ENGR 210 / CSCI B441
"Digital Design"

Memory

Course Evals plz!

Andrew Lukefahr

Announcements

- P9 – SPI
  - This one is new. Might be some changes.
  - Last one

- Final: **~~Thursday~~ 5/6 @ 12.40-2:40pm**

Friday

FixME!

P9 SPI QuickStart

- We build the Vivado project for you:

```
git clone https://github.com/ENGR210/P9_SPI.git
cd P9_SPI
make setup
vivado vivado/vivado.xpr
```

- We provide you with Testbenches

- Same ones as the Autograder!

ROM vs RAM

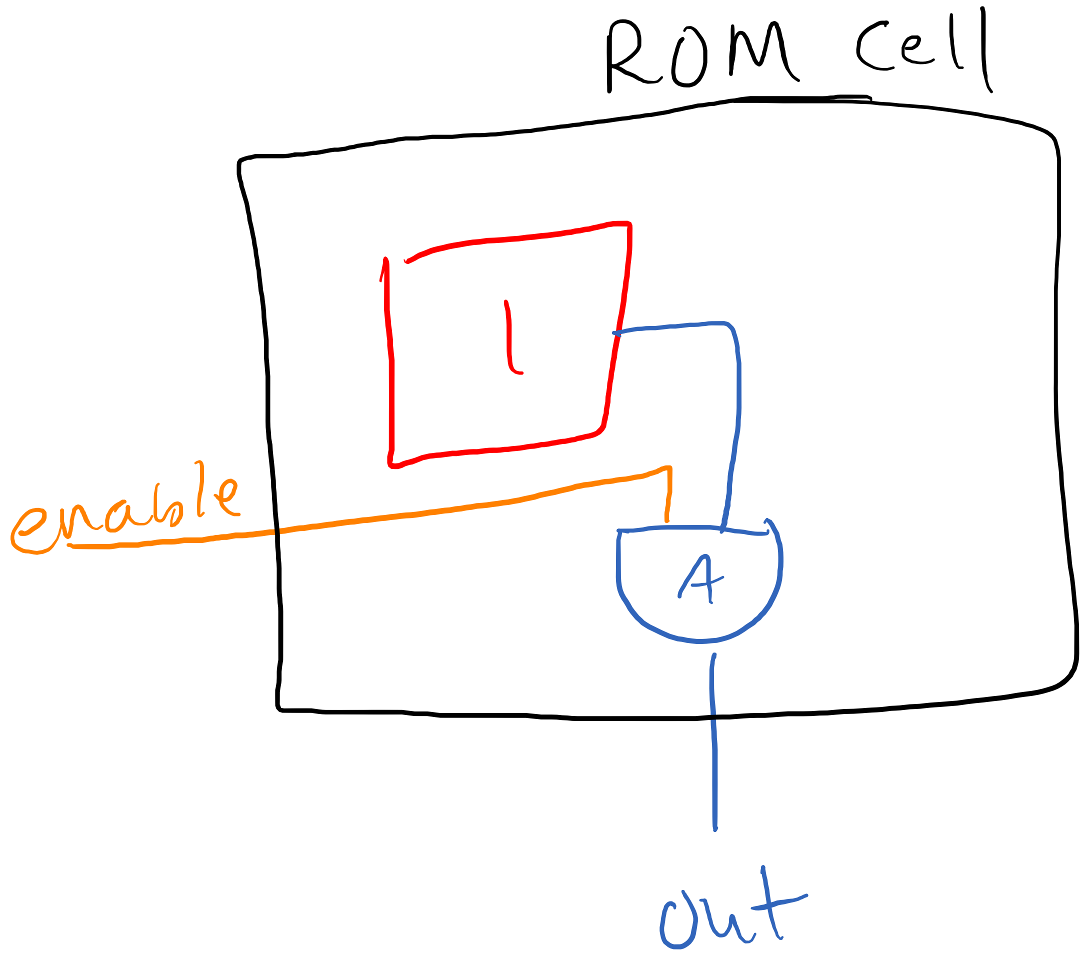
- ROM – Read-Only Memory
  - Input: address
  - Output: fixed value

