Make compound sentences
- Practice, Practice, Practice

together and then give
you lots of practice.

Share
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
2:15 / 3:35

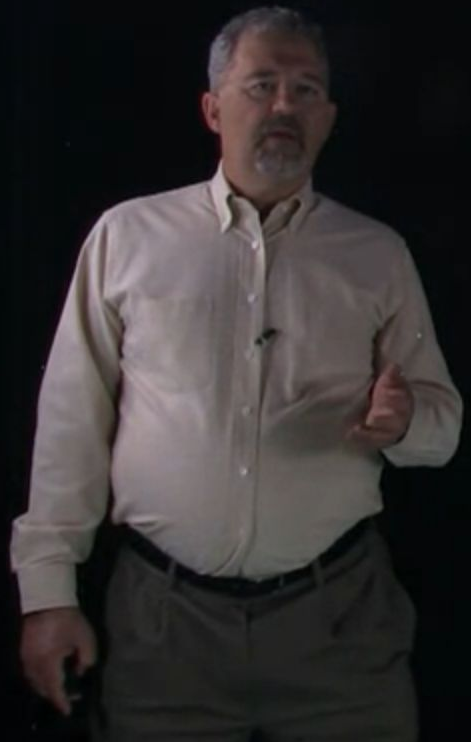
Writing Verilog for A 4-bit Comparator



Verilog code that describes the circuit.

Electrical, Computer & Energy Engine
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

Videos in this Module



1. Verilog for fun and profit
2. Your First Verilog Phrase
3. Verilog Rules and Syntax
4. Verilog Statements and Operators
5. Verilog Modules, Port Modes and Data Types
6. Verilog Structure
7. Testing Your Verilog with ModelSim: Download and Installation
8. Testing Your Verilog with ModelSim: Adding to Your Toolkit

So here are the videos in this module.

Copyright © 2019 University of Colorado

Be Boulder.

© Regents of the University of Colorado

University of Colorado **Boulder**

© Regents of the University of Colorado

Share

Your First Verilog Phrase

Professor Tim Scherr

University of Colorado **Boulder**

0:01 / 11:07

An Introduction to Verilog

In this video, you will learn:

- **The history of the Verilog HDL Language**
- **An approach to learning Verilog**
- **A "first phrase" design example (a comparator), done 3 different ways**

the history of the
Verilog HDL language,

Copyright © 2019 University of Colorado

An Introduction to Verilog

Verilog, standardized as **IEEE 1364**, is a hardware description language (HDL) used to model electronic systems.

It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction.

In Verilog, a design consists of modules.

A simple AND gate in Verilog would look something like the following:

In addition, most designs import library modules.

Share
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2013 University of Colorado
0:45 / 11:07

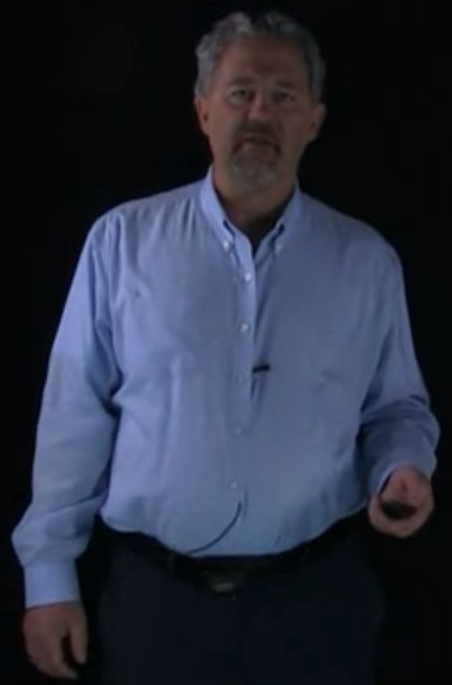
An Introduction to Verilog

```
// And gate
//
module andgate (x1, x2, f);
        input x1, x2;
        output f;
        assign f = x1 & x2;
endmodule
```

So here we see an initial
couple of lines of comments,

Copyright © 2019 University of Colorado

What is Verilog?



Verilog = **Verif**ying **Log**ic

Verilog - modeling language created by Gateway Automation in 1984. Then to Cadence -> open standardization. Now public domain -> Open Verilog International (OVI) (now known as Accellera) organization.

Verilog was later submitted to IEEE and became IEEE Standard 1364-1995, commonly referred to as Verilog-95.

commonly referred to as Verilog-95.

Share

Copyright © 2019 University of Colorado

What is Verilog?

Extensions to Verilog-95 were submitted back to the IEEE. These extensions became IEEE Standard 1364-2001 known as Verilog-2001. Another update occurred in 2005.

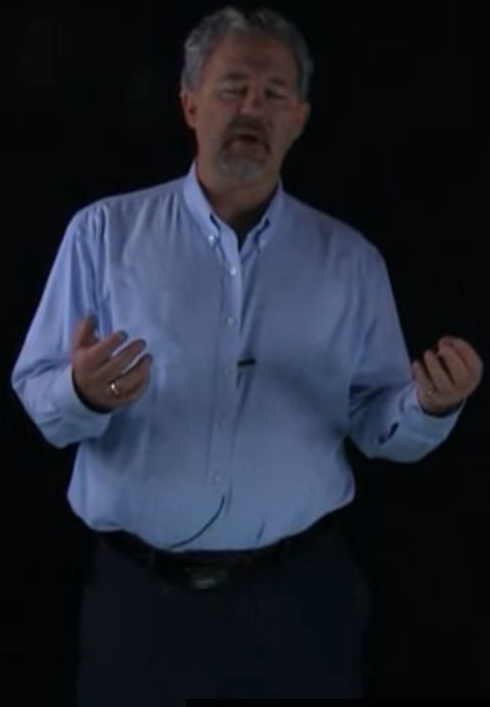
As of 2009, the SystemVerilog and Verilog language standards were merged into SystemVerilog 2009 (IEEE Standard 1800-2009). The current version is IEEE standard 1800-2017.

SystemVerilog 2009 which is now IEEE Standard 1800-2009.

https://en.wikipedia.org/wiki/Verilog

Copyright © 2019 University of Colorado

How can you learn Verilog?



Like any other language…

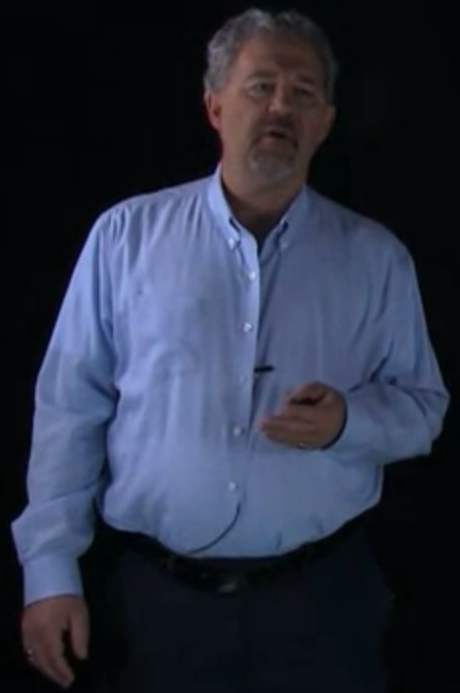
If you were going to learn Spanish…

- Learn key phrases, practice them
- Learn Grammar and Syntax
- Make compound sentences
- Practice, Practice, Practice

practice until the point comes

Share
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
4:59 / 11:07

Three Modeling Styles in Verilog



Structural modeling (Gate-level)

Use predefined or user-defined primitive gates.

