Intro to Digital Design
L9: Memory, Computer Components, FPGAs

Instructor: Naomi Alterman

Teaching Assistants:

Derek de Leuw
Isabel Froelich
Kevin Hernandez
Sathvik Kanuri
Aadithya Manoj

Administrivia

- Lab 8 – Project
  - Check-ins this week during your normal lab time
    - **Graded!** Turn in a block diagram and a module implementation (with testbench)!
  - Final demo of finished project due before end of next week, reports no later than Friday, March 13 @ 11:59 pm
  - Return your lab kit to TAs at your final demo

Administrivia

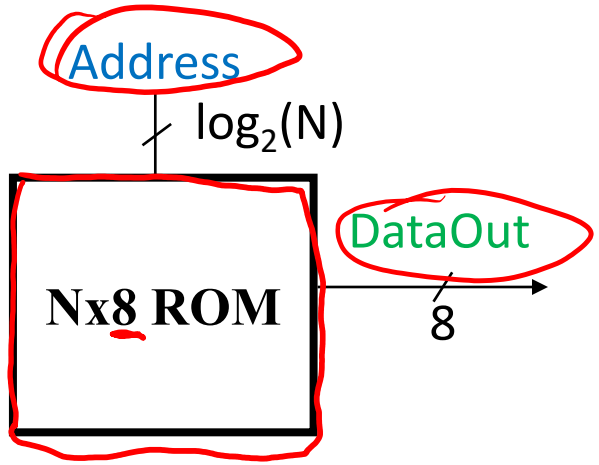
- **Quiz 3** is next week: Tuesday, March 10
  - We'll start class with a quick wrap-up and then give you time to get settled
  - 60 (+10) minutes, worth 14% of your course grade
  - Topics: Timing, Routing Elements, Computational Building Blocks, Verilog
  - Past Quiz 3 (+ solutions) on website: Course Info → Quizzes
    - **Note:** Your Quiz 3 will be a little different – focus on problem solving

Outline

- **Memory**
- CPU teaser
- Serial I/O
- FPGAs

Storage Element: Idealized ROM

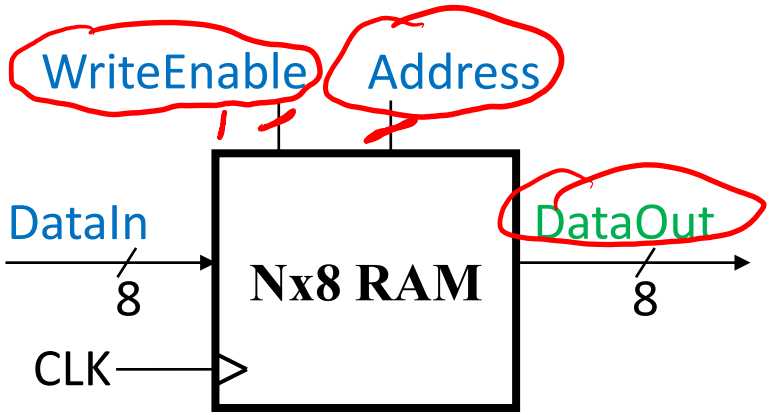
- "**R**ead **O**nly **M**emory"
  - NxM: An array of N M-Bit words
  - Address input selects the word driven on the output
  - Data "burned in" by the manufacturer (or your bitfile) and retained even if power is lost
- Difference between a ROM and your seven segment decoder is *density*
  - (and thus underlying circuit implementation)
- What can we do with a truly *read-only* memory?
  - Math look-up tables (trig functions)
  - FSM state transitions
  - Bitmap images and other simple patterns



DataOut = ROM[Address]

Storage Element: Idealized RAM

- "**R**andom **A**ccess **M**emory"
  - Can now ***write*** as well as read
  - On clock edge, DataIn written to memory cell at Address if WriteEnable = 1
  - DataOut is still completely combinational (changes to Address *immediately* change the data output)
  - Data lost on reset or power-off

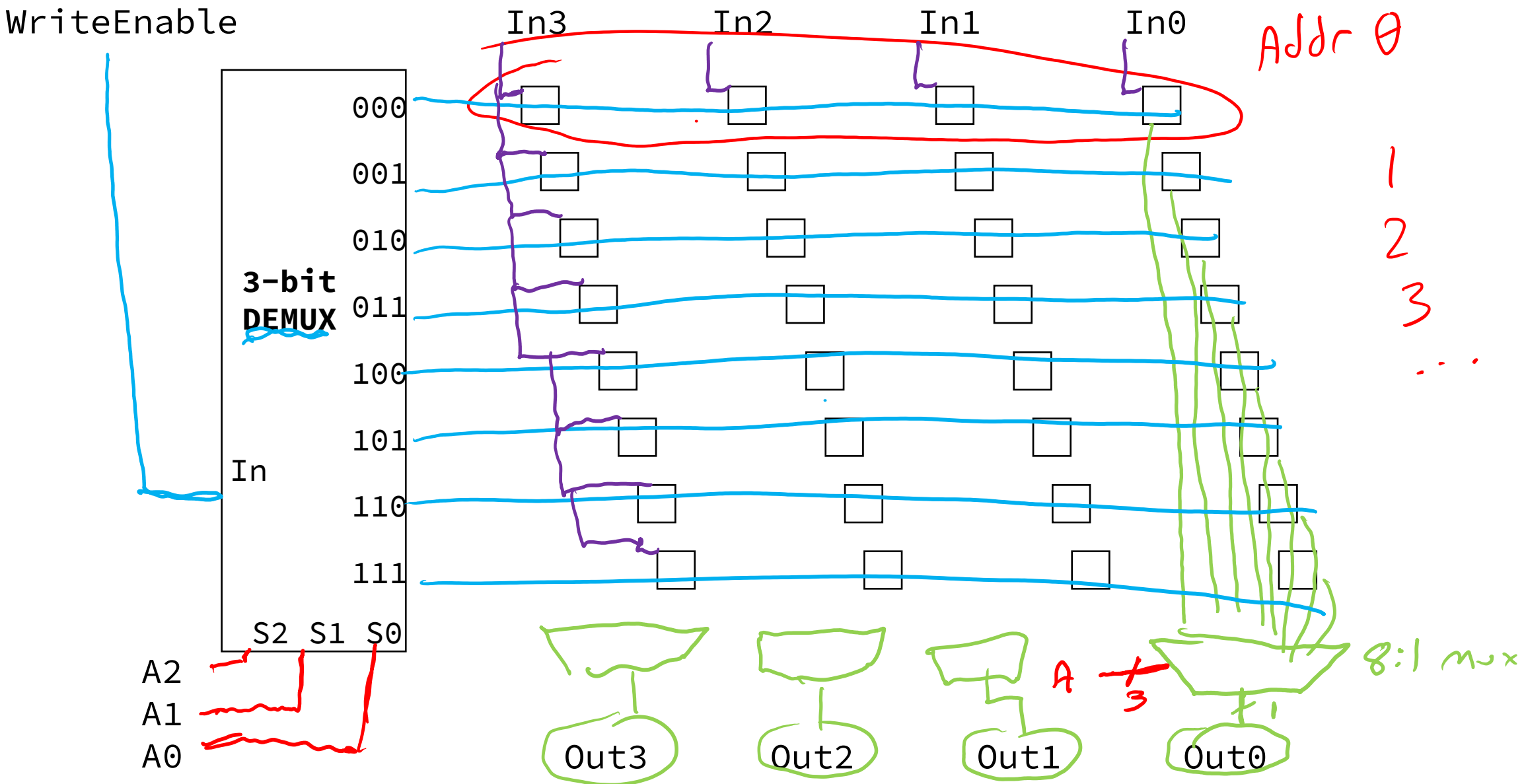


- Simplest and most robust implementation: "SRAM" – implemented as an array of flip flops with muxed inputs and outputs
- Higher-density but more touchy: "DRAM" – implemented with circuit wizardry

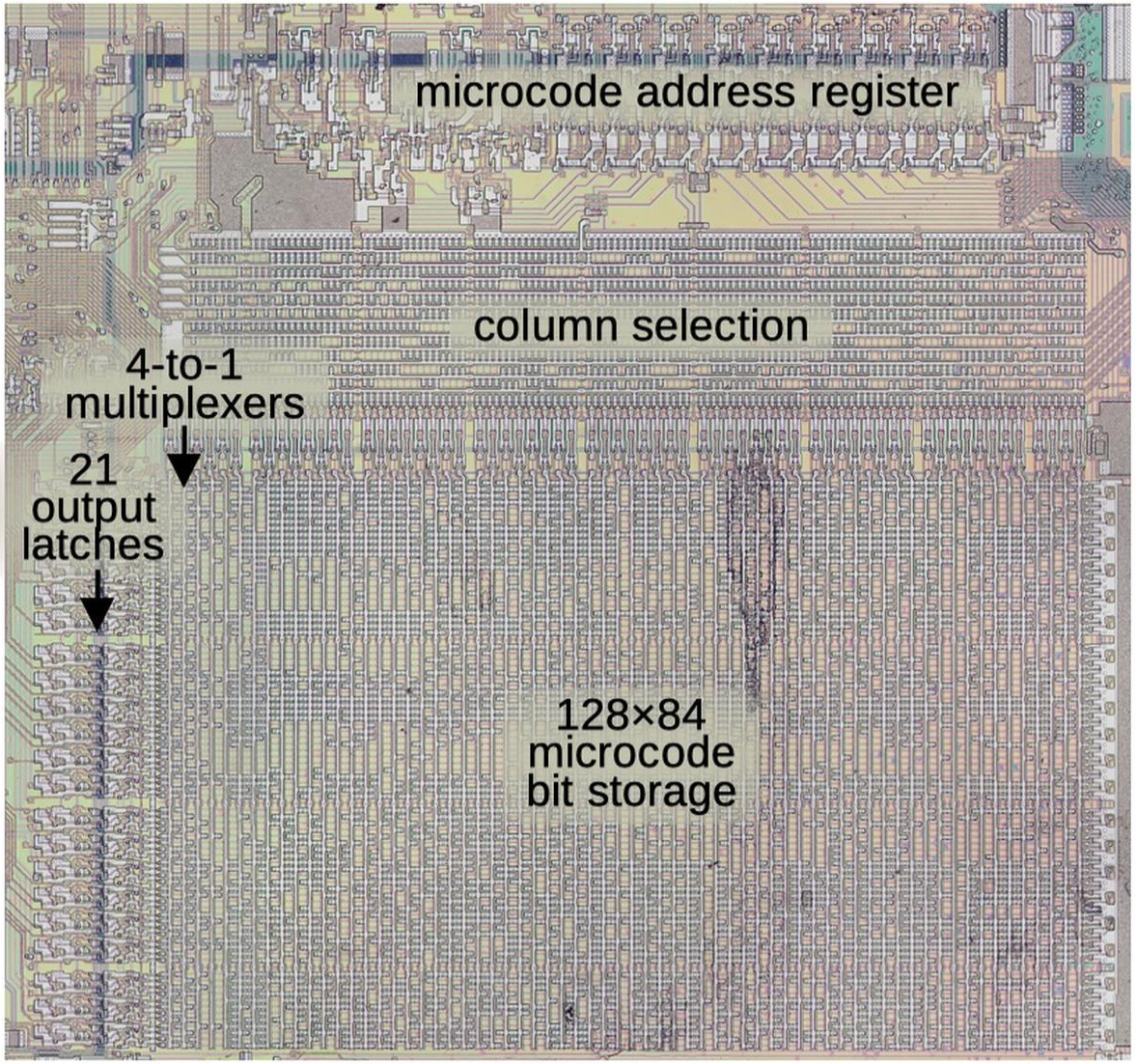
8 4-bit words

Blue = FF En.