- RAM – Random-Access Memory
  - Read/Write version of a ROM

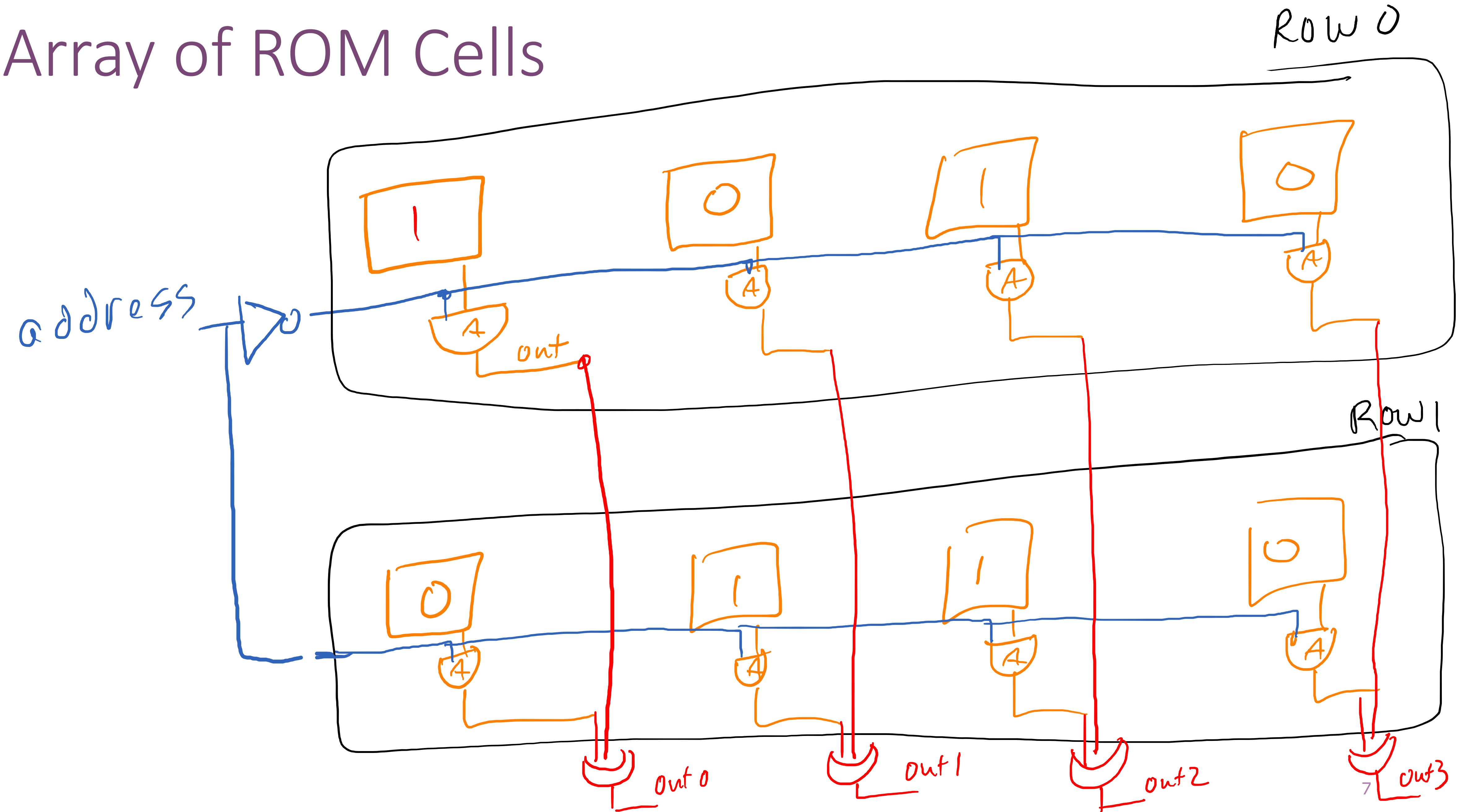
Rom Cell

ROM: Read Only Memory

RAM: Random-Access Memory

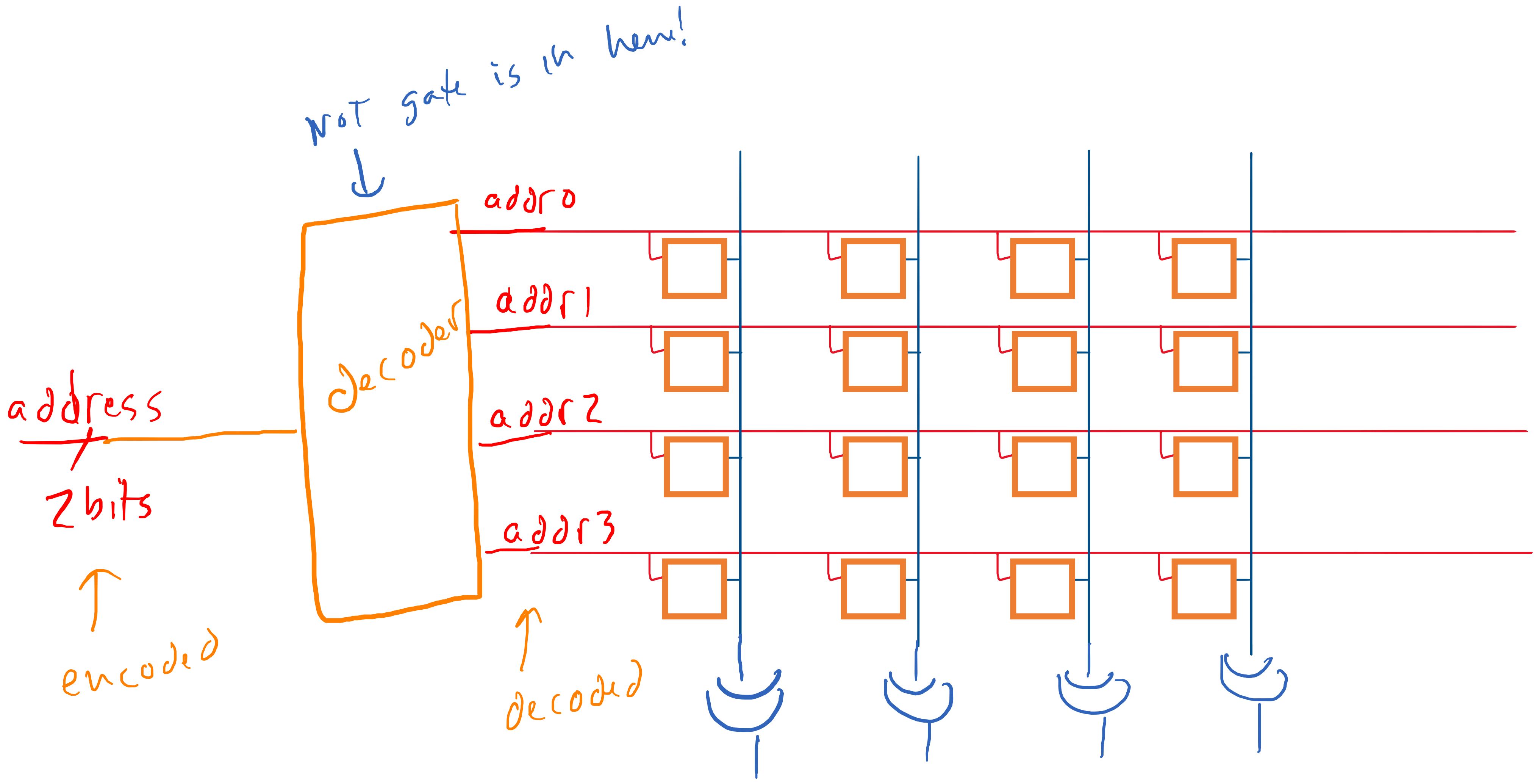


Array of ROM Cells



addr0 = 1 if (address = 00) else 0
addr1 = 1 if (address = 01) else 0

2-bit ROM



2-bits inputs
4-bits output

2-bit ROM of AND + OR

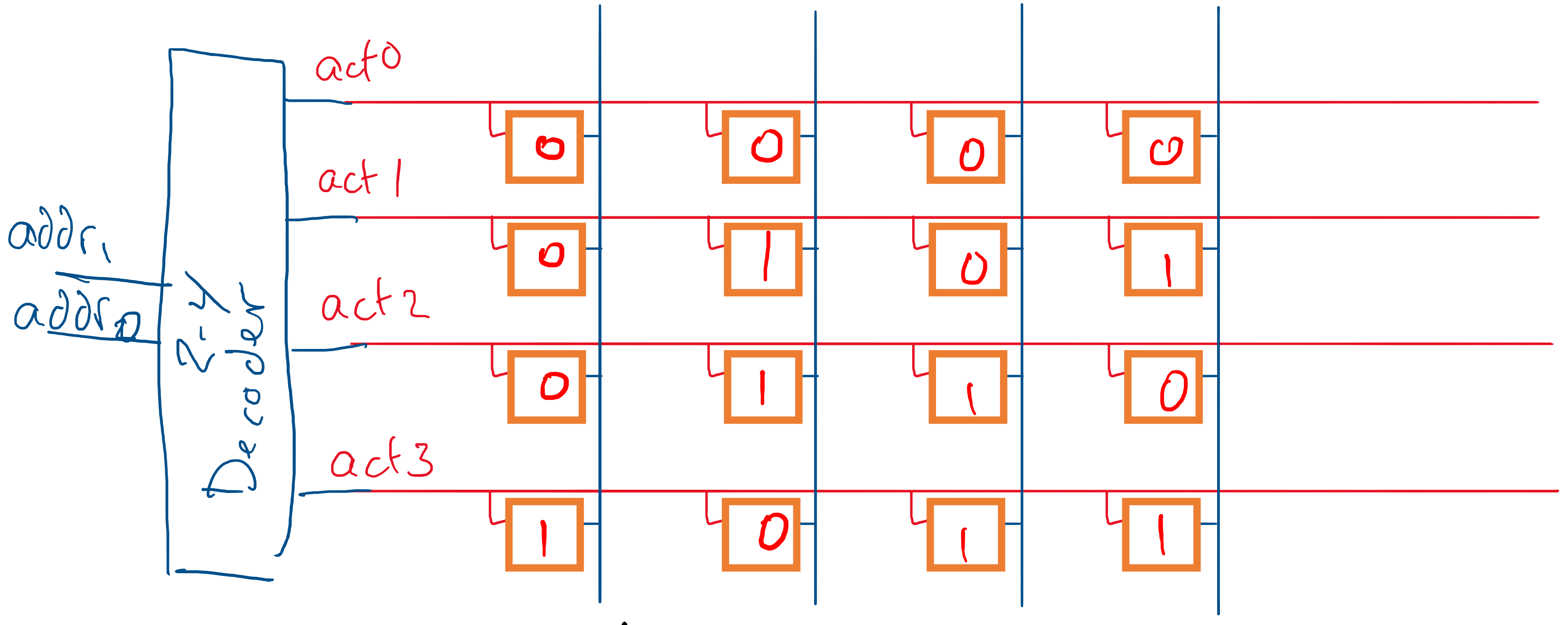
addr	out
00	0000
01	0101
10	0110
11	1011

addr	out_3
00	0
01	0
10	0
11	1

AND!

addr	out_2
00	0
01	1
10	1
11	0

XOR



out_3 out_2 out_1 out_0

|| $\Uparrow$ || ||

$addr_1$ & $addr_0$ XOR $addr_1$ $addr_0$

ROM in Verilog

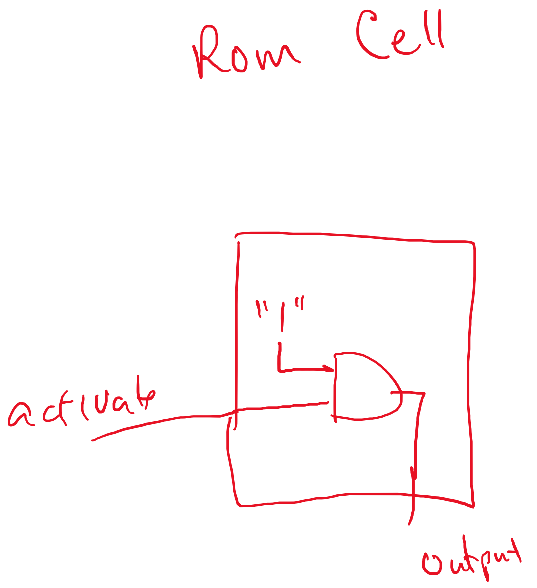
```
module ROM (
  input [1:0] addr,
  output [3:0] data
)
  logic [3:0] array [0:3]; //2D Array
  assign array = { 4’b0011, 4’b0110, 4’b0101, 4’b1100}
  assign data = array[addr];
endmodule
```

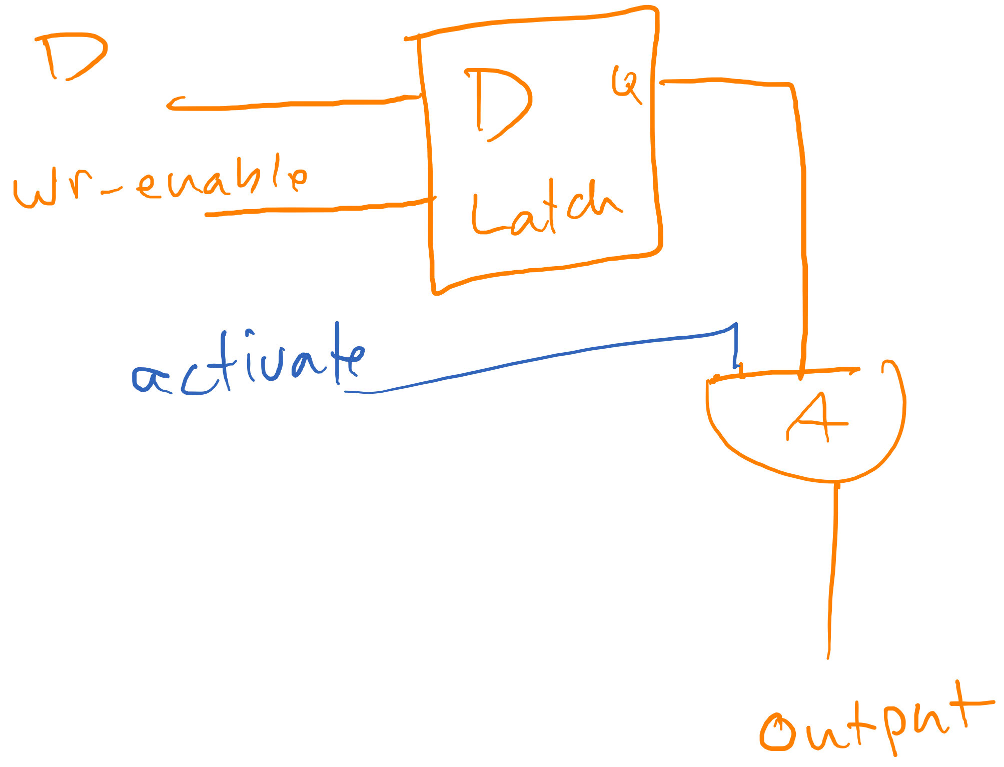
RAM

- Similar to ROM
- BUT WRITABLE!

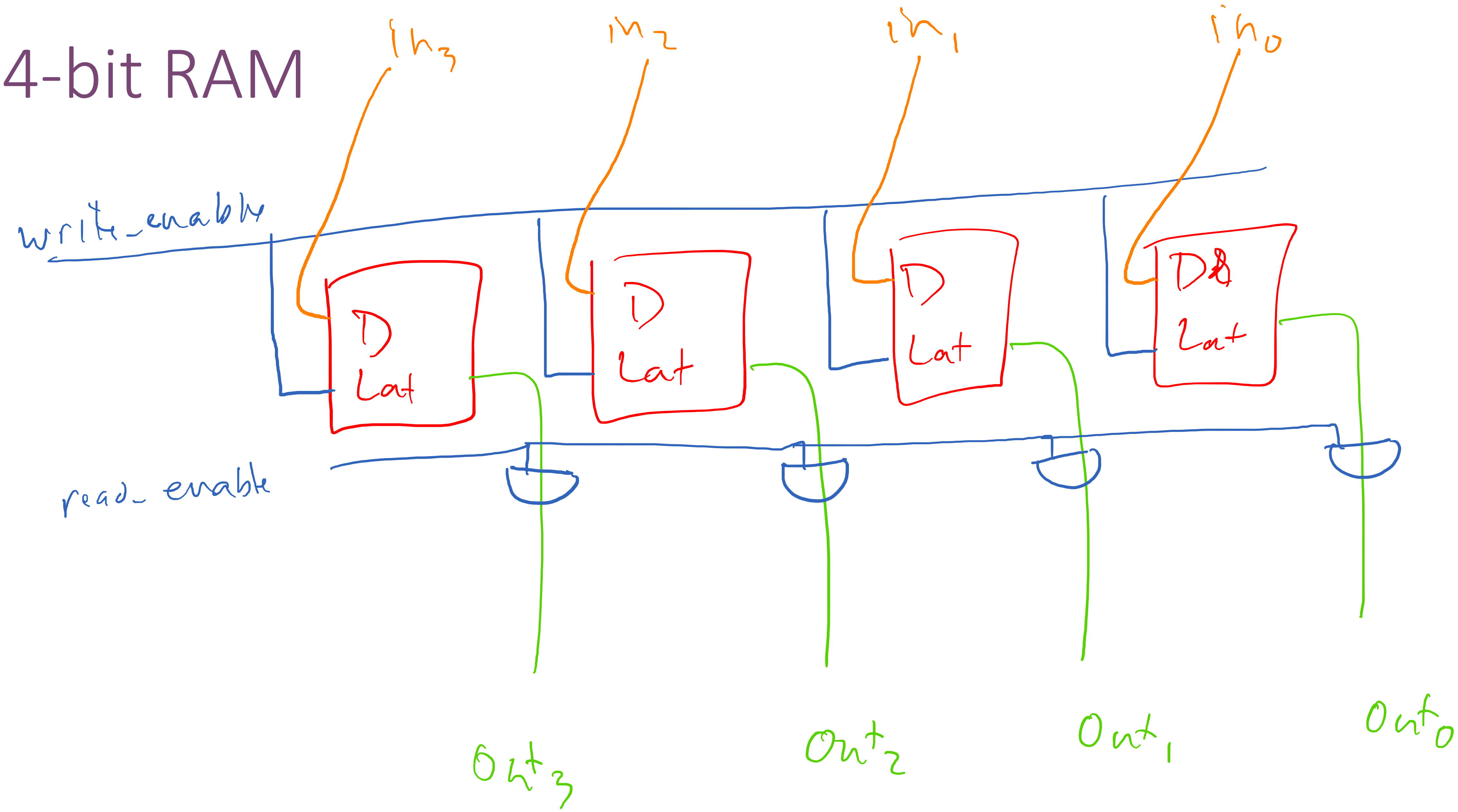
RAM

- Similar to ROM
- BUT WRITABLE!

