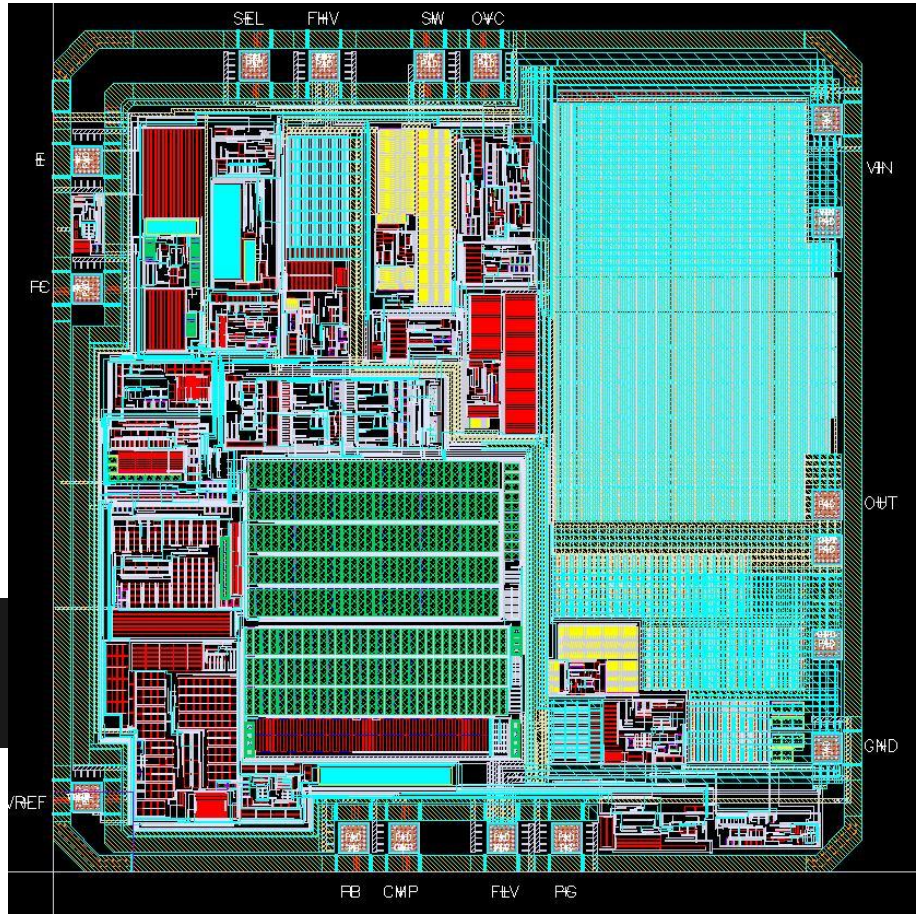
CHAPTER 14
SYNTHESIS OPTIMIZATION

ECE4514

DIGITAL DESIGN 2



Topics

- Constraints
- Trade-offs with speed versus area.
- Resource sharing for area optimization.
- Pipelining, retiming, and register balancing for performance optimization.
  - The effect of reset on register balancing
  - Handling resynchronization registers
- Optimizing FSMs.
- Handling black boxes.
- Physical synthesis for performance.

Specifying Constraints

- Timing:
  - Specify clock rates - register to register timing
  - Input/Output timing - relate I/O data to clock
  - Handle asynchronous clock domains
  - Multi-cycle paths
- Placement:
  - Explicitly place components
  - Confine logic to specific areas
- Plus lots more

Synopsys Design Constraint - SDC

```
# Constrain clock port clk with a 10-ns requirement
create_clock -period 10 [get_ports clk]
# Automatically apply a generate clock on the output of phase-locked loops (PLLs)
# This command can be safely left in the SDC even if no PLLs exist in the design
derive_pll_clocks
# Constrain the input I/O path
set_input_delay -clock clk -max 3 [all_inputs]
set_input_delay -clock clk -min 2 [all_inputs]
# Constrain the output I/O path
set_output_delay -clock clk 2 [all_outputs] 2
```

Synopsys Design Constraint - SDC

For more detailed specification of constraints, refer to the SDC and Timequest API Reference Manual

Summary of SDC objects:
http://www.vlsi-expert.com/2011/02/synopsys-design-constraints-sdc-basics.html

Speed vs. Area Tradeoff

- Synthesis tools have knobs and switches
  - Optimize for Area
  - Optimize for Speed (Timing closure)
- In General
  - Faster circuits require more parallelism
  - Parallel structures require more area
- Speed optimizations don’t always produce faster circuits

Congestion Confounds Constraints

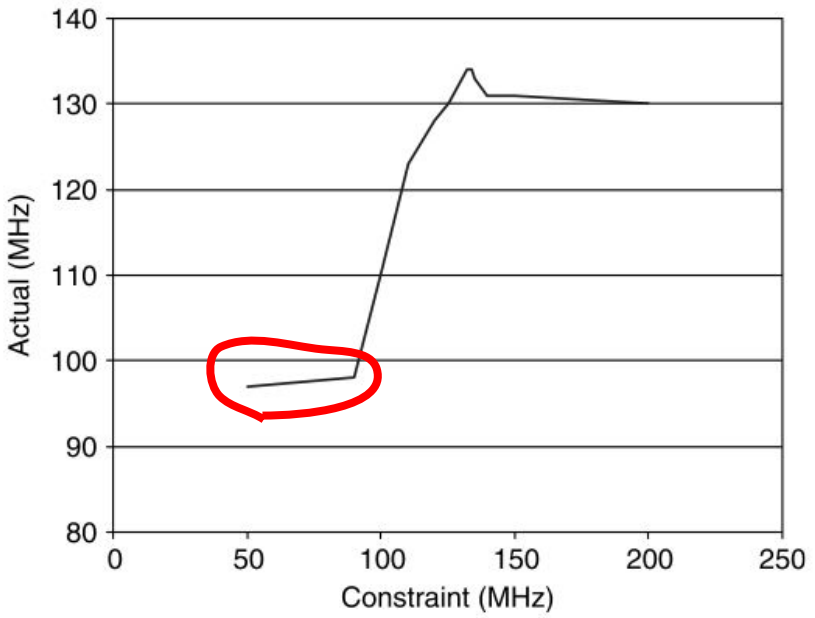
- Speed-optimized logic consumes extra resources
- There are limited logic and routing resources
- What happens when these collide?
- Congestion, routability not necessarily visible to Synthesis tool

Case Study

- Constrain critical path to small region
  - forcing congestion
- Change timing constraints
  - change how heavily tool optimizes for speed
- Clock speed of RISC MPU on FPGA

