Review of Material from Lecture 2

EL GY 6463: Advanced Hardware Design

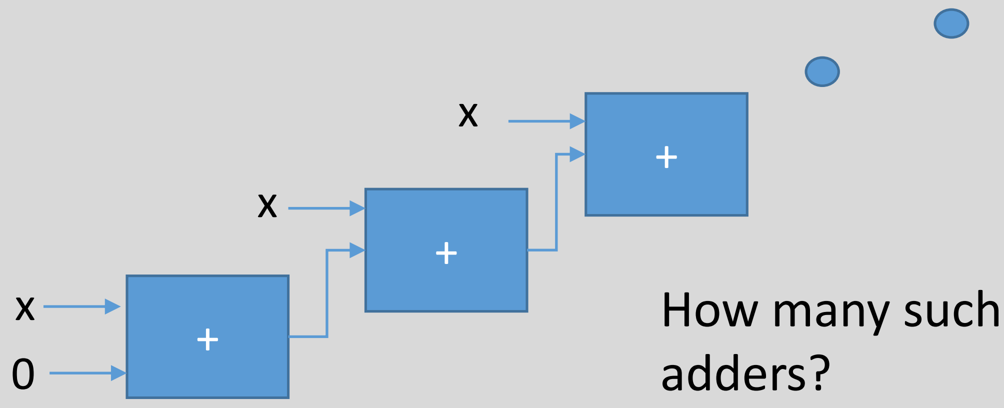
Instructor: Siddharth Garg

Finite State Machines (Data path and Control path Synthesis)

- Say you want to build a multiplier but only have access to adders and comparators.
  - Solution: repeated addition. Simple Verilog combinational description

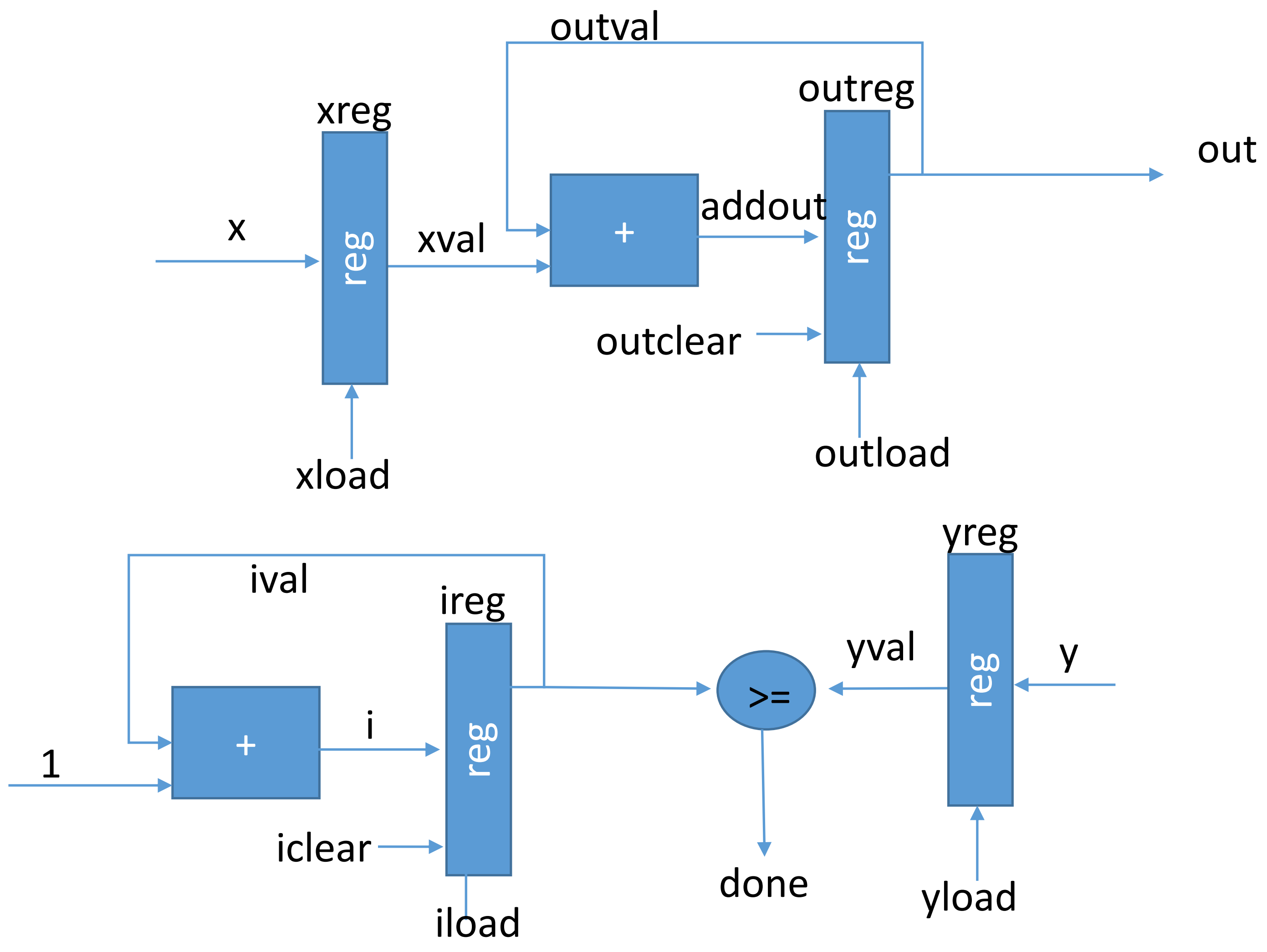
```
module mult #(parameter W = 8)
             (input [W-1:0] x, input [W-1:0] y; output reg [2*W-1:0] out);

reg [W-1:0] i;

always @(*)
   out = 0;
   for (i=0; i<y; i=i+1) begin
      out = out + x;
   end

endmodule
```

Will work in simulation but does not synthesize?



How many such adders?

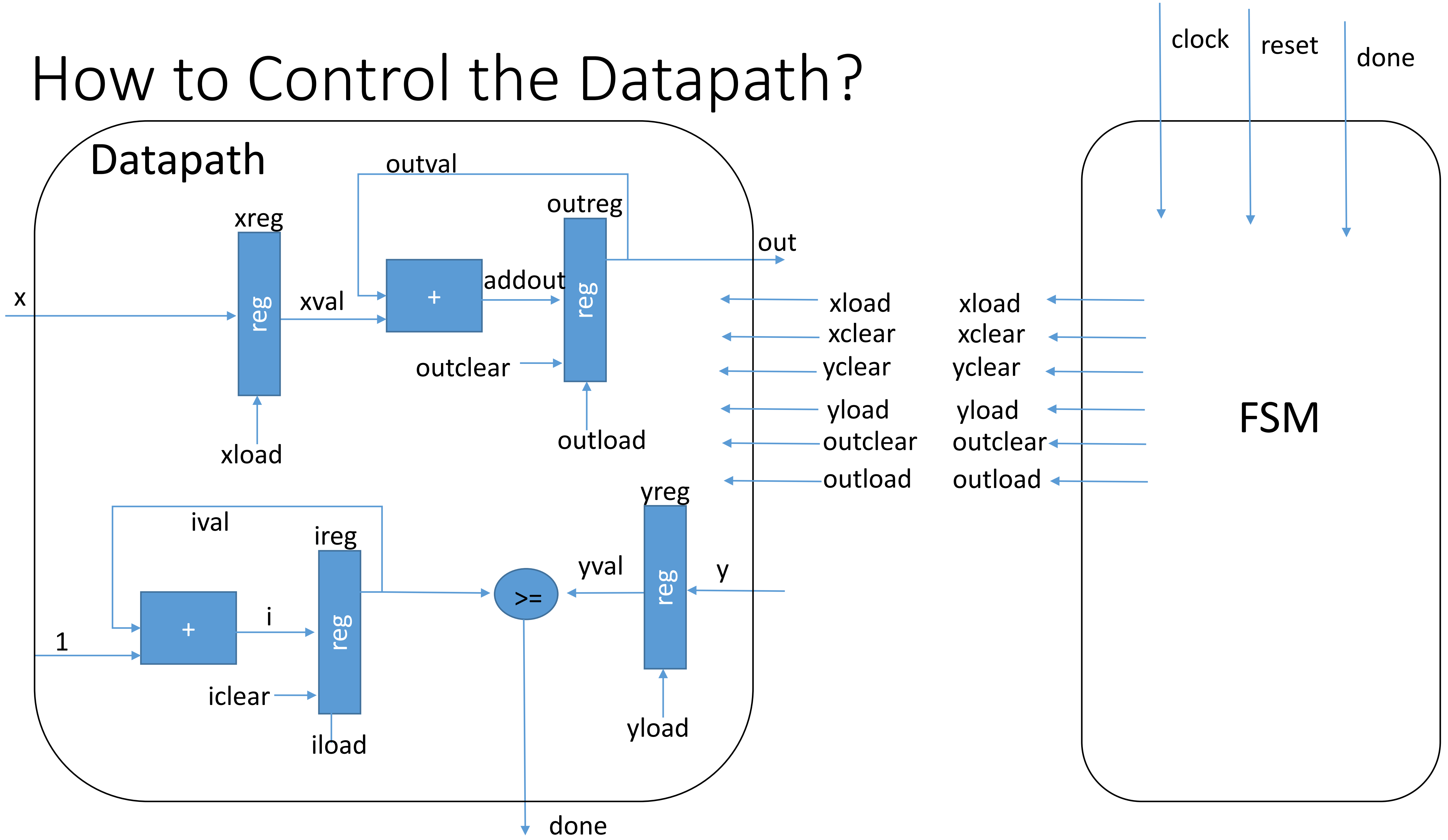
Better Solution: Use One Adder Repeatedly



```
module increment
# (parameter W = 8)
(input [W-1:0] in1,
output reg [W-1:0] out);


always @ (*) begin
    out = in1 + 1;
end
endmodule
```

