

Intro to Digital Design

Computer Components, FPGAs

Instructor: Justin Hsia

Teaching Assistants:

Emilio Alcantara

Eujean Lee

Naoto Uemura

Pedro Amarante

Wen Li

Relevant Course Information

- ❖ Lab 8 – Project
 - Reports due Friday, May 31 @ 11:59 pm
 - Project check-ins this week during demos
 - Demos can be scheduled outside of the lab hours by making a private Ed Discussion post
- ❖ Lab kit (+ Okiocam) return when you are done
- ❖ **Quiz 3** is next week: Tuesday, May 28 @ 1:40 pm
 - 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

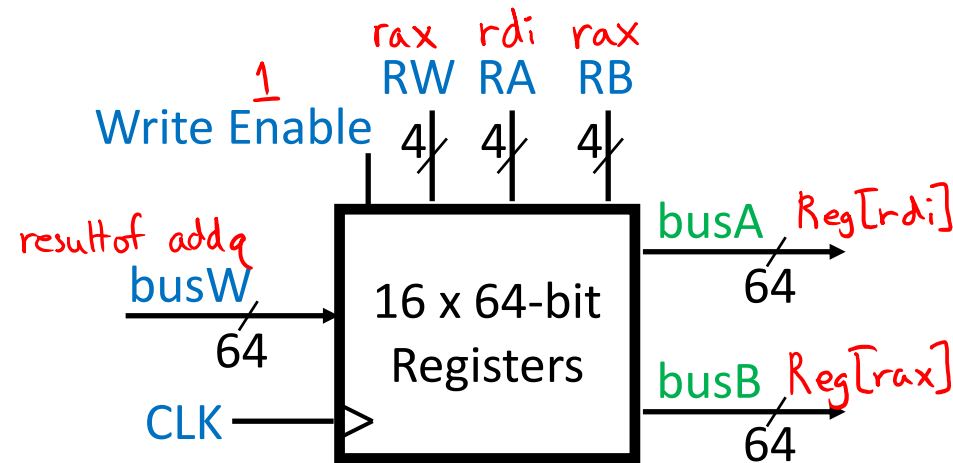
Outline

- ❖ **Computer Components (Cont.)**
 - Register File
 - Datapath teaser
 - Serial Communication
- ❖ FPGAs
- ❖ Course Wrap-up