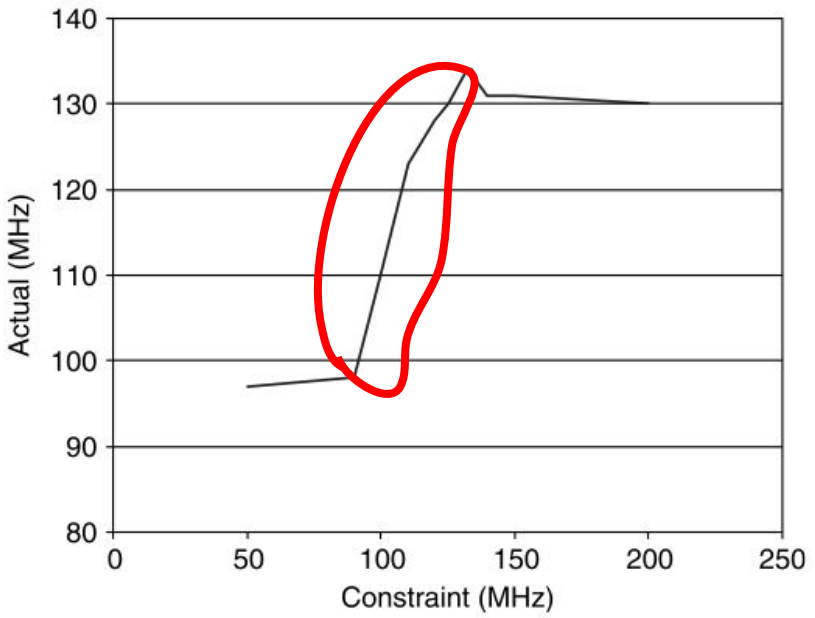
Underconstrained

- Clock speed < 95MHz
- Runs at ~95MHz without major optimization



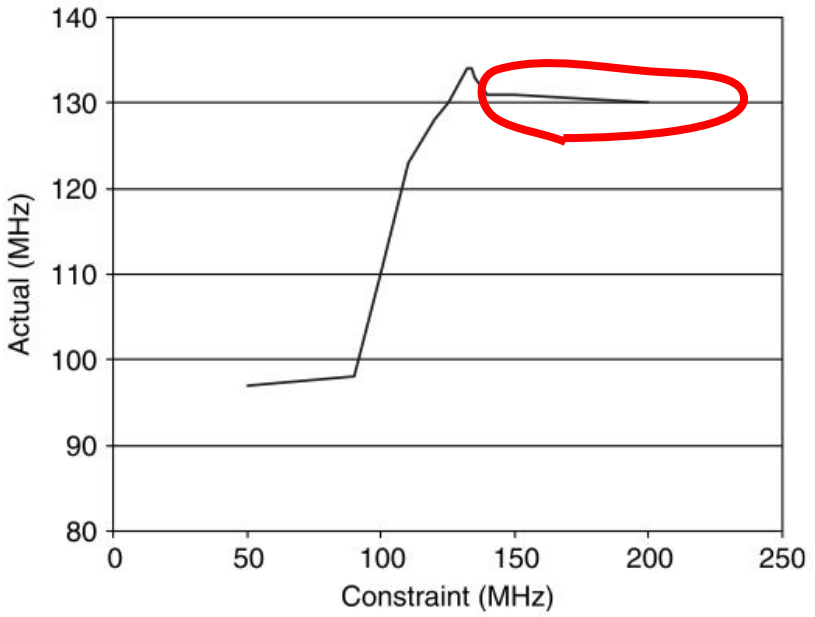
Optimization Region

- From 95-135MHz
- Constraint is met
- Optimization effort scales with constraint



Overconstrained

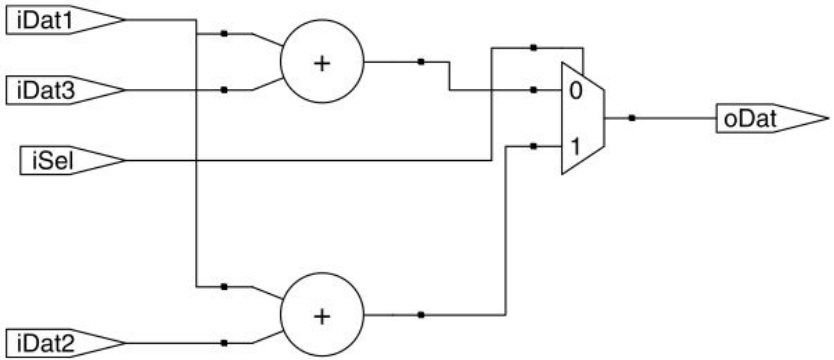
- Reached max. frequency
- The little engine that couldn’t
  - Tool gives up if it thinks timing can’t be met



Resource Sharing Option

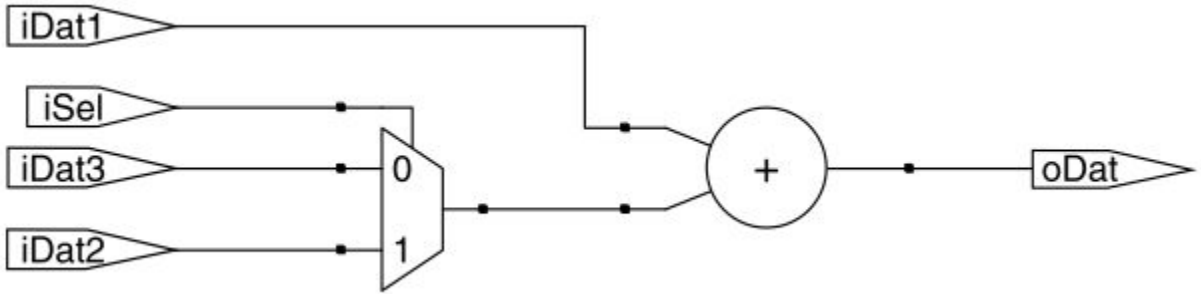
```
module addshare (
 output oDat,
 input iDat1, iDat2, iDat3,
 input iSel);

 assign oDat = iSel ? iDat1 + iDat2: iDat1 + iDat3;
endmodule
```