Dataflow modeling

Use assignment statements (assign)

Behavioral modeling

Use procedural assignment statements (always)

procedural statements known as

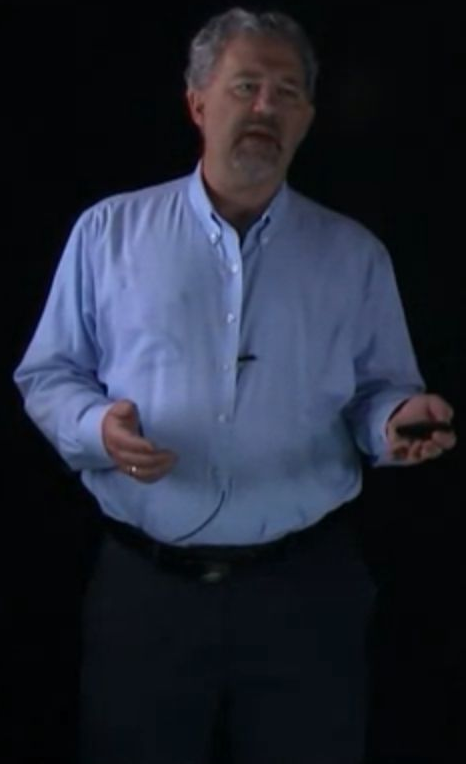
Share

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

Copyright © 2019 University of Colorado

7:01 / 11:07

Structural 4-bit Comparator

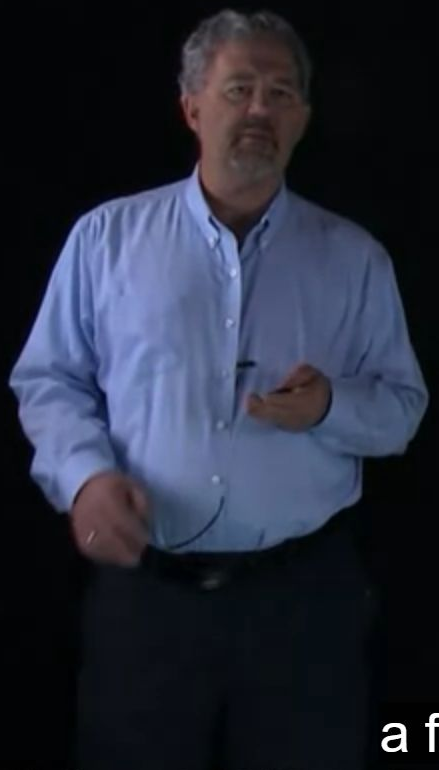


```
module comparator(
input[3:0] a,b,output out);   // Verilog-2001


wire n0, n1, n2, n3;
xnor xnor0( n0, a[0], b[0] );
xnor xnor1( n1, a[1], b[1] );
xnor xnor2( n2, a[2], b[2] );
xnor xnor3( n3, a[3], b[3] );
and and0( out, n0, n1,n2, n3 );
endmodule
```

It uses library modules for the gates

Copyright © 2019 University of Colorado

Dataflow 4-bit Comparator



```
module COMPARATOR
    (A, B, Y);              // Verilog-1995 Syntax
    input [3:0] A, B ,
    output Y;

    assign Y = & ( (A~^B));
endmodule
```

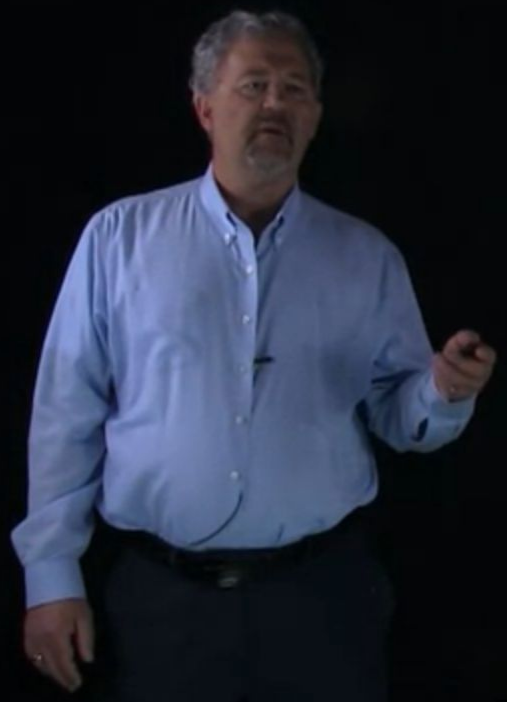
a function of the input vectors.

Copyright © 2019 University of Colorado

Share

Press Esc to exit full screen

Behavioral 4-bit Comparator



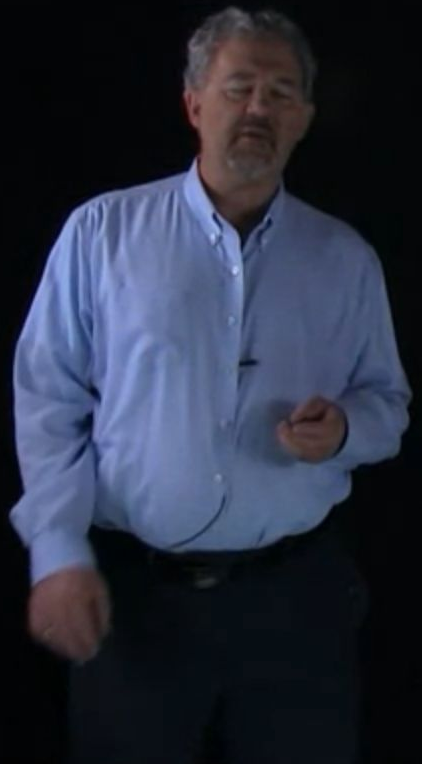
```
module COMPARATOR
    (input [3:0] A, B ,
    output Y);

    integer N;
    reg Y;

    always @(A or B)
     begin: COMPARE
       Y = 0;
       if (A == B)
          Y = 1;
     end
endmodule
```

So here the output is initialized to zero.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

Copyright © 2019 University of Colorado

Summary – Verilog Introduction



In this video, you have learned:

- **The history of the Verilog HDL Language from inception in 1984 until the latest IEEE revision in 2012**
- **An approach to learning Verilog, involving assimilation of vocabulary, phrases and syntax**
- **A "first phrase" design example (a comparator), modeled by use of structure, dataflow, or behavior.**

data flow or behavioral descriptions.

