Intro to Digital Design
L7: Encoders, Decoders, Registers, and Counters (oh my!)

Instructor: Naomi Alterman

Teaching Assistants:

Derek de Leuw
Isabel Froelich
Kevin Hernandez
Sathvik Kanuri
Aadithya Manoj

Administrivia

- Lab 7 – Useful Components
  - Modifying Lab 6 game to play against a tunable computer opponent!
  - Implement some common circuit elements

- 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

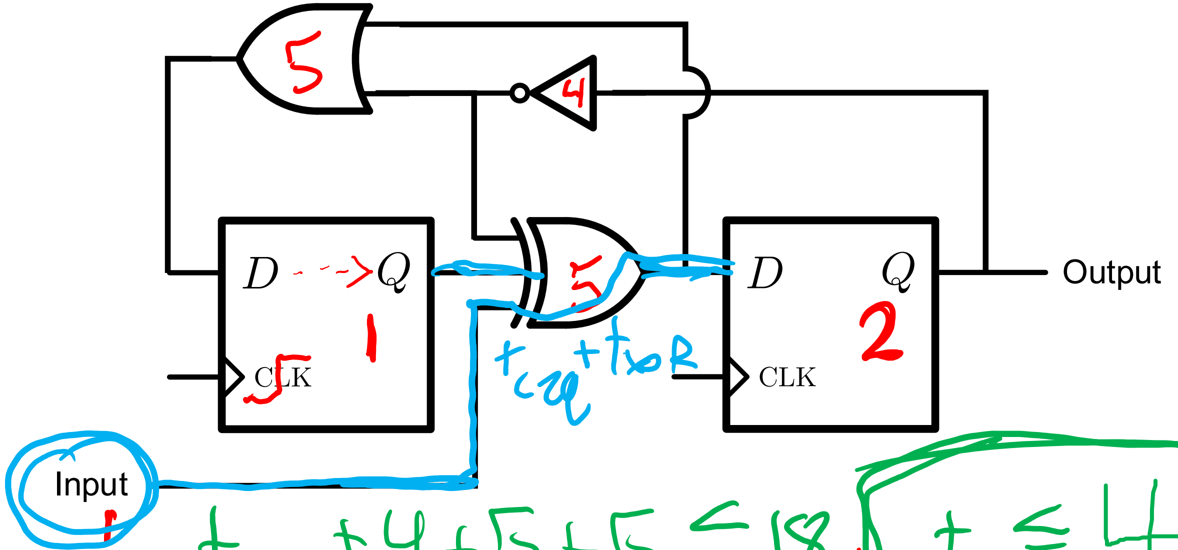
SDS Timing Question

$t_{hold} \leq t_{CLmin}$

$t_{CLmax} \leq t_{period} - t_{setup}$

$20 - 2 = 18$

- The circuit below has the following timing parameters
  - $t_{period} = 20\ ns$, $t_{setup} = 2\ ns$
  - $t_{XOR} = t_{OR} = 5\ ns$, $t_{NOT} = 4\ ns$
  - Input changes 1 ns after clock trigger



- What is the max t_{C2Q}?

$t_{c2q} + t_{not} + t_{XOR} + t_{OR} \leq 18$

$t_{c2q} + 4 + 5 + 5 \leq 18$ (14)

$t_{c2q} \leq 4$

- If $t_{C2Q} = 3\ ns$, what is the max t_{hold}?

$t_{input} + t_{XOR} \geq t_{hold}$

$6 \geq t_{hold}$

Outline

- **Circuit Routing Elements**
- Register Revisited

Standard Circuit Routing Elements

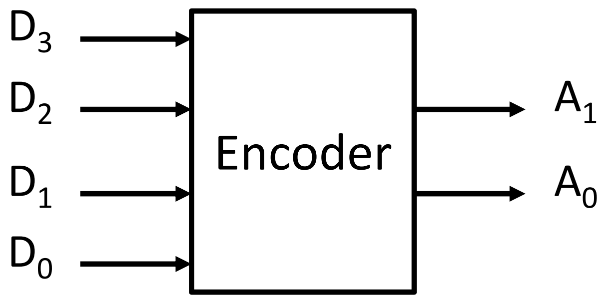
- Multiplexor (mux)
  - Pass one of N inputs to single output
- **Simple Encoder**
  - One of N inputs is active and output tells you which one (in binary)
- **1-of-N Binary Decoder**
  - Interpret binary input to assert one of N output wires
- **Demultiplexer (demux)**
  - Pass single input onto one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

+ Arbiter

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 **binary encoder** is a **one-hot** to **binary** converter
  - One-hot means at most only one line of the input bus (out of $m \leq 2^n$) will be high
  - Output is the index/"address" of said line in the bus, as an n bit binary number
  - Referred to as an m:n encoder (read as "m-to-n")



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

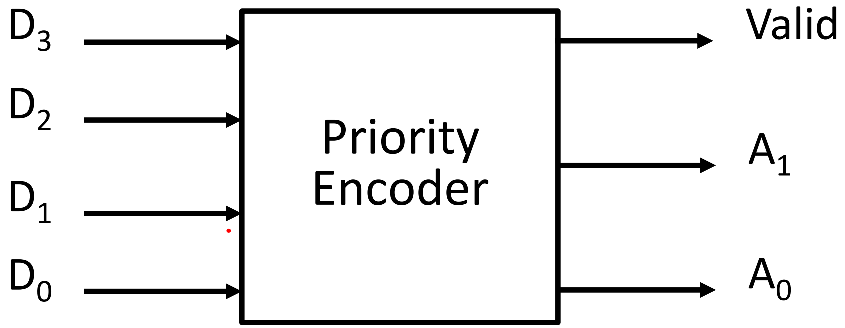
$A_0 = D_1 + D_3$

$A_1 = D_2 + D_3$

Priority Encoder

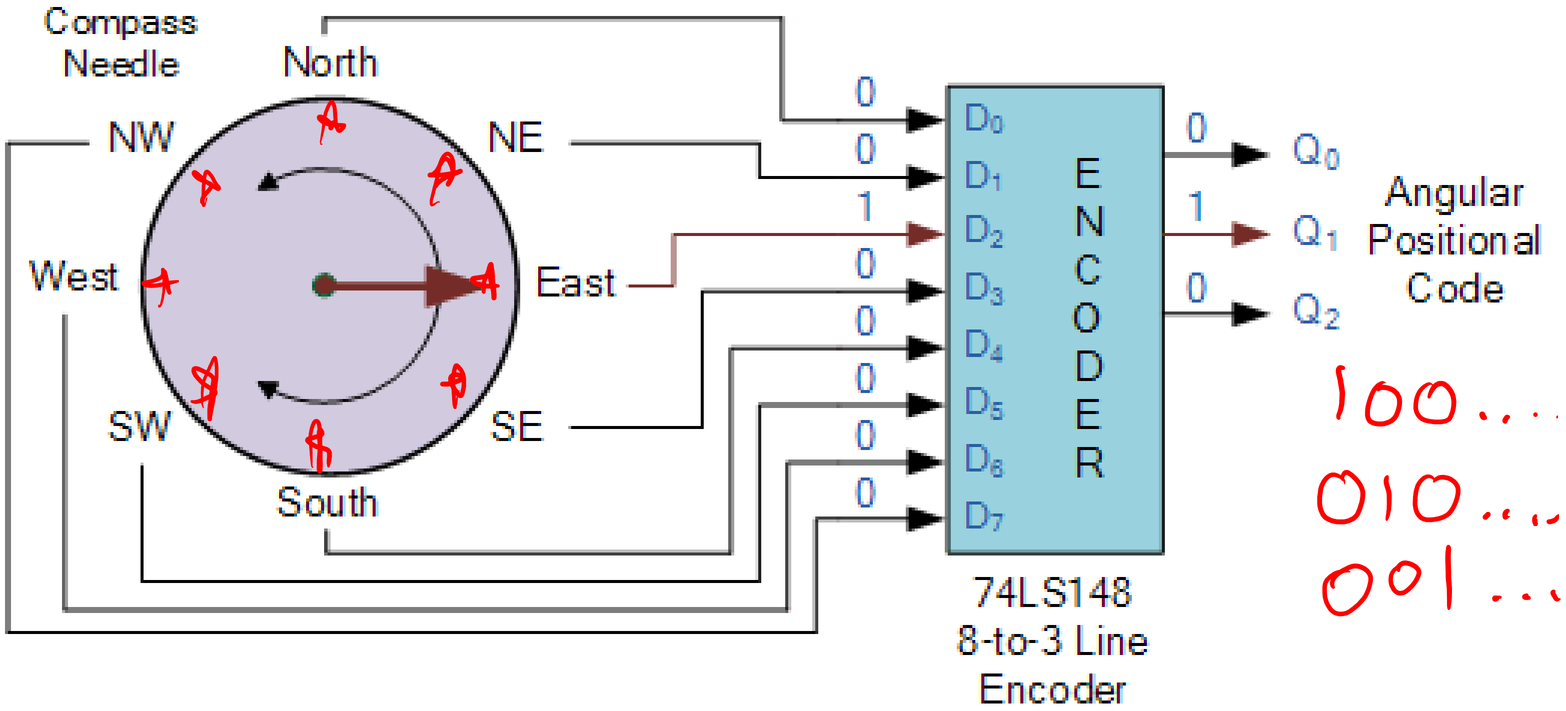
- Use for inputs that might not be one-hot. Most-significant bit *gets priority*/“wins”
- Add an output to identify when at least 1 input active