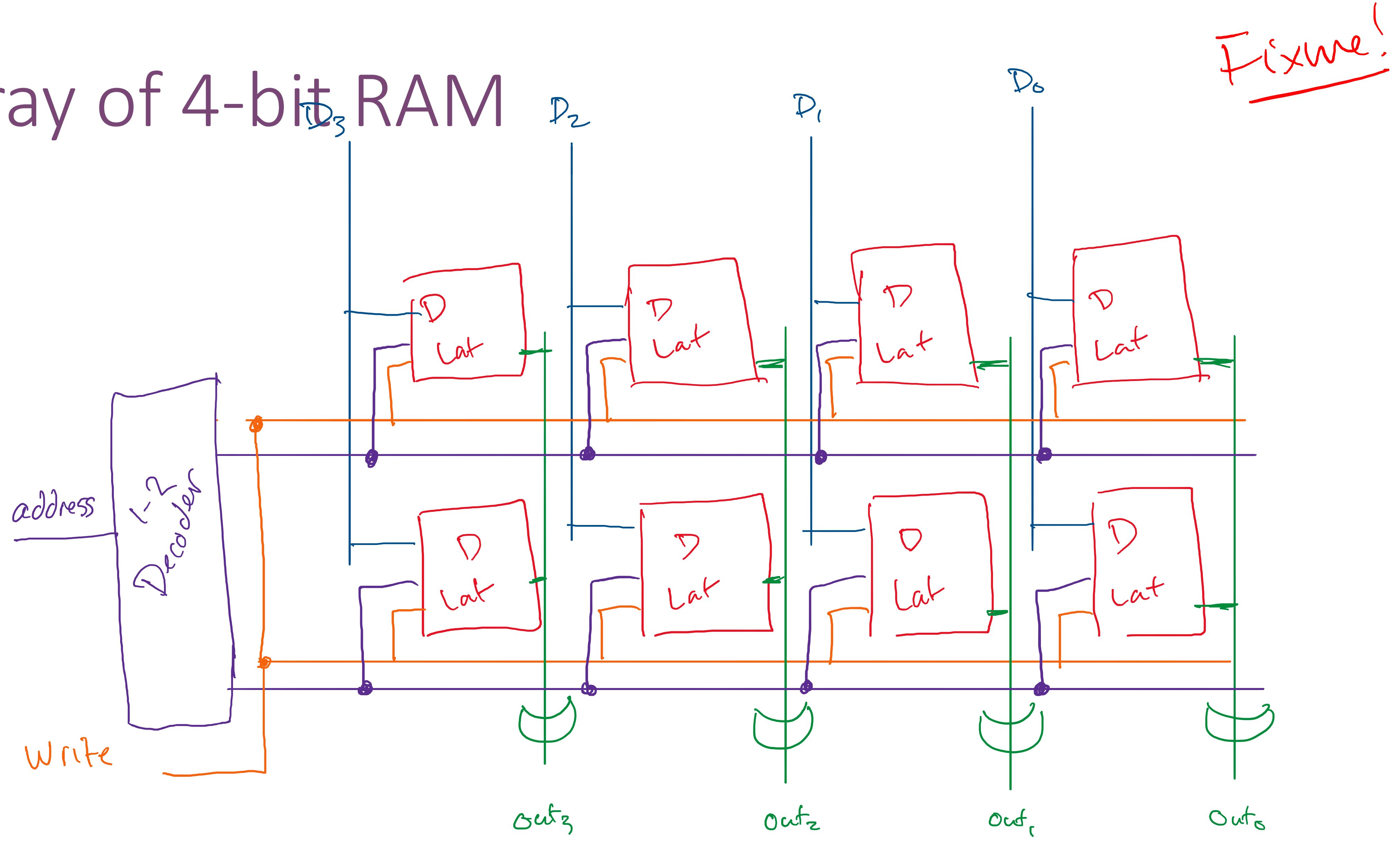




4-bit RAM


in₃
in₂
in₁
in₀
write_enable
read_enable
D Lat
D Lat
D Lat
D Lat
Out₃
Out₂
Out₁
Out₀

Array of 4-bit RAM



Flip-Flop RAM in Verilog

```
module RAM (
  input        clk,
  input [1:0] addr,
  input         set,
  input [3:0]  set_data,
  output [3:0] read_data
)
  logic [3:0] array [0:3]; //2D Array



  assign read_data = array[addr];
endmodule
```

Flip-Flop RAM in Verilog

```
module RAM (
  input        clk,
  input [1:0] addr,
  input        set,
  input [3:0]  set_data,
  output [3:0] read_data
)
  logic [3:0] array [0:3]; //2D Array
  always_ff @(posedge clk) begin
       if (set) array[addr] <= set_data;
  end
  assign read_data = array[addr];
endmodule
```

Aside: Latch RAM in Verilog

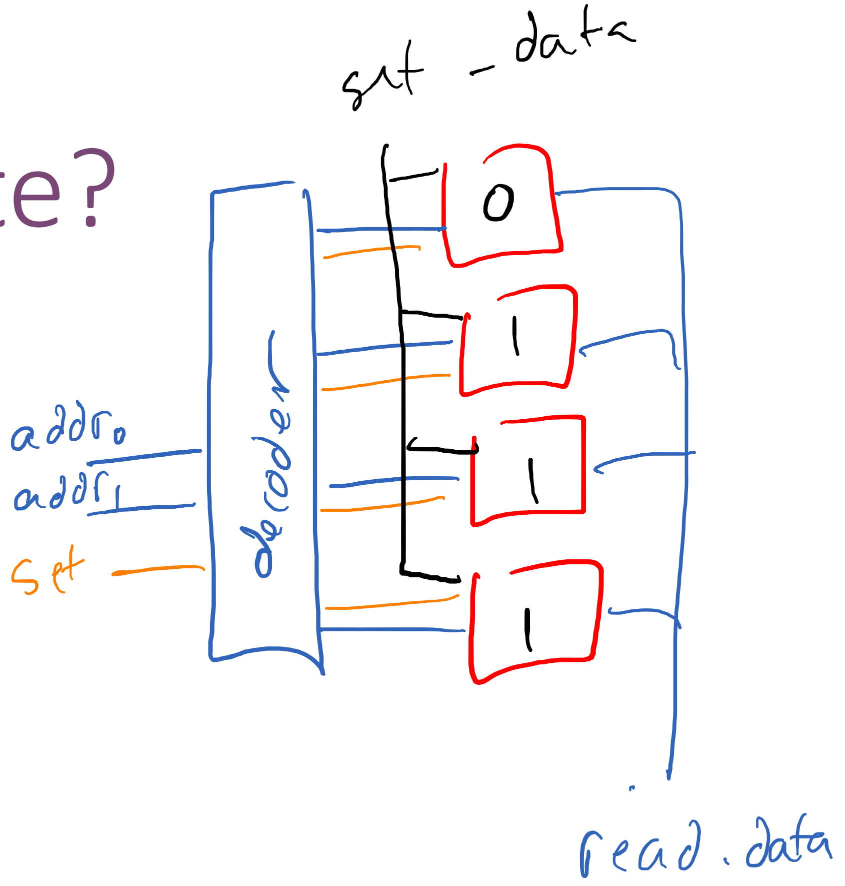
```
module RAM (          ← does not need clk

  input [1:0] addr,
  input        set,
  input [3:0]  set_data,
  output [3:0] read_data
)
  logic [3:0] array [0:3]; //2D Array
  always_latch begin //if you really want a latch
       if (set) array[addr] = set_data; ← not have default
  end
  assign read_data = array[addr];
endmodule
```

Any glitch on `set` will kill this.

Do not use in class!

Can I make this into an AND gate?

```
module RAM (
  input        clk,
  input [1:0] addr,
  input         set,
  input [~~3~~:0]  set_data,
  output [~~3~~:0] read_data
)
  logic [~~3~~:0] array [0:3]; //2D Array
  always_ff @(posedge clk) begin
       if (set) array[addr] <= set_data;
  end
  assign read_data = array[addr];
endmodule
```



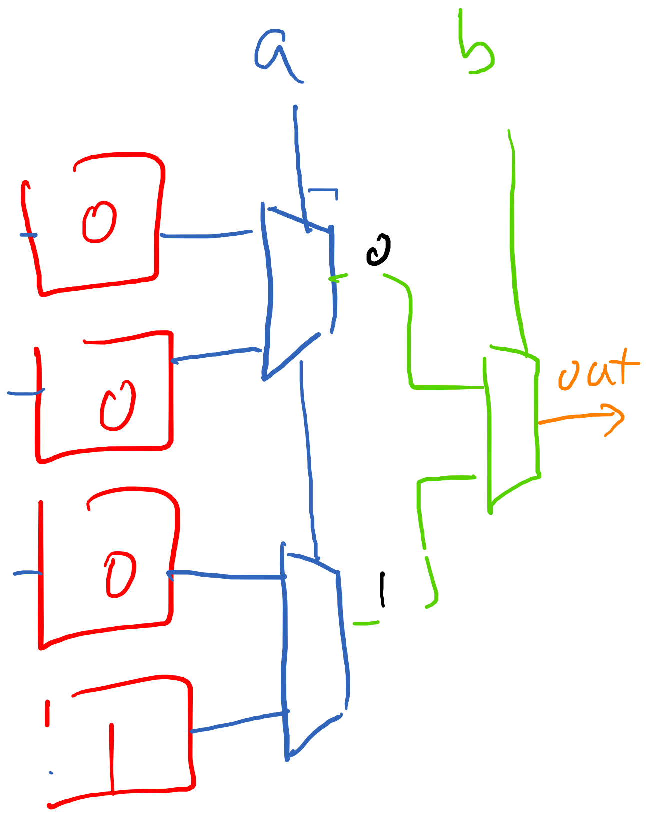
Look-Up Table (LUT)

- DON'T compute a Boolean equation
- DO pre-compute all solutions in a table
- DO look up the Boolean result in the table

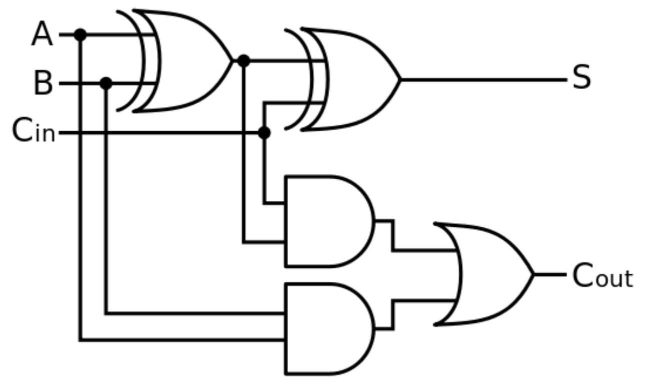
- Examples:

```
s = a ^ b;
c = a & b;
```

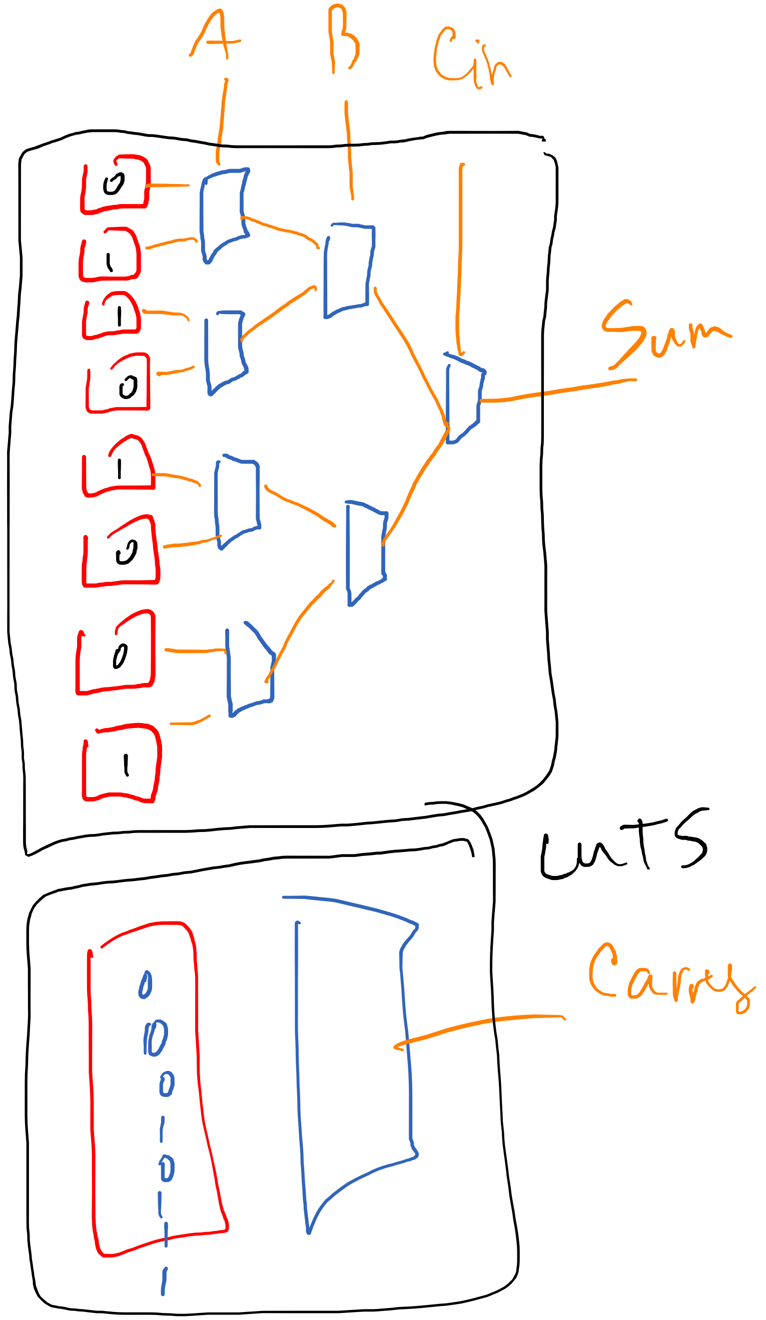
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



Full-Adder LUT



Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



3-input LUT

LUT size

- Why not a 1000-input,100-output LUT?