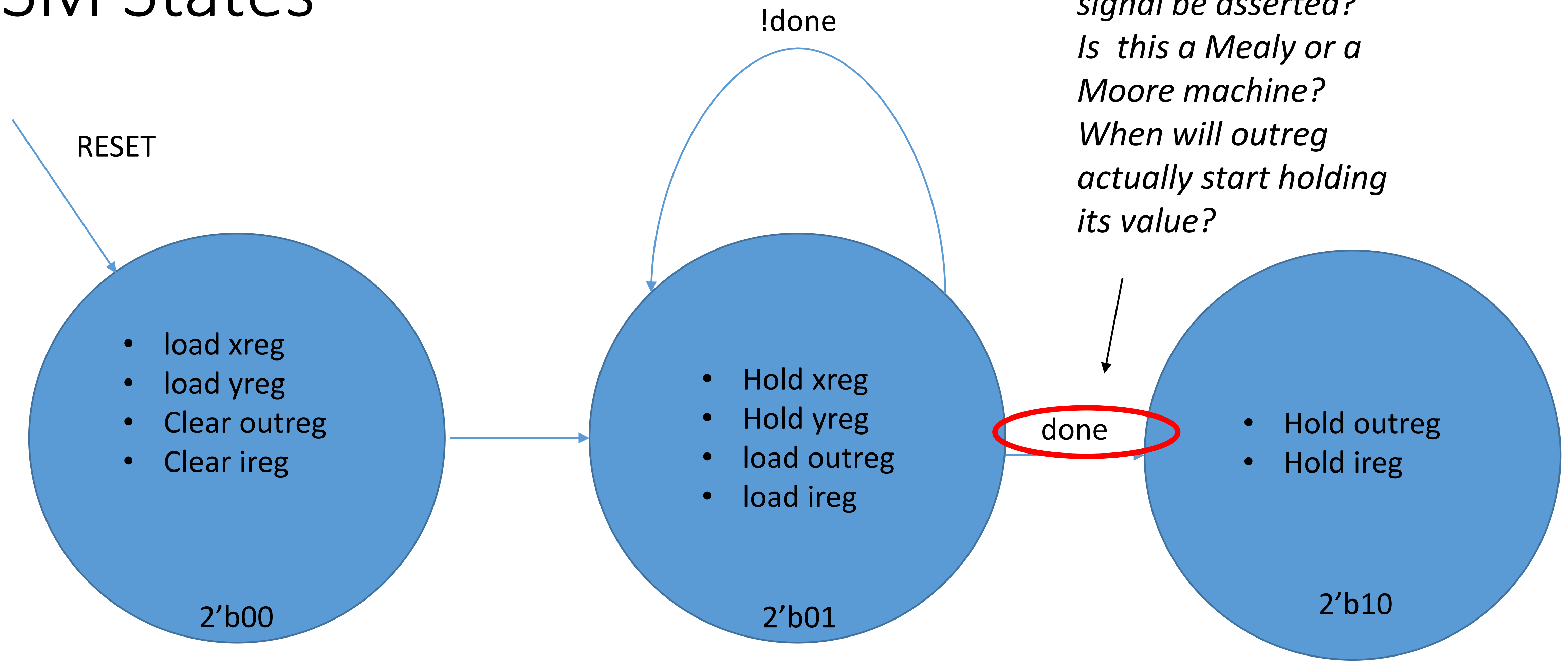
```
module comparator
# (parameter W = 8)
(input [W-1:0] ival,
 input [W-1:0] yval,
 output reg done);

always @ (*) begin
   done = 0;
   if (ival   >= yval ) //Yes?
     done=1;
end
endmodule
```

How to Control the Datapath?



FSM States

Question: for what value of i should done signal be asserted?
Is this a Mealy or a Moore machine?
When will outreg actually start holding its value?



Increment Module

```
module comparator
# (parameter W = 8)
(input [W-1:0] ival,
 input [W-1:0] yval,
 output reg done);

always @ (*) begin
  done = 0;
  if (ival +1   >= yval -1 )
    done=1;
end
endmodule
```

Note:

- ival starts from 0 instead of 1
- When done signal is asserted, outreg only stops loading in the next clock cycle

Putting it All Together

```
module multiplier
# (parameter W = 8)
(input [W-1:0] x, input [W-1:0] y, input clk, input reset, output [2*W-1:0] out);

//variable declarations. Question: what datatype for xval, xclear, xload?

assign  out = addout;

myreg #(W) xreg(x, xload, xclear, clk, xval); //xreg
myreg #(W) yreg(y, yload, yclear, clk, yval); //yreg
myreg #(2*W) outreg(addout, outload, outclear, clk, outval); //outreg
myreg #(W) ireg(i, iload, iclear, clk, ival); //ireg

adder #(W) add1 (xval, outval, addout); //out = out+x
increment inc1 (ival, i);  //i = i+1
comparator #(W) comp1 (ival, yval, done); //i+1 >= y-1

myfsm fsm1 ( clk, reset, done, xload, xclear, yload, yclear, iload, iclear,
outload, outclear); //controller

endmodule
```

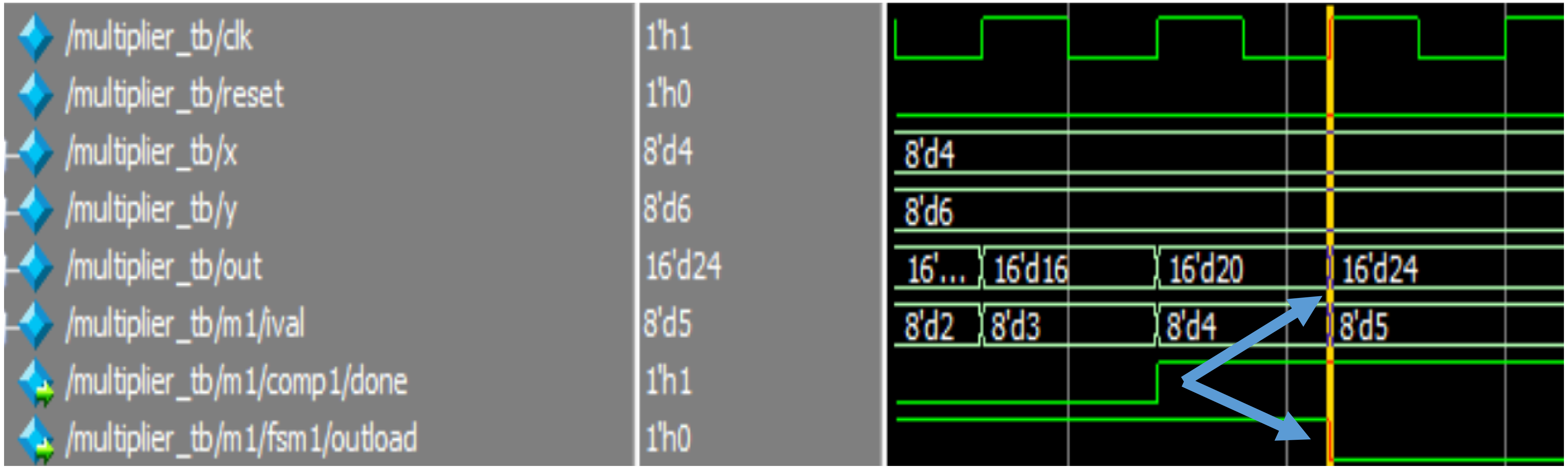
FSM Design (In-Class Exercise)

```
module myfsm (input clk, input reset, input done, output reg xload, output….);

reg [2:0] cs, ns;

always @ (posedge clk, posedge reset) begin
   //SEQUENTIAL PORTION: YOUR CODE HERE
end

always @(*) begin
  ns = cs;
  xload = 0; yload = 0;  iload =  0;  outload = 0;
  xclear = 0; yclear = 0; iclear = 0; outclear = 0;
  case (cs)
     2'b00 :  begin
     // YOUR CODE HERE
     end
     2'b01: begin
     //YOUR CODE HERE
     end
     2'b10 : begin
     // YOUR CODE HERE
     end
     default :
  endcase
end
endmodule
```

Solution

```
case (cs)
     //START
     2'b00 :  begin
         xload = 1; yload = 1;  iclear = 1; outclear = 1;
         ns = 2'b01;
     end
     //working
     2'b01: begin
         xload = 0; yload = 0; iload = 1; outload = 1;
         if (done)
             ns = 2'b10;
     end
     //finished
     2'b10 : begin
         iload = 0; outload = 0;
     end

     default :  // put in don't cares
```

Desired Simulation Output



/multiplier_tb/clk 1'h1
/multiplier_tb/reset 1'h0
/multiplier_tb/x 8'd4
/multiplier_tb/y 8'd6
/multiplier_tb/out 16'd24
/multiplier_tb/m1/ival 8'd5
/multiplier_tb/m1/comp1/done 1'h1
/multiplier_tb/m1/fsm1/outload 1'h0
8'd4
8'd6
16'... 16'd16 16'd20 16'd24
8'd2 8'd3 8'd4 8'd5

One clock cycle difference. Think of done as “almost done”

Assignment 1: Review

Problem 1

- Huffman Decoder. What is a Huffman code?

A naivve code would use the same number of bits to represent each symbol. For instance, if we have 4 symbols {A,B,C,D}, a naiive code would use 2 bits for each.

{A = 00, B = 01, C = 10, D = 11}.

But what if some symbols are more frequent than others?

AAABAAAADACAAABBAAA

Idea: Use fewer bits for the most frequent symbols.

Example Codebook

Codeword	Original Symbol
0	A
10	B
110	C
111	D

Let'say symbol A occurs with 70% likelihood and all other symbols occur with 10% likelihood. What is the compression ratio compared to a Naiive encoding?

Prefix Code. A Huffman code is an example of a Prefix code. That is, no codeword is the prefix of another codeword.