Copyright © 2019 University of Colorado

Be Boulder.

© Regents of the University of Colorado

University of Colorado **Boulder**

© Regents of the University of Colorado

Share

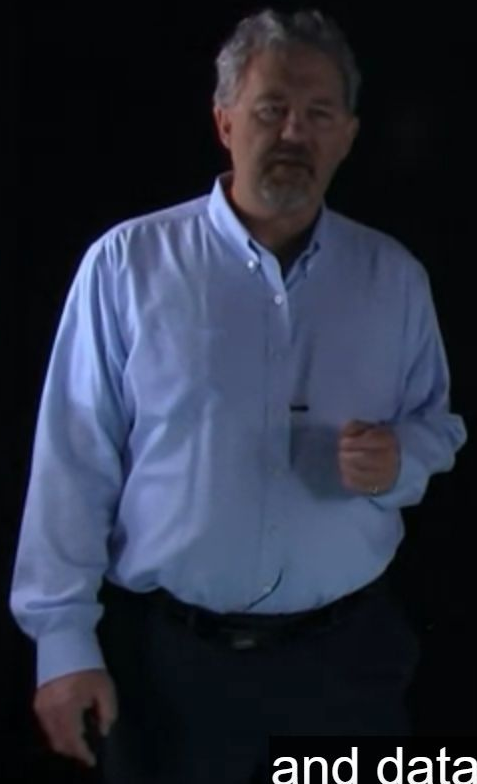
Verilog Syntax

Professor Tim Scherr

University of Colorado **Boulder**

0:01 / 12:27

Verilog Syntax



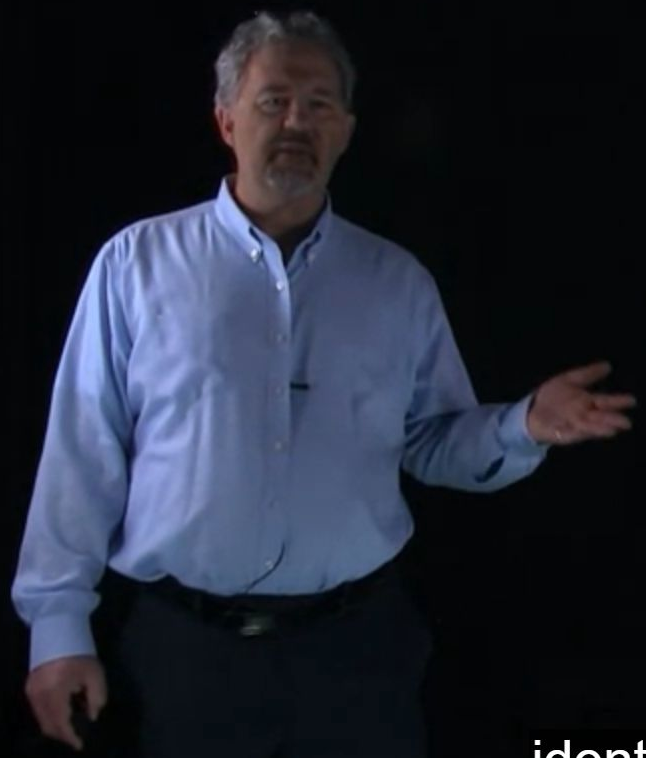
In this video, you will learn:

- The basics of Verilog Syntax
- Verilog variable values and data types
- Suggested editors that can be used to create Verilog code.

and data types, and some suggested editors that can be used to create Verilog code.

Share
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

Verilog Syntax



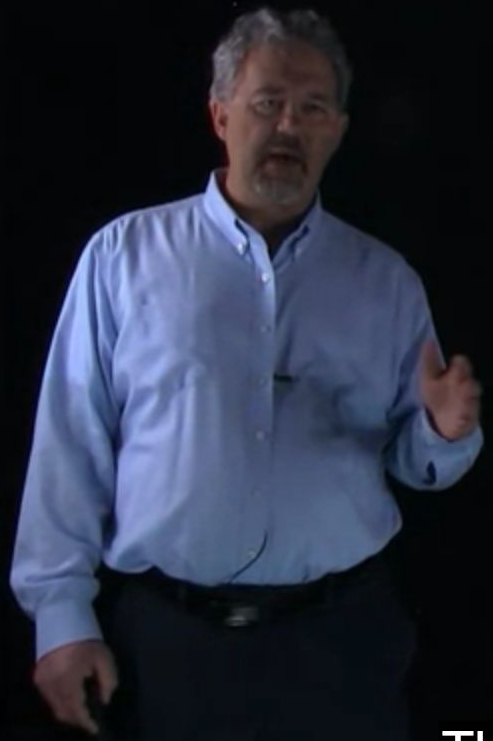
Many syntax rules are much like in C software:

- Whitespace is ignored
- Comments are either //... or /* ... */
- Identifiers are words for variables, function names, etc. and can begin with _ or a letter and can include letters, digits, and underscores and are CASE SENSITIVE.

identifiers are words for variables, function names, etc.

Electrical, Computer & Energy
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

