ECE 2300
Digital Logic & Computer Organization

Fall 2016

More Counters
Shift Registers
Verilog

Cornell University

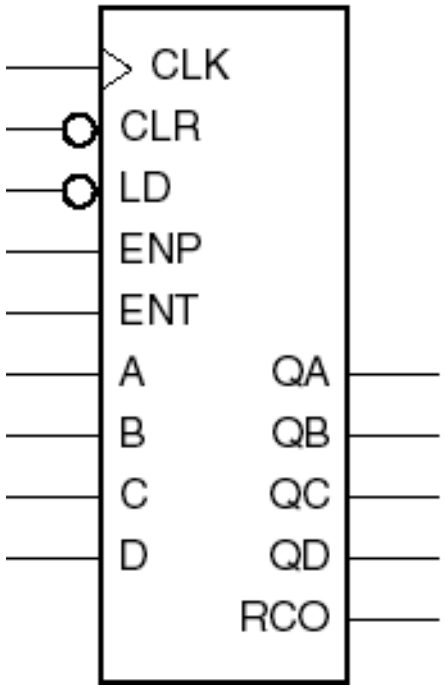
Binary Counter

- Counts in binary in a particular sequence
- Advances at every tick of the clock
- Many types

Up	Down	Free Running	Divide-by-n	n-to-m
0 0 0	1 1 1	0 0 0	0 0 0	n
0 0 1	1 1 0	0 0 1	0 0 1	n+1
0 1 0	1 0 1	0 1 0	0 1 0	n+2
0 1 1	1 0 0	0 1 1	0 1 1	.
1 0 0	0 1 1	1 0 0	1 0 0	.
1 0 1	0 1 0	1 0 1	.	.
.	.	1 1 0	.	m-1
.	.	1 1 1	n-1	m
.	.	0 0 0	0 0 0	n
		.	0 0 1	n+1
		.		.
		.		.

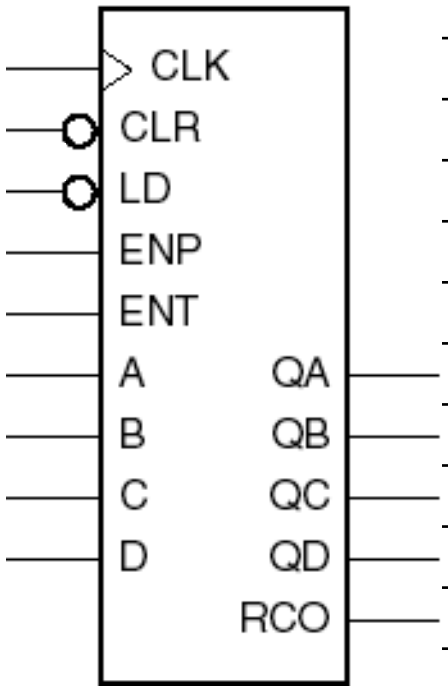
4-Bit Binary MSI Counter (’163)

- **Inputs**
  - CLK = clock input
  - CLR = synchronous clear
  - ENT AND ENP = enable counting
  - LD = load A,B,C,D (like a register)
  - A-D = load inputs



- **Outputs**
  - QA-QD = current number in sequence
  - RCO = ripple carry out
    = 1 if QA-QD = 1111, and ENT is 1

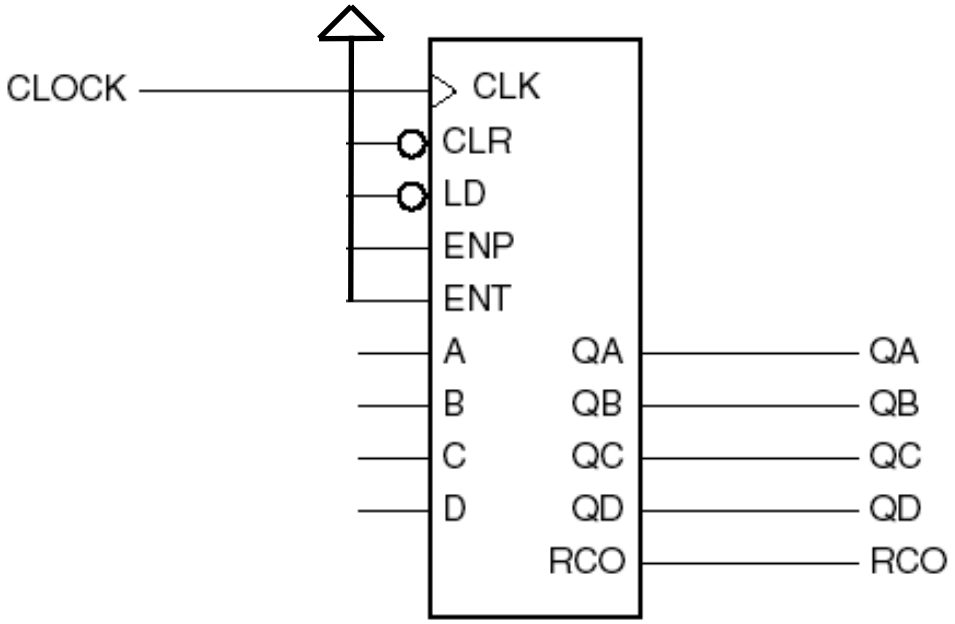
’163 Truth Table



Inputs				Current State				Next State			
/CLR	/LD	ENT	ENP	QD	QC	QB	QA	QD*	QC*	QB*	QA*
0	X	X	X	X	X	X	X	0	0	0	0
1	0	X	X	X	X	X	X	D	C	B	A
1	1	0	X	X	X	X	X	QD	QC	QB	QA
1	1	X	0	X	X	X	X	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