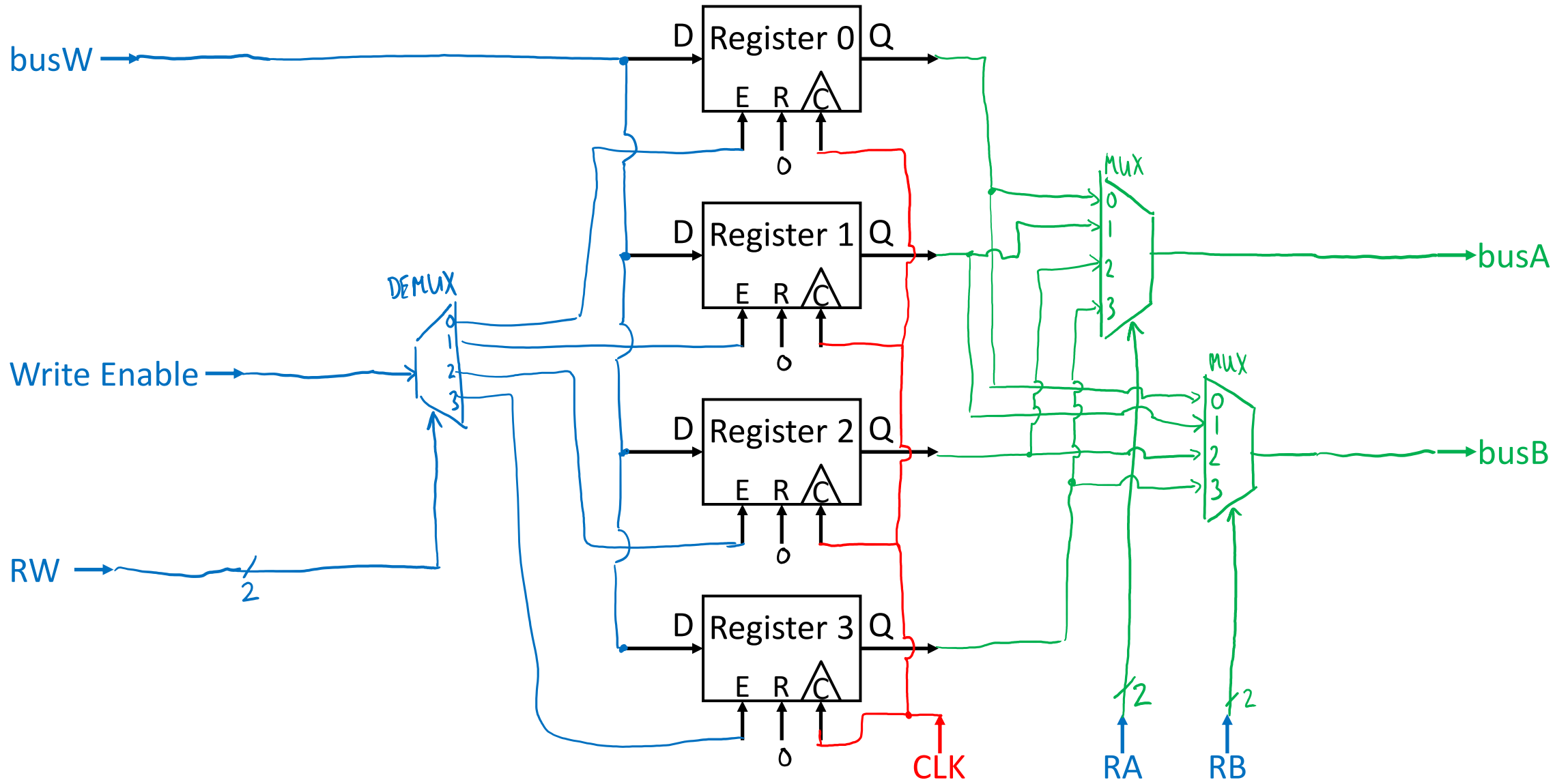
Storage Element: Register File

- ❖ 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

addq (%rdi), %rax
 need two register values

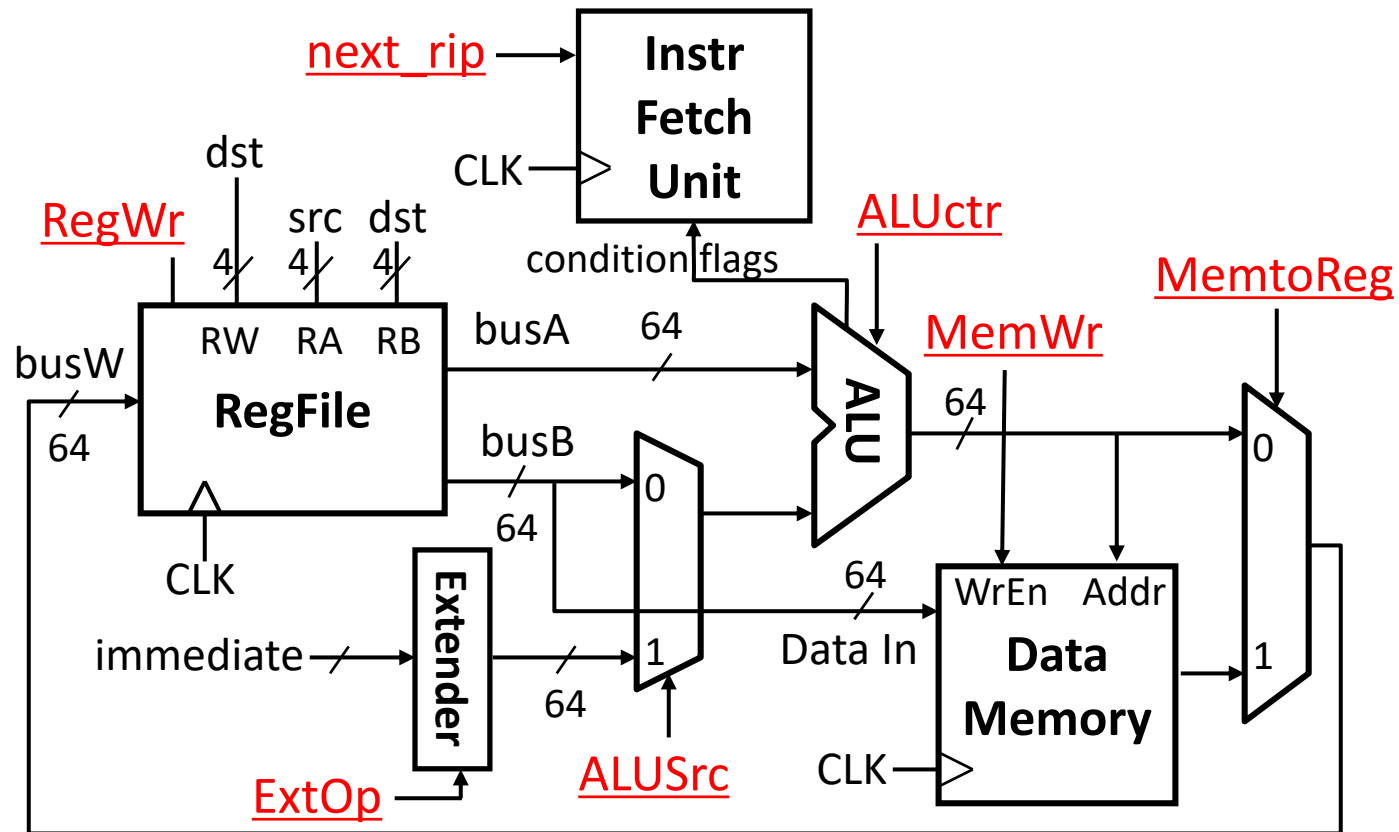


Simple Register File (4 Registers)



Datapath Teaser

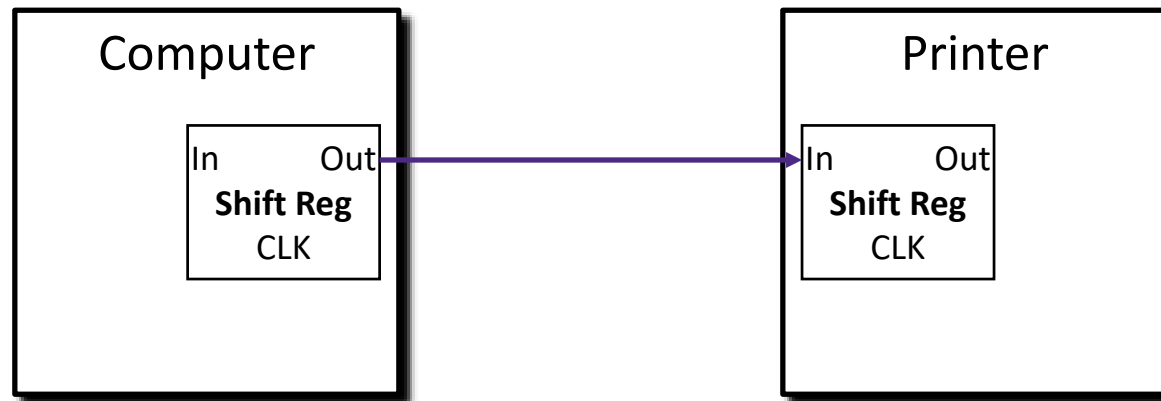
- ❖ Instructions of the form: operation src, dst
 - Signals in **red** are set by Control based on operation
 - This is inaccurate/incomplete but gives you a vague idea of connecting parts



Transfer of Data

- ❖ Two modes of communication
 - **Parallel:** all bits transferred at the same time
 - **Serial:** one bit transferred at a time

- ❖ Shift registers can be used for serial transfer:

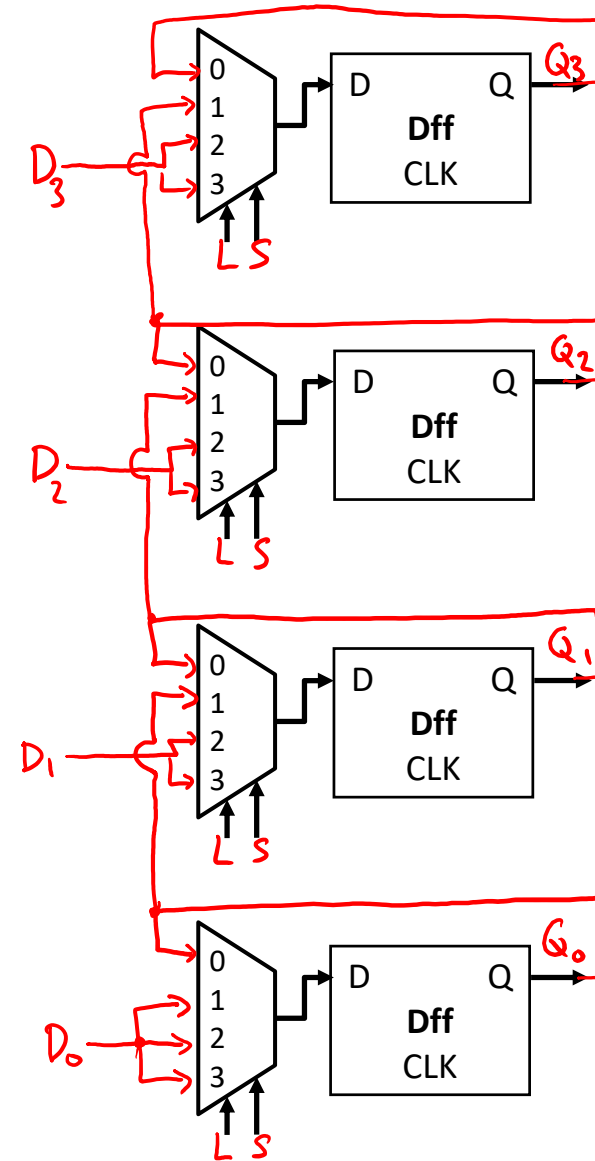


- ❖ In general, shift registers can allow for either or both modes (S for serial, P for parallel) at input and output
 - Examples: SISO, SIPO, PISO *no PIPD because that's a normal register!*