8x4 SRAM



Memory IRL: Microcode ROM in the original x86



https://www.righto.com/2022/11/how-8086-processors-microcode-engine.html

Outline

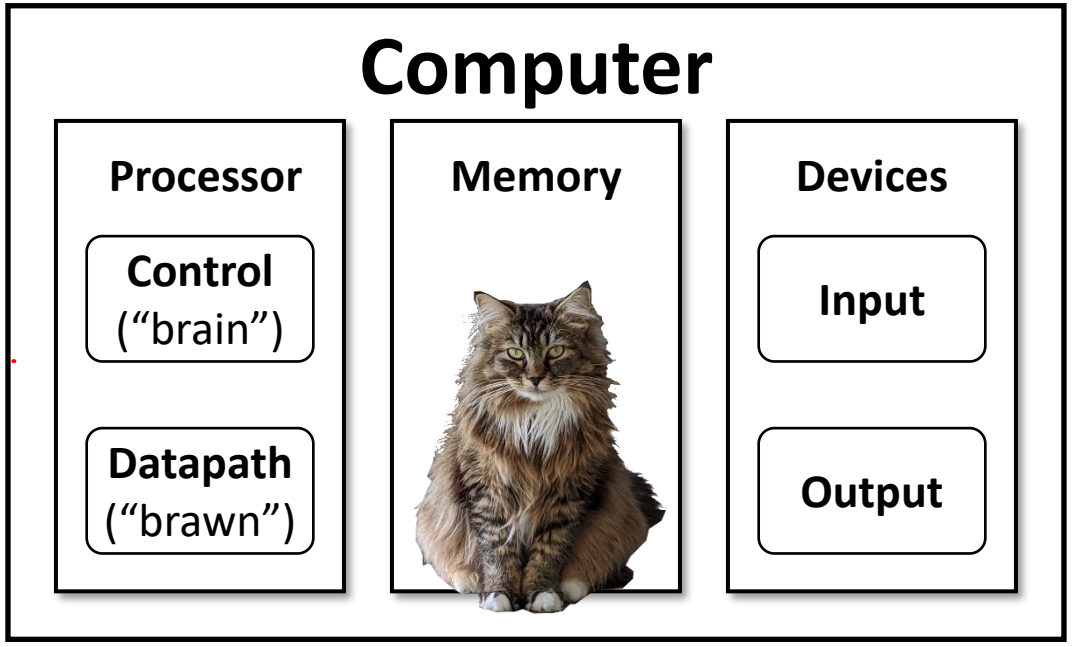
- Memory
- **CPU teaser**
- Serial I/O
- FPGAs

What is a computer?

- The **central processing unit (CPU)**
  - A little machine that takes a list of **instructions** and executes them **sequentially**, one after another
  - Most instructions do basic math on local scratch memory called a **register file** ("ALU instructions")
  - Some instructions can set our next position in the instruction list ("**branches**" and "**jumps**")
  - Some instructions save register values to the larger off-chip **main memory** ("stores" and "loads")

Five Components of a CPU

- The machine that actually executes those instructions is usually broken down like this:
  - Control
  - Datapath
  - Memory
  - Input
  - Output



Executing an Instruction

addq (%rdi), %rax
0x48 03 07

- Depends on ISA, but generally:
  - Instruction Fetch
  - Instruction Decode
  - Data Fetch → RegFile? MainMem?
  - Computation → ALU
  - Store Result

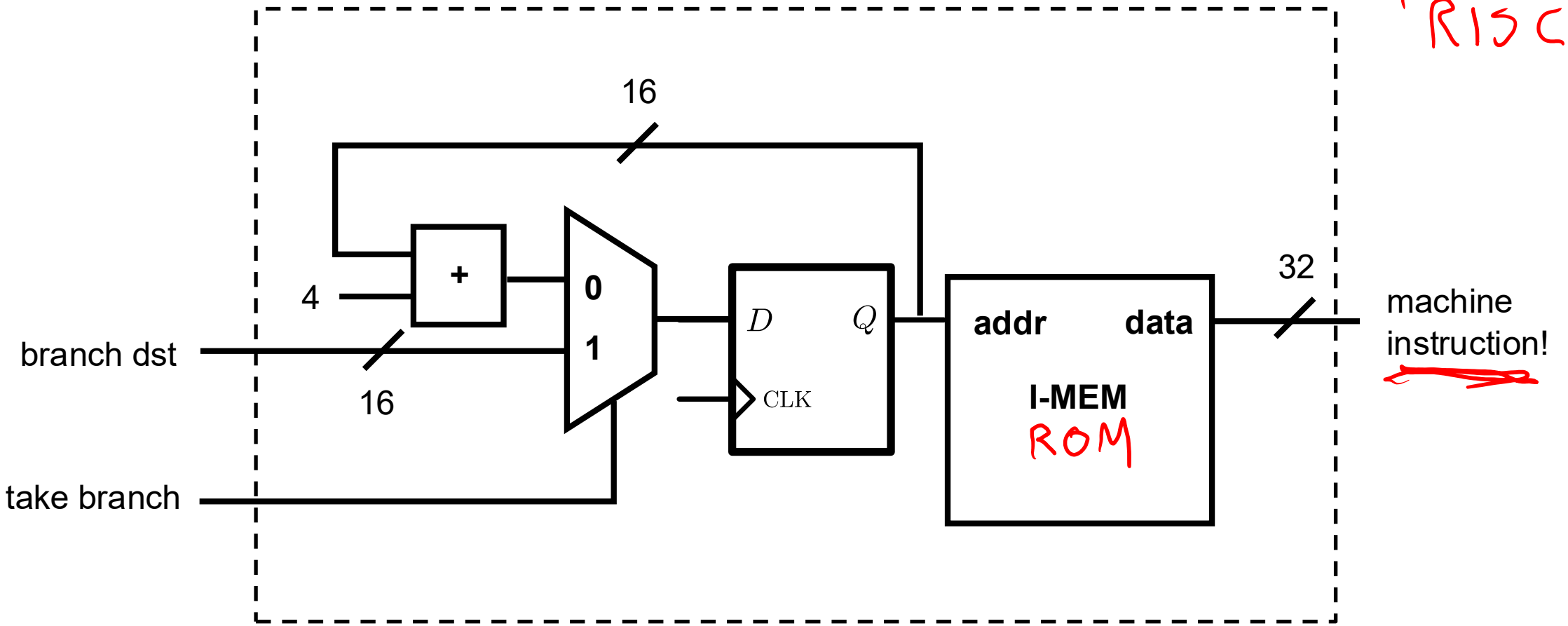
- Basic Datapath Components (idealized)
  - Register File
  - Memory Management Unit
  - Arithmetic Logic Unit (ALU)
  - Routing Elements

ROMs and RAMs

Muxes, adders, encoders, etc

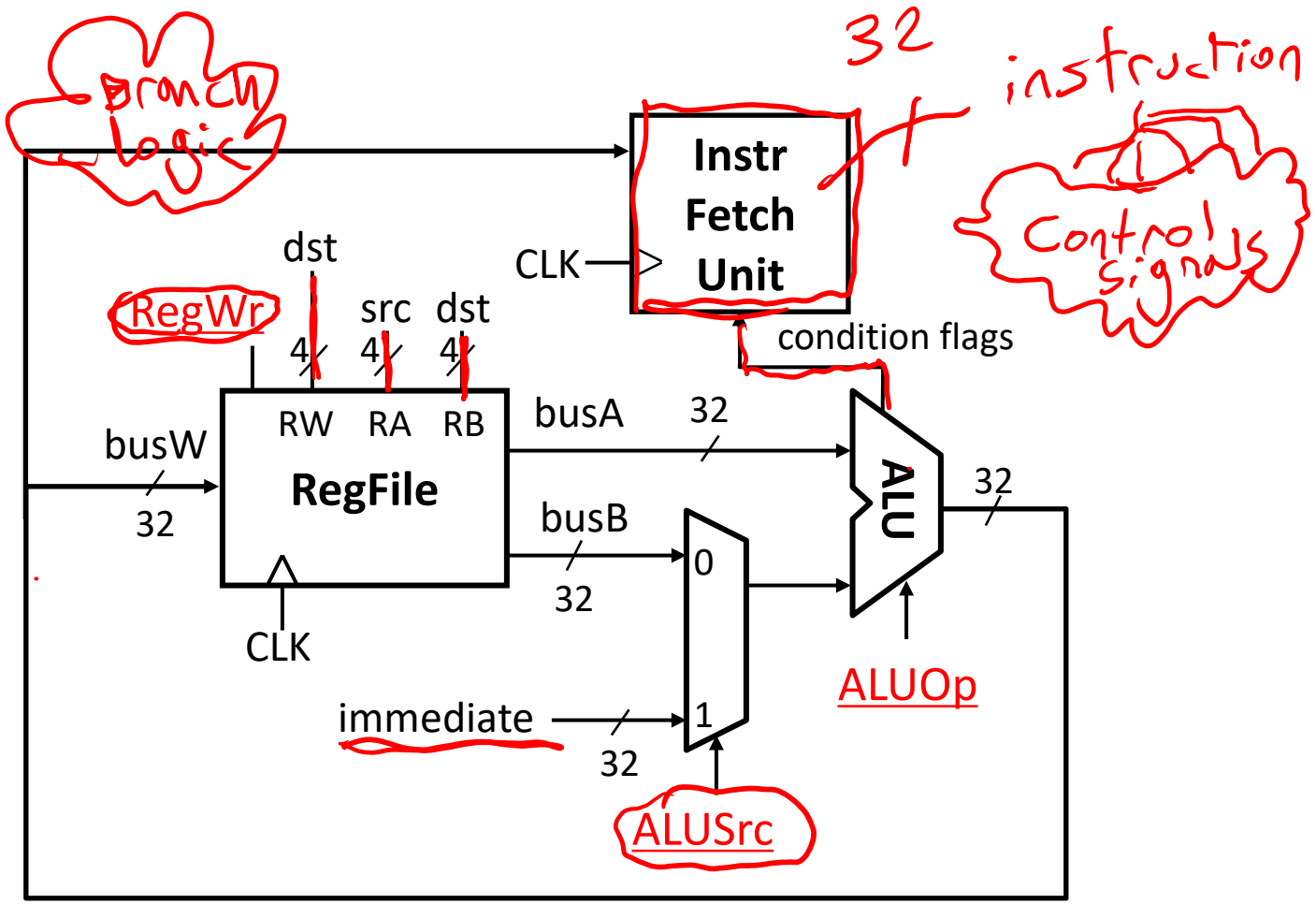
Instruction Fetch Unit (16 bit memory, 32 bit instructions)