NO SHARING



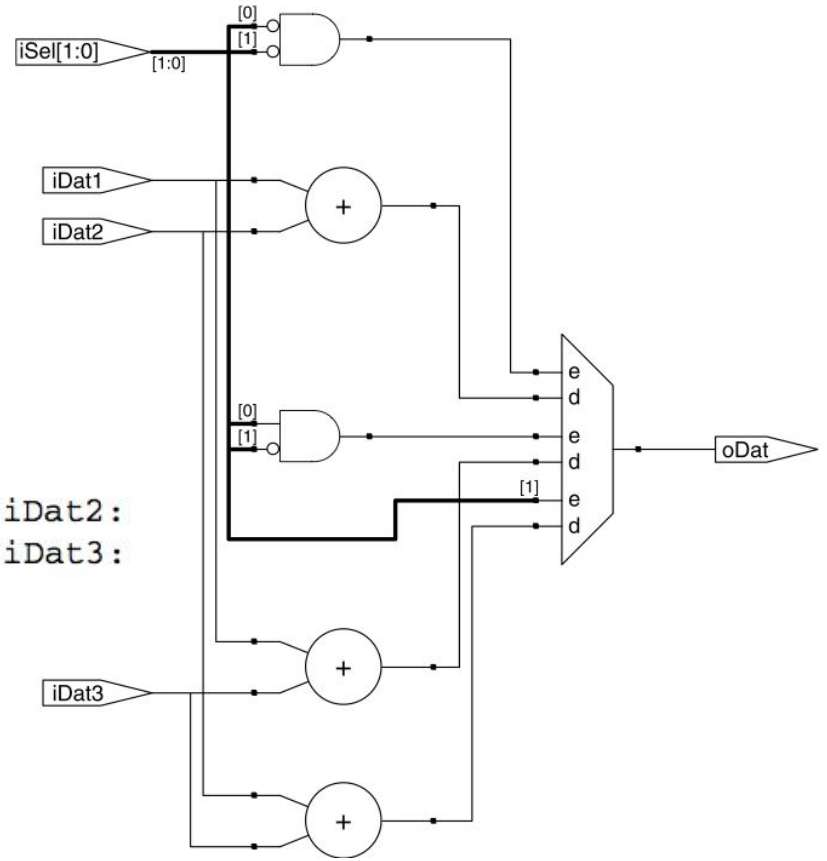
Resource Sharing

SHARING ON

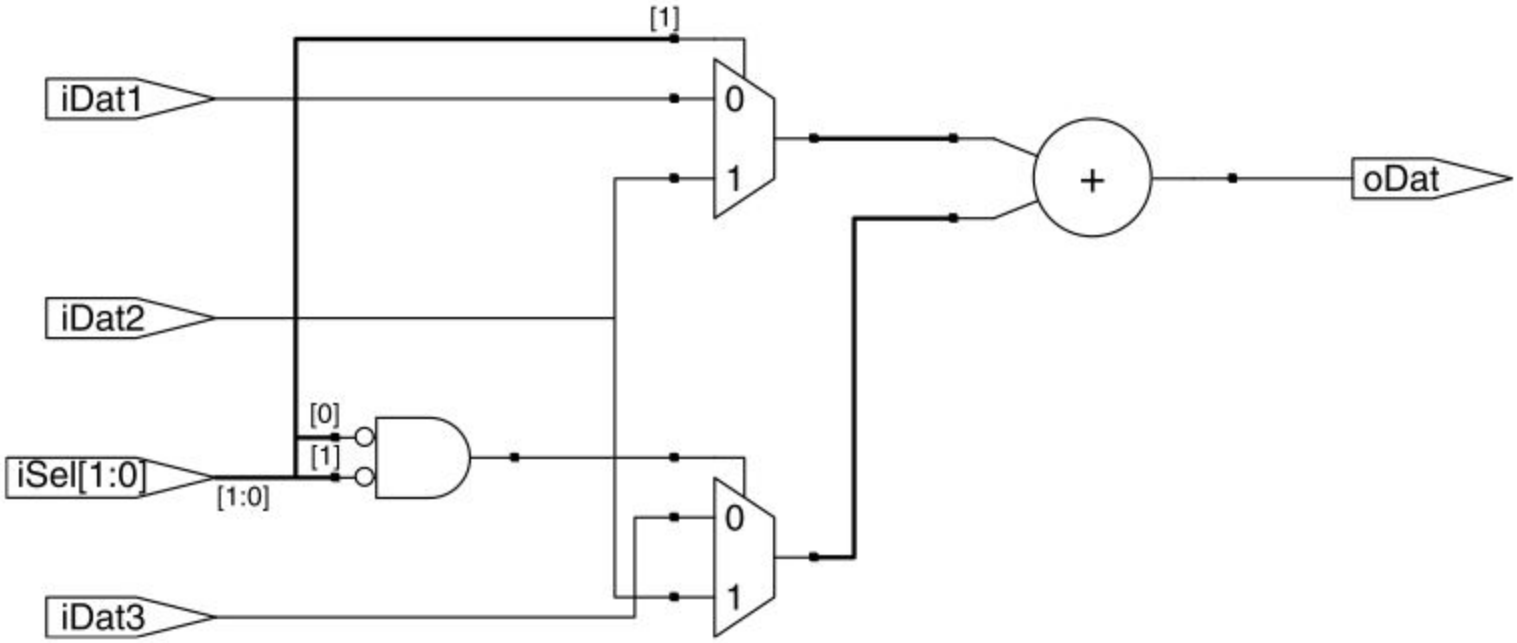


No Sharing

```
module addshare (
 output       oDat,
 input        iDat1, iDat2, iDat3,
 input [1:0]  iSel);

 assign oDat = (iSel == 0) ? iDat1 + iDat2:
               (iSel == 1) ? iDat1 + iDat3:
               iDat2 + iDat3;
endmodule
```