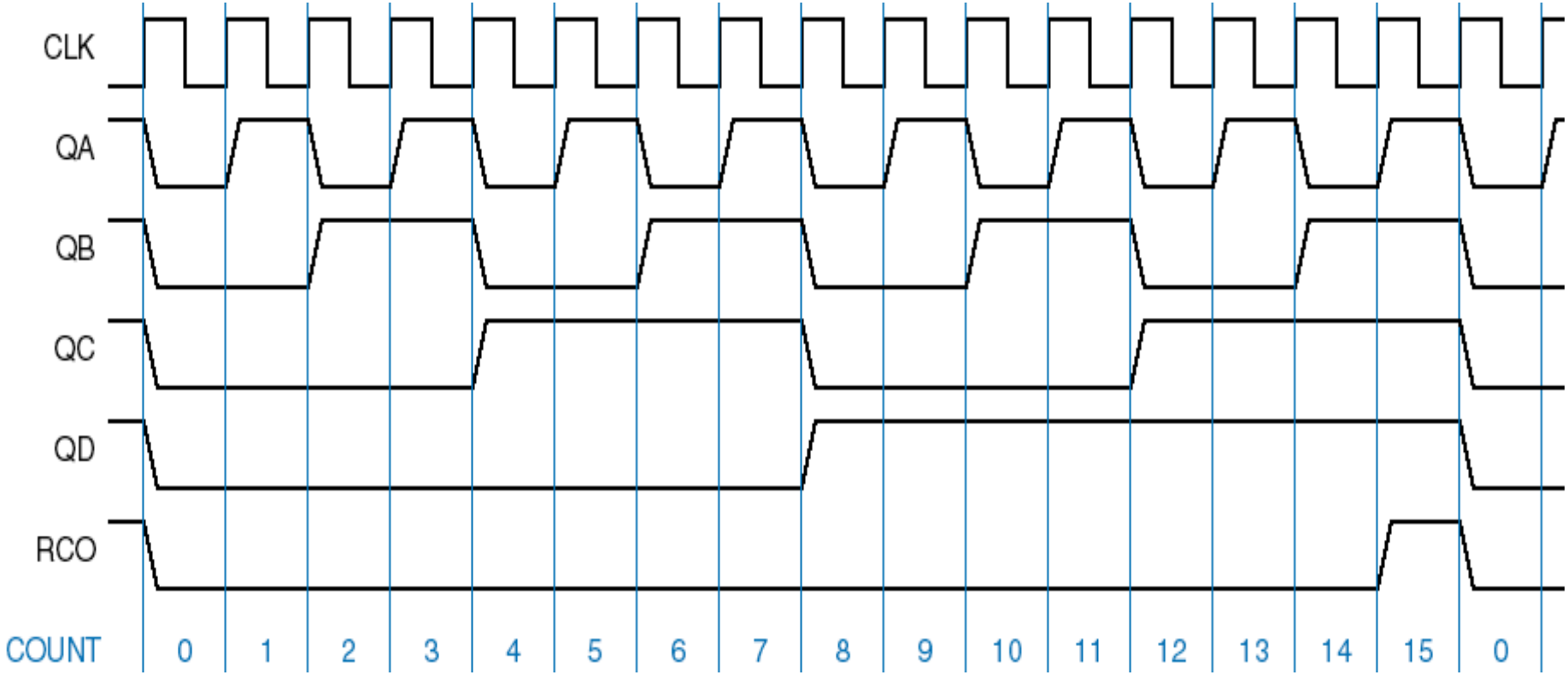
Free Running Up Counter

- Always enabled
- Increments every rising clock edge



0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
.
.
.

