Learning to Speak Verilog

Professor Tim Scherr

University of Colorado **Boulder**

An Introduction to Verilog

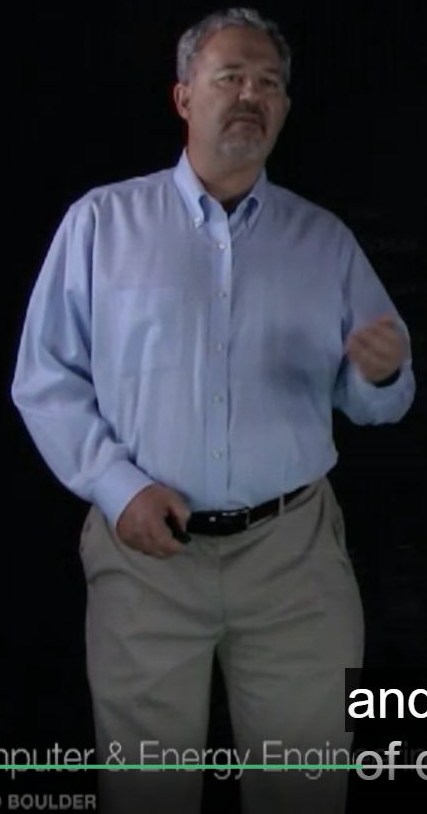
Verilog is one of the most powerful and efficient languages for logic design.

Now that the basics have been introduced, in this Module we will further explore how to use Verilog to design circuits by looking at many examples and techniques.

for logic design.

Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

Learning to Speak Verilog

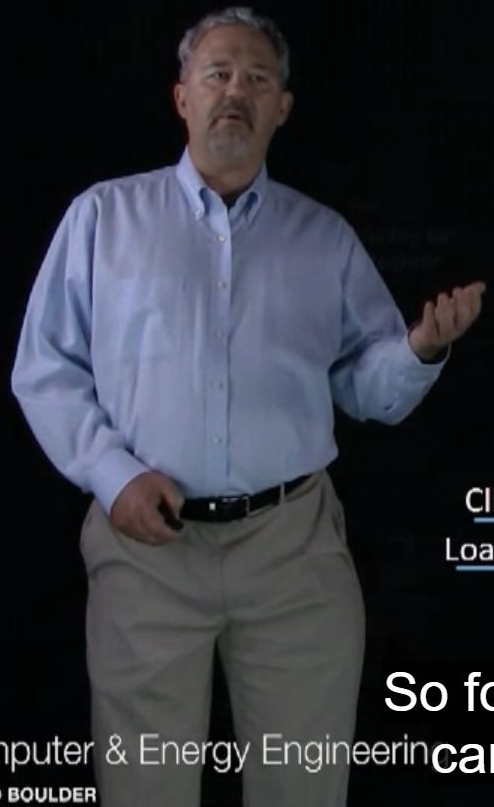


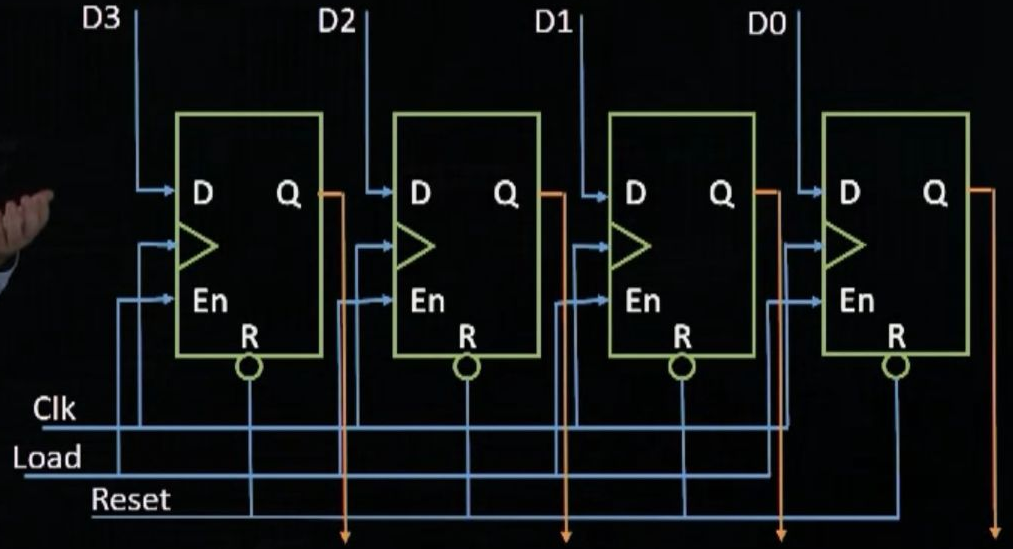
Like any other language…

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

and work through a lot of different examples.

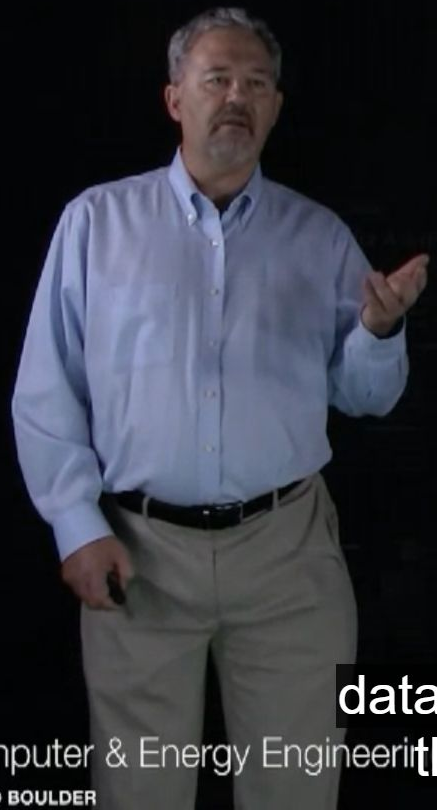
Writing Verilog for A 4-bit Register





So for example, we can practice by

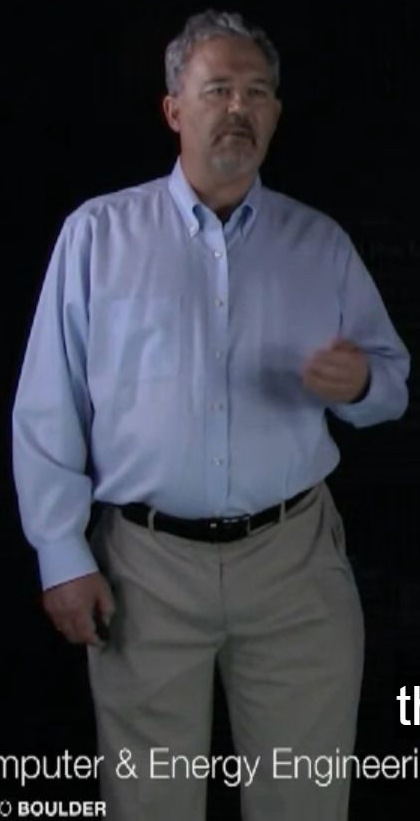
Writing Verilog for A 4-bit Register



```
// Data Register
//
module Dreg (
  input wire [3:0] d,
  input wire clk, reset, load,
  output reg [3:0] q
  );
  always @(posedge clk or negedge reset)
  begin
    if (!reset)  // asynchronous reset
      q <= 0;
    else if (load == 1)
      q <= d;
  end
endmodule
```

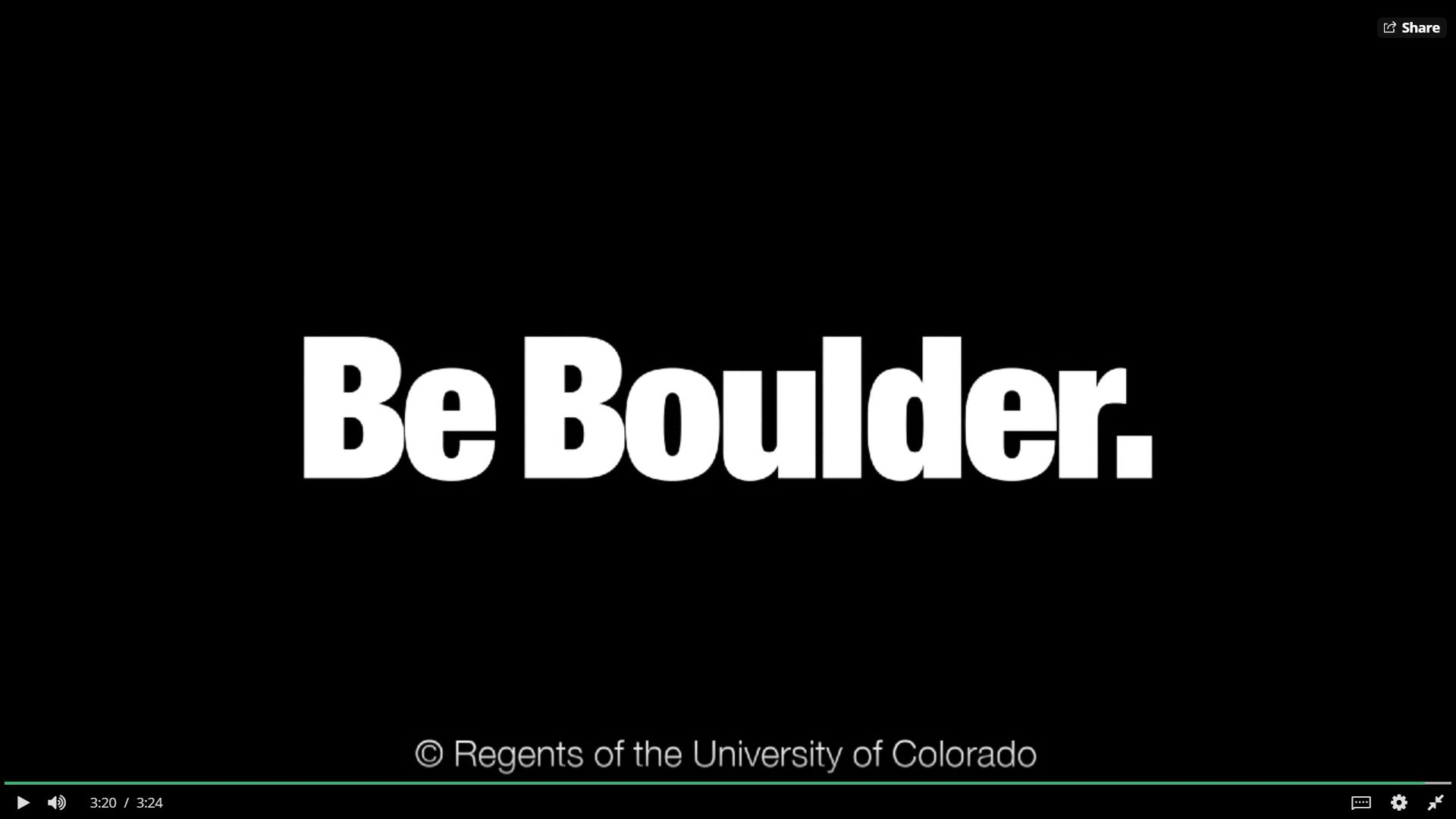
data register in Verilog,
there are others.

