

# Intro to Digital Design

## Encoders, Decoders, Registers, Counters

**Instructor:** Justin Hsia

**Teaching Assistants:**

Caitlyn Rawlings

Emilio Alcantara

Naoto Uemura

Donovan Clay

Joy Jung

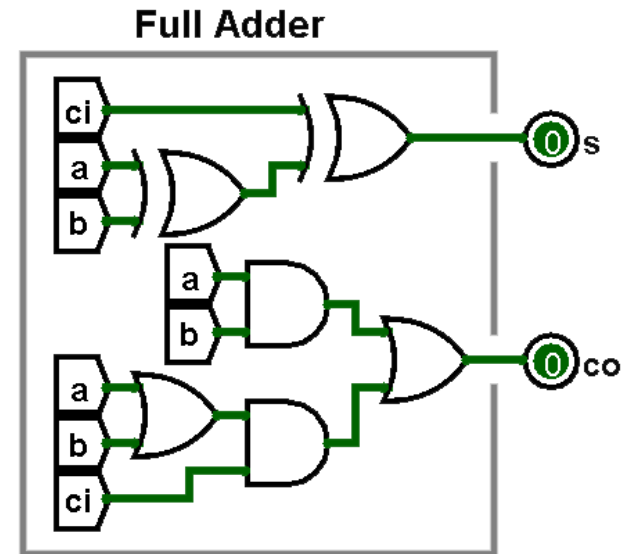
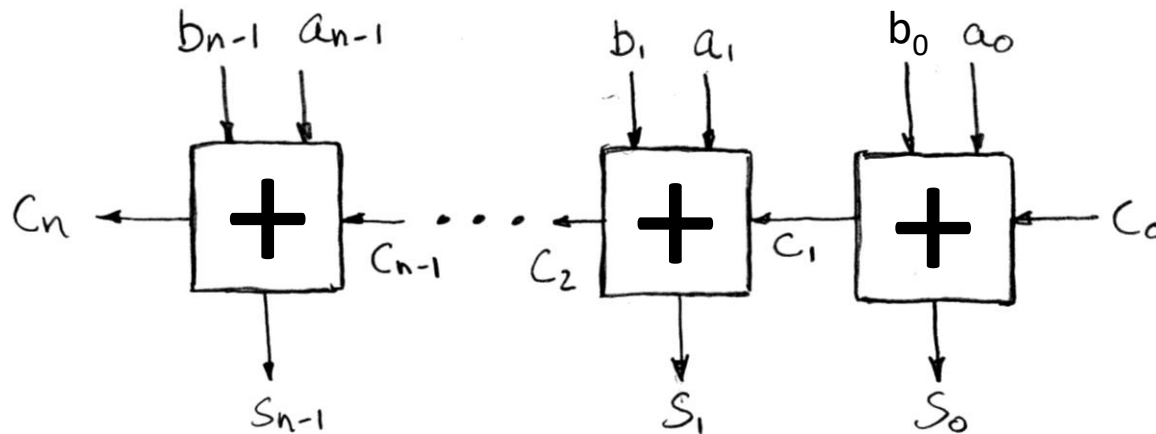
# Relevant Course Information

- ❖ 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: Files → Quizzes

# Practice Problem

Inputs  $A, B, c_0$  from registers (show up at  $t_{C2Q}$ ).  
Output  $S$  goes directly to a register input.

- ❖ For an  $n$ -bit ripple-carry adder, what is the shortest and longest time  $S$  changes after each clock cycle?
  - Assume all gates have a delay of 1 ns and  $c_0 = \text{constant } (0/1)$
  - Use variables for all other timing values



# Outline

- ❖ **Circuit Routing Elements**
- ❖ Register Revisited

# Standard Circuit Routing Elements

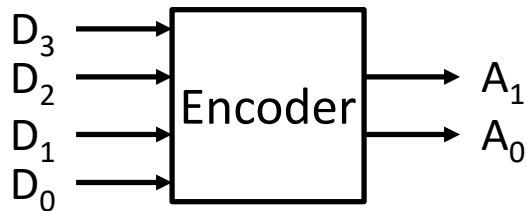
- ❖ Multiplexor (mux)
  - Pass one of  $N$  inputs to single output
- ❖ **Simple Encoder**
  - One of  $N$  inputs is active and output tells you which one (in binary)
- ❖ **1-of- $N$  Binary Decoder**
  - Interpret binary input to assert one of  $N$  output wires
- ❖ **Demultiplexer (demux)**
  - Pass 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	1	0		
0	1	0	0		
1	0	0	0		