Free Running Counter Timing Diagram

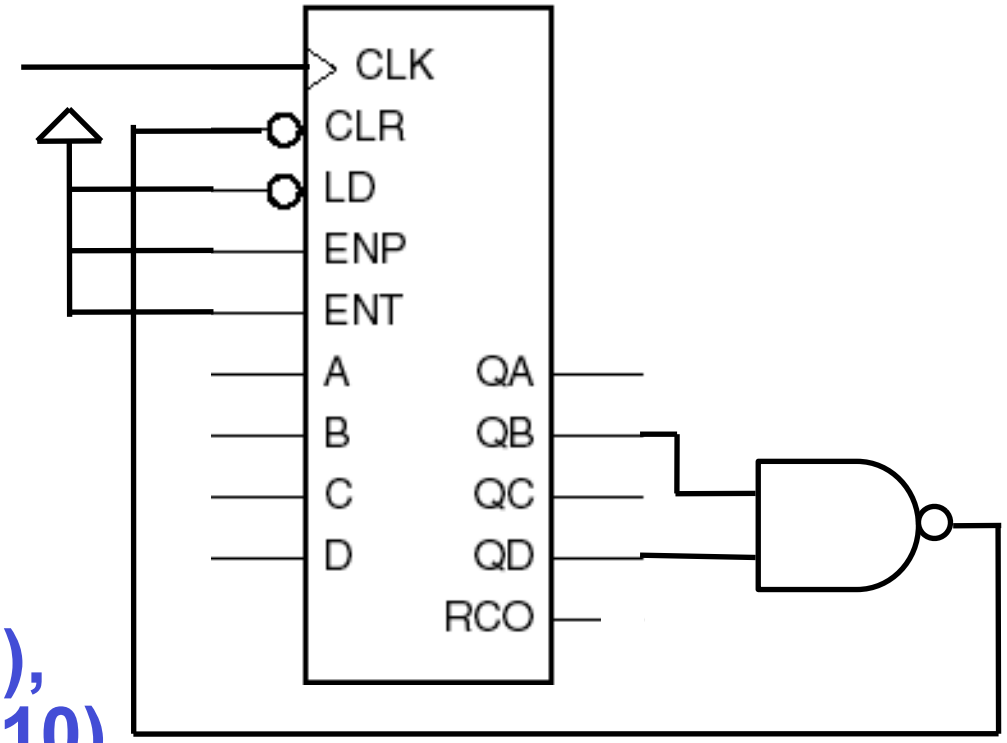


Divide-by-11 Counter

- **Count sequence**

  0000
  0001
  ⋮
  1001
  1010
  0000
  ⋮

- **To divide by 11 (1011), assert clear at 10 (1010)**

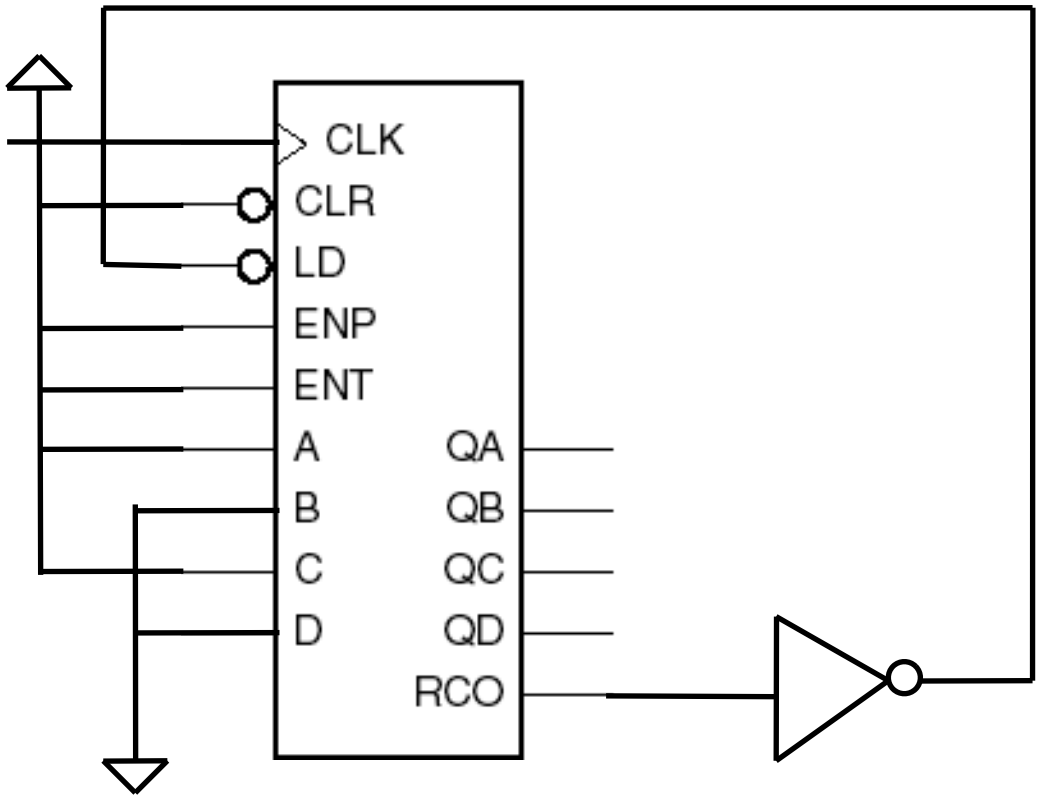


5-to-15 Counter

- **Count sequence**

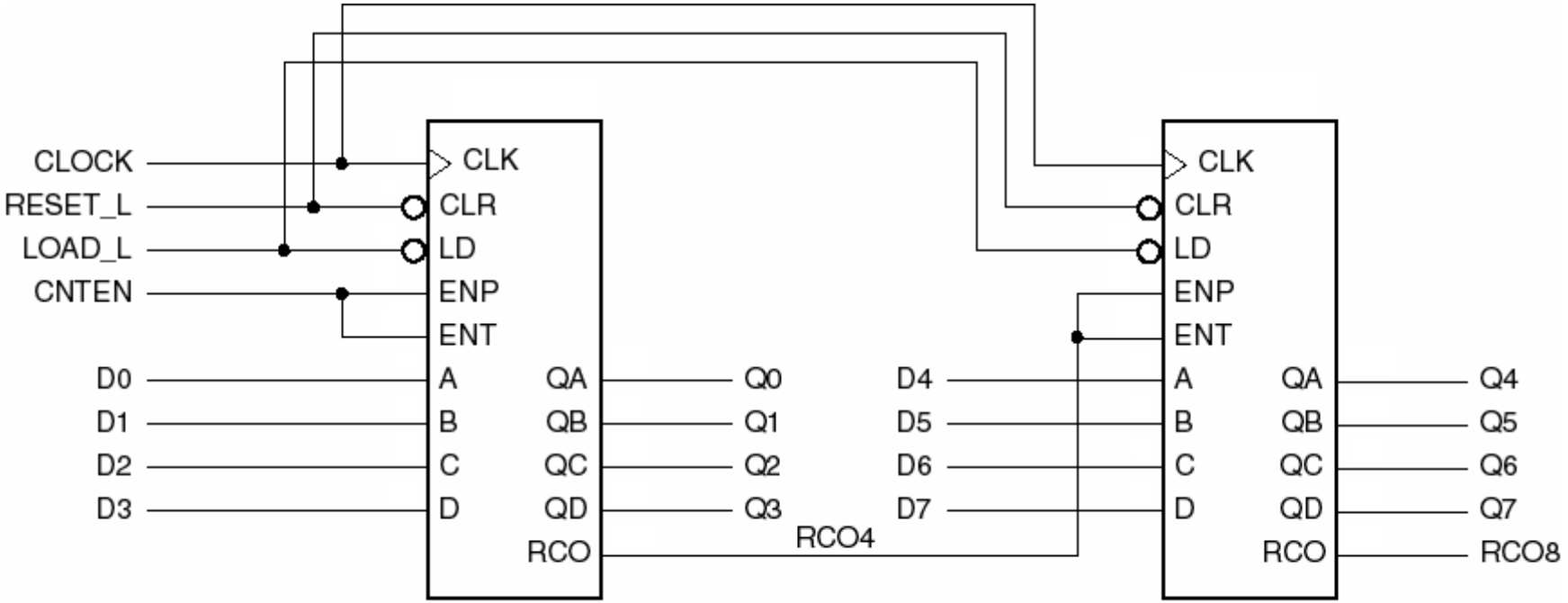
  0101
  0110
  ⋮
  1110
  1111
  0101
  ⋮

