Intro to Digital Design

L7: Encoders, Decoders, Registers, and Counters (oh my!)

Instructor: Naomi Alterman

Teaching Assistants:

Derek de Leuw

Isabel Froelich

Kevin Hernandez

Aarjav Jain

Packard Stephenson

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

Outline

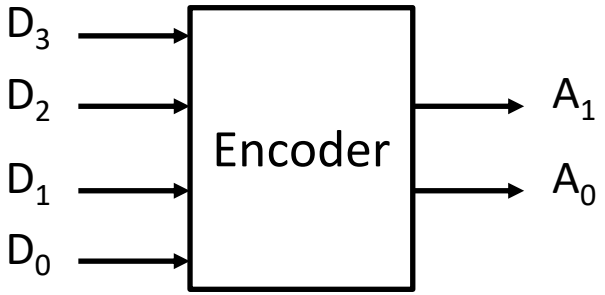
- **Circuit Routing Elements**
- Register Revisited
- Serial I/O

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



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

"zero hot" – at most one wire hot

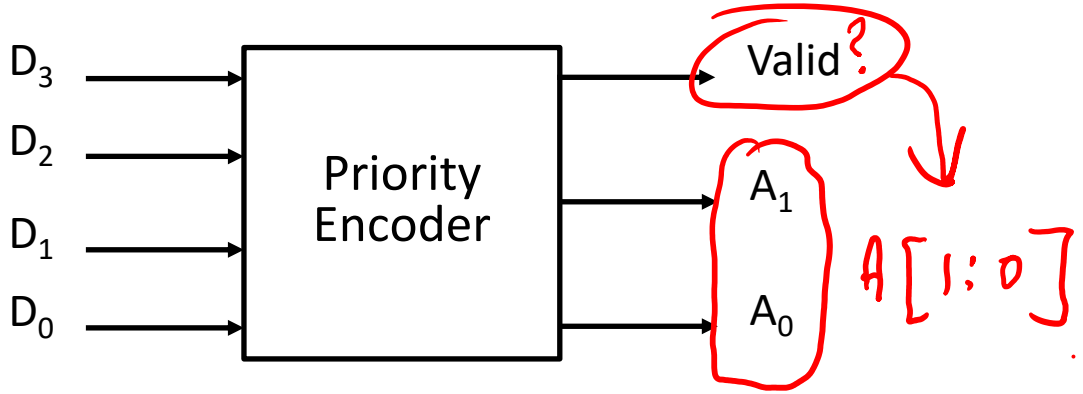
one hot

- 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")

Priority Encoder

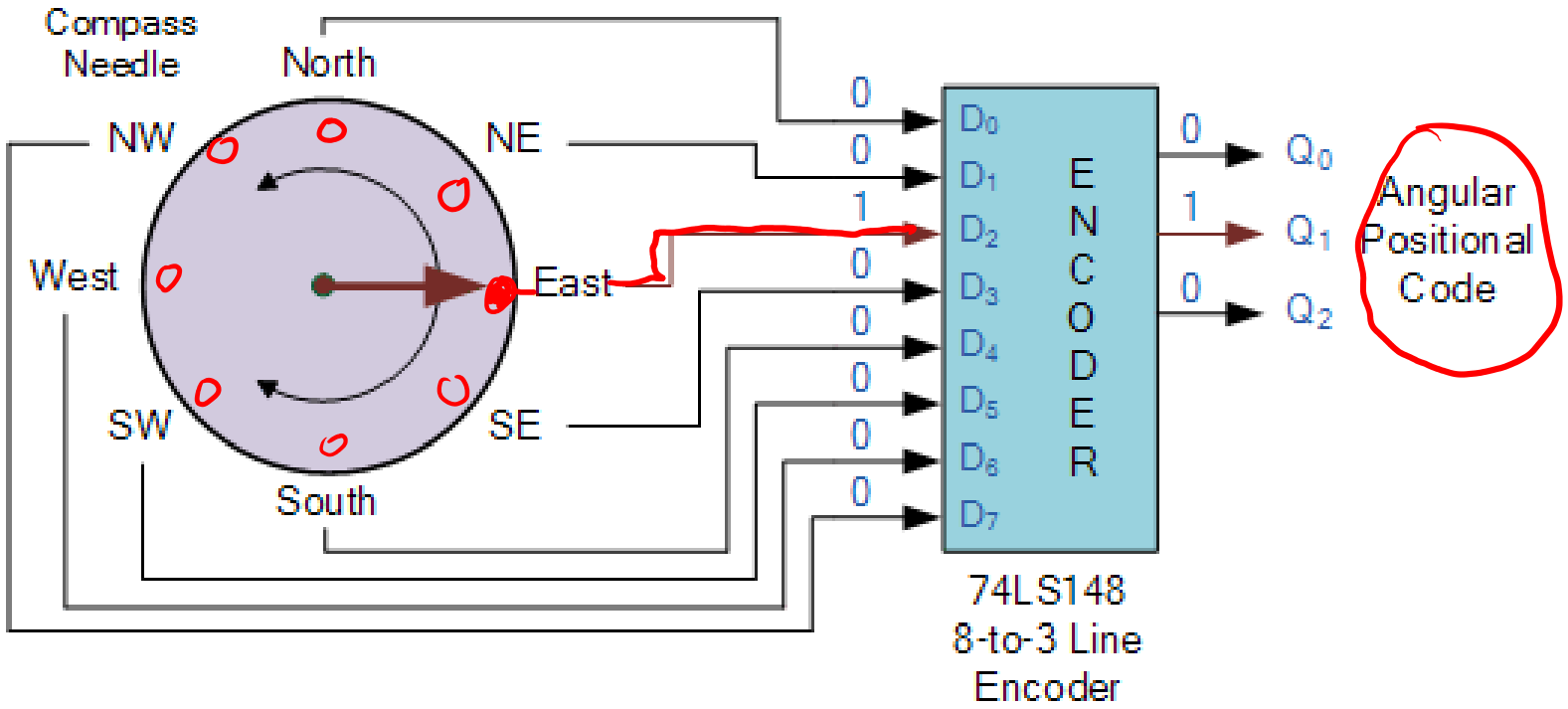
- Use for inputs that might not be one-hot. Most-significant bit *gets priority*/"wins"
- Add an output to identify when at least 1 input active