Example, let's look at codeword 110 (C). Prefixes of this codeword:

- 1 (not a codeword)
- 11 (not a codeword)
- 110 (Codeword = C)

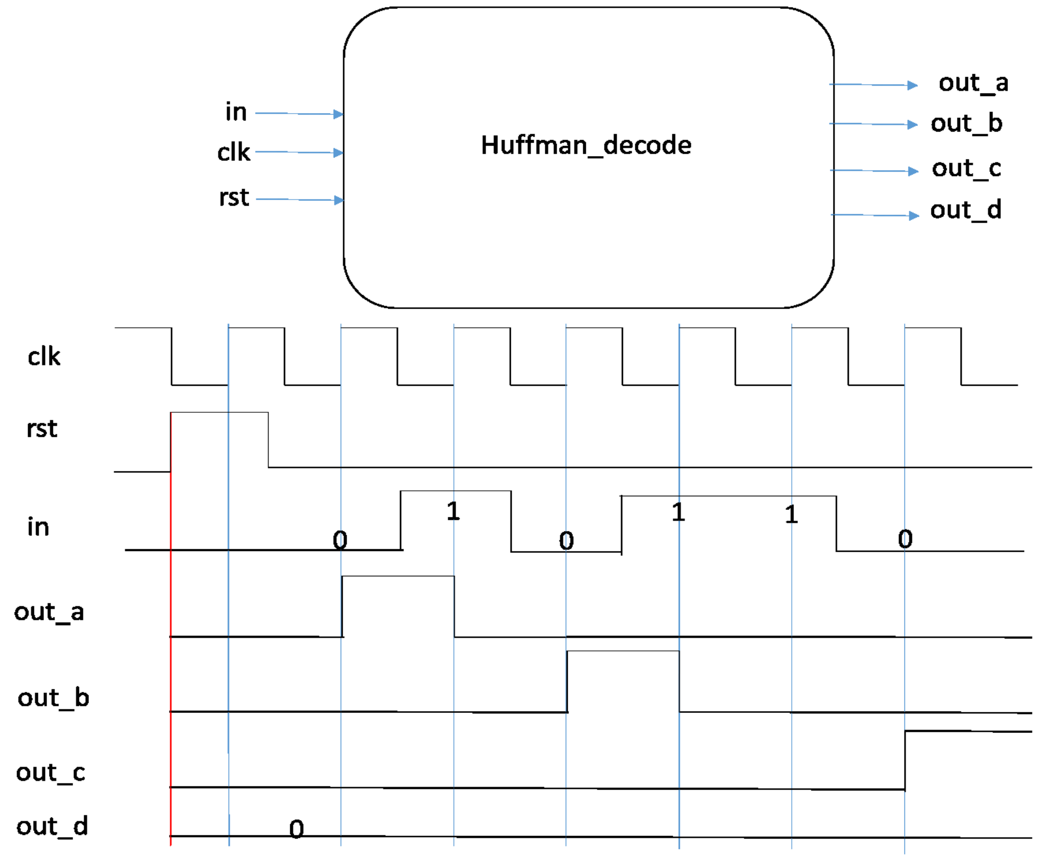
[redacted]

This means symbols can be decoded as bits from the compressed stream arrive! No need to know what the future bits will be.

- Consider what would happen if the codeword for A were 1!

0	0	1	1	0	1	0	1	1	1	0
A	A	-	-	C	-	B	-	-	D	A

Understanding Specifications



Question: is this a Mealy or a Moore machine?
(when do the outputs change?)

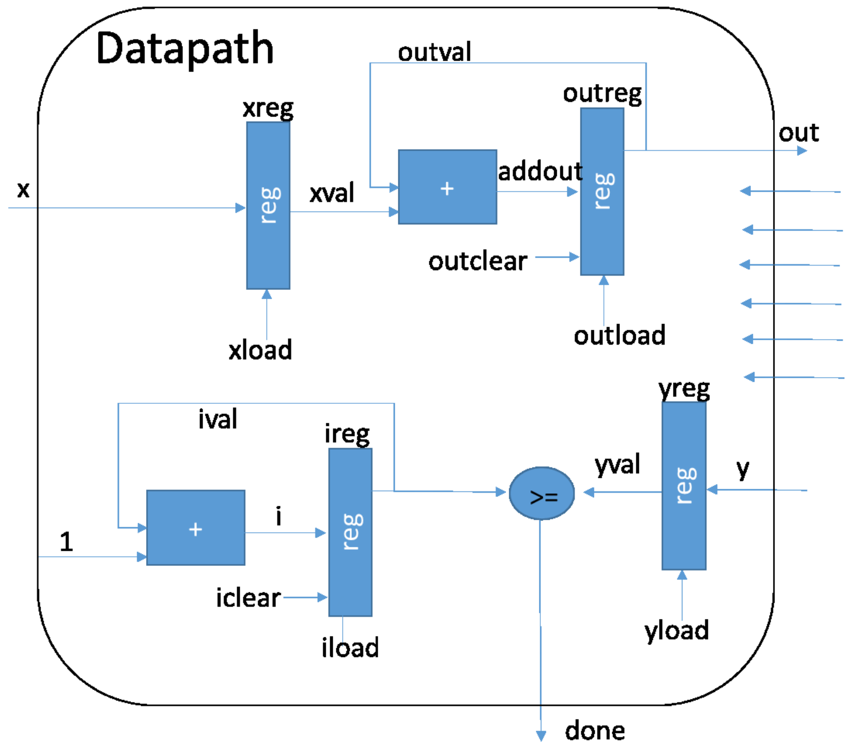
Problem 2: Divider

```
out = 0;
   for (i=0; i<y; i=i+1) begin
      out = out + x;
   end
```

Multiplier Code

```
quotient = 0;
while (a>=b) {
   a = a – b;
   quotient = quotient + 1;
}
remainder = a;
```

Divider Code



What will the datapath for the divider look like?

Lecture 3: Synthesis

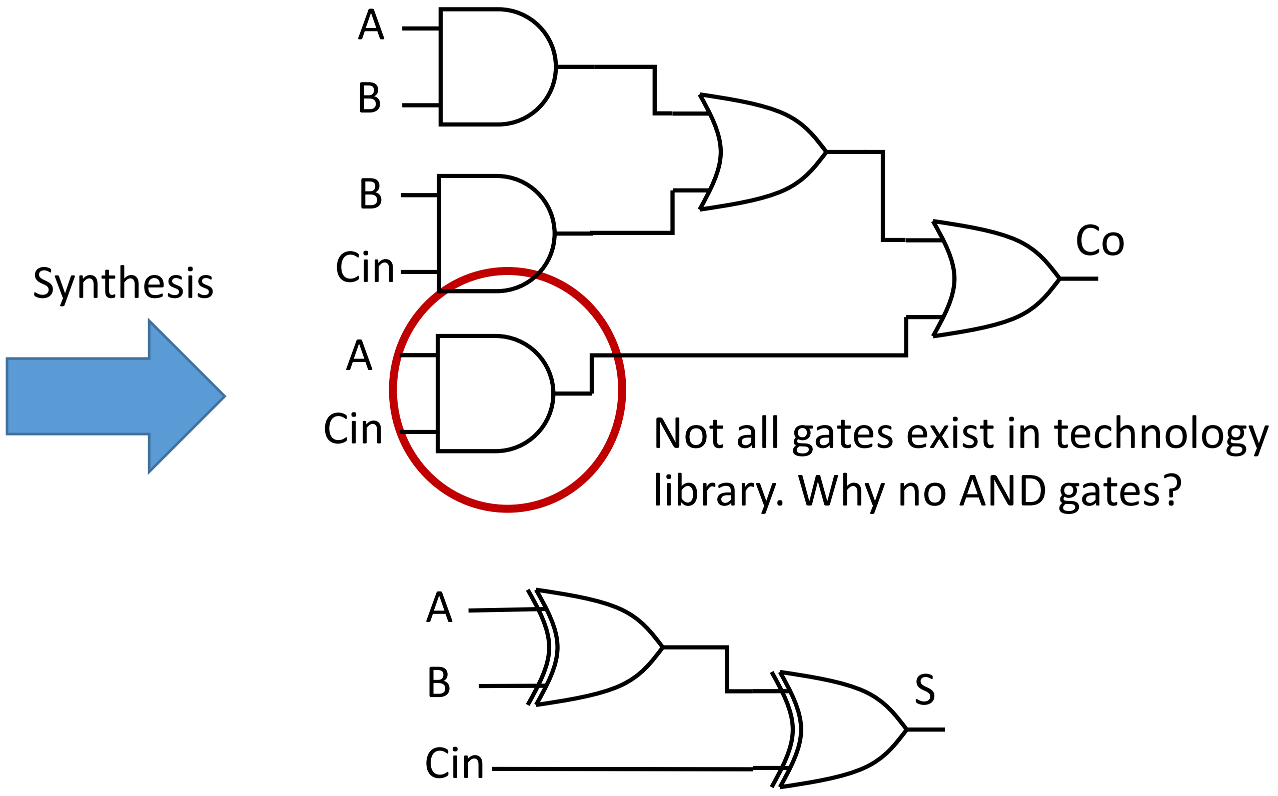
EL GY 6463: Advanced Hardware Design

Instructor: Siddharth Garg

What is Synthesis?

- Converts a Verilog description of logic functionality (combinational or FSM) to a netlist of logic Boolean gates + sequential elements like latches and flip-flops
  - The types of logic gates and latches/flip-flops is specified in a technology library

```
module fulladder ( input a,
                    input b,
                   input cin,
                   output sum,
                   output cout);

assign {cout,sum} =  a + b + cin;

endmodule
```



Not all gates exist in technology library. Why no AND gates?

Cadence RC Synthesis Tool Demo

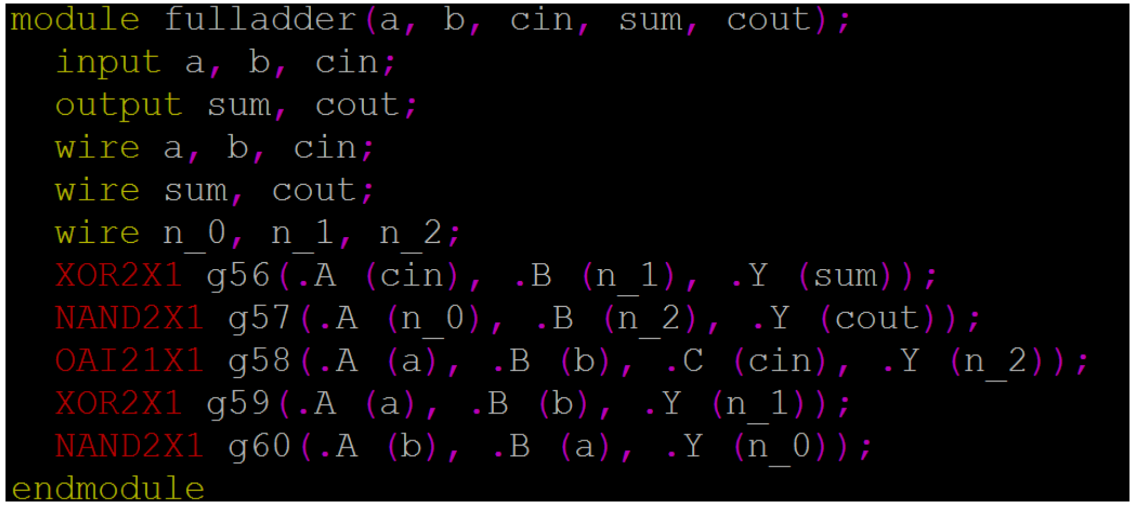
- You all have accounts on gauss.poly.edu
  - Please try connecting and logging in. If you cannot login, please send ASAP an email to Julio Rivera (jr847@nyu.edu) and cc me.

