Intro to Digital Design
Encoders, Decoders, Registers, Counters

Instructor: Justin Hsia

Teaching Assistants:

Emilio Alcantara
Eujean Lee
Naoto Uemura
Pedro Amarante
Wen Li

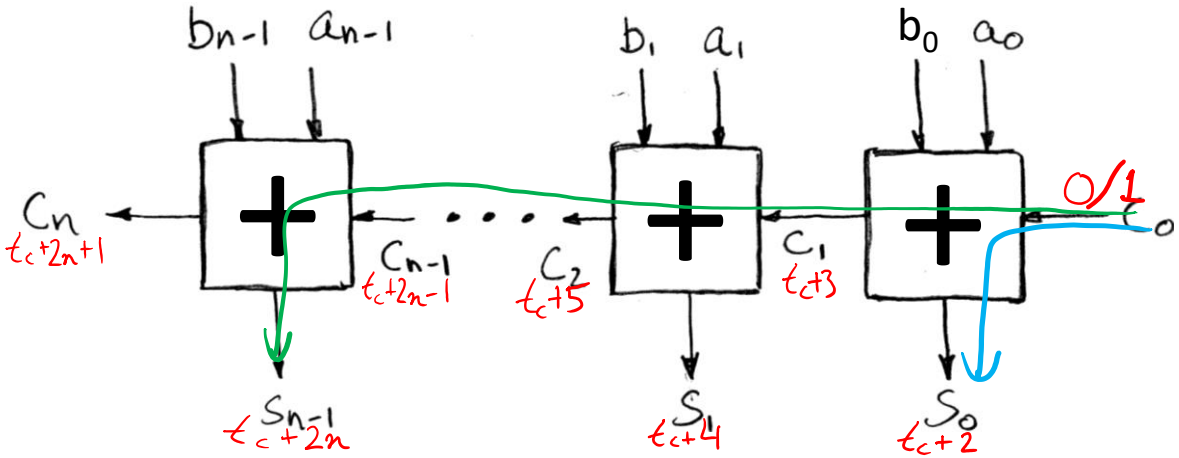
Relevant Course Information

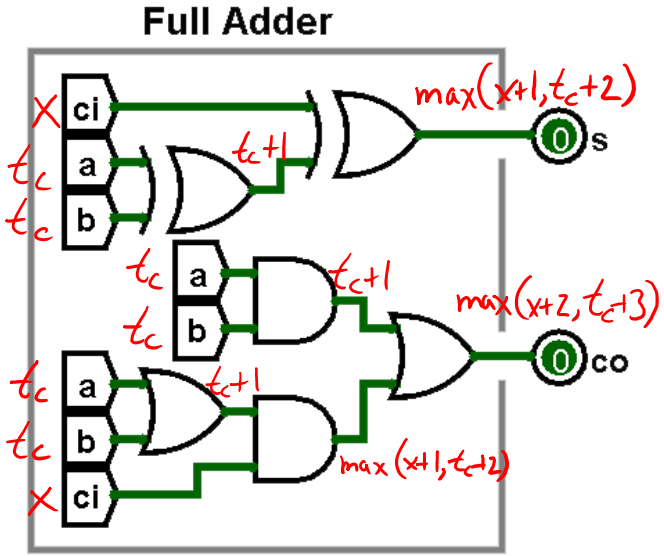
- Lab 7 – Useful Components
  - Modifying Lab 6 game to implement common circuit elements
  - Build a tunable computer opponent!

- Quiz 2 is next week in lecture
  - Last 30 minutes (+ 5 min buffer), 10% of your course grade
  - On Lectures 4-5: Sequential Logic, Timing, FSMs, and Verilog
  - Past Quiz 2 (+ solutions) on website: Course Info → Quizzes

Practice Problem

- For an n-bit ripple-carry adder, what is the shortest and longest time that output S changes after each clock cycle?
  - A, B, c_0 from registers (show up at t_{C2Q}); S goes directly to a register input.
  - Assume all gates have a delay of 1 ns; use variables for all other timing values





bit s_i settled at time $t_{c2q}+2i+2$

shortest(s_0): $t_{c2q}+1 \geq t_{hold}$

longest(s_{n-1}): $t_{c2q}+2n \leq t_{period} - t_{setup}$

Outline

- **Circuit Routing Elements**
- Register Revisited

Standard Circuit Routing Elements

- Multiplexor (mux)
  - Pass one of N inputs to single output
- **Simple Encoder**
  - One of N inputs is active and output tells you which one (in binary)
- **1-of-N Binary Decoder**
  - Interpret binary input to assert one of N output wires
- **Demultiplexer (demux)**
  - Pass single input onto one of N outputs

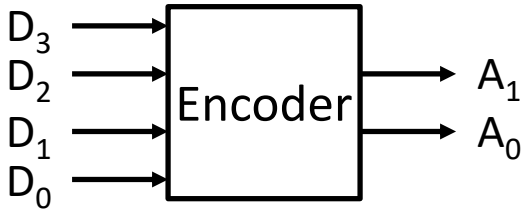
Encoder

- A device or circuit that converts information from one format or code to another
  - Examples: decimal to binary, keyboard press to character, rotary encoder for odometer, analog-to-digital converter

- A **simple encoder** is a one-hot to binary converter
  - One-hot means at most only one input line (out of $m \leq 2^n$) will be high
  - Output is the binary representation (n bits wide) of the asserted line's bit numbering or "address"
  - Referred to as an m:n encoder (read as "m-to-n")

Simple Encoder Implementation

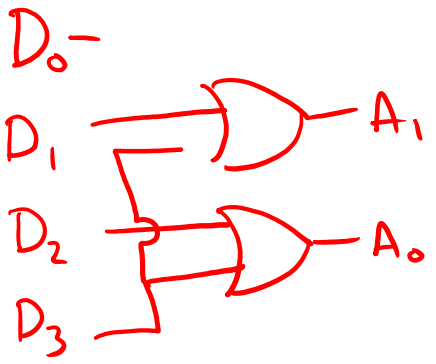
- 4:2 Encoder



D_3	D_2	D_1	D_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$A_1 = D_2 + D_3$