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

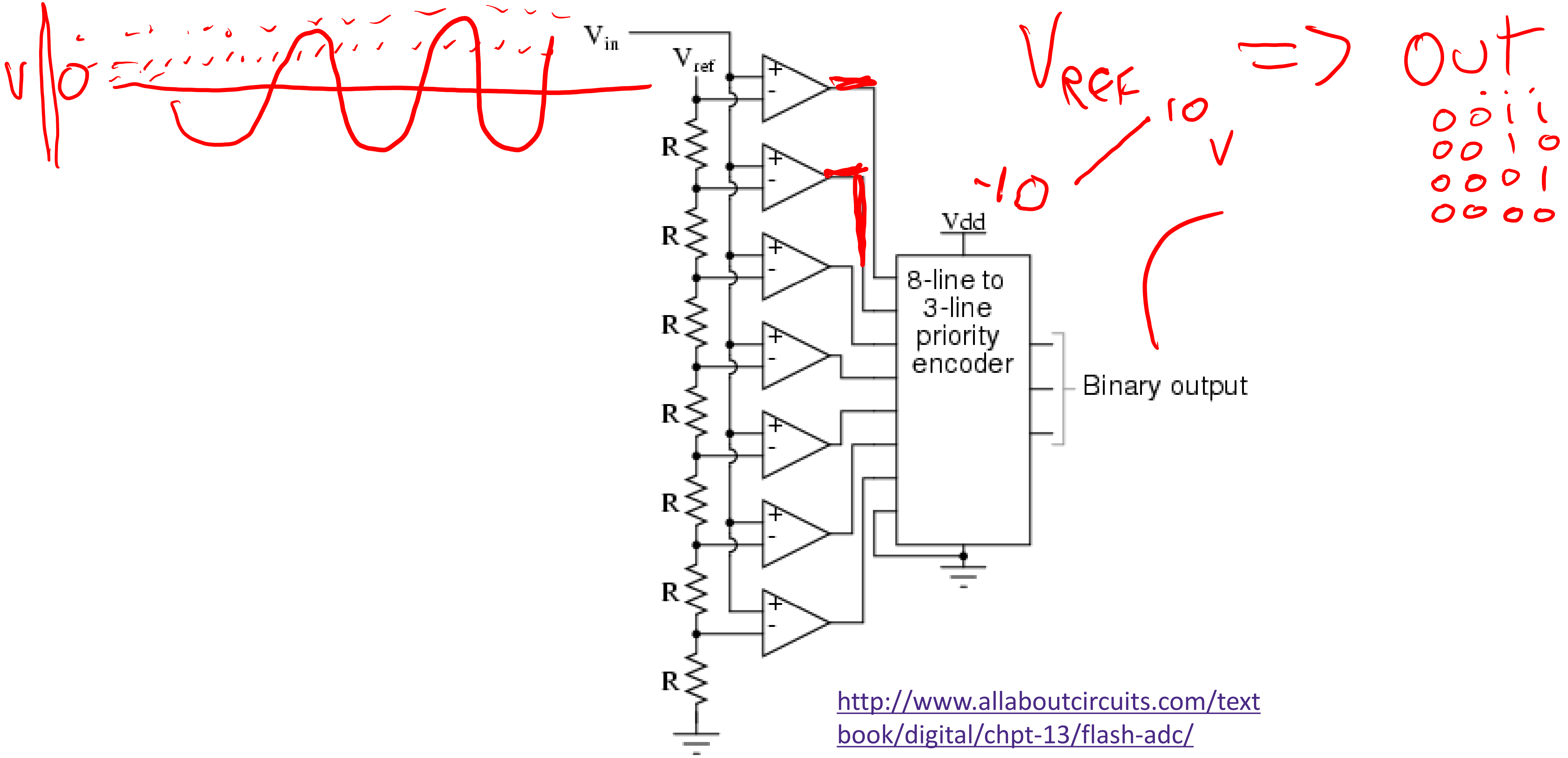


100 .... North (000)
010 .... NorEast (001
001 .... East (010)

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

Encoder Examples

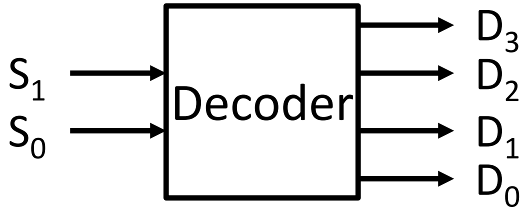
- Analog-to-Digital Converter (ADC)



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

Decoder

D[s]=1 D[...]=0



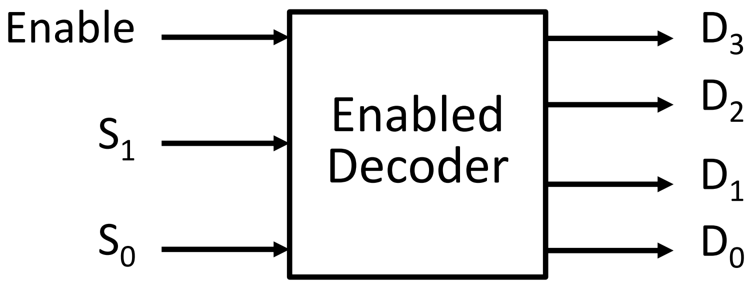
- A device or circuit that converts or interprets information from an encoded format
  - Examples: binary to decimal, CPU instruction decoder, video decoder (analog to digital)