Verilog Syntax

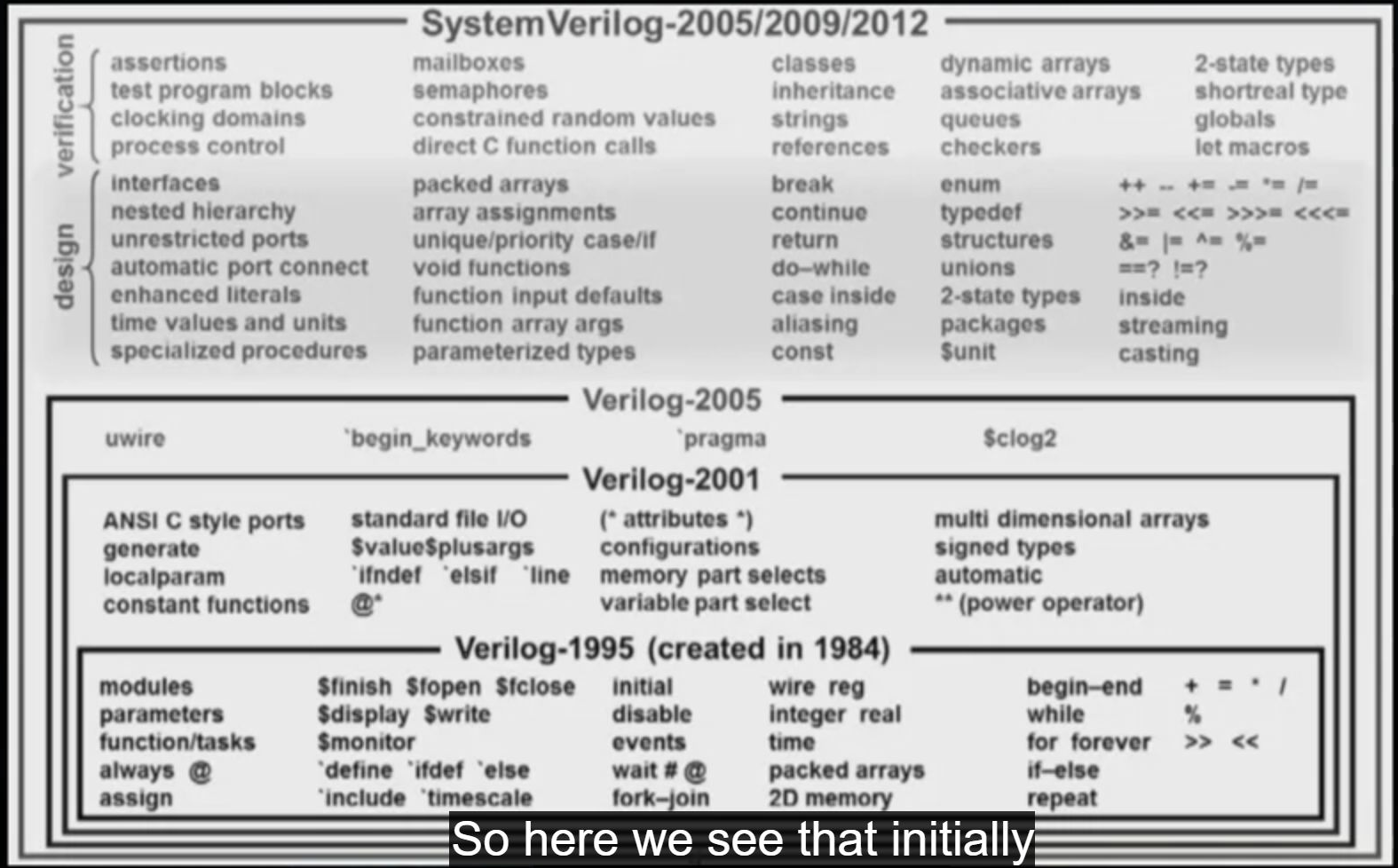


Many syntax rules are much like in C software:

- Keywords can't be used as identifiers, they are **assign**, **case**, **while**, **wire**, **reg**, **and**, **or**, **nand**, and **module** just to name a few.
- The number of keywords and capabilities have expanded as Verilog and then System Verilog was developed as shown on the next slide.

The number of keywords and capabilities have expanded as Verilog and

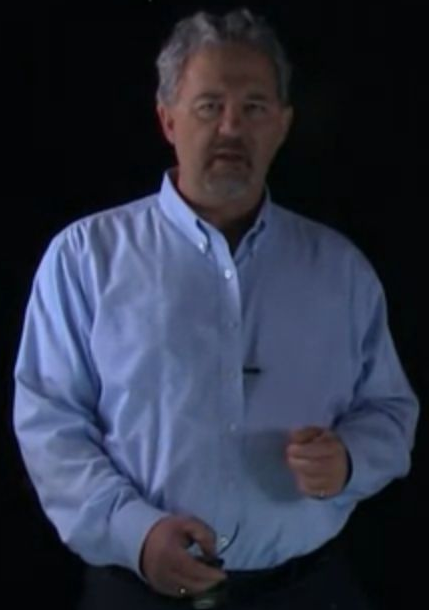
yright © 2019 University of Colorado



So here we see that initially Verilog 1995, this is

Copyright © 2019 University of Colorado

Verilog Syntax



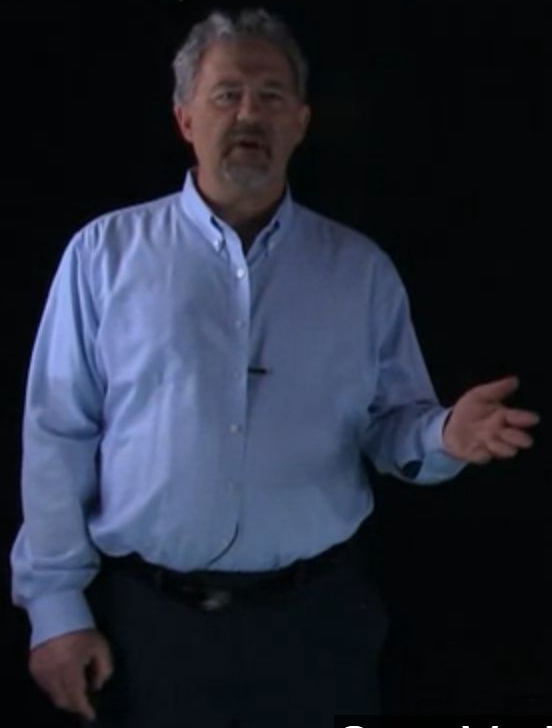
Many syntax rules are much like in C software:

- Number literals are expressed this way:
  3'b001, a 3-bit number,
  5'd30, (=5'b11110), and
  16'h5ED4, (=16'd24276).
- In general: size_in_bits'baseNumber
- Underscores can be added for improved readability as in 32'h1234_5678

helpful in trying to
make the them more clear.

Copyright © 2019 University of Colorado

Verilog Values

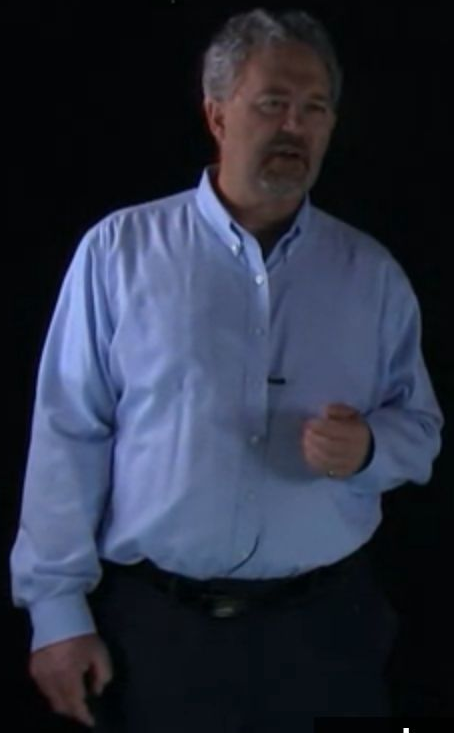


Verilog consists of only four basic values.
Almost all Verilog data types store all these values:
0 (logic zero, or false condition)
1 (logic one, or true condition)
x (unknown logic value or contention)
z (high impedance state, floating)
x and z have limited use for synthesis.

So a Verilog variable can assume any of four basic values so

Copyright © 2019 University of Colorado

Verilog Data Types



Significant Data types available in Verilog include:
wire,
reg,
integer,
real,
time,
parameter and
event.
SystemVerilog adds the type logic.
We will discuss these in more detail in future videos.