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

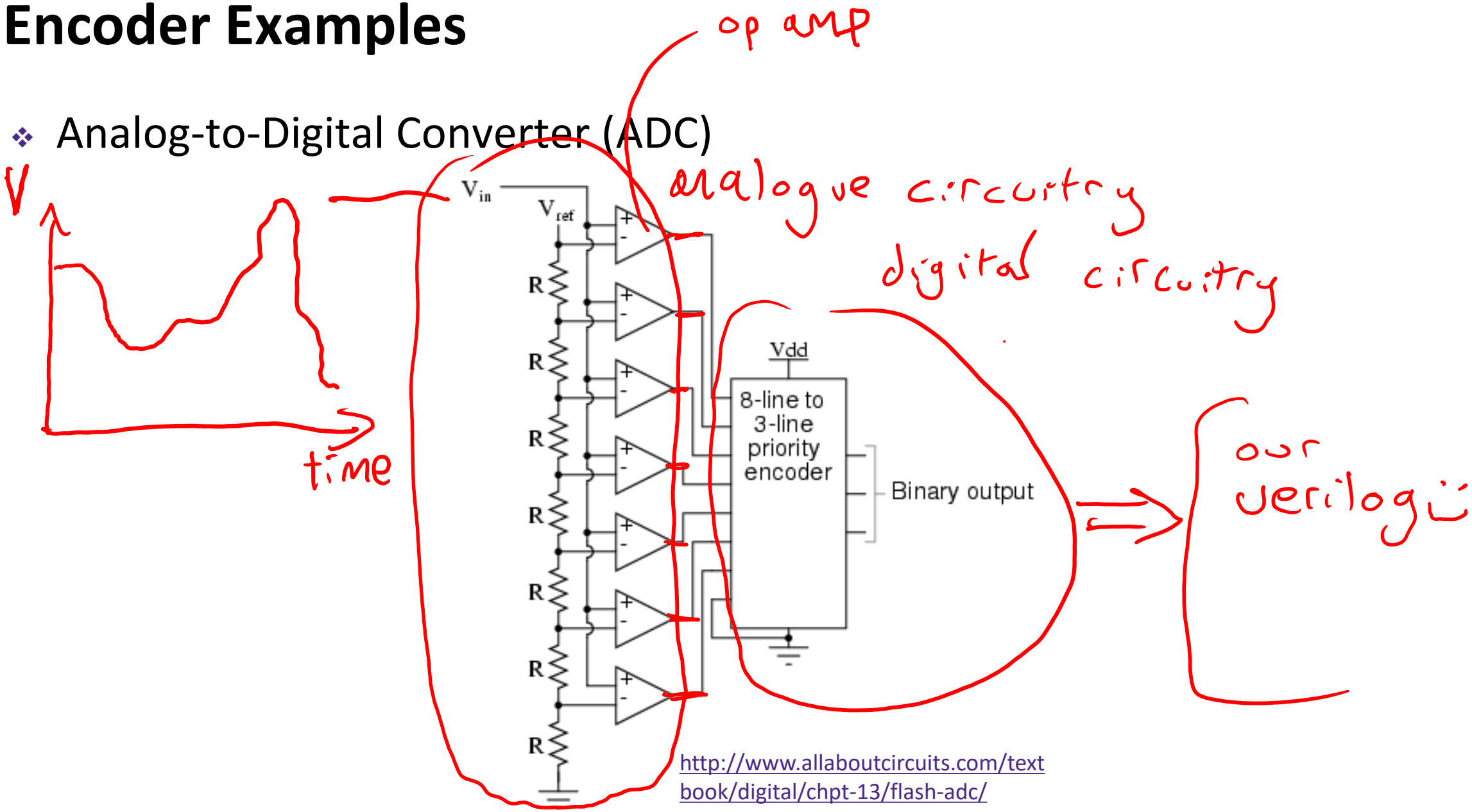
- Navigation (Compass) Encoder



https://www.futurlec.com/74LS/74LS148.shtml
http://www.electronics-tutorials.ws/combination/comb_4.html

Encoder Examples

- Analog-to-Digital Converter (ADC)

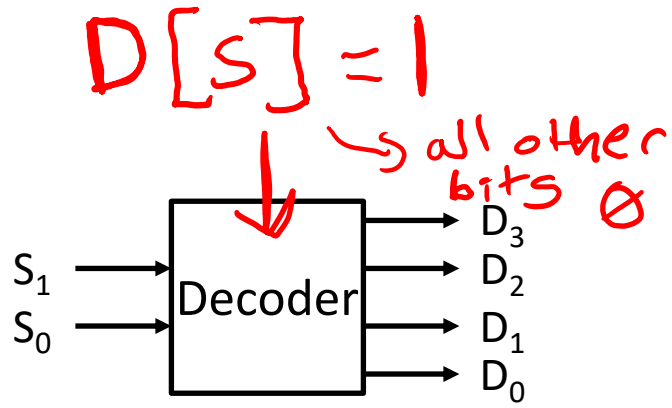


http://www.allaboutcircuits.com/textbook/digital/chpt-13/flash-adc/

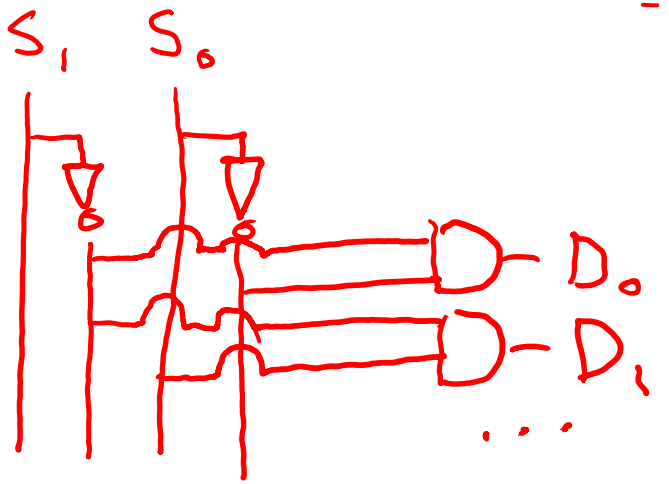
Decoder

- 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")