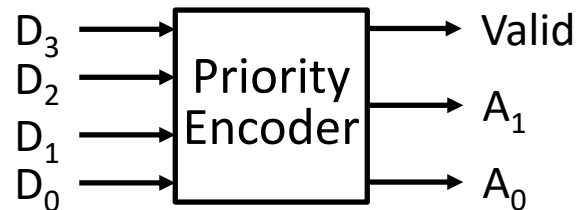
## ❖ Two issues:

- 1) What if multiple inputs are hot?
- 2) What if no inputs are hot?

# 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

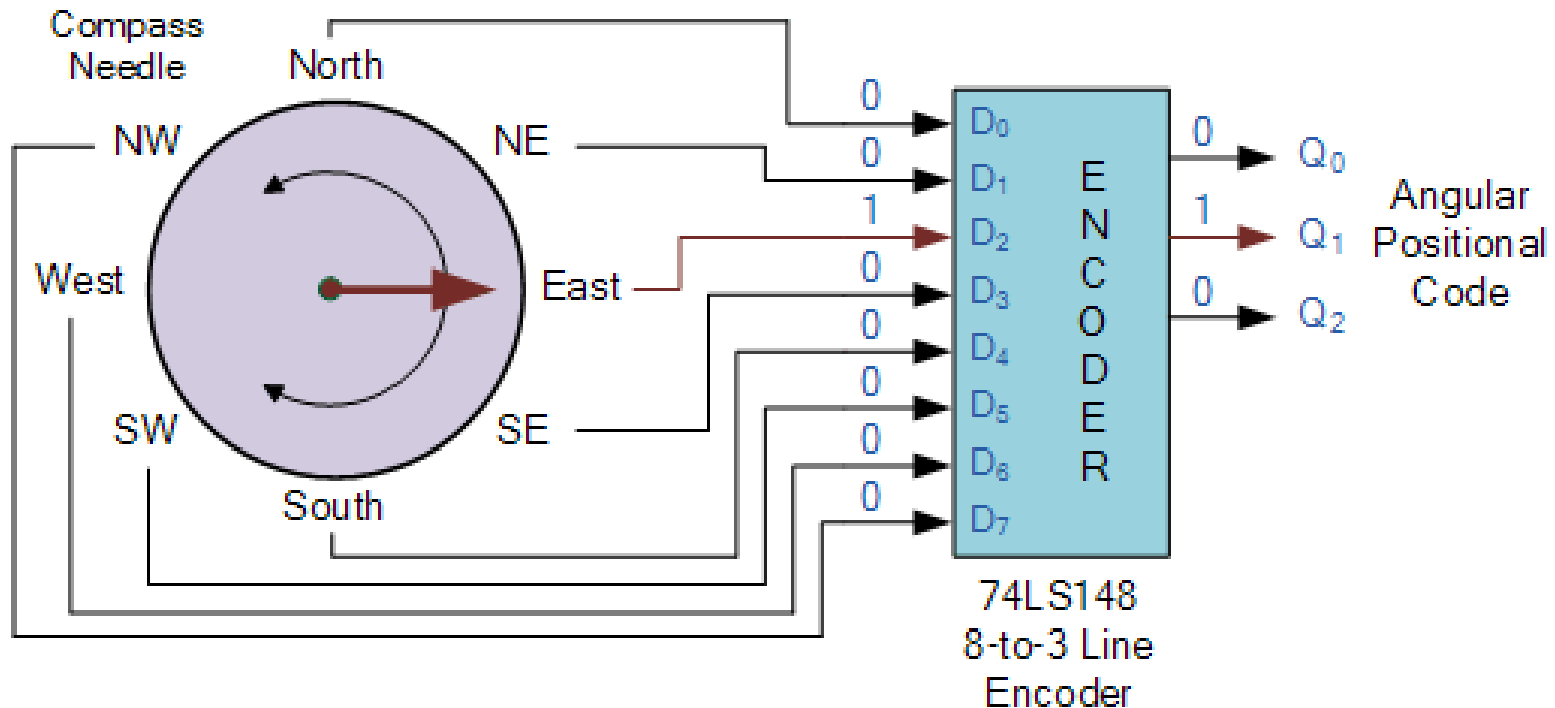
$D_3$	$D_2$	$D_1$	$D_0$	$A_1$	$A_0$	Valid
0	0	0	0			
0	0	0	1			
0	0	1	X			
0	1	X	X			
1	X	X	X			