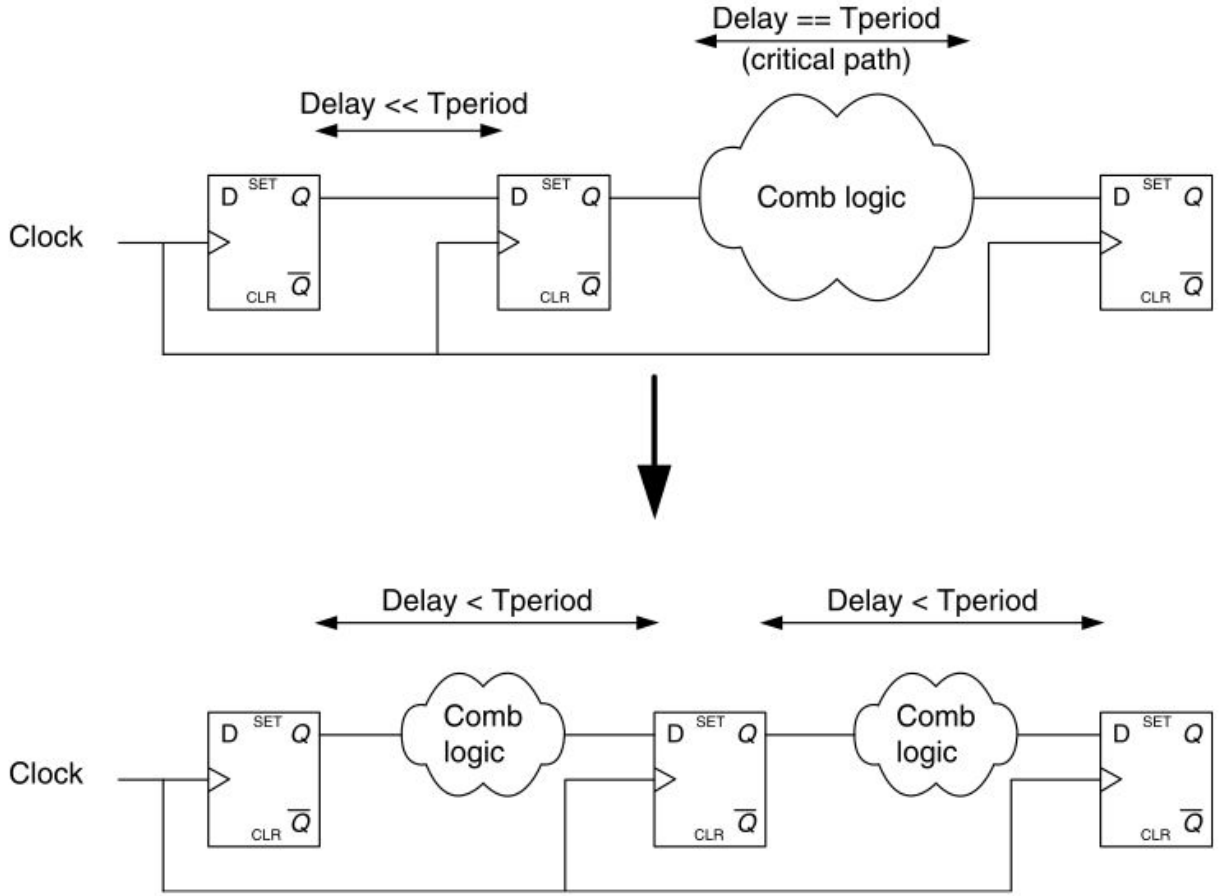
Shared Adder



Sharing is Caring…

About Resource Usage

Retiming


Delay == Tperiod
(critical path)
Delay << Tperiod
Clock
D
SET
Q
CLR
Comb logic
Delay < Tperiod
Delay < Tperiod
Comb logic
Comb logic

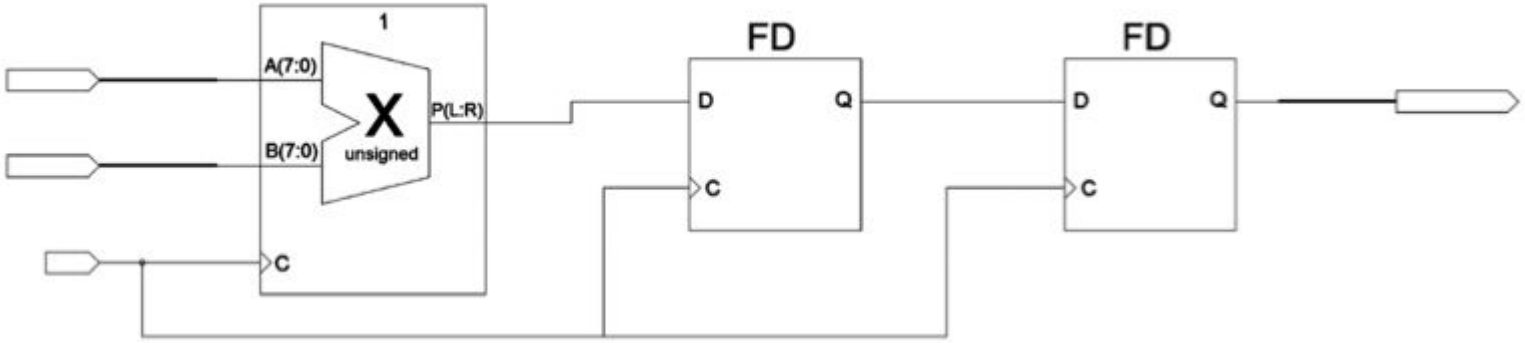
Bad Pipeline

```
module multpipe #(parameter width = 8, parameter depth = 3) (
 output [2*width-1: 0] oProd,
 input  [width-1: 0]   iIn1, iIn2,
 input                 iClk);
 reg    [2*width-1: 0] ProdReg [depth-1: 0];
 integer               i;

 assign oProd      = ProdReg [depth-1];

 always @(posedge iClk) begin
   ProdReg[0]      <= iIn1 * iIn2;

   for(i=1;i <depth;i=i+1)
      ProdReg[i]    <= ProdReg [i-1];
 end
endmodule
```

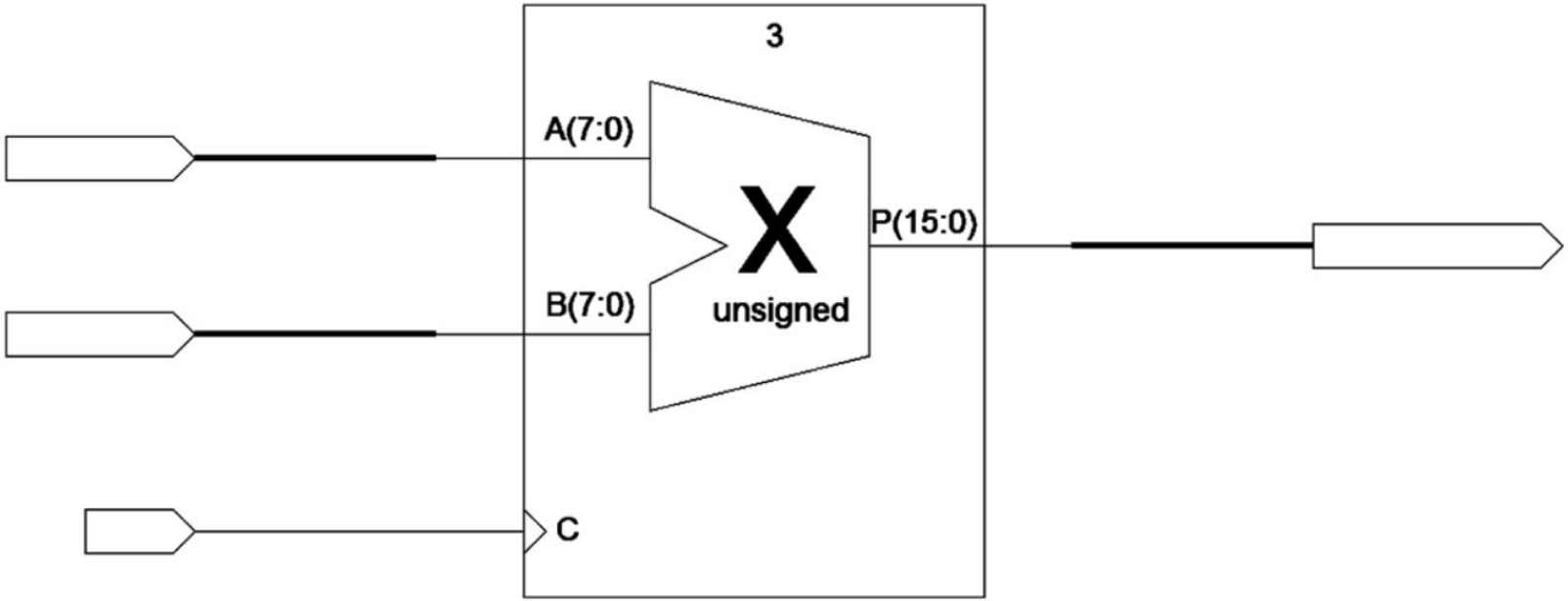
Pipelining multiplier?