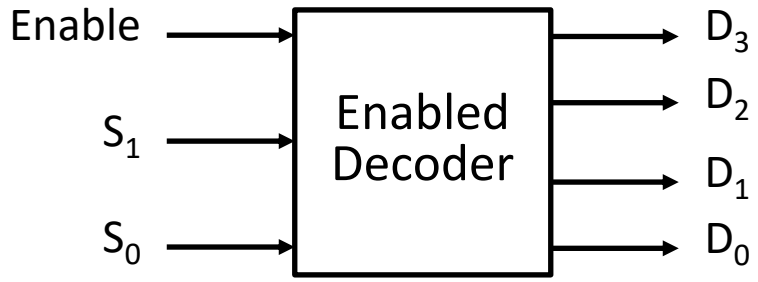


S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



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

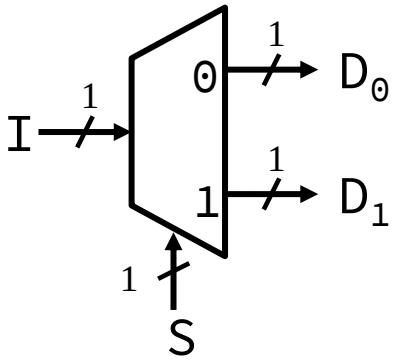
Enabled Decoder in Verilog

```
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

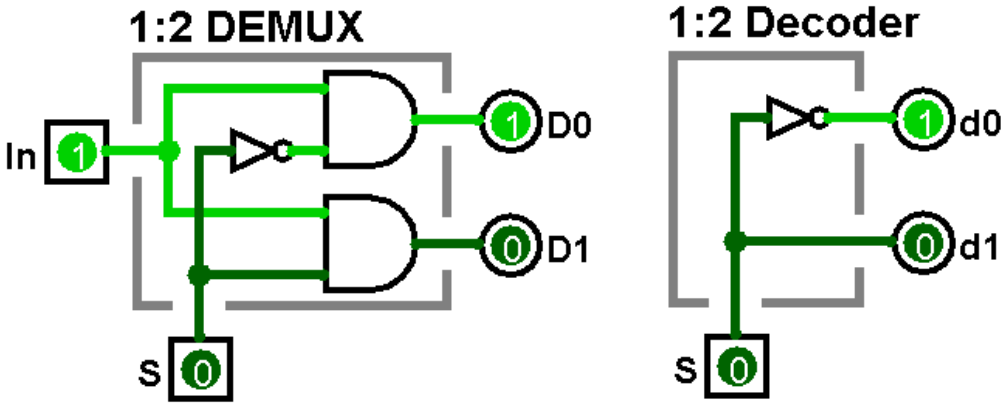
taking a signal and routing it to one of N outputs

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



- **Truth Table:**

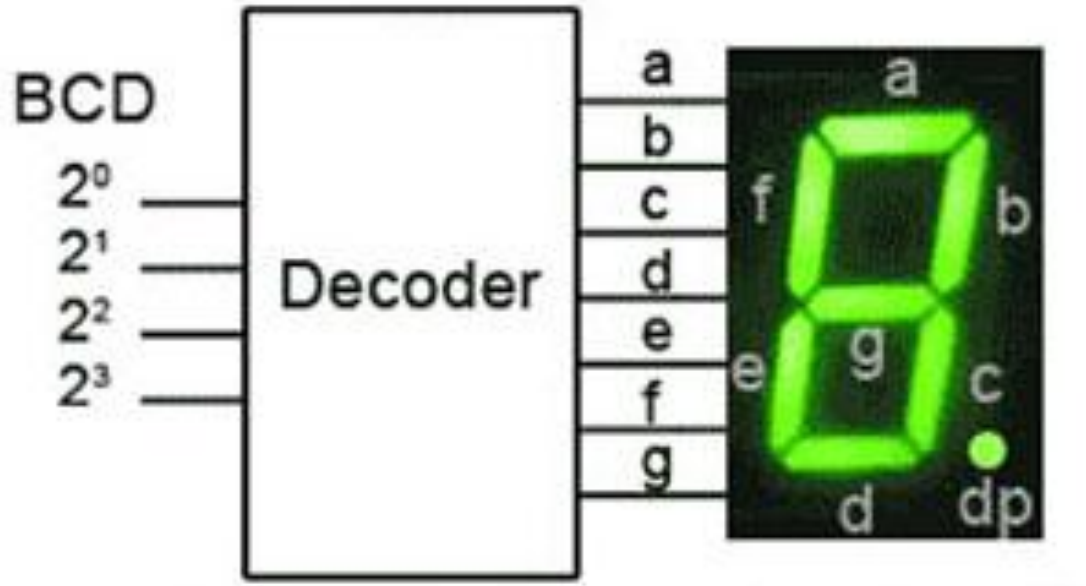
S	I	D_1	D_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



- 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!

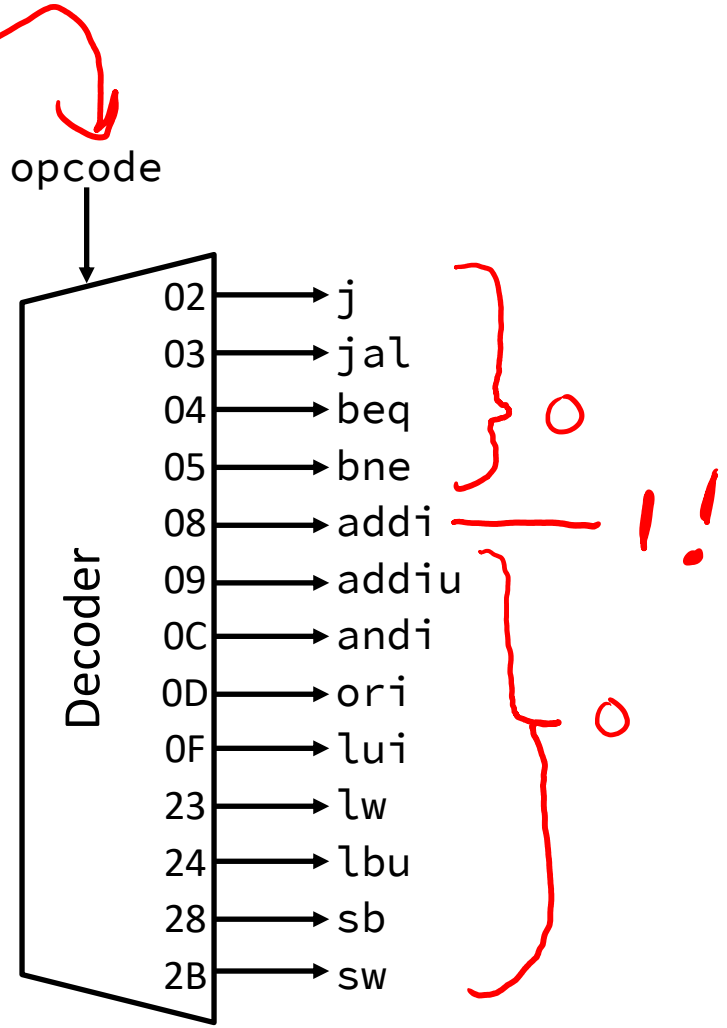


TI SN54/74 series:
https://www.ti.com/lit/ds/symlink/sn54ls47.pdf?ts=1778495638678

Decoder Examples

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

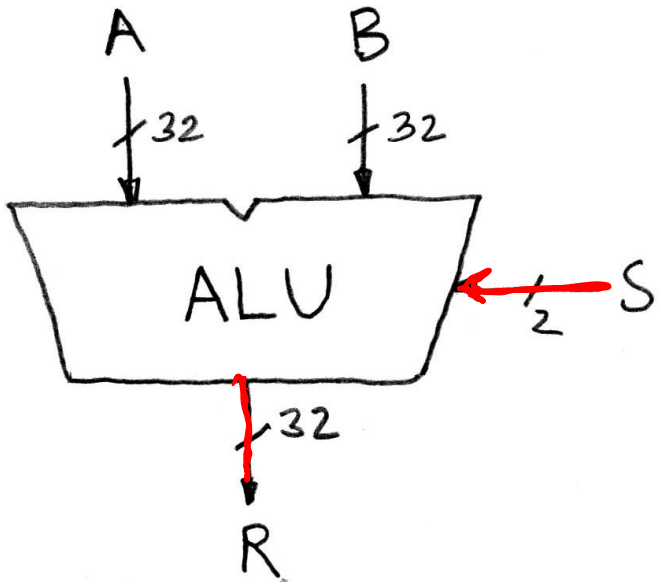
Instruction	Name	Opcode
addi	Add Imm.	001000
addiu	Add Imm. Unsigned	001001