# Encoder Examples

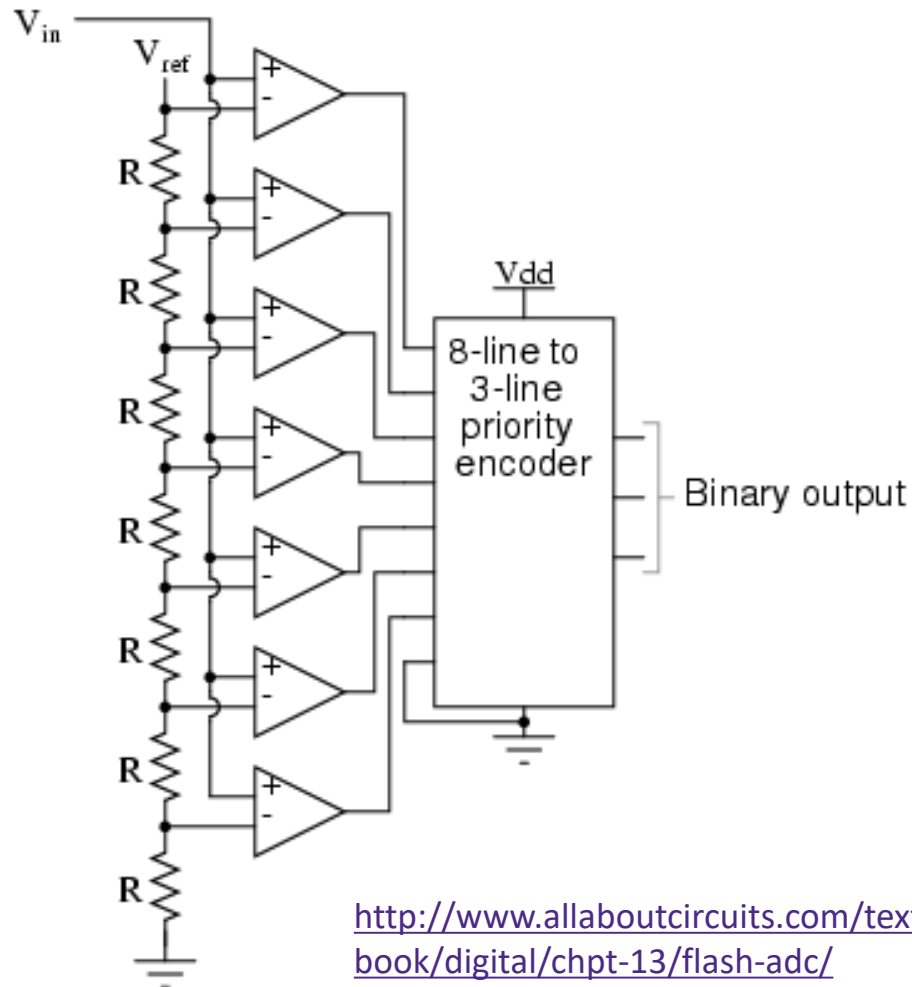
## ❖ Navigation (Compass) Encoder



[http://www.electronics-tutorials.ws/combination/comb\\_4.html](http://www.electronics-tutorials.ws/combination/comb_4.html)