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

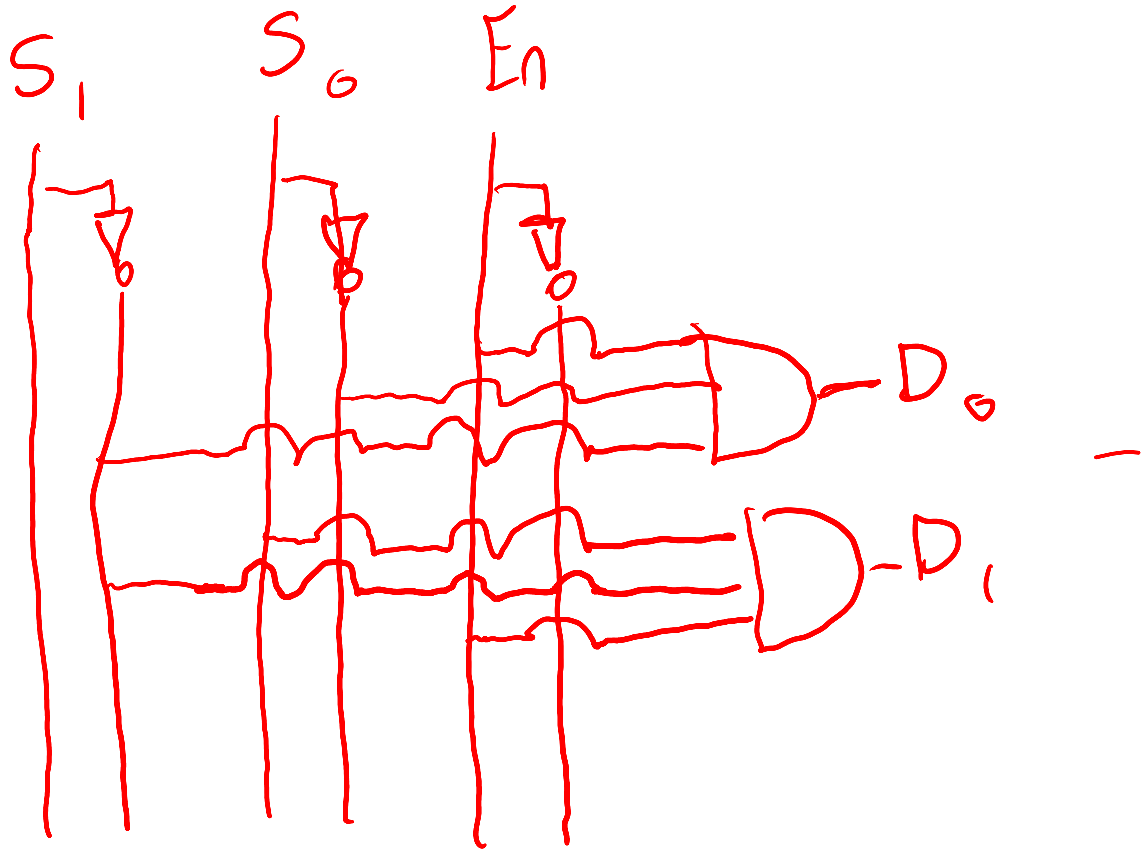
- A **binary decoder** is a binary to one-hot converter
  - n input bits serve as bit number or "address" specifier
  - Only corresponding output out of $m \leq 2^n$ will be asserted
  - Referred to as an n:m decoder (read as "n-to-m")