$A_0 = D_1 + D_3$

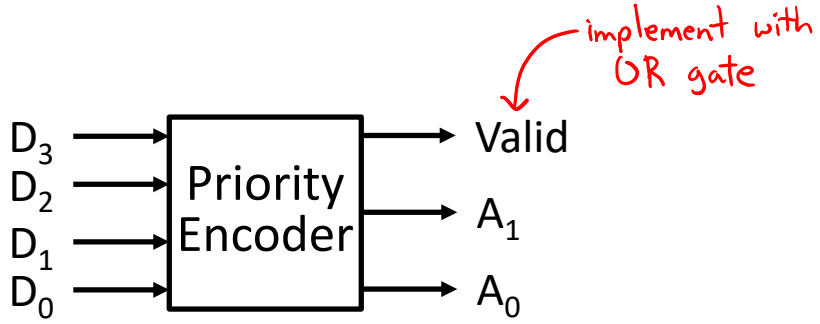


- Two issues:
  1) What if multiple inputs are hot? D_2 and D_1 hot acts like D_3 was hot
  2) What if no inputs are hot? acts like D_0 was hot when it wasn't

Priority Encoder

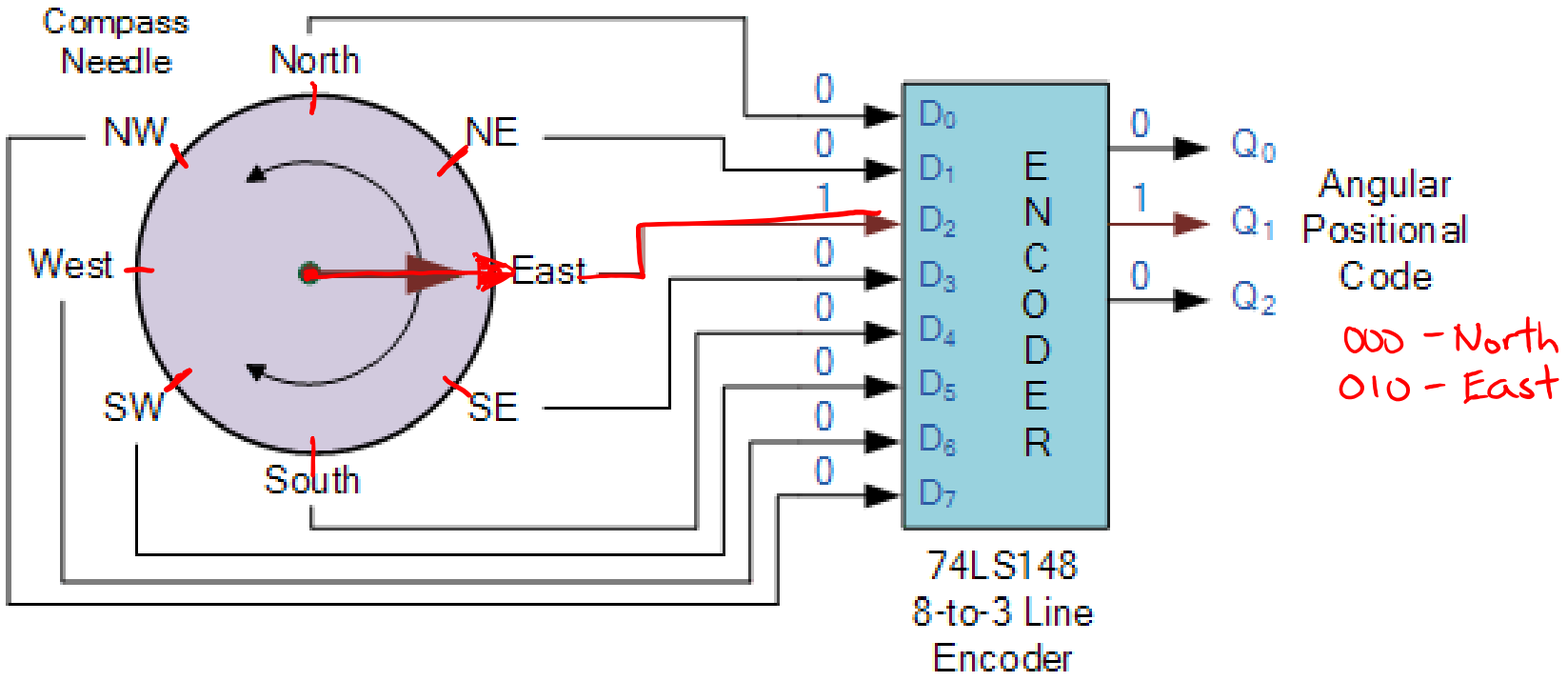
1) Use *priorities* to resolve the problem of multiple active input lines
   - Example: Highest ID active is given priority ("wins")
2) Add an output to identify when at least 1 input active

D_3	D_2	D_1	D_0	A_1	A_0	Valid
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1