# Encoder Examples

## ❖ Analog-to-Digital Converter (ADC)

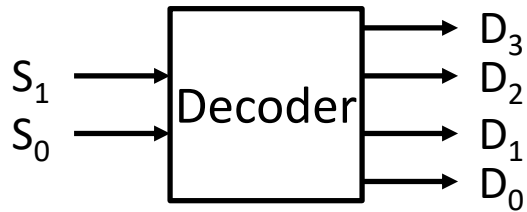


# Decoder

- ❖ 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



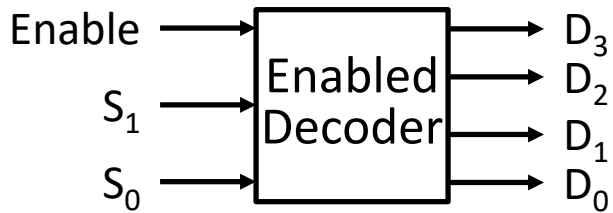
$S_1$	$S_0$	$D_3$	$D_2$	$D_1$	$D_0$
0	0				
0	1				
1	0				
1	1				

## ❖ Issue:



# Enabled Decoder

- ❖ Only have active output when Enable signal is high



Enable	S <sub>1</sub>	S <sub>0</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
0	X	X				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

# Enabled Decoder in Verilog

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

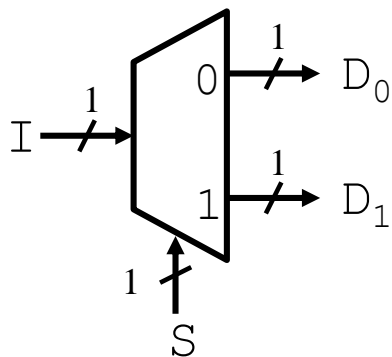
    always_comb begin

end

endmodule
```

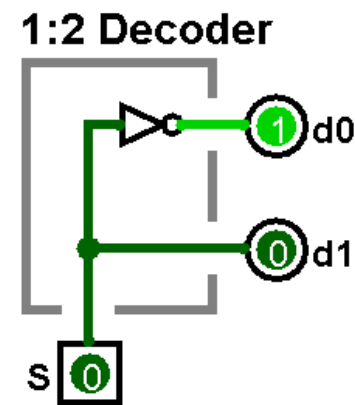
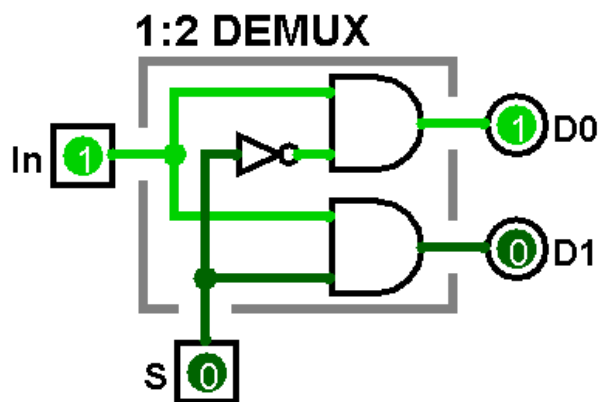