- 3 inputs => 2^3 rows = 8 rows
- 4 inputs => 2^4 rows = 16 rows
- 5 inputs => 2^5 rows = 32 rows
- …
- 64 inputs => 2^{64} rows = 1.85 x10^{19} rows

- LUT input size does **not** scale well.

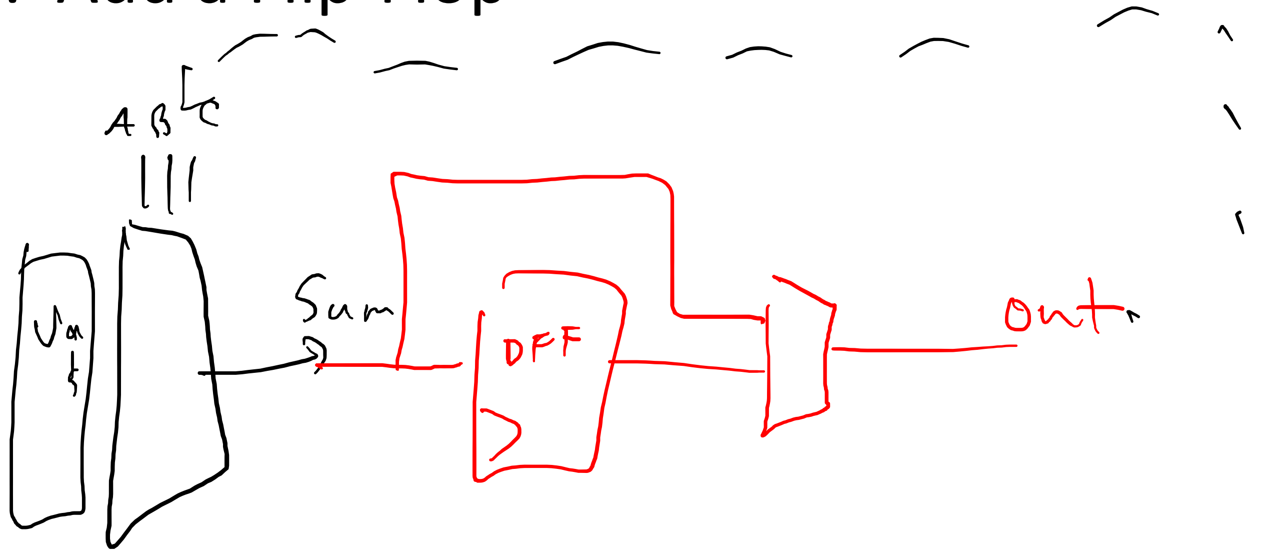
Divide and Conquer with LUTs

- 3-Bit Full Adder

Sequential Logic

- Problem: How do we handle sequential logic?
  - LUTs cannot contain state

- Solution: Add a Flip-Flop



Basic Logic Element

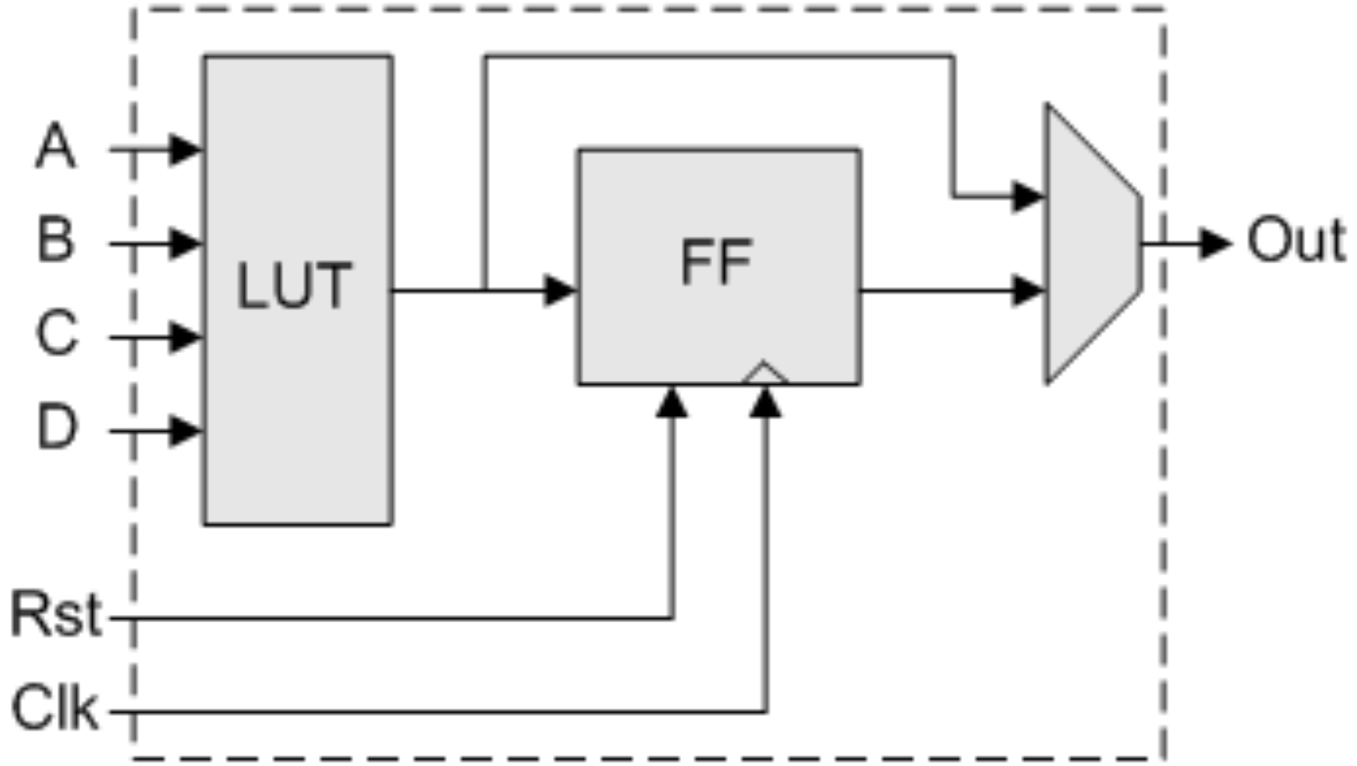


Fig.4 Example of Configurable Logic Cell.

Basic Logic Element

- What if I only want to store a value?