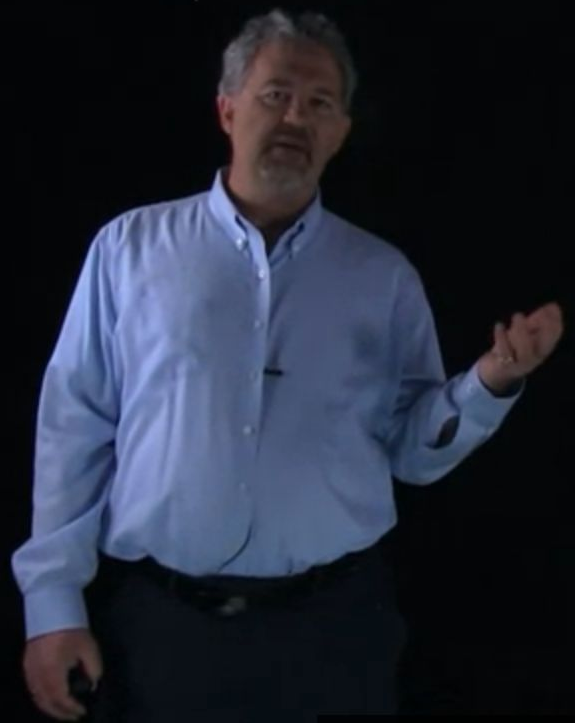
we just wanted to give you an idea
that there are these different kinds

Copyright © 2019 University of Colorado

Share

Press Esc to exit full screen

Text Editors for VHDL/Verilog



Desired Features: Keyword highlighting, auto indent, etc.

Free:

VI
EMACs with VHDL extensions
PSpad
Sigasi

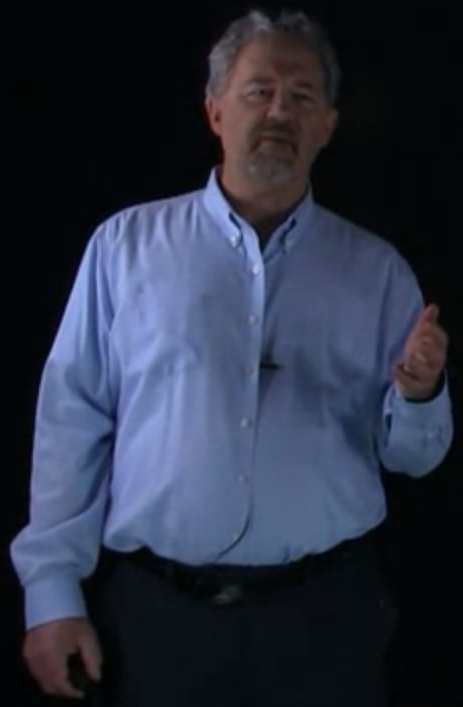
Commercial:

UltraEdit
Multi-Edit
Slick-Edit

So it's very helpful with VHDL, if you want to write VHDL EMAC's a good choice.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

An Introduction to Sigasi



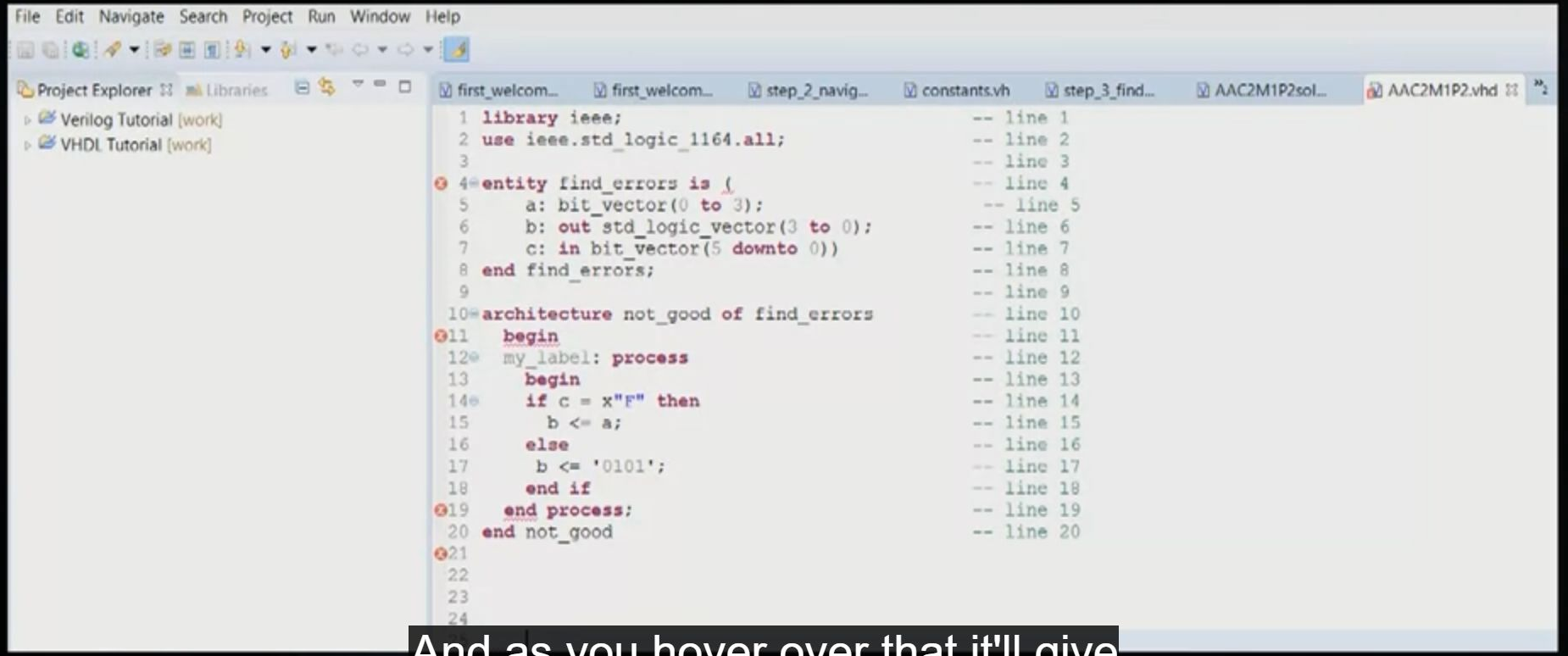
Sigasi Studio guides you through complex code designs. With instant feedback on errors and auto-completion suggestions, it aids you in reducing development time and helping you and your team write better VHDL, Verilog, and SystemVerilog code.

Download and try it from:
https://www.sigasi.com/edu-stud/

So it knows Verilog syntax and also
VHDL and System Verilog for that matter.