"RISC"



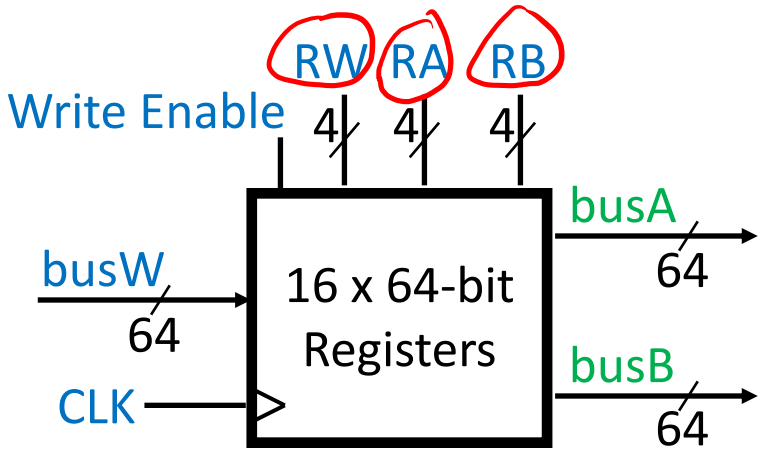
Datapath Teaser

- Super-simplified and incomplete example of a CPU datapath
  - Assumes instructions of the form: `operation src, dst`
- No main memory in this example, just registers
- Signals in **red** are set by Control based on the instruction's bits

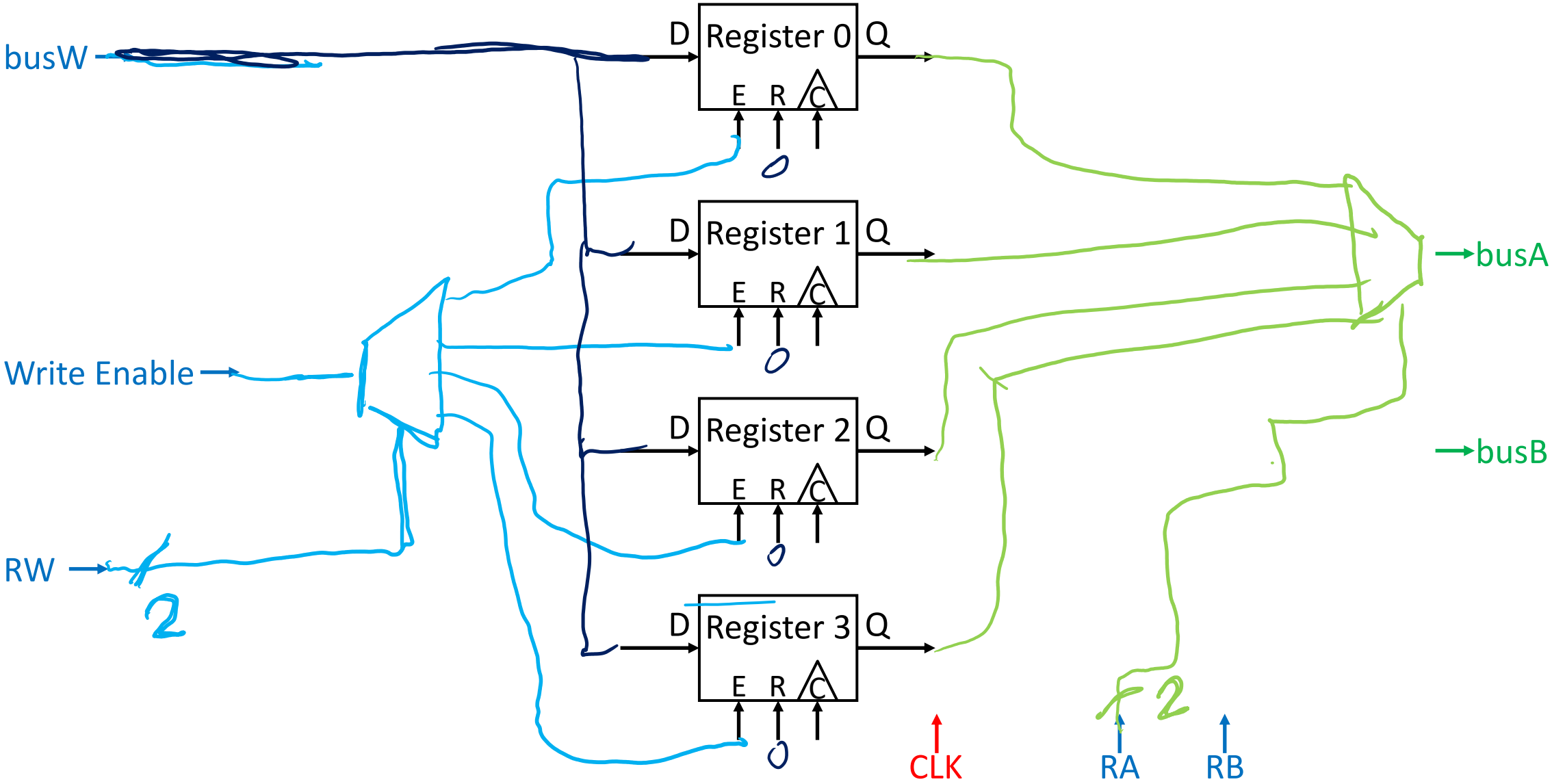


Zoom in on the Register File

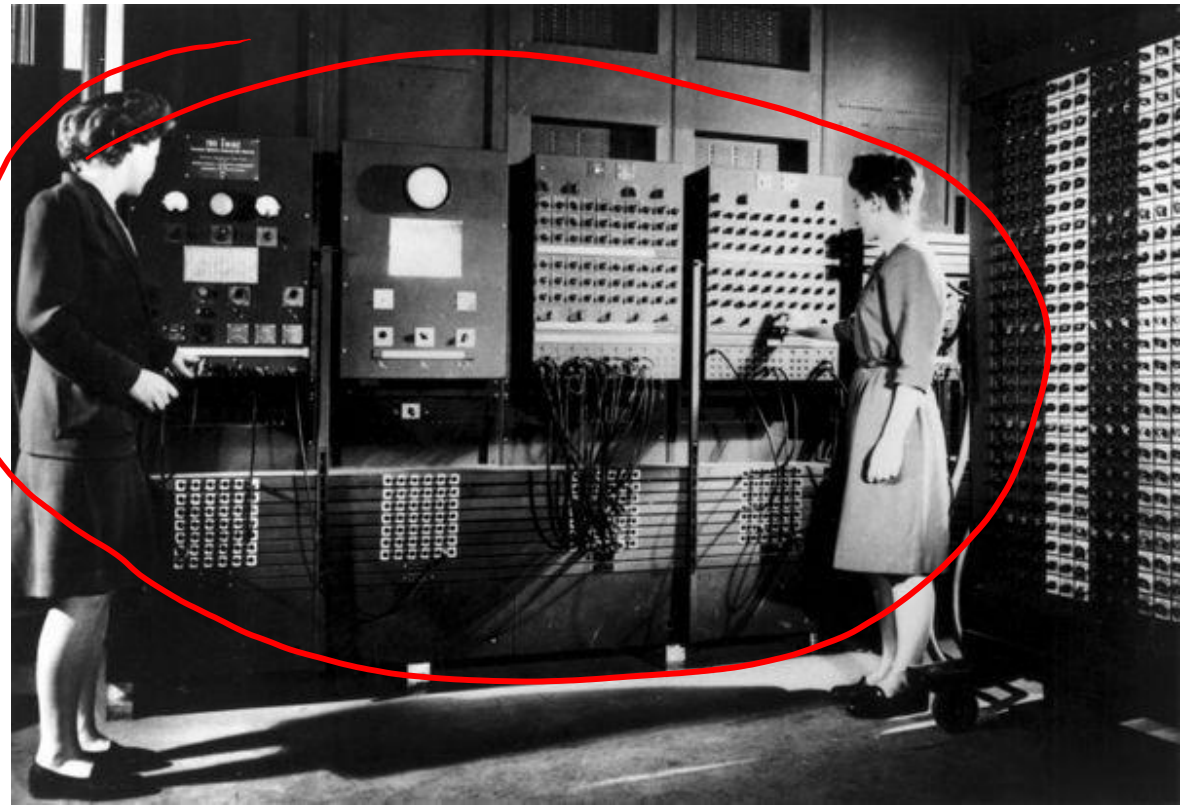
- Just a regular run o' the mill SRAM
- Contains all programmer-accessible registers
  - Output buses busA and busB
  - Input bus busW
- Register selection
  - Place data of registers RA/RB (numbers) onto busA/busB
  - Store data on busW into register RW (number) when Write Enable is 1