Videos in this Module



1. Learning to Speak Verilog
2. Combinatorial Circuits
3. Synchronous Logic: Latches and Flip Flops
4. Synchronous Logic: Counters and Registers
5. Interface: Buses and Tri-state Buffers
6. Modular Design in Verilog
7. Test Benches in Verilog - Combinatorial
8. Test Benches in Verilog – Synchronous
9. Memories of Verilog
10. Finite State Machines in Verilog

the following topics
one per video.


Share
Be Boulder.
© Regents of the University of Colorado
3:20 / 3:24

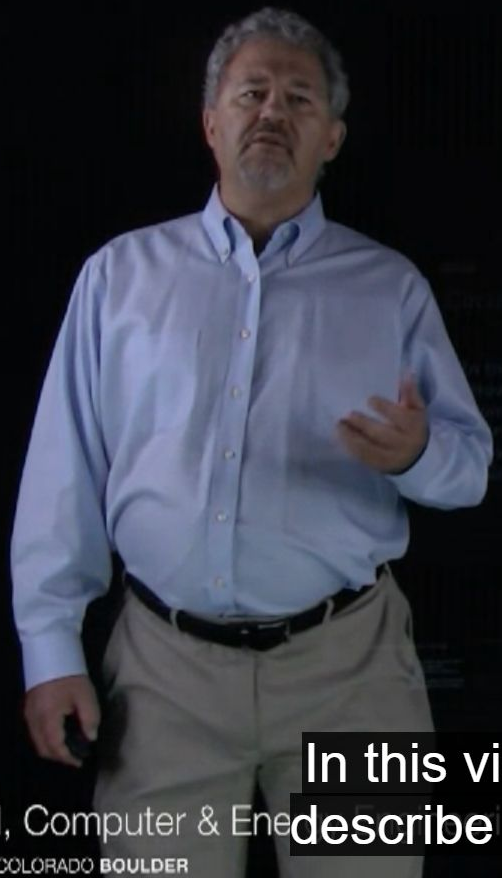
University of Colorado **Boulder**

Combinatorial Circuits

Professor Tim Scherr

University of Colorado **Boulder**

Combinatorial Circuits in Verilog



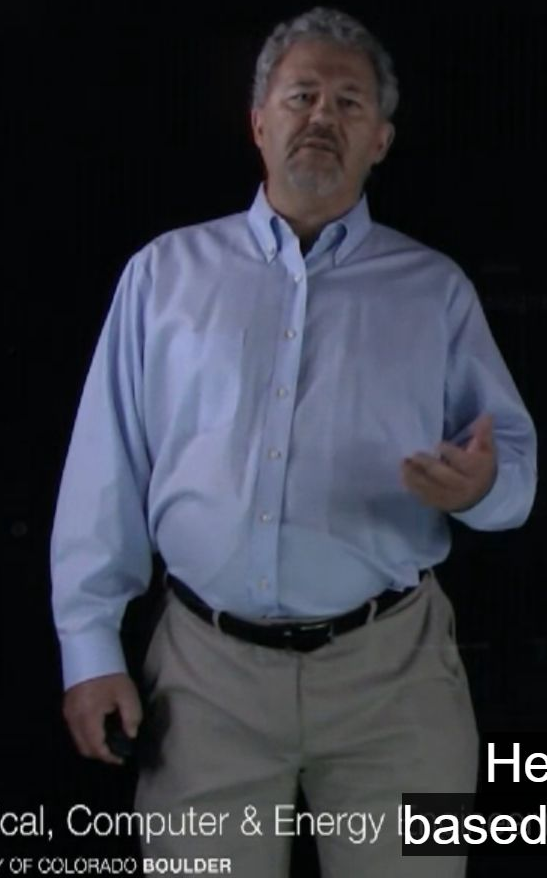
In this video, you will learn:

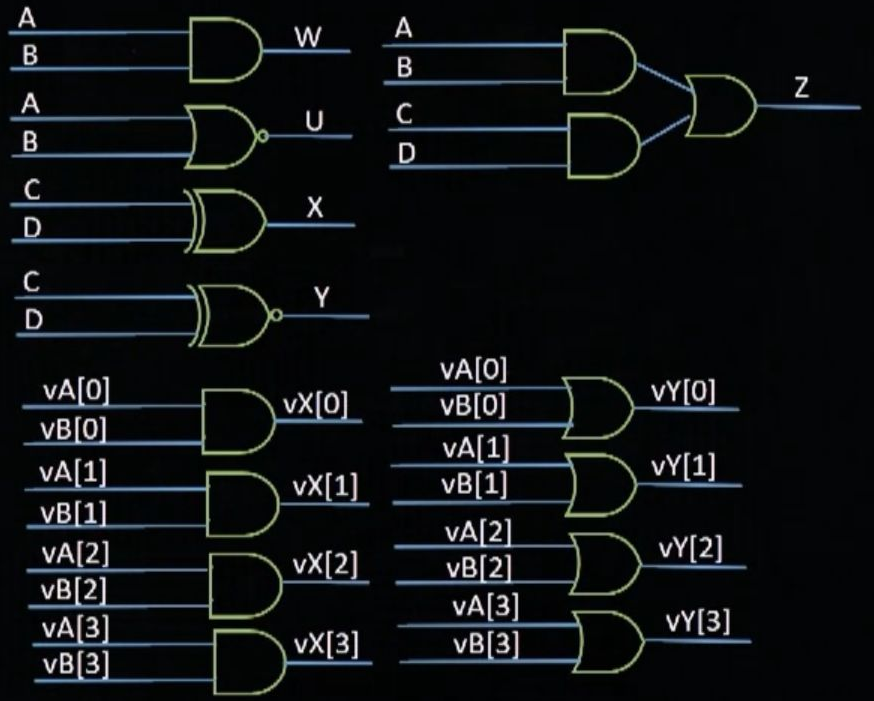
- **How to describe combinatorial circuits in Verilog**
- **How to reduce vector sizes using reduction operators**

In this video, you will learn how to describe combinatorial circuits and

Electrical, Computer & Ene
UNIVERSITY OF COLORADO BOULDER

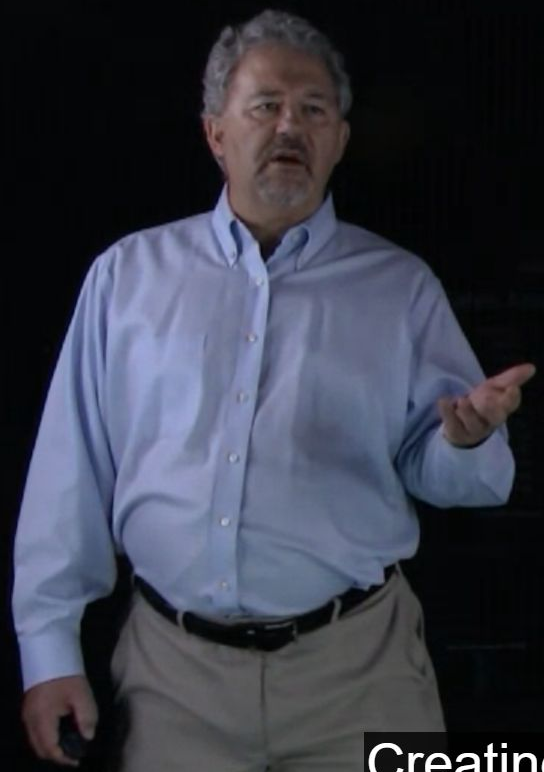
Basic Gates Assigned in Verilog





Here's a set of circuits based on simple Gates here.