- **Load 5 (0101) when count reaches 15 (1111)**



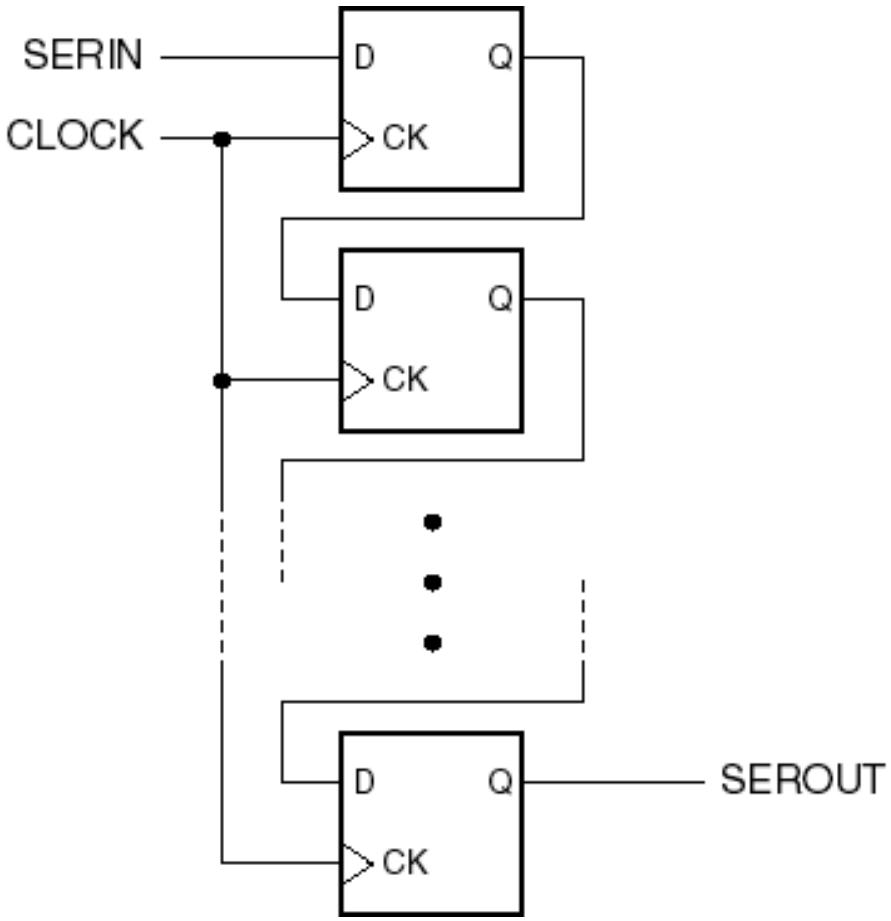
Cascading Counters

- Build larger counter from smaller ones
- Next counter enabled when previous = 11...11
- 8-bit cascaded counter