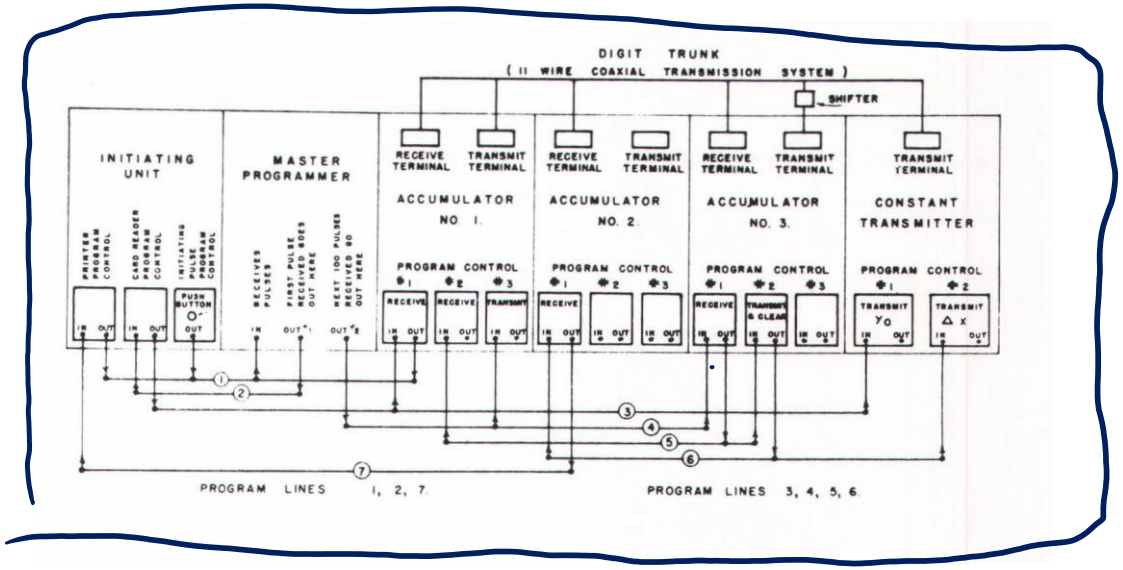


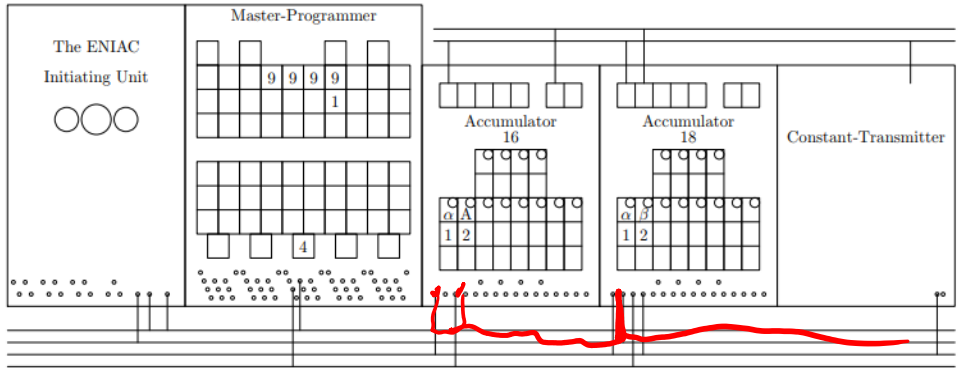
Simple Register File (4 Registers)



Early Language







Above: Frances Bilas and Betty Jean Jennings in front of the ENIAC

Right: ENIAC program sheets

http://www.computerhistory.org/revolution/birth-of-the-computer/4/78 https://xii.hope.net/audio/C02_ENIAC_The_Hack_That_Started_It_All.mp3
https://www.cs.drexel.edu/~bls96/eniac/eniac5.pdf http://ds-wordpress.haverford.edu/bitbybit/bit-by-bit-contents/chapter-seven/7-5-assembly-language-programming-2/

Miso Moment

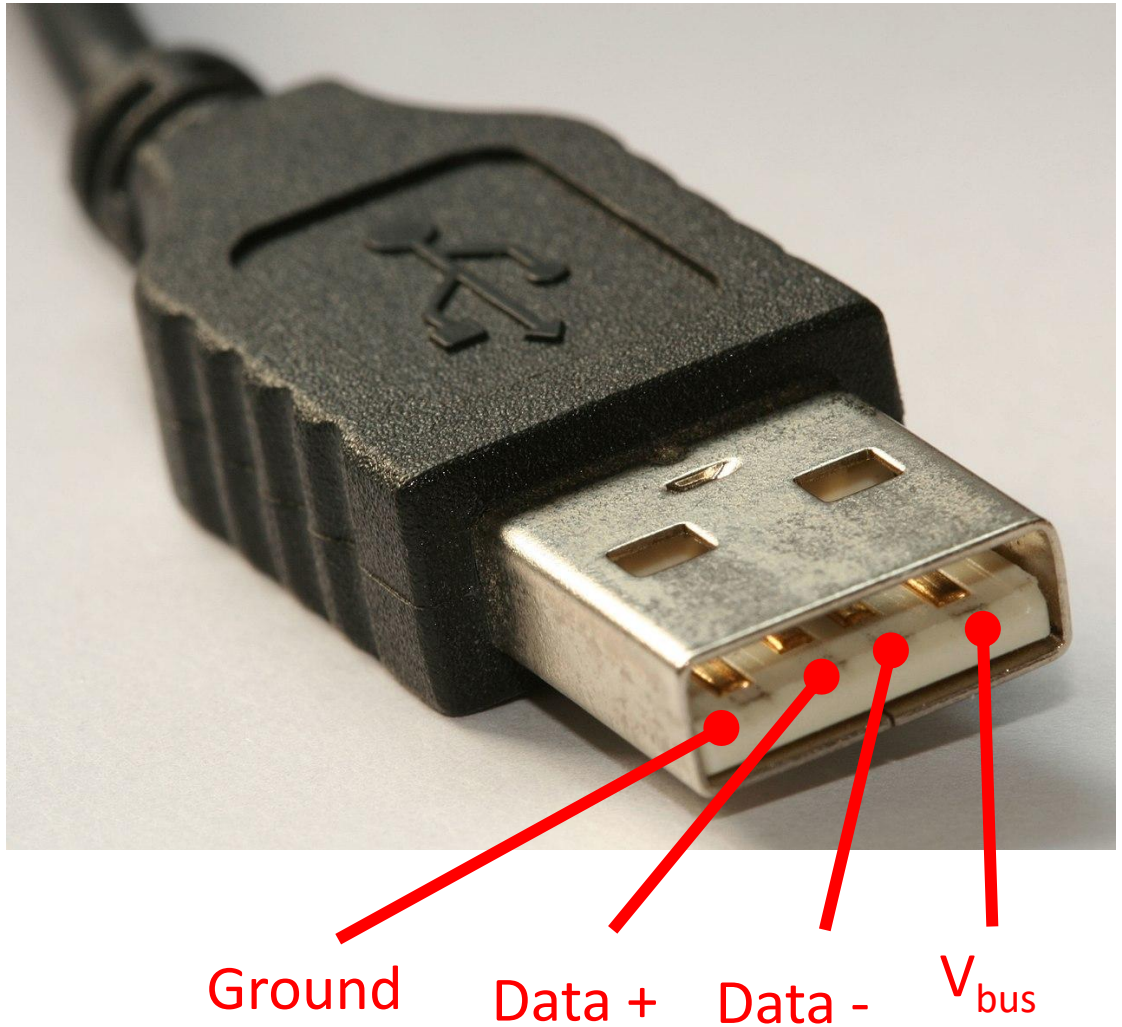
Outline

- Memory
- CPU teaser
- **Serial I/O**
- FPGAs

Data Transfer

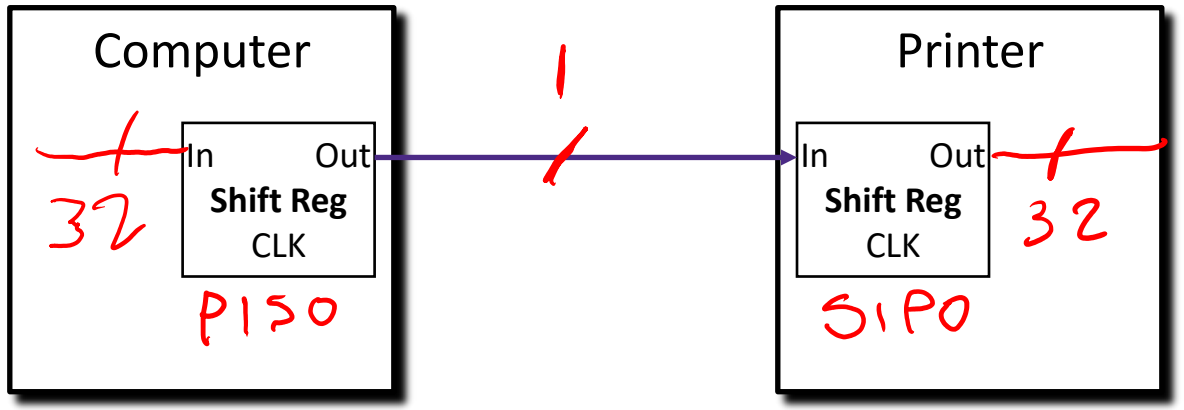
https://commons.wikimedia.org/wiki/Category:USB_2_Standard-A_plugs#/media/File:Type_A_USB_Connecter_alt.jpg

- 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:

