

Intro to Digital Design

L9: Advanced Verilog, CPUs, FPGAs

Instructor: Naomi Alterman

Teaching Assistants:

Derek de Leuw

Isabel Froelich

Kevin Hernandez

Aarjav Jain

Packard Stephenson

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)!
- Your finished circuit due no later than Friday, June 5 @ 11:59 pm
- Return your lab kit to TAs at your final demo

Administrivia

- ❖ **Quiz 3** is next week: Tuesday, June 2
 - 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

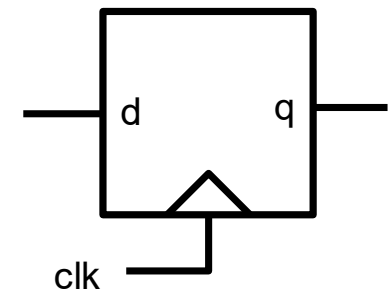
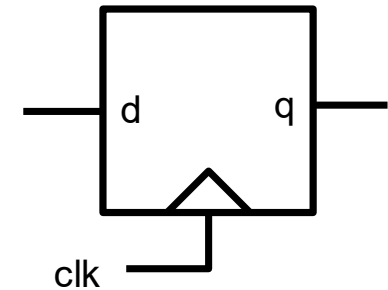
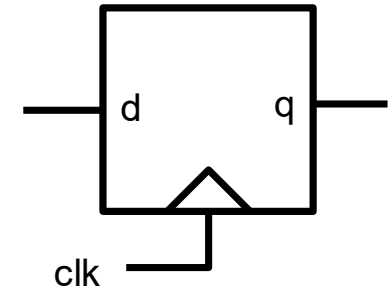
Practice

Implement a **counter** that goes through the state sequence:

000 → 001 → 011 → 010 → 110 →
 111 → 101 → 100 → 000 → ...

Include Enable and Reset signals

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			



Practice

Implement a **counter** that goes through the state sequence:

000 → 001 → 011 → 010 → 110 →
111 → 101 → 100 → 000 → ...

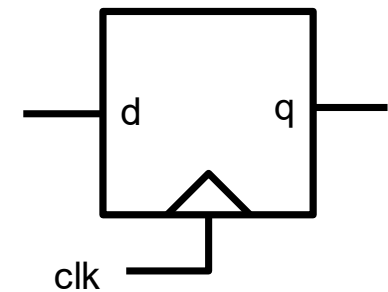
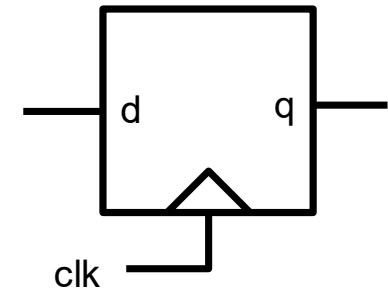
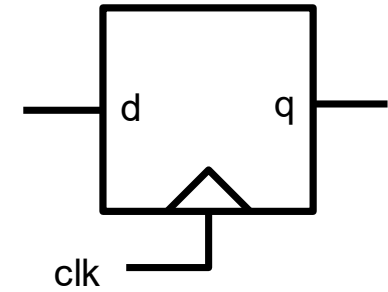
Include Enable and Reset signals

P ₂	P ₁	P ₀	N ₂	N ₁	N ₀
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

$$N_2 = P_2 P_0 + P_1 \overline{P_0}$$

$$N_1 = \overline{P_2} P_0 + P_1 \overline{P_0}$$

$$N_0 = \overline{P_2} \overline{P_1} + P_2 P_1$$



Outline

- ❖ **Advanced Verilog Topics**
 - **Verilog generate**
 - **SystemVerilog Arrays**
- ❖ History: TTL
- ❖ CPU teaser
- ❖ FPGAs

Add/Sub in Verilog (parameterized)

❖ Variable-width add/sub (with overflow, carry)

```
module addN #(parameter N=32) (OF, CF, S, sub, A, B);
  output logic      OF, CF;
  output logic [N-1:0] S;
  input  logic      sub;
  input  logic [N-1:0] A, B;
  logic  [N-1:0] D;    // possibly flipped B
  logic          C2;   // second-to-last carry-out

  always_comb begin
    D = B ^ {N{sub}}; // replication operator
    {C2, S[N-2:0]} = A[N-2:0] + D[N-2:0] + sub;
    {CF, S[N-1]} = A[N-1] + D[N-1] + C2;
    OF = CF ^ C2;
  end
endmodule // addN
```

Advanced Verilog: generate

- ❖ Condense your code using loops and conditionals
 - Often used with assign and module instantiation

```
genvar <loop_var>;  
  
generate  
  for (<init>; <cond>; <update>) begin : <label>  
    // do something with loop_var  
  end  
endgenerate
```

- ❖ Details:
 - Loop variables must be declared as genvar outside of generate statement
 - Block statements (for/if) *must* have begin and end and be labeled

Using generate to chain full adders together

Reminder: `module fulladd (cout, s, cin, a, b);`

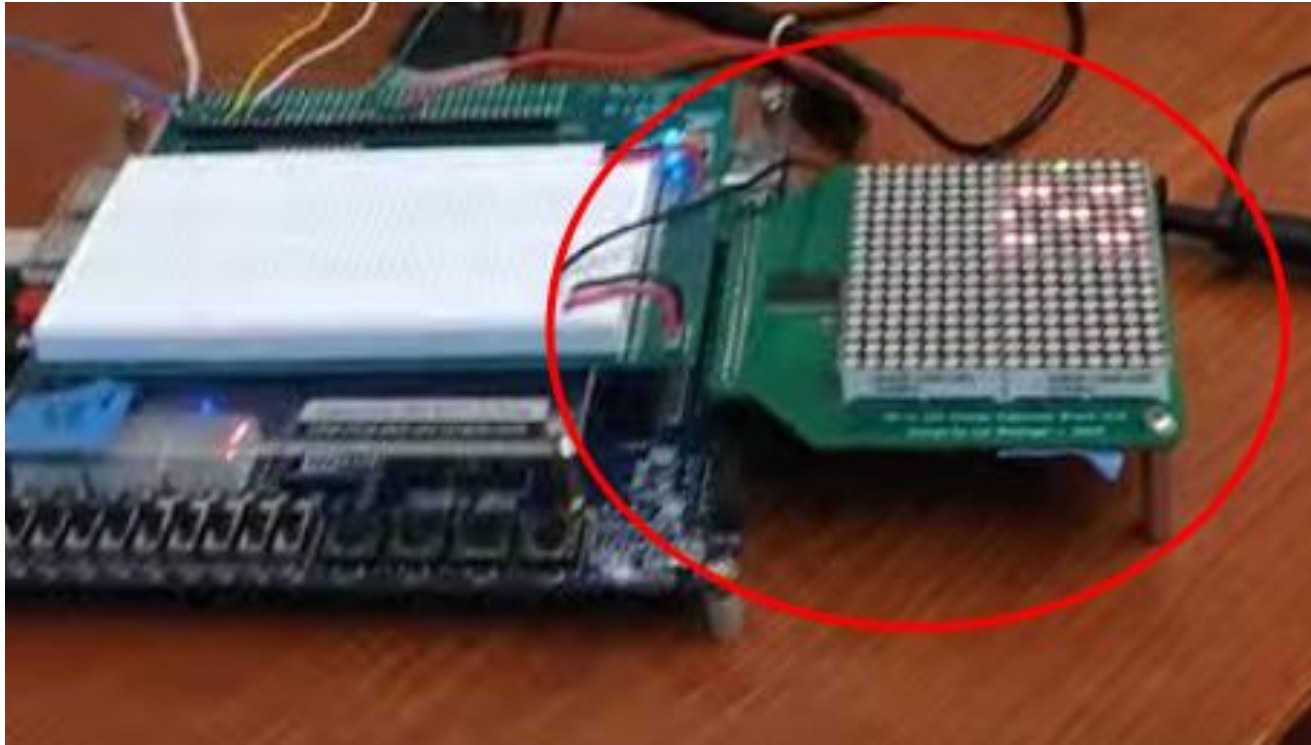
```
module addNgen #(parameter N=32) (OF, CF, S, sub, A, B);
  output logic OF, CF;           // overflow and carry flags
  output logic [N-1:0] S;       // sum output bus
  input logic sub;              // subtract signal
  input logic [N-1:0] A, B;     // input busses
  logic [N:0] C;                // carry signals between modules

  genvar i;
  generate
    for (i=0; i<N; i=i+1) begin : adders
      fulladd fadd (C[i+1], S[i], C[i], A[i], sub^B[i]);
    end
  endgenerate

  assign C[0] = sub;
  assign CF = C[N];
  assign OF = C[N] ^ C[N-1];
endmodule // addNgen
```

SystemVerilog Arrays

- ❖ We can (and do!) have **multidimensional arrays** :
 - `logic [15:0][15:0] RedPixels;`
 - When accessed, index from left to right