Bad Pipeline

```
module multpipe #(parameter width = 8, parameter depth = 3) (
 output [2*width-1: 0] oProd,
 input  [width-1: 0]   iIn1, iIn2,
 input                 iClk);
 reg    [2*width-1: 0] ProdReg [depth-1: 0];
 integer               i;

 assign oProd     = ProdReg [depth-1];

 always @(posedge iClk) begin
   ProdReg[0]     <= iIn1 * iIn2;

   for(i=1;i <depth;i=i+1)
      ProdReg[i]    <= ProdReg [i-1];
 end
endmodule
```



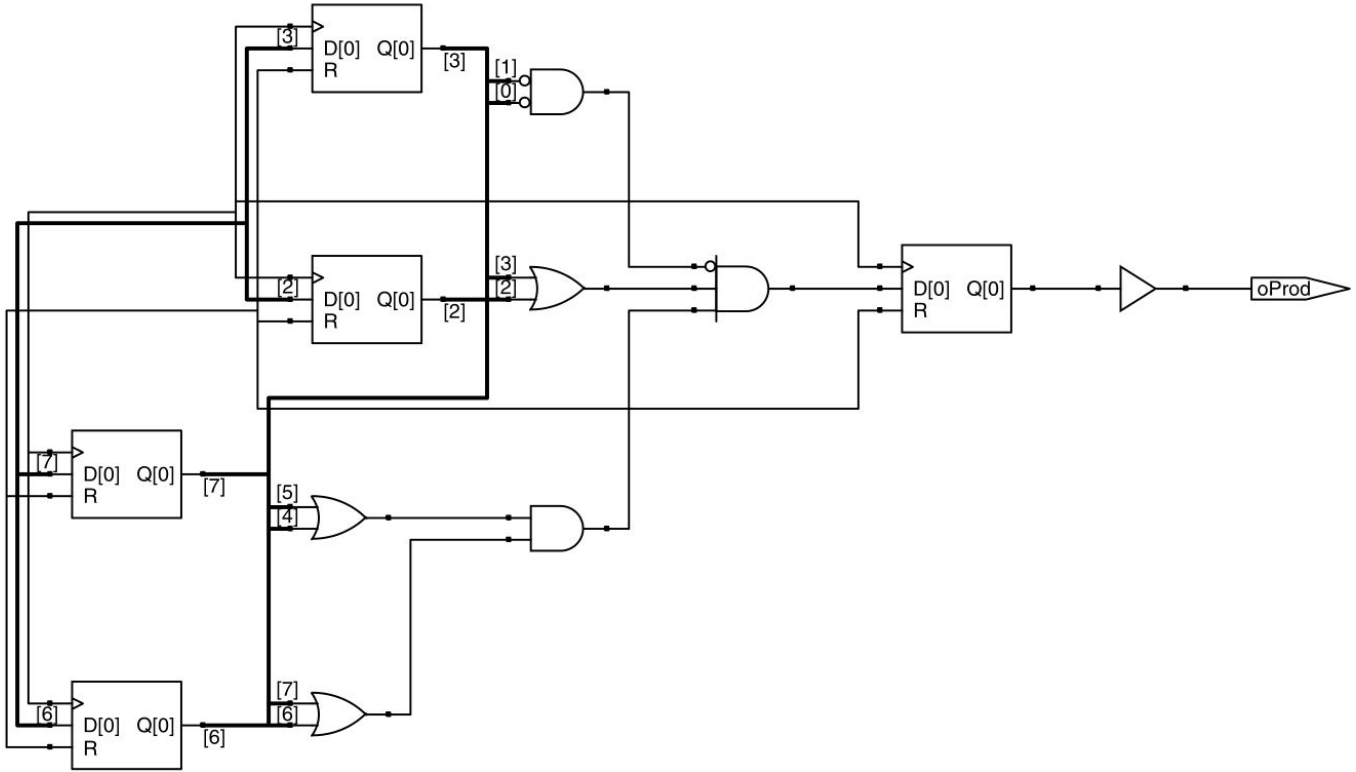
Use DSP Regs



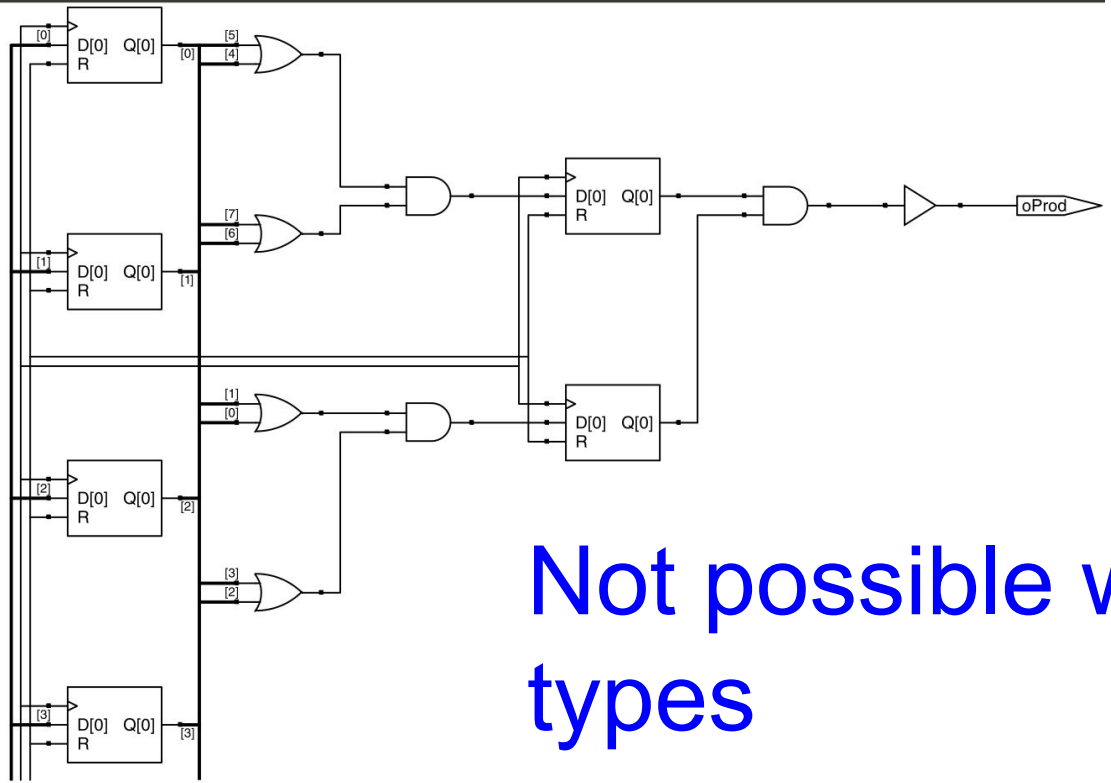
Register Balancing

```
input [7:0] iIn1,
input       iReset,
input       iClk);
reg   [7:0] inreg1;

always @(posedge iClk)
  if(iReset) begin
    inreg1  <= 0;
    oProd   <= 0;
  end
  else begin
    inreg1   <= iIn1;