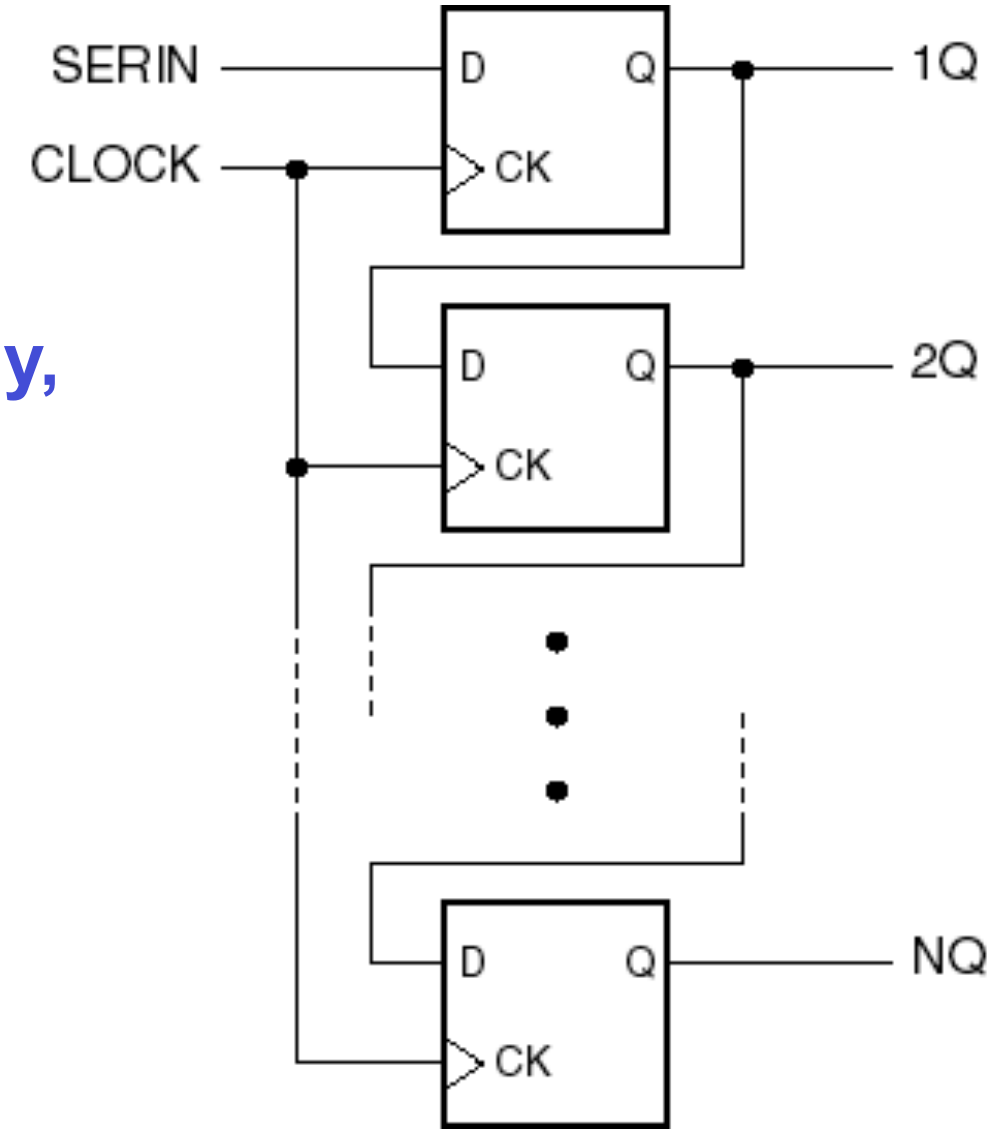
Shift Register

- Data shifted through a series of D flip-flops
- SERIN appears on SEROUT after n clock ticks
- *Serial-in, serial-out*
- Applications
  - Serial data transmission
  - Computer arithmetic

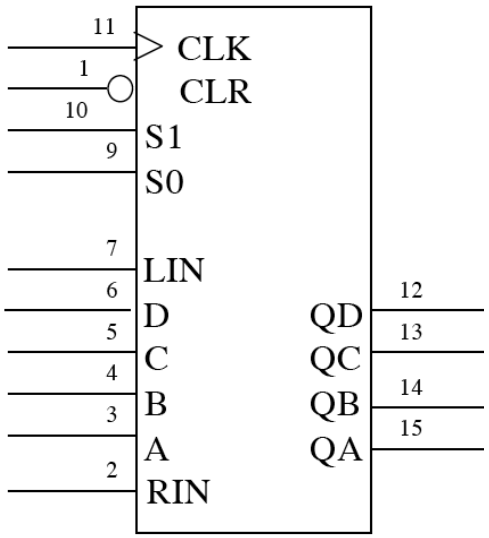


Serial-In, Parallel-Out Shift Register

- **Data shifted in serially, read out in parallel**
- ***Serial to parallel conversion***



’194 Shift Register



	Inputs		Next State			
Function	***S1***	***S0***	***QA****	***QB****	***QC****	***QD****
Hold	0	0	QA	QB	QC	QD
Shift Right	0	1	RIN	QA	QB	QC
Shift Left	1	0	QB	QC	QD	LIN
Load	1	1	A	B	C	D