Encoder Examples

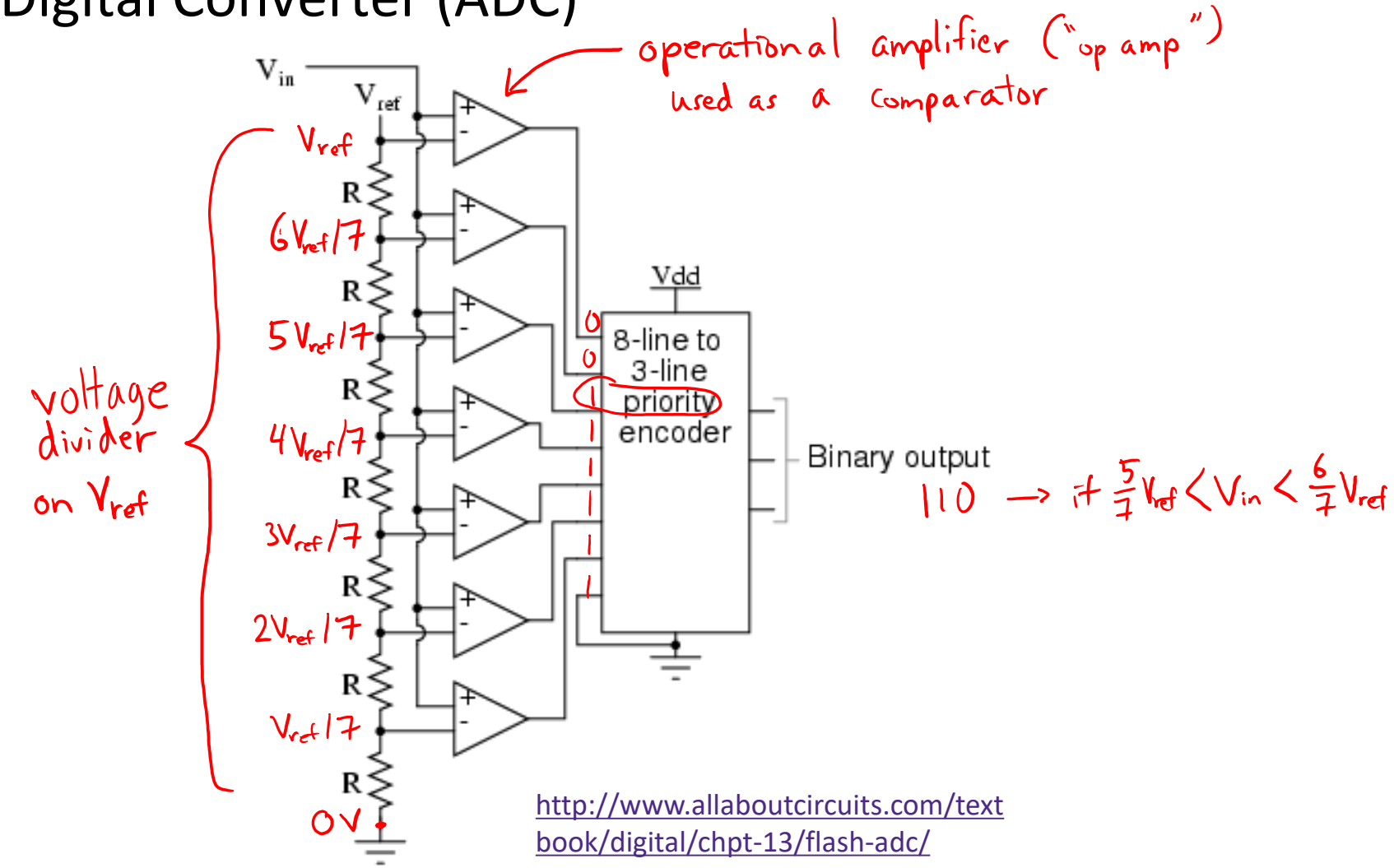
- Navigation (Compass) Encoder



http://www.electronics-tutorials.ws/combination/comb_4.html

Encoder Examples

- Analog-to-Digital Converter (ADC)



http://www.allaboutcircuits.com/textbook/digital/chpt-13/flash-adc/

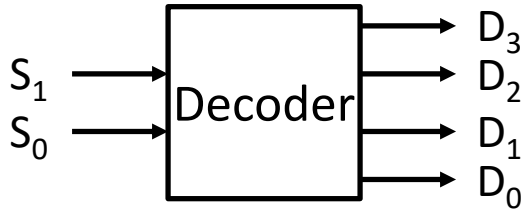
Decoder

- A device or circuit that converts or interprets information from an encoded format
  - Examples: binary to decimal, CPU instruction decoder, video decoder (analog to digital)

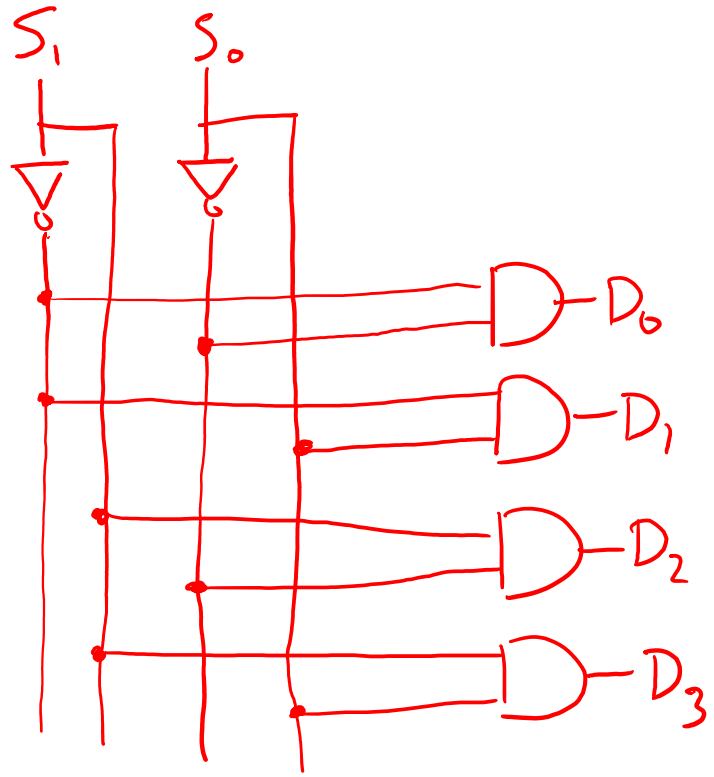
- A **binary decoder** is a binary to one-hot converter
  - n input bits serve as bit number or "address" specifier
  - Only corresponding output out of $m \leq 2^n$ will be asserted
  - Referred to as an n:m decoder (read as "n-to-m")

1-of-N Binary Decoder Implementation

- 2:4 Decoder



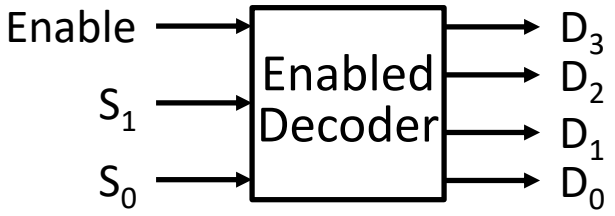
S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