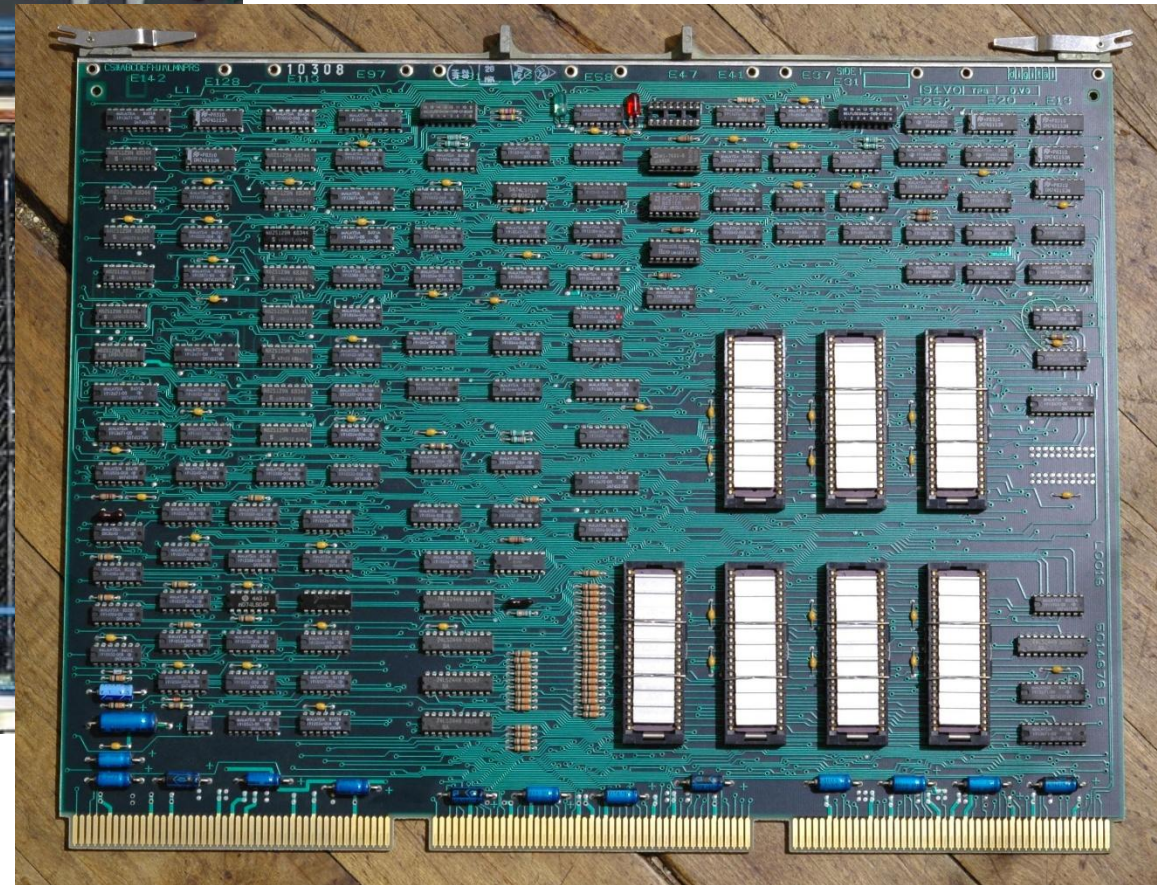
Packed and Unpacked Arrays

- ❖ A *bus* is also known as a *vector* or **packed array**
 - e.g., `logic [31:0] 32_bit_number;`
 - Can only be made of single bit datatypes
- ❖ “Regular” array syntax is known as an **unpacked array**
 - e.g., `logic four_bits_for_four_bells[4:0];`
 - Can be made of any datatype
- ❖ They can be **combined** when using multidimensional arrays:
 - Accessed left to right, **starting with unpacked dimensions**
 - e.g., `logic [31:0] five_32_bit_numbers[4:0];`
 - `five_32_bit_numbers[2][15:0]` // The 15 LSBs of the 3rd word

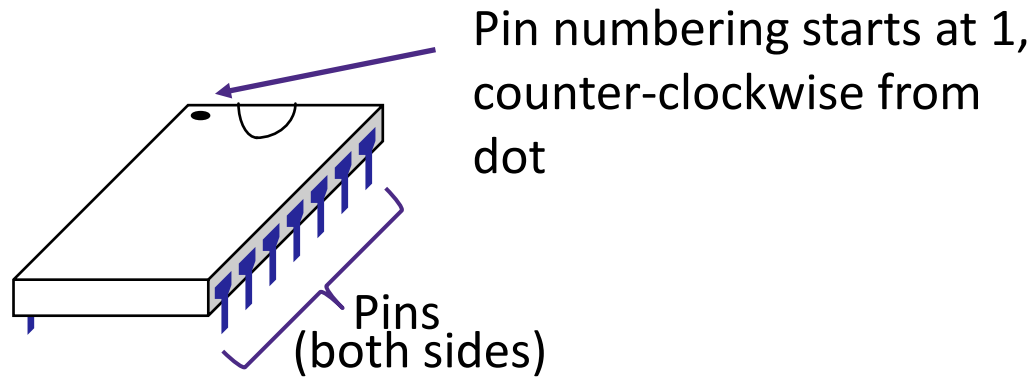
Outline

- ❖ Advanced Verilog Topics
- ❖ **History: TTL**
- ❖ CPU teaser
- ❖ FPGAs

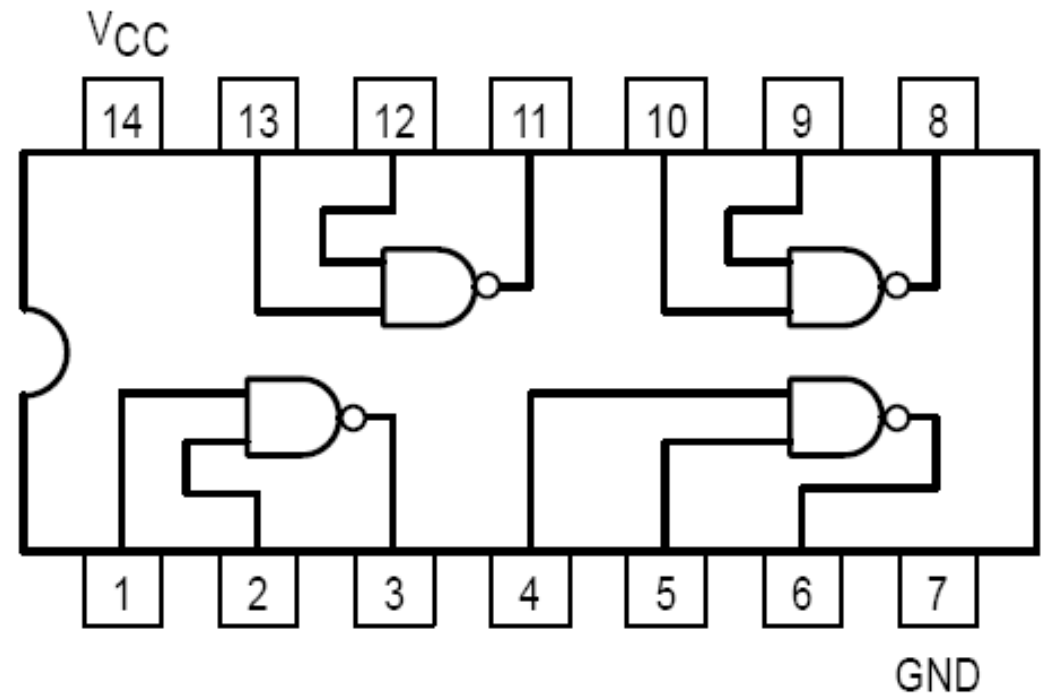
TTL computers



Transistor-Transistor Logic (TTL) Packages

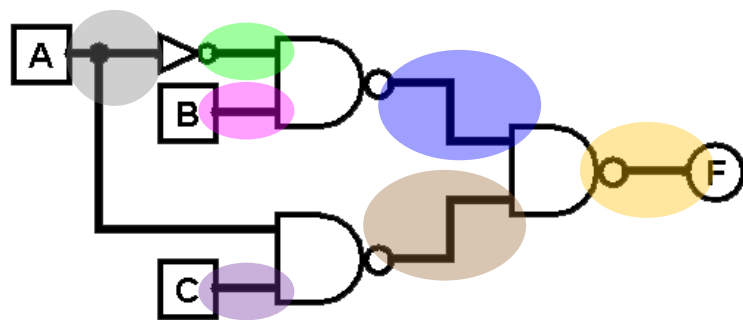


- ❖ Diagrams like these and other useful/helpful information can be found on part **data sheets**
 - It's really useful to learn how to read these



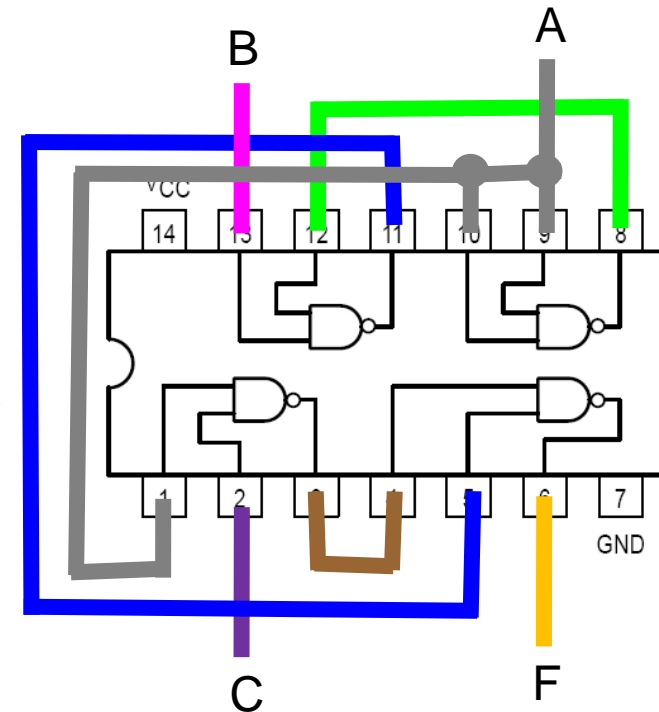
Mapping truth tables to circuit boards

- ❖ Given a truth table:
 - 1) Write the Boolean expression
 - 2) Minimize the Boolean expression
 - 3) Draw as gates
 - 4) Map to available gates
 - 5) Determine # of packages and their connections