Copyright © 2019 University of Colorado

An Introduction to Sigasi



And as you hover over that it'll give you suggestions for fixing the error,

Copyright © 2019 University of Colorado

Summary – Verilog Introduction

In this video, you have learned:

- **The basics of Verilog Syntax**
- **Verilog variable values and data types**
- **A list of possible editors that can be used to create Verilog code, including Sigasi, which is capable of syntax correction on the fly.**

So in summary in this video, you have learned the basics of Verilog syntax.

Copyright © 2019 University of Colorado

Be Boulder.

© Regents of the University of Colorado

University of Colorado **Boulder**

© Regents of the University of Colorado

Share

Press Esc to exit full screen

Verilog Statements and Operators

Professor Tim Scherr

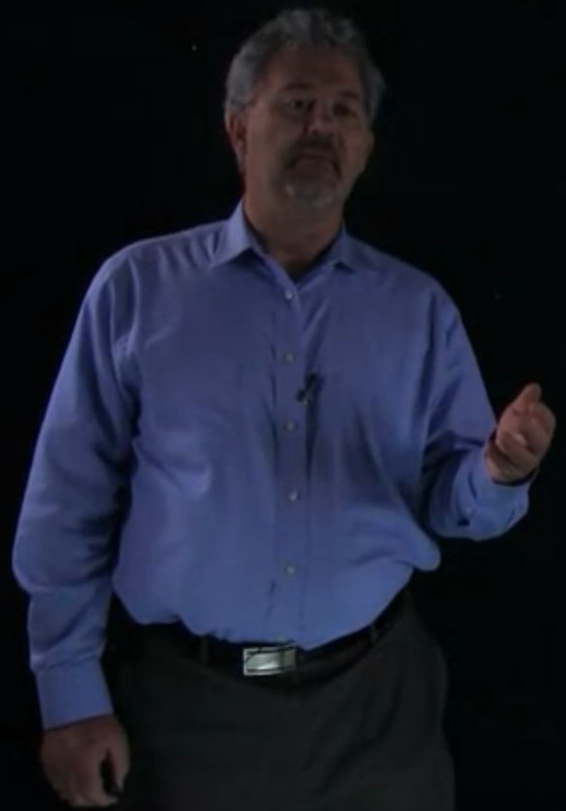
University of Colorado **Boulder**

0:01 / 16:15


Share
Verilog Fundamentals
In this video, you will learn:
• How to use assignment statements
• The wide range of Operators in Verilog and how they work
In this video, you will learn how to use the assignment statements and
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
0:11 / 16:15

Share
Press Esc to exit full screen

Verilog Fundamentals



Assignments

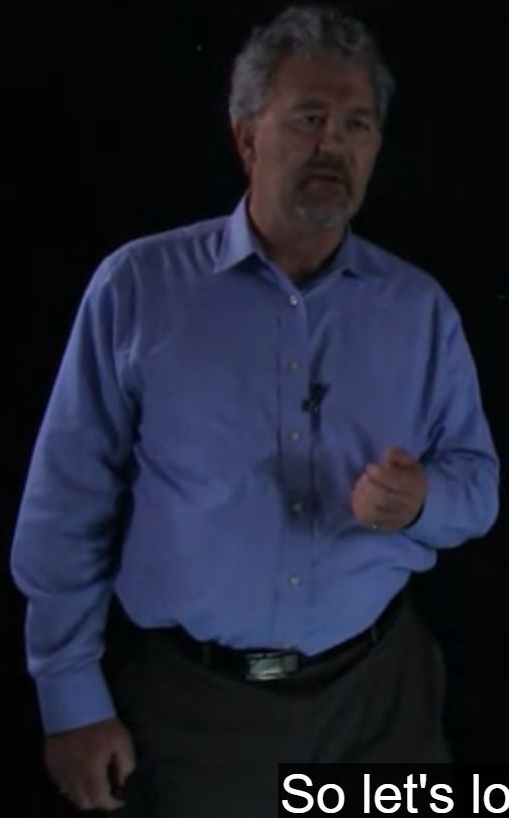
The fundamental statement in Verilog is the assignment statement. All assignment statements outside of an always block are concurrent – they happen at the same time, and are not sequential. The output of the operation on the right hand side of the = symbol is continuously assigned to the variable on the left hand side, as in

```
assign A = B & C;  // an and gate
```

The variable on the LHS must be a net, not a reg when outside of an always block.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
1:25 / 16:15

Verilog Fundamentals



Assignments - Continuous

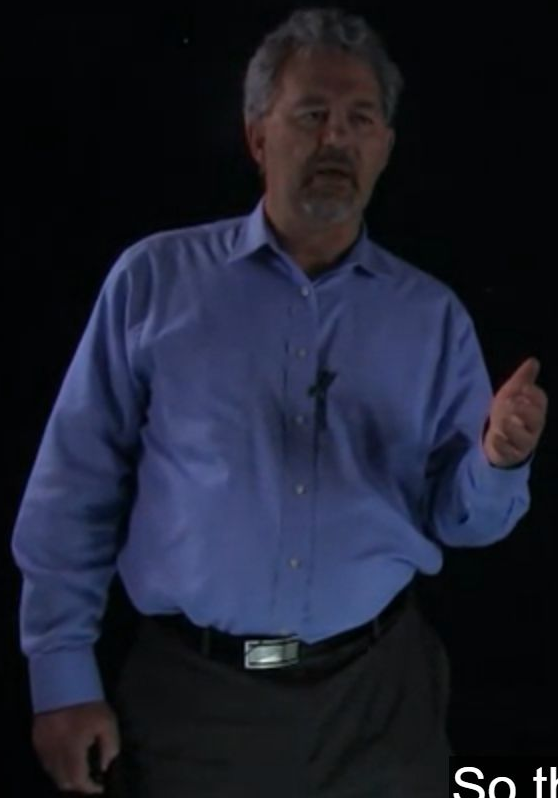
Assume we wanted to create this circuit:

How would this be coded in Verilog?

So let's look at an example where we're using continuous assignment statements.