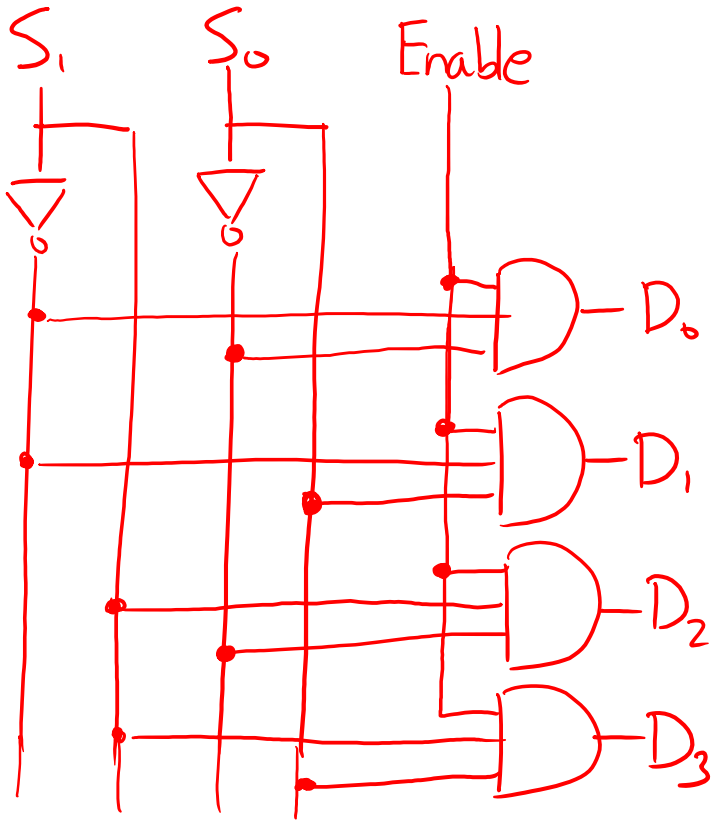
- Issue:
  - What do we do if we want nothing to happen?

Enabled Decoder

- Only have active output when Enable signal is high



Enable	S_1	S_0	D_3	D_2	D_1	D_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

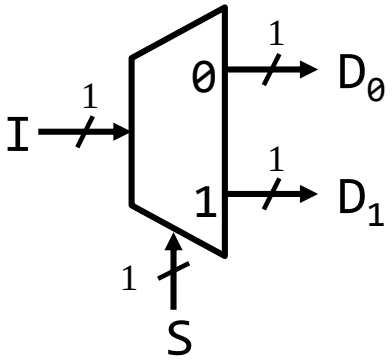


Enabled Decoder in Verilog

```
module enDecoder2_4 (out, in, enable);
  output logic [3:0] out;
  input  logic [1:0] in;
  input  logic       enable;

  always_comb begin

    if (enable)
      case (in)
        2'b00:  out = 4'b0001;
        2'b01:  out = 4'b0010;
        2'b10:  out = 4'b0100;
        2'b11:  out = 4'b1000;
      endcase
    else
      out = 4'b0000;

  end

endmodule  // enDecoder2_4
```

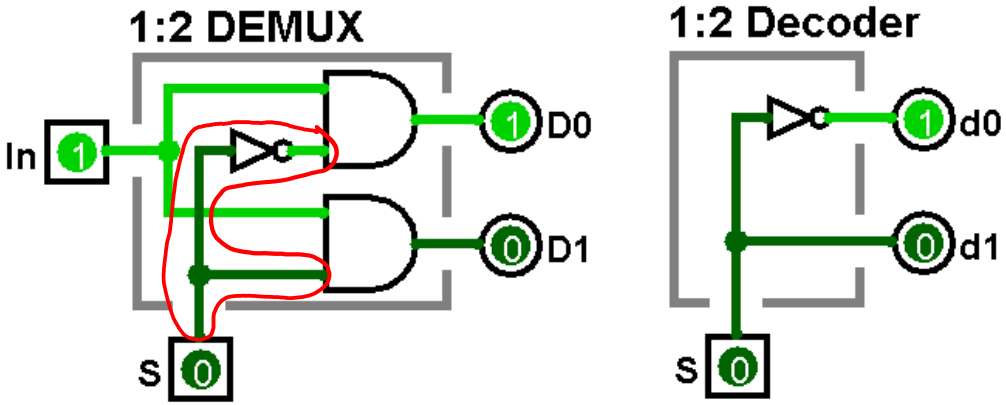
Decoder Examples: Demultiplexer

- **1-bit 1-to-2 DEMUX:**



- **Truth Table:**

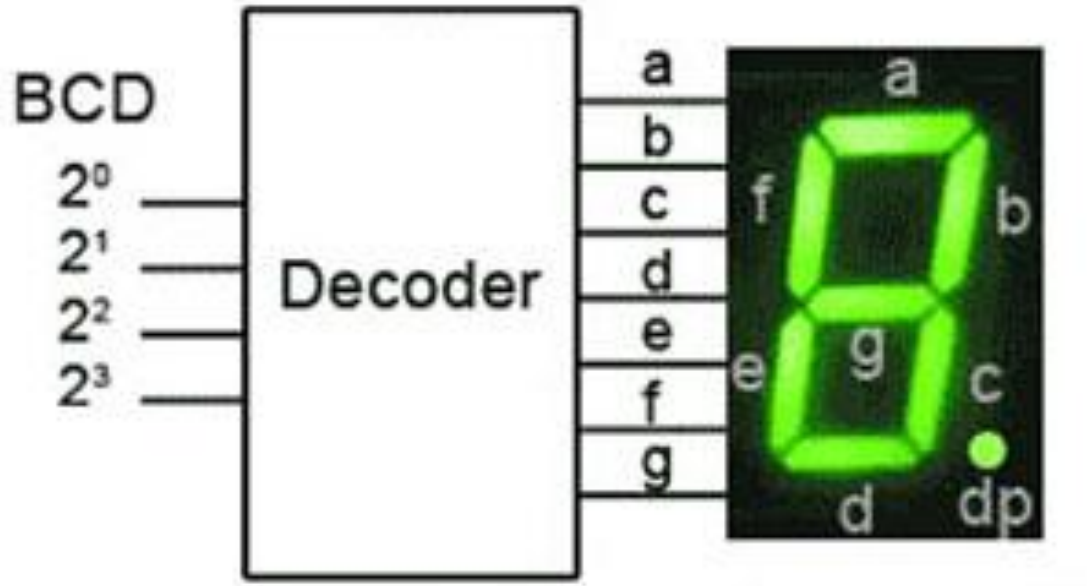
S	I	D_1	D_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



- More generally, AND d_i output from decoder with *every* input bit that is wired to DEMUX output D_j

Decoder Examples

- Binary to 7-seg display
  - You've already made this in this class!