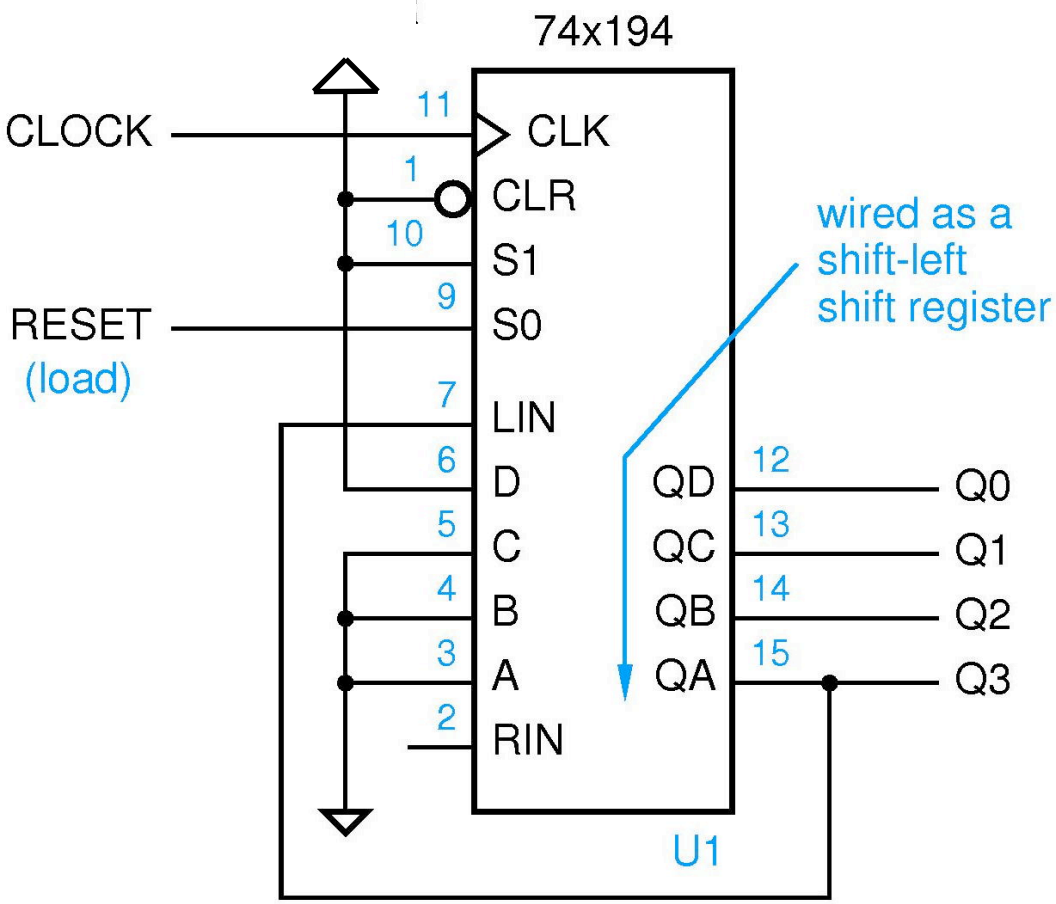
Ring Counter

- **Single 1 shifts through the outputs**

0 0 0 1
0 0 1 0
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
.
.
.

4-bit Ring Counter Using ’194

- **Load 0001 when RESET asserted**
- **Then shift left and wrap around**
  - **0001**
  - **0010**
  - **0100**
  - **1000**
  - **0001**
- **Wraparound: Tie QA to LIN**



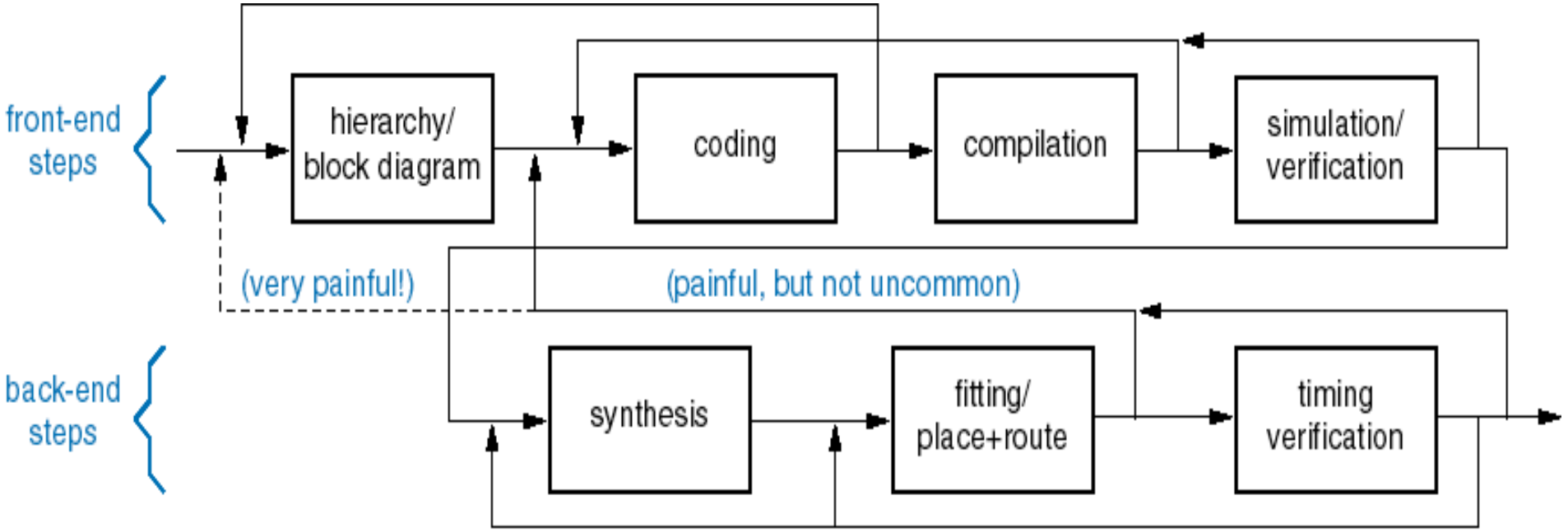
Hardware Description Languages (HDLs)

- Describe hardware behavior

- Advantages
  - Efficiently code large, complex designs
  - Can code at a more abstract level than schematics
  - More understandable than schematics
  - Synthesis tools can automatically generate hardware

- Typical features
  - Structure and instantiation
  - Bit-level behavior
  - Concurrency

HDL-based Design Flow



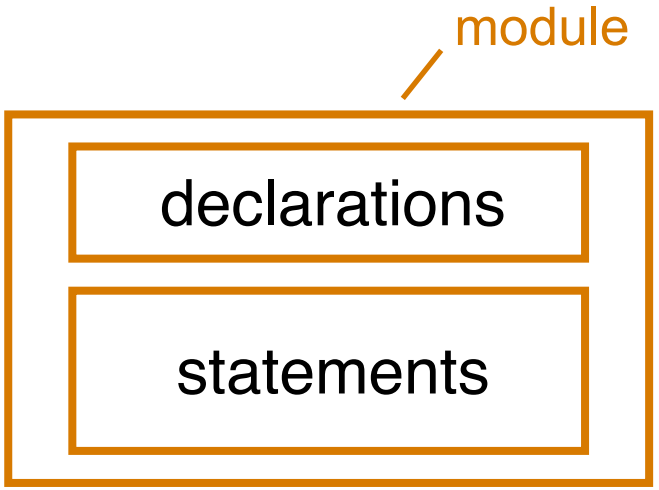
- **Back-end differs by target technology**
  - **FPGA, ASIC, full-custom chip**