Basic Gates Assigned in Verilog



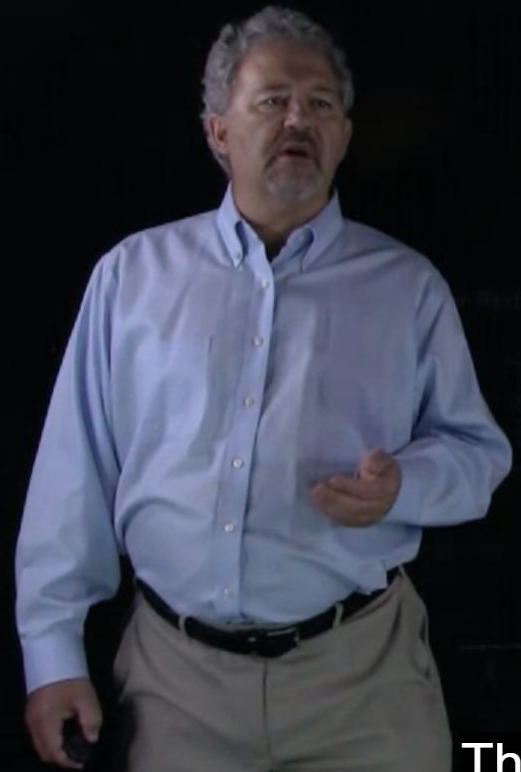
```
module gates (          // module and name
  input A, B, C, D,
  input [3:0] vA, vB,
  output W, U, X, Y, Z,
  output [3:0] vX, vY);

  assign W = A & B;     // scalar AND Gate
  assign U = ~(A | B); // scalar NOR Gate
  assign X = C ^ D; //scalar XOR Gate
  assign Y = C ~^ D; //scalar XNOR Gate
  assign Z = (A & B) | (C & D); // AND-OR gates
  assign vX = vA & vB; // Vector bitwise AND
  assign vY = vA | vB; // Vector bitwise OR

endmodule
```

Creating basic Gates and verilog is easy using assignment statements

Share

Press Esc to exit full screen

Vector Reduction in Verilog



```
module gates (          // module and name
  input [3:0] vA, vB, vC, vD,
  output W, U, X, Y, Z);
  assign W = & vA; // Vector Reduction AND Gate
  assign U = ~| vB; // Vector reduction NOR Gate
  assign X = ^ vD; // Vector reduction XOR Gate
  assign Y = | vA & vB ; // bitwise or reduction?
  assign Z = | (vA & vB); // bitwise AND,
                          // reduction OR

endmodule
```

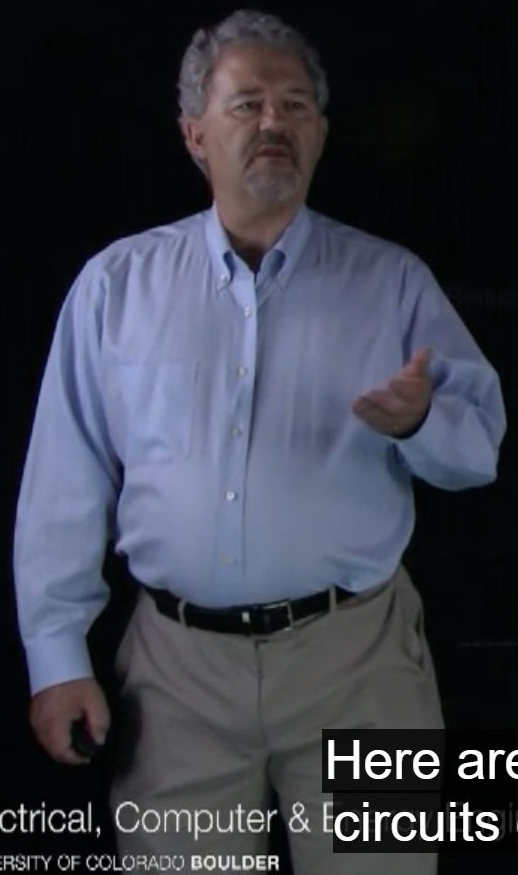
They can also be combined in a reduction operation

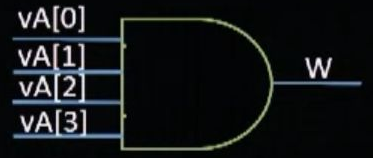
Electrical, Computer & Energy Engineering

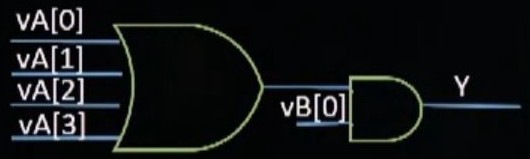
UNIVERSITY OF COLORADO BOULDER

1:40 / 5:28

Vector Reduction Gates in Verilog

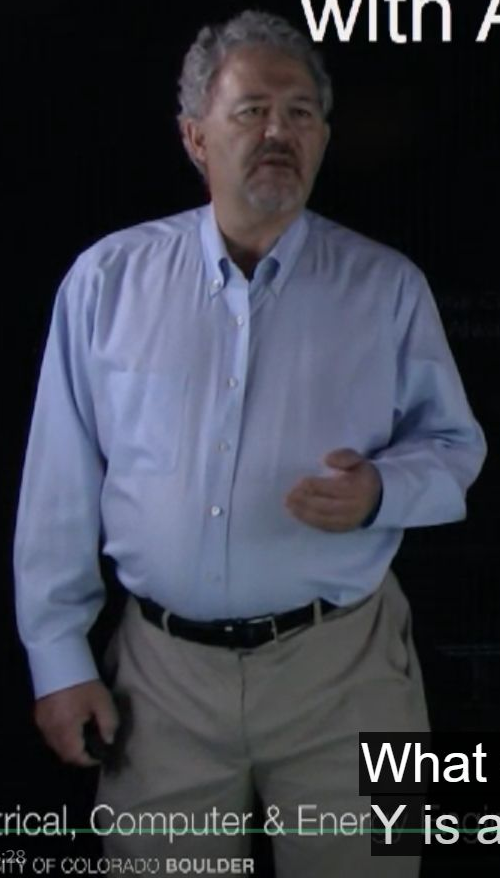






Here are the gate diagrams for the gate circuits described in the previous slide.

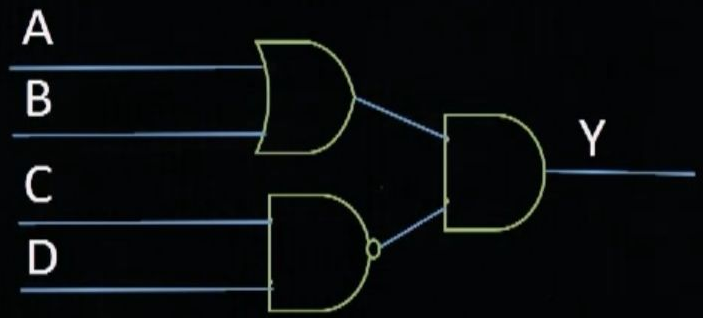
Procedural Combinational Logic with Always Procedures



```
module always_combo (
   input A, B, C, D,
   output reg Y);

   always @(A or B or C or D)
   begin
     if ((C==1) && (D==1))
       Y <= 0;
     else if ((A==1) || (B==1))
       Y <= 1;
     else
       Y <= 0;
   end
endmodule
```

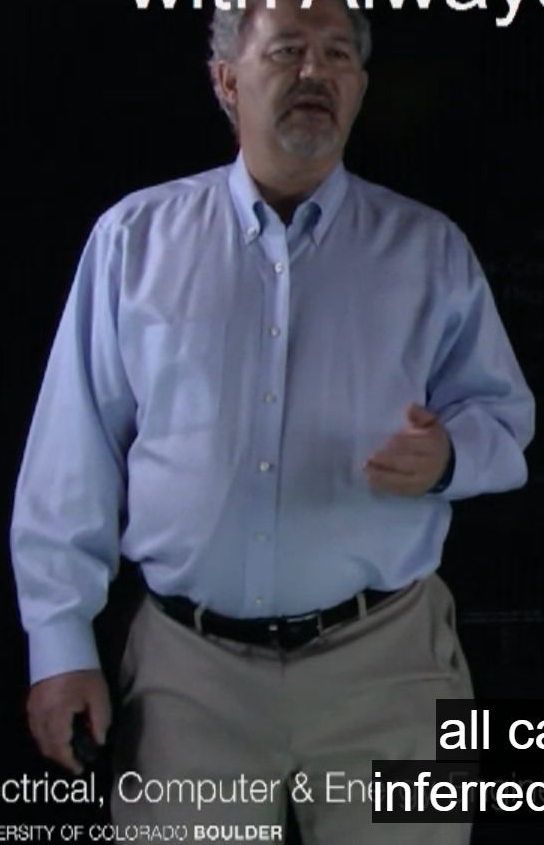
What circuit will this produce since Y is a rag does this mean that we

Procedural Combinational Logic with Always Procedures - Result



Here's the Circuit.