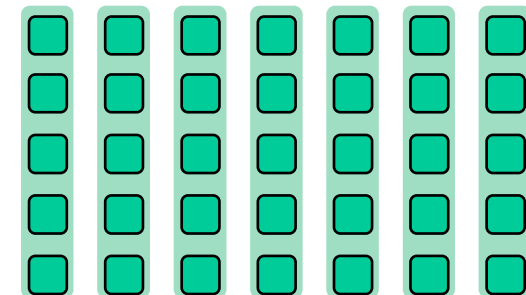
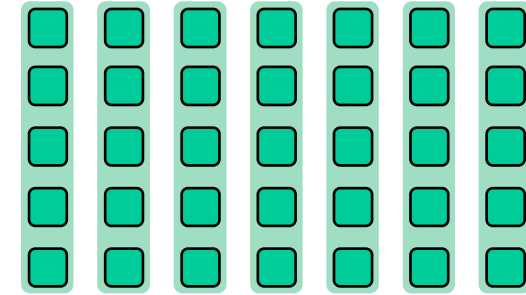
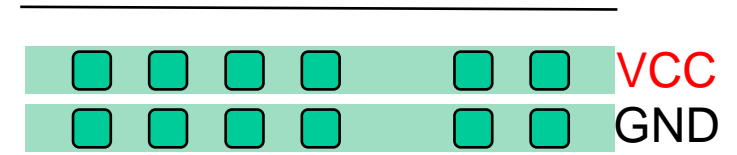
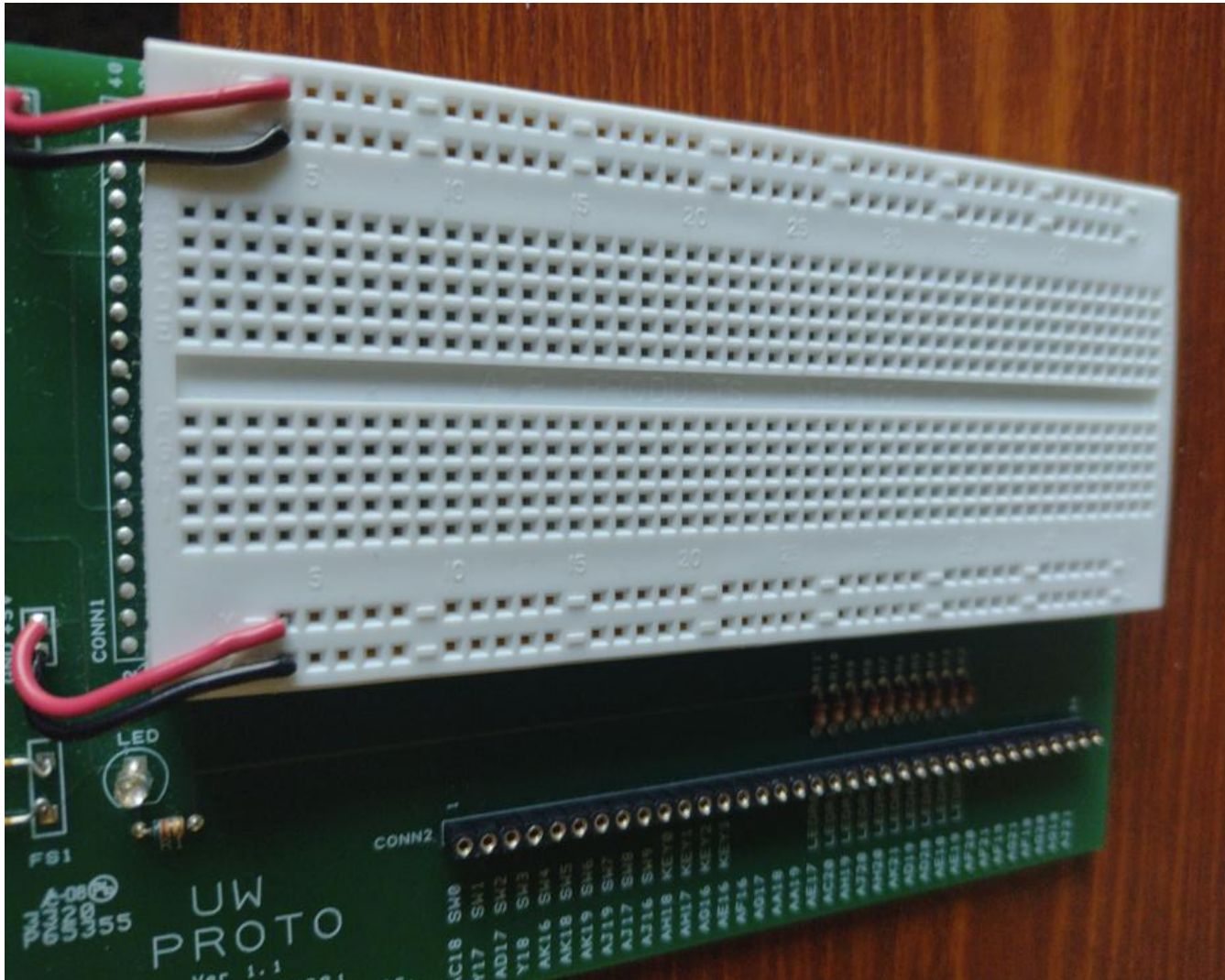


7 nets (wires) in this design

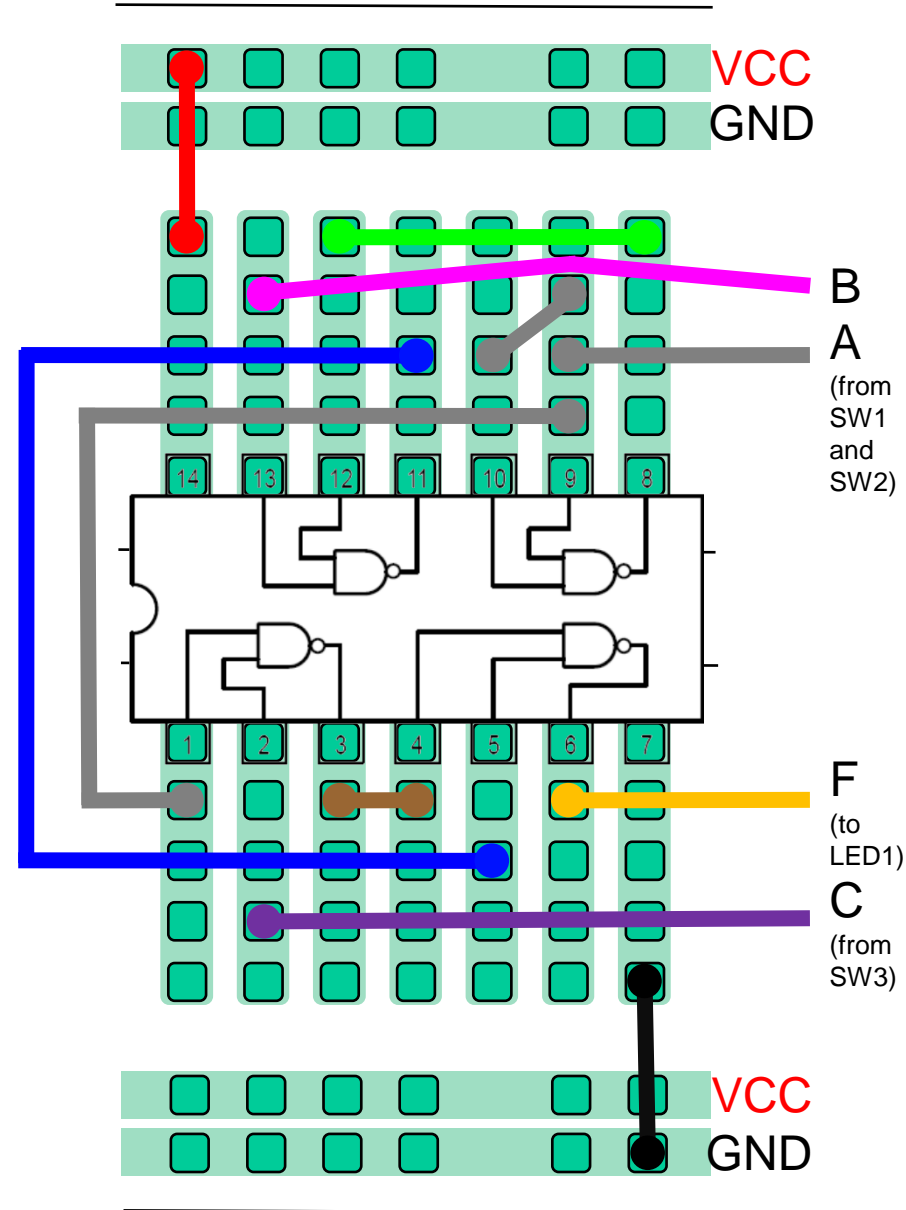
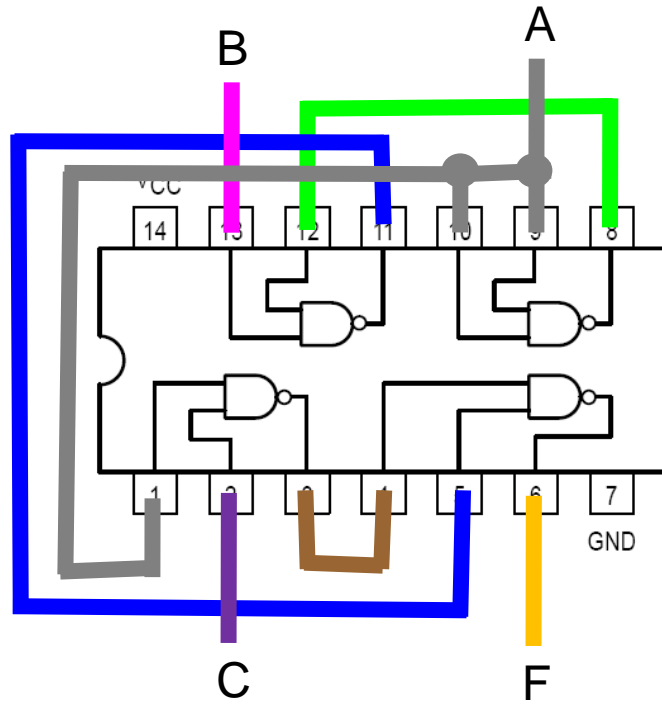
(4)