Verilog

- Developed in the early 1980s by Gateway Design Automation (later bought by Cadence)
- Syntactically similar to C
- Supports modeling, simulation, and synthesis
- We will use a subset of language features
  - Textbook describes additional features that may or may not work with Quartus II

Verilog Program Structure

- **System is a collection of *modules***
  - **Module corresponds to a single piece of hardware**



- **Declarations**
  - **Describe names and types of inputs and outputs**
  - **Describe local signals, variables, constants, etc.**

- **Statements specify what the module does**

Verilog Program Structure

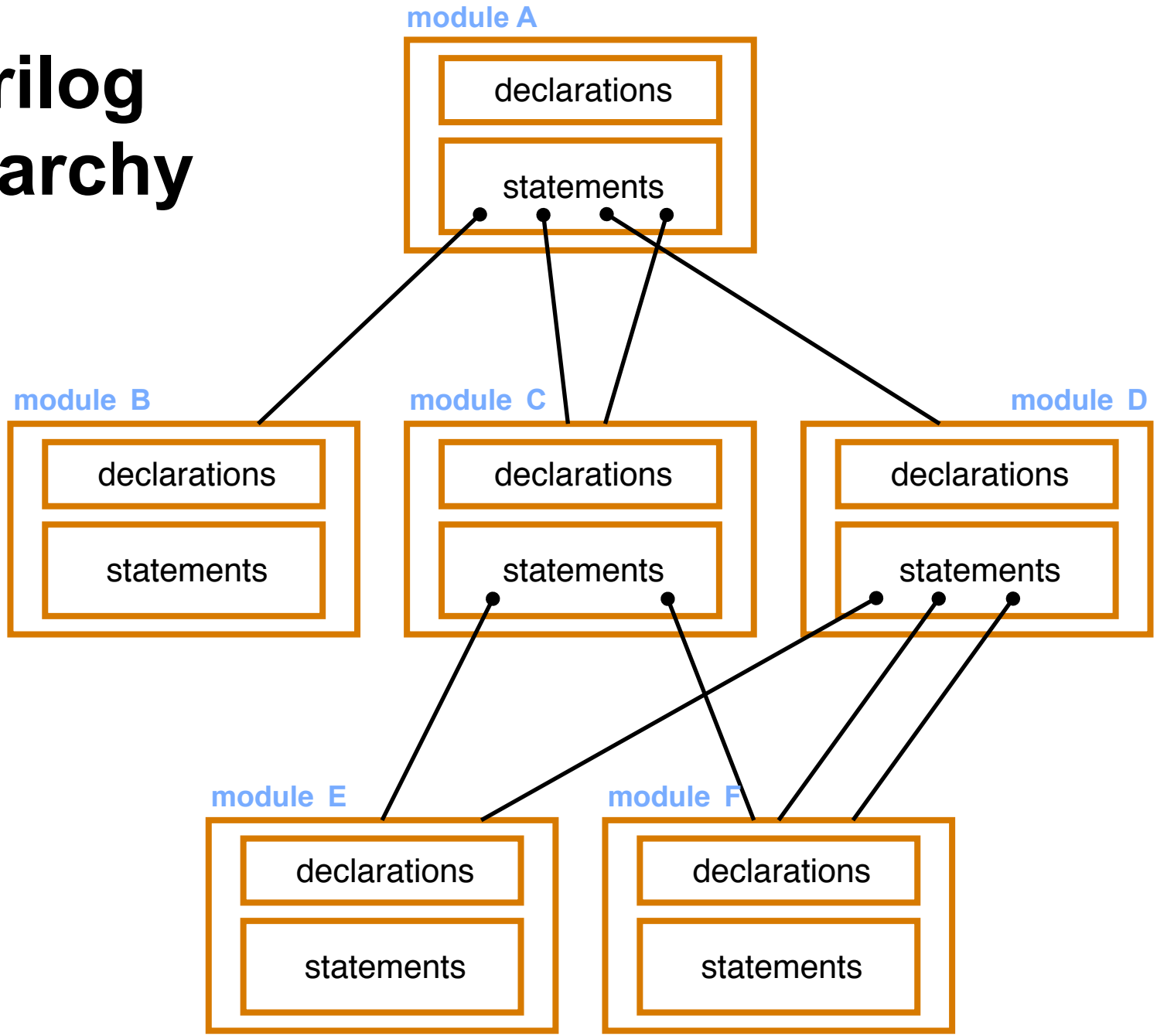
```
module V2to4dec(i0,i1,en,y0,y1,y2,y3);
   input i0,i1,en;
   output y0,y1,y2,y3;
   wire noti0,noti1;

   not U1(noti0,i0);
   not U2(noti1,i1);
   and U3(y0,noti0,noti1,en);
   and U4(y1,    i0,noti1,en);
   and U5(y2,noti0,    i1,en);
   and U6(y3,    i0,    i1,en);
endmodule
```

Declarations

Statements

Verilog Hierarchy



Signal Values

- **Verilog signals can take 4 values**
  - 0 Logical 0, or false
  - 1 Logical 1, or true
  - x Unknown logical value
  - z High impedance (Hi-Z)