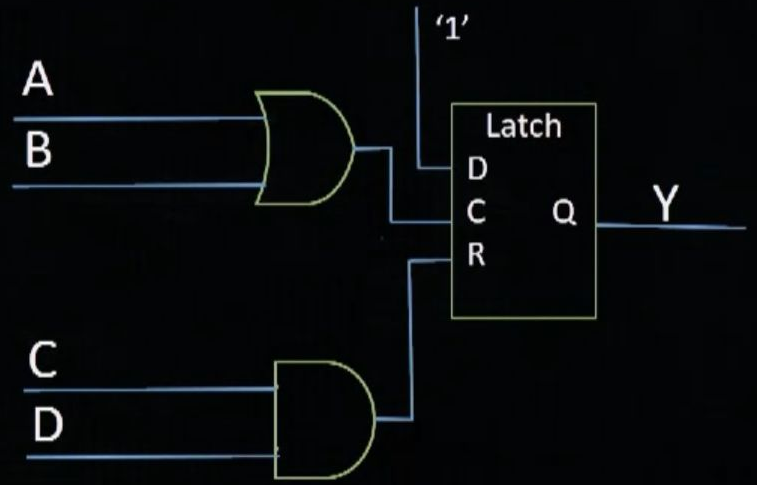
Procedural Combinational Logic with Always Procedures – Gotcha!



```
module always_combo (
   input A, B, C, D,
   output reg Y);

   always @(A or B or C or D)
   begin
     if ((C==1) && (D==1))
       Y <= 0;
     else if ((A==1) || (B==1))
       Y <= 1;
    end
endmodule
```

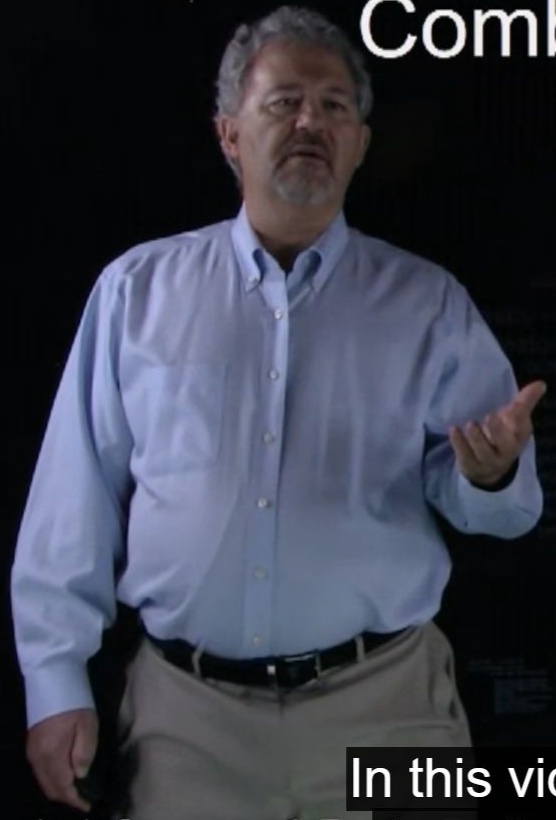
all cases, then latches may be inferred this is because if the output

Procedural Combinational Logic with Incomplete Assignment - Result



So here's the resulting circuit
a latch has been inferred now

Summary – Verilog for Combinatorial Circuits



In this video, you have learned:

- **How to describe combinatorial circuits in Verilog using either assignment statements or always procedures**
- **How to reduce vector sizes using reduction operators**
- **How to prevent unintentional latches in our logic circuits**

In this video you have learned how to describe combinational circuits and

References

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

Be Boulder.

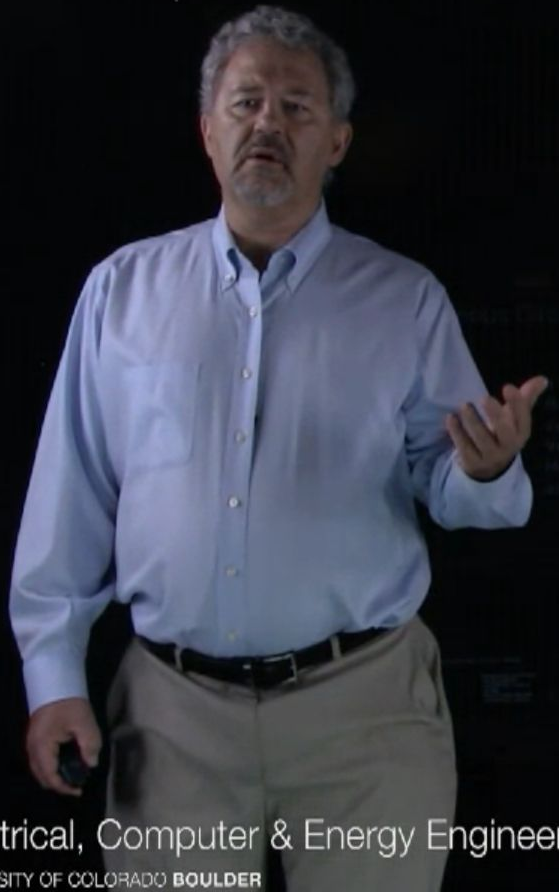
University of Colorado **Boulder**

Synchronous Logic: Latches and Flip Flops

Professor Tim Scherr

University of Colorado **Boulder**

Synchronous Circuits in Verilog

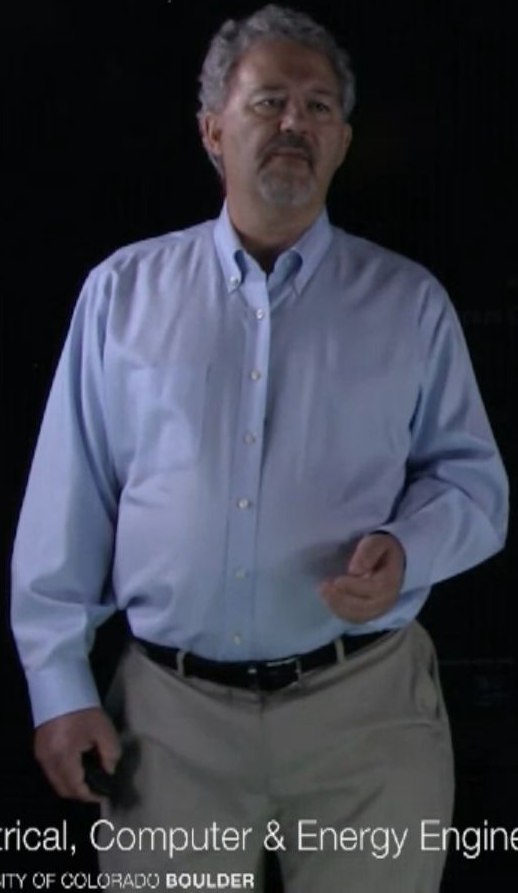


In this video, you will learn:

- **How to describe synchronous circuits in Verilog**
- **How to design flip-flops and latches in Verilog**

In this video, you will learn how to

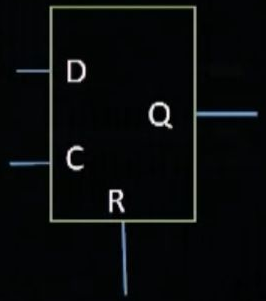
Synchronous Circuits in Verilog



```
// D flip flop
//
module DFF (
   input D, Ck,
   output reg Q
);
   always @(posedge Ck)
      Q <= D;
endmodule
```

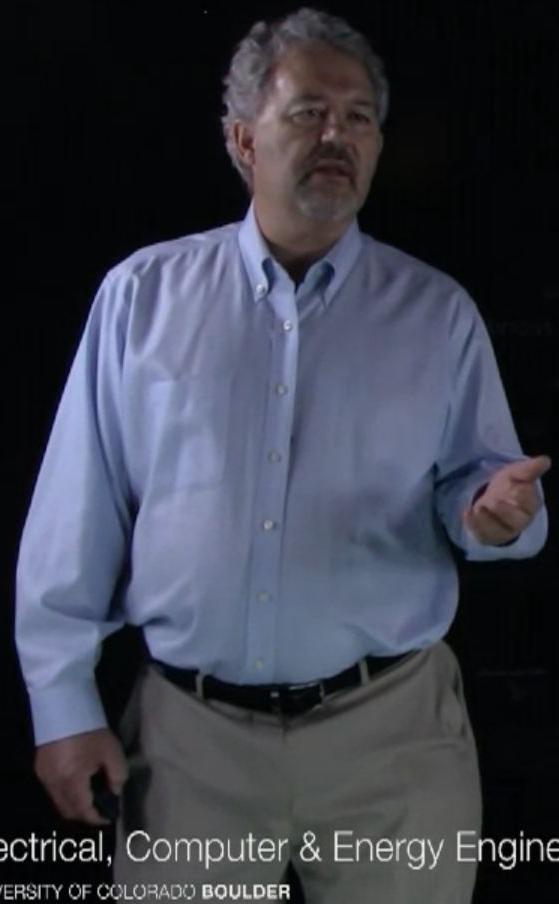
Synchronous Logic: D Latch

How can we make D Latches in Verilog?



Even more fundamental than a D flip-flop is the D latch.

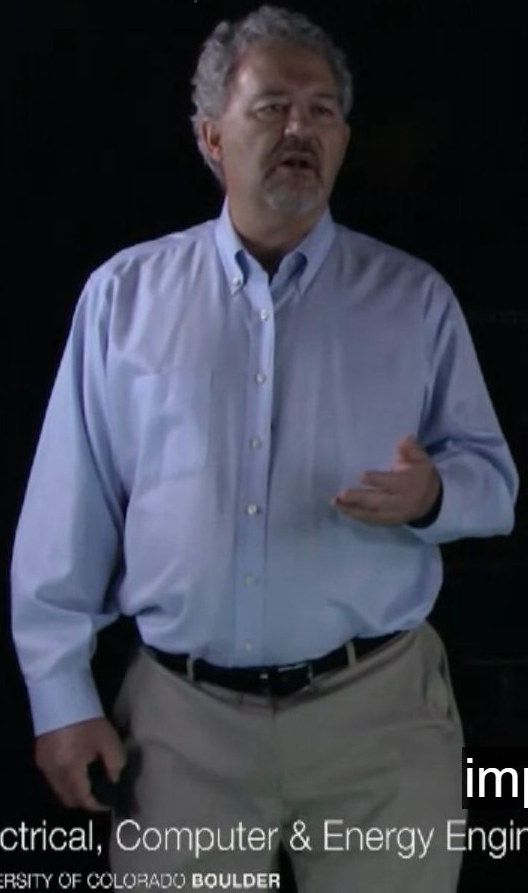
Synchronous Logic: D Latch



```
// D latches
//
module DLatches ( d, clk, aclr, qldc, qld);
   input d, clk, aclr;
   output reg qldc, qld;
   always @(clk or d)   // no posedge!
   begin
      if (clk == 1) qld <= d;
   end
```

with and without an asynchronous clear.