andi	And Imm.	001100
beq	Branch On Equal	000100
bne	Branch On Not Equal	000101
j	Jump	000010
jal	Jump and Link	000011
lbu	Load Byte Unsigned	100100
lui	Load Upper Imm.	001111
lw	Load Word	100011
ori	Or Imm.	001101
sb	Store Byte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

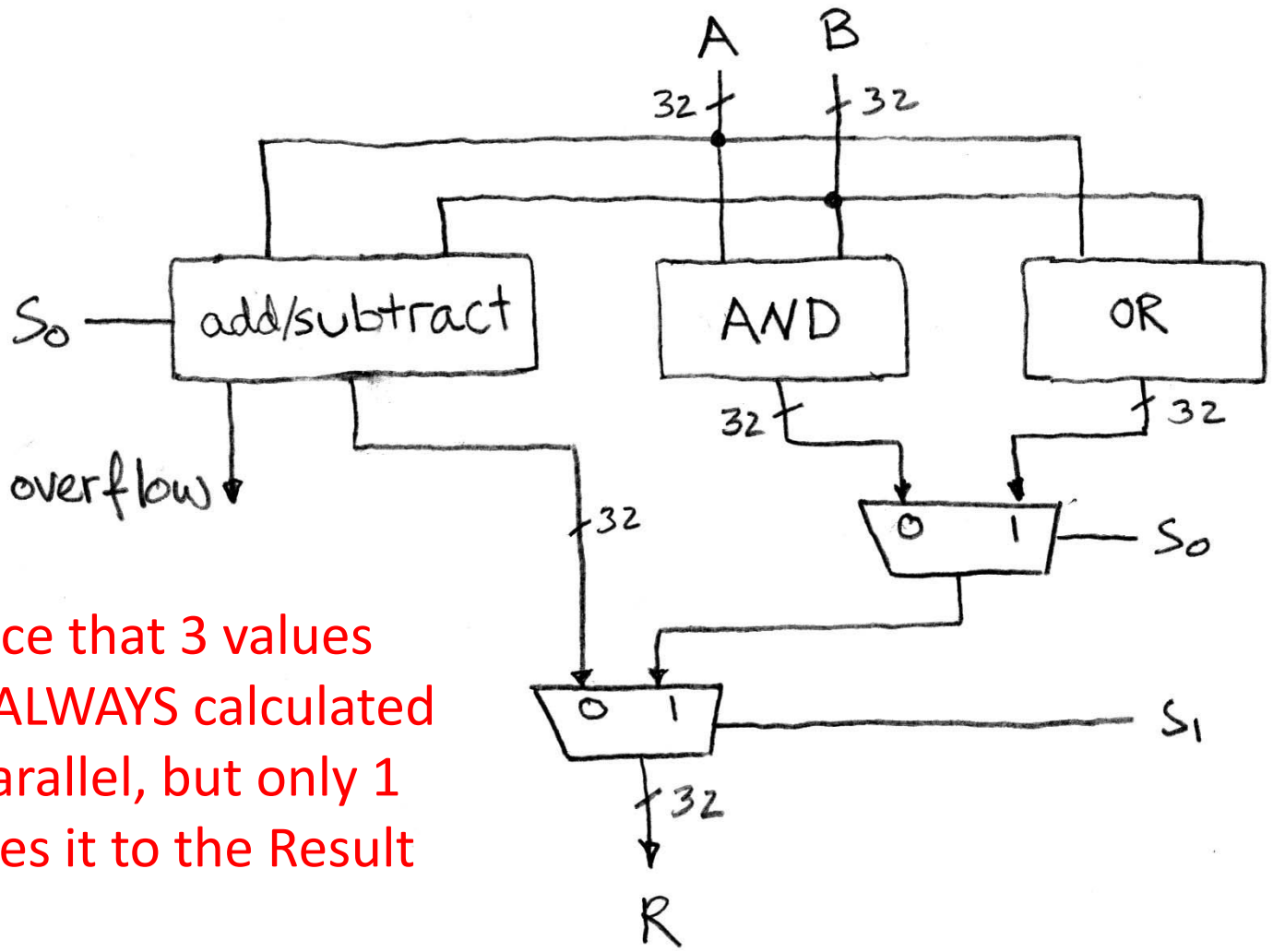
- **Schematic:**



decoded (or encoded, depending on perspective) bits of the opcode

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

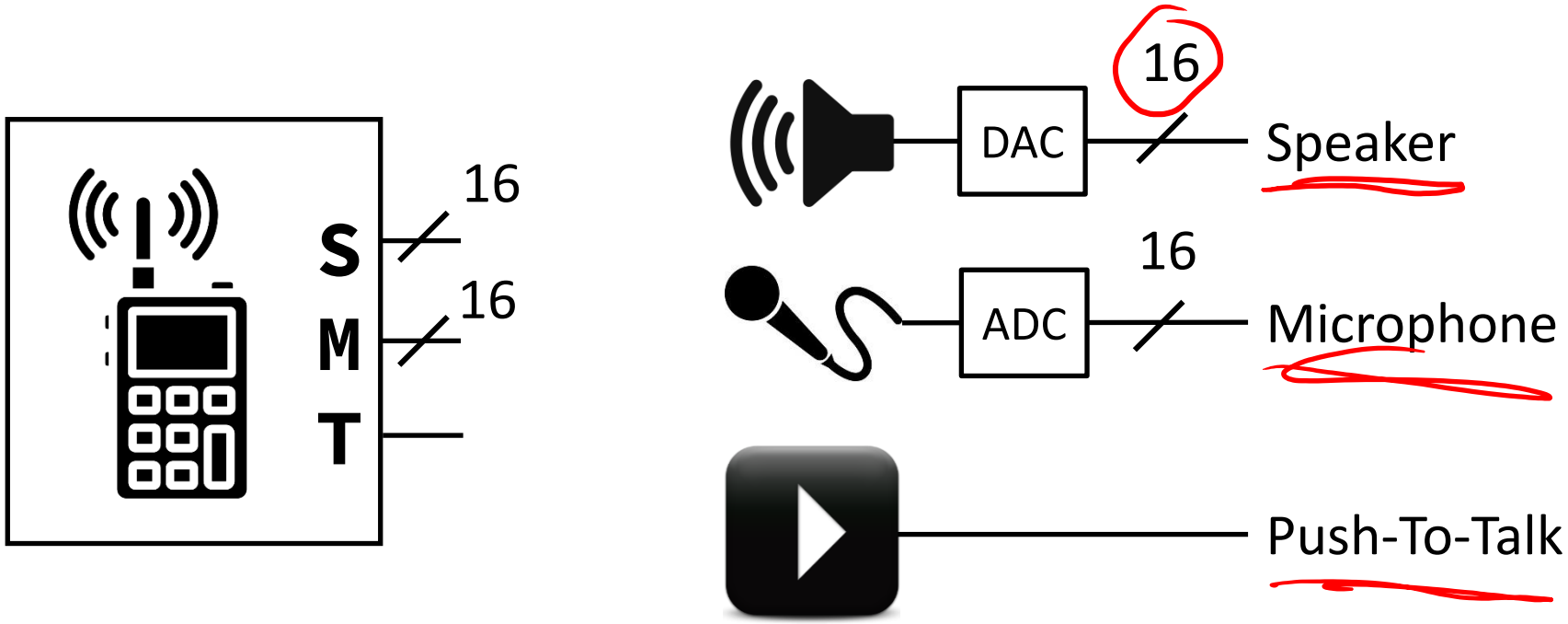
Simple ALU Schematic



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

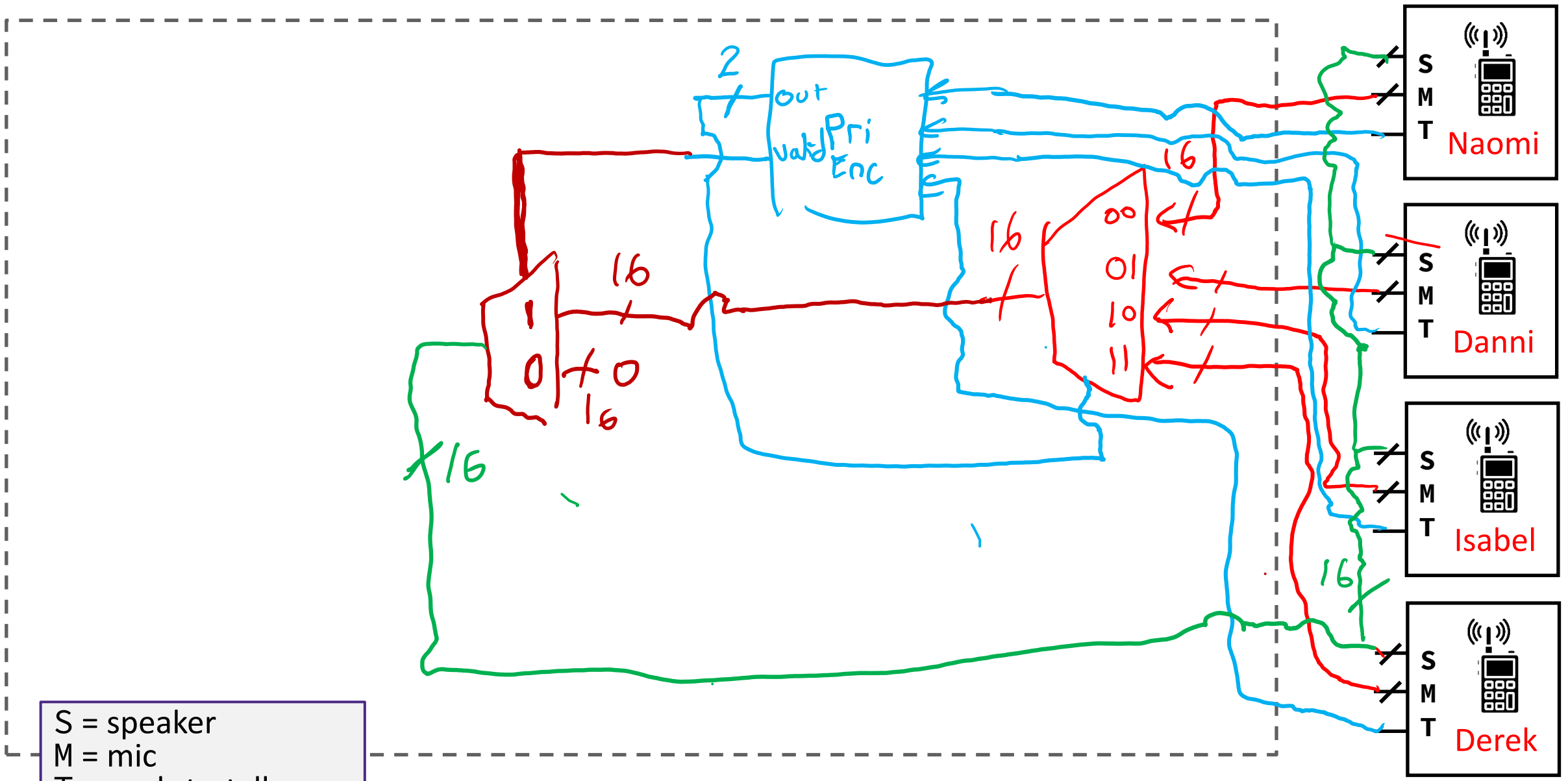
1 Design Example: Walkie-Talkie System

- 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)



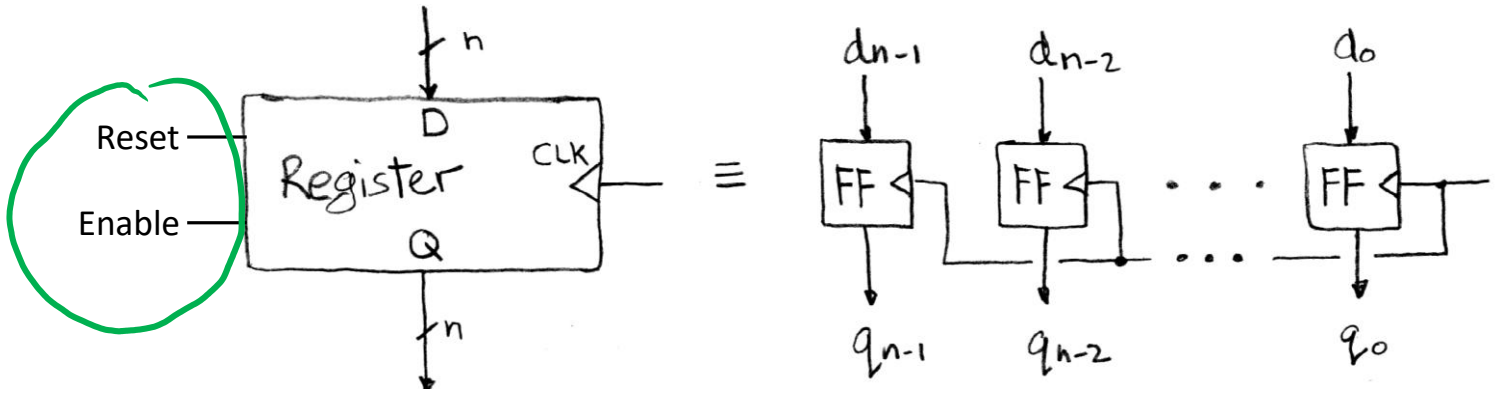
S = speaker
M = mic
T = push-to-talk

Miso Moment

Outline

- Circuit Routing Elements
- **Register Revisited**
- Serial I/O

State Element Revisited: Register