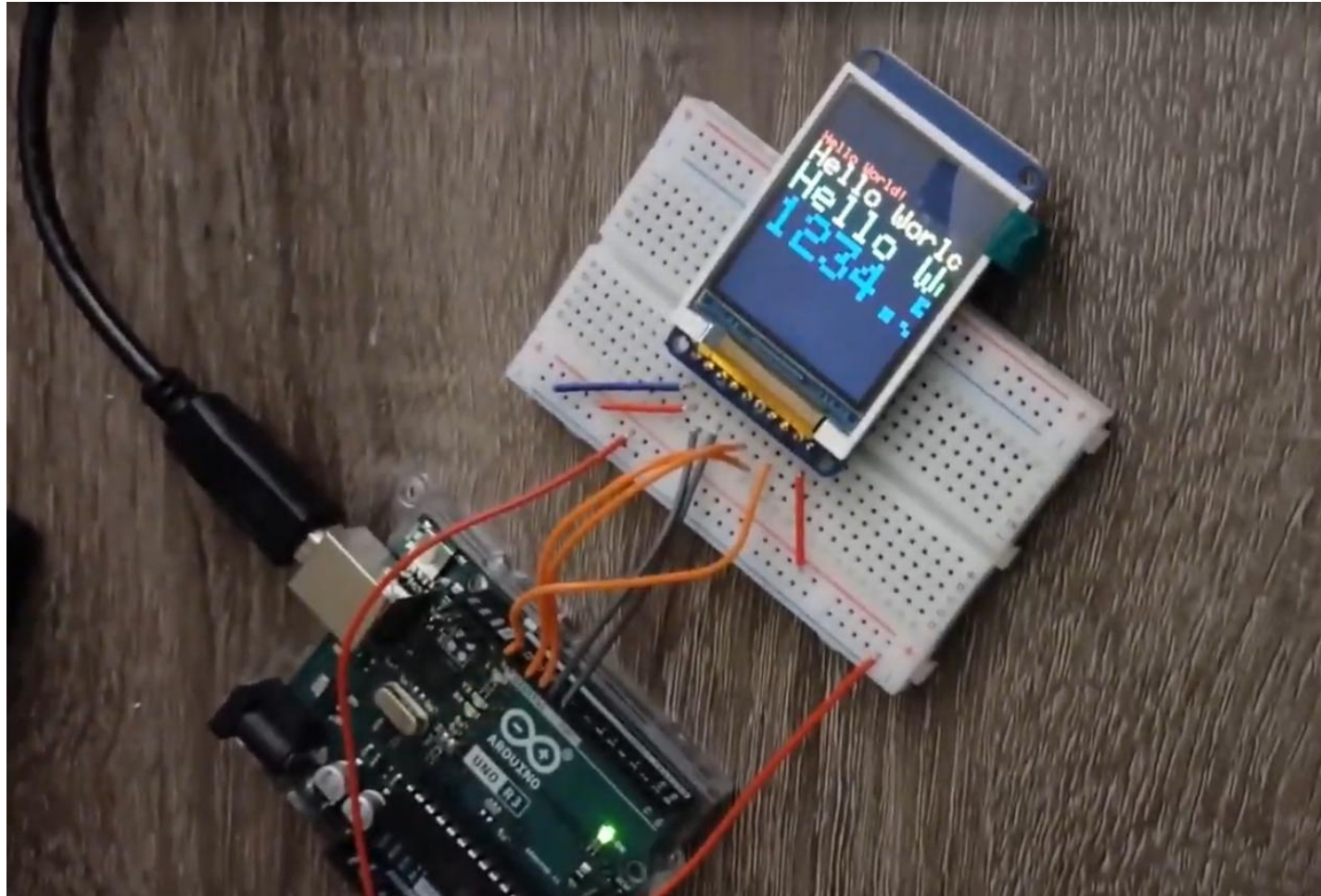
Prototyping with breadboards



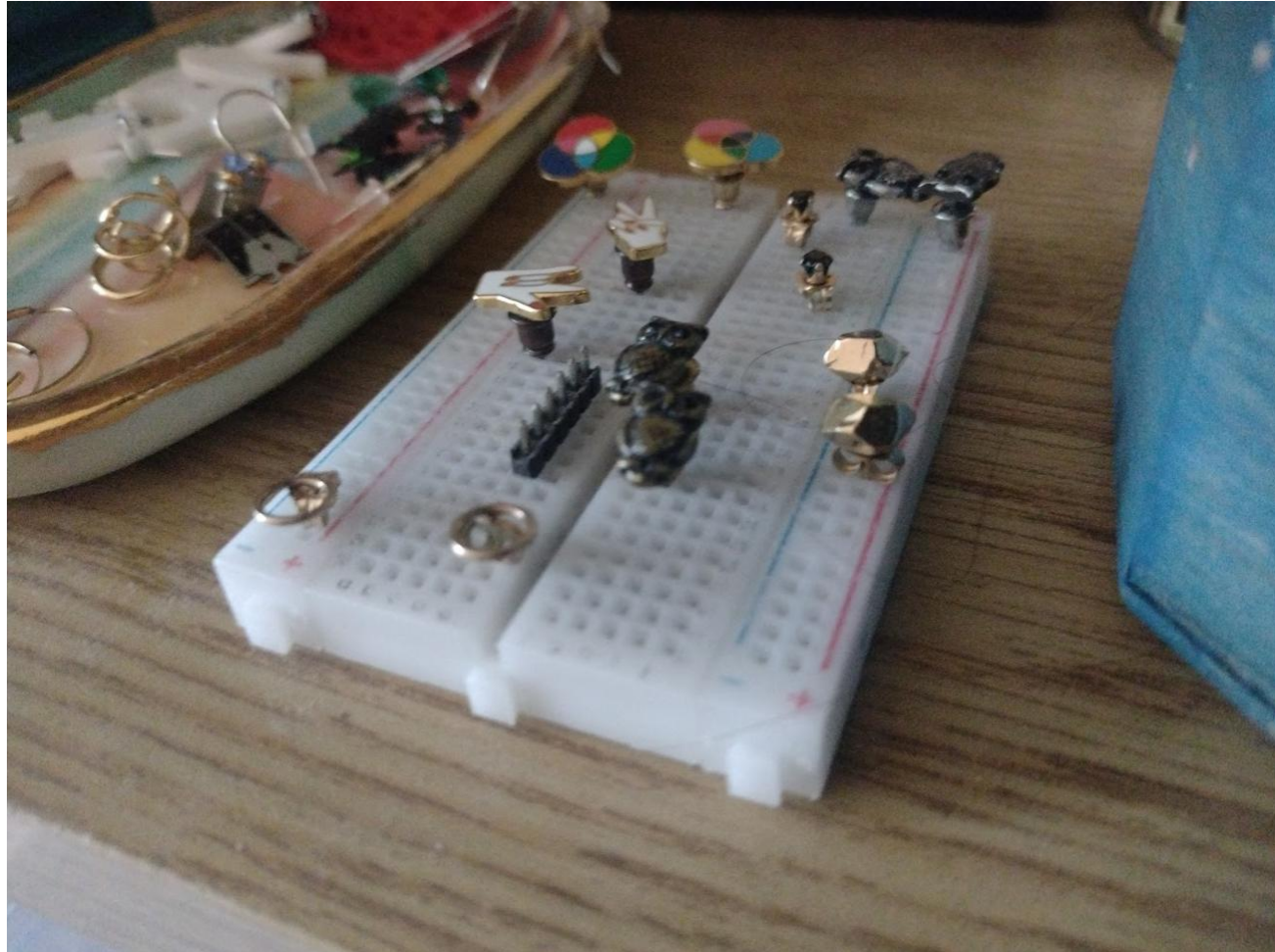
Breadboarding circuits



Breadboarding Modern Components



Breadboarding Modern Fashion



Miso Moment



Outline

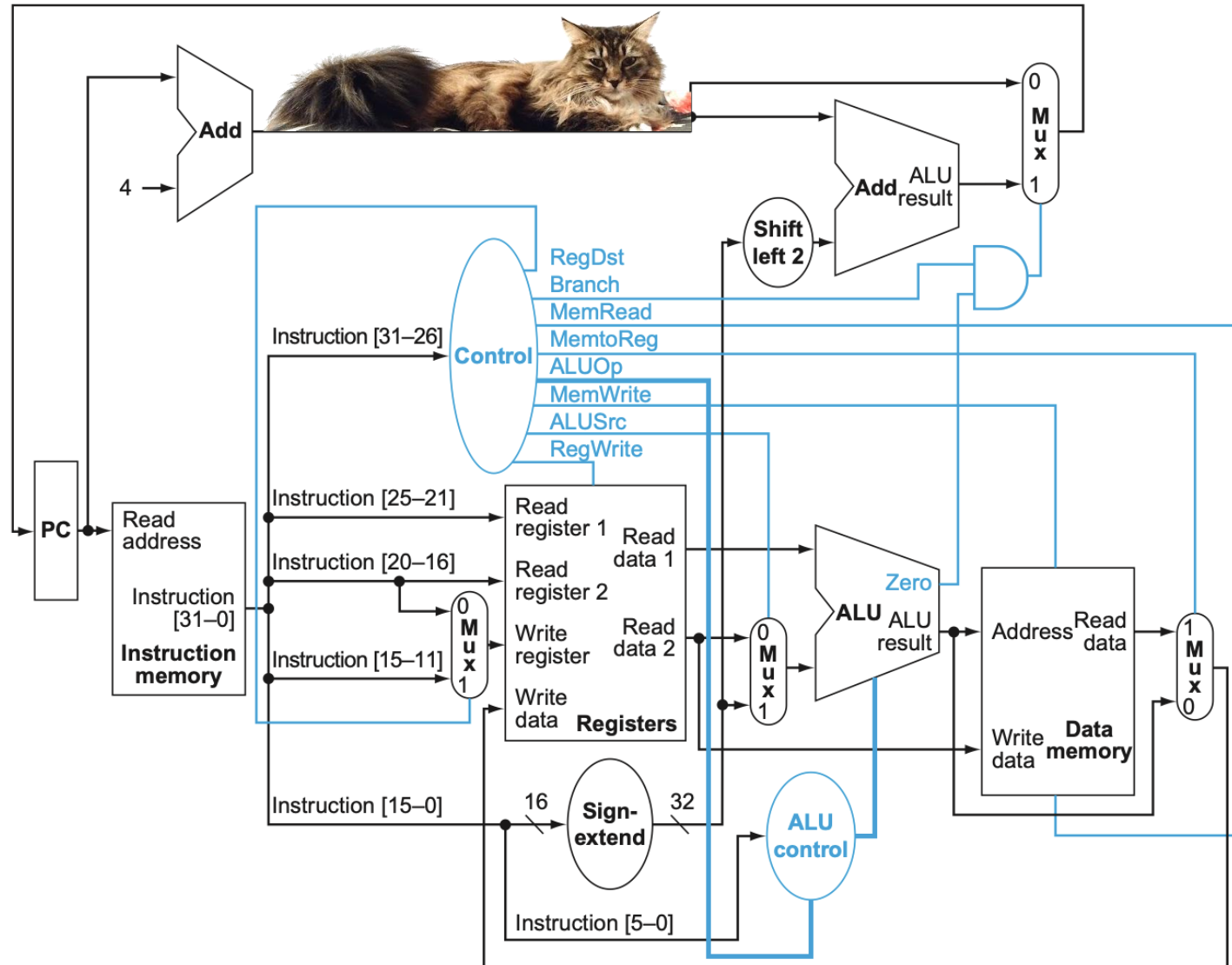
- ❖ Advanced Verilog Topics
- ❖ History: TTL
- ❖ **CPU teaser**
- ❖ FPGAs

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”)

Block-diagramming a CPU



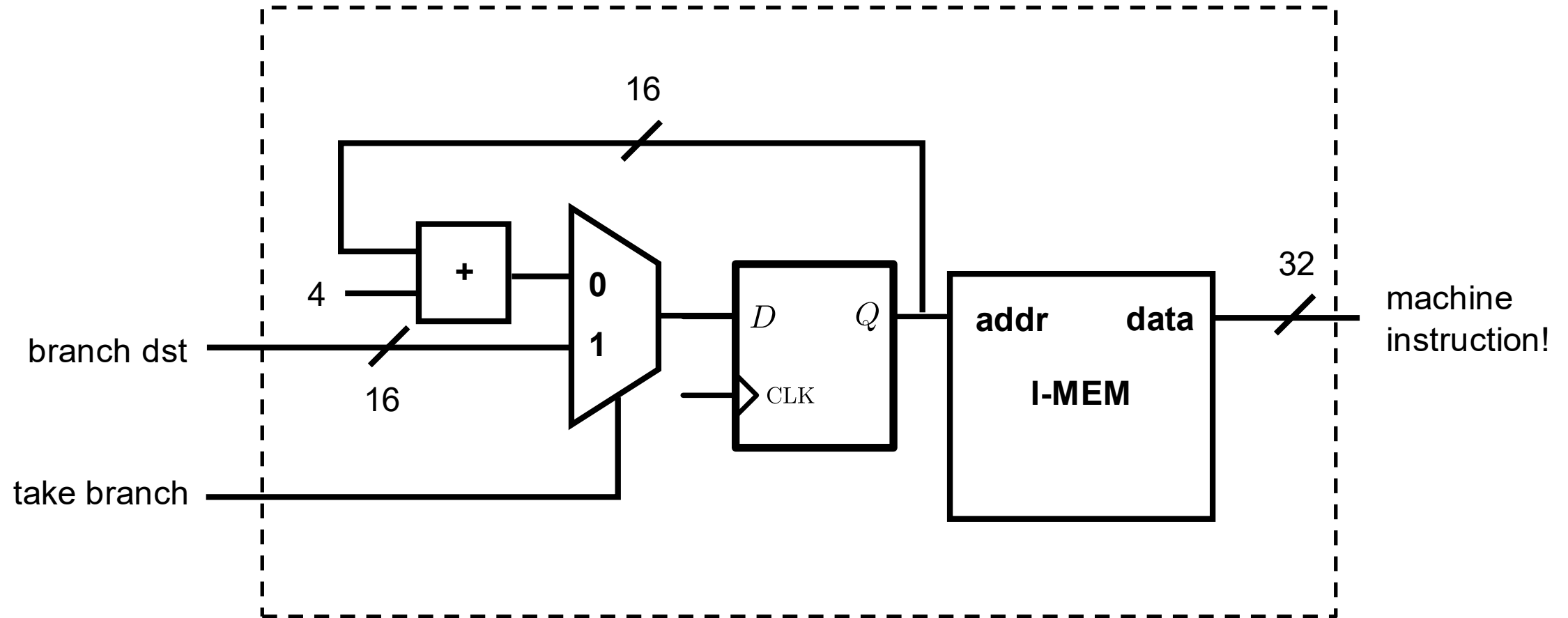