Synchronous Logic: D Latch

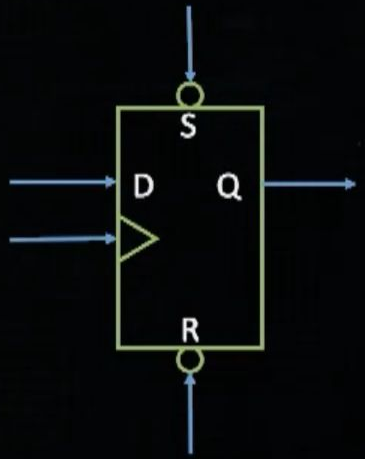


```
// D Latches
//
module DLatches ( d, clk, aclr, qldc, qld);
   input d, clk, aclr;
   output reg qldc, qld;
   always @(clk or d)   // no posedge!
   begin
      if (clk == 1) qld <= d;
   end
   always @(clk or d or aclr)
// note d is on the sensitivity list
   begin
      if (aclr == 1) qldc <= 0;
      else if (clk == 1) qldc <= d;
   end
endmodule
```

imply the logical or function,

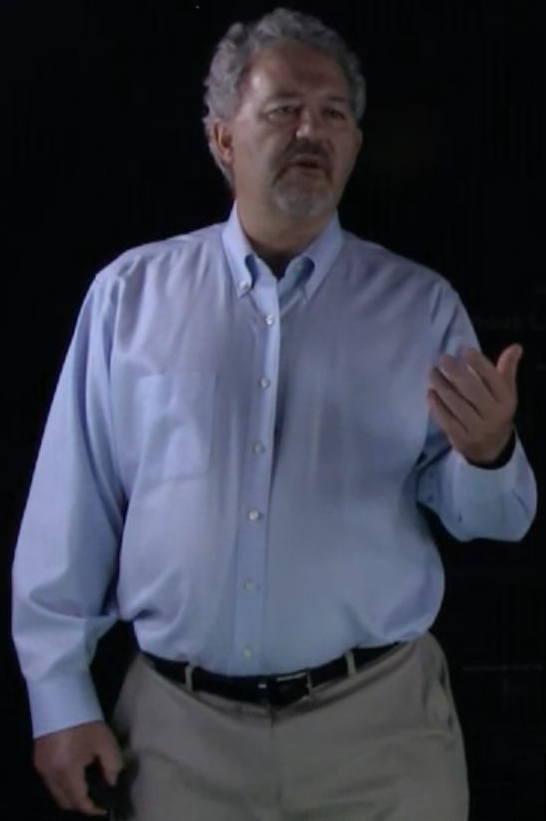
Synchronous Logic: D Flip Flops

How can we make D Flip-flops in Verilog, including those with asynchronous or synchronous reset?



So how can we describe D flip-flops then in Verilog,

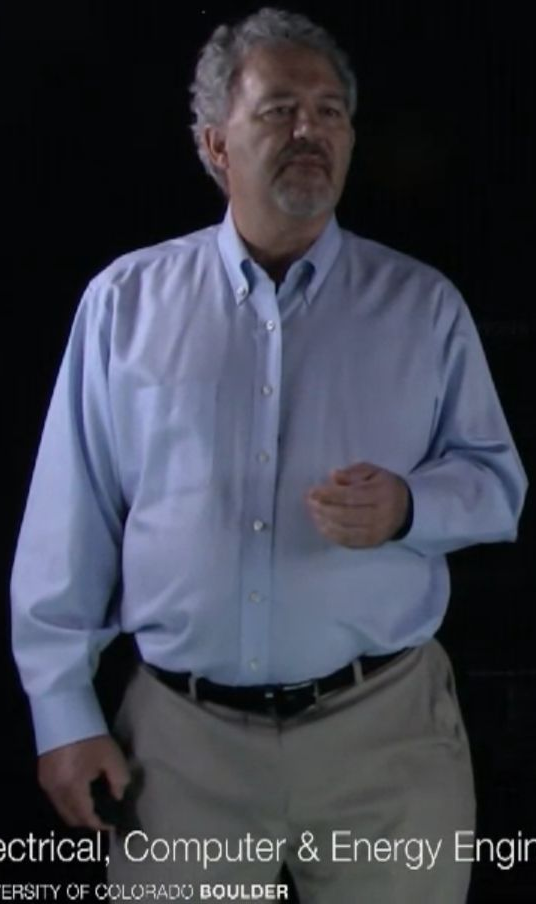
Synchronous Logic: D Flip Flops



```
// D Flip Flops
//
module DFF (d, clk, clr, reset, qld, qlda,
qlds);
   input d, clk, clr, reset;
   output reg qld, qlda, qlds;
   always @(posedge clk)   // with posedge!
   begin
    qld <= d;  //standard FF
    if (reset == 0) qlds <= 0;
    else qlds <= d;  // FF with sync reset
   end
```

Here's the code for D flip flop with either a

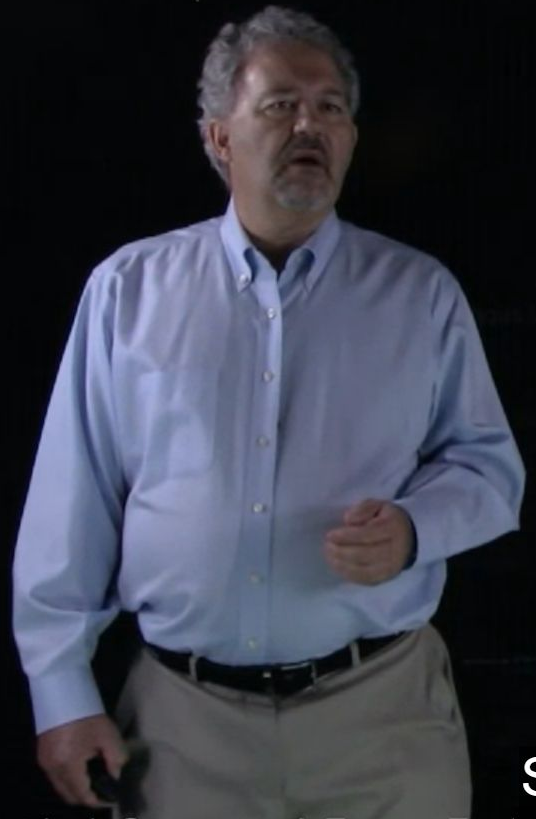
Synchronous Logic: D Flip Flops



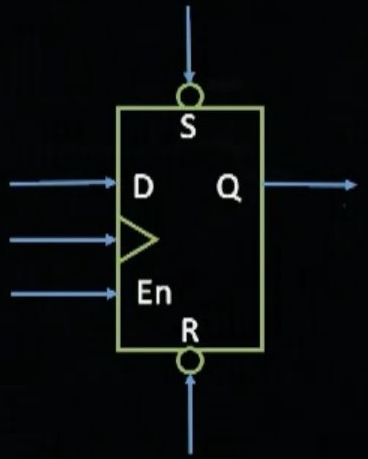
```
// D Flip Flops
//
module DFF (d, clk, clr, reset, qld, qlda,
qlds);
   input d, clk, clr, reset;
   output reg qld, qlda, qlds;
   always @(posedge clk)   // with posedge!
   begin
    qld <= d;  //standard FF
    if (reset == 0) qlds <= 0;
    else qlds <= d;  // FF with sync reset
   end
   always @(posedge clk or negedge clr)
   begin
      if (clr == 0) qlda <= 0;
      else qlda <= d; // FF with async reset
   end
endmodule
```

synchronous or an asynchronous clear.

Synchronous Logic: D Flip Flops

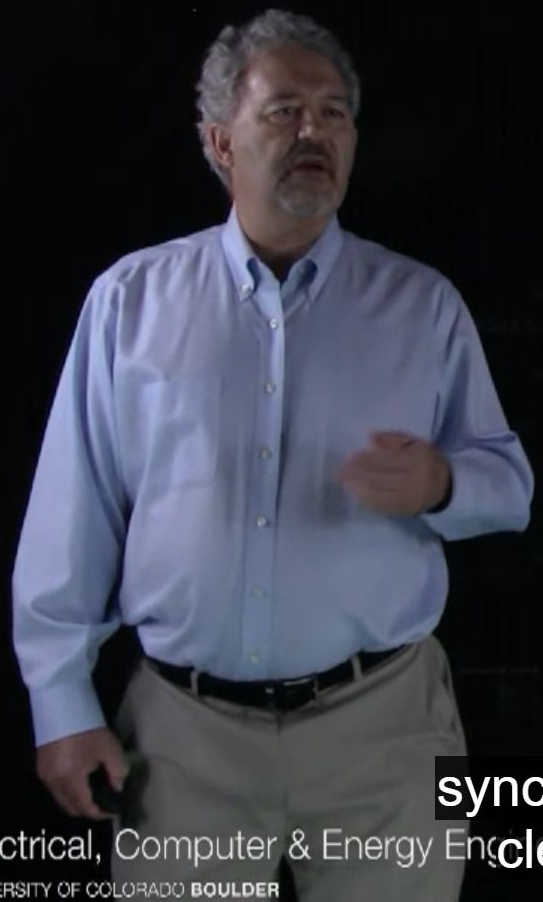


How can we make Clock Enable in Verilog?



So how can we make a clock enable in Verilog?