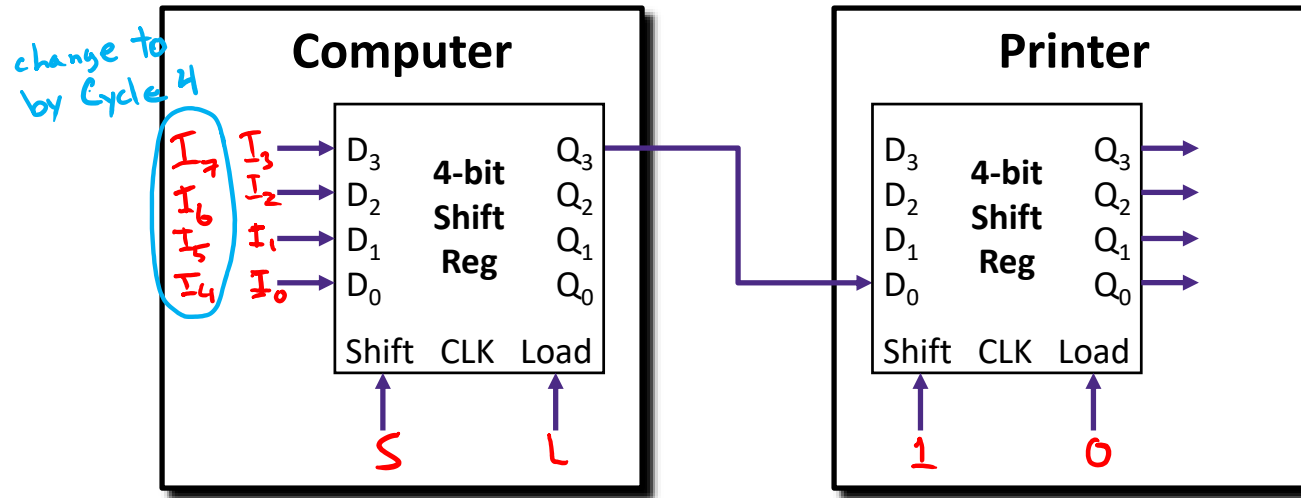
Shift Register w/Parallel Load

❖ **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	Shift	Load	Q0	Q1	Q2	Q3
0	X	1	X	X	X	X
1	1	0	I_0	I_1	I_2	I_3
2	1	0	X	I_0	I_1	I_2
3	1	0	X	X	I_0	I_1
4	X	1	X	X	X	I_0
5	1	0	I_4	I_5	I_6	I_7
6	1	0	X	I_4	I_5	I_6
			⋮			

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	I_1	I_2	I_3	X
1	0	I_0	I_1	I_2	I_3
1	0	I_7	I_0	I_1	I_2
			⋮		

Handwritten red annotations include arrows connecting the circled I_3 in the Computer table to the circled I_3 in the Printer table, and another arrow connecting the circled I_3 in the Printer table to the circled I_3 in the Computer table. A blue arrow points to the start of the Computer table at Cycle 4.

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) // load takes precedence
      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]; // last bit is shift out