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



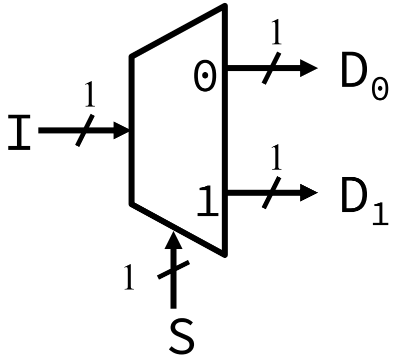
Enabled Decoder in Verilog

2:4

```
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

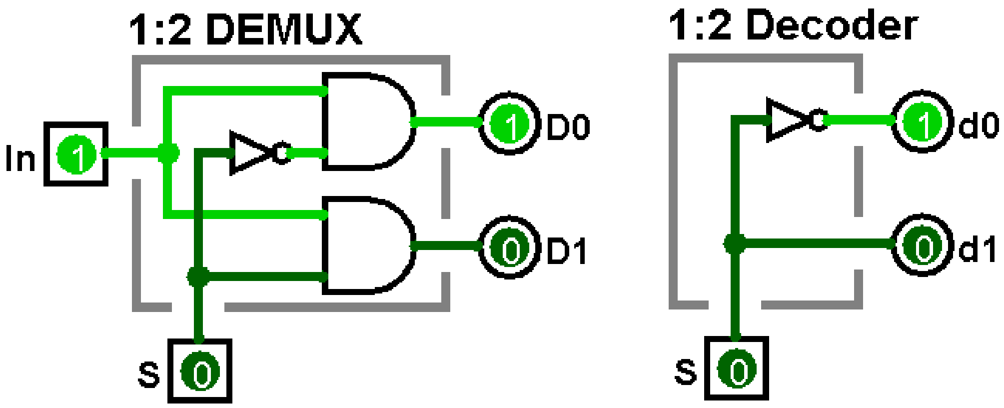
- **1-bit 1-to-2 DEMUX:**



- **Truth Table:**

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

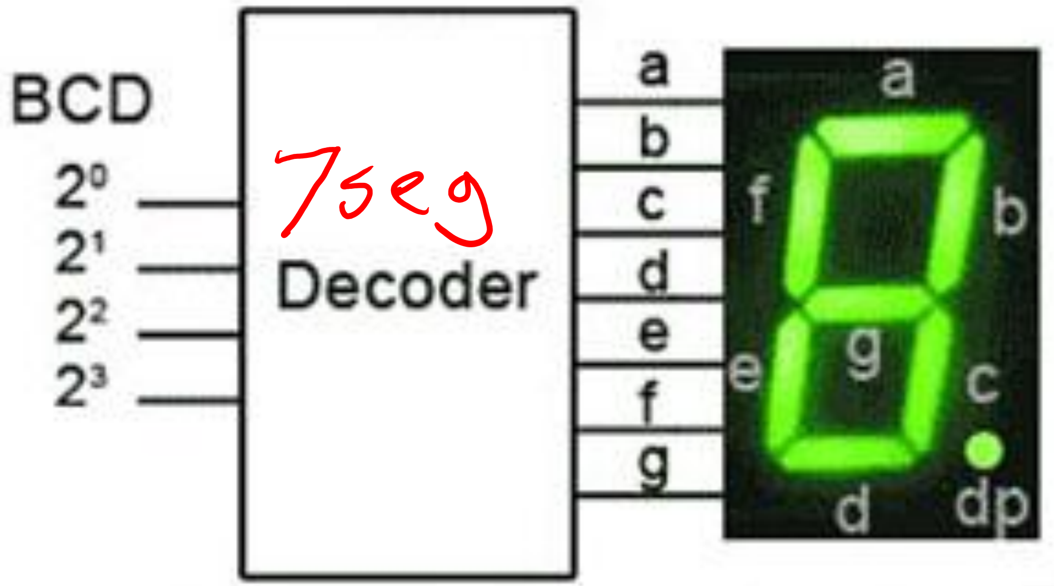
enable sig. for each output bus



- 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!

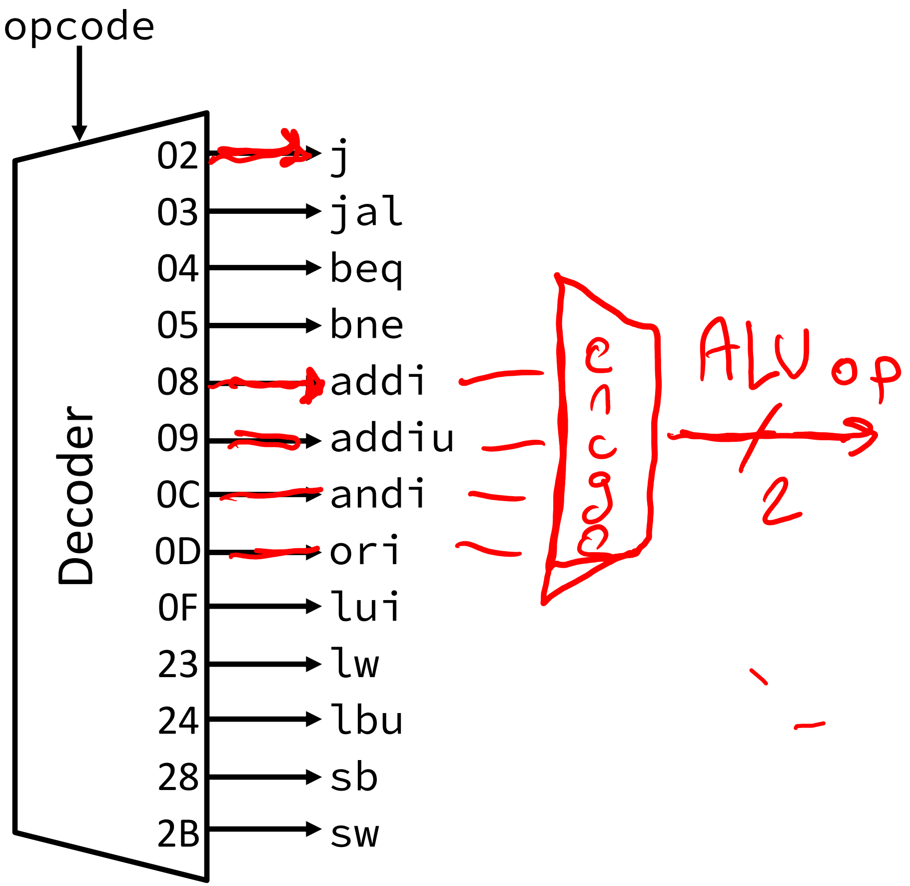


http://www.learnabout-electronics.org/Digital/dig44.php

Decoder Examples

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

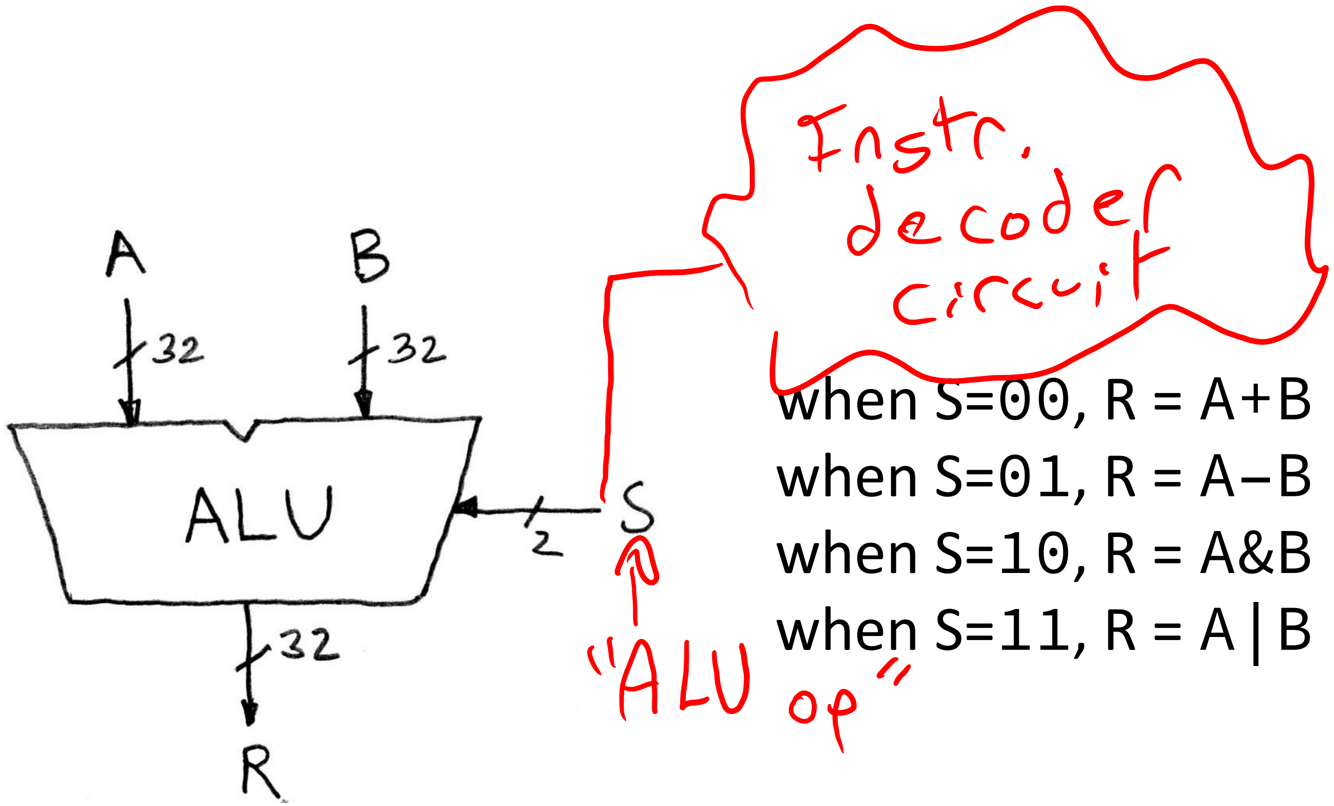
Instruction	Name	Opcode
addi	Add Imm.	001000
addiu	Add Imm. Unsigned	001001
andi	And Imm.	001100
beq	Branch On Equal	000100
bne	Branch On Not Equal	000101
j	Jump	000010
jal	Jump and Link	000011
lbu	Load Byte Unsigned	100100
lui	Load Upper Imm.	001111
lw	Load Word	100011
ori	Or Imm.	001101