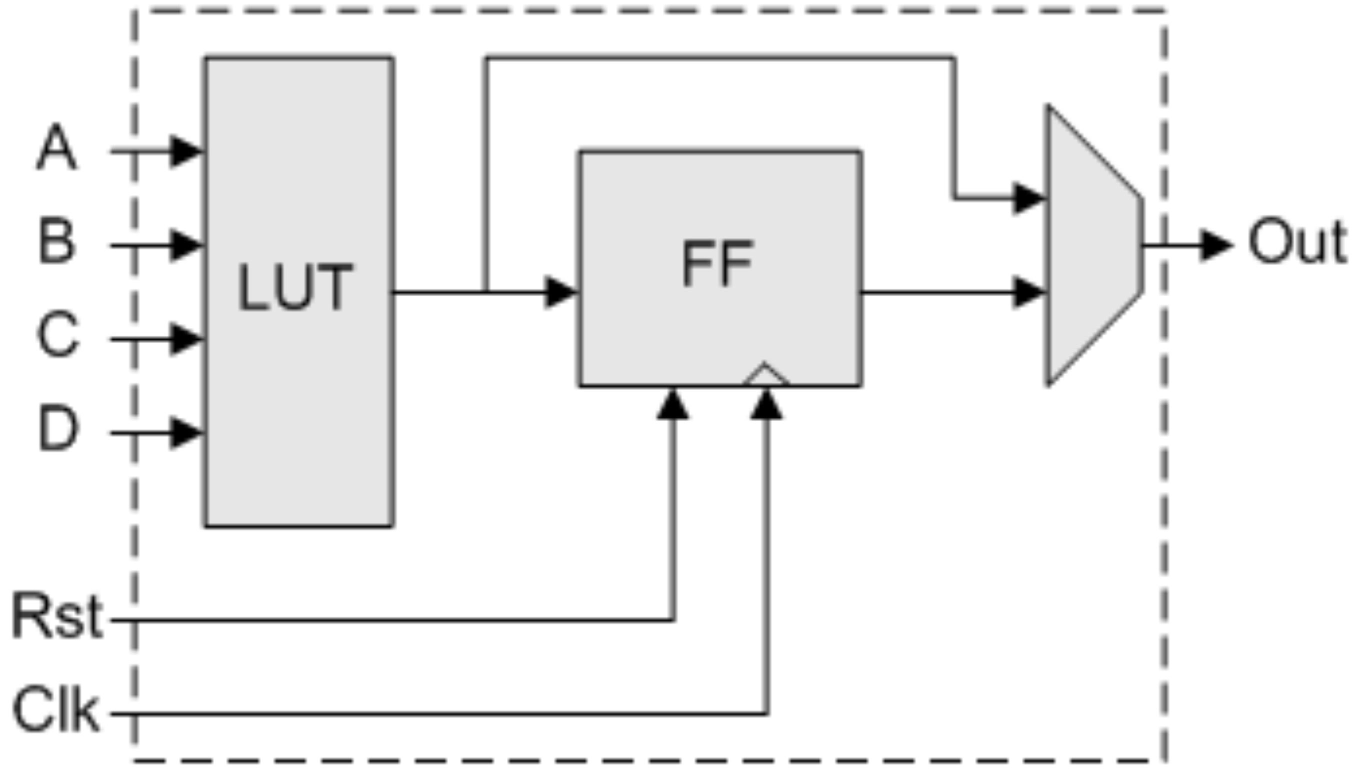
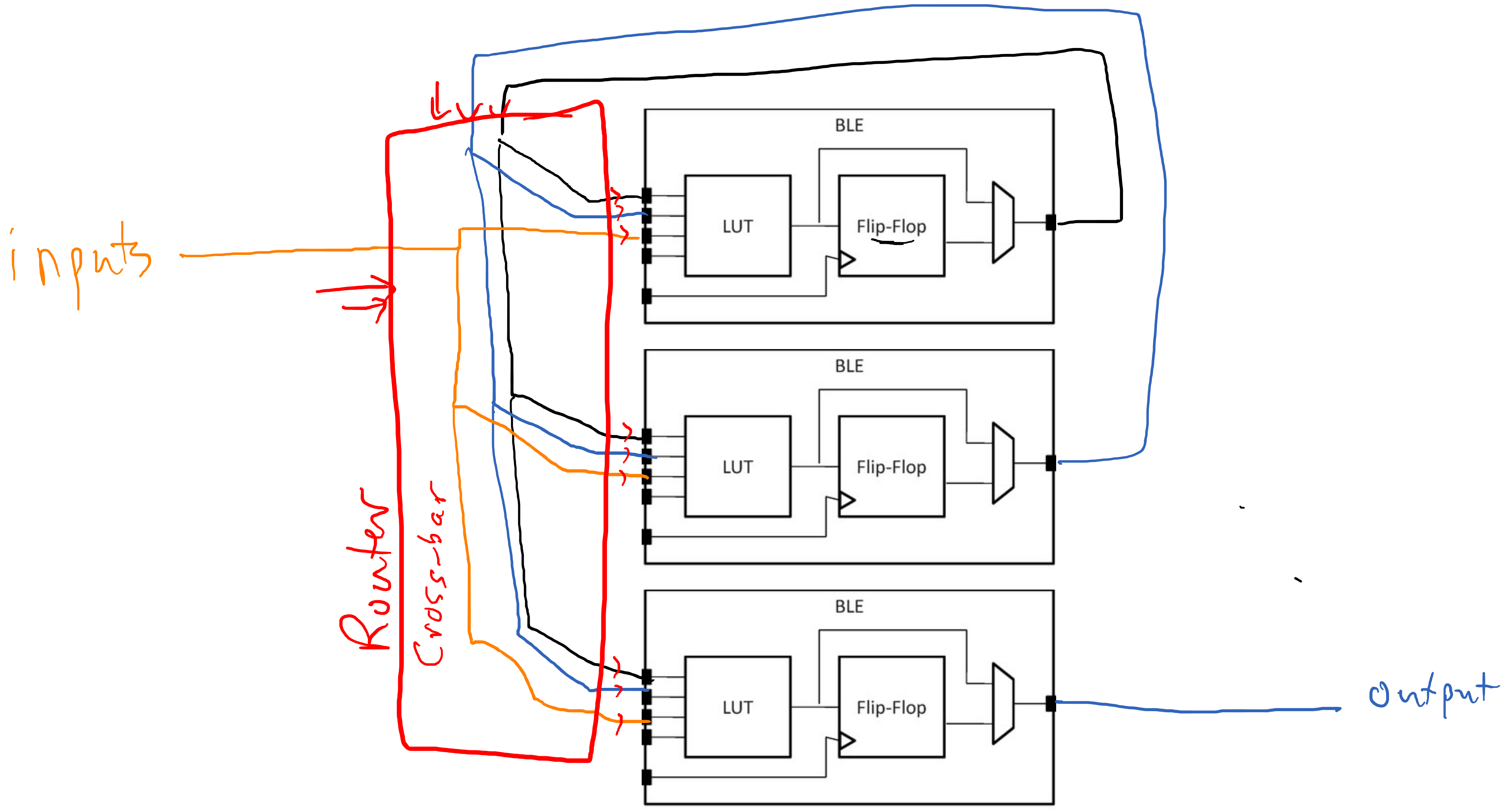
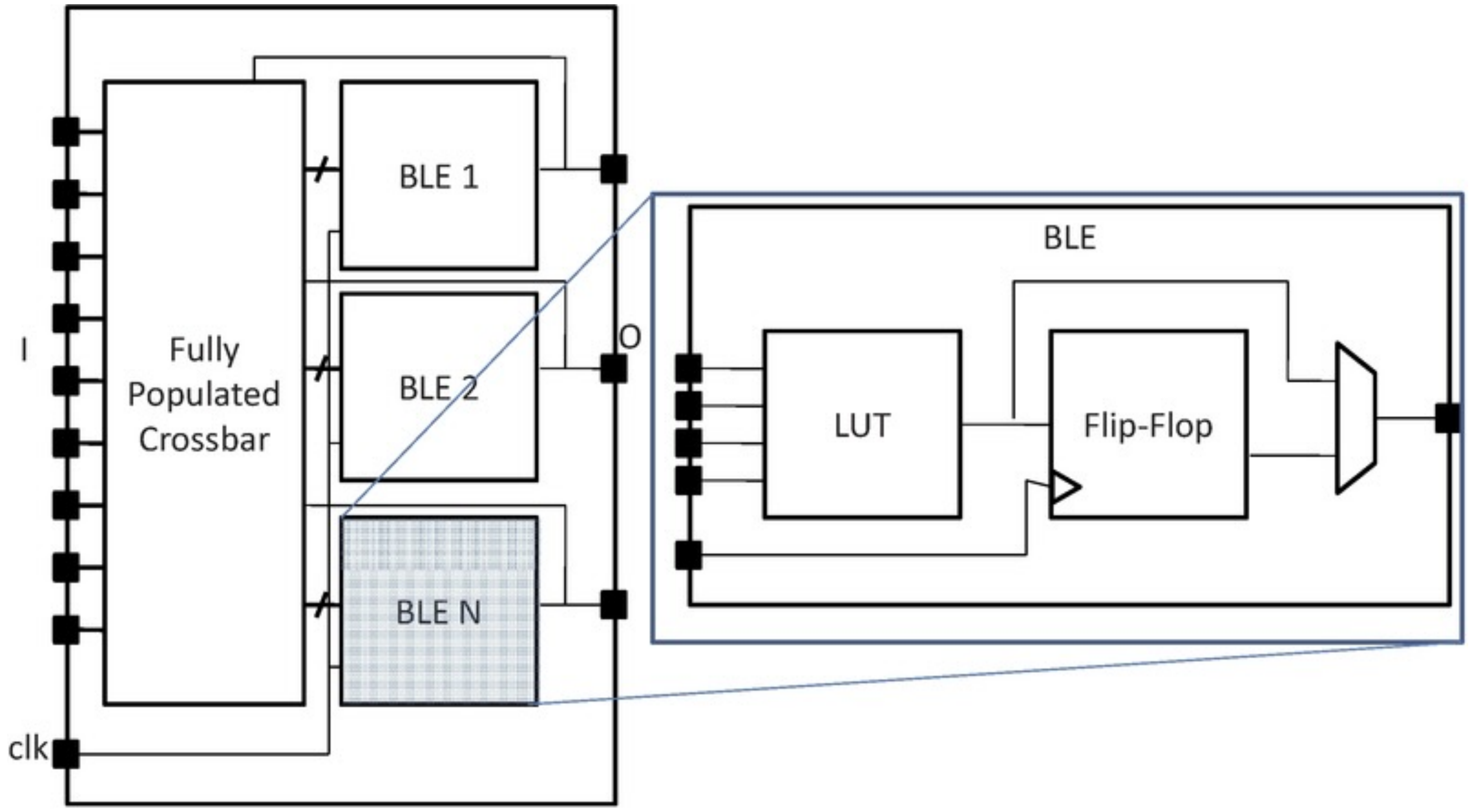


Fig.4 Example of Configurable Logic Cell.

Configurable Logic Block (CLB)

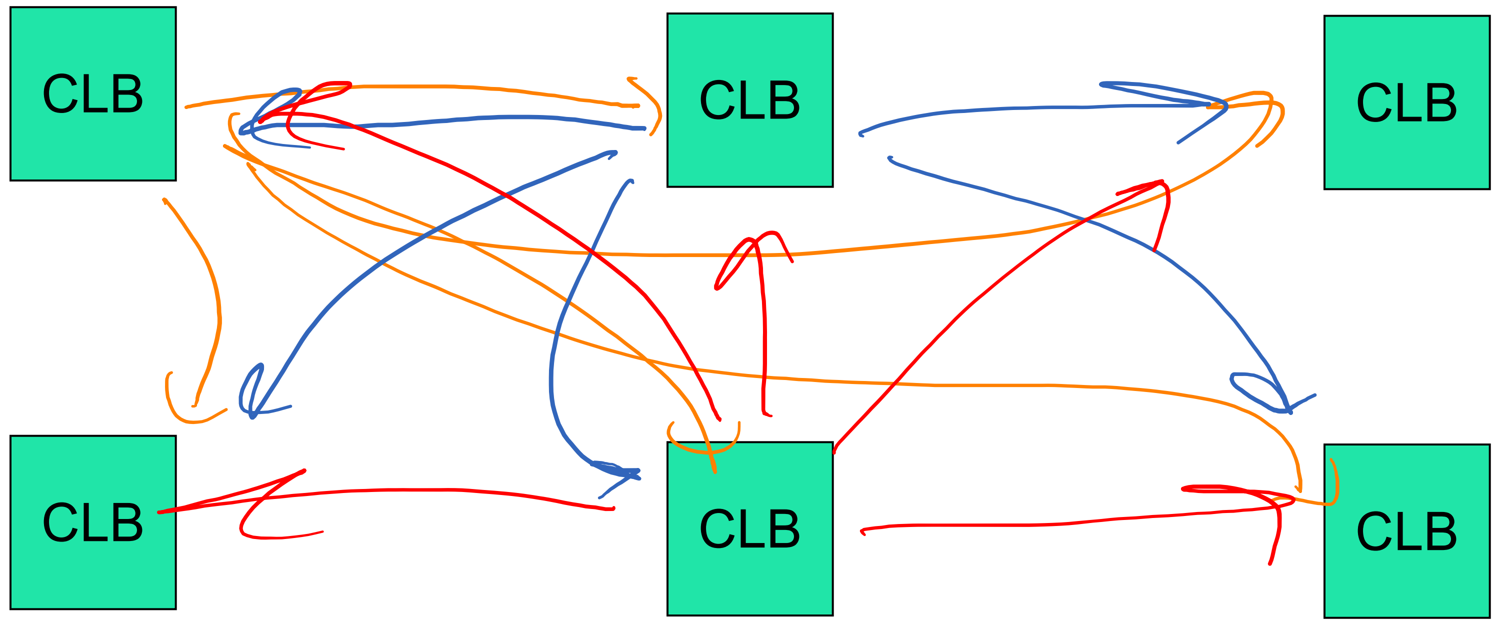


Configurable Logic Block (CLB)



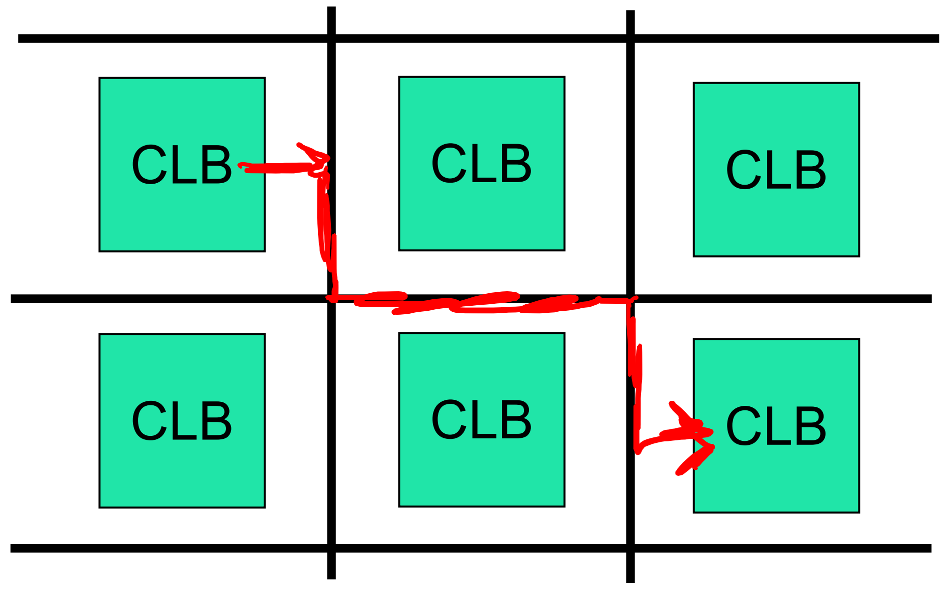
Connecting CLBs

- Q: How do CLBs talk to each other?
- A: Put wires everywhere!