endmodule // leftshifter
```

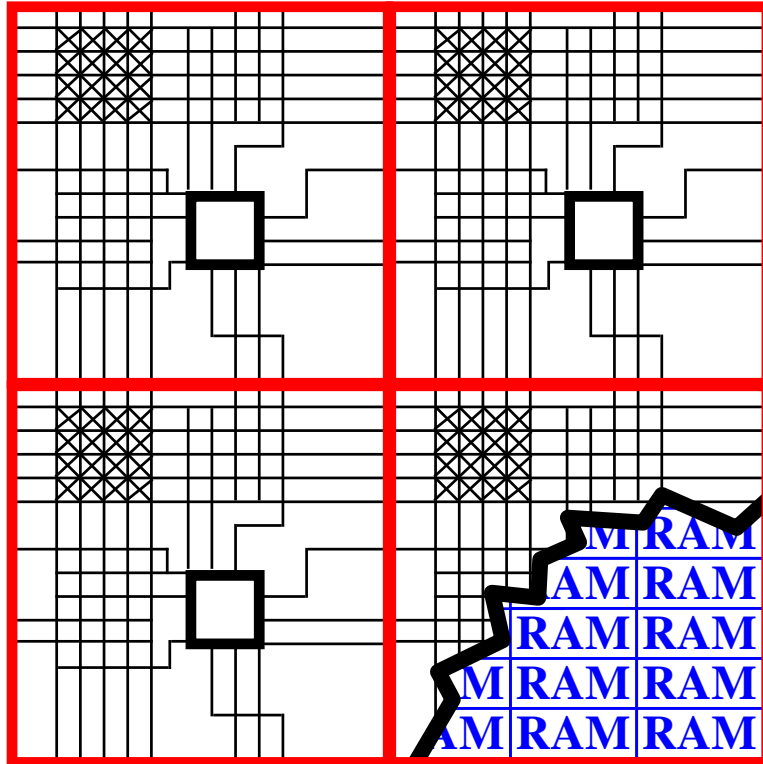



Technology Break

Outline

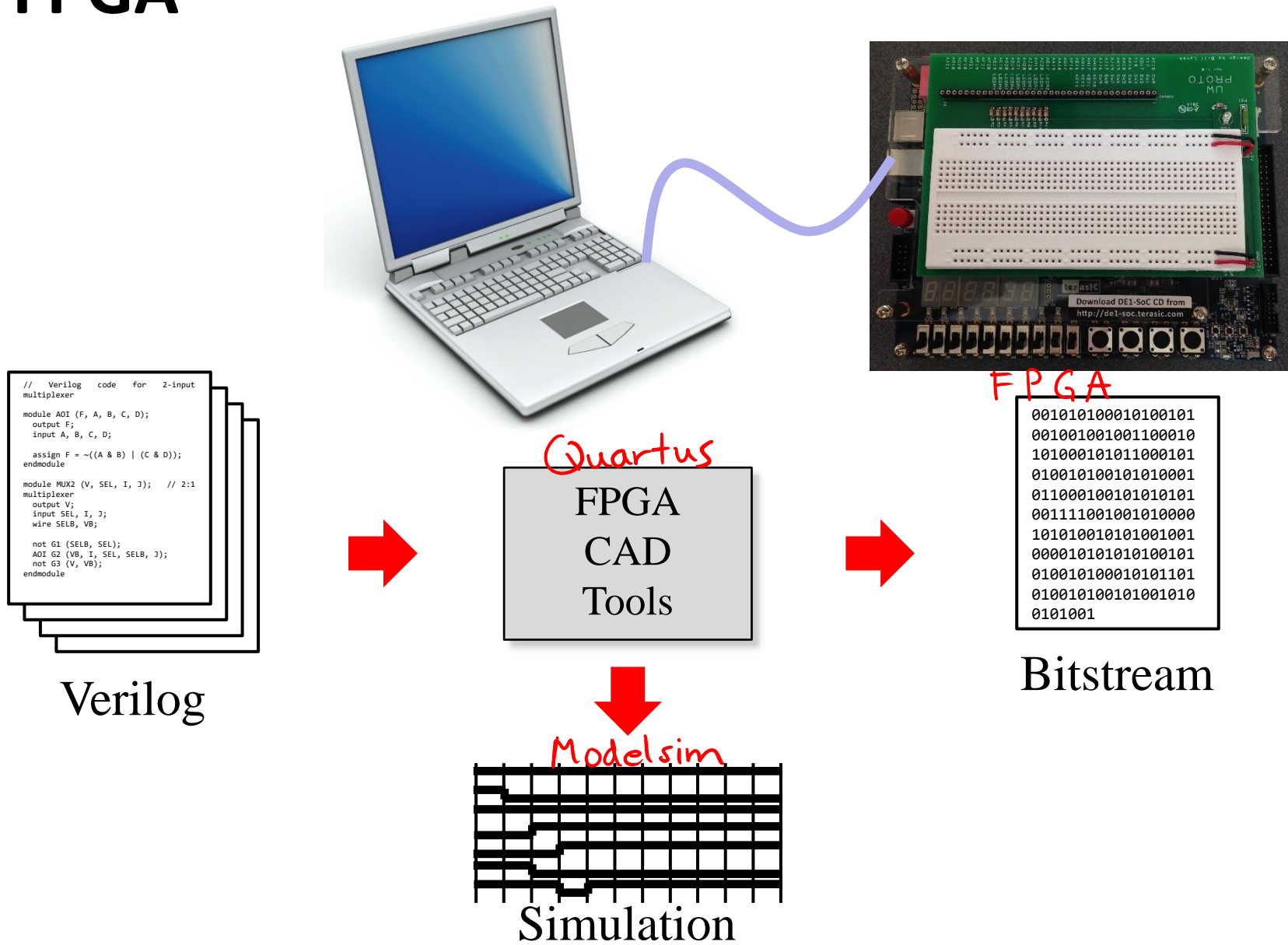
- ❖ Computer Components (Cont.)
 - Register File
 - Datapath teaser
 - Serial Communication
- ❖ **FPGAs**
- ❖ Course Wrap-up

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

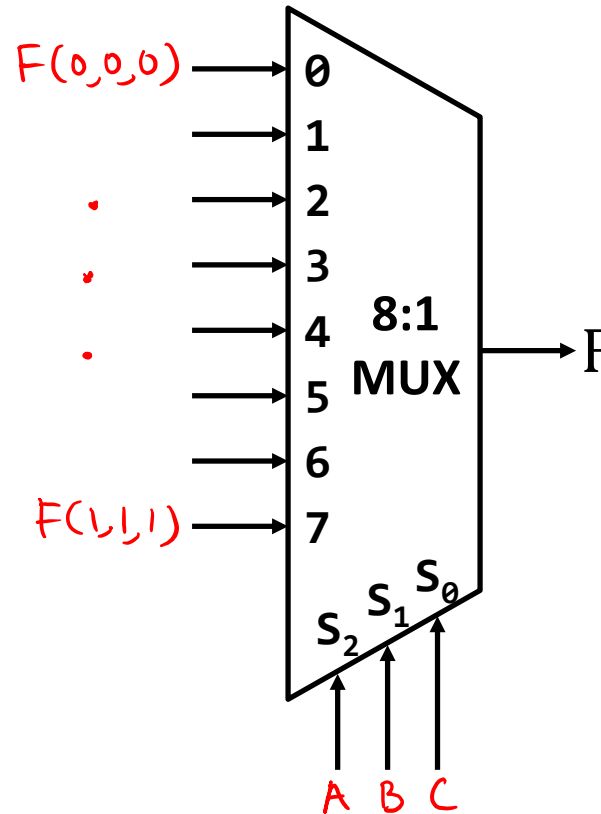
Using an FPGA