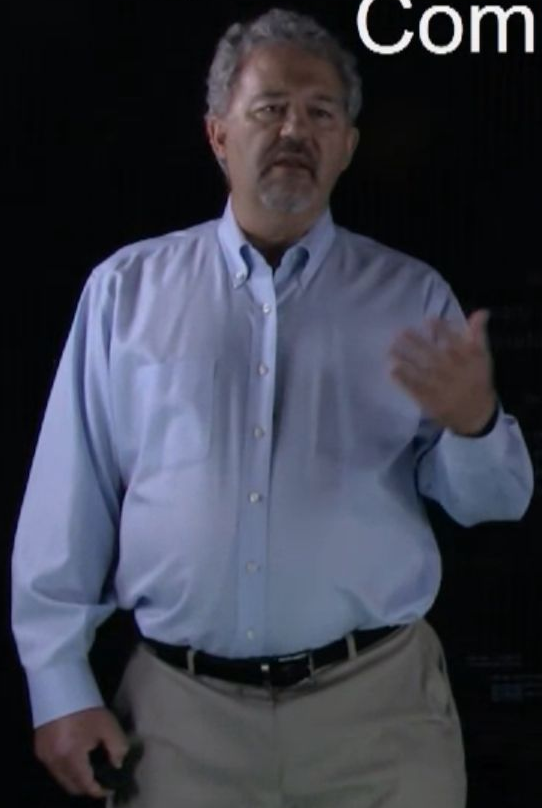
Synchronous Logic: D Flip Flops



```
// D Flip Flops with clock enable
//
module DFFe (d, clk, ce, clr, reset, qlda,
qlds);
   input d, clk, ce, clr, reset;
   output reg qld, qlda, qlds;
   always @(posedge clk)   //
   begin // FF with sync reset & clk enable
    if (reset == 1) qlds <= 0;
    else if (ce == 1) qlds <= d;
   end
   always @(posedge clk or posedge clr)
   begin // FF with async reset & clk enable
      if (clr == 1) qlda <= 0;
      else if (ce == 1) qlda <= d;
   end
```

synchronous or asynchronous clear and a clock enable.

Summary – Verilog for Combinatorial Circuits



In this video, you have learned:

- **How to describe synchronous circuits in Verilog**
- **How to design flip-flops and latches in Verilog**

In this video, you have learned how to

References

[1] V. Angelov. (2009). *Introduction to Verilog.* [Online]. Available: https://vdocuments.mx/amp/introduction-to-verilog-physikalisches-institut-angelovvhdlvhdlss09teil10pdf.html

Be Boulder.

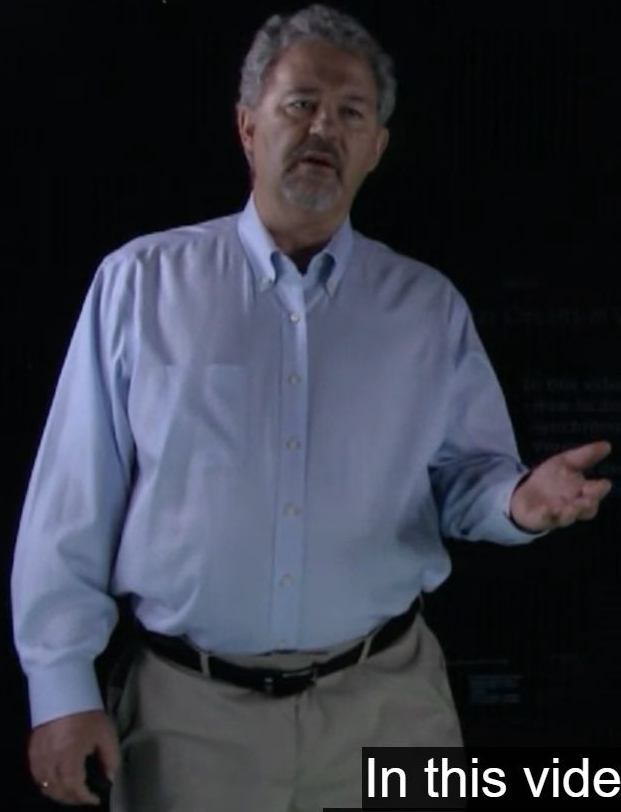
University of Colorado **Boulder**

Synchronous Logic: Counters and Registers

Professor Tim Scherr

University of Colorado **Boulder**

Synchronous Circuits in Verilog



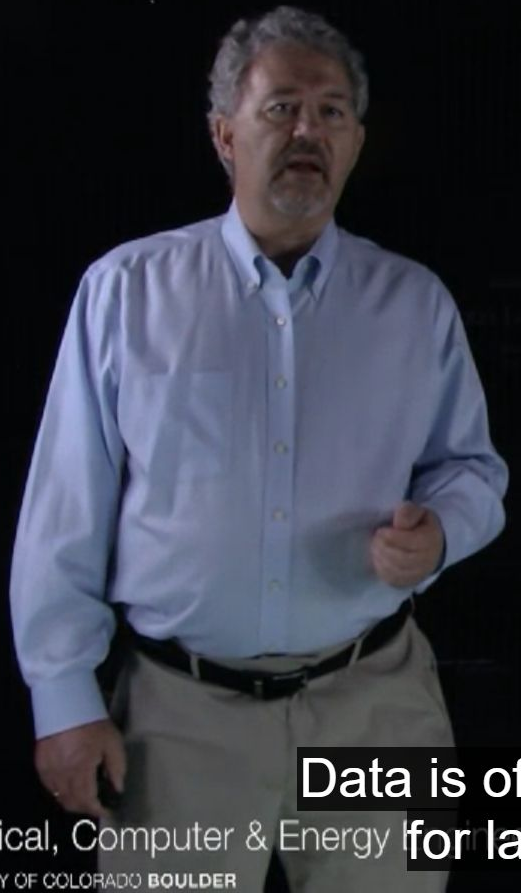
In this video, you will learn:

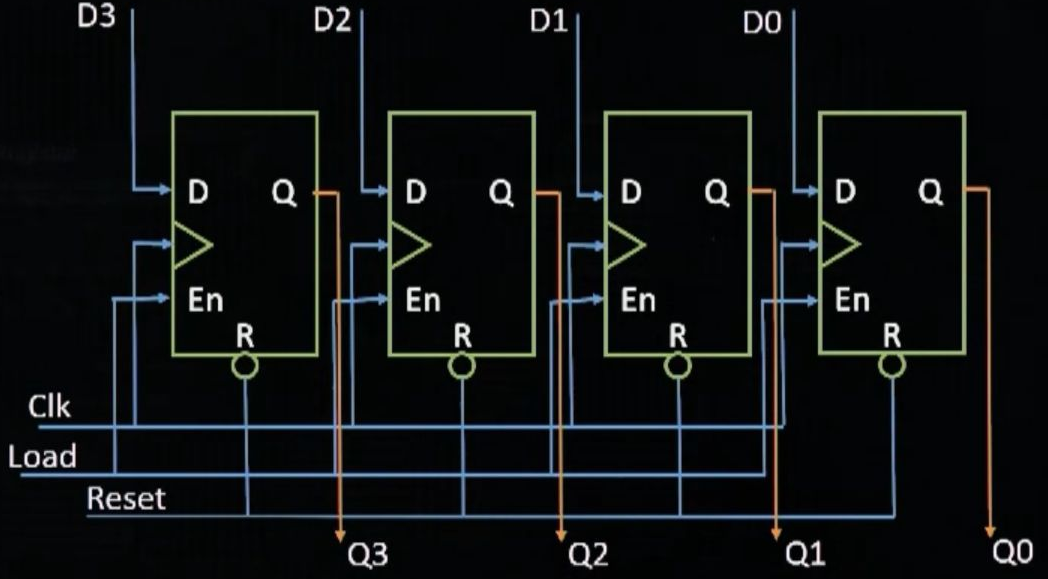
- **How to describe basic synchronous circuits in Verilog**
- **How to design registers and counters in Verilog**

In this video, you will learn how to describe basic sequence circuits and

Synchronous Logic: Register

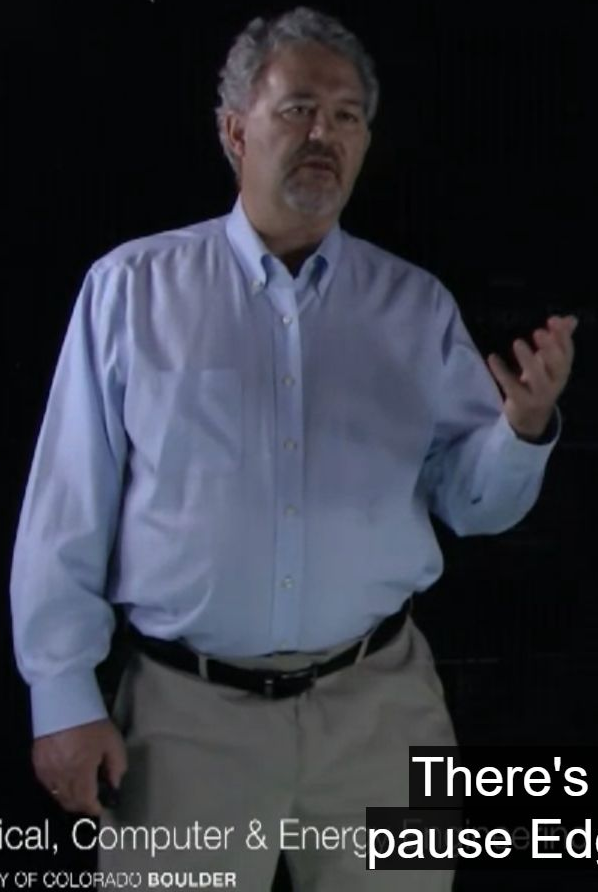
Data is oftentimes stored in a register for later use. How can we make a data register in Verilog?





Data is often times stored in a register for later use like shown here.