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

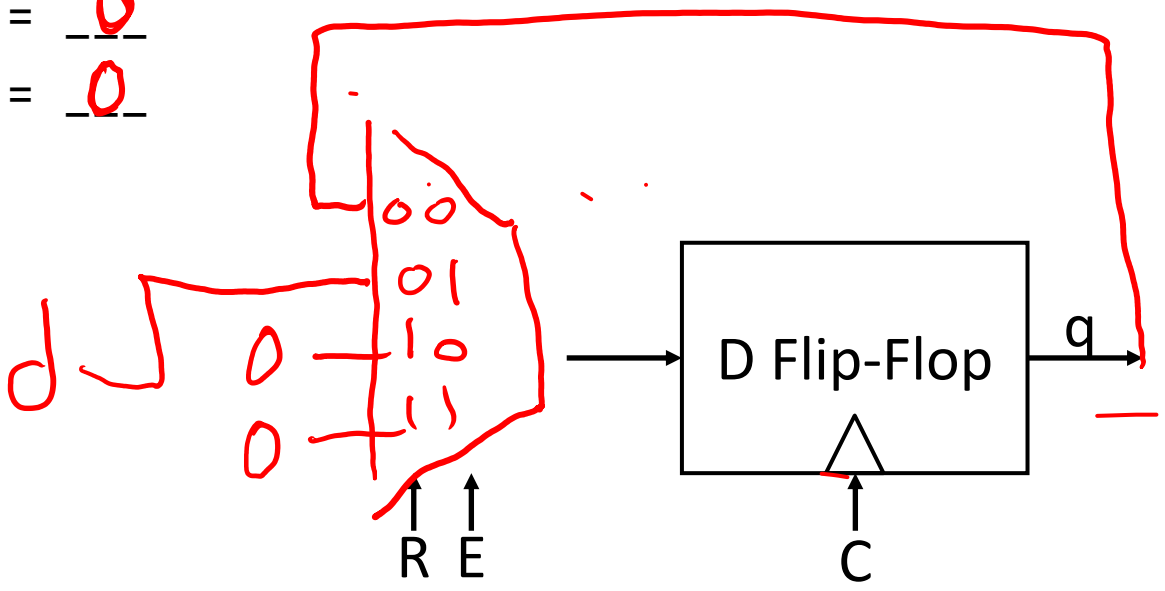
- Desired behaviors (synchronous)
  - Output Q resets to zero when `Reset` signal is high
  - Hold current value unless `Enable` signal is high

Controlled Register

Not a truth table! Just a table

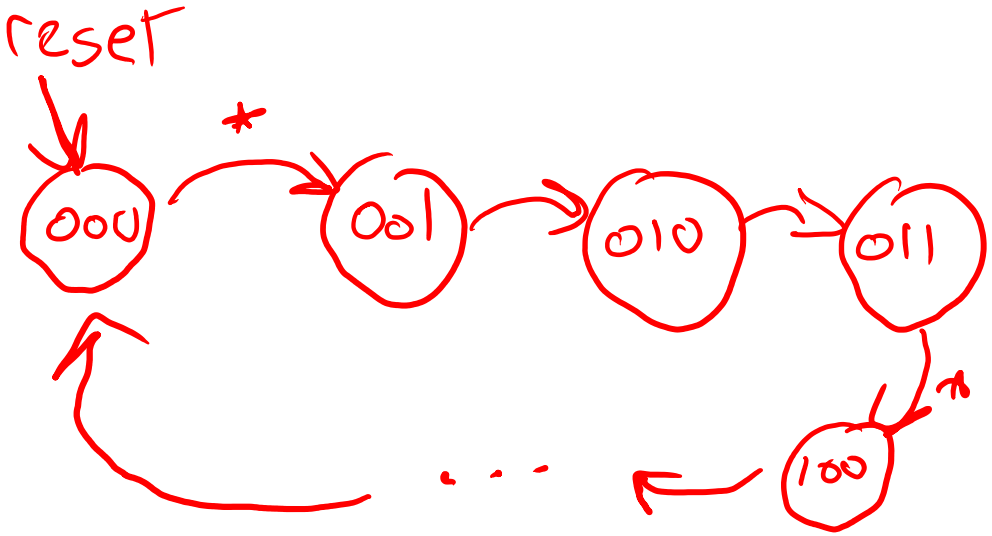
- Here using shorthand C (clock), R (reset), E (enable)

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



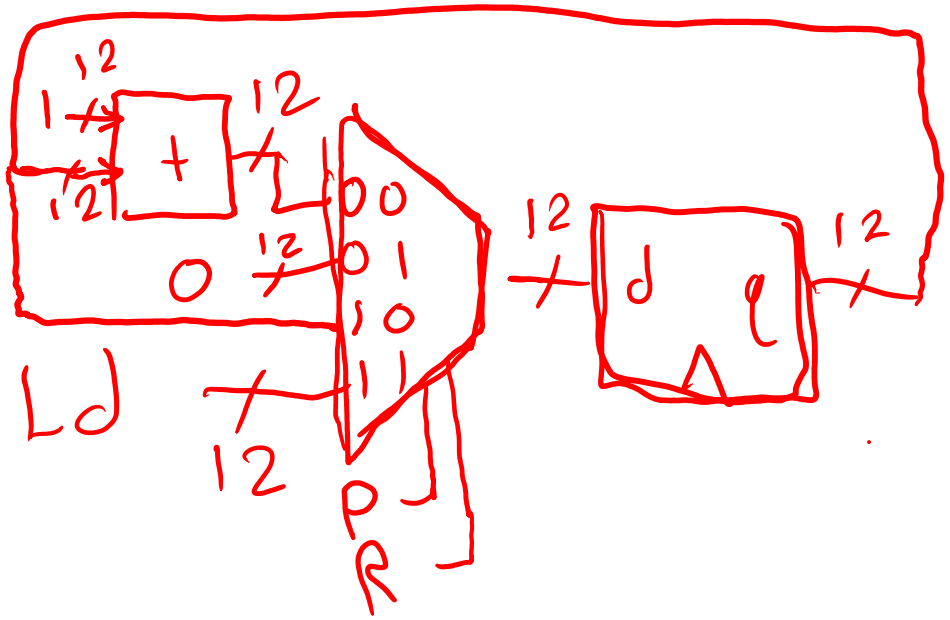
Counters

- 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:



2 Complex Binary Counter

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



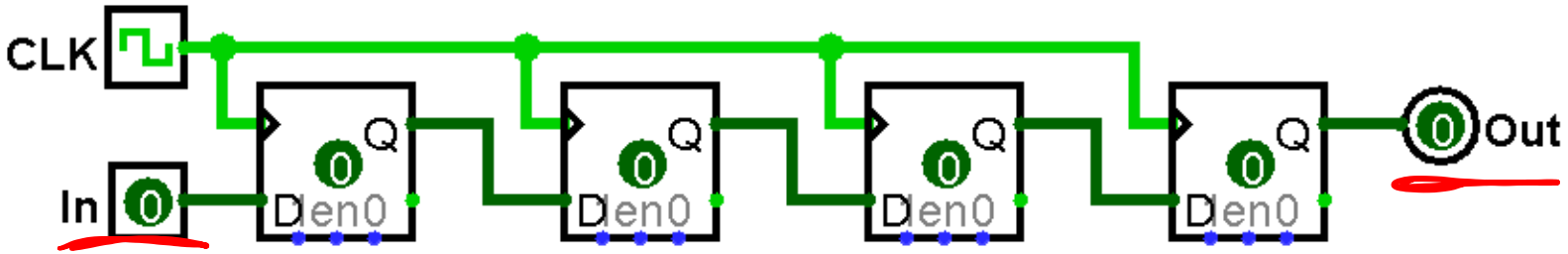
P	R	action
0	0	q = old q+1
0	1	q = 0
1	0	q = old q
1	1	q = load val.

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



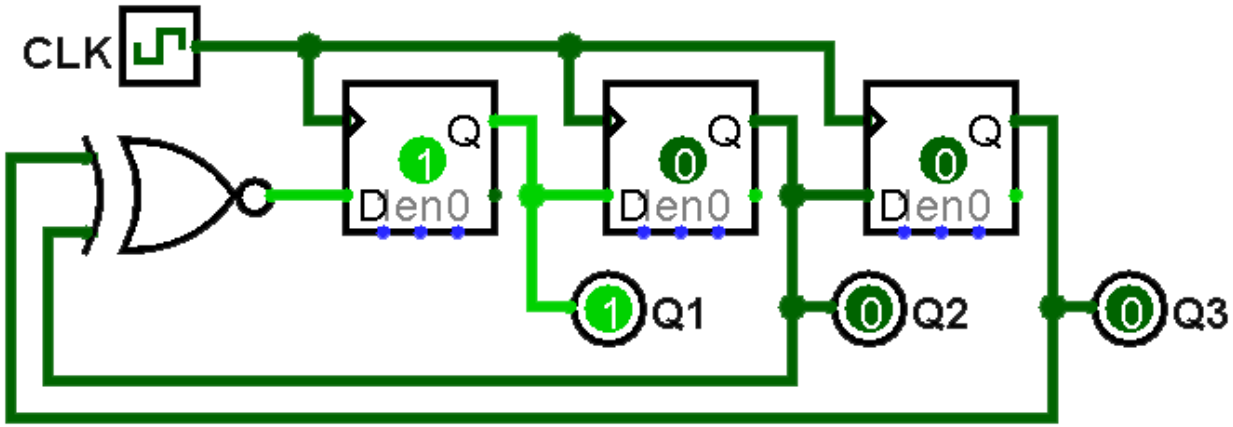
- Where do we get the input from?
  - External input (*e.g.*, delay a signal)