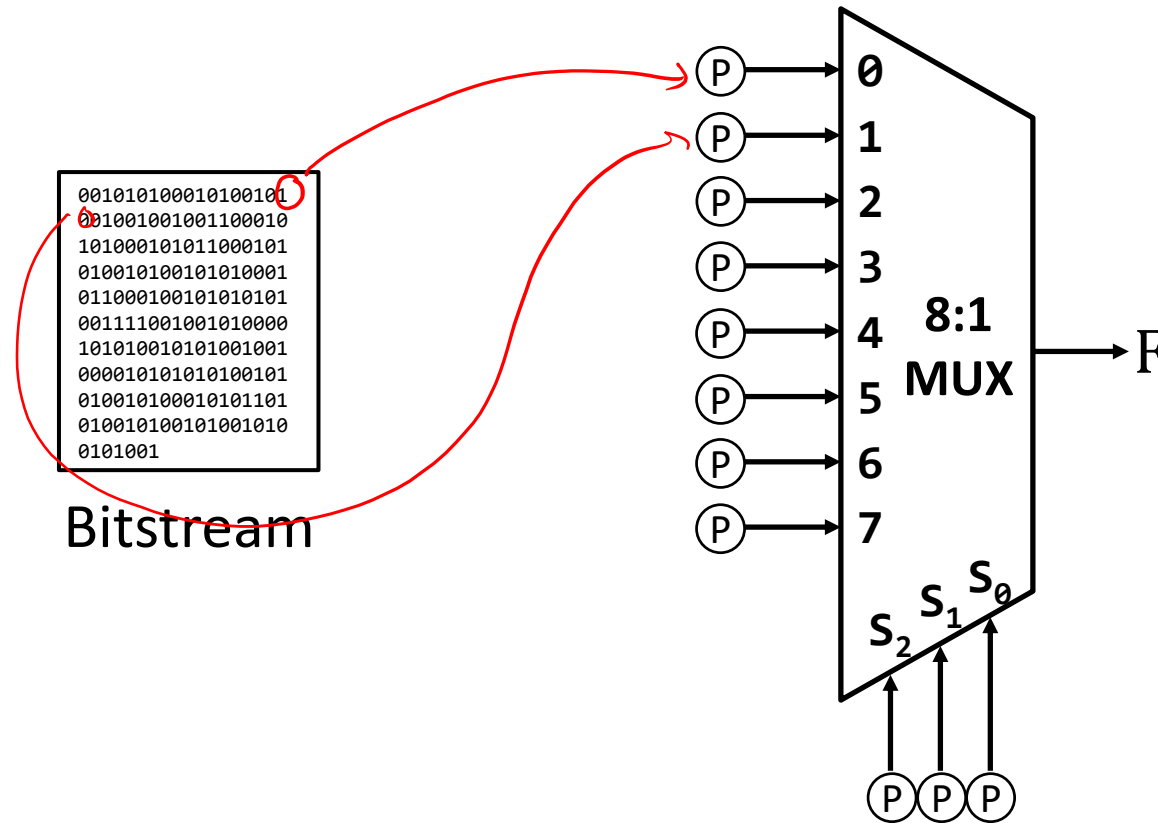
FPGA Combinational Logic

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

A	B	C	F
0	0	0	$F(0,0,0)$
0	0	1	$F(0,0,1)$
.	.	.	.
.	.	.	.
.	.	.	.
1	1	0	$F(1,1,0)$
1	1	1	$F(1,1,1)$



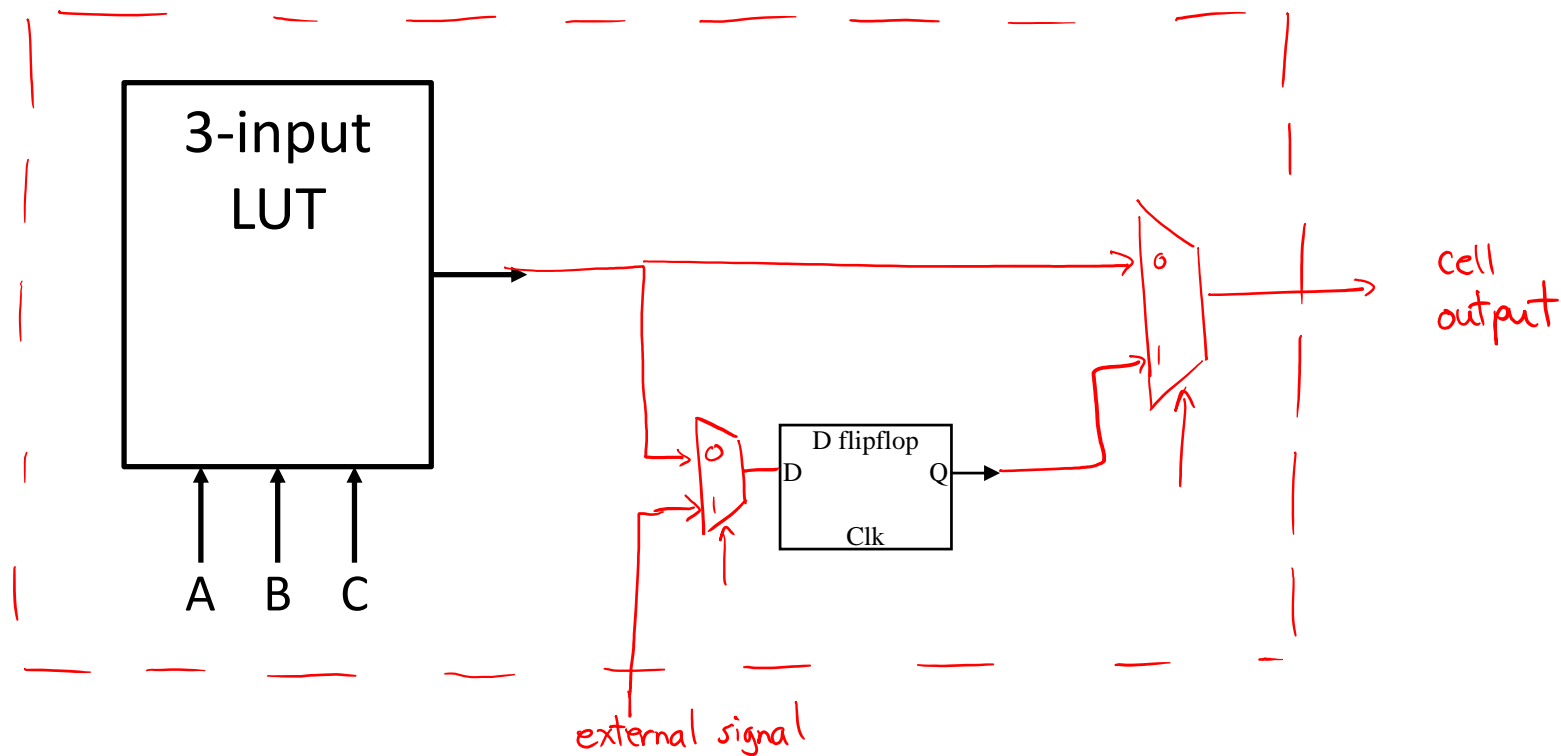
FPGA Programming



Ⓟ = 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?

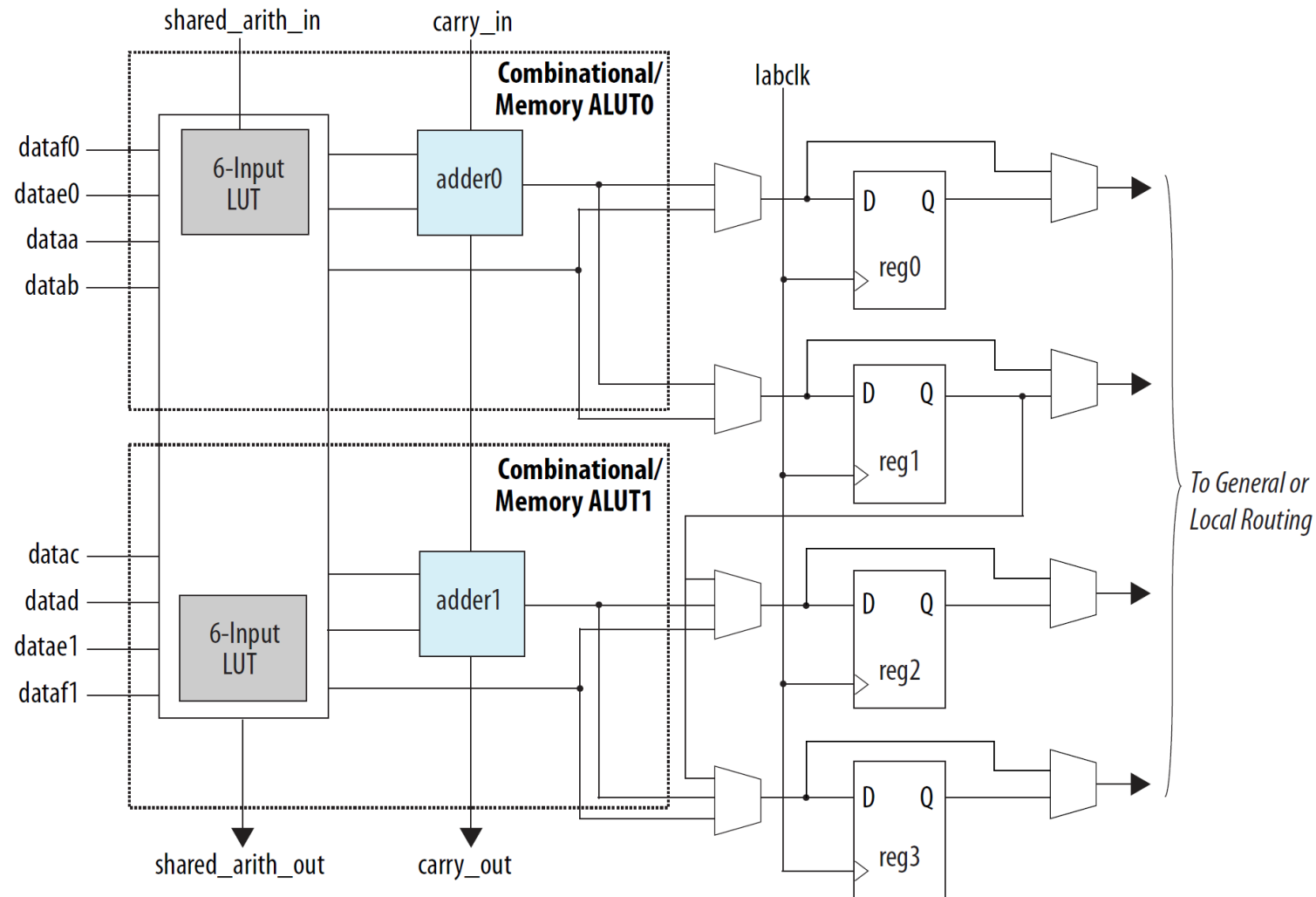


- ❖ Creates an Logic Cell (or Logic Element)