*From Patterson & Hennessy
4th ed.*

Executing an Instruction

- ❖ Broadly speaking (varies by CPU), usually looks like this:
 - Instruction Fetch
 - Instruction Decode
 - Data Fetch
 - Computation
 - Store Result

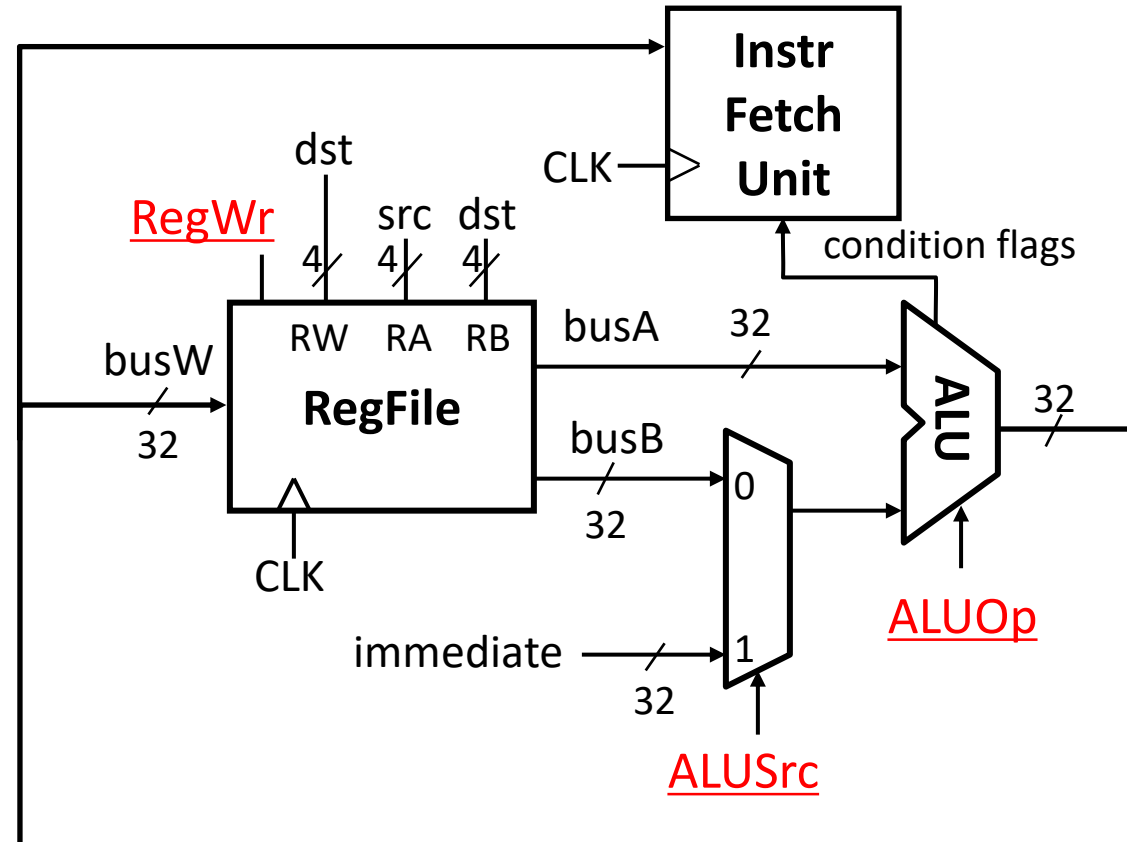
- ❖ Basic Datapath Components (idealized)
 - Register File
 - Memory Management Unit
 - Arithmetic Logic Unit (ALU)
 - Routing Elements
 - } ROMs and RAMs
 - } Muxes, adders, encoders, etc