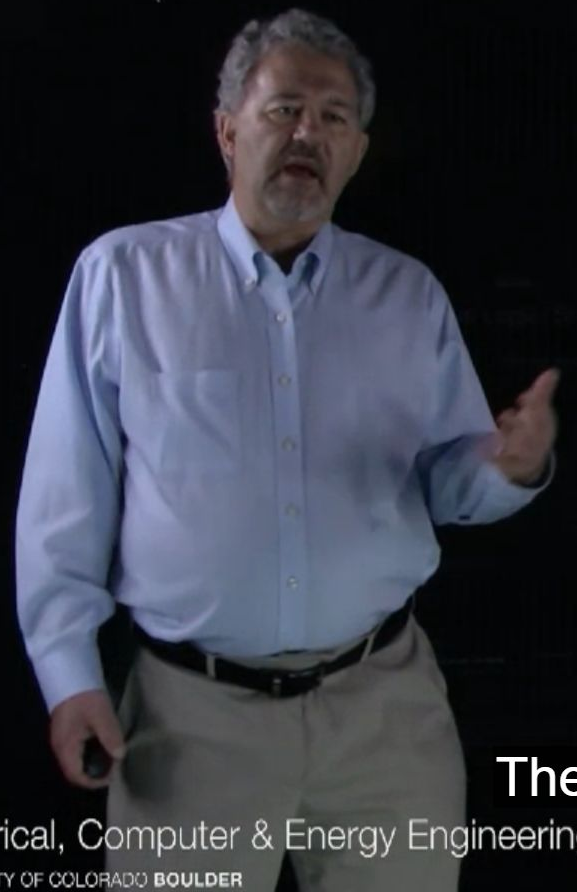
Synchronous Logic: Data Register



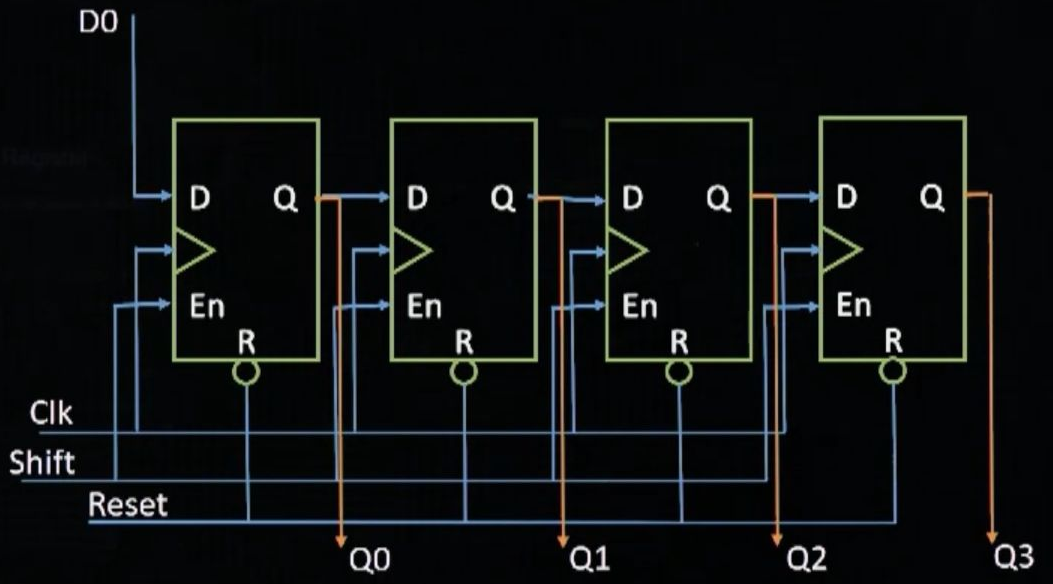
```
// Data Register
//
module Dreg (
  input wire [3:0] d,
  input wire clk, reset, load,
  output reg [3:0] q
  );
  always @(posedge clk or negedge reset)
  begin
    if (!reset)  // asynchronous reset
      q <= 0;
    else if (load == 1)
      q <= d;
  end
endmodule
```

There's an always block with
pause Edge clock negated reset

Synchronous Logic: Shift Register

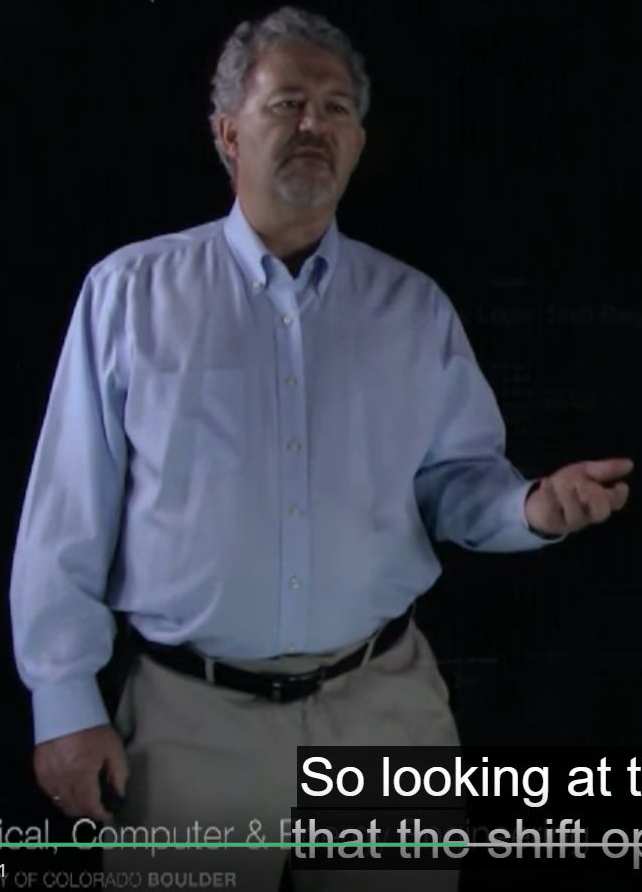


How can we make a shift register in Verilog?



Then it registers easy.

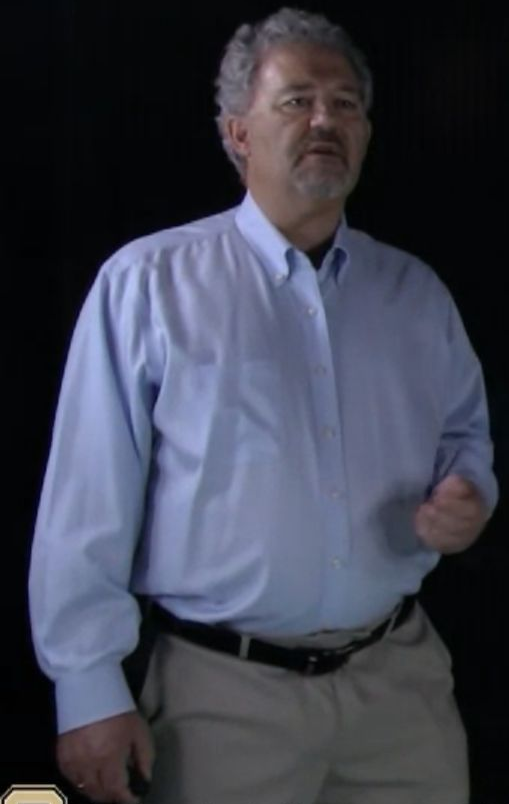
Synchronous Logic: Shift Register



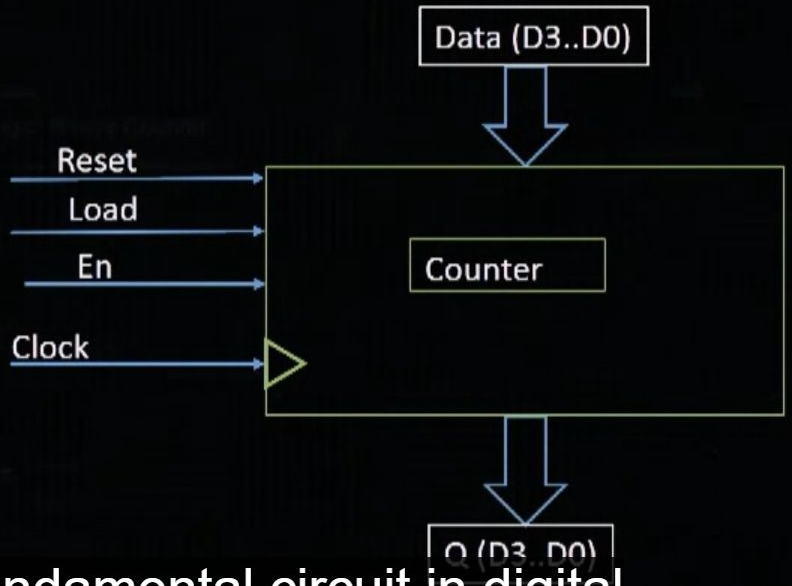
```
// Shift Register
//
module Shifter (clk, reset, D0, shift, Q);
  input clk, reset, D0, shift;
  output reg [3:0] Q;
  always @(posedge clk)   //
  begin
    if (!reset)
      Q <= 4'b0000;
    else if (shift == 1)
    begin
      Q <= Q << 1;
      Q[0] <= D0;
    end
    else ;
  end
```

So looking at the rest of the code note that the shift operator is used to create

Synchronous Logic: Binary Counter

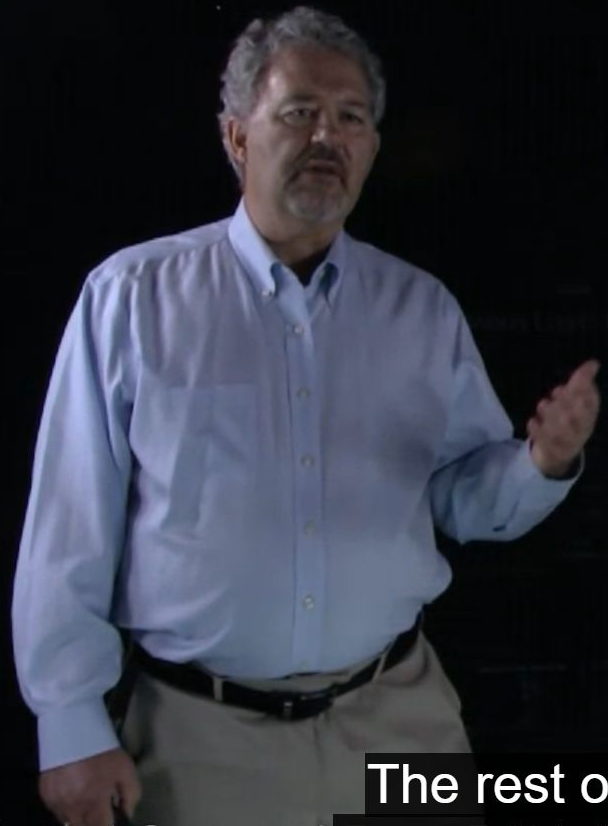


How can we make a binary counter in Verilog?