Electrical, Computer &
UNIVERSITY OF COLORADO BOULDER
opyright © 2019 University of Colorado

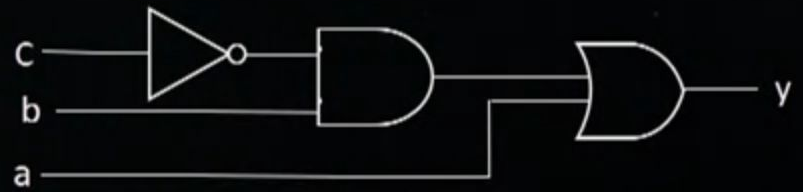
Verilog Fundamentals



Assignments - Continuous

Answer: assign y = (a | b) & ~c;

Note that if we leave out the parens, we get:



assign y = a | b & ~c;

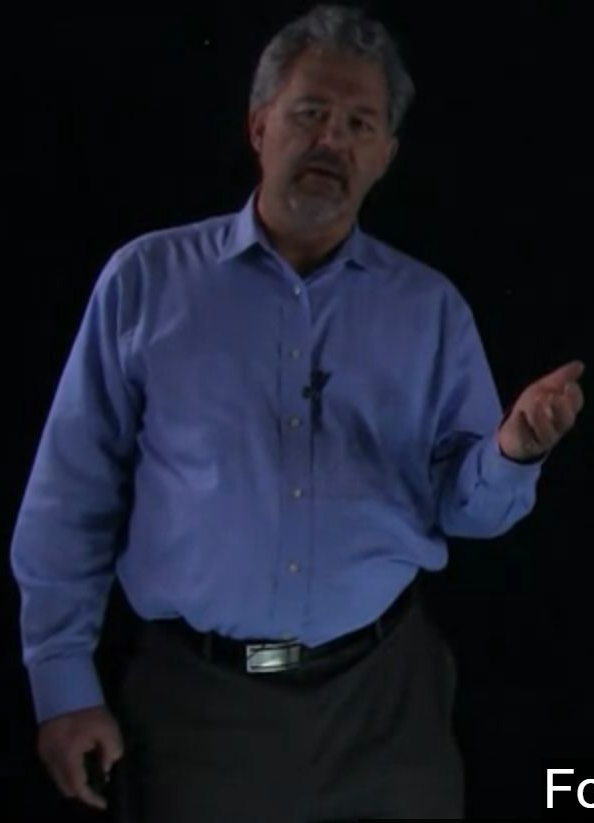
Because of the default order of evaluation.

So the answer then is to assign y equal to a or

Electrical, Computer & Energy Engin
UNIVERSITY OF COLORADO BOULDER

Copyright © 2019 University of Colorado

Verilog Fundamentals



Assignments – Procedural

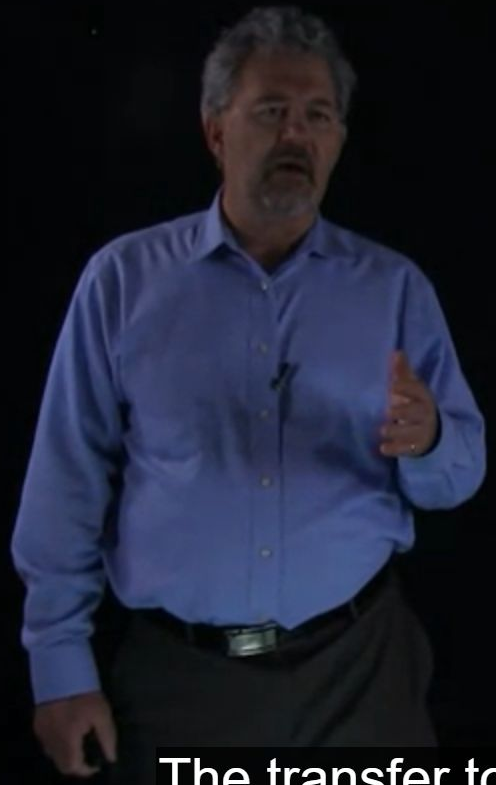
Procedural (blocking) assignments (=) are done sequentially in the order the statements are written. A second assignment is not started until the preceding one is complete. For example:

```
always @( posedge clk)
        begin
        Q2=Q1; Q1=D; // shift register
        Q1=D; Q2=Q1; //single or parallel ff.
        end
```

For example here,
we have an always block that has

Copyright © 2019 University of Colorado

Verilog Fundamentals



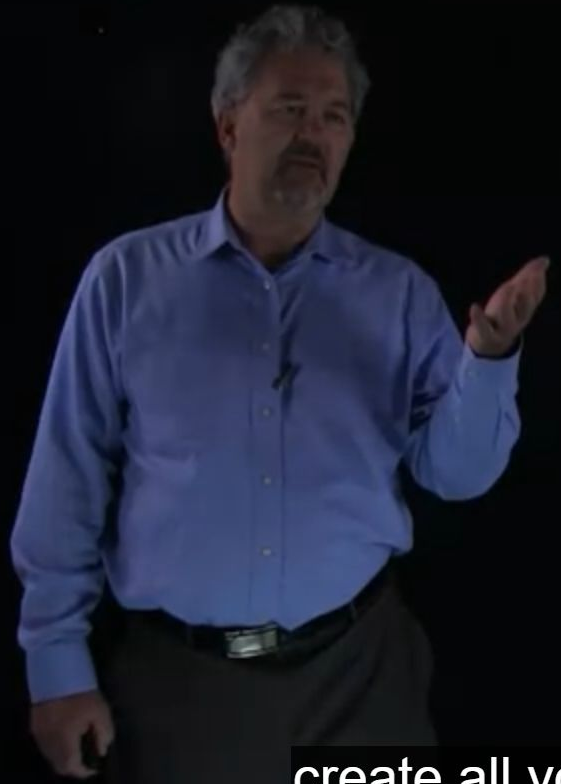
Assignments – nonblocking

RTL (nonblocking) assignments (<=), which follow each other in the code, are started in parallel. The right hand side of nonblocking assignments is evaluated starting from the completion of the last blocking assignment or if none, the start of the procedure. The transfer to the left hand side is made according to the delays. An intra-assignment delay in a nonblocking statement will not delay the start of any subsequent statement blocking or nonblocking.

The transfer to the left hand side is made according to delays that are defined.

Electrical, Computer & Energy
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
www.cs.upc.edu/.../secretsofhardware/VerilogIntroduction

Verilog Fundamentals



Assignments – blocking/nonblocking Guidelines

- One generally should not mix "<=" or "=" in the same procedure.
- "<=" best mimics what physical flip-flops do; use it only for "always @ (posedge clk ..) type procedures describing sequential circuits.
- "=" best corresponds to what c/c++ code would do; use it for combinational procedures, within or outside of always blocks.

create all your sequential logic using nonblocking statements.

Copyright © 2019 University of Colorado

Verilog Operators

Operators, by precedence	Description
[ ]	bit-select or part-select
( )	parenthesis, sets precedence order
!, ~	Logical and bit-wise NOT
&, |, ~&, ~|, ^, ~^, ^~	Reduction AND, OR, NAND, NOR, XOR, XNOR; &(4'b0101) = 0 & 1 & 0 & 1 = 1'b0
+, -	unary sign, plus, minus; i.e. +18, -8
{ }	Concatenation: {3'b101, 3'b110} creates 6'b101110;
{{ }}	Replication; {3{3'b110}} creates 9'b110110110
*, /, %	Multiply, divide, modulus; Note: / and % may not be supported for synthesis
+, -	[illegible]

You just have to put the two vectors inside of the curly braces and

Share
Electrical, Computer & Energy
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado
10:03 / 16:15