sb	Store Byte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



Arithmetic and Logic Unit (ALU)

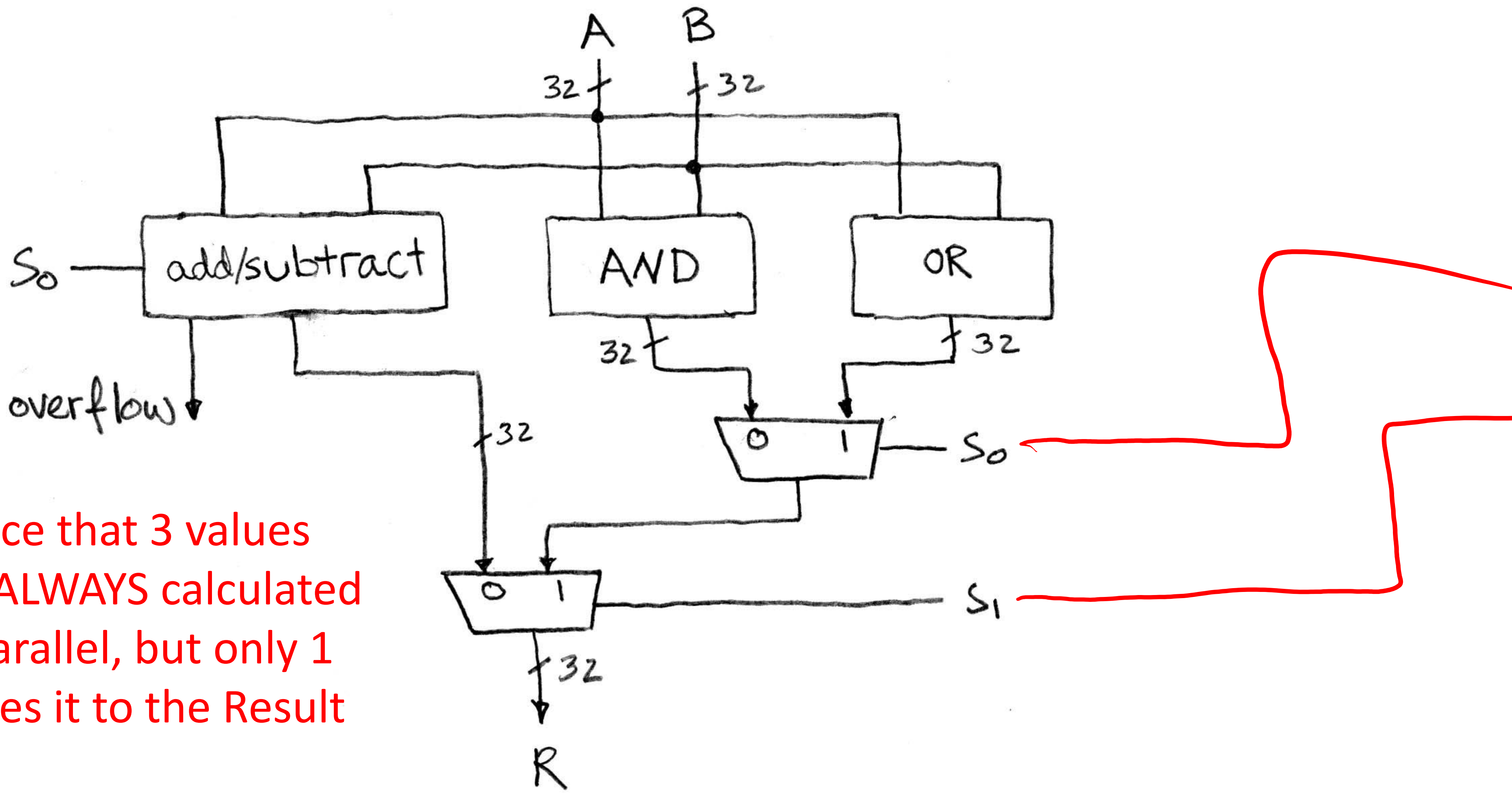
- Decoded instruction signals go to special logic block called the "Arithmetic and Logic Unit" (ALU)
  - Here's an easy one that does ADD, SUB, bitwise AND, and bitwise OR (for 32-bit numbers)

- **Schematic:**



when S=00, R = A+B
when S=01, R = A−B
when S=10, R = A&B
when S=11, R = A|B

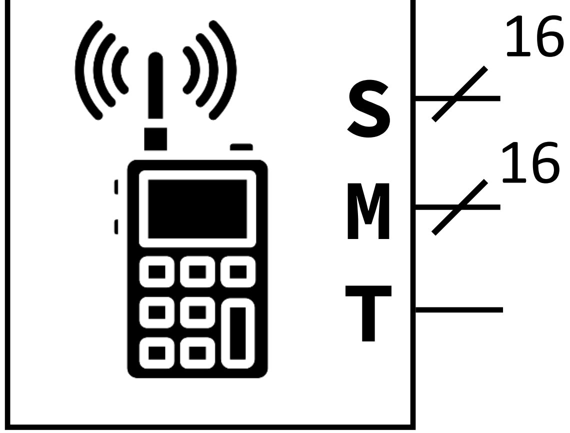
Simple ALU Schematic

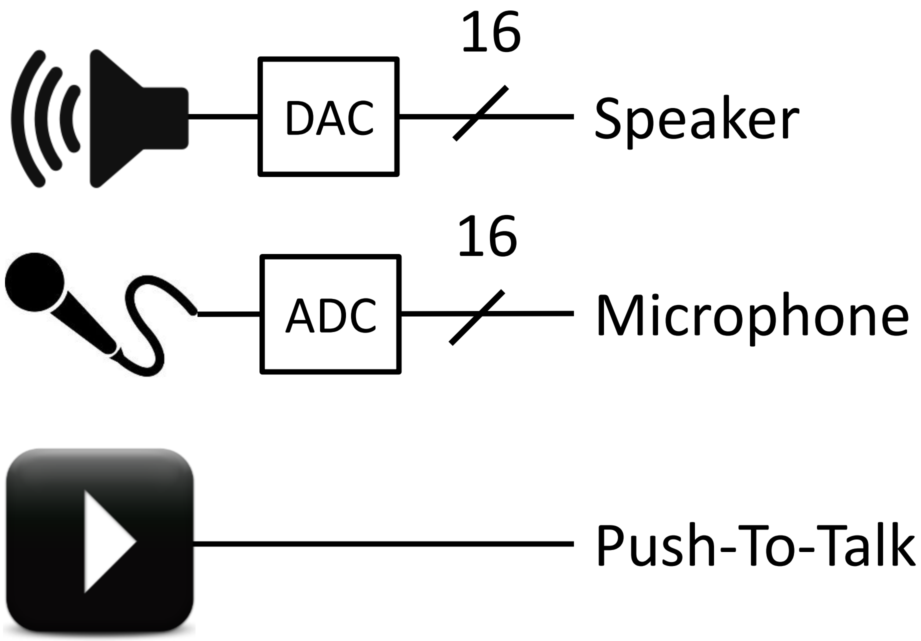


Notice that 3 values are ALWAYS calculated in parallel, but only 1 makes it to the Result

Design Example: Walkie-Talkie System

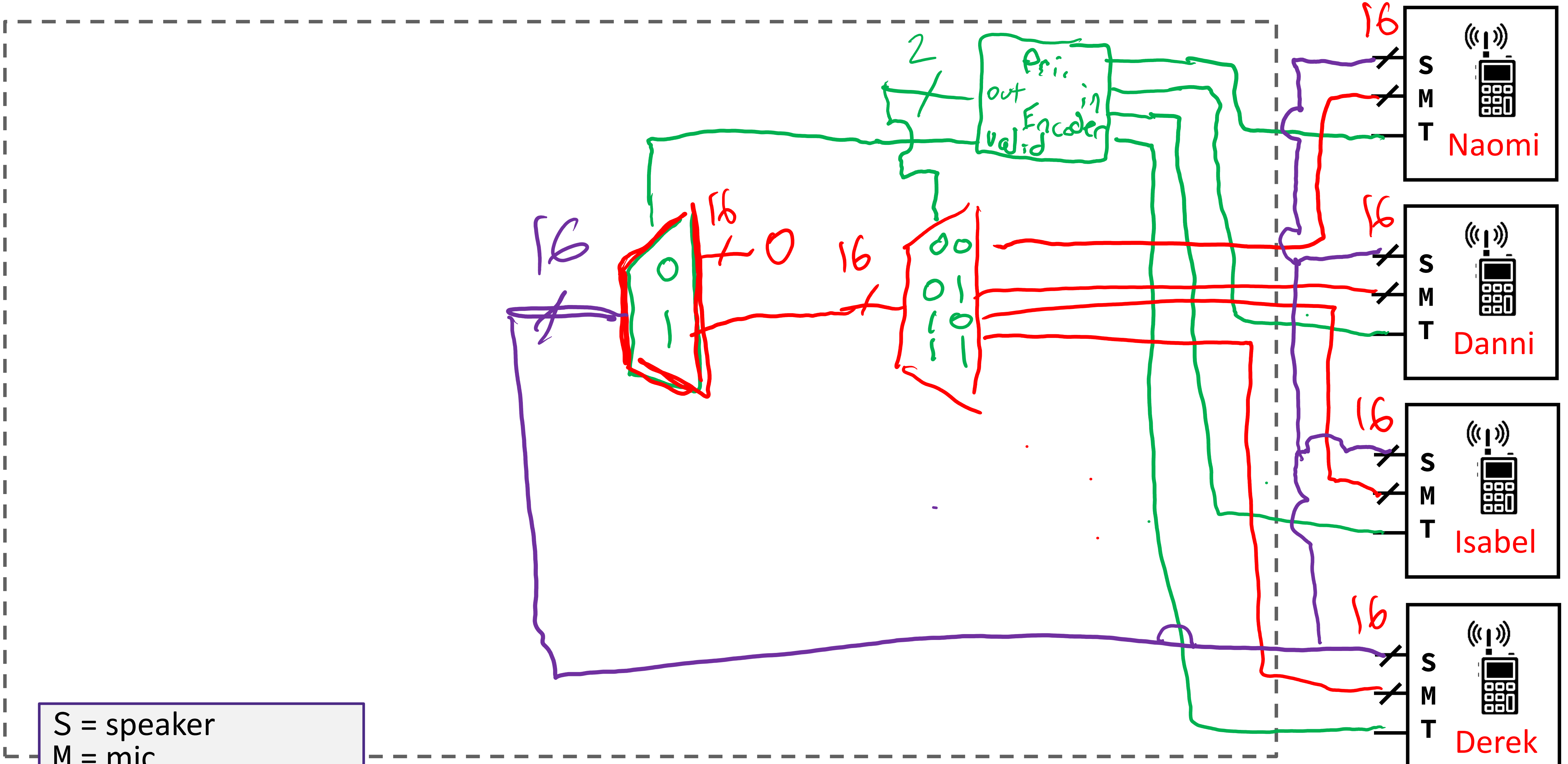
- Design "base station" for a simple closed voice communication system
  - Multiple listeners, one talker at a time





Walkie-Talkie Base Station

(All S/M 16-bit)



S = speaker
M = mic
T = push-to-talk

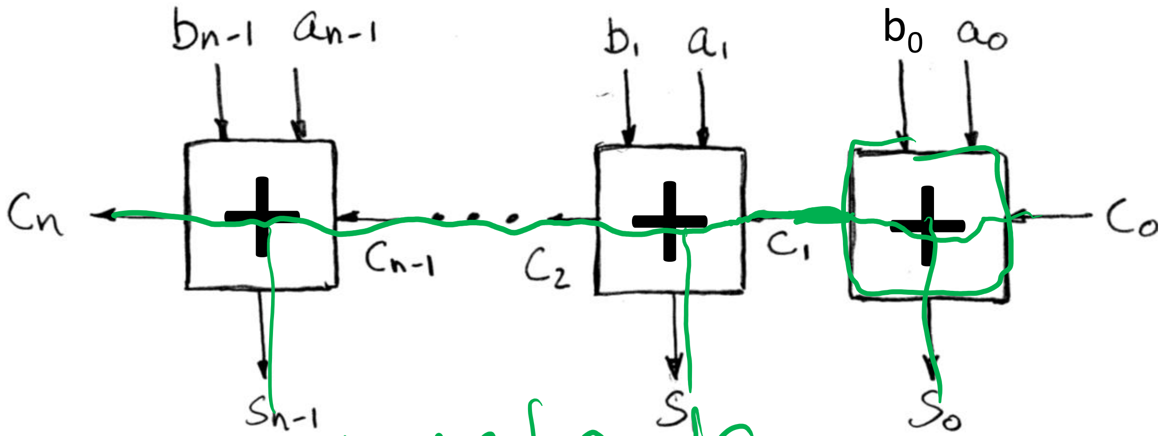
Miso Moment

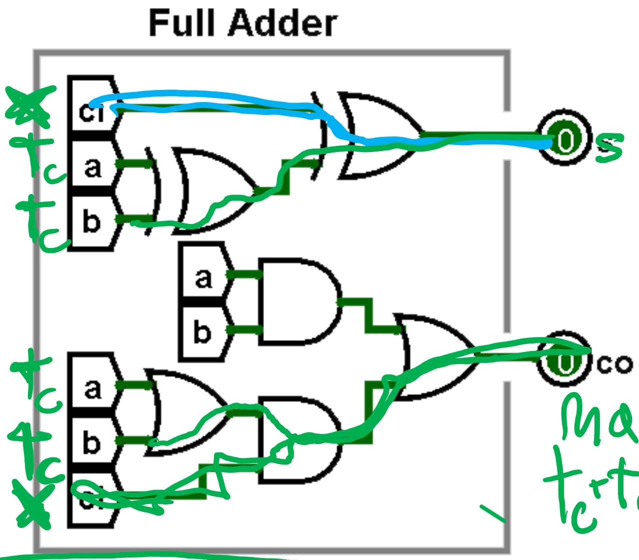