  - FAQ: can I get Cadence RC on my desktop? No. commercial tool, thousands of dollars per license

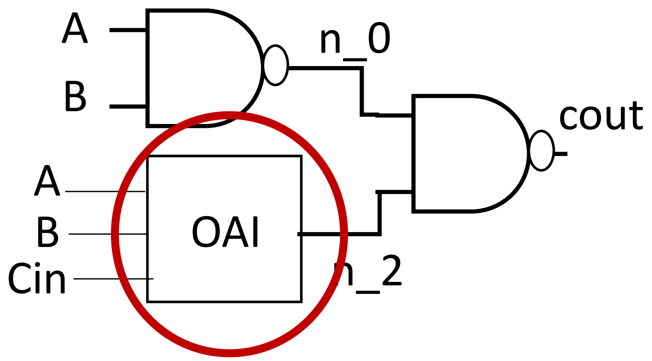
Full Adder Synthesis Output

```
module fulladder ( input a,
                   input b,
                   input cin,
                   output sum,
                   output cout);

assign {cout,sum} =  a + b + cin;

endmodule
```

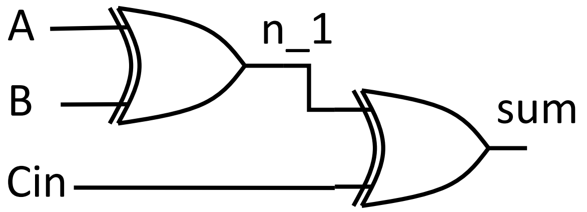
Synthesis



```
module fulladder(a, b, cin, sum, cout);
  input a, b, cin;
  output sum, cout;
  wire a, b, cin;
  wire sum, cout;
  wire n_0, n_1, n_2;
  XOR2X1 g56(.A (cin), .B (n_1), .Y (sum));
  NAND2X1 g57(.A (n_0), .B (n_2), .Y (cout));
  OAI21X1 g58(.A (a), .B (b), .C (cin), .Y (n_2));
  XOR2X1 g59(.A (a), .B (b), .Y (n_1));
  NAND2X1 g60(.A (b), .B (a), .Y (n_0));
endmodule
```



What is an OAI gate?

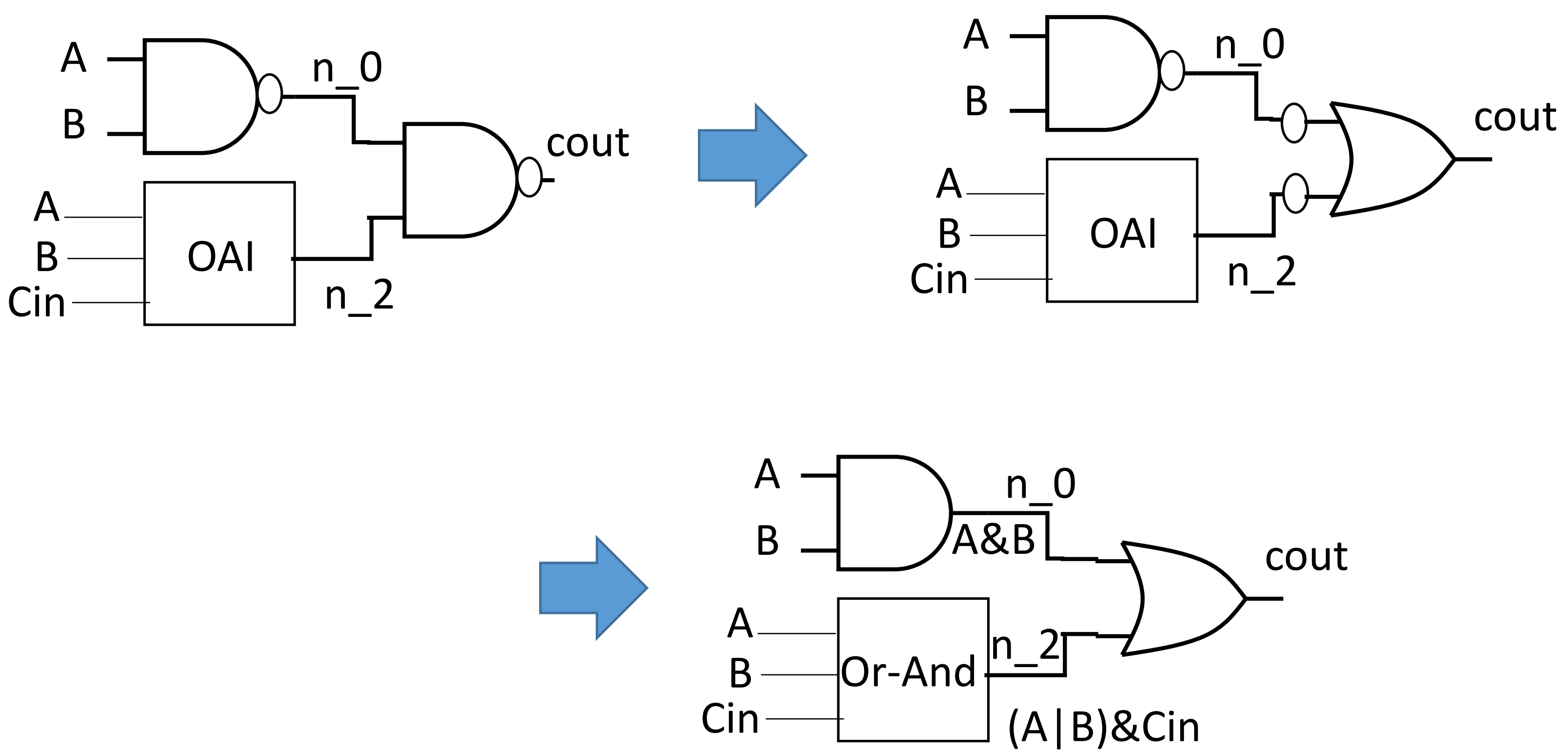


OAI(x,y,z) = ((x|y) & z)’

In class exercise: prove that the synthesized netlist for cout gives the correct output. Recell that:

cout = a&b | b&c | c&a

Solution to In-class Exercise



Question: how did the synthesis tool choose this specific solution?

- What metrics is the tool trying to optimize for?

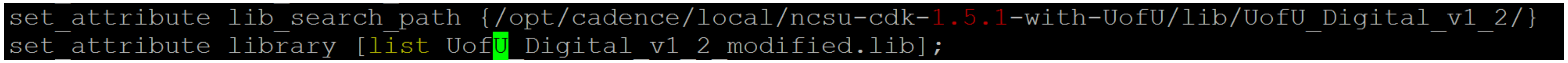
Optimization Metrics

- Three primary metrics that a synthesis tool cares about
  - Area: what is the total sum of areas of each gate
  - Delay: what is the delay from the input to the output (more on this very shortly)
  - Power: what is the circuit power consumption (full lecture dedicated to power consumption later in the term)

- Let’s look at area first. How does the synthesis tool know the area of each gate?
  - Look in the technology library

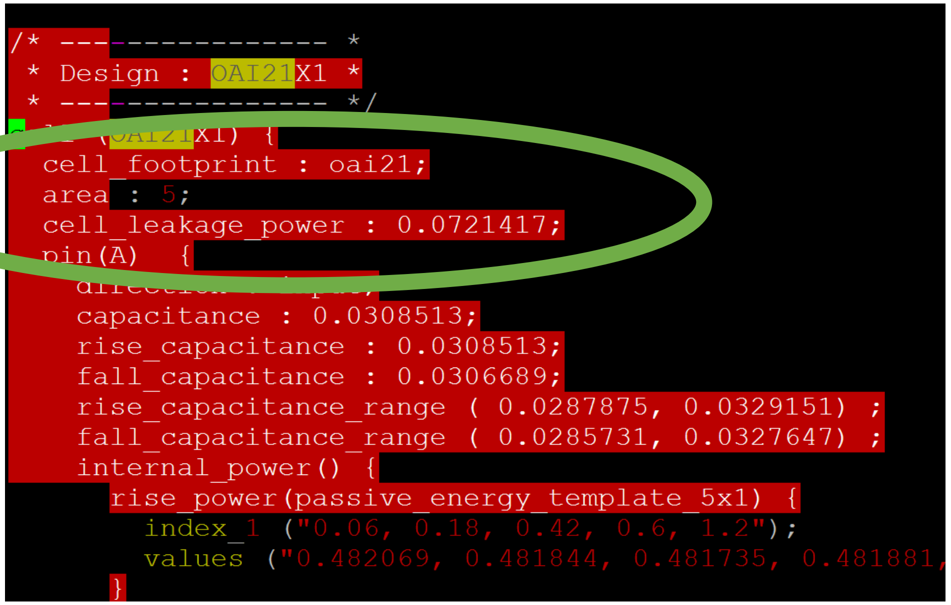
Technology library

- RC needs us to tell it where the technology library exists
  - set_attribute lib_search_path
  - set_attribute library
  - In “run.tcl”

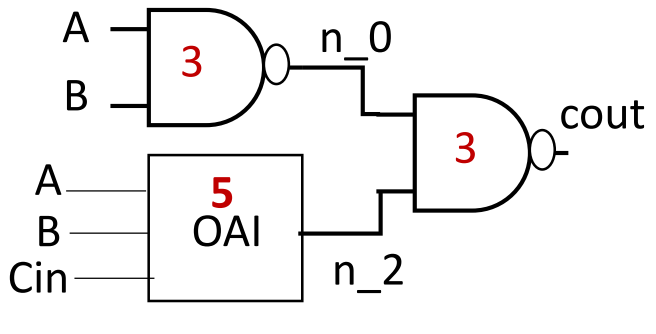


```
set_attribute lib_search_path {/opt/cadence/local/ncsu-cdk-1.5.1-with-UofU/lib/UofU_Digital_v1_2/}
set_attribute library [list UofU_Digital_v1_2_modified.lib];
```

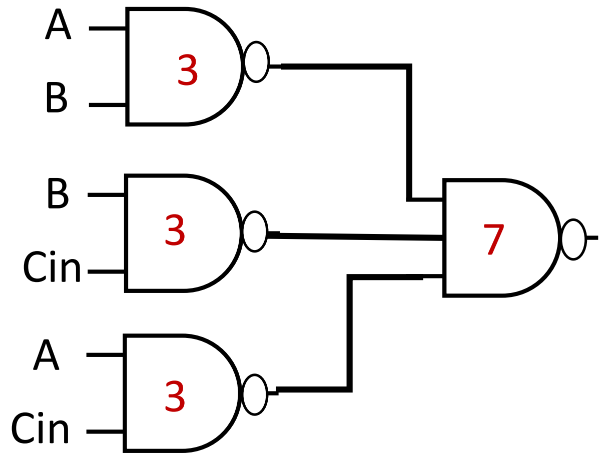
- Let’s take a look at UofU_Digital_v1_2_modified.lib



```
/* ----------------- *
 * Design : OAI21X1 *
 * ----------------- */
cell (OAI21X1) {
  cell_footprint : oai21;
  area : 5;
  cell_leakage_power : 0.0721417;
  pin(A)  {
    direction : input;
    capacitance : 0.0308513;
    rise_capacitance : 0.0308513;
    fall_capacitance : 0.0306689;
    rise_capacitance_range ( 0.0287875, 0.0329151) ;
    fall_capacitance_range ( 0.0285731, 0.0327647) ;
    internal_power() {
      rise_power(passive_energy_template_5x1) {
        index_1 ("0.06, 0.18, 0.42, 0.6, 1.2");
        values ("0.482069, 0.481844, 0.481735, 0.481881,
      }
```



11



16

How Did the Synthesis Tool Get to This Solution?