# Decoder Examples: Demultiplexer

## ❖ 1-bit 1-to-2 DEMUX:



## ❖ Truth Table:

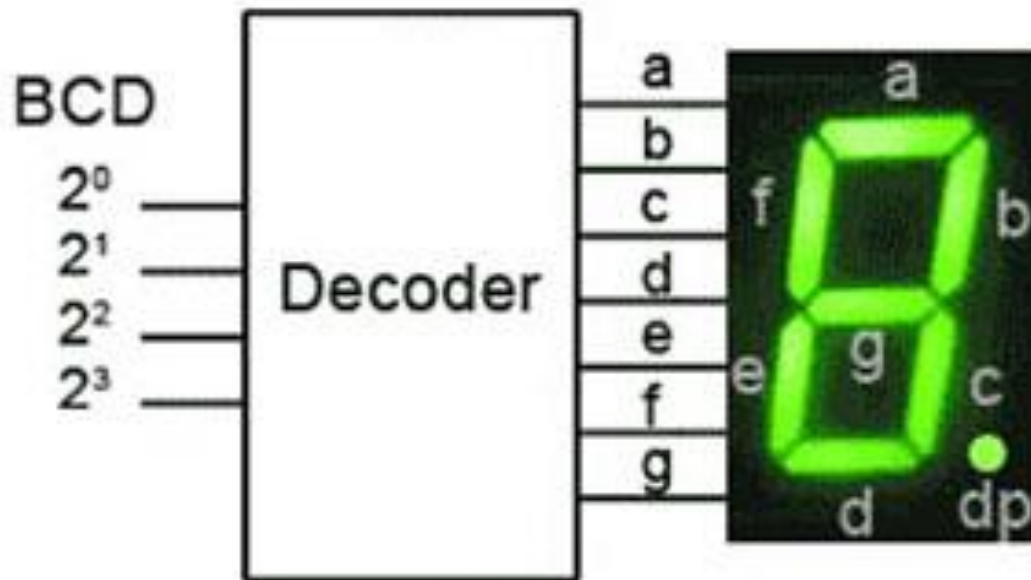
S	I	D <sub>1</sub>	D <sub>0</sub>
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



- 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!



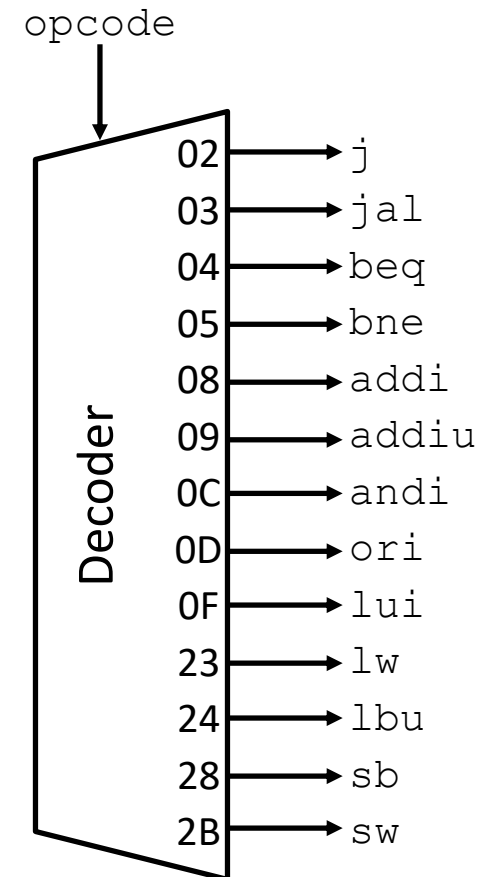
<http://www.learnabout-electronics.org/Digital/dig44.php>



# Decoder Examples

- ❖ MIPS instruction decoder
  - Upper 6 bits of a 32-bit MIPS instruction
  - Part of the control portion of a CPU

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

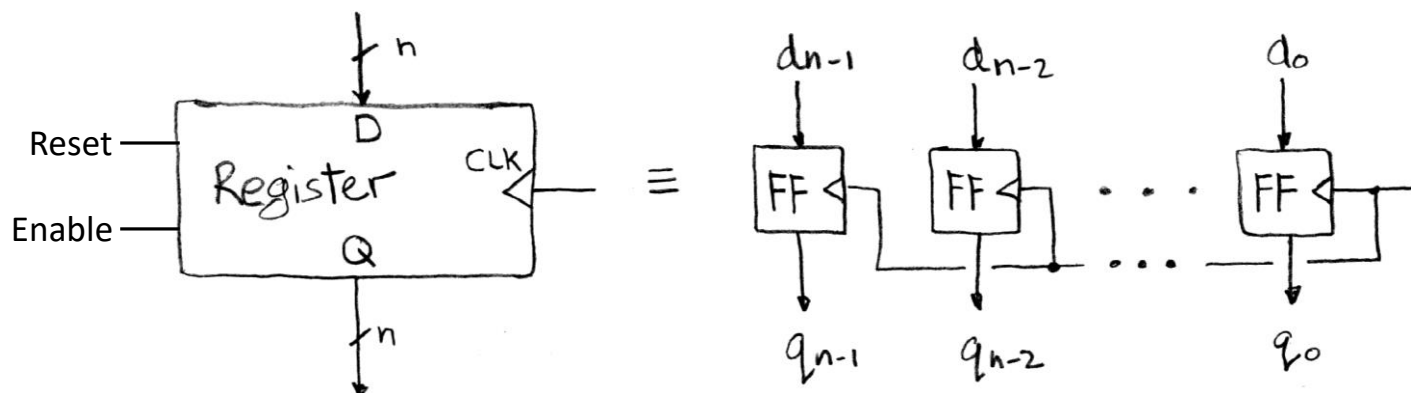


# Technology Break

# Outline

- ❖ Circuit Routing Elements
- ❖ **Register Revisited**

# State Element Revisited: Register

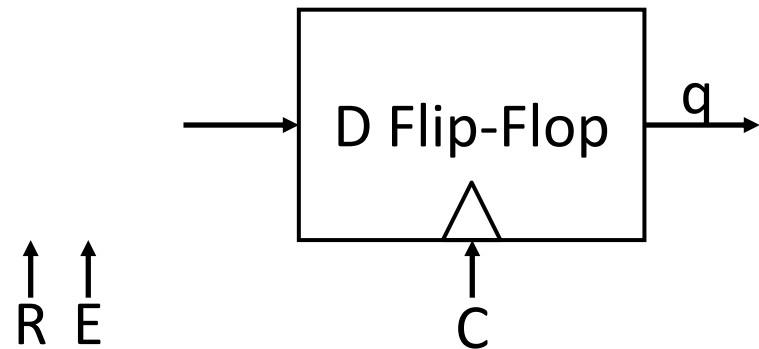


- ❖  $n$  instances of flip-flops together
  - One for every bit in input/output bus width