More SDS Practice

$t_{hold} \leq t_{CLmin}$

$t_{CLmax} \leq t_{period} - t_{setup}$

- 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





early: $t_{C2Q} + 1ns$

stage	C_o	S_n
1	t_c+3	t_c+2
2	t_c+5	t_c+4
3	t_c+7	t_c+6

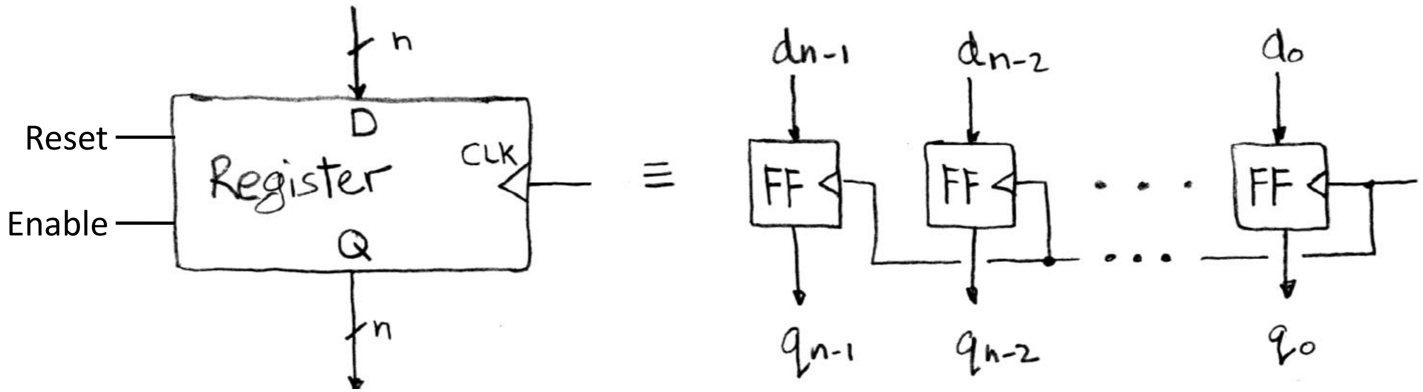
$S_n = t_c + 2n$

max($t_c + t_{OR} + t_{And} + t_{OR}$, $x + t_{and} + t_{OR}$)

Outline

- Circuit Routing Elements
- **Register Revisited**

State Element Revisited: Register



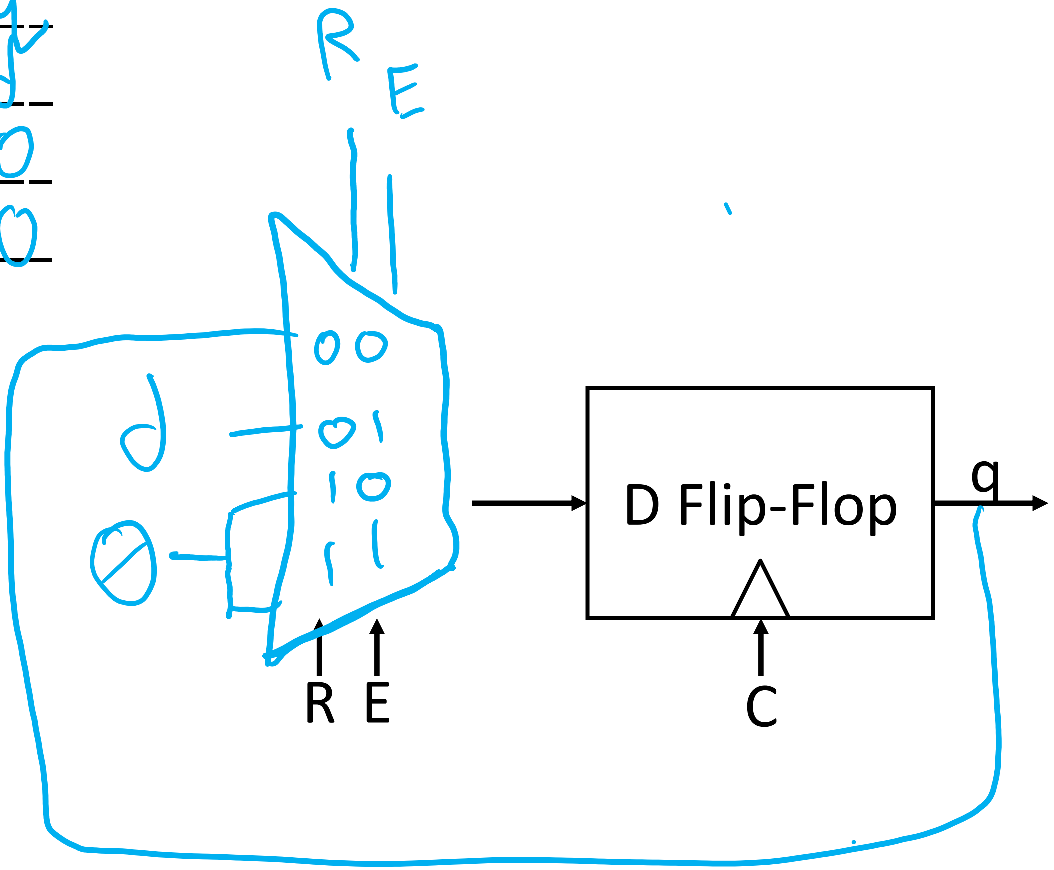
- n instances of flip-flops together
  - One for every bit in input/output bus width

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

Controlled Register

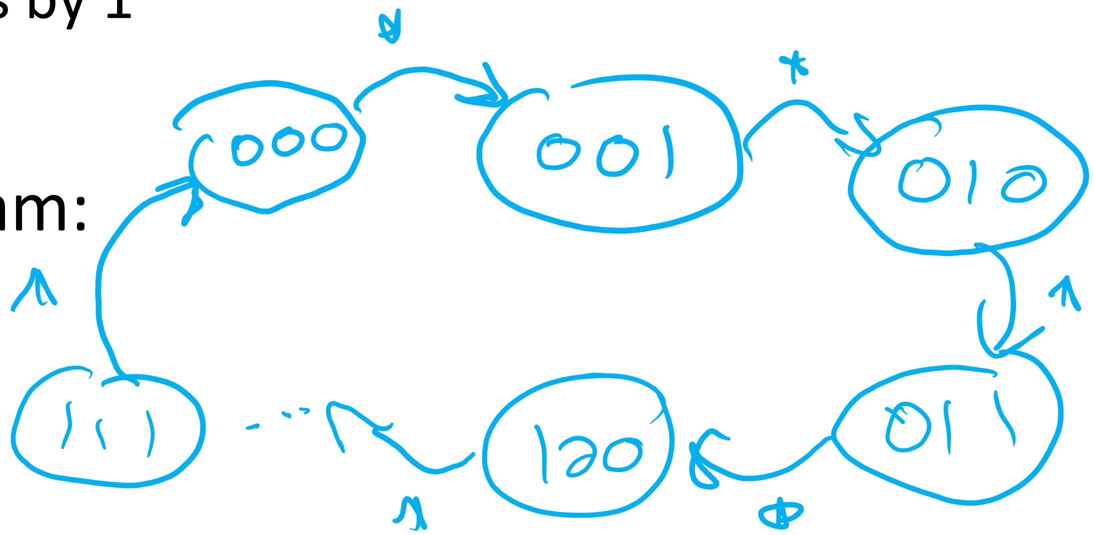
- Here using shorthand C (clock), R (reset), E (enable)

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



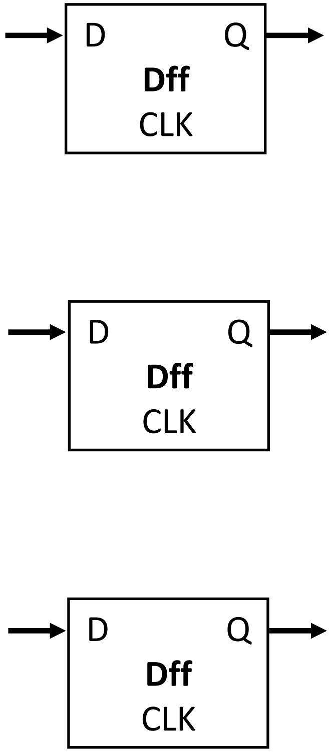
Counters

- 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:



Efficient 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				