- *Many, many* steps from a behavioral Verilog description to a structural description mapped to a given technology
  - Technology dependent logic minimization (this step does not depend on the technology library)

  - Technology mapping (only gates from the technology library allowed)

  - Further technology dependent optimizations like gate sizing (NANDx1, NANDx2…)

Two-level Logic Minimization

- Any Boolean function can be expressed in "*sum of products*" form
  - F = (a&b) | (b'&c) | (a'&b'&c') (3 product terms, 7 literals)

    Product term   Literal

  - Goal of two level logic minimizations: determine an SOP expression for F which has the *fewest product terms* OR *fewest literals*
    - Why should minimizing the number of product terms or literals matter?

Two-Level Minimization: Prime Implicants

- Start with all entries in the Truth table that output a ‘1’ (ON-Terms)

W	X	Y	Z	Label
0	0	0	0	M0
0	1	0	1	M5
0	1	1	1	M7
1	0	0	0	M8
1	0	0	1	M9
1	0	1	0	M10
1	0	1	1	M11

$\Rightarrow$

W	X	Y	Z	Label
-	0	0	0	$\bigcirc$ M0, M8
0	1	-	1	$\bigcirc$ M5, M7
1	0	0	-	M8, M9
1	0	-	0	M8, M10
1	0	-	1	M9, M11
1	0	1	-	M10,M11

All pairs of terms that differ in a single entry

W	X	Y	Z	Label
1	0	-	-	$\bigcirc$ M8, M9, M10, M11

All sets of 4 terms that differ in only two entires

Prime implicants are entries in these tables that are not subsets of other entries

Example modified from MIT OCW

Two-Level Minimization: Covering Problem

- For all prime implicants, list the min terms that they cover

PI	M0	M5	M7	M8	M9	M10	M11
{M0,M8}	1			1			
{M5, M7}		1	1				
{M8, M9, M10, M11}				1	1	1	1

- Every time you pick a PI, it “covers” all its minterms
  - Our goal: pick the smallest number of PIs so all minterms are covered.
  - How many in this example?

Solution

- In this example, all PIs need to be picked to cover all minterms

PI	M0	M5	M7	M8	M9	M10	M11
{M0,M8}	1			1			
{M5, M7}		1	1				
{M8, M9, M10, M11}				1	1	1	1

- F = (x’&y’&z’) \| (w’&x&z) \| (w&x’)
  - How many two input AND/OR gates assuming true and complemented forms of inputs?

Set Cover

- But in general, not all min terms might be required for the cover

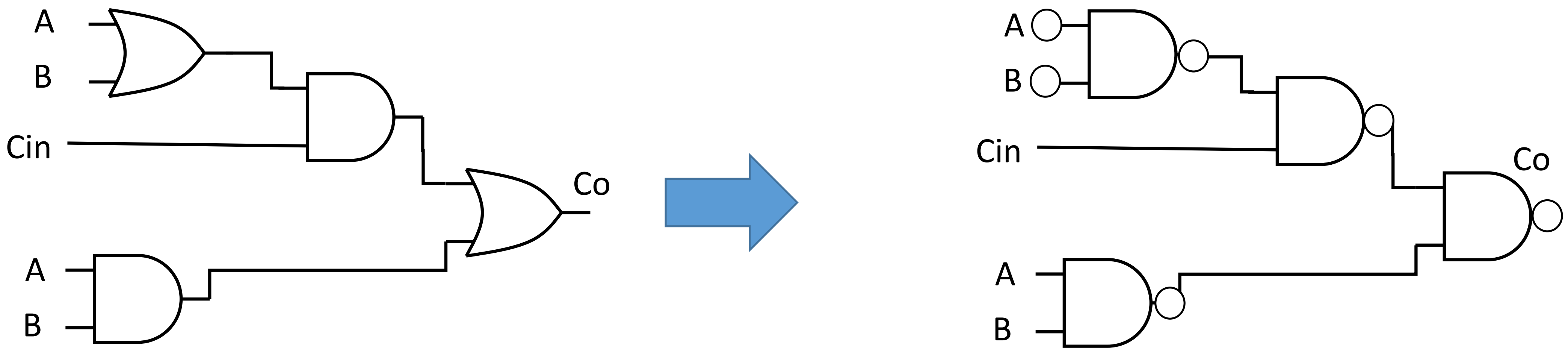
PI	M0	M5	M7	M8	M9	M10	M11
{M0,M8}	1			1			
{M5, M7}		1	1				
➡ {M8, M9, M10, M11}				1	1	1	1
➡ {M0, M5, M7, M8}	1	1	1	1			

- The set-cover problem in general is NP-complete
  - No poly-time solution unless P=NP
  - Cadence RC implements several heuristic solutions to solve such problems

Technology Mapping

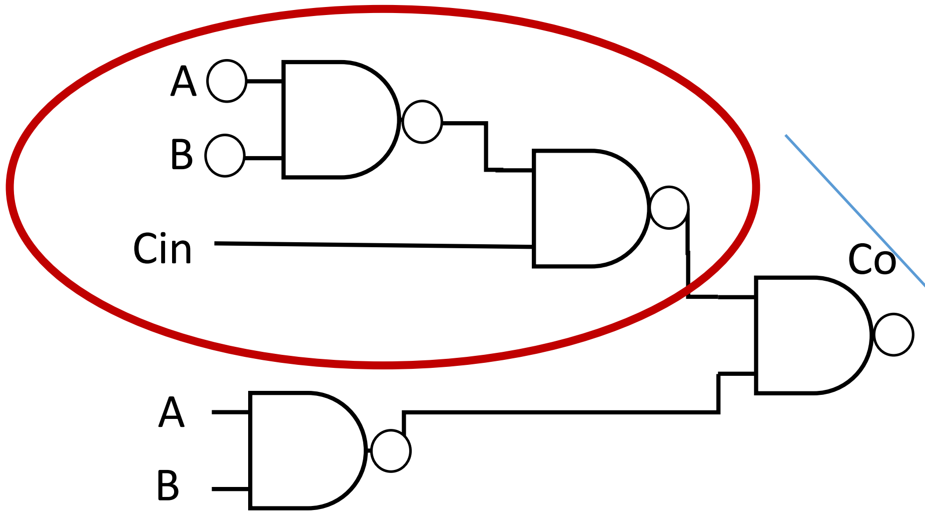
- OK, so you now have a minimized SOP formula for your Boolean function.
  - How does that translate to gates in the technology library?

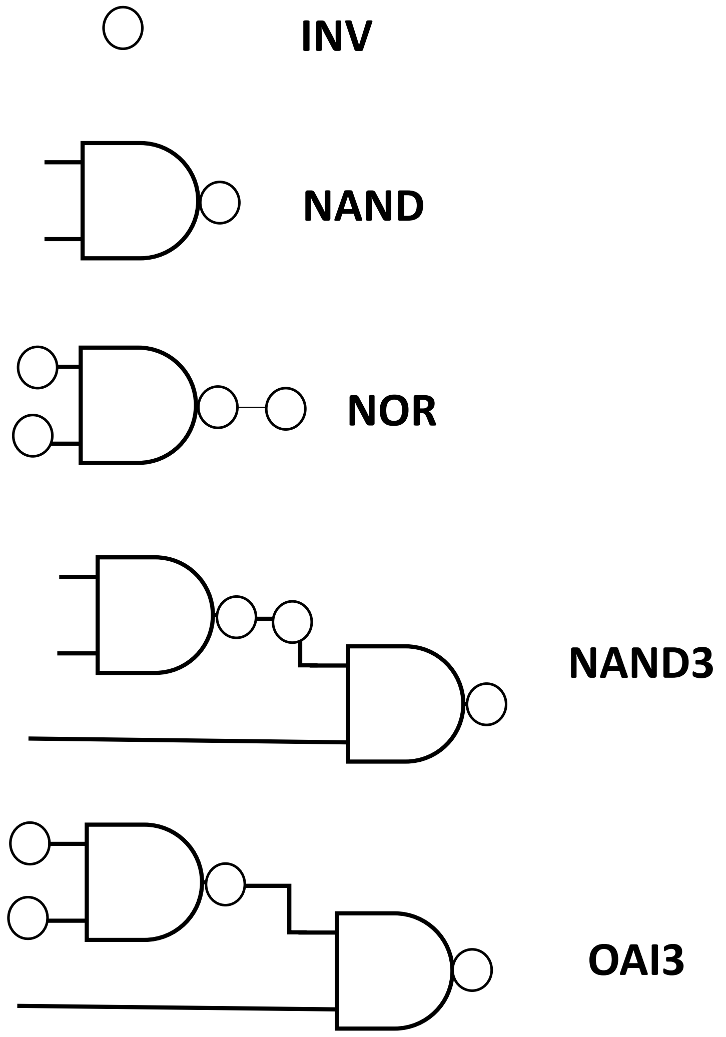
- **STEP 1:** represent your Boolean formula as a circuit consisting of "elemental" gates. Typically NAND and INVs
  - F = (a&b) | (b&c) | (c&a) = ((a | b) &c) | (c &a)



Technology Mapping

STEP 2: represent every gate in your tech library with "elemental" gates. Typically NAND and INVs