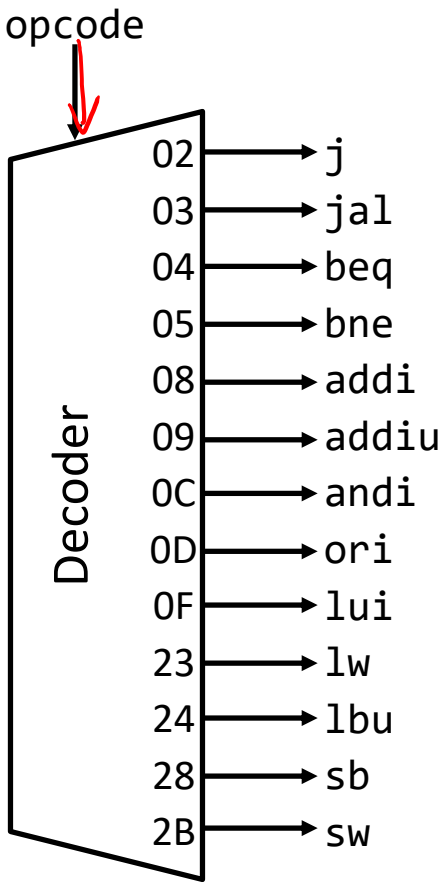
http://www.learnabout-electronics.org/Digital/dig44.php

Decoder Examples

- MIPS instruction decoder (RISC) vs. x86-64 (CISC)
  - Upper 6 bits of a 32-bit MIPS instruction
  - Part of the control portion of a CPU

take CSE 469 !

Instruction	Name	Opcode
addi	Add Imm.	001000
addiu	Add Imm. Unsigned	001001
andi	And Imm.	001100
beq	Branch On Equal	000100
bne	Branch On Not Equal	000101
j	Jump	000010
jal	Jump and Link	000011
lbu	Load Byte Unsigned	100100
lui	Load Upper Imm.	001111
lw	Load Word	100011
ori	Or Imm.	001101
sb	Store Byte	101000
sw	Store Word	101011

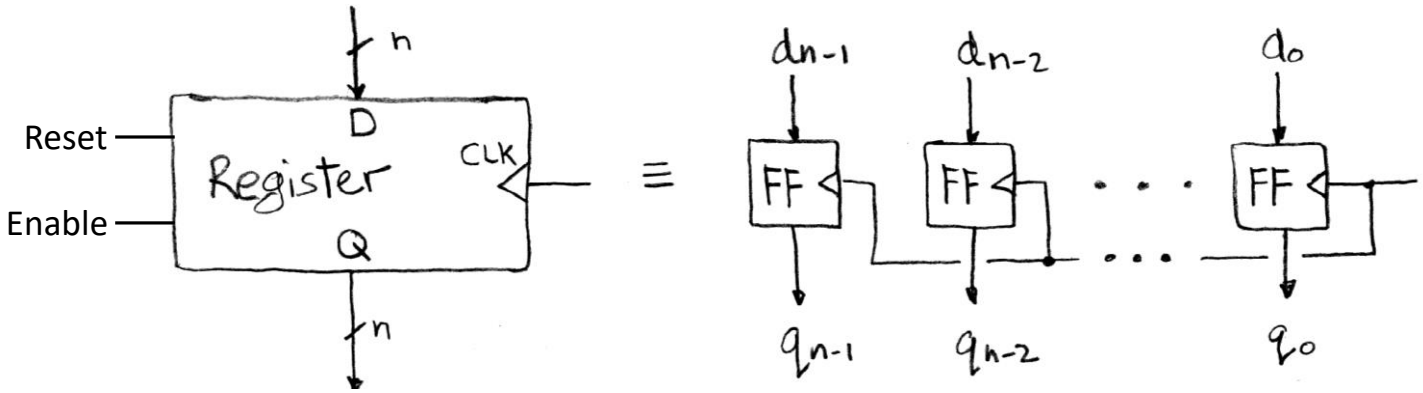


Technology Break

Outline

- Circuit Routing Elements
- **Register Revisited**

State Element Revisited: Register



- n instances of flip-flops together
  - One for every bit in input/output bus width