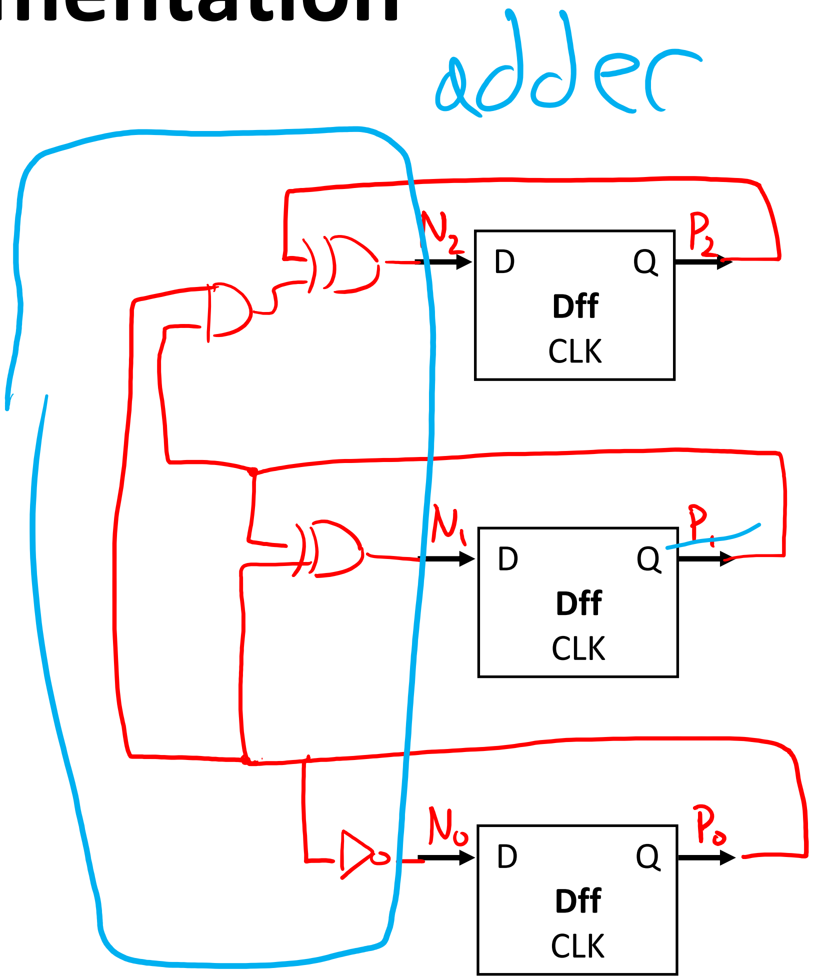
Efficient Binary Up-Counter Implementation

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

$N_0 = \overline{P_0}$

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

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



N_2 (P_2 \ P_1P_0):

	00	01	11	10
0	0	0	1	0
1	1	1	0	1

N_1 (P_2 \ P_1P_0):

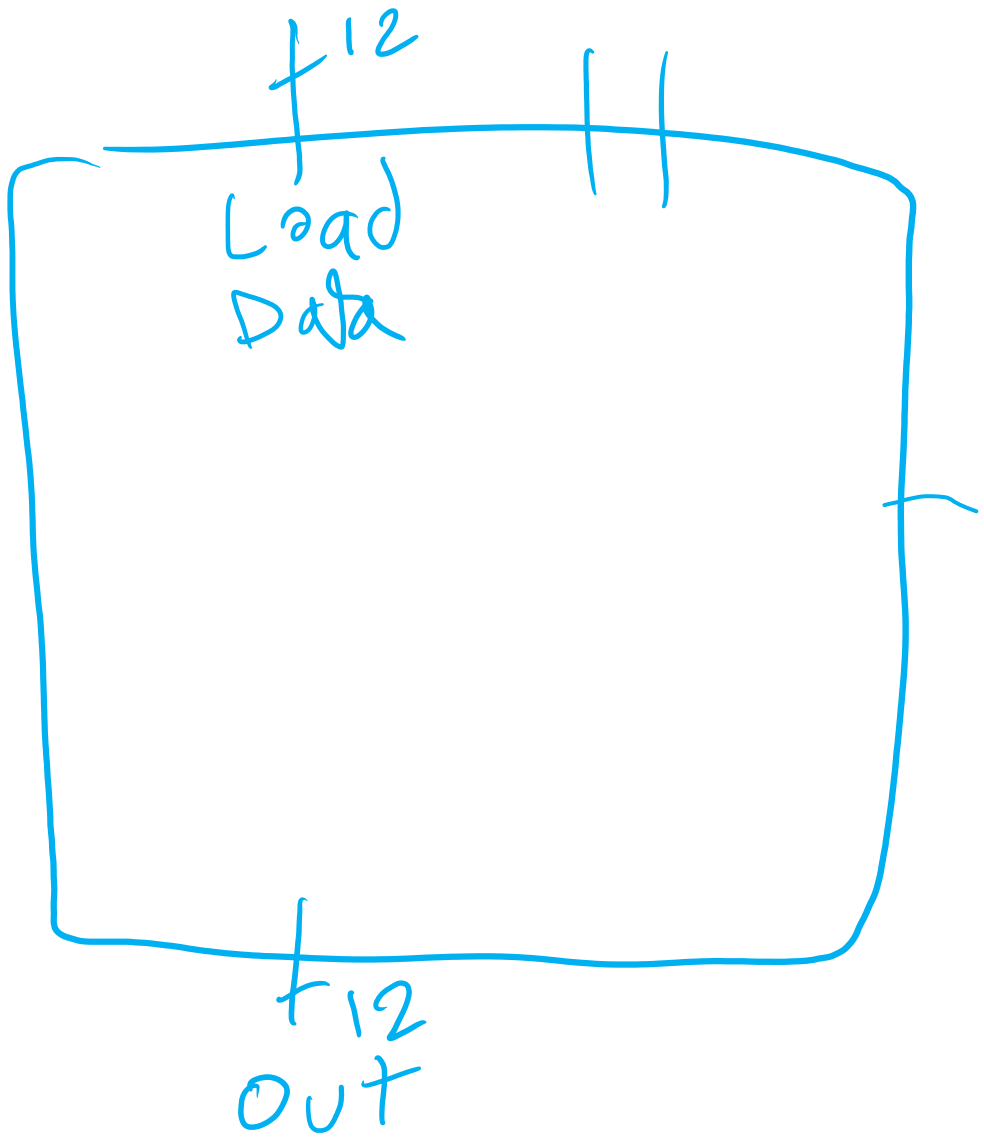
	00	01	11	10
0	0	1	0	1
1	0	1	0	1

N_0 (P_2 \ P_1P_0):

	00	01	11	10
0	1	0	0	1
1	1	0	0	1

Complex Binary Counter

- 👸 "Build me a 12-bit counter that can be paused, reset to 0, or reset to an arbitrary value!"

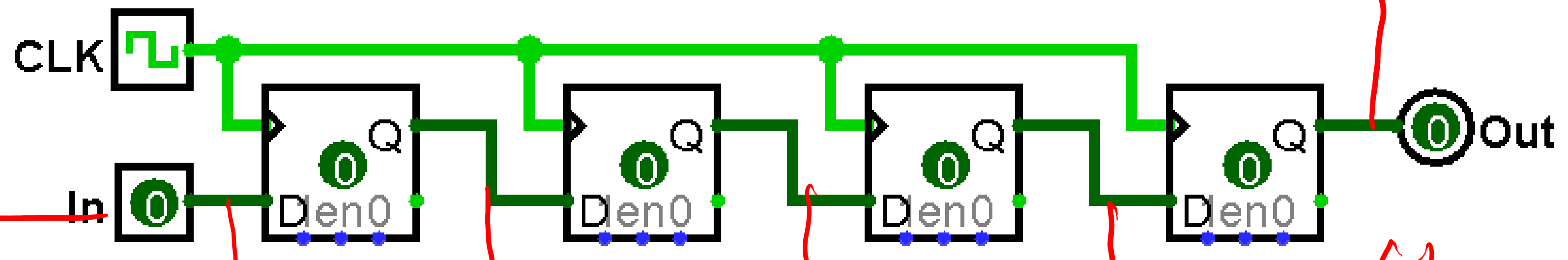


Complex Binary Counter in Verilog

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

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

  always_ff @(posedge clk) begin
    case ({load, enable})
      2'b00: out <= out;
      2'b01: out <= out + 1;
      2'b10: out <= data;
      2'b11: out <= 0;
    endcase
  end
endmodule  // upcounter
```