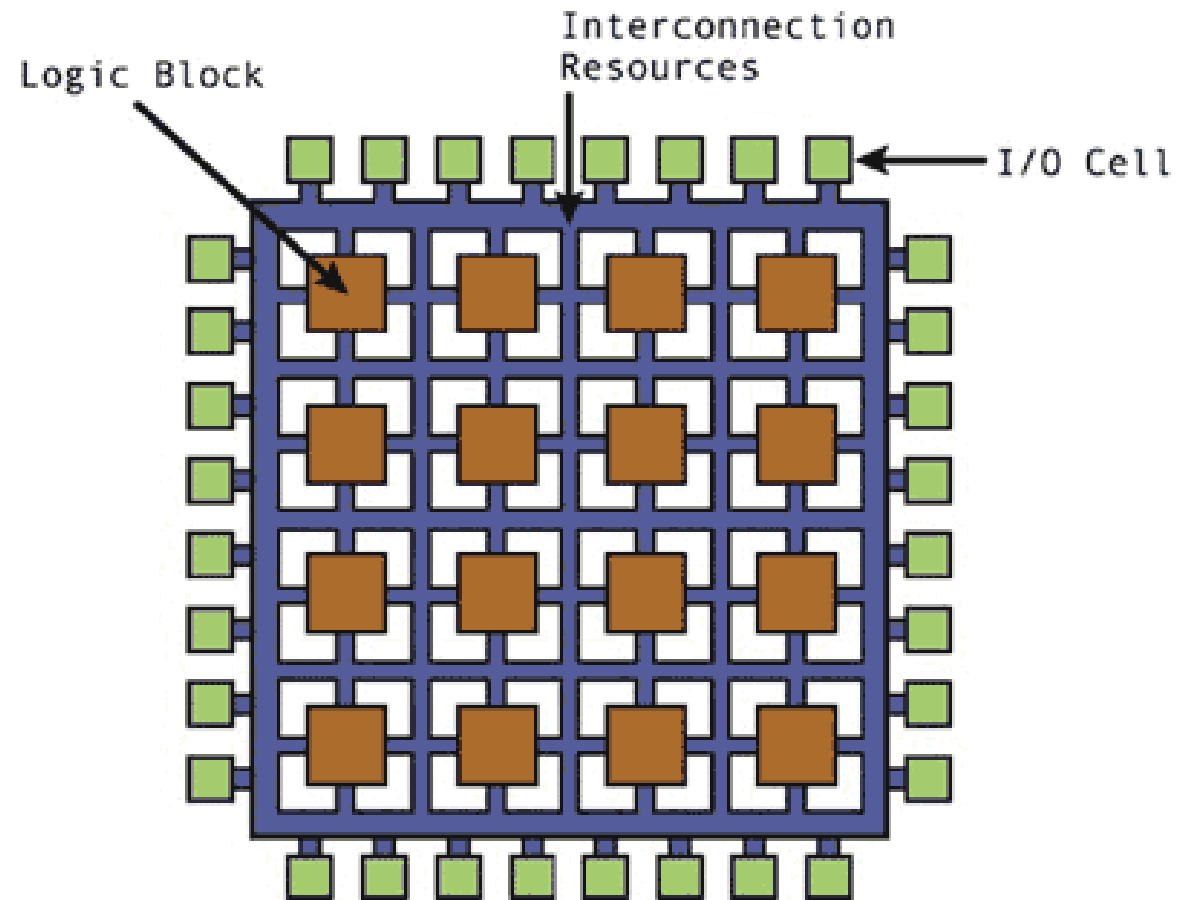
Cyclone V Adaptive Logic Modules

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

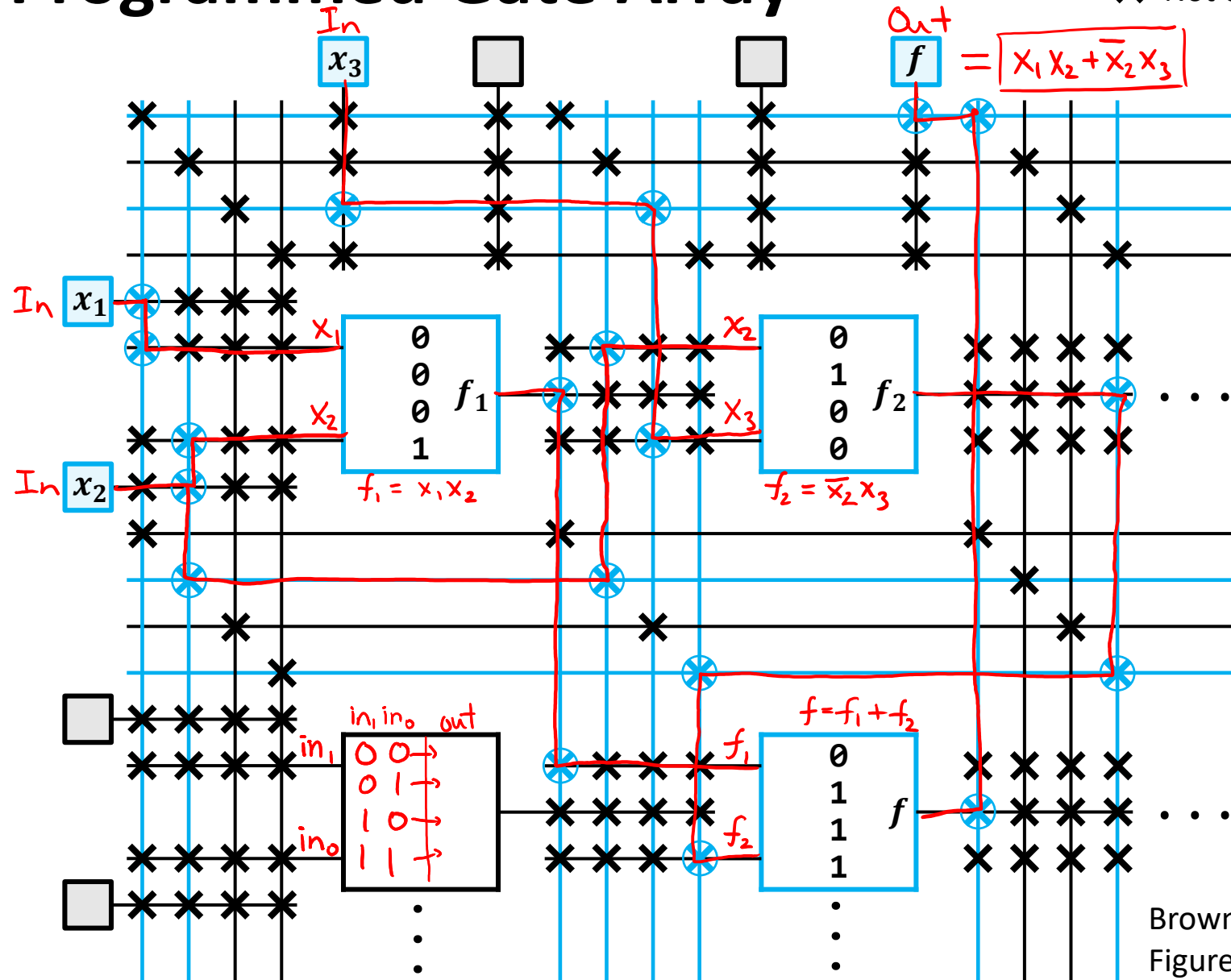


<http://www.chipdesignmag.com/print.php?articleId=434?issuelD=16>

Example Programmed Gate Array

⊗ Connected

× Not connected



Brown & Vranesic
Figure B.39

FPGA CAD

❖ CAD = “Computer-Aided Design”