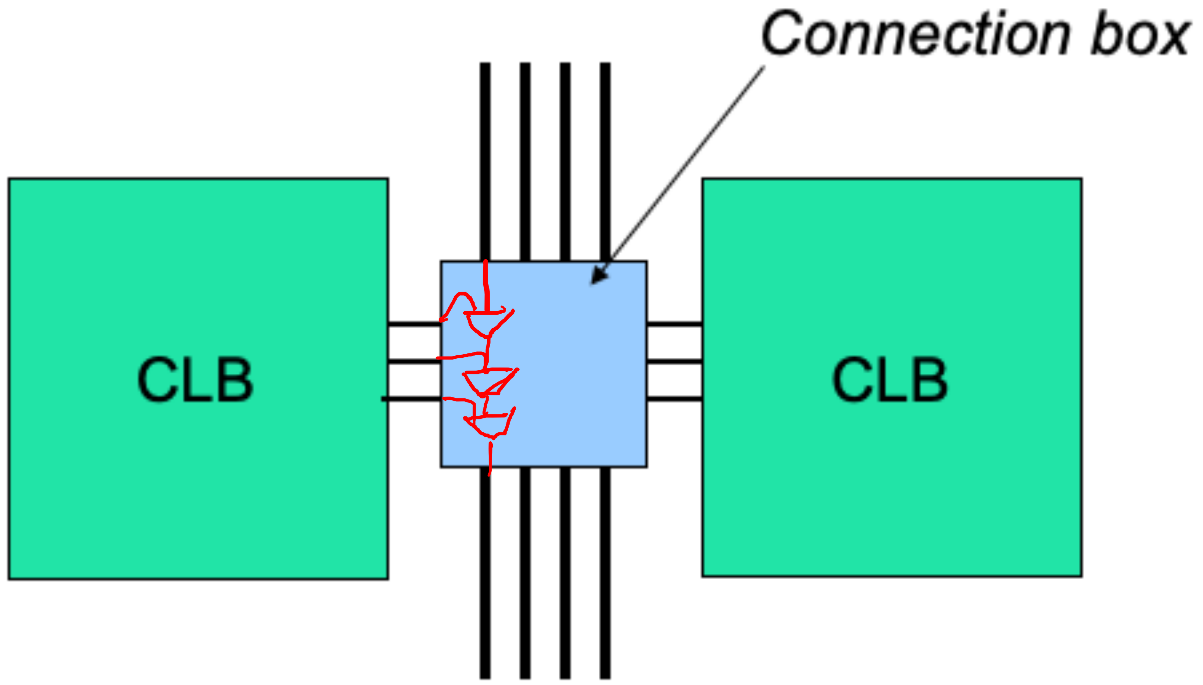
Connecting CLBs

- Q: How do CLBs talk to each other?
- A: Put wires everywhere (ok, almost everywhere)!

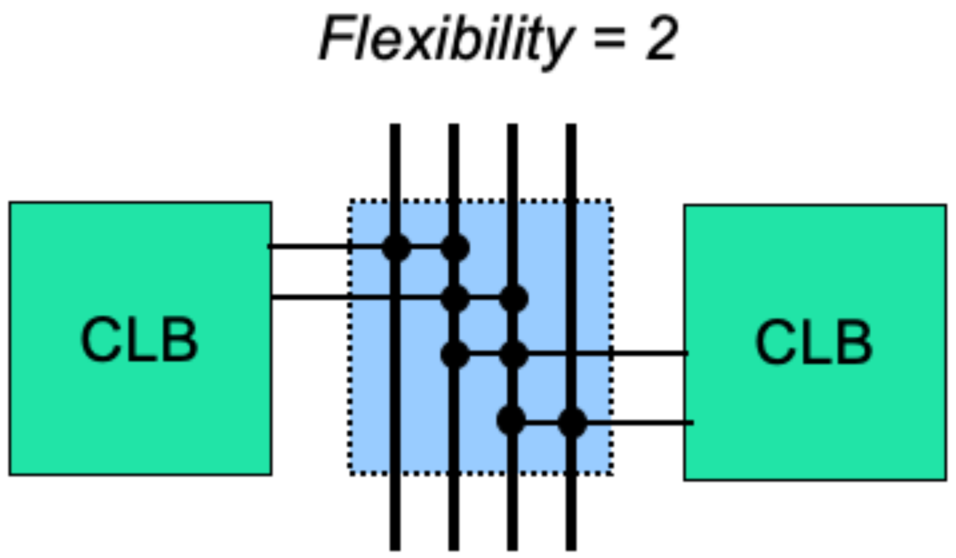


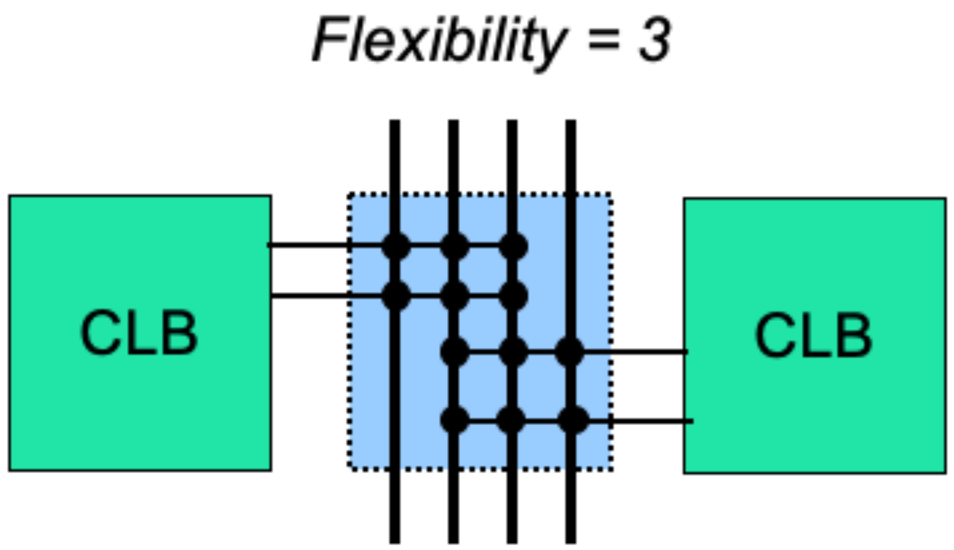
How to connect CLBs to wires?

- “Connection box”
  - Device that allows inputs and outputs of CLB to connect to different wires



Connection Box Flexibility

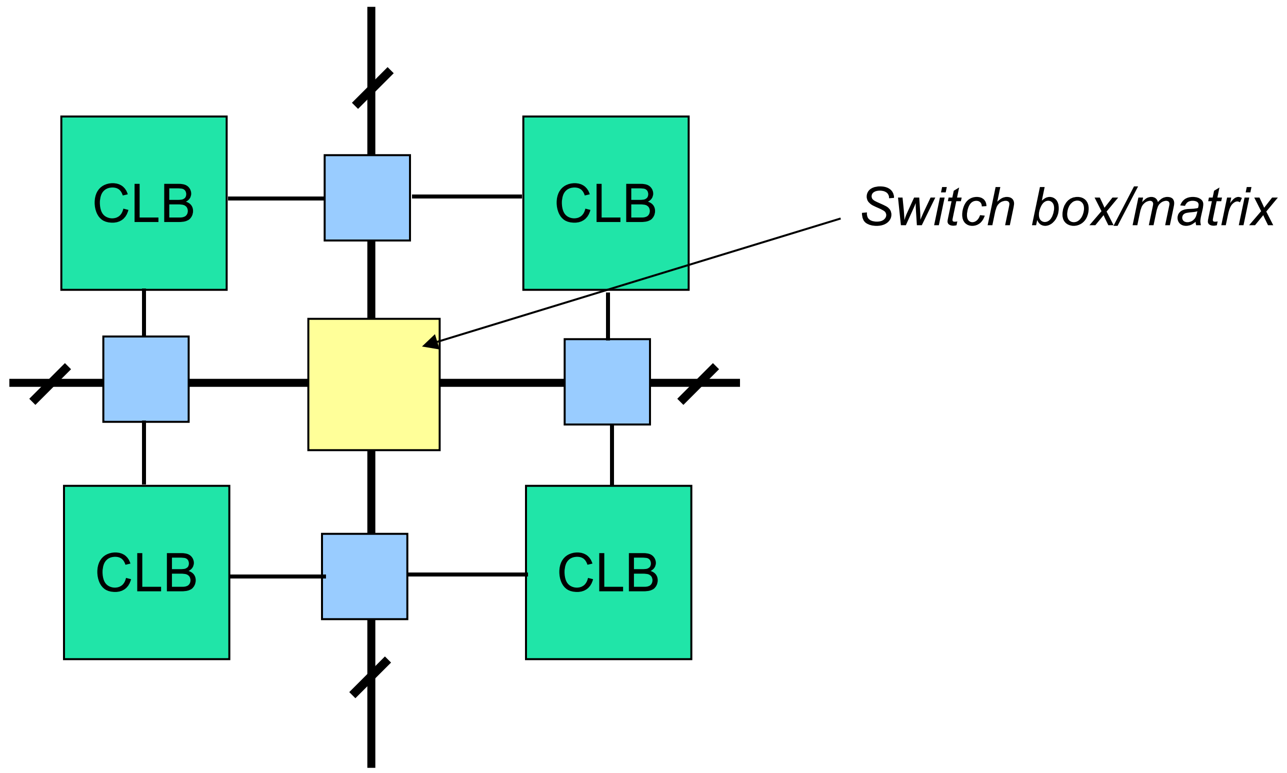




Dots represent* ***possible* *connections*

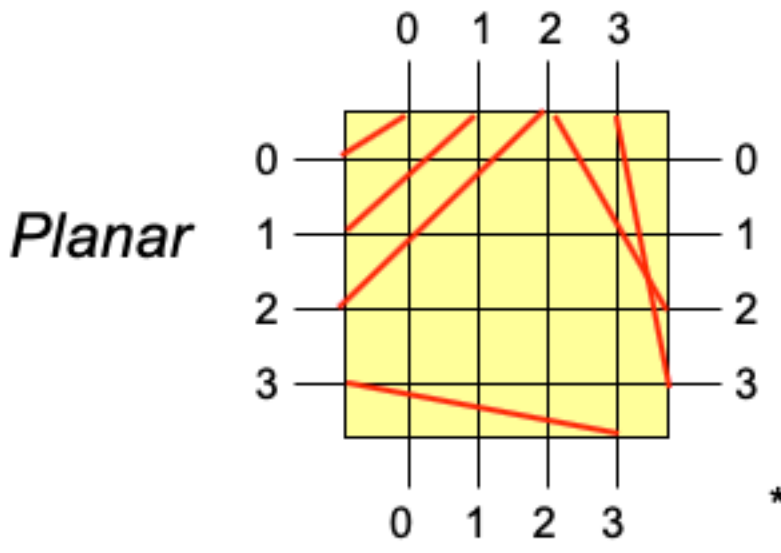
Switch Box

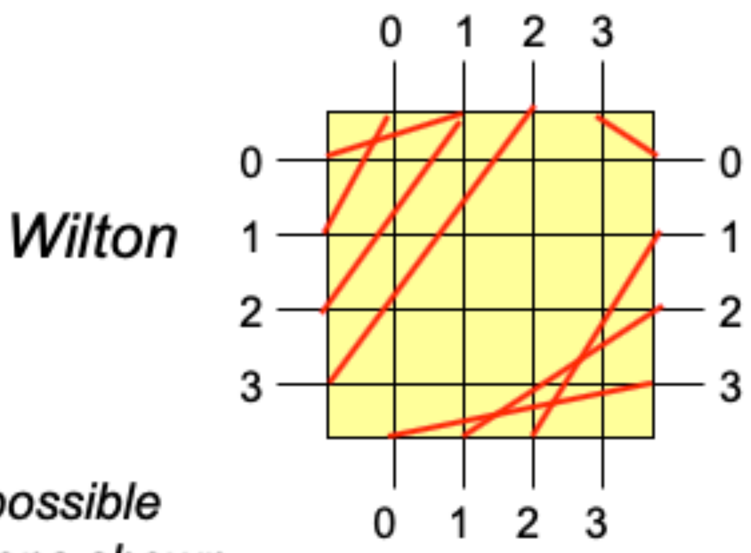
- Connects horizontal and vertical routing channels