    oProd <= (inreg1[0]|inreg1[1]) & (inreg1[2]|inreg1[3]) &
           (inreg1[4]|inreg1[5]) & (inreg1[6]|inreg1[7]);
  end
endmodule
```

Imbalanced Circuit

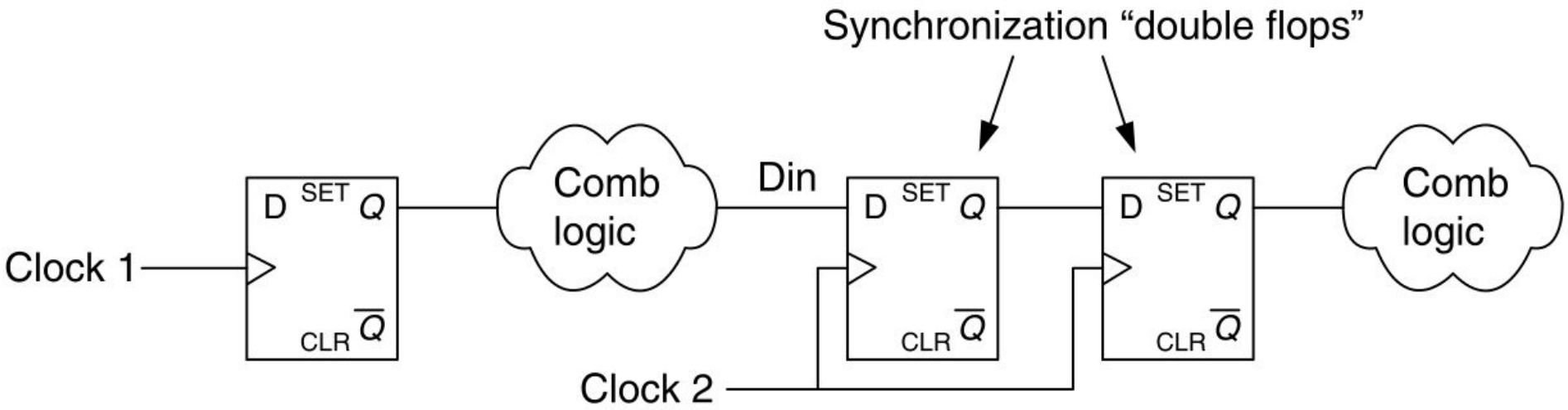


Balanced Circuit



Not possible with mixed reset types

Resynchronization Issues

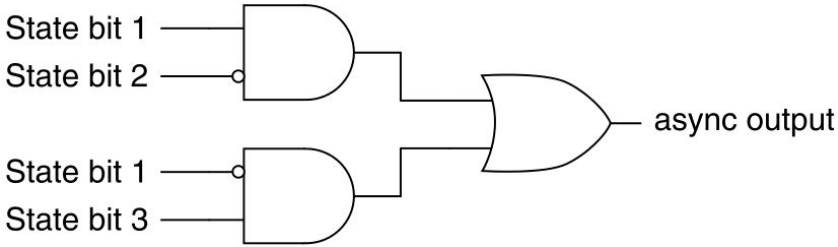


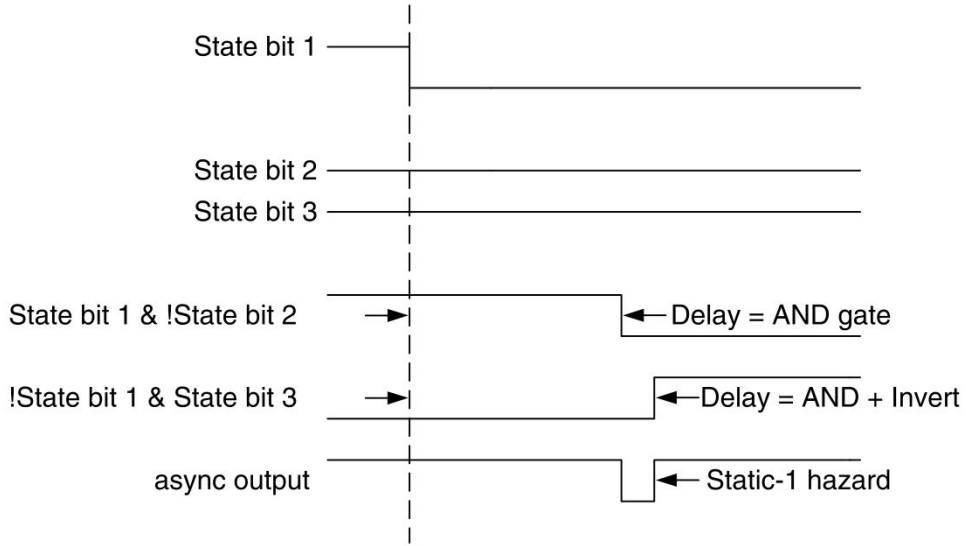
Shouldn’t rebalance across clock domains

FSM Guidelines

- Use standard FSM architecture/style
  - http://www.altera.com/support/examples/verilog/ver_statem.html
  - http://www.asic-world.com/tidbits/verilog_fsm.html
- Encoding matters
  - one-hot for performance
  - gray encoding for asynchronous, low-power
  - generic encoding, let synthesis tool decide

Why Gray Code for Async?