```
// 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

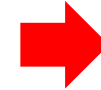
module MUX2 (V, SEL, I, J); // 2:1
multiplexer
output V;
input SEL, I, J;
wire SELB, VB;

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

Verilog



FPGA
CAD
Tools



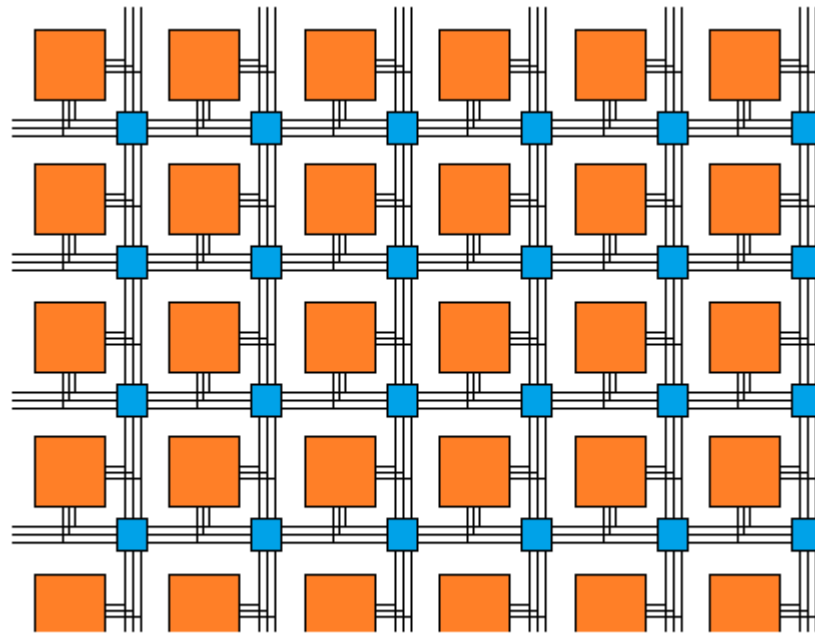
```
001010100010100101
001001001001100010
101000101011000101
010010100101010001
011000100101010101
001111001001010000
101010010101001001
000010101010100101
010010100010101101
010010100101001010
0101001
```

Bitstream

- 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

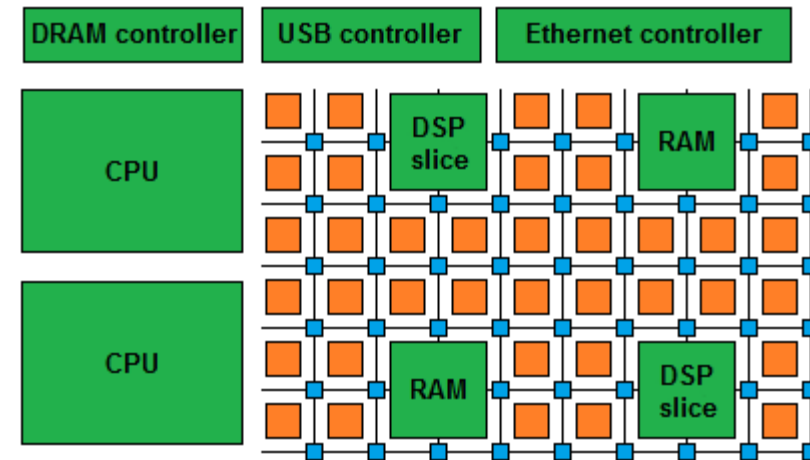
Gate Array vs. SoC

❖ SoC = "System on a Chip"



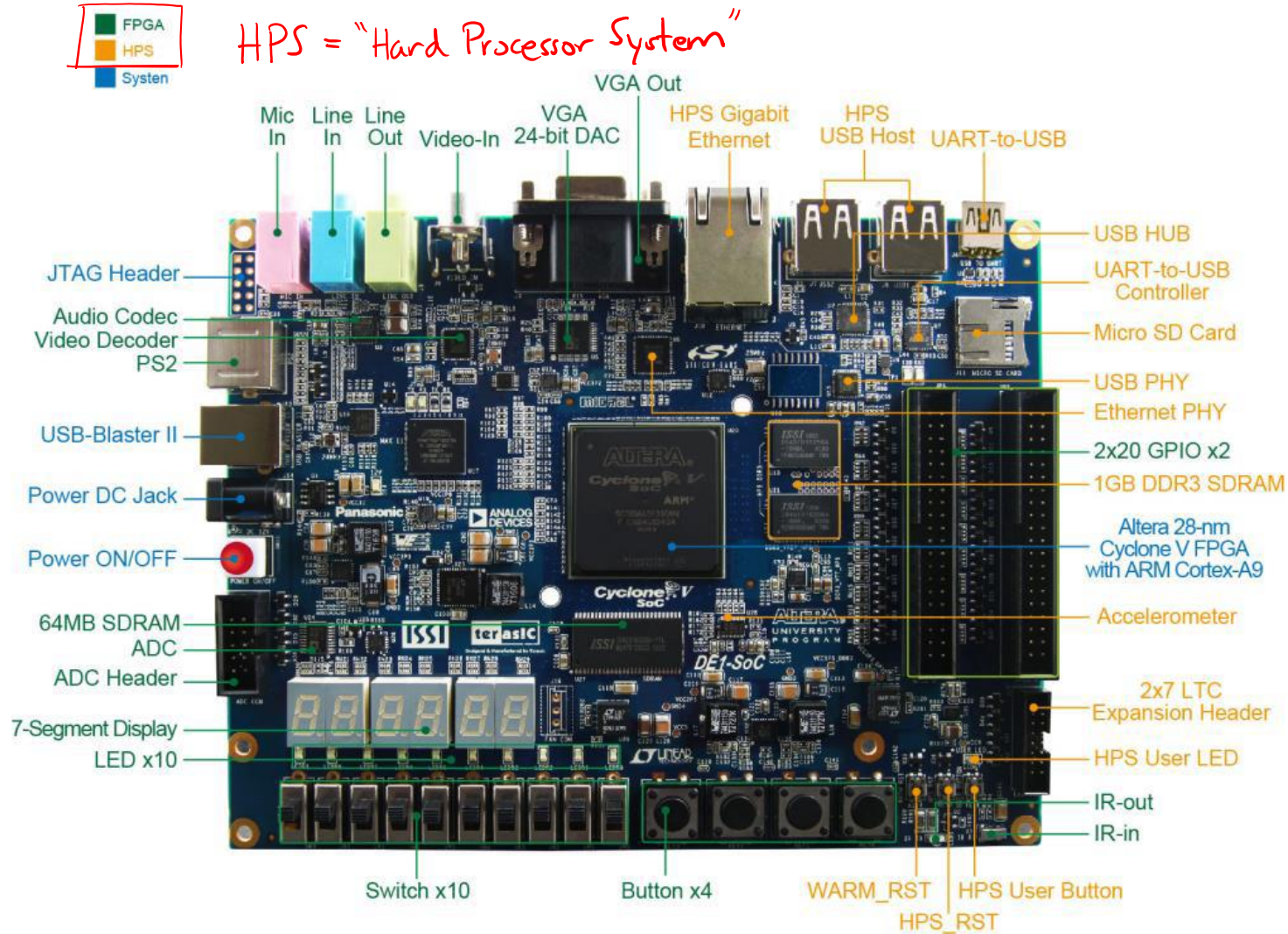
"Clean slate" FPGA: programmable gates and routers