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



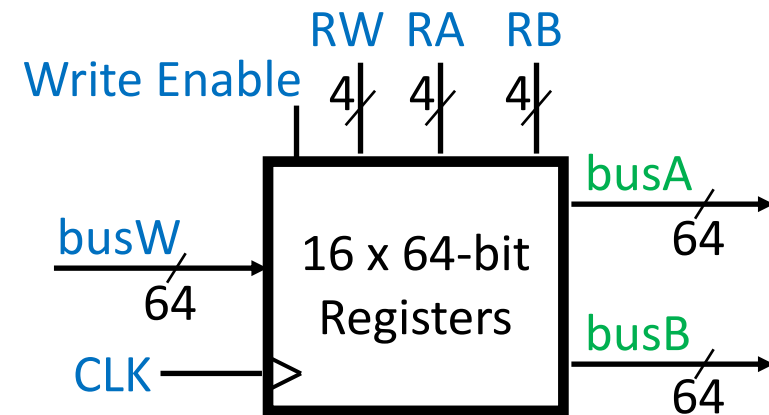
Datapath Teaser

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

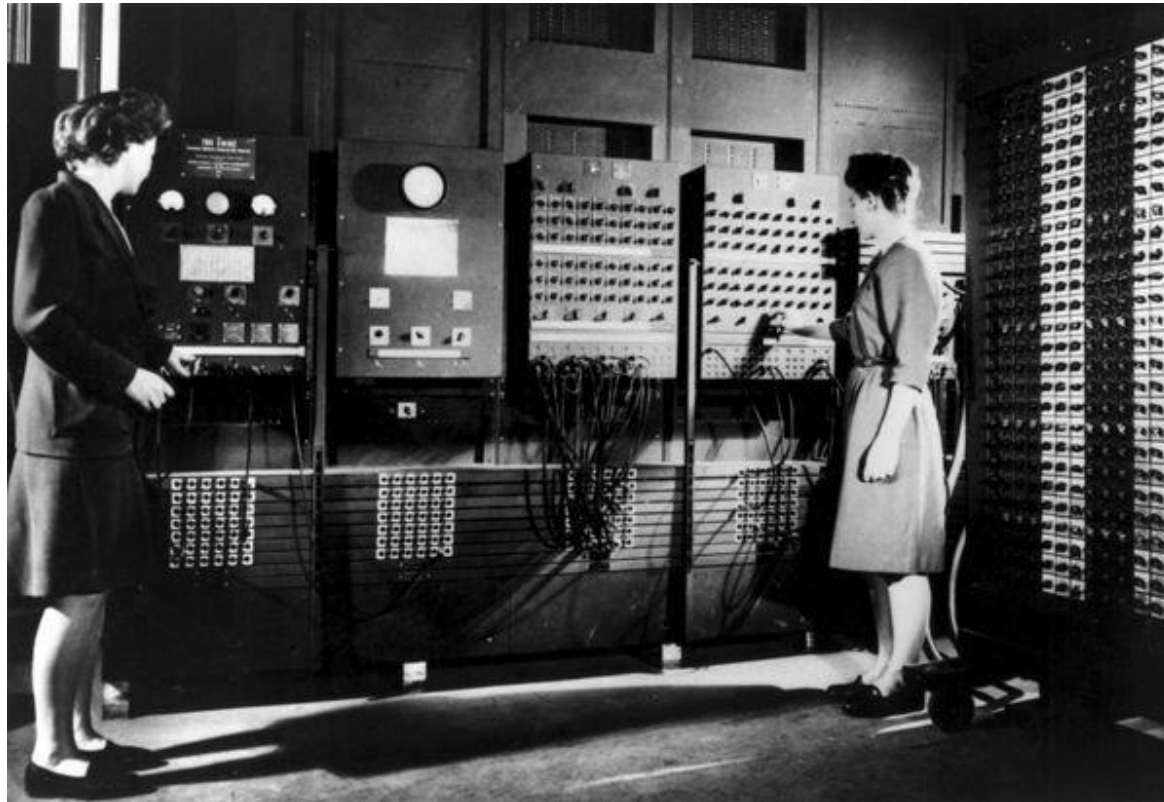


Zoom in on the Register File

- ❖ 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

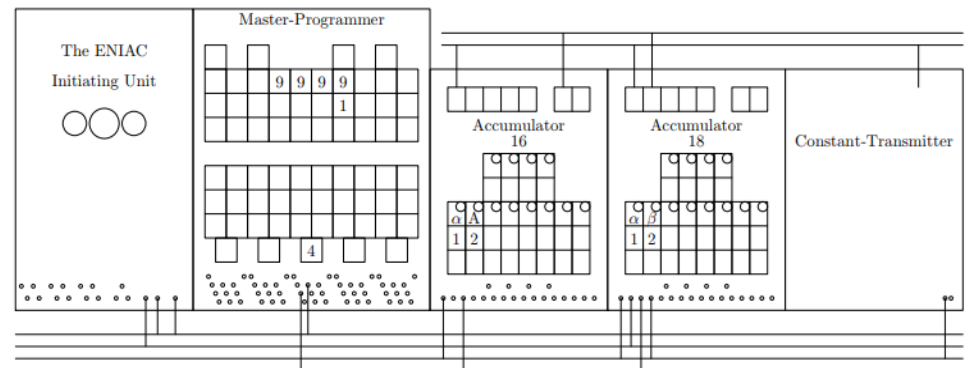
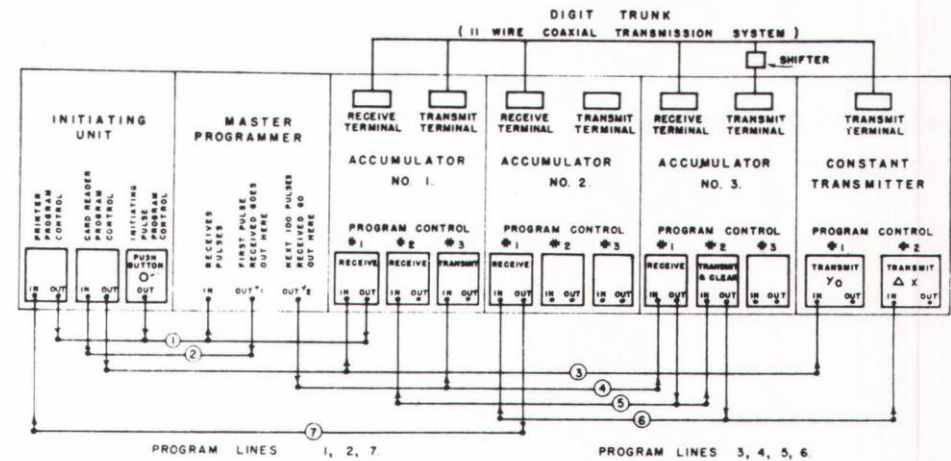


Early Language



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

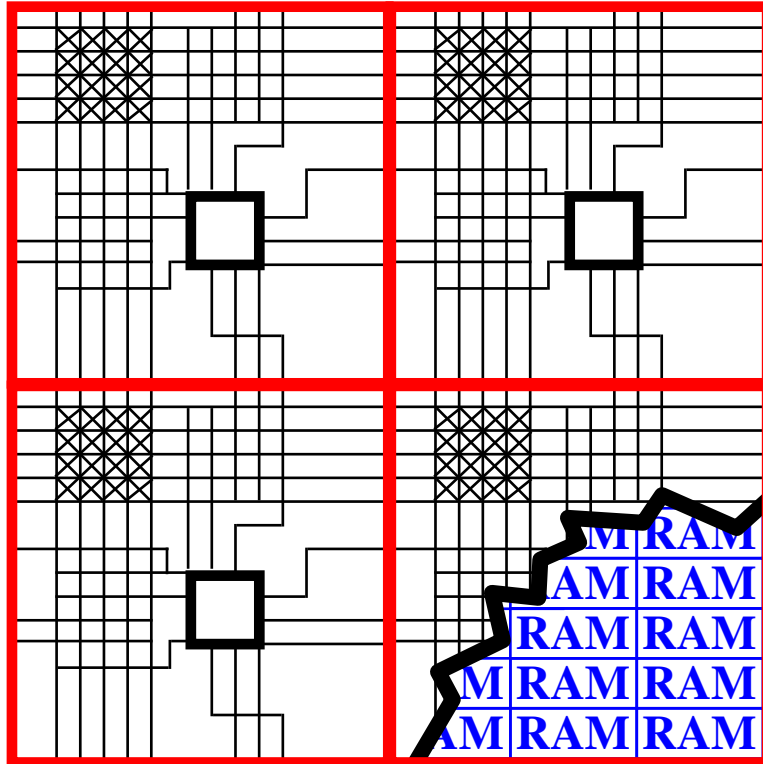
<https://www.cs.drexel.edu/~bls96/eniac/eniac5.pdf>


<http://ds-wordpress.haverford.edu/bitbybit/bit-by-bit-contents/chapter-seven/7-5-assembly-language-programming-2/>