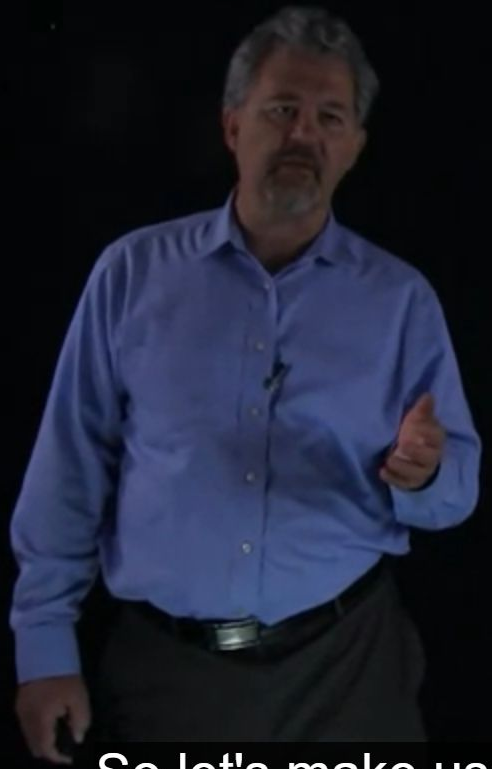
Verilog Operators

Operators, by precedence	Description (continued)
<<, >>	Shift left, shift right; X<<2 multiplies X by 4
<, <=, >, >=	comparison tests. Reg and wire variables are taken as positive numbers
==, !=	logical equality test, logical inequality test
===, !==	case equality, case inequality; not synthesizable
&	bit-wise AND; AND together all the bits in a word If X=3'b101 and Y=3'b110, then X&Y = 3'b100, X^Y = 3b011
^, ~^, ^~	bit-wise XOR, bit-wise XNOR
|	bit-wise OR;
&&	logical AND for use in decisions; Treats all variables as False(zero) or True(nonzero). (3&&0) is (T&&F) = 0
||	logical OR; (2||-3) is (T||T) = 1
?:	conditional, usually synthesizes to a mux. X = (cond)?T:F

So here's the rest of the Verilog operators in another chart.

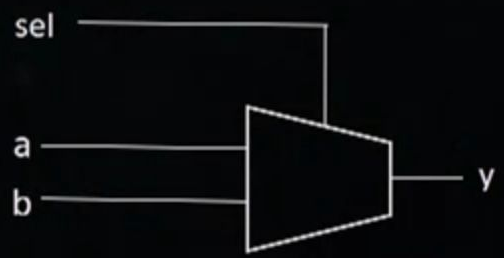
Share
Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

Verilog Fundamentals



How to make a Multiplexer

Assume we wanted to create this circuit:



How would this be coded in Verilog?

So let's make use of some of the operators
in order to describe some circuits.

Copyright © 2019 University of Colorado

Share

Press Esc to exit full screen

Verilog Fundamentals

How to make a Multiplexer

We could use a conditional assignment:

```
module mux(y, a, b, sel);
output y;
input a, b, sel;
assign y = sel ? a : b;
endmodule
```

The implementation would look something like this:

The implementation produced by the synthesizer will look a lot

Stephen A. Edwards, 2009 March, Verilog 1995, ... Available: www.cs.columbia.edu/~sedwards/classes/2004/

Copyright © 2019 University of Colorado

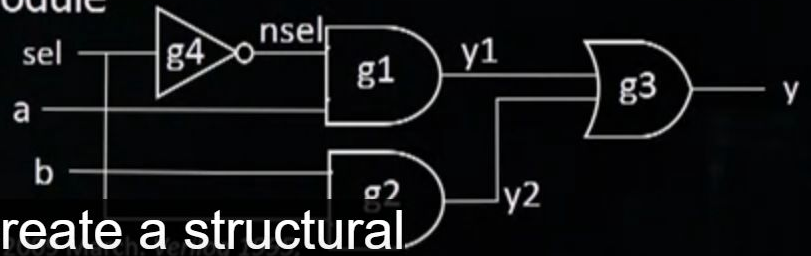
Electrical, Computer & Energy ... UNIVERSITY OF COLORADO BOULDER

13:10 / 16:15

Verilog Fundamentals

How to make a Multiplexer
We could use gate primitives:

```
module mux(y, a, b, sel);
output y;
input a, b, sel;
and g1(y1, a, nsel),
      g2(y2, b, sel);
or   g3(y, y1, y2);
not g4(nsel, sel);
endmodule
```



Knowing that, we could create a structural model by using gate primitives.

Share
Electrical, Computer & Energy
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

Verilog Fundamentals

How to make a Multiplexer

We could use an always combinational block:

```
module mux(y, a, b, sel);
output y;
input a, b, sel;
reg y;
always @(a or b or sel)
   if (sel) y = b;
   else y = a;
endmodule
```

We need to use it here because wires can't be assigned to

Electrical, Computer & Energy En
UNIVERSITY OF COLORADO BOULDER
Copyright © 2019 University of Colorado

Share

Press Esc to exit full screen

Verilog Fundamentals

How to make a Multiplexer

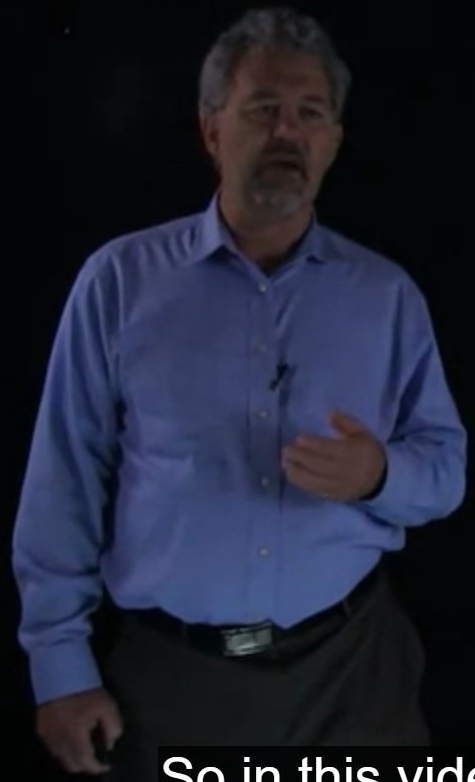
Lastly, we could use a User-Defined Primitive (UDP):

```
primitive mux(y, a, b, sel);
output y;
input a, b, sel;

table
   1?0 : 1;
   0?0 : 0;
   ?11 : 1;
   ?01 : 0;
   11? : 1;
   00? : 0;
endtable
endprimitive
```

This behavior is a little more optimistic and likely more robust.

Copyright © 2013 University of Colorado

Summary – Verilog Introduction



In this video, you have learned:

- **How to use assignment statements**
- **When and where to use blocking or nonblocking assignments**
- **The wide range of Operators in Verilog and how they work**
- **4 different ways to make a multiplexer**

So in this video, you have learned how to use assignment statements, when and

pyright © 2019 University of Colorado

Be Boulder.

© Regents of the University of Colorado

University of Colorado **Boulder**

© Regents of the University of Colorado