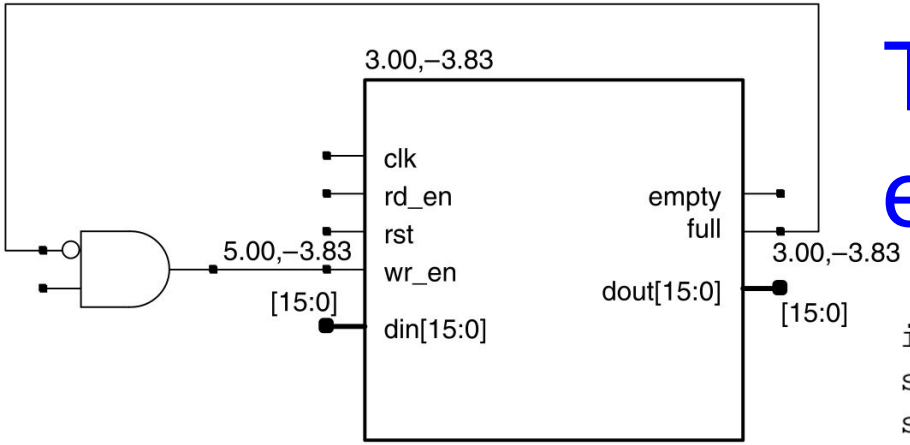


Unreachable States

```
module safesm (
 output [1:0] oCtrl,
 input        iClk, iReset);
 reg    [1:0] state;

 // used to alias state to oCtrl
 assign oCtrl = state;

 parameter STATE0 = 0,
           STATE1 = 1,
           STATE2 = 2,
           STATE3 = 3;

 always @(posedge iClk)
   if(!iReset) state <= STATE0;
   else
     case(state)
       STATE0: state <= STATE1;
       STATE1: state <= STATE2;
       STATE2: state <= STATE0;
     endcase
endmodule
```

Why do we care?

Blackboxes



Timing info helps estimate max freq.

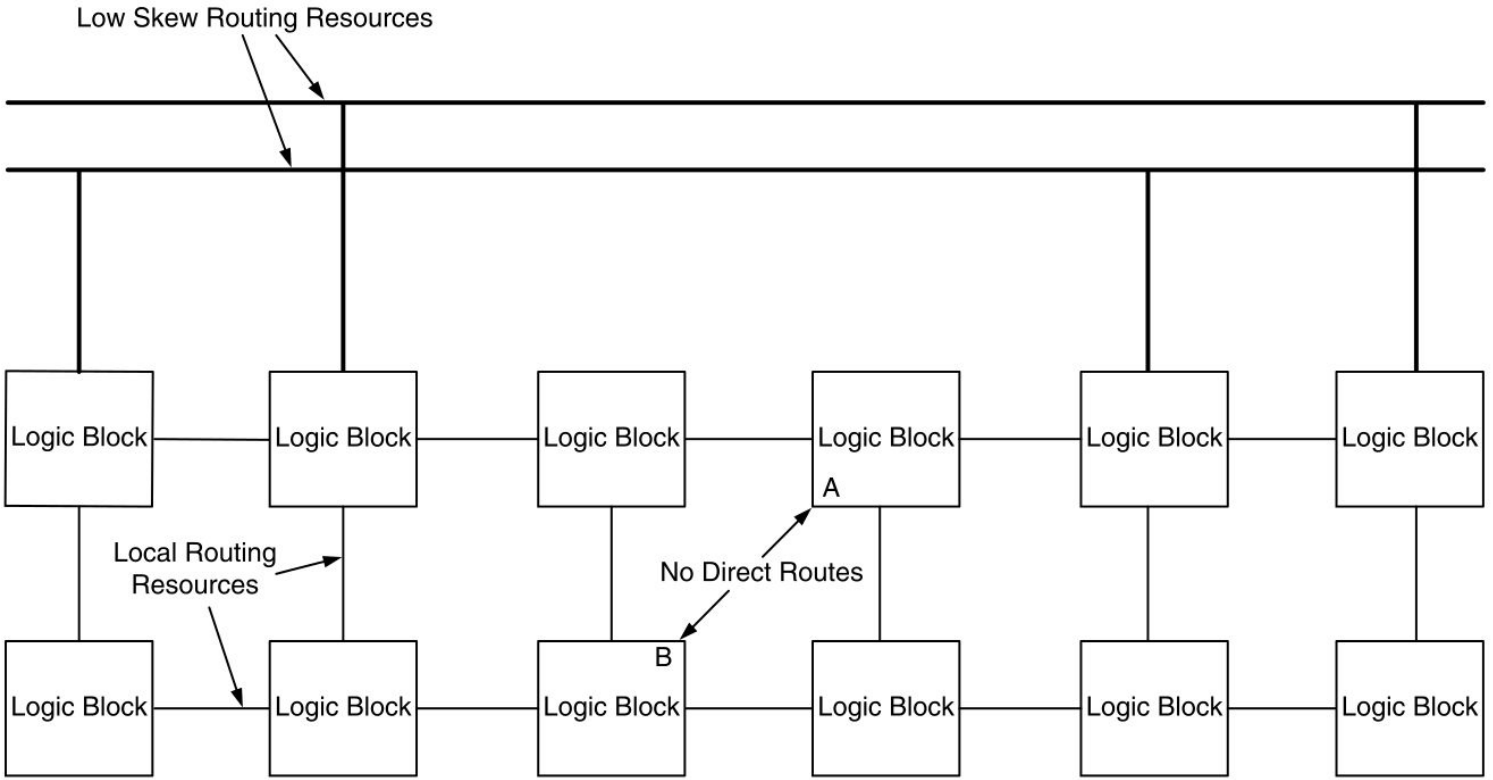
```
    input           wr_en) /* synthesis syn_black_box
    syn_tsu1 =      "din[15:0]->clk=4.0"
    syn_tsu2 =      "rd_en->clk=3.0"
    syn_tsu3 =      "wr_en->clk=3.0"
    syn_tco1 =      "clk->dout=4.0"
    syn_tco2 =      "clk->empty=3.0"
    syn_tco3 =      "clk->full=3.0"
    */;
endmodule
```