Switch Box Connections

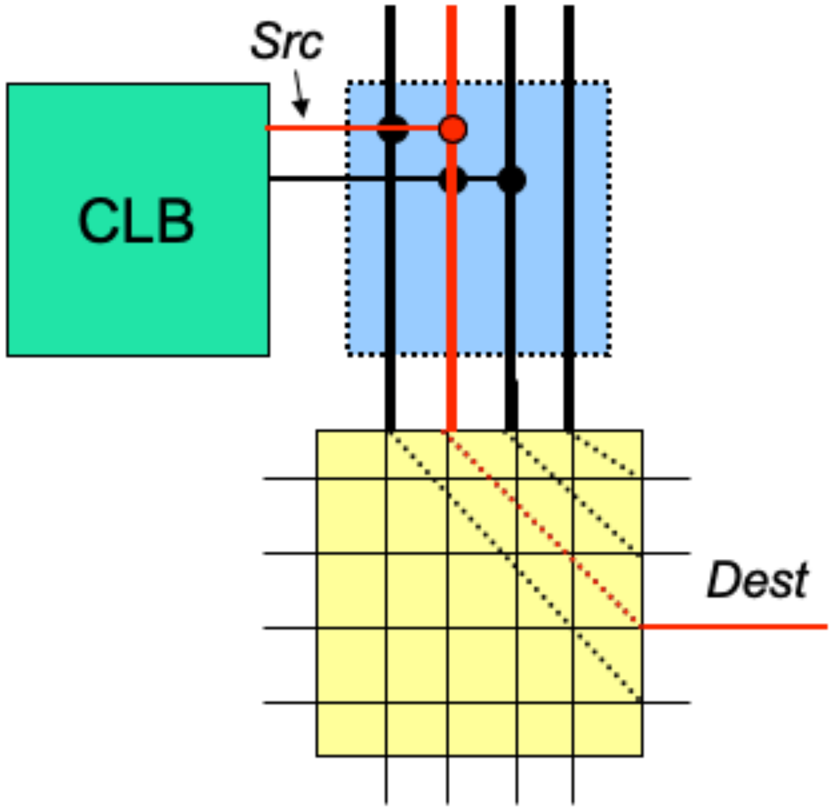
- Programmable connections between inputs and outputs

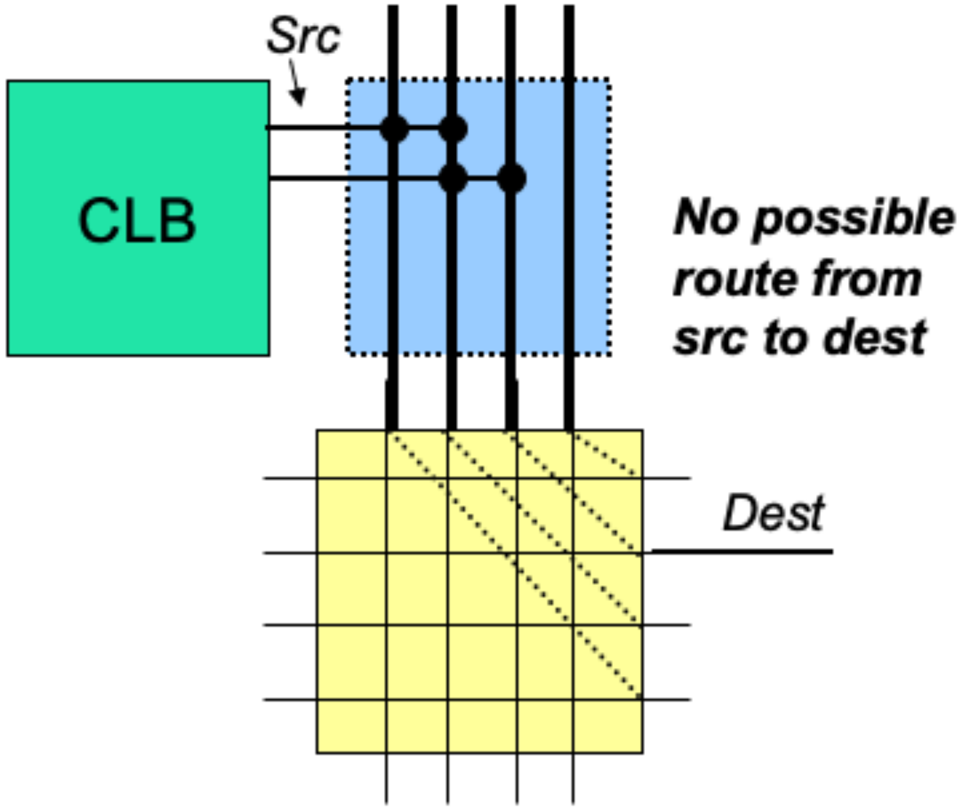




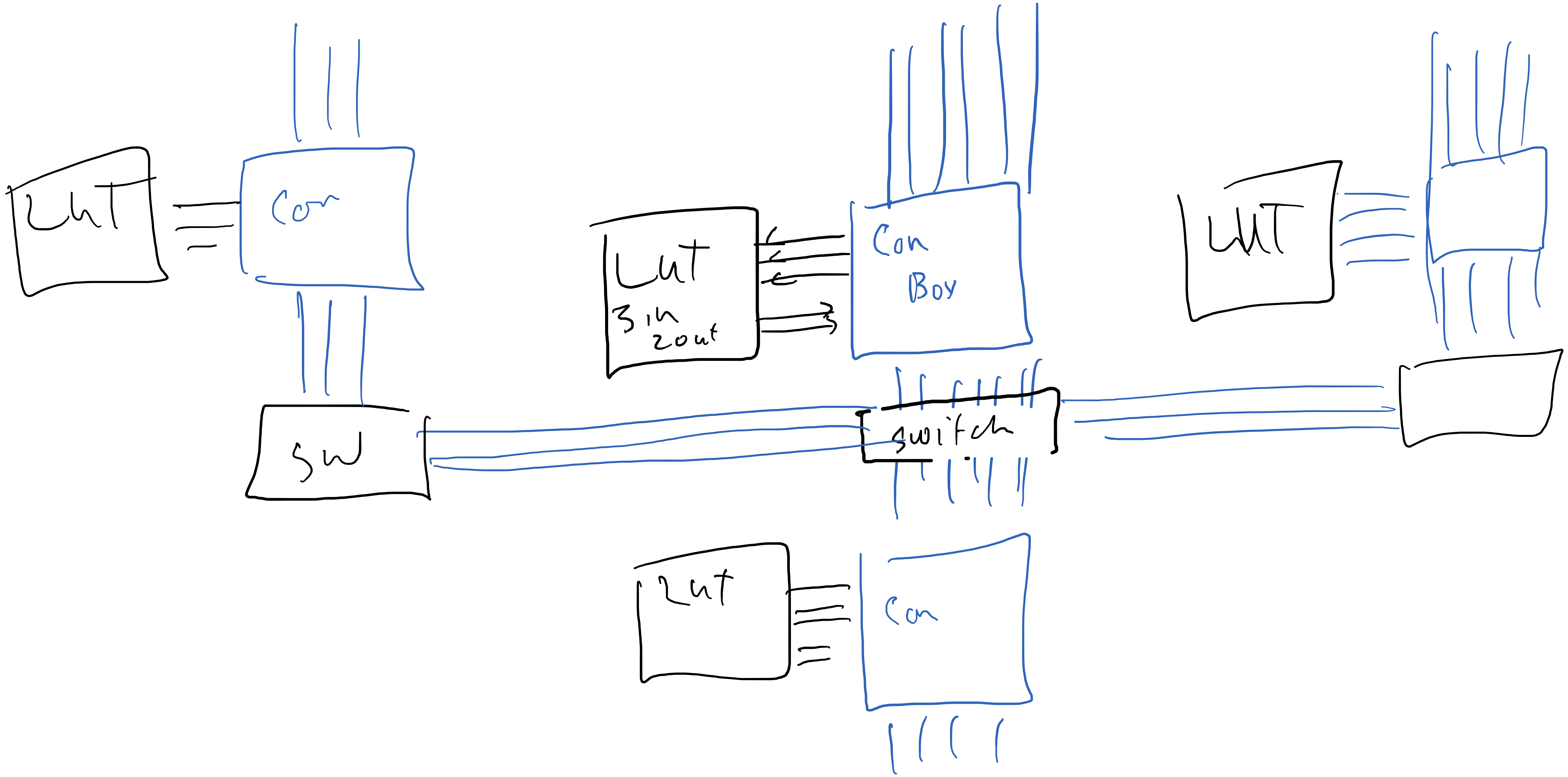
*Not all possible connections shown

Switch Box Connections



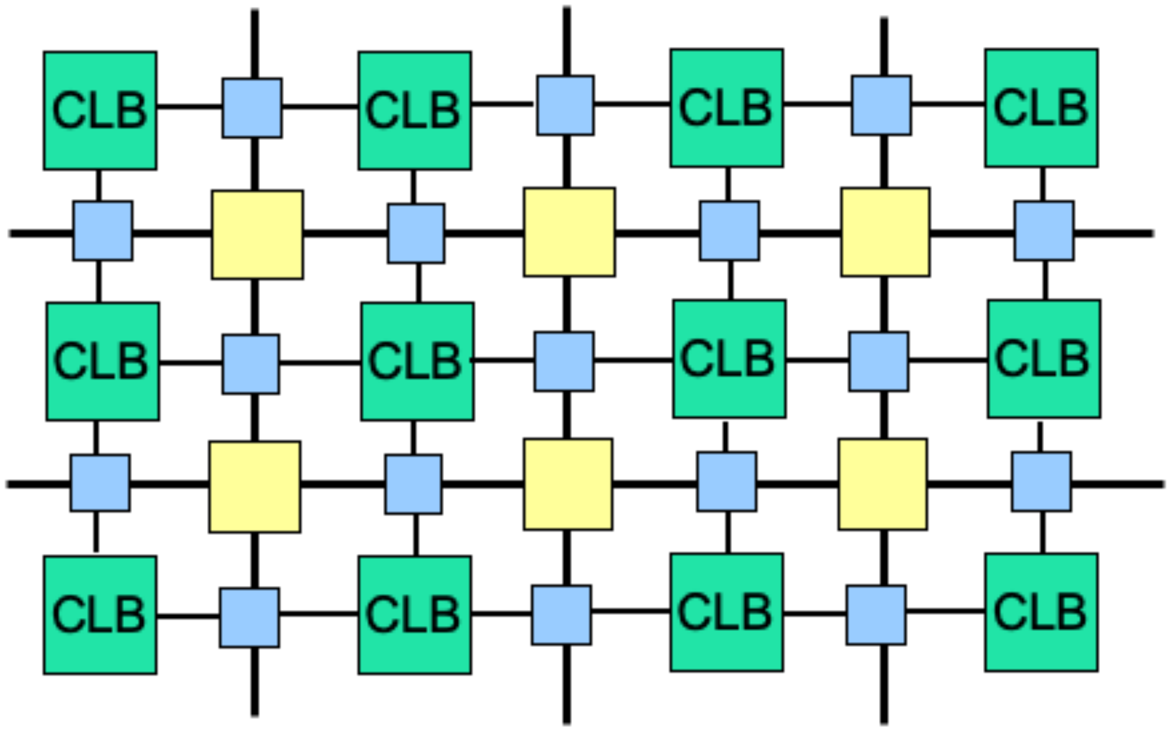


How to connect wires to each other?