Vectors

- **Grouping of 1-bit signals**
  - reg[7:0] byte1, byte2, byte3;
  - wire[0:3] asel;
  - Right-most bit is always least significant
- **Bit selection**
  - byte1[5:2]
- **Concatenation**
  - {byte1,byte2}
- **Bitwise Boolean operators**
  - byte1 & byte2
- **Literals**
  - 8'b11011

Arithmetic and Shift Operators

- **Arithmetic**
  - `+` Addition
  - `-` Subtraction
  - `*` Multiplication
  - `/` Division
  - `%` Modulus

- **Shift**
  - `<<` Shift left
  - `>>` Shift right

Bitwise Boolean Operators

- **Bitwise Boolean Operators**
  - ~ NOT
  - & AND
  - | OR
  - ^ Exclusive OR
  - ^~ Exclusive NOR

Variables

- Store values during program execution, but may not have physical significance
- Common types are reg, logic, and integer
  - reg: single bit or vector of bits
    - logic: introduced as alternative to reg in SystemVerilog
  - integer: integer value
- Only used in *procedural code*
  - Cannot be changed from outside the procedure

Procedural Statements

- **Procedural statements are similar to conventional programming language statements**
  - **Begin-end blocks**
    - begin *procedural-statement* … *procedural-statement* end
  - **If**
    - if (*condition*) *procedural-statement* else *procedural-statement*
  - **Case**
    - case (*sel-expr*) *choice* : *procedural-statement* … endcase
  - **While**
    - while (*logical-expression*) *procedural-statement*
  - **Repeat**
    - Repeat (*integer-expression*) *procedural-statement*

Conditionals

- **Logical operators used in conditional expressions**

  && logical AND

  || OR

  ! logical NOT

  == logical equality

  != logical inequality

  > greater than

  >= greater than or equal

  < less than

  <= less than or equal

- **Don’t confuse with Boolean operators**

Nets

- A *net* provides connectivity between elements
- Default net type is a *wire*
  - wire noti0, noti1;
  - Any signal not in the input/output list or net declaration is assumed to be a wire
- Other net types
  - tri, wand, wor, ...
  - Only wire and tri supported by Quartus II

Verilog programming styles

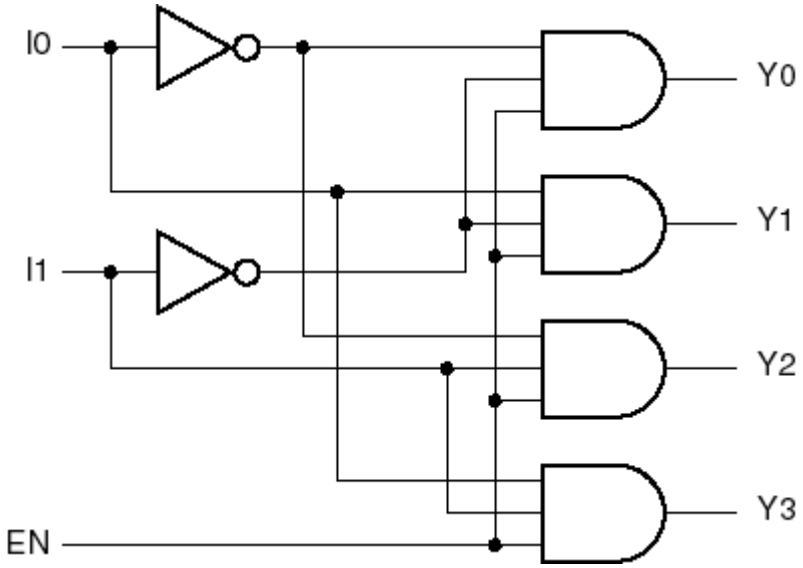
- **Structural**
  - Textual equivalent of drawing a schematic
  - Uses *instance* statements
- **Dataflow**
  - Describes circuit in terms of flow of data and operations on that data
  - Uses *continuous-assignment* statements
  - Called “structural” in textbook
- **Behavioral**
  - High-level algorithmic description
  - Uses *procedural code*

Structural Style

```
module V2to4dec_struct( i0,i1,en,y0,y1,y2,y3 );
   input i0,i1,en;
   output y0,y1,y2,y3;
   wire noti0,noti1;

   not U1(noti0,i0);
   not U2(noti1,i1);
   and U3(y0,noti0,noti1,en);
   and U4(y1,    i0,noti1,en);
   and U5(y2,noti0,    i1,en);
   and U6(y3,    i0,    i1,en);
endmodule
```

Correspondence with schematic



Dataflow Style

```
module V2to4dec_df( i0,i1,en,y0,y1,y2,y3 );
  input i0,i1,en;
  output y0,y1,y2,y3;

  assign y0 = en & ~i0 & ~i1;
  assign y1 = en &   i0 & ~i1;
  assign y2 = en & ~i0 &   i1;
  assign y3 = en &   i0 &   i1;
endmodule
```

Correspondence with Boolean logic

New value is assigned to the lhs whenever any value on the rhs changes

Behavioral Style

```
module V2to4dec_beh( i0,i1,en,y0,y1,y2,y3 );
   input i0,i1,en;
   output y0,y1,y2,y3;
   reg y0,y1,y2,y3;

   always @ (en or i0 or i1)
   begin
     y0 = en & ~i0 & ~i1;
     y1 = en &  i0 & ~i1;
     y2 = en & ~i0 &  i1;
     y3 = en &  i0 &  i1;
   end
endmodule
```

- **Procedural code**
- **Key element is the *always block***
- **Begin-end block groups sequences of procedural statements**
- **reg needed to change the output values**

Before Next Class

- H&H 3.4, 4.6, 4.9

Next Time

More Verilog

Finite State Machines