HDL HW2: Pipelined add/subtract and multiplier

HDL HW2 Outlines

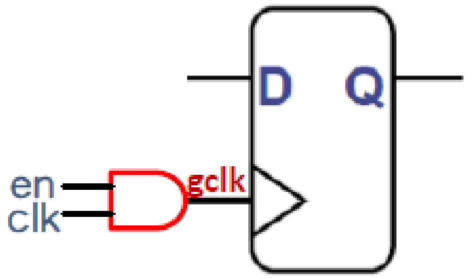
- 2-stage pipelined add/sub multiplier d=(a+/-b)*c
  - add/sub in 1st stage
  - multiplier in 2nd stage
- clock gating to reduce power when c=0
  - Prime Time dynamic analysis for power measurement
- RTL and gate-level simulation
  - Synopsys Design Compiler for logic synthesis
  - comparison of power report in static analysis and dynamic analysis with half of input c are zeros
- FPGA implementation
- Lab slides for EDA tools
  - PrimeTime
  - Xilinx vivado

non-pipelined vs. pipelined

- d=(a+b)*c if s=1 and d=(a-b)*c if s=0
- non-pipelined
  - using either continuous assignment in **assign** with ternary operator **? :** , or
  - using procedural assignments inside **always @(…)** begin … end
- pipelined
  - 1st stage: conditional add/sub
  - 2nd stage: multiplier
  - use pipeline registers with asynchronous reset
  - identify the delay T_1, T_2 in each pipelined stage
    - ✓ throughput = 1 / max(T_1, T_2)
    - ✓ latency = $T_1 + T_2$
    - ✓ pipeline can increase throughput, but not latency
- area, delay, power reports from Synopsys DC

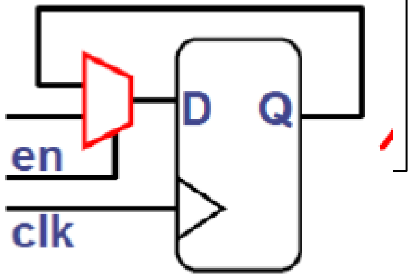
clock gating

```
module dff(Q, D, clk);
input D, clk;
output Q;
reg Q;
wire gclk, en;
// clock signal is from the output of AND
// glitch might cause extra clock edges
assign gclk = clk & en;
always @(posedge gclk)
   Q <= D;
endmodule
```



gclk might have glitches !!!
cause unexpected latching

```
module dff(Q, D, clk);
input D, clk;
output Q;
reg Q;
wire en;

// data input from MUX
always @(posedge clk)
if (en) begin
Q <= D;
end
endmodule
```

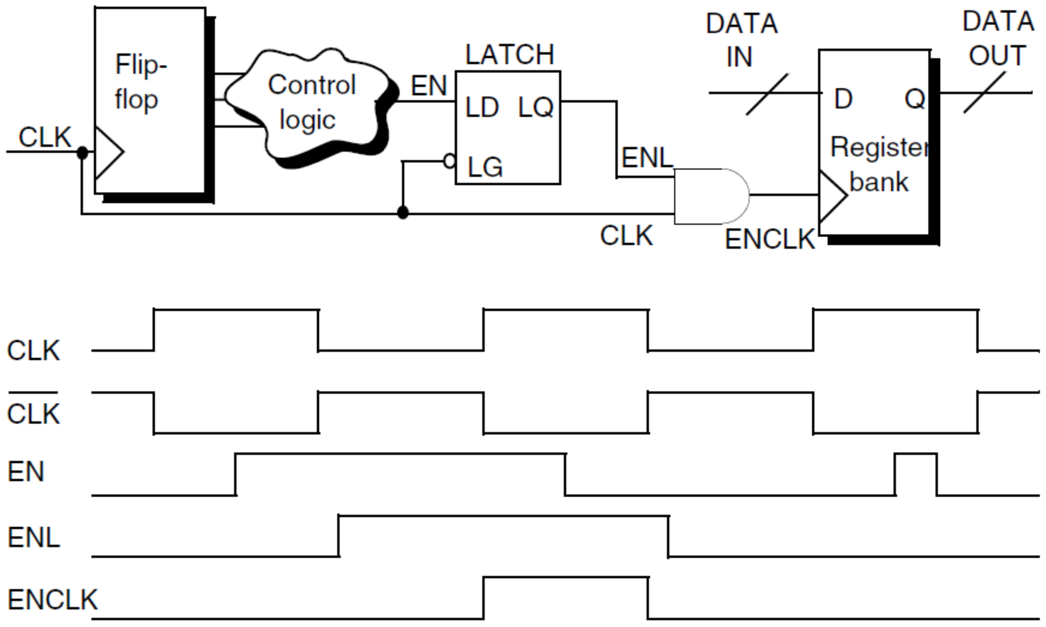


The clk port might still have switching power!
not efficiently reduce dynamic power

Latch-based clock gating (Safe Design)