- ❖ Desired behaviors (synchronous)
  - Output  $Q$  resets to zero when  $\text{Reset}$  signal is high
  - Hold current value unless  $\text{Enable}$  signal is high

# Controlled Register

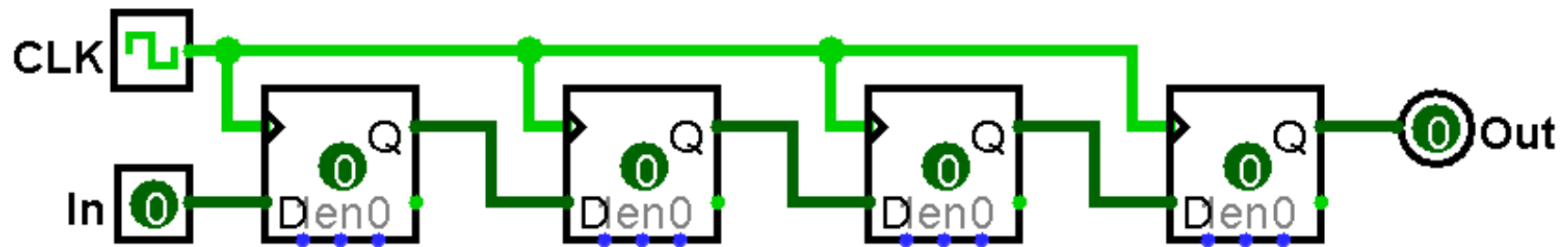
- ❖ Here using shorthand C (clock), R (reset), E (enable)

Reset	Enable	Action
0	0	q = _____
0	1	q = _____
1	0	q = _____
1	1	q = _____



# Shift Register

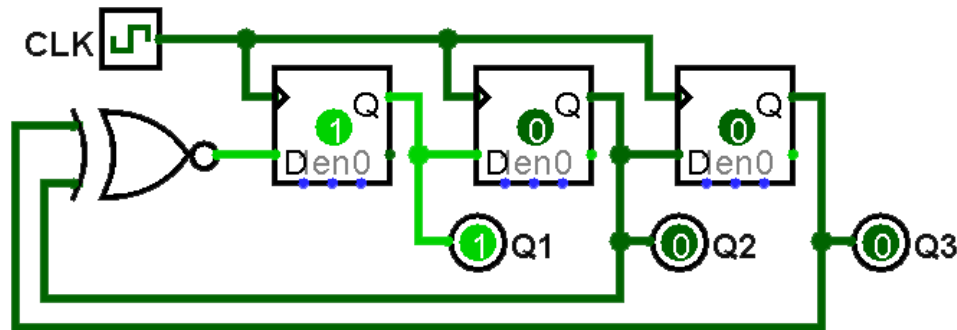
- ❖ Register that shifts the binary values in one or both directions



- ❖ 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

# Linear Feedback Shift Register (LFSR)

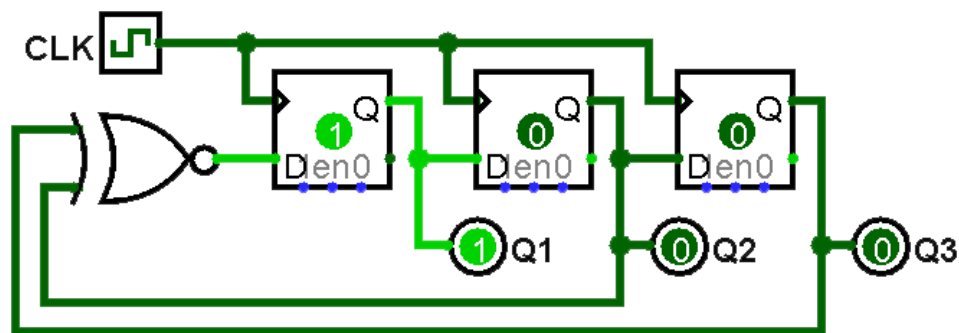
- ❖ Shift register input is a logical combination of the current state bits:



- ❖ 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
```

```
endmodule
```