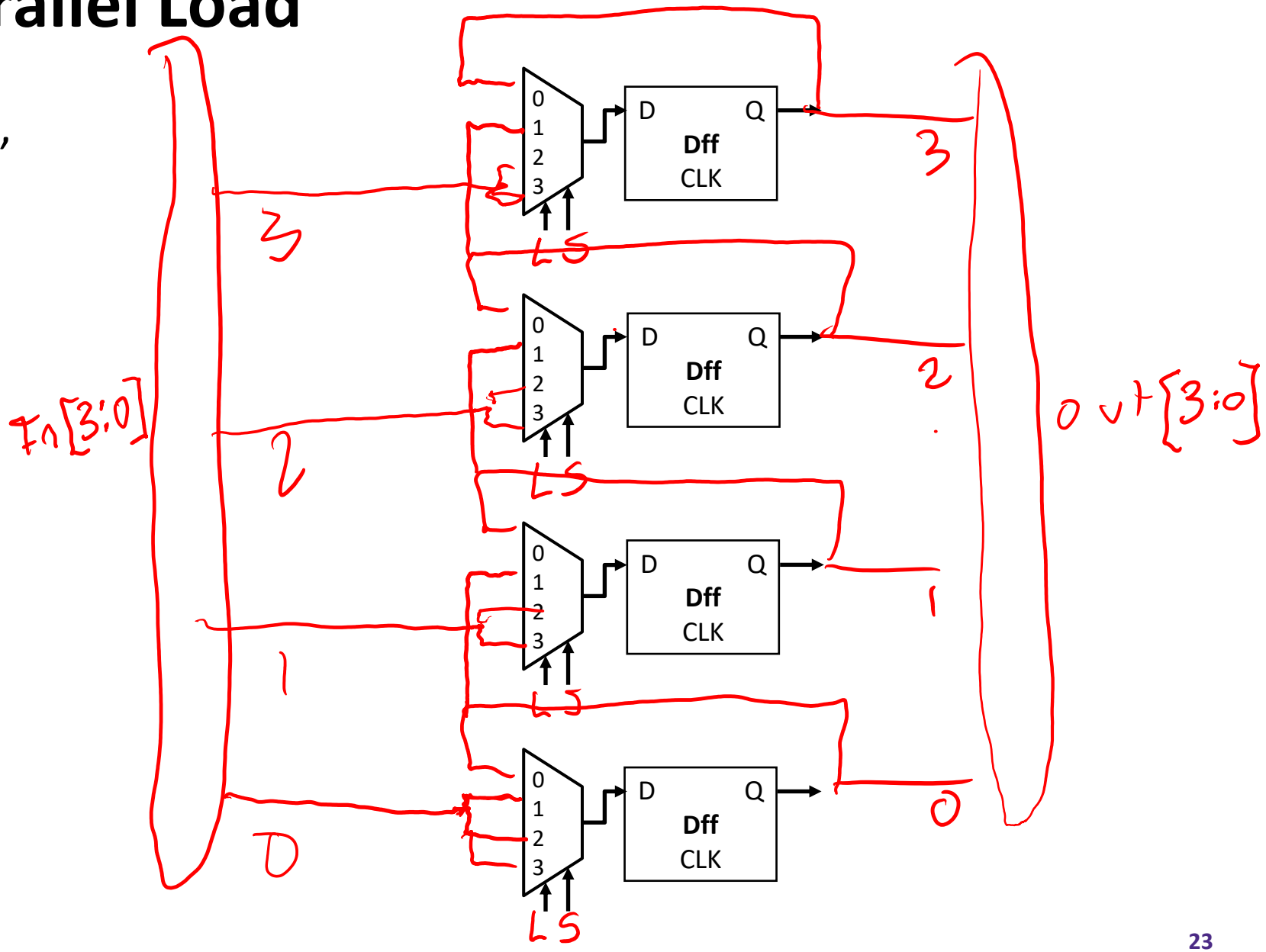


- In general, shift registers can allow for **either** or **both** modes (S for serial, P for parallel) at input and output
  - Examples: SISO, SIPO, PISO

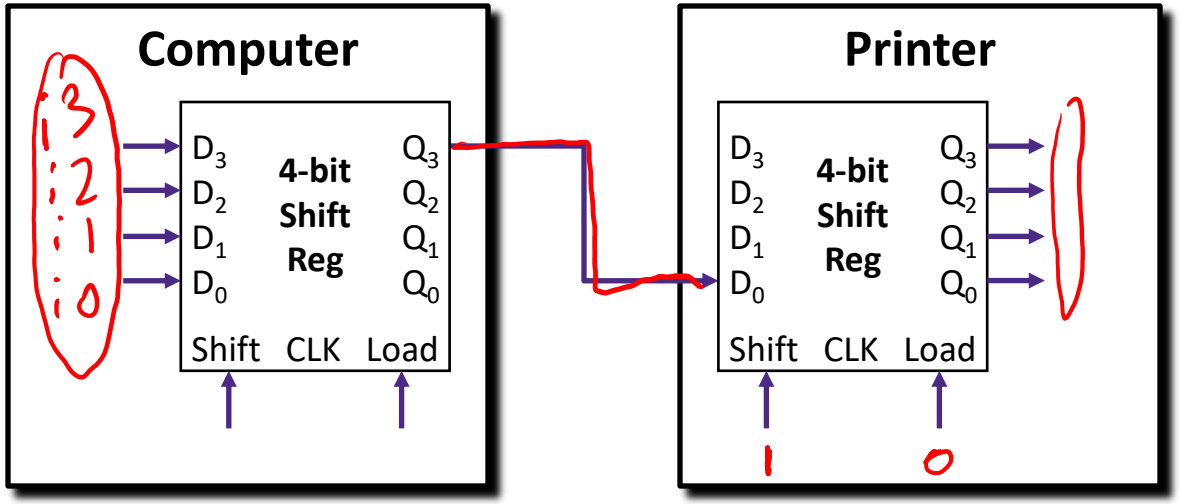
Shift Register w/Parallel Load

- **Note:** To avoid extra input, use D_0 input as "shift in"

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



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	i0	i1	i2	i3
1	0	X	i0	i1	i2
1	0	X	X	i0	i1
1	0	X	X	X	i0
X	X	X	X	X	X
X	X				

Shift	Load	Q0	Q1	Q2	Q3
1	0	X	X	X	X
1	0	X	X	X	X
1	0	i3	X	X	X
1	0	i2	i3	X	X
1	0	i1	i2	i3	X
1	0	i0	i1	i2	i3
1	0				

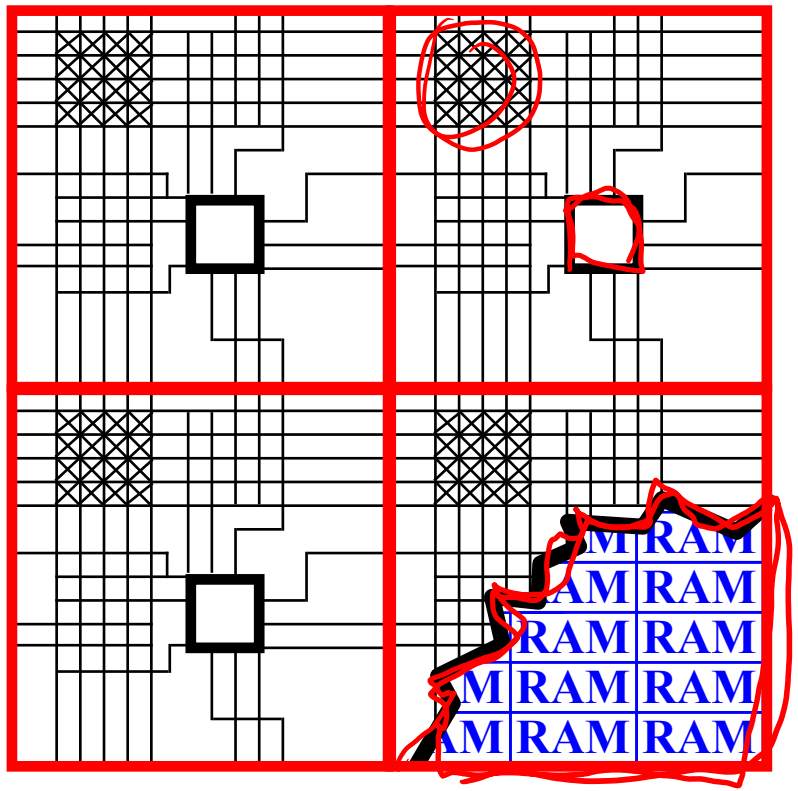
Shifter in Verilog

```
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
```

Outline

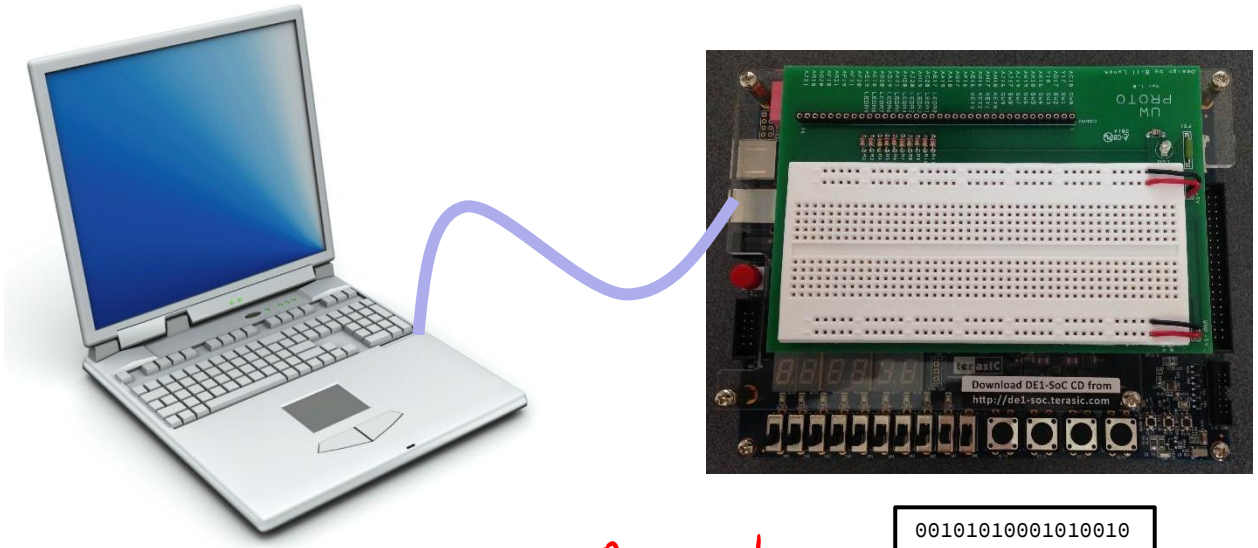
- Memory
- CPU teaser
- Serial I/O
- **FPGAs**

Field Programmable Gate Arrays (FPGAs)



- Logic cells ( ) embedded in a general routing structure
- Logic cells usually contain:
  - 6-input Boolean function calculator
  - Flip-flop (1-bit memory)
- All features are electronically (re)programmable

Using an FPGA




```
//  Verilog  code  for  2-input
multiplexer

module AOI (F, A, B, C, D);
  output F;
  input A, B, C, D;

  assign  F  =  ~((A  &  B)  |  (C  &
D));
endmodule

module MUX2 (V, SEL, I, J);    //
2:1 multiplexer
  output V;
  input SEL, I, J;
  wire SELB, VB;

  not G1 (SELB, SEL);
  AOI G2 (VB, I, SEL, SELB, J);
  not G3 (V, VB);
endmodule
```

Verilog

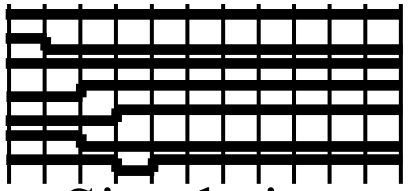
FPGA CAD Tools

Quartus

```
00101010001010010
10010010010011000
10101000101011000
10101001010010101
00010110001001010
10101001111001001
01000010101001010
10010010000101010
10100101010010100
01010110101001010
01010010100101001
```