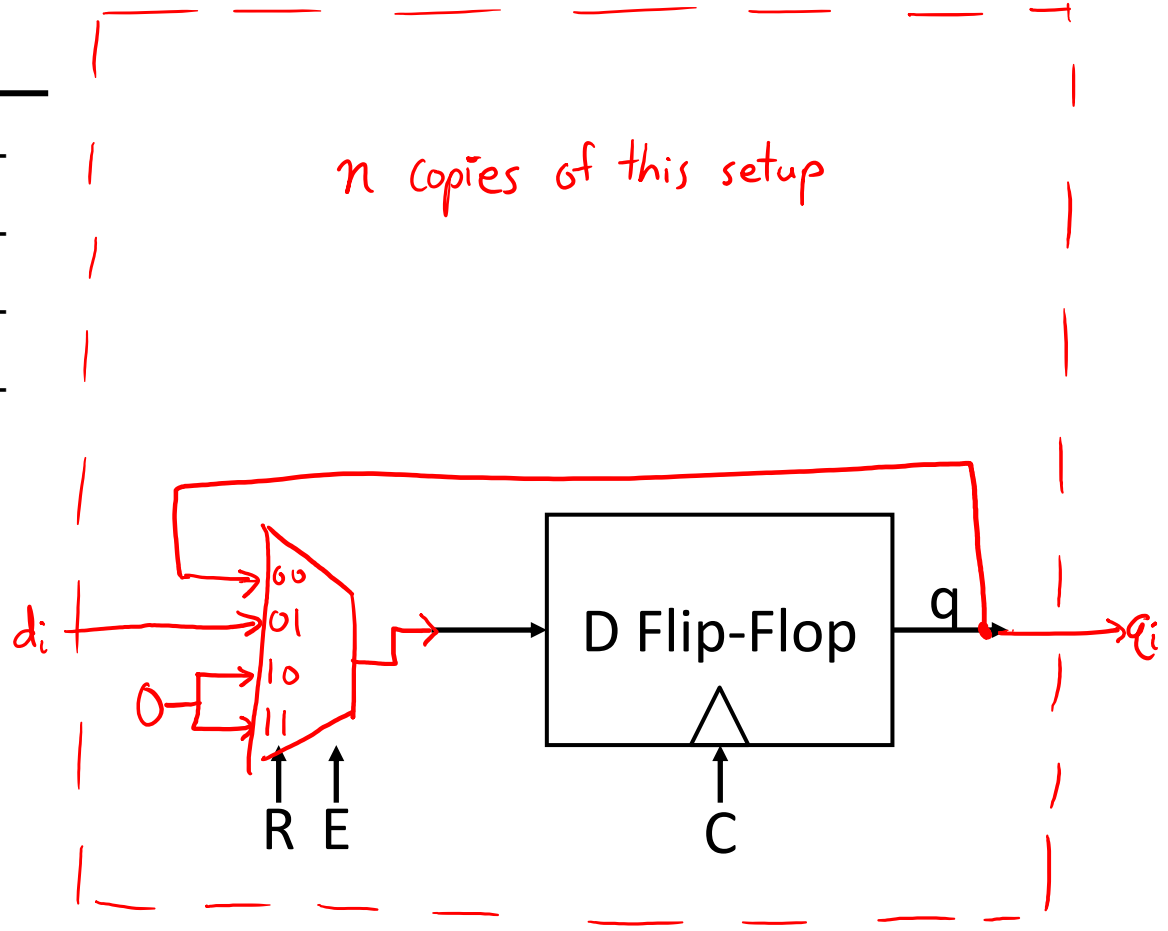
- Desired behaviors (synchronous)
  - Output Q resets to zero when `Reset` signal is high
  - Hold current value unless `Enable` signal is high

Controlled Register

- Here using shorthand C (clock), R (reset), E (enable)

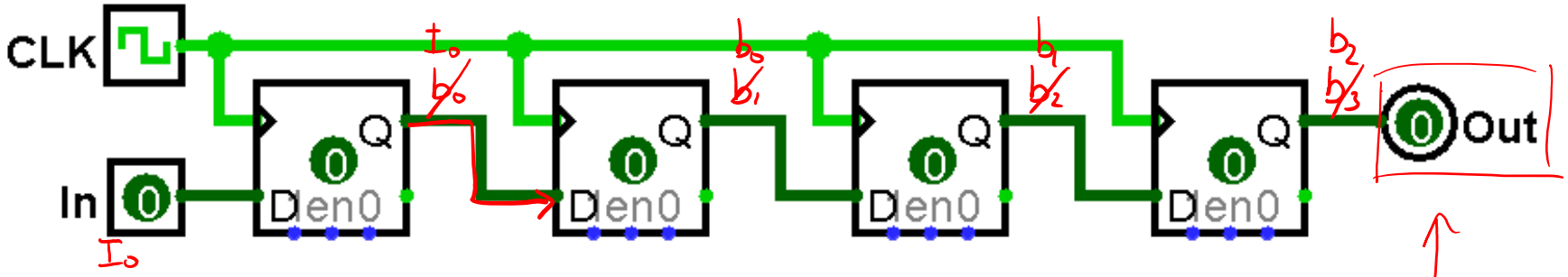
Reset	Enable	Action
0	0	q = q
0	1	q = d
1	0	q = 0
1	1	q = 0

priority



Shift Register

- Register that shifts the binary values in one or both directions

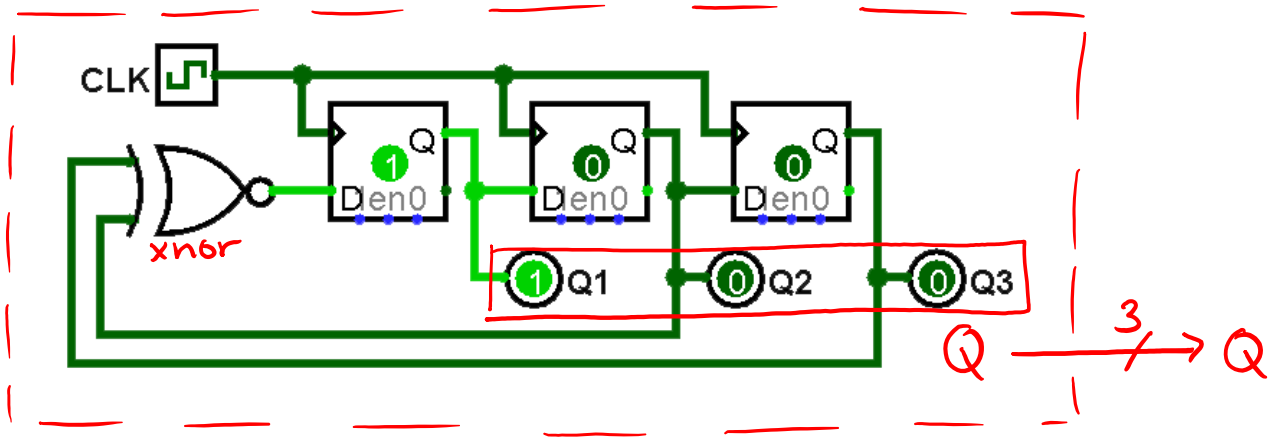


- Where do we get the input from?
  - External input (*e.g.*, delay a signal)
  - Function of current bits (*e.g.*, linear-feedback shift register)
- What is the output data of interest?
  - Last (oldest) bit of sequence
  - Entire set of current bits

usually related

Linear Feedback Shift Register (LFSR)