Shift Register

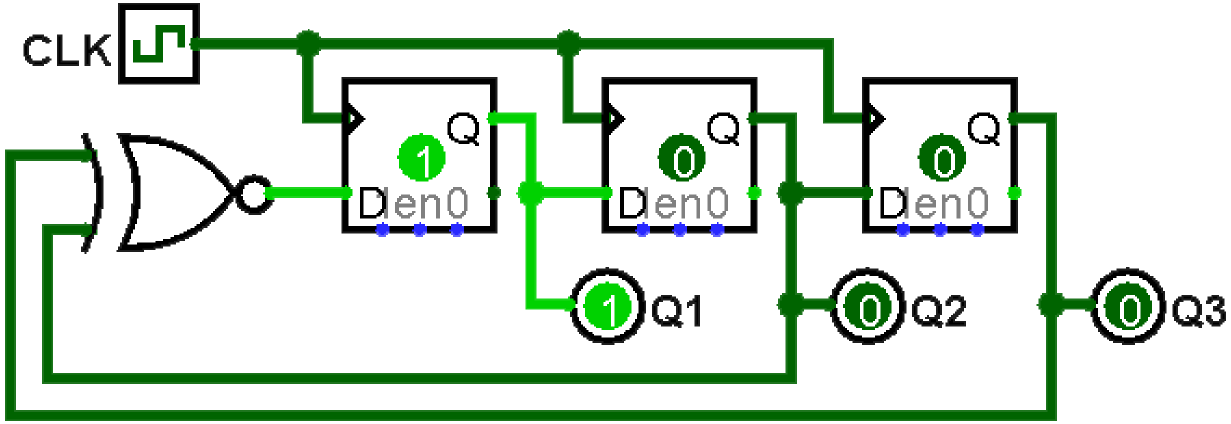
- Register that shifts the binary values in one or both directions



- 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:

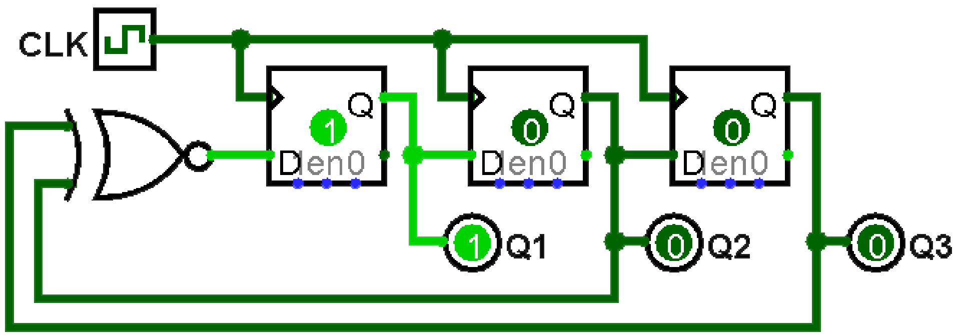


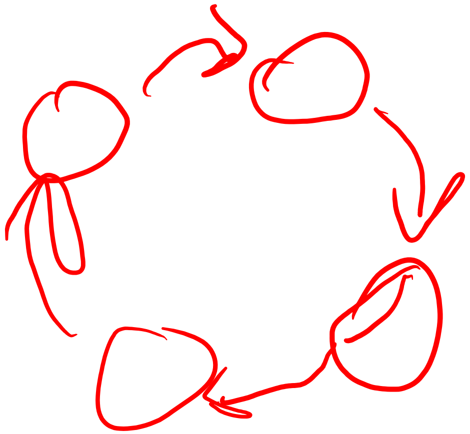
- Example: pseudo-random number generator
  - Input: no external input!
  - Output: all state bits together as a bus {Q3, Q2, Q1}
    - cycles through 000 -> 001 -> 011 -> 110 -> 101 -> 010 -> 100 -> 000

xnor

LFSRs are counters too!

- Just not…uh…in order
  - Visit values on the number line at most once
  - Next value is *always* solely determined by current value
  - Choice of gates and "taps" determine the sequence
    - Most are boring and predictable 😪
    - A few "look random" 🙌 🎉





Simple LFSR in Verilog

```
module LFSR (Q, clk);

  output logic [2:0] Q;
  input  logic clk;

  always_ff @(posedge clk) begin
    Q[1] <= Q[0];
    Q[2] <= Q[1];
    Q[0] <= ~(Q[1]^Q[2]);
  end

endmodule
```