FPGA "Fabric"

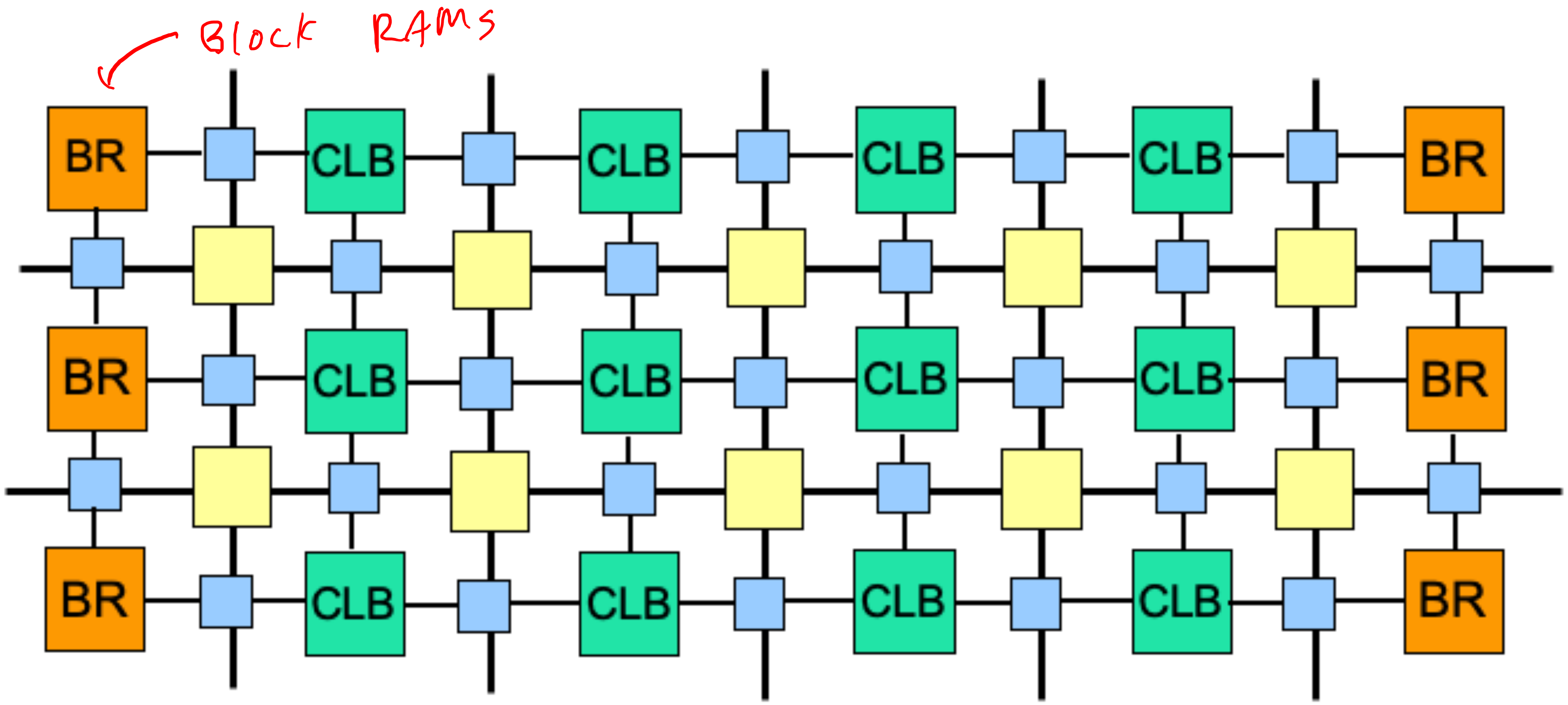
- 2D array of CLBs + interconnects



- Am I missing anything?

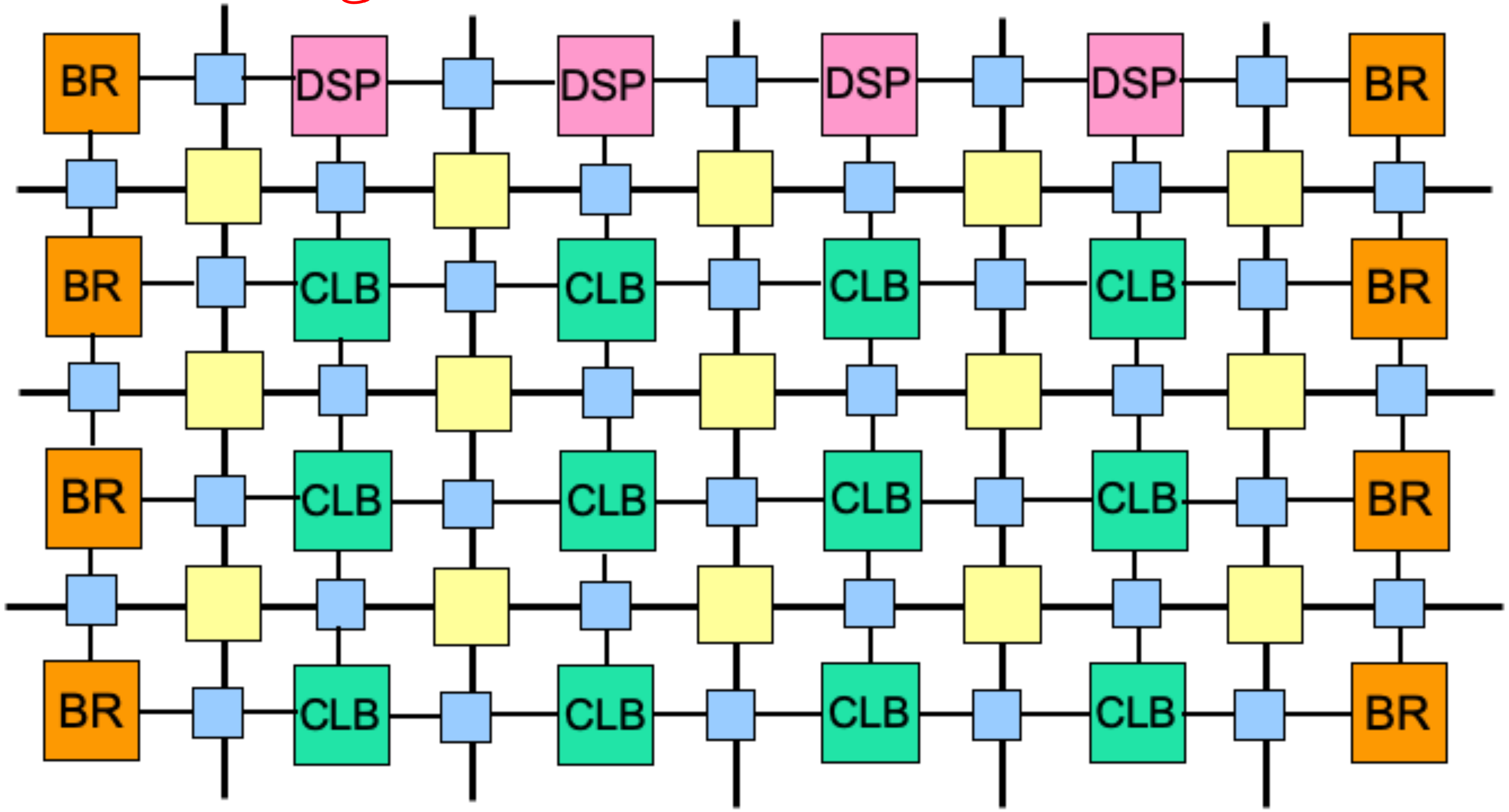
Block RAM

- Special blocks of just RAM

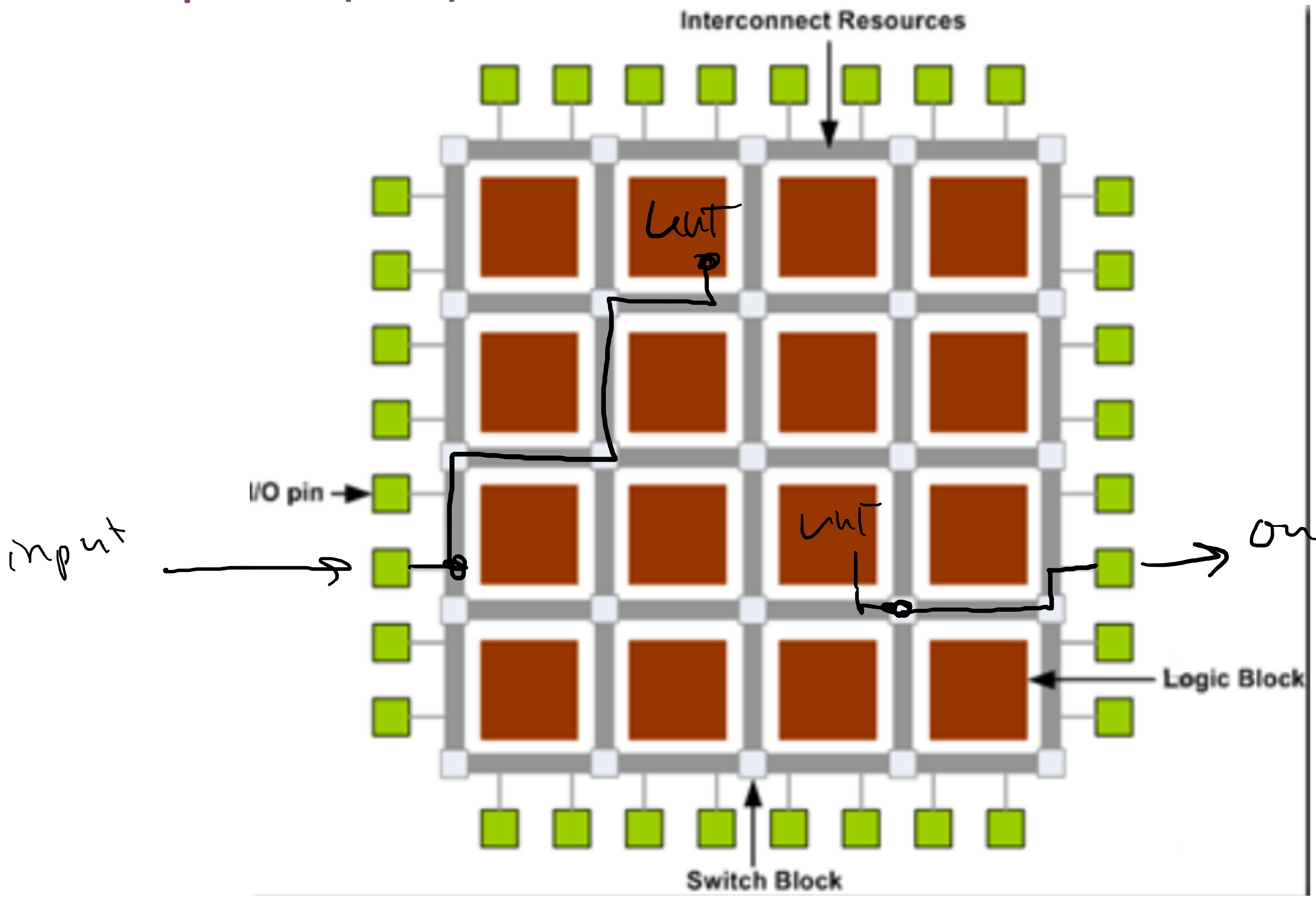


DSPs → Digital Signal Processor → dedicated math units

multiply



Input/Output (IO)



Next Time

- What’s next?
- Review