  - Function of current bits (*e.g.*, linear-feedback shift register)

LFSR

- What is the output data of interest?
  - Last (oldest) bit of sequence
  - Entire set of current bits

Linear Feedback Shift Register (LFSR)

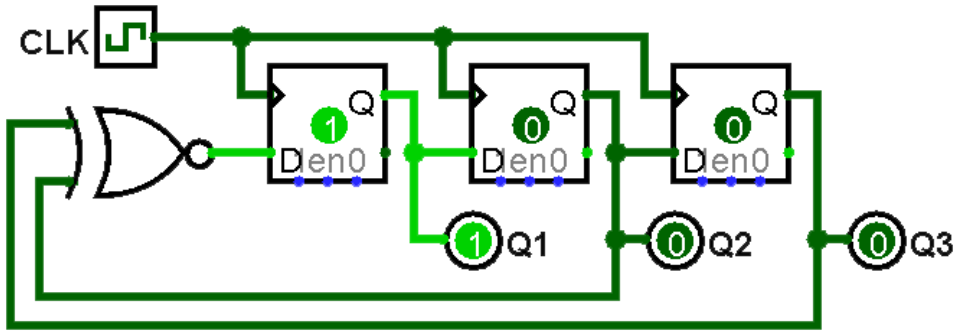
- Shift register input is a logical combination of the current state bits:

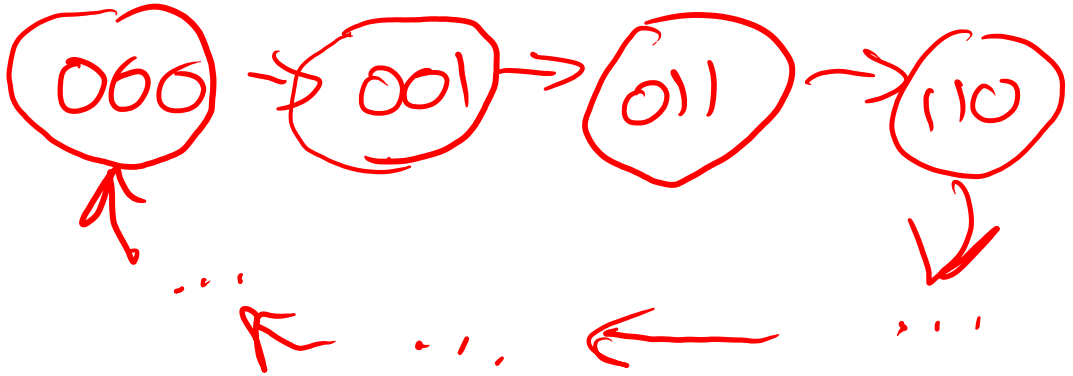


- Example: pseudo-random number generator
  - Input: no external input!
  - Output: all state bits together as a bus {Q3, Q2, Q1}
    - cycles through 000 -> 001 -> 011 -> 110 -> 101 -> 010 -> 100 -> 000

LFSRs are counters too!

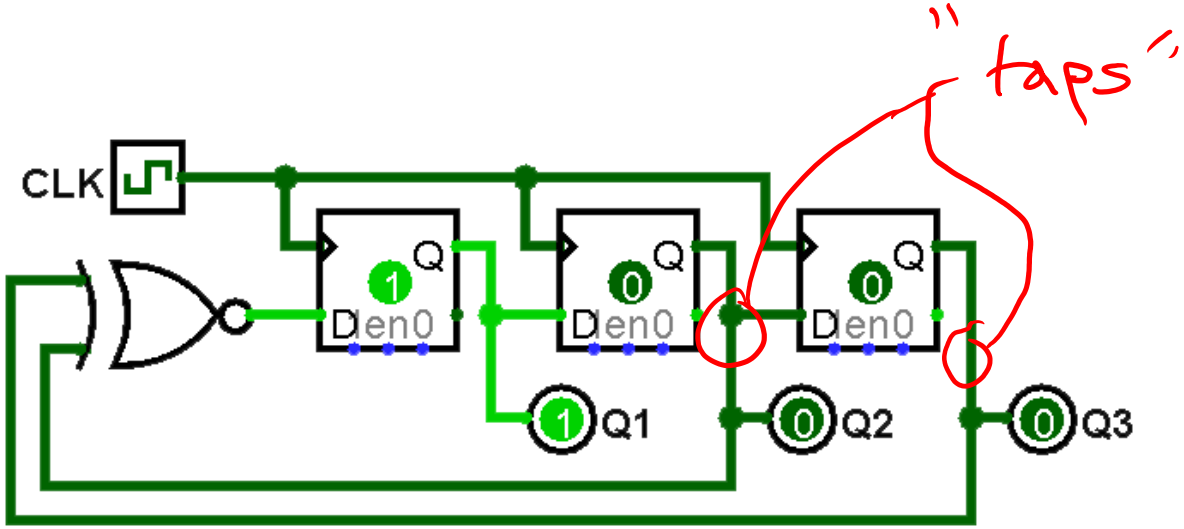
- Just not…uh…in order
  - Visit values on the number line at most once
  - Next value is *always* solely determined by current value
  - Choice of gates and "taps" determine the sequence
    - Most are boring and predictable 😢
    - A few "look random" 👐 🎉





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
```