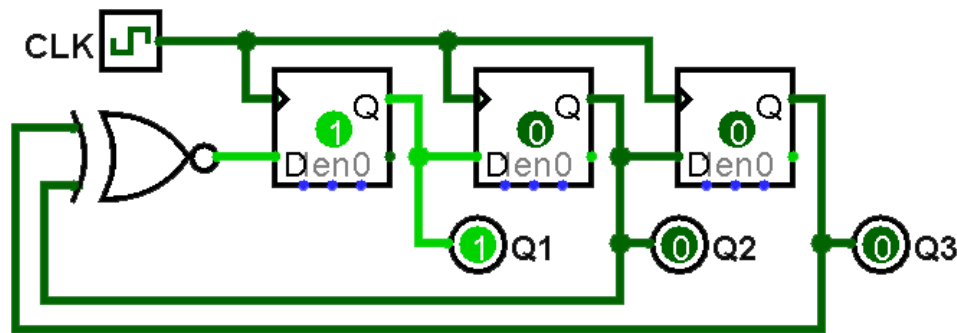


# Counters

- ❖ **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

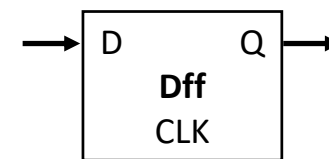
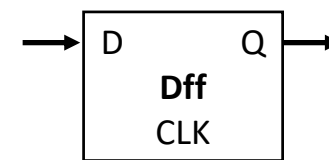
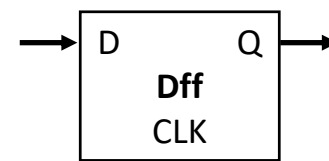
- ❖ A LFSR is also a counter!
  - The logical combination determines the state sequence



- ❖ State diagram:

# Binary Up-Counter Implementation

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



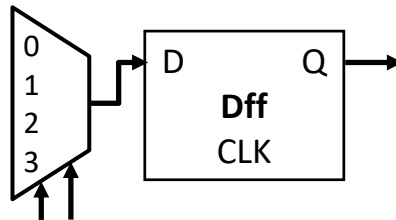
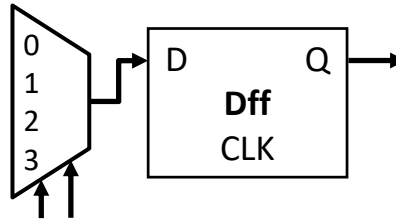
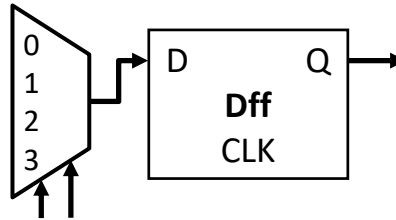
$N_2$	00	01	11	10
0				
1				

$N_1$	00	01	11	10
0				
1				

$N_0$	00	01	11	10
0				
1				

# Complex Binary Counter

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



# 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
```