A fundamental circuit in digital systems is the counter suppose.

Synchronous Logic: Binary Counter

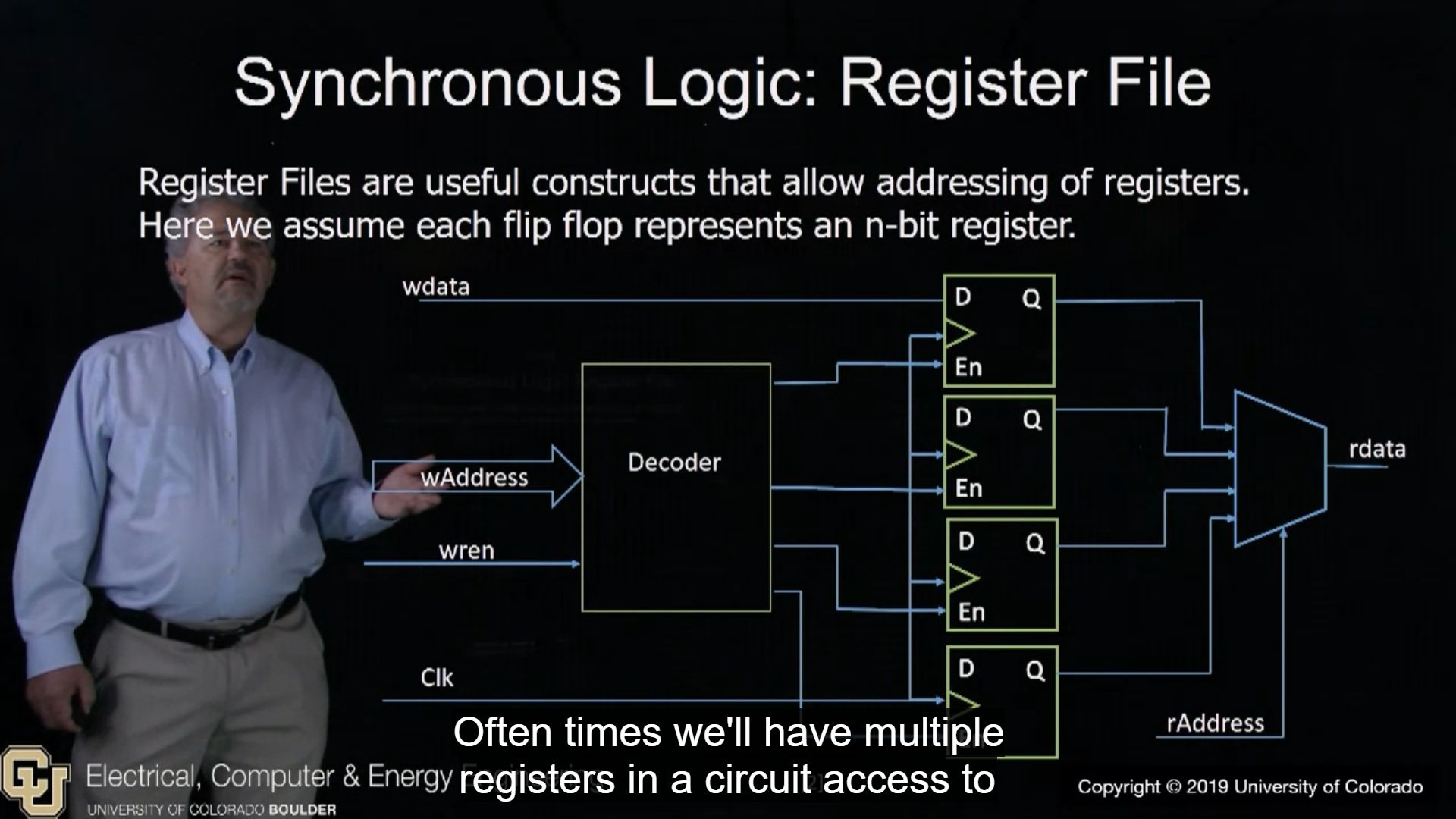


```
// Binary Counter
//
module Counter (
  input wire [3:0] d,
  input wire clk, reset, load, en,
  output reg [3:0] q
  );
  always @(posedge clk)
  begin
    if (reset)  // synchronous reset
      q <= 0;
    else if (load == 1'b1)
      q <= d;
    else if (en == 1'b1)
      q <= q + 1;
  end
```

The rest of the code shows that this is a completely synchronous circuit reset is

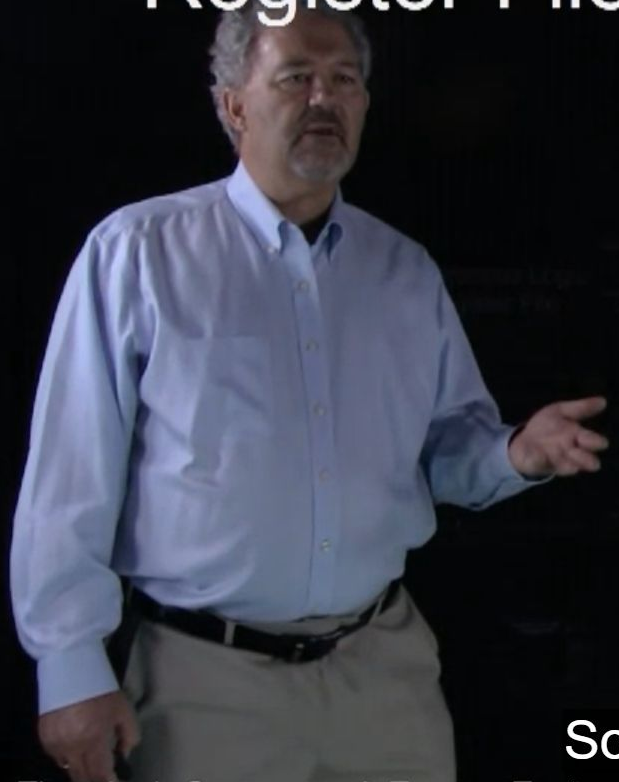
Synchronous Logic: Register File

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



Often times we'll have multiple registers in a circuit access to

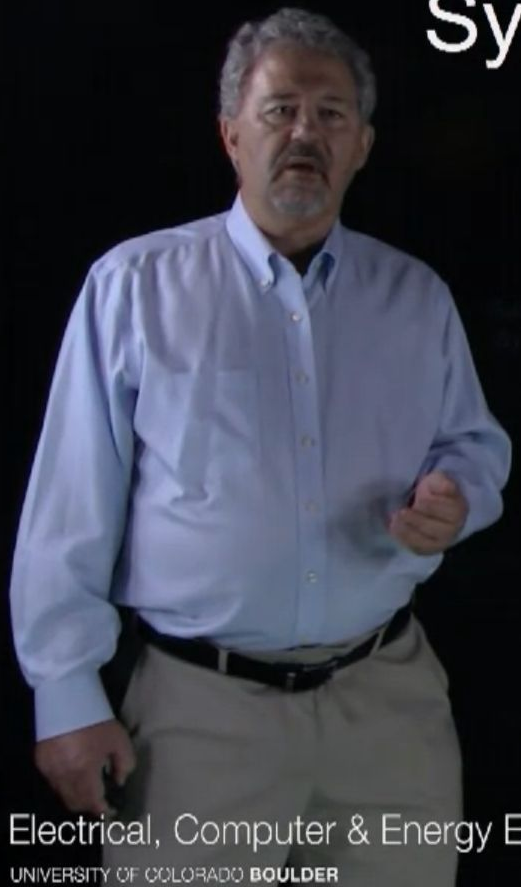
Synchronous Logic: Register File



```
// Data Register File
// 4 x 8
module regFile #(
  parameter Dwidth = 8, // #bits in word
            Awidth = 2  // #address bits
)
(
  input wire clk, wren,
  input wire [(Dwidth1):0] wdata,
  input wire [(Awidth1):0] waddr, raddr,
  output wire [(Dwidth-1):0] rdata
);
// Signal Declaration
reg [Dwidth-1:0] array_reg [2**Awidth-1:0];

always @(posedge clk)
  if (wren)  // synchronous enable
    array_reg[waddr] <= wdata;
assign rdata = array_reg[raddr];
```

So in the rest of the code to create the register file,

Summary – Verilog for Synchronous Circuits



In this video, you have learned:

- **How to describe synchronous circuits in Verilog, including**
  - **Data registers**
  - **Shift registers**
  - **Counters**
  - **Register Files**

References

[1] C. Unsalan and B. Tar, "Sequential Circuits" in *Digital System Design with FPGA,* McGraw-Hill India, 2017, pp. 215-217.

[2] Pong P. Chu, "Regular Sequential Circuit" in *Embedded SOPC Design with NIOS II Processor and Verilog Examples*, Hoboken, NJ, Wiley, ch. 5, sec. 5.2, pp. 100-102

Be Boulder.

University of Colorado **Boulder**