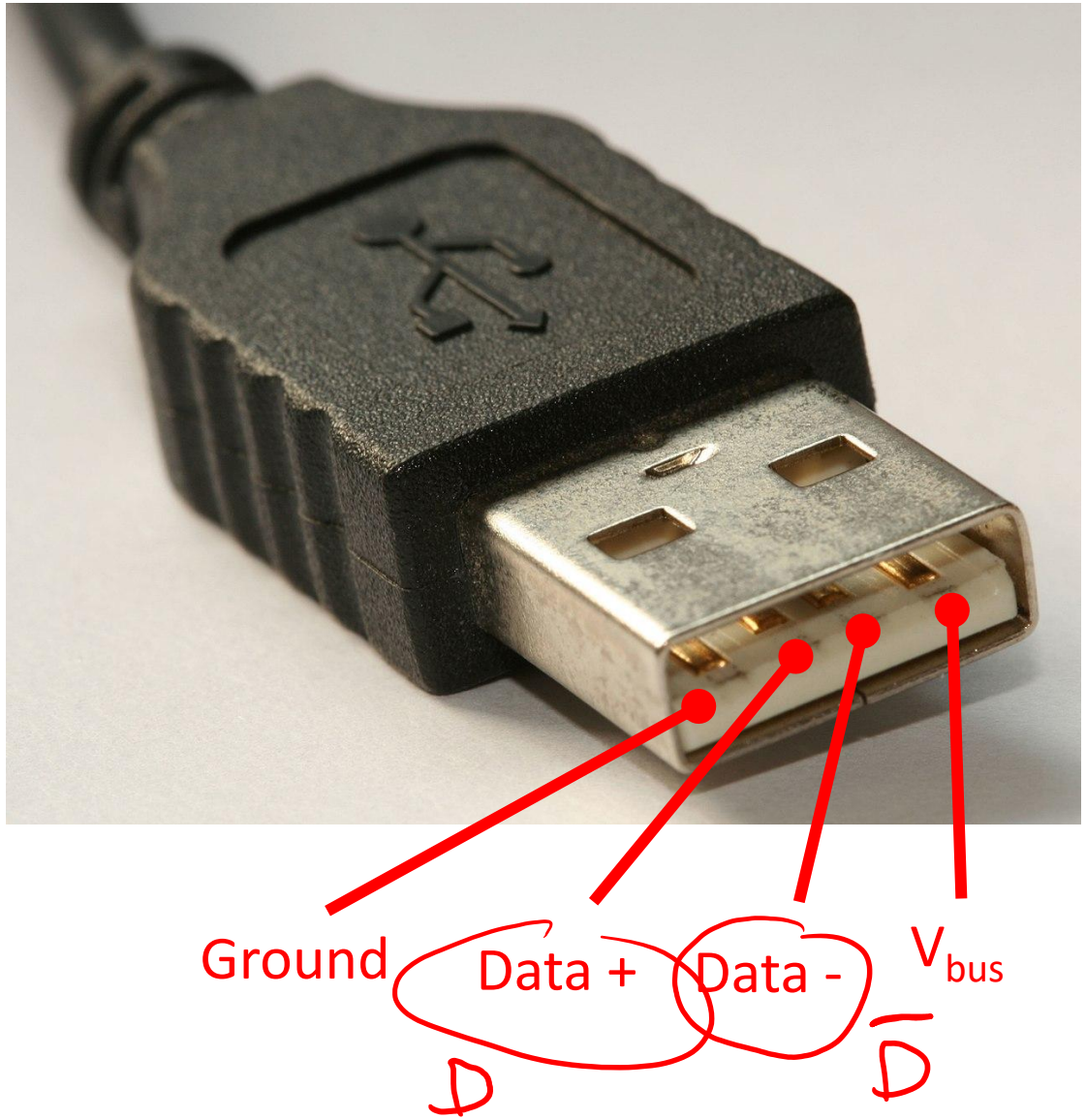
Outline

- Circuit Routing Elements
- Register Revisited
- **Serial I/O**

Data Transfer

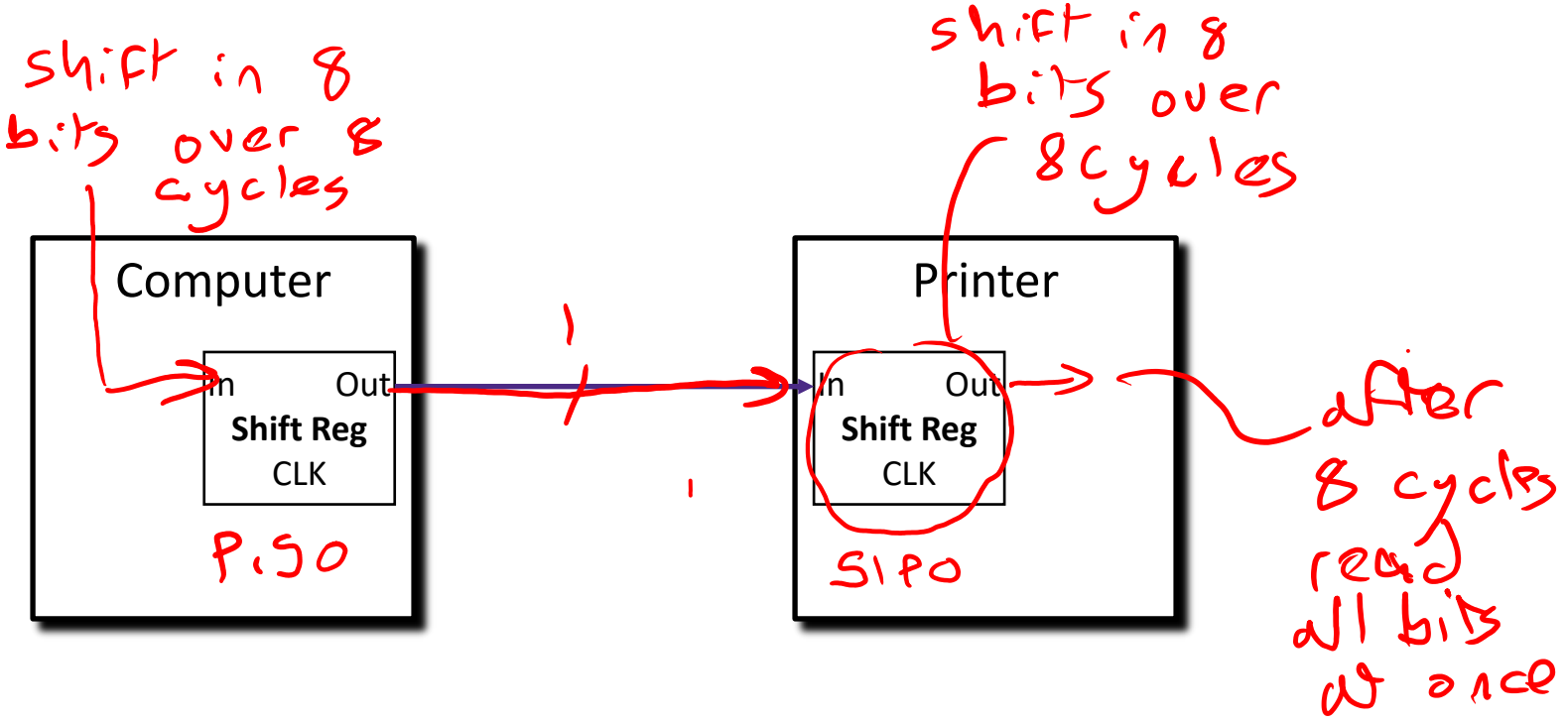
https://commons.wikimedia.org/wiki/Category:USB_2_Standard-A_plugs#/media/File:Type_A_USB_Connecter_alt.jpg

- How do we get stuff **in and out** of a chip?
  - **Parallel:** all bits transferred at once
  - **Serial:** one bit transferred at a time
- USB: classic example of a **serial** I/O standard
  - Only one ("differential") data line
- So how do we get 32-bit data onto the 1-bit USB wire?