Outline

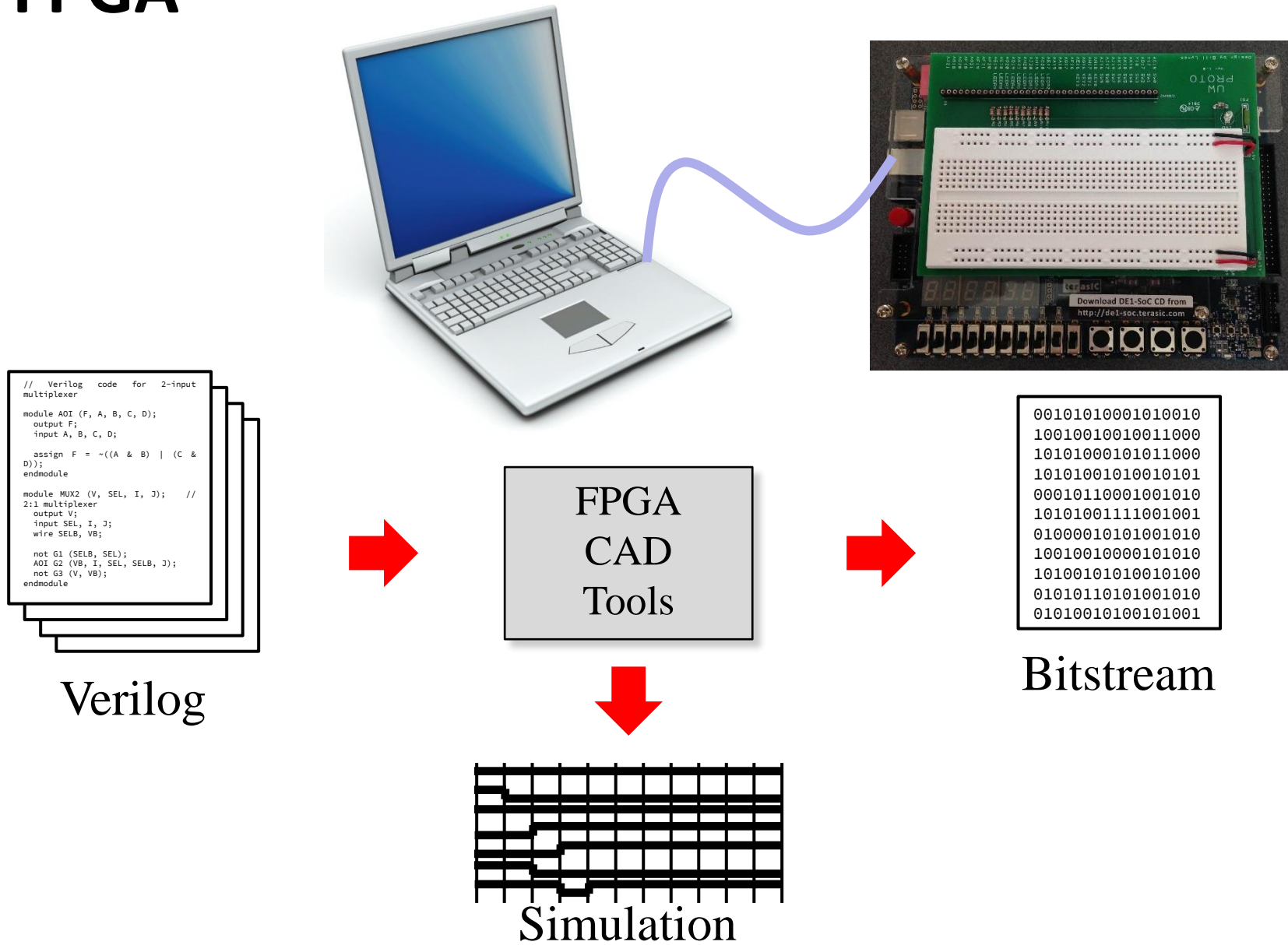
- ❖ Advanced Verilog Topics
- ❖ History: TTL
- ❖ CPU teaser
- ❖ **FPGAs**

Field Programmable Gate Arrays (FPGAs)



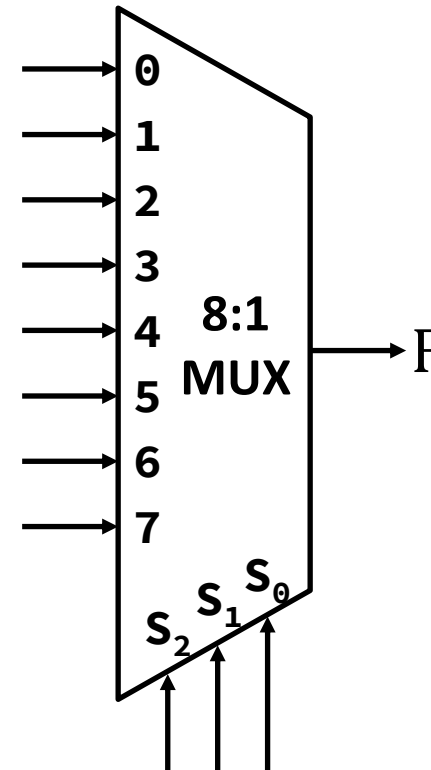
- ❖ Logic cells () embedded in a general routing structure
- ❖ Logic cells usually contain:
 - 6-input Boolean function calculator
 - Flip-flop (1-bit memory)
- ❖ All features are electronically (re)programmable