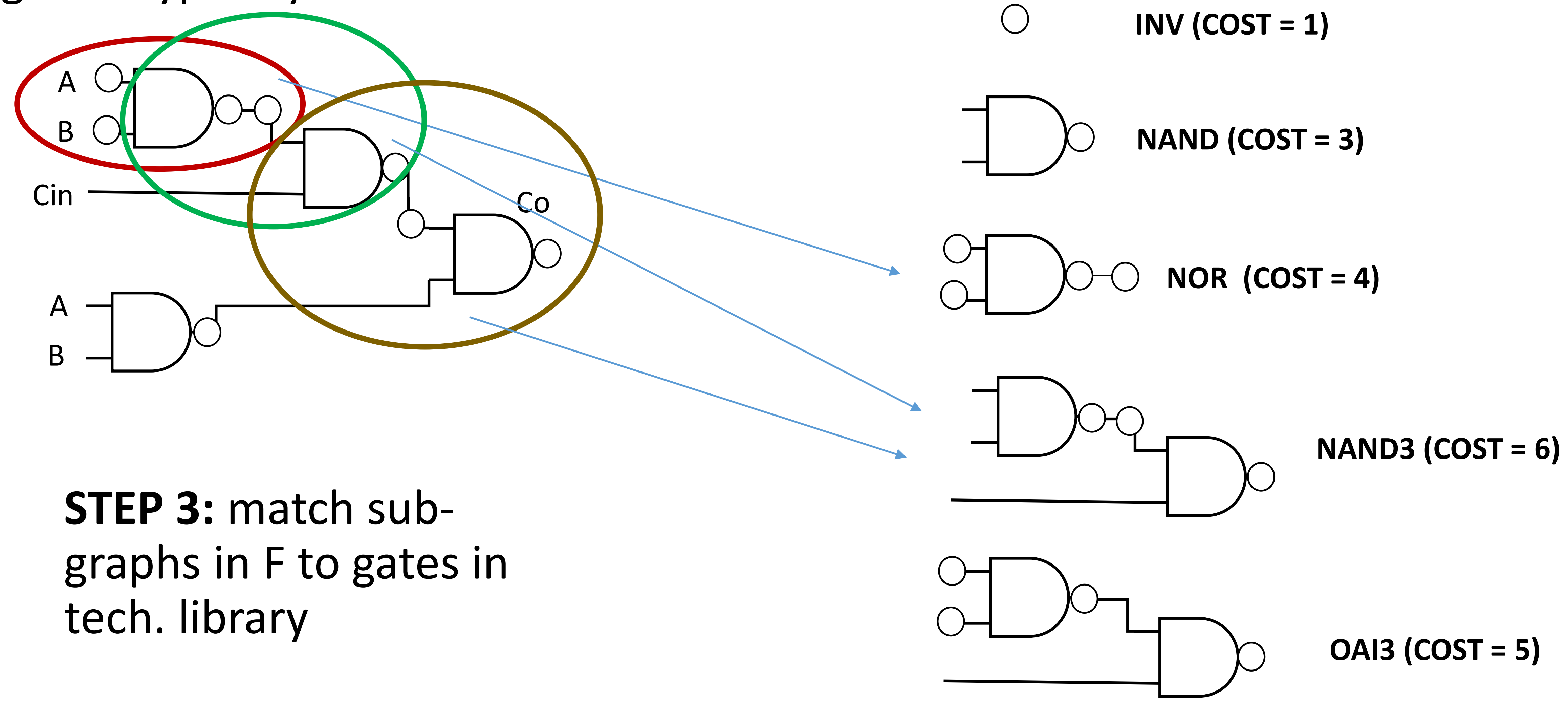




STEP 3: match sub-graphs in F to gates in tech. library

Technology Mapping

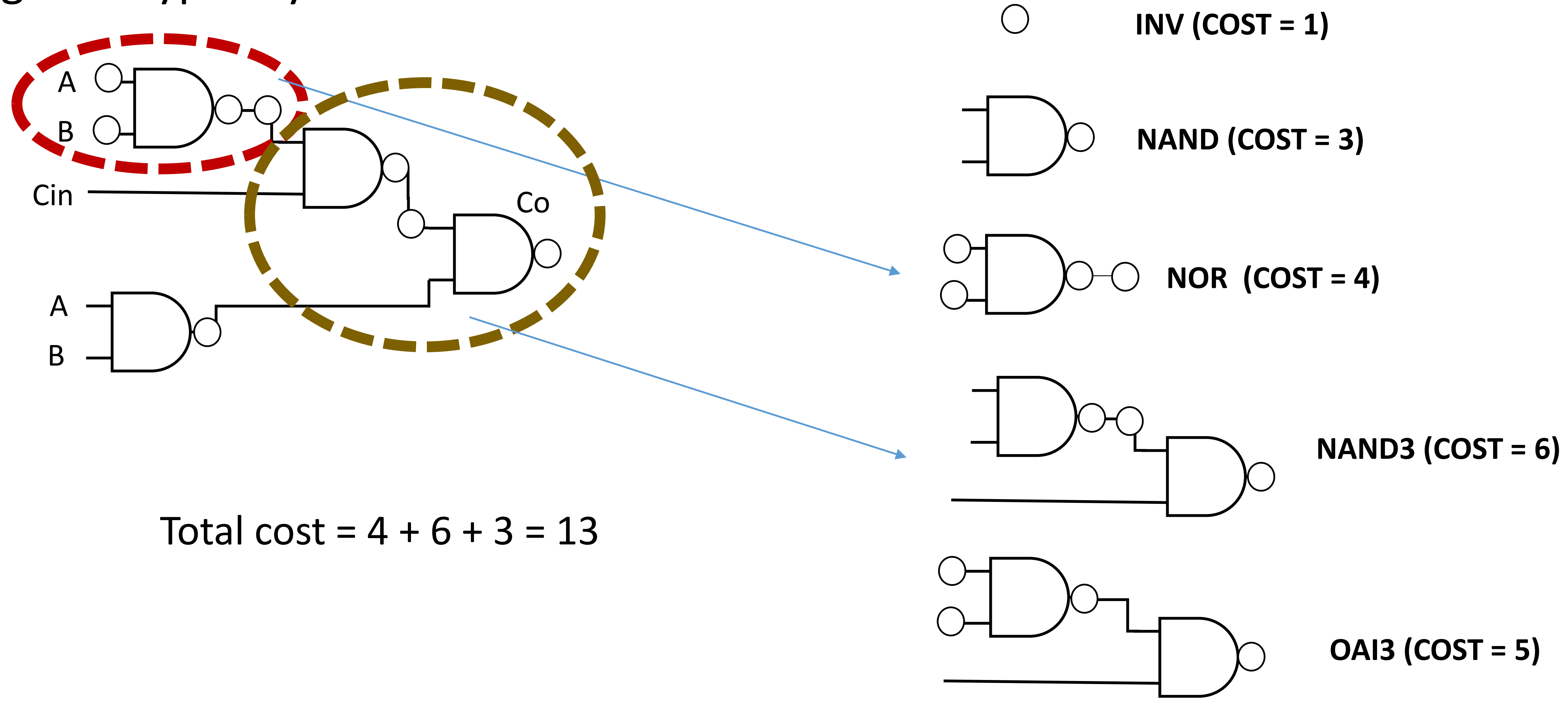
STEP 2: represent every gate in your tech library with "elemental" gates. Typically NAND and INVs



STEP 3: match sub-graphs in F to gates in tech. library

Technology Mapping

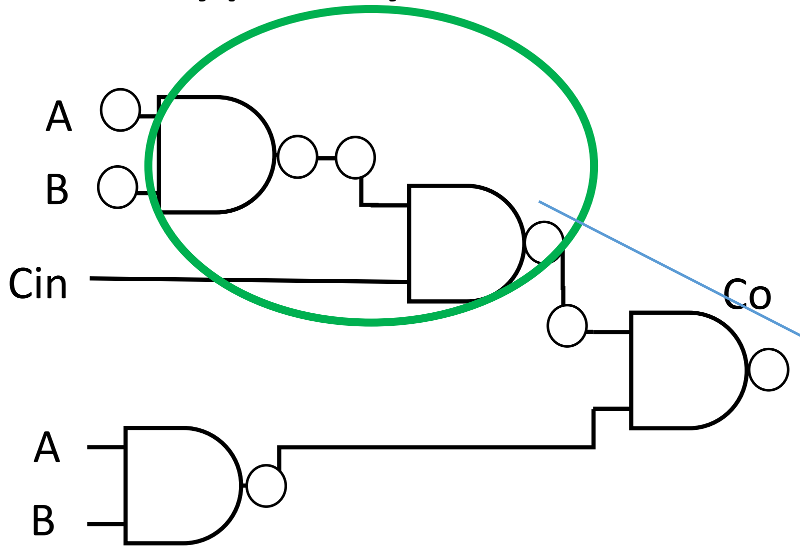
STEP 2: represent every gate in your tech library with “elemental” gates. Typically NAND and INVs



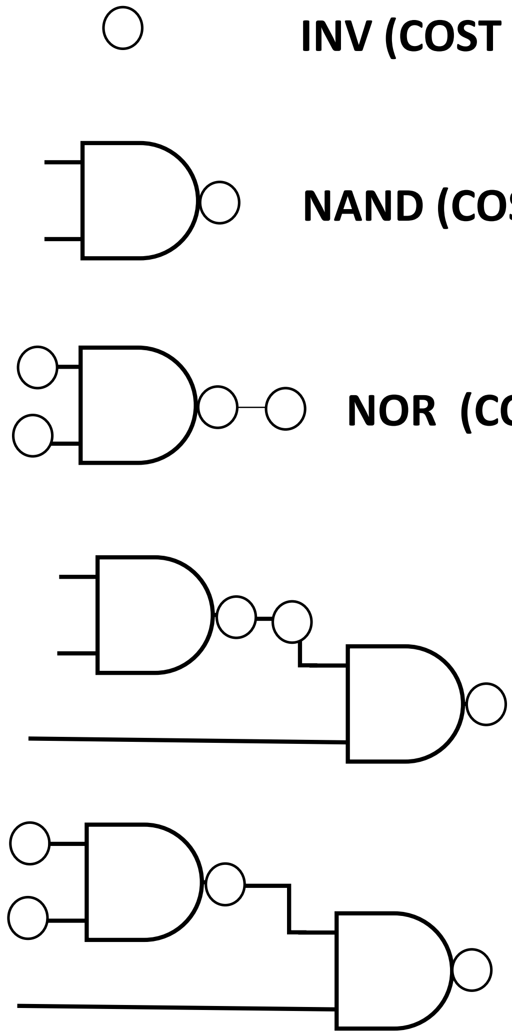
Total cost = 4 + 6 + 3 = 13

Technology Mapping

STEP 2: represent every gate in your tech library with "elemental" gates. Typically NAND and INVs