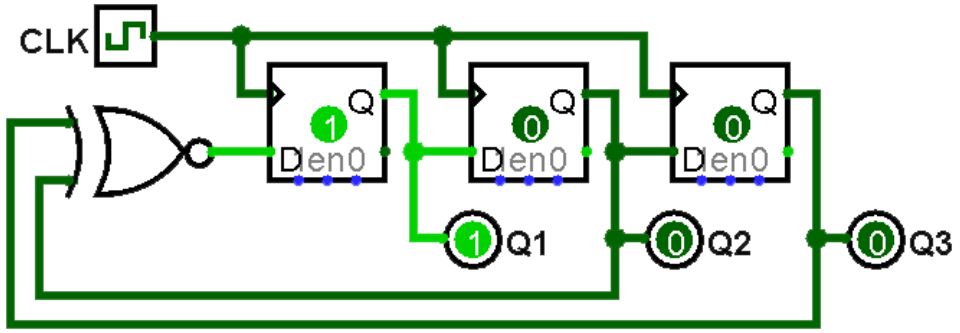
- Shift register input is a logical combination of the current state bits:



- Example: pseudo-random number generator
  - Input: no external input!
  - Output: all state bits together as a bus

Simple LFSR in Verilog

- How to implement this in Verilog?




```
module LFSR(clk,q1,q2,q3);
  input logic clk;
  output logic q1,q2,q3;

  always_ff @(posedge clk)
  begin
    q2 <= q1;
    q3 <= q2;
    q1 <= ~(q2 ^ q3);
  end
endmodule
```

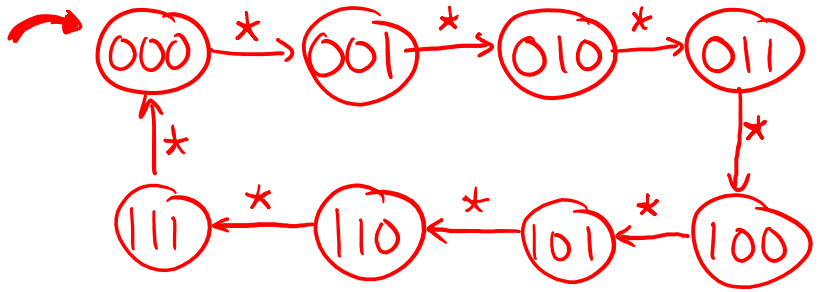
```
module LFSR #(parameter WIDTH=3) (Q, clk);

  output logic [WIDTH-1:0] Q;  // present state
  input  logic clk;            // clock input

  always_ff @(posedge clk)
    Q <= {Q[WIDTH-2:0], ~(Q[WIDTH-1] ^ Q[WIDTH-2])};

endmodule
```

take advantage of concatenate operator

Counters

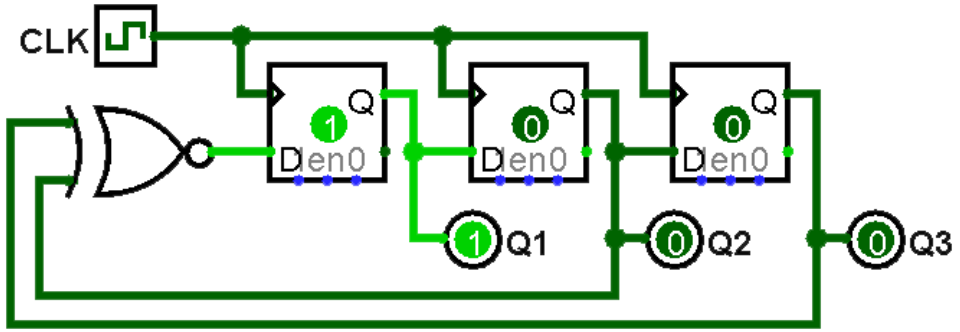
- A register that goes through a specific state sequence
  - More general than what you typically think of as a "counter"
- Examples:
  - *n-bit Binary Counter:* counts from 0 to 2^N-1 in binary
  - *Up Counter:* Binary value increases by 1
  - *Down Counter:* Binary value decreases by 1

- 3-bit binary up counter state diagram:



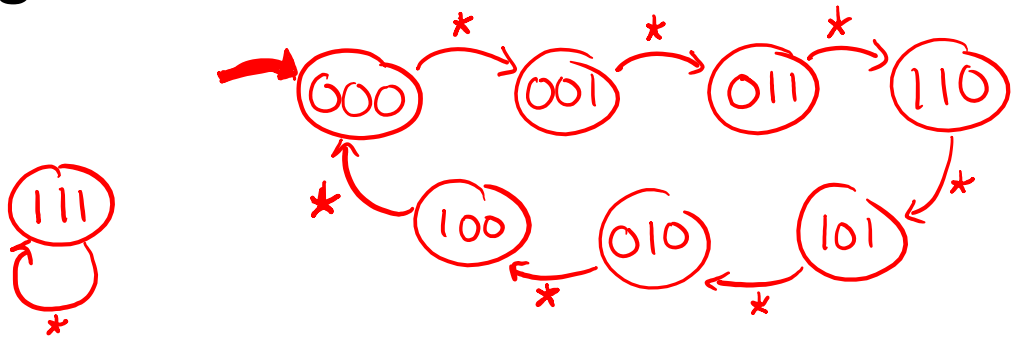
LFSR Revisited

- A LFSR is also a counter!
  - The logical combination determines the state sequence



Q2	Q3	xnor
0	0	1
0	1	0
1	0	0
1	1	1

- State diagram:



"pseudo-random" sequence of states determined by:
1. # of state bits
2. gate(s) used in feedback
3. which bits are connected to gate(s)