Using an FPGA



FPGA Combinational Logic

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

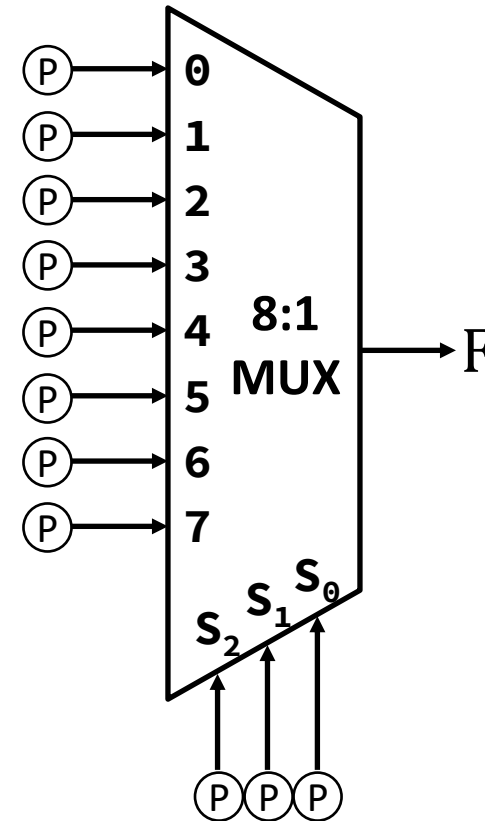


FPGA Programming

```

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

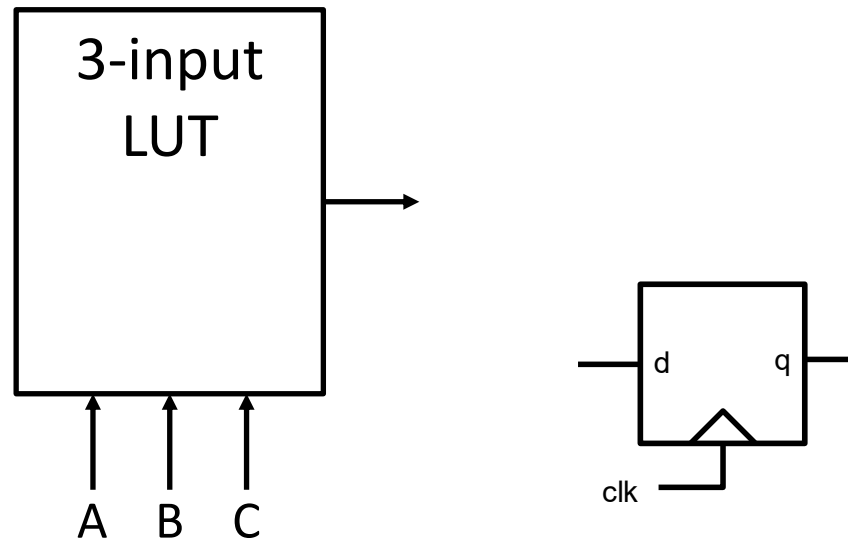
Bitstream



Ⓟ = 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?

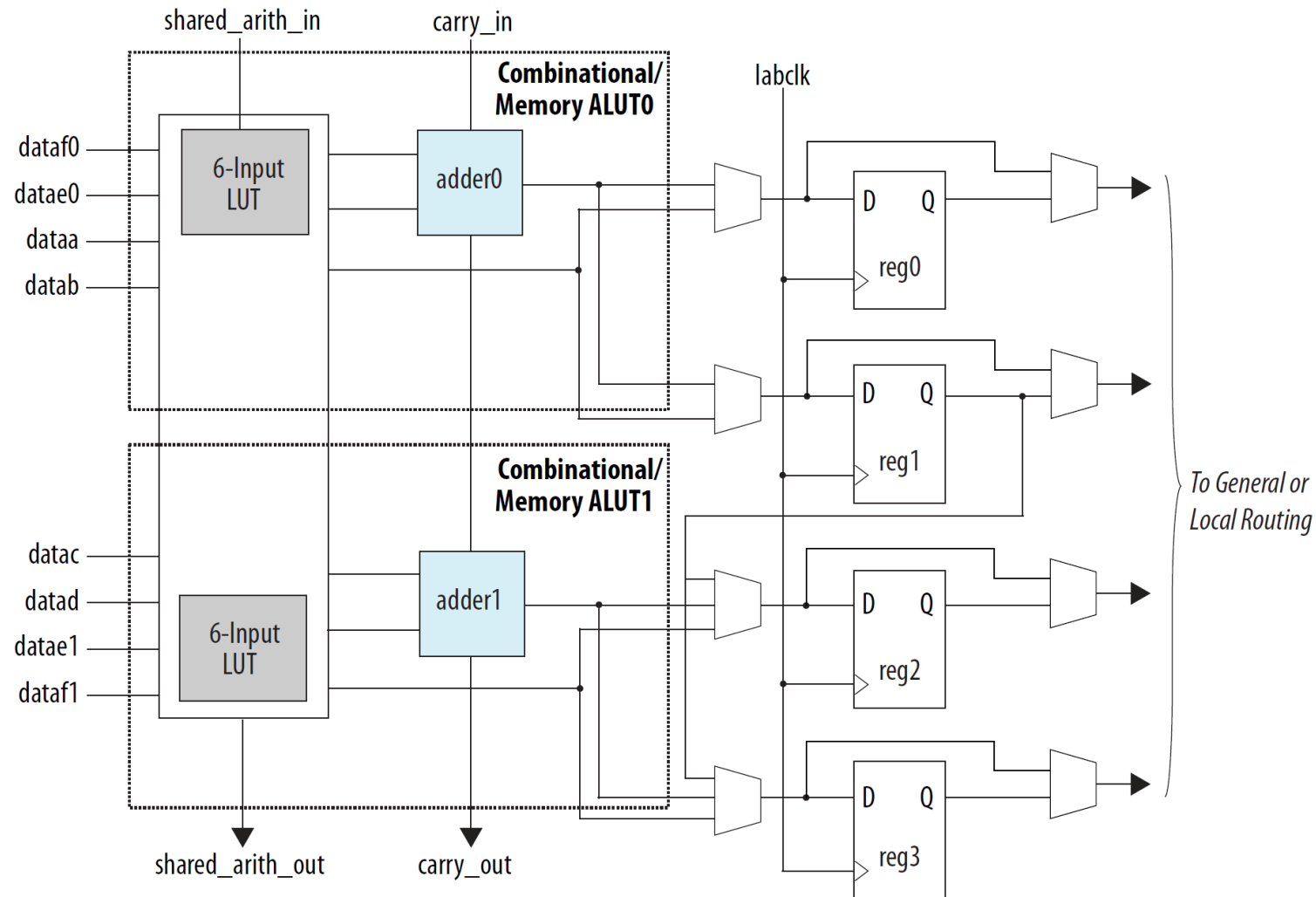


- ❖ Creates a Logic Cell (or Logic Element)

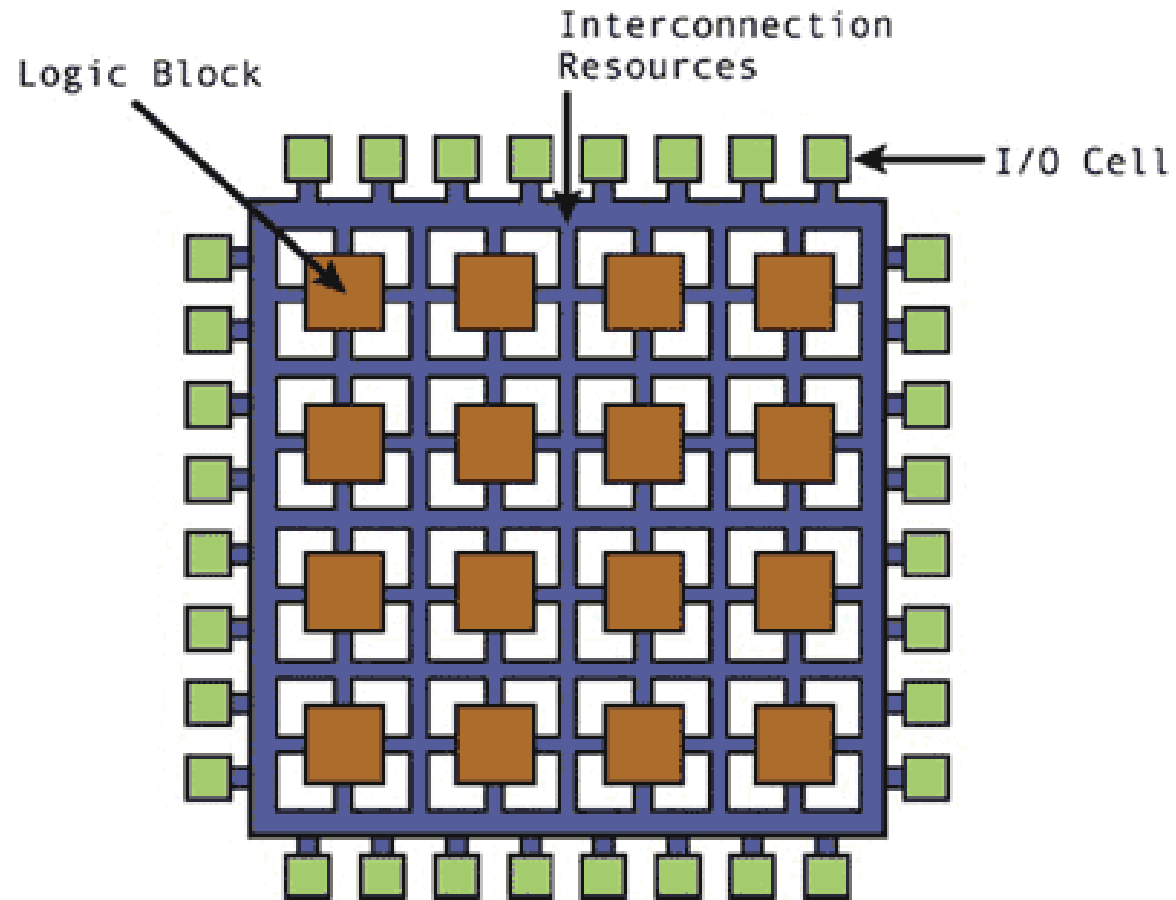
Cyclone V Adaptive Logic Modules

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



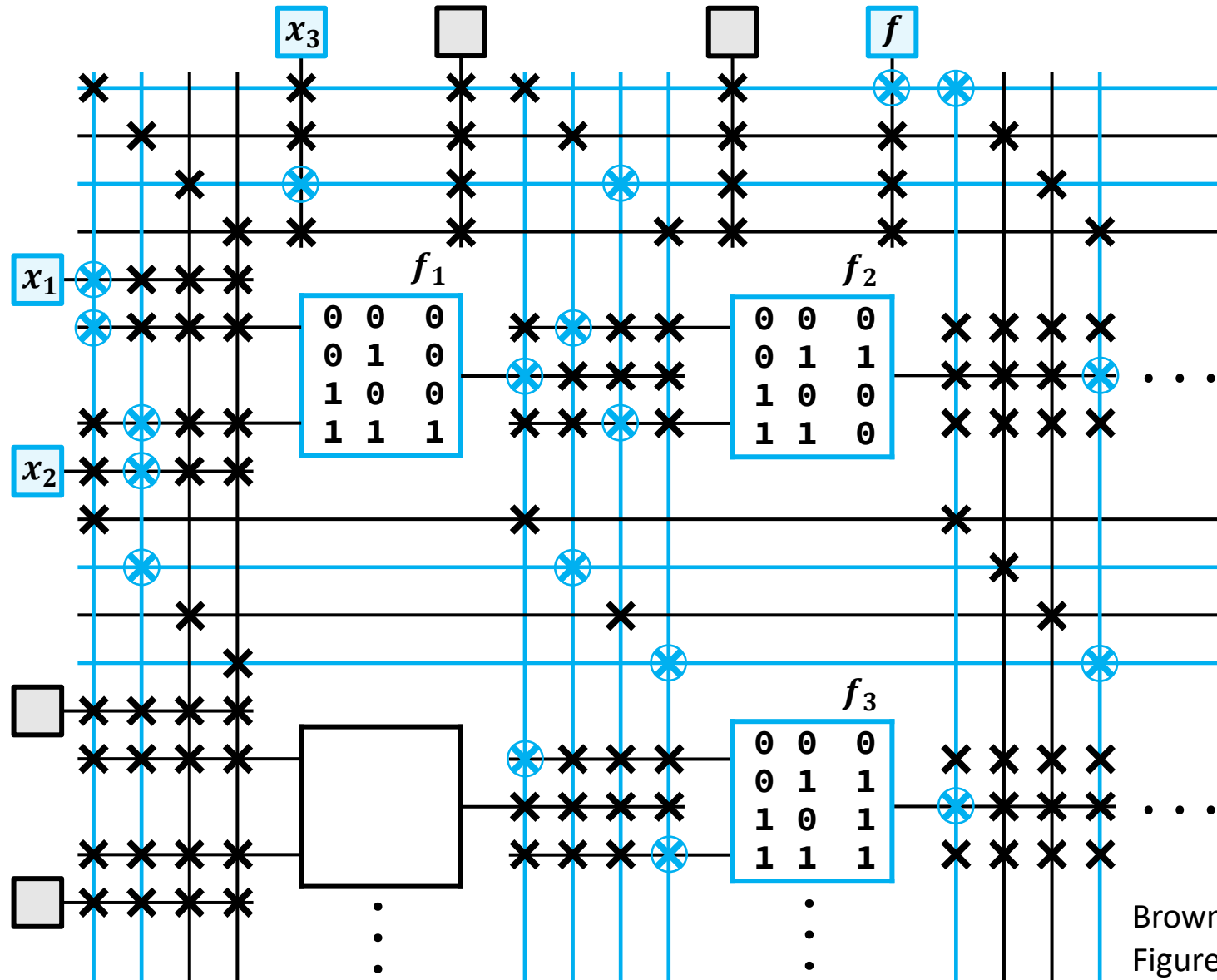
Generic FPGA Logic Layout



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



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

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