Can’t optimize blackbox

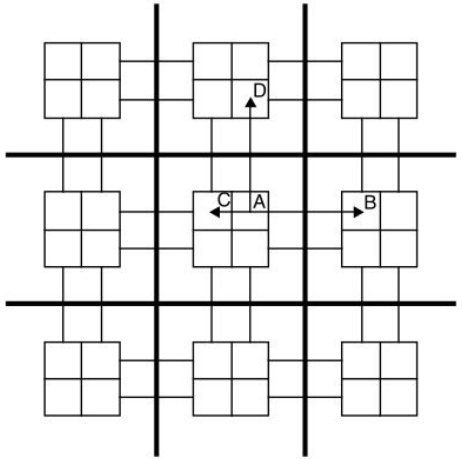
Physical Synthesis

- Use physical layout information
  - Estimate placement of components
  - Crudely place components
  - Consider placement while optimizing
- Good placement estimation greatly assists routing

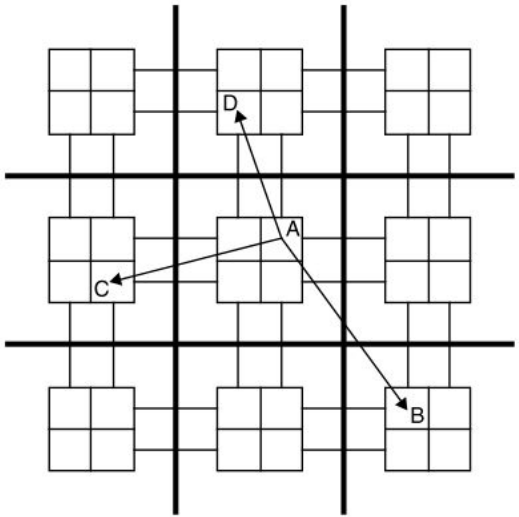
FPGA Routing Matrix



Graph Based Synthesis



Placement based on minimal physical distances and route lengths



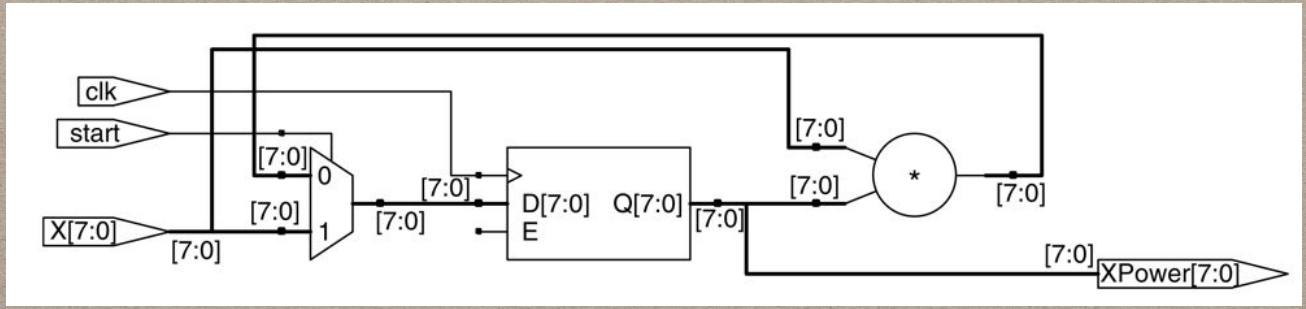
Placement based on minimal distances weighted with congestion and availability of resources

Summary

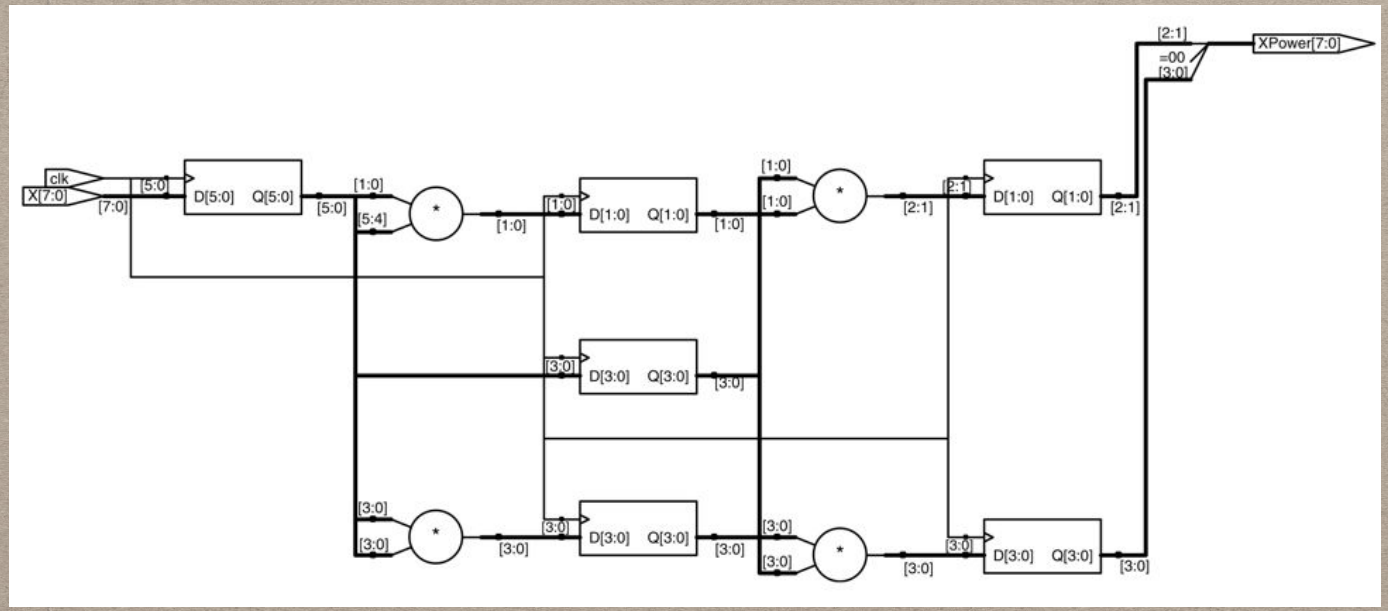
- If resource sharing is activated, verify that it is not adding delay to the critical path.
- Register balancing should not be applied to noncritical paths.
- Adjacent flip-flops with different reset types may prevent register balancing from taking place.
- Constrain resynchronization registers such that they are not affected by register balancing.
- Design state machines with standard coding styles so they can be identified and re-optimized by the synthesis tool.
- Use gray codes when driving asynchronous outputs.
- If a black box is required, include the timing models for the I/O.
- Physical synthesis provides tighter correlation between synthesis and layout.

Power3 Low Area

```
XPower = 1;
  for (i=0;i < 3; i++)
    XPower = X * XPower;
```



Power3 Parallel Structures



Peak Region

- Reached max clock
- Why does performance drop?