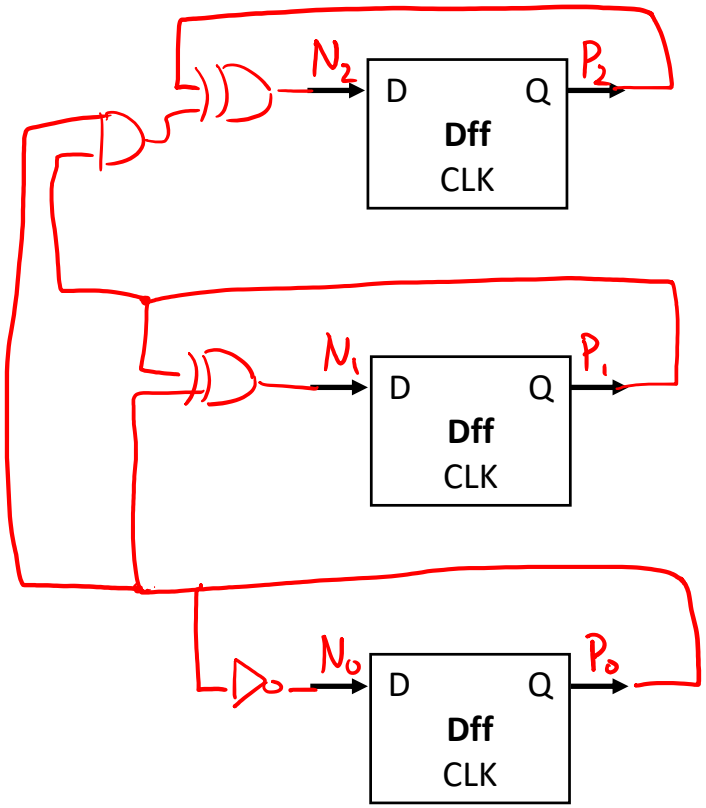
Binary Up-Counter Implementation

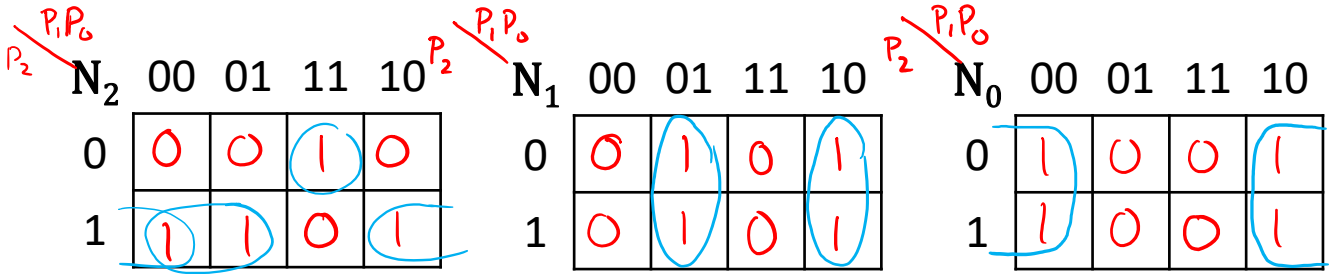
P_2	P_1	P_0	N_2	N_1	N_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

$N_0 = \overline{P_0}$

$N_1 = \overline{P_1}P_0 + P_1\overline{P_0}$
$= P_0 \oplus P_1$

$N_2 = P_2\overline{P_0} + P_2\overline{P_1} + \overline{P_2}P_1P_0$
$= P_2(\overline{P_0}+\overline{P_1}) + \overline{P_2}(P_1P_0)$
$= P_2(\overline{P_1P_0}) + \overline{P_2}(P_1P_0) = P_2 \oplus (P_1P_0)$





Complex Binary Counter

Load	Count	Action
0	0	Old Q
0	1	Up count
1	0	Parallel load (D)
1	1	Reset

$N_0 = \overline{P_0}$

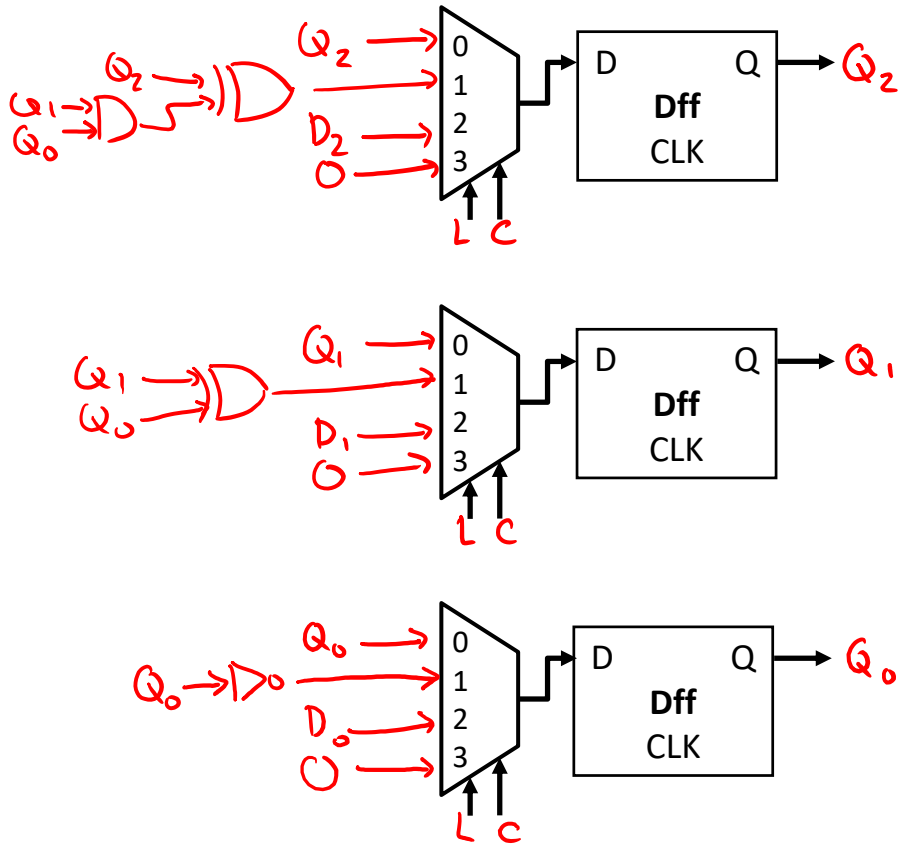
$N_1 = \overline{P_1}P_0 + P_1\overline{P_0}$

$= P_0 \oplus P_1$

$N_2 = P_2\overline{P_0} + P_2\overline{P_1} + \overline{P_2}P_1P_0$

$= P_2(\overline{P_0} + \overline{P_1}) + \overline{P_2}(P_1P_0)$

$= P_2(\overline{P_1P_0}) + \overline{P_2}(P_1P_0) = P_2 \oplus (P_1P_0)$



Up Counter in Verilog (no load)

```
module upcounter #(parameter WIDTH=8)
  (out, enable, reset, clk);

  output logic [WIDTH-1:0] out;
  input  logic enable, reset, clk;

  always_ff @(posedge clk) begin
    if (reset)
      out <= 0;
    else if (enable)
      out <= out + 1;
  end
endmodule  // upcounter
```