Transfer of Data

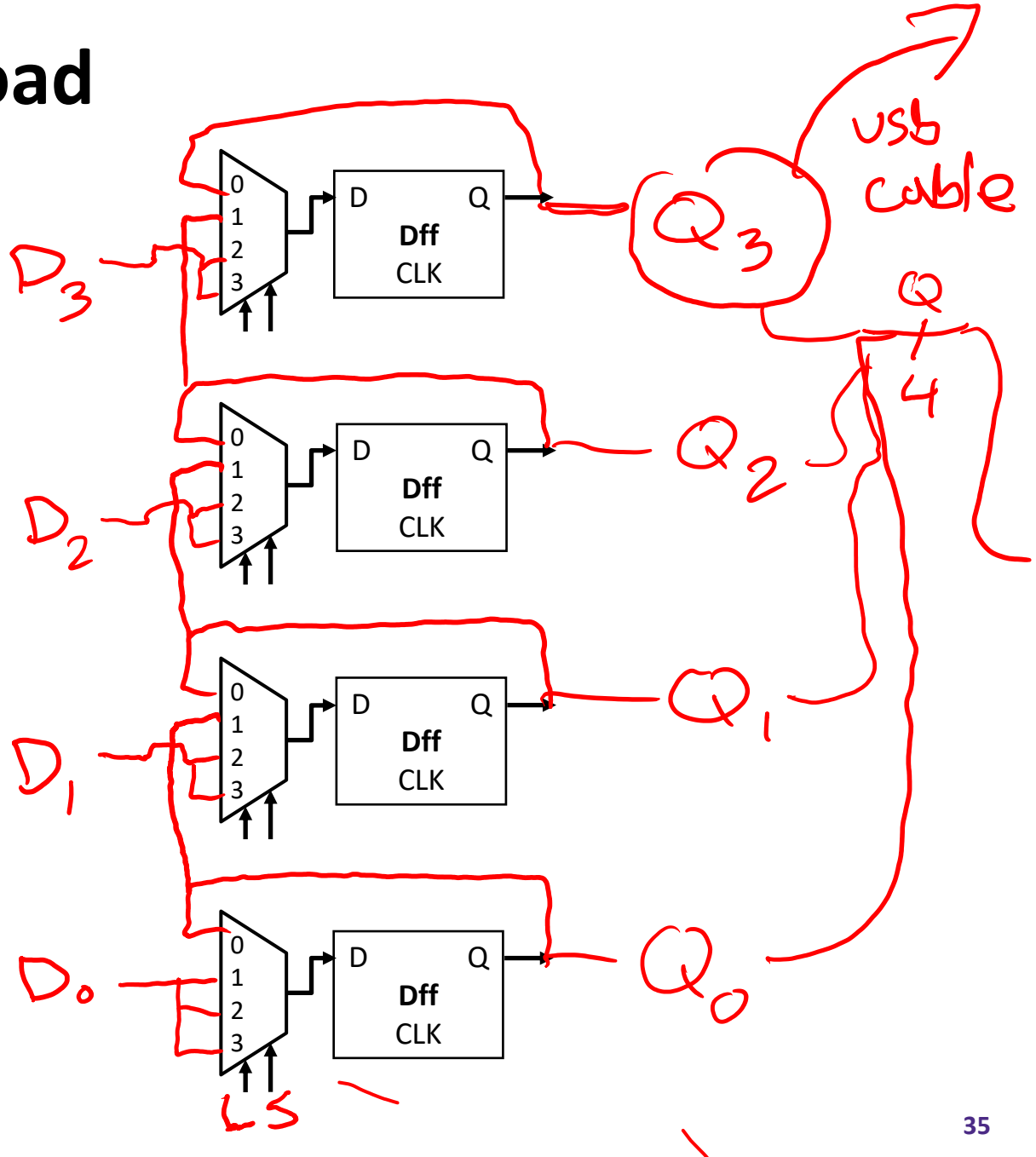
- Shift registers can be used for serial transfer:



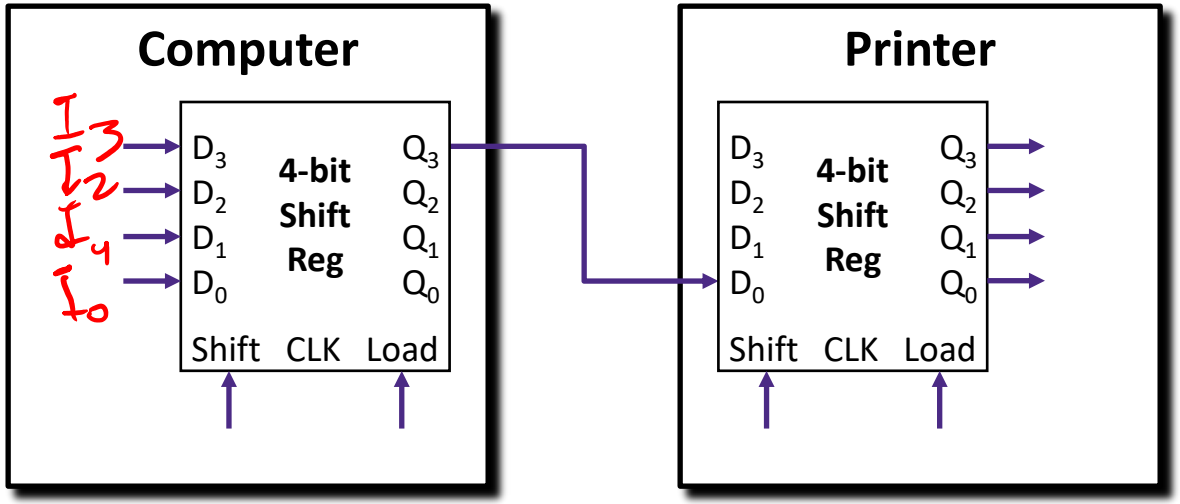
- In general, shift registers can allow for **either** or **both** modes (S for serial, P for parallel) at input and output
  - Examples: SISO, SIPO, PISO

3 Shift Register w/Parallel Load

Load	Shift	Action
0	0	Q = old Q
0	1	Shift (left)
1	X	Parallel load



4 Conversion between Parallel & Serial



Cycle
0
1
2
3
4
5
6

Shift	Load	Q0	Q1	Q2	Q3
0	1	X	X	X	X
1	0	I_0	I_1	I_2	I_3
1	0	x	I_0	I_1	I_2
1	0	x	x	I_0	I_1
1	0				
1	0				
1	0				

Shift	Load	Q0	Q1	Q2	Q3
1	0	X	X	X	X
1	0	x	x	x	x
1	0	I_3	x	x	x
1	0	I_2	I_3	x	x
1	0				
1	0				
1	0				

Shifter in Verilog

```
module leftshifter #(parameter WIDTH=8)
  (s_out, p_out, shift, load, d_in, clk);

  output logic s_out;              // serial out
  output logic [WIDTH-1:0] p_out; // parallel out
  input  logic [WIDTH-1:0] d_in;  // data in (shift in d0)
  input  logic shift, load, clk;

  always_ff @(posedge clk) begin
    if (load)
      p_out <= d_in;
    else if (shift)
      p_out <= {p_out[WIDTH-2:0], d_in[0]};
  end

  assign s_out = p_out[WIDTH-1];

endmodule  // leftshifter
```