FPGA

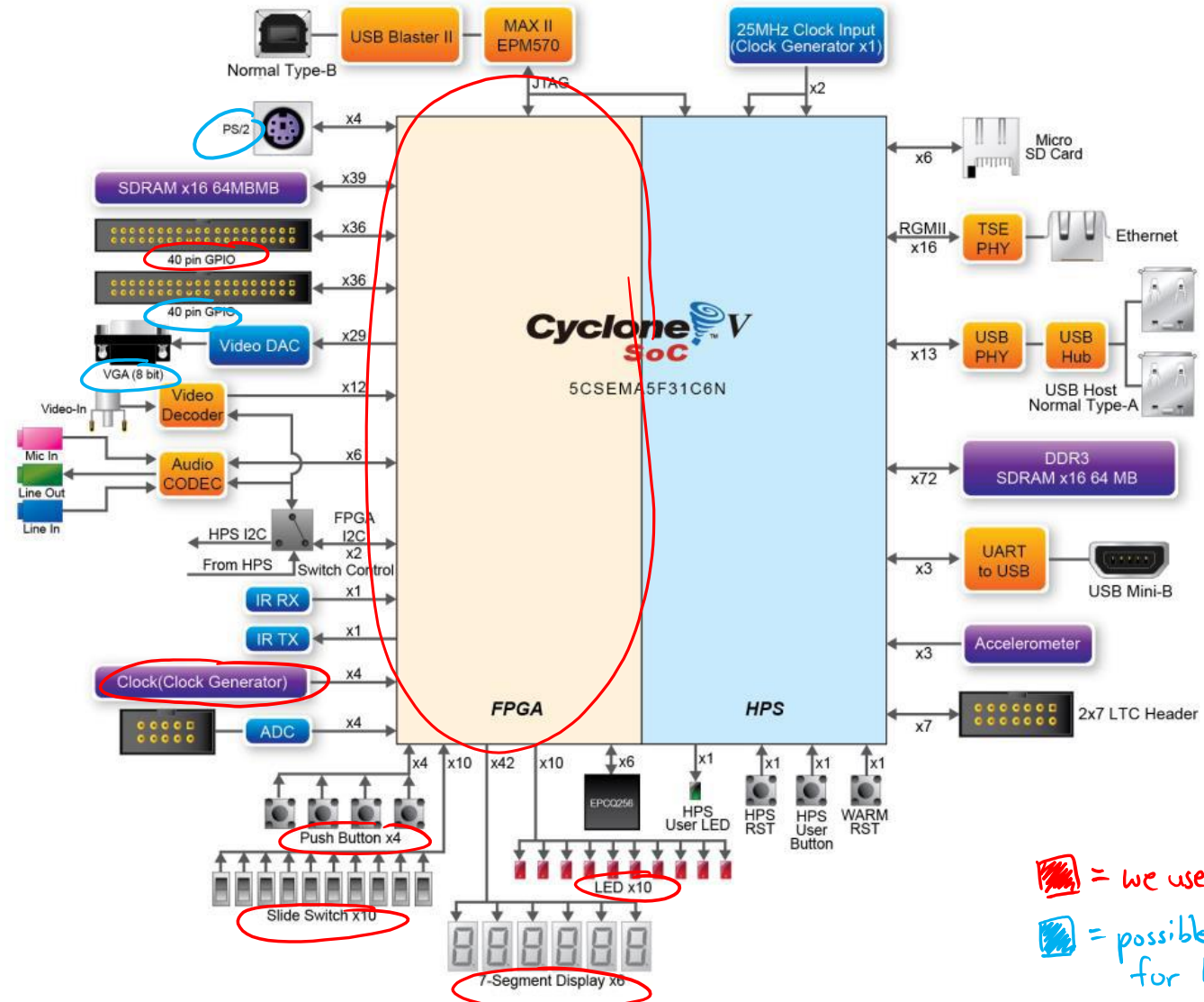




Modern FPGA: lots of hard, not-field-programmable gates

DE1-SoC Board



DE1-SoC Board



 = we used this quarter!
 = possible peripherals for Lab 8

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

Outline

- ❖ Computer Components (Cont.)
 - Register File
 - Datapath teaser
 - Serial Communication
- ❖ FPGAs
- ❖ **Course Wrap-up**

Course Wrap-Up

- ❖ Combinational Logic
 - Karnaugh Maps
 - ❖ Sequential Logic
 - Timing Considerations
 - ❖ Finite State Machines
 - ❖ Routing Elements
 - Multiplexor, Encoder, Decoder, Demultiplexor
 - ❖ Computational Building Blocks
 - Adders, Registers, Shift Registers, Counters
 - ❖ *SystemVerilog* ← Labs
- Quiz 1
- Quiz 2
- Quiz 3

Digital Workhorses

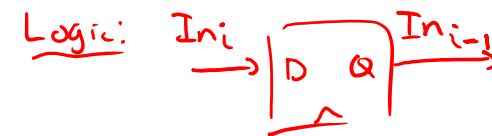
❖ Multiplexors

- Logic – can implement arbitrary logic as LUTs
- Routing – select between many simultaneous computations

❖ Flip-flops

- Store information
 - registers, shift registers, counters, FSMs, RAM cells
- Synchronization of system
 - Delay element – “holds up” information
- Deal with metastability

registers/state → FSMs
→ counters
→ LFSRs (shift registers)



Takeaways

- ❖ You can do a lot without actually digging into the hardware details!
 - Verilog is programmatic way to generate design hardware
 - High-level view is sufficient for system design in many cases
- ❖ ... but the hardware details are important
 - Timing especially – synchronization, metastability
 - Bits are always present – resets, unexpected situations
 - Design affects speed, power, size
- ❖ A CPU is not required for many applications

Takeaways

- ❖ You now know how to design and implement fairly complex digital designs!
 - Could breadboard + wire packages
 - DE1-SoC (\$322): <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836>

Final Words

- ❖ Please fill out course evaluation!
 - Released online next week online; let's improve this class
- ❖ Future classes *Signal Conditioning*
 - EE 205/215 – details of electronics
 - EE/CSE 371 – digital design techniques and considerations (algorithms, communication, complexity)
 - EE/CSE 469 – computer architecture (CPU design)
- ❖ Hope you had fun this quarter!
- ❖ Huge thanks to your TAs!