Bitstream

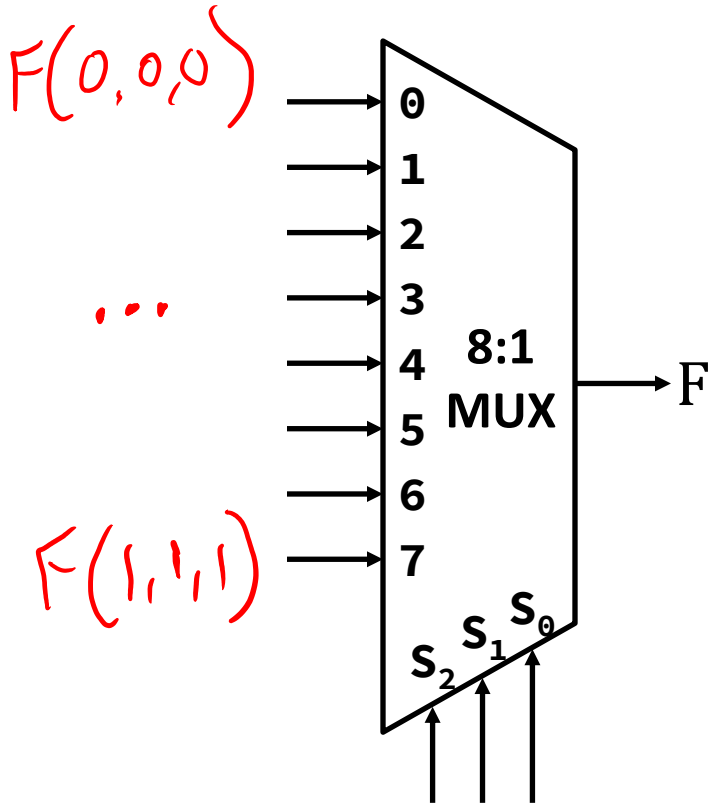
Modelsim



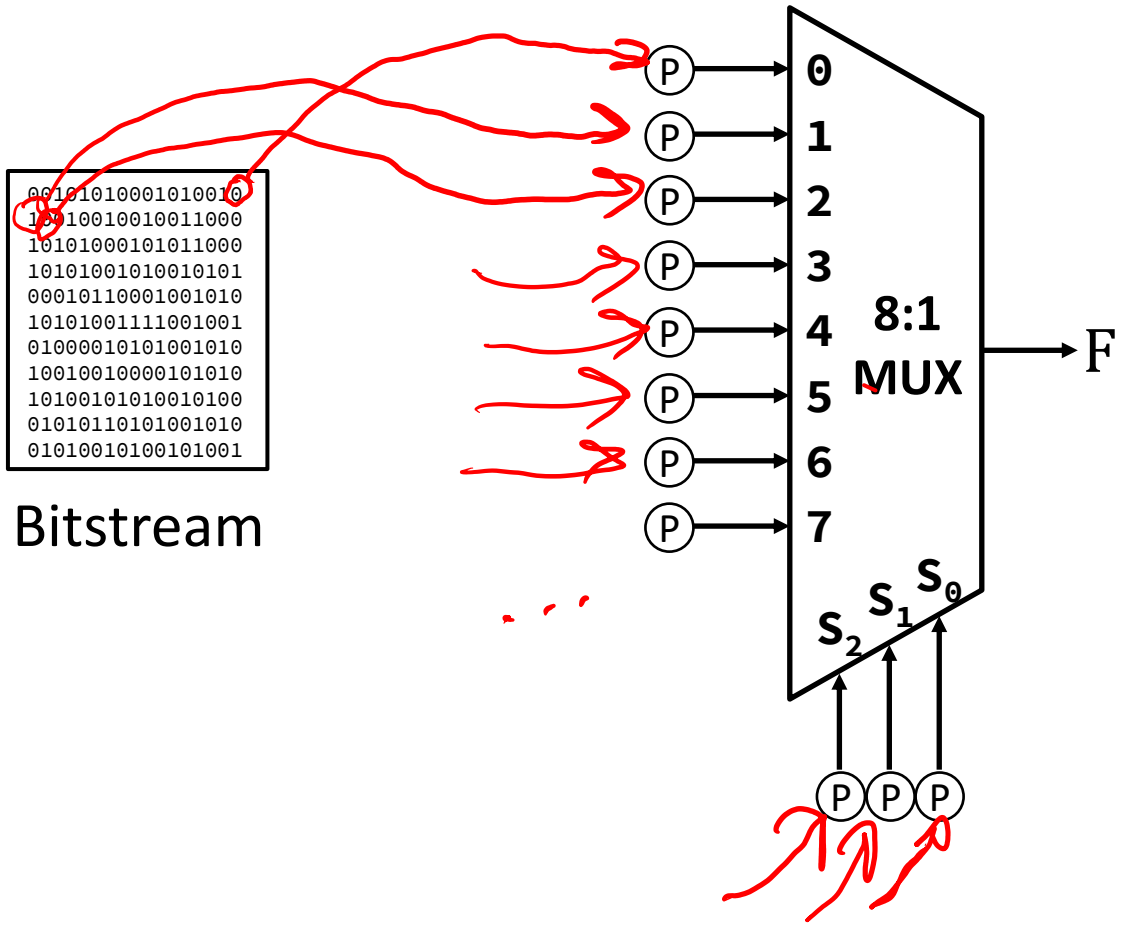
Simulation

FPGA Combinational Logic

- Create arbitrary combinational binary function F(A,B,C) using MUXes
  - Creates a **Lookup Table (LUT)**



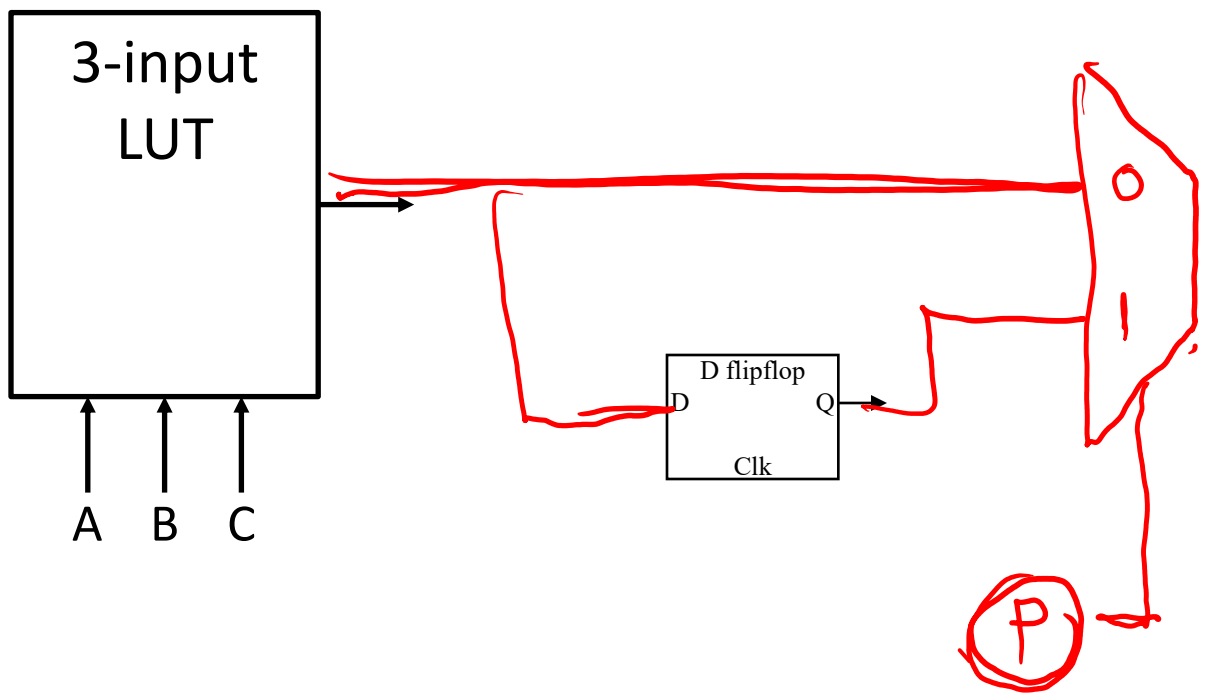
FPGA Programming



Ⓟ = 1 memory cell (stores 1 bit of info)

FPGA Sequential Logic

- How do we put DFF's onto LUT outputs only when we need them?

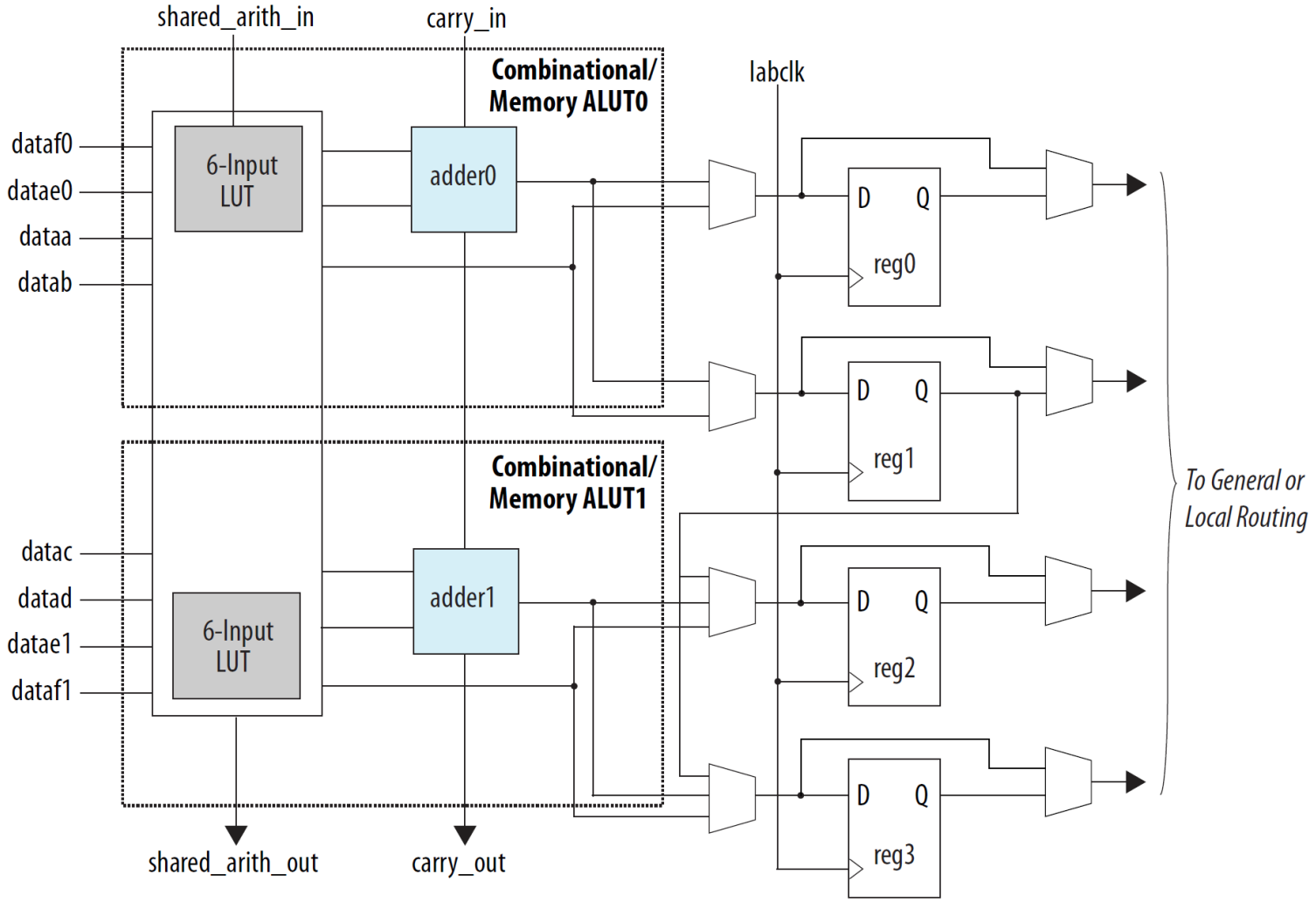


- Creates a Logic Cell (or Logic Element)