Total cost = 7 + 3 + 3 + 3*1 = 16



INV (COST = 1)

NAND (COST = 3)

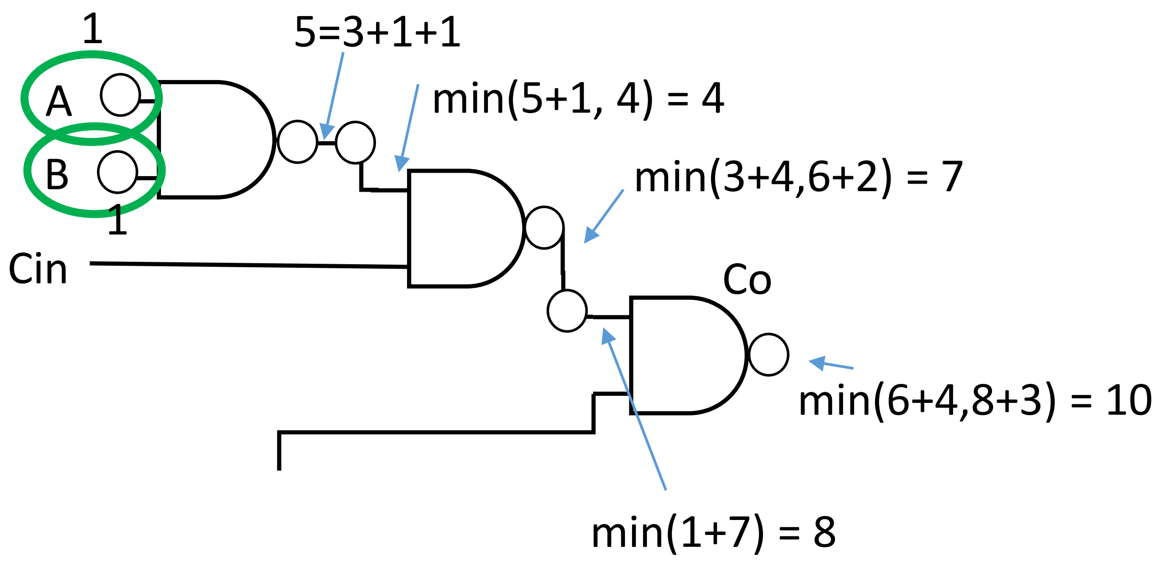
NOR (COST = 4)

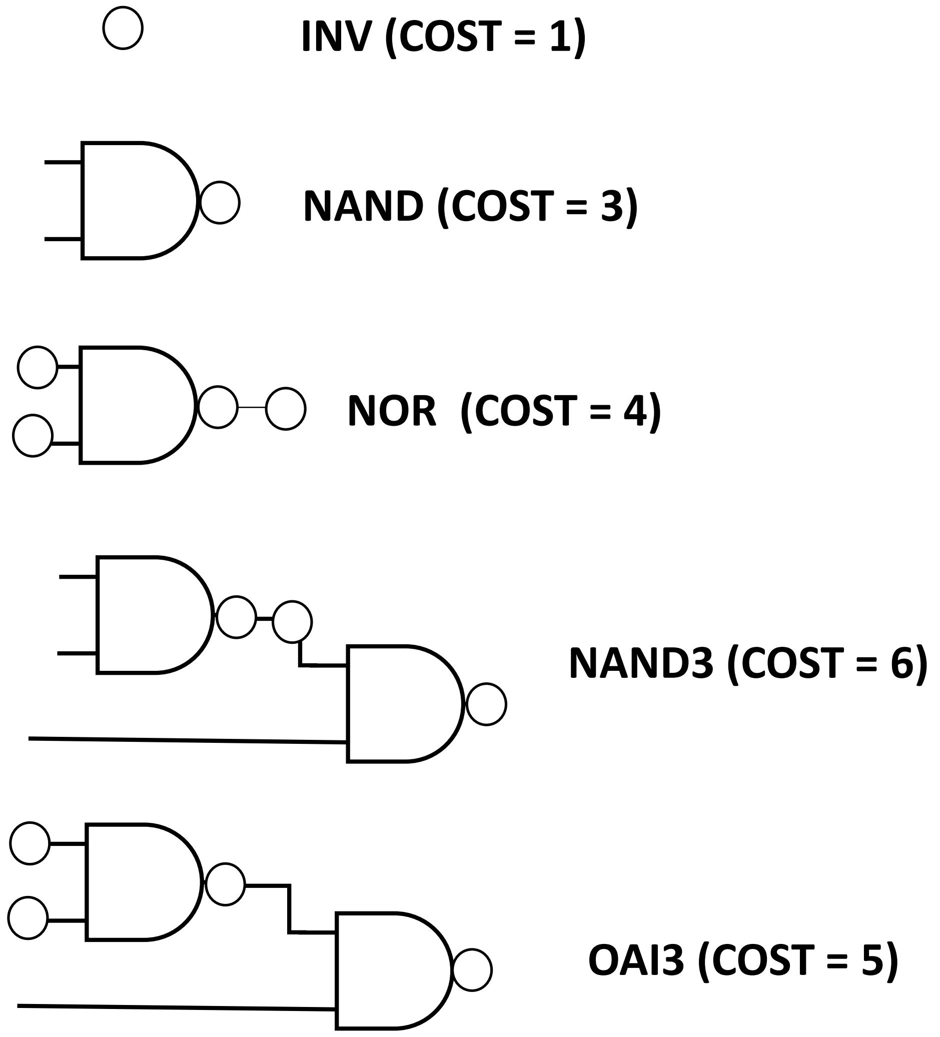
NAND3 (COST = 6)

OAI3 (COST = 5)

Dynamic Programming

Is there a systematic way to find optimal mapping? Yes! (Dynamic programming)

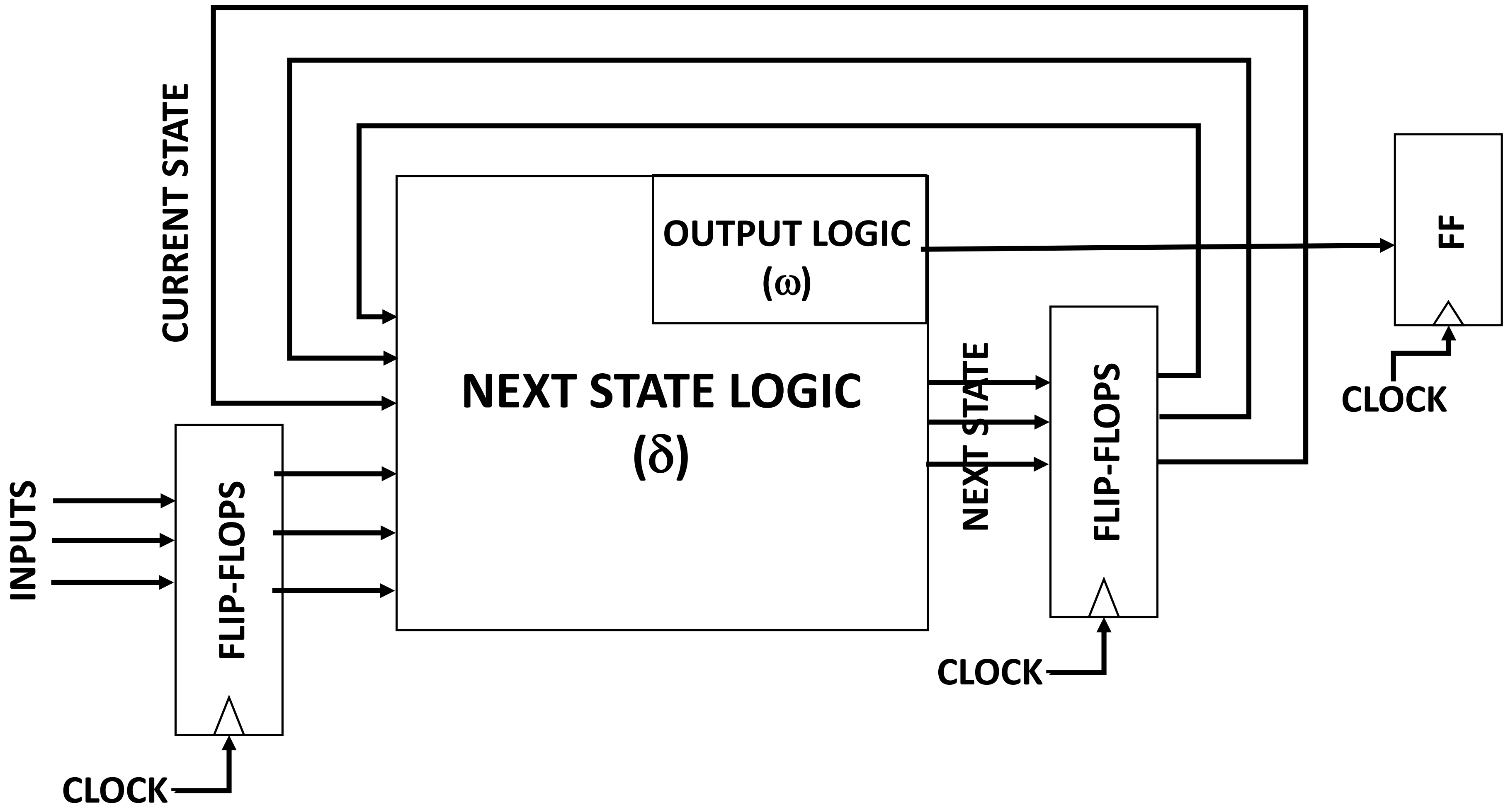




Optimization Metrics

- Three primary metrics that a synthesis tool cares about
  - Area: what is the total sum of areas of each gate
  - Delay: what is the delay from the input to the output (more on this very shortly)
  - Power: what is the circuit power consumption (full lecture dedicated to power consumption later in the term)

- Let’s look at delay now
  - But first a word on FSM timing

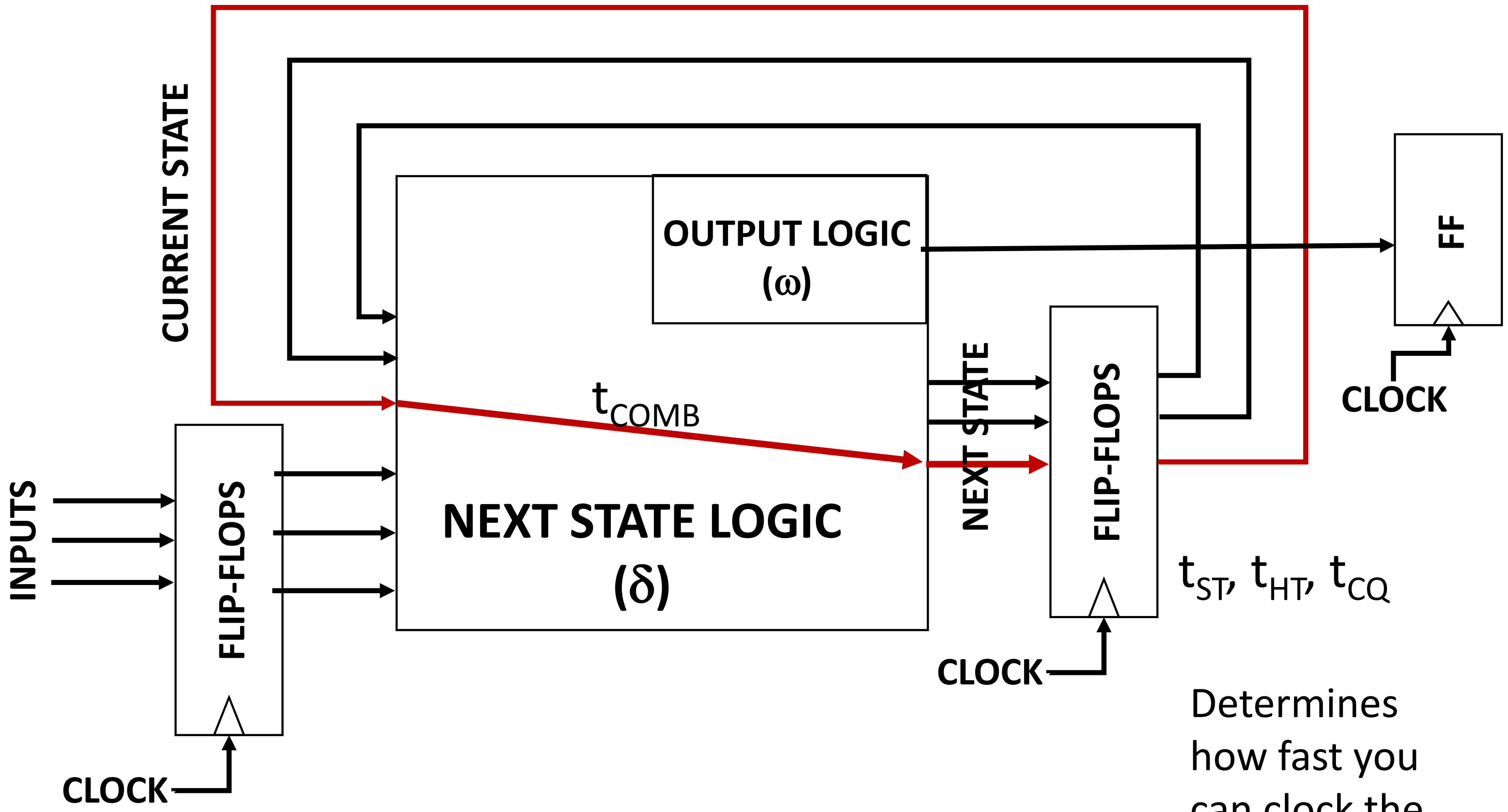


Flip-flops have three relevant timing parameters:

--- Setup time (ST): inputs of FF should be ready ST ns before positive edge

--- Hold time (HT): inputs should remain stable for HT ns after pos edge

--- Clock-q (CQ) propagation delay: outputs ready CQ ns after pos edge



Determines how fast you can clock the circuit

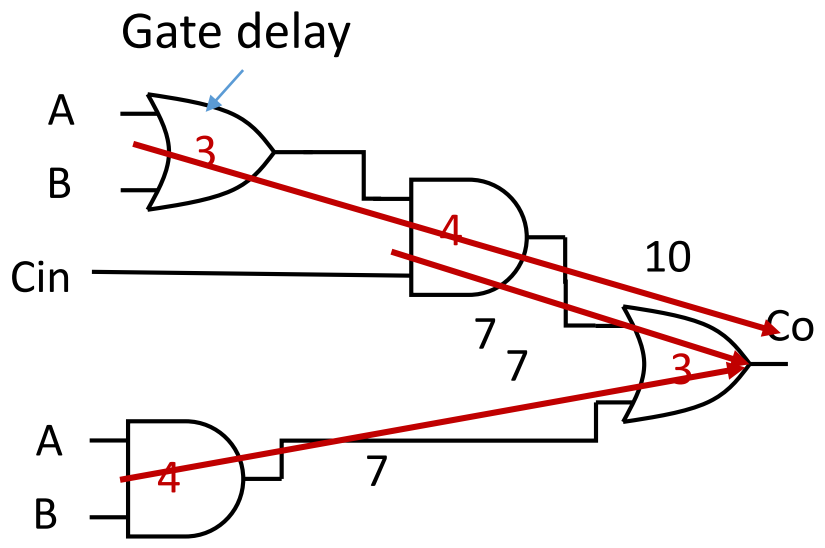
Setup time constraint: $t_{CQ} + t_{COMB} + t_{ST} < T_{CLK}$

Hold time constraint: $t_{CQ} + t_{COMB} > T_{HT}$

Automatically handled by most synthesis tools using buffers

Determining T_{COMB}

- T_{COMB} is the worst-case (largest) delay from an input to an output



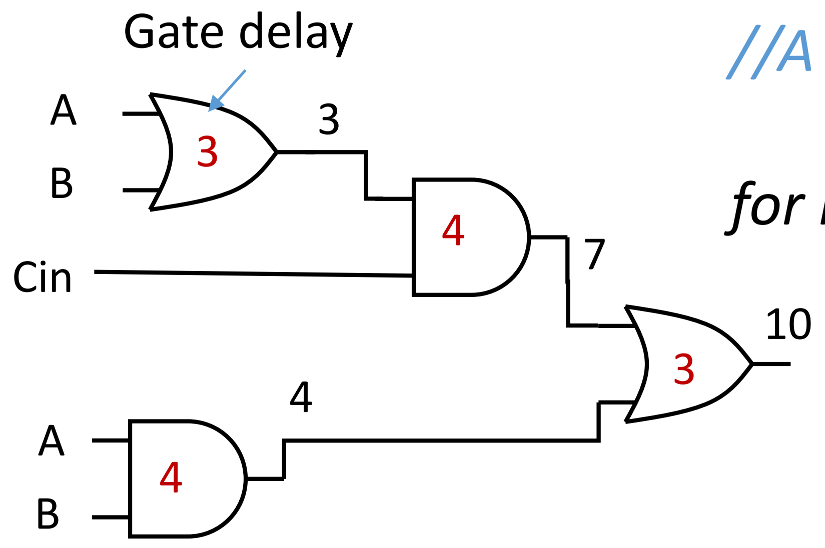
How many input-output paths are there in this netlist?

Determine T_{COMB}

In general, how many paths can there be for a block of combinational logc with N gates?

Static Timing Analysis

- Polynomial time algorithm to determine T_{COMB}

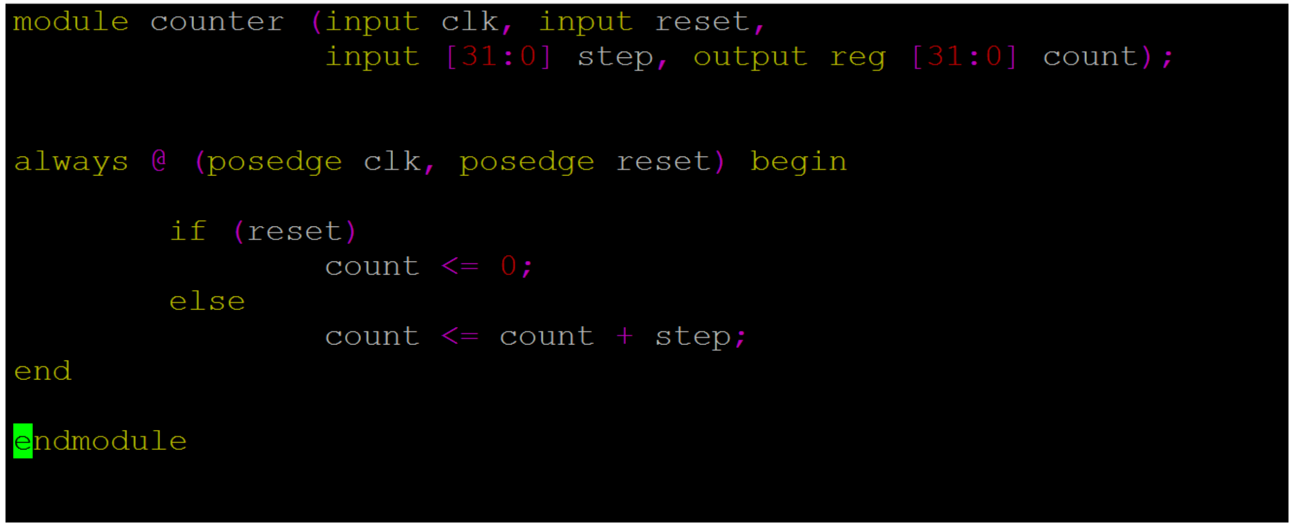


```
levelize_gates();
//A gate is at level i if all its predecessors are at level i-1

for i=1:1:numGates //in levelized order
    $AT(gate_i) = delay(gate_i) + \max_{j \in inputs(i)} AT(gate_j)$
```

Latest time at which the output of gate i can change

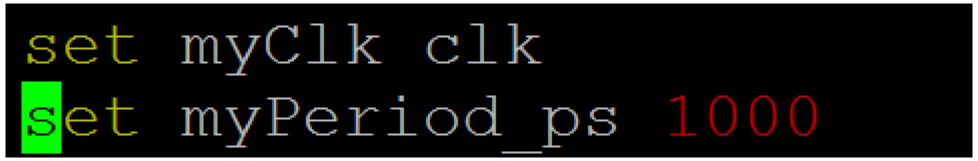
FSM Synthesis

- Simple up-counter



```
module counter (input clk, input reset,
                input [31:0] step, output reg [31:0] count);


always @ (posedge clk, posedge reset) begin

        if (reset)
                count <= 0;
        else
                count <= count + step;
end

endmodule
```

- run.tcl script specifies a clock period constraint to RC



```
set myClk clk
set myPeriod_ps 1000
```

In Class Demo

Target Clock Period	Slack (positive/negative)	Area
10000 ps		
1000 ps		