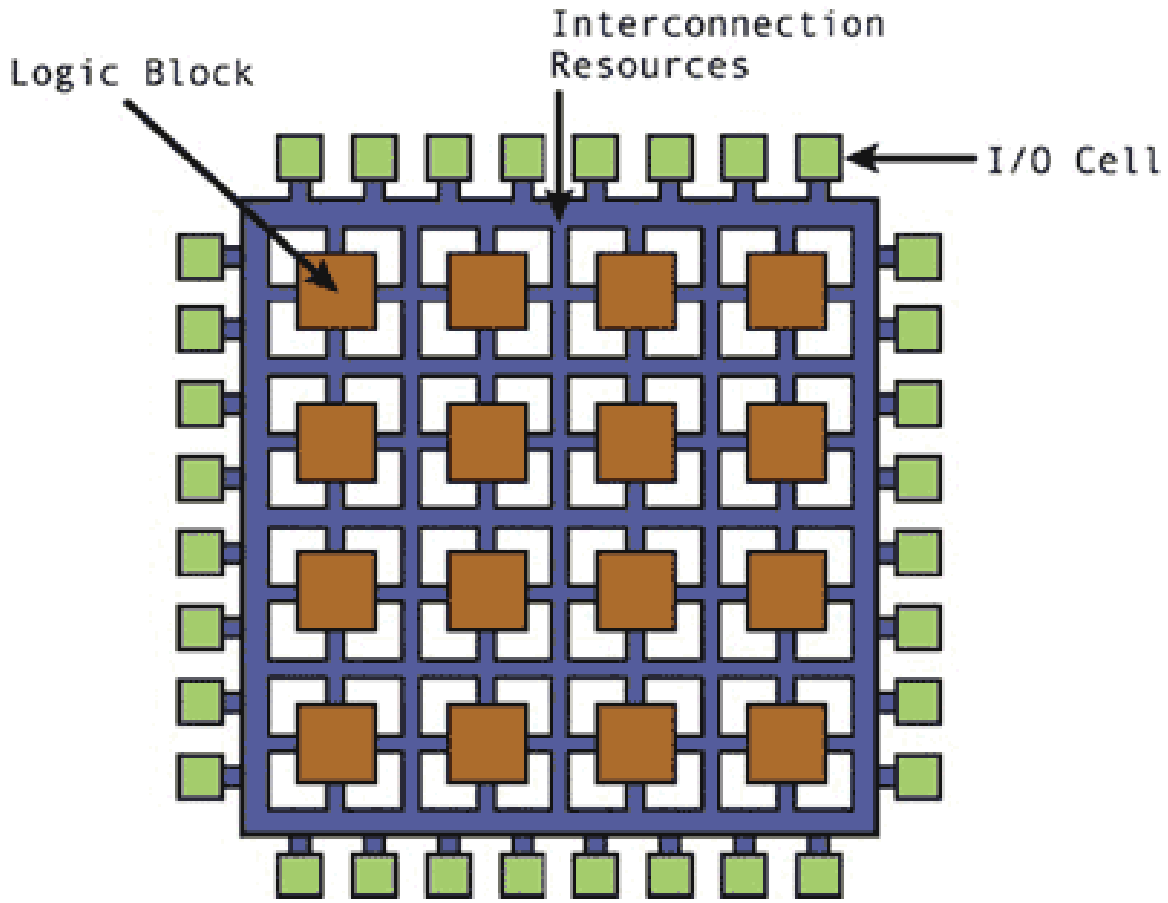
Cyclone V Adaptive Logic Modules

https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-v/cv_5v2.pdf

Figure 1-5: ALM High-Level Block Diagram for Cyclone V Devices

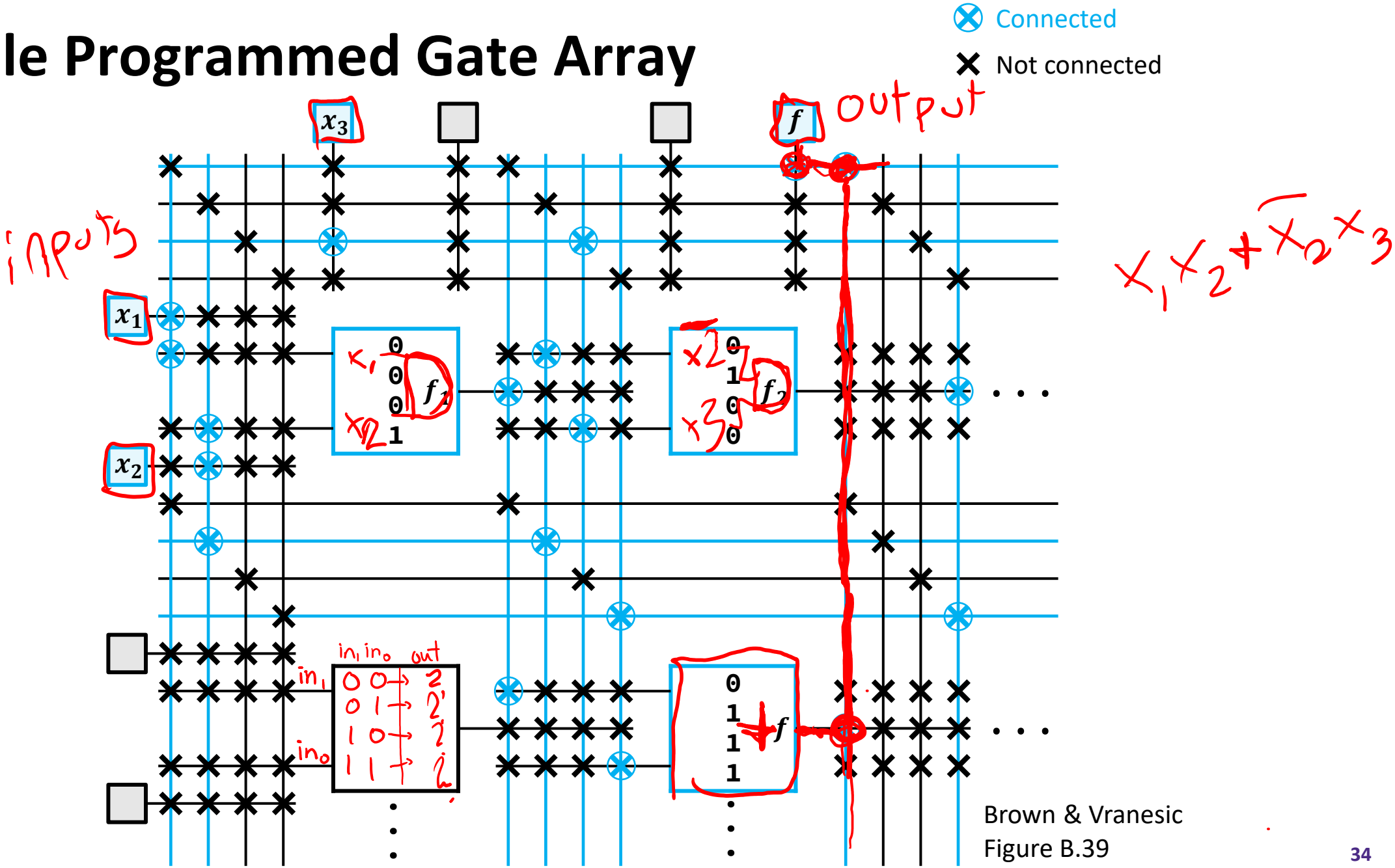


Generic FPGA Logic Layout



http://www.chipdesignmag.com/print.php?articleId=434?issueId=16

Example Programmed Gate Array



Brown & Vranesic
Figure B.39

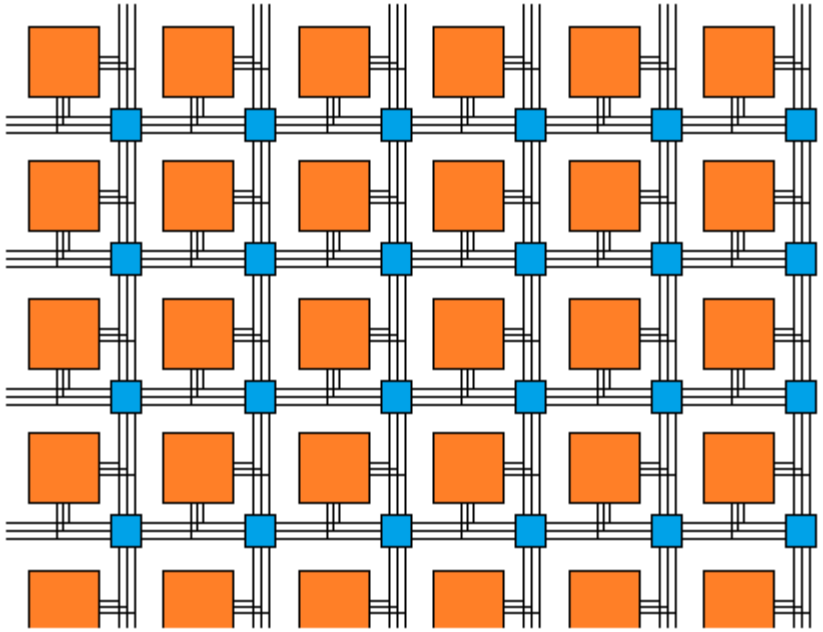
FPGA CAD

- CAD = "Computer-Aided Design"

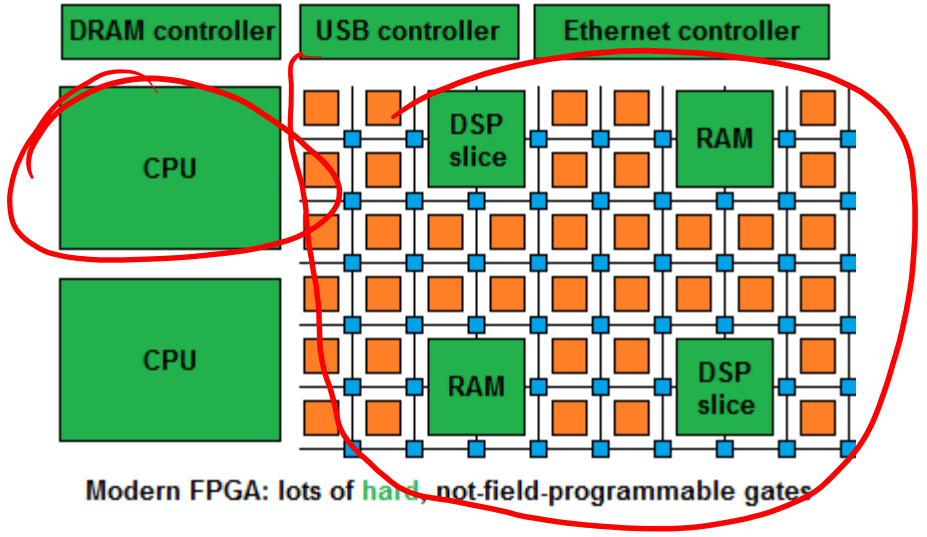
1) **Tech Mapping:** Convert Verilog to LUTs
2) **Placement:** Assign LUTs to specific locations
3) **Routing:** Wire inputs to outputs
4) **Bitstream Generation:** Convert mapping to bits

Gate Array vs. SoC

- SoC = “System on a Chip”

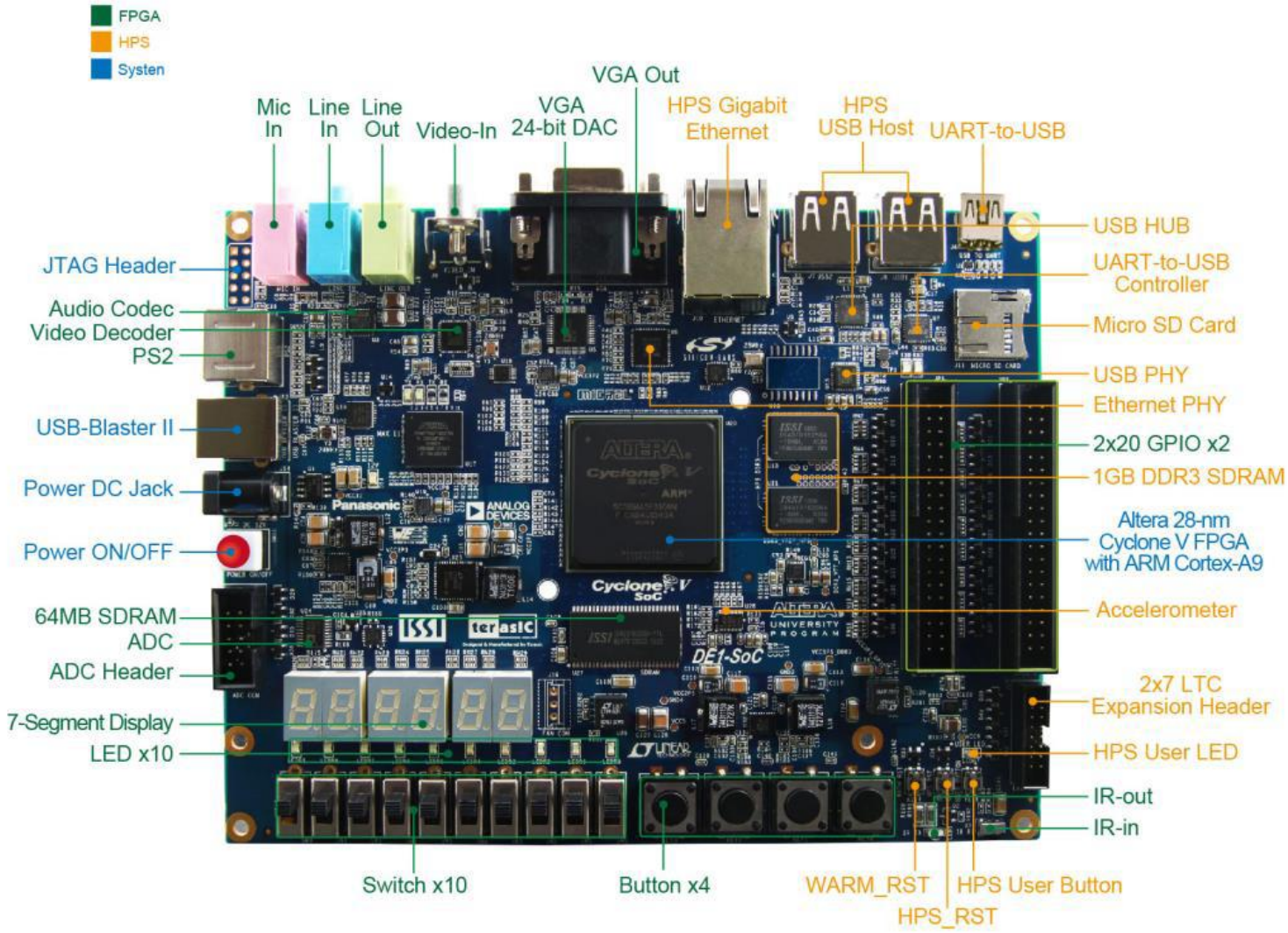


"Clean slate" FPGA: programmable gates and routers

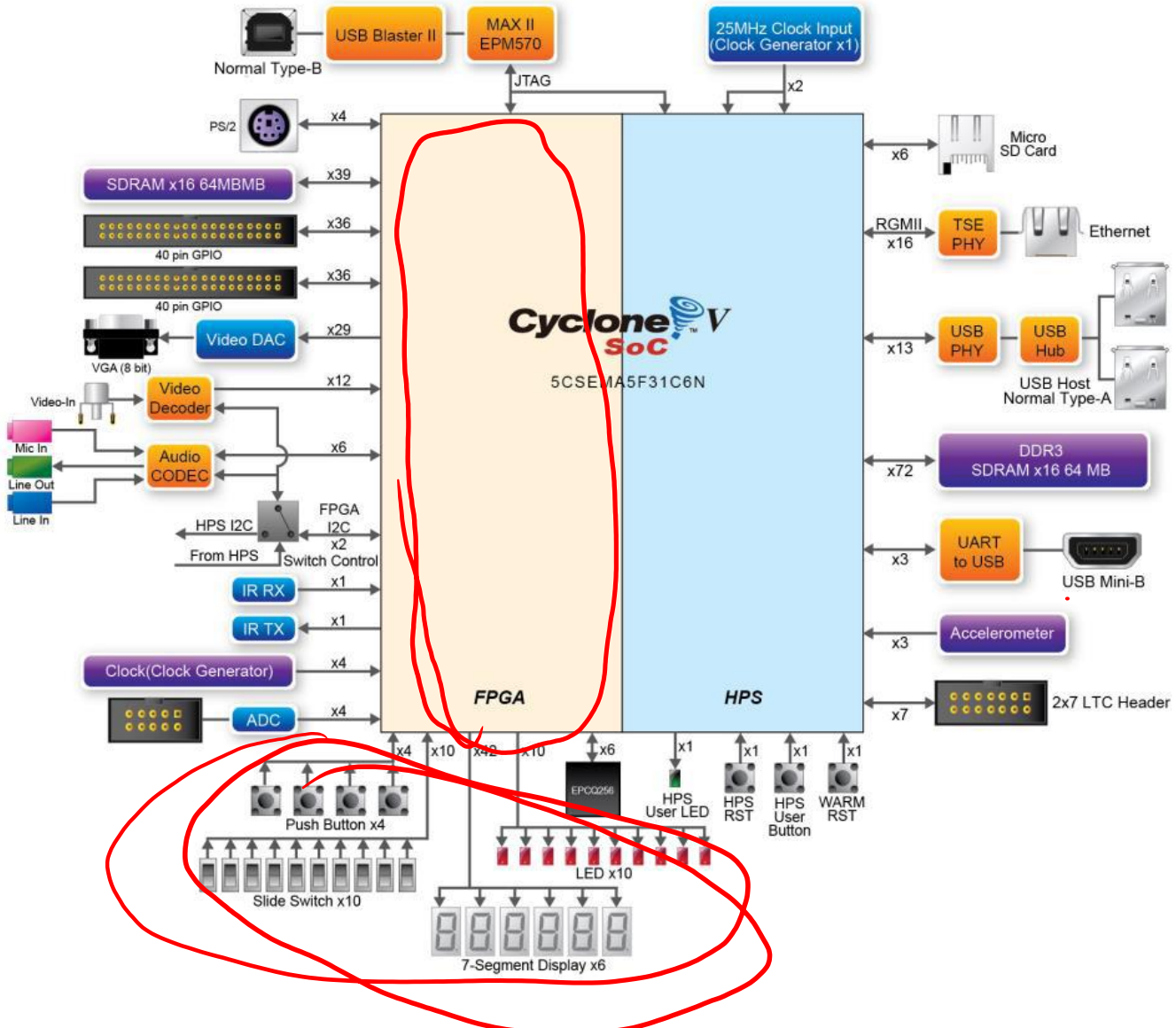


Modern FPGA: lots of hard, not-field-programmable gates

DE1-SoC Board



DE1-SoC Board



FPGAs vs. CPUs

- Individual CPUs designed to handle sequential instruction execution
  - Design and layout determined by architecture
  - Computations "limited" to instruction set
  - Powerful, but also requires things like OS
- FPGAs
  - Programmable, so specially-designed logic for application
    - But can be difficult to specify certain applications
  - Good for parallelizing computations
    - Can perform separate computations if separated via routing