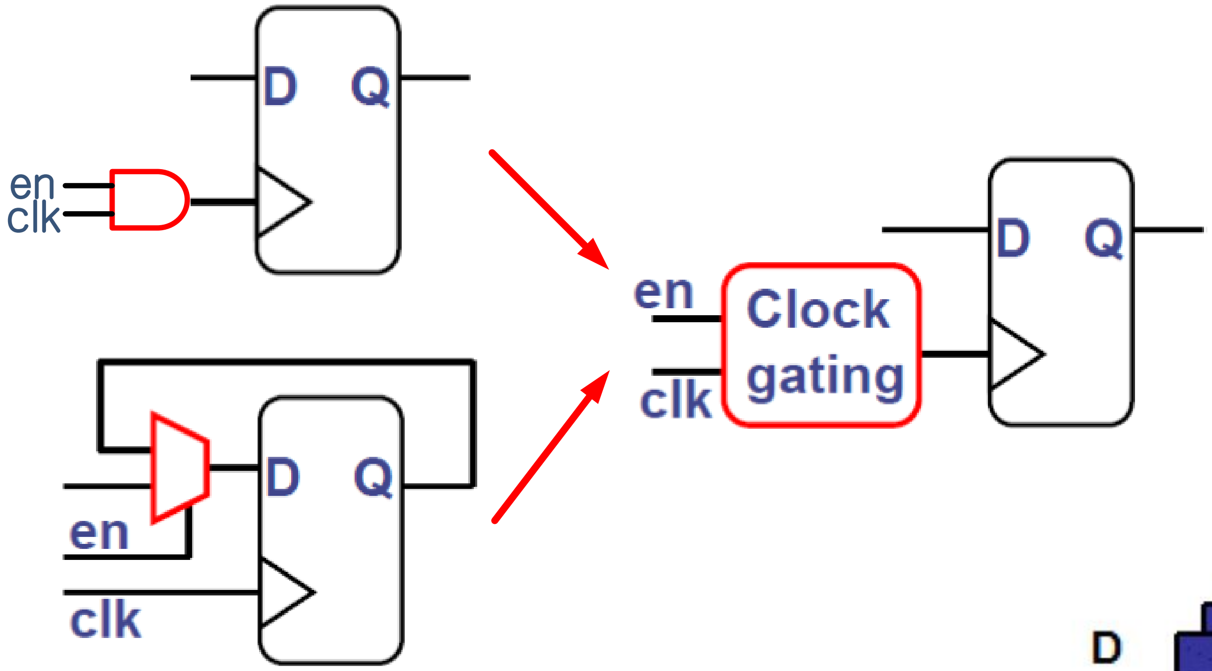
- avoid glitches in clock signals
  - glitches incur unwanted signal edges
- clock-gating signal is en is latched by a latch controlled by ~clk before feeding into the AND gate
  - signal to the input of AND is stable when clk goes high

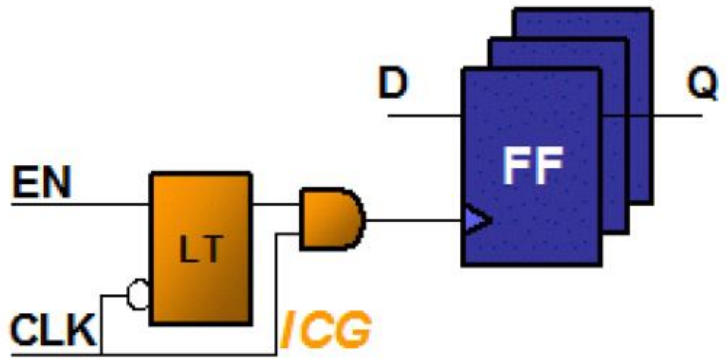
```
module safe_clock_gated_dff (Q, D, clk);
input D, clk;  output Q;
reg tmp, Q;

always @ (CLK or EN) // latch
  if (!CLK)  tmp = EN;
assign gclk = tmp1 & CLK;  // AND gate
always @(posedge gclk)  // CG FF
  Q <= D;
endmodule
```



Synthesized Clock Gating

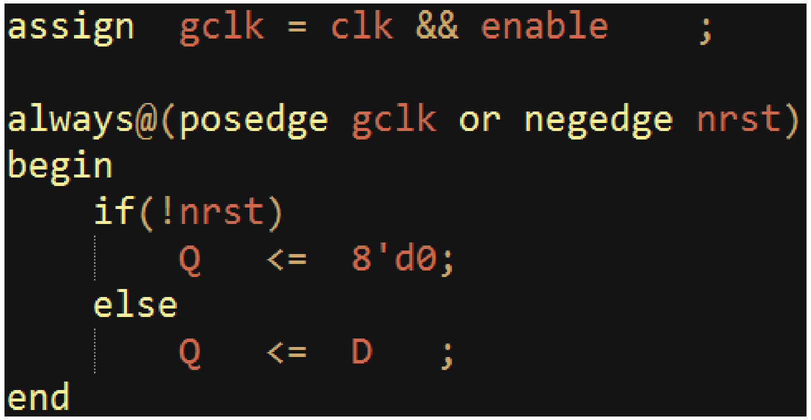
- When synthesis is done with proper commands
  - Both design leads to safe clock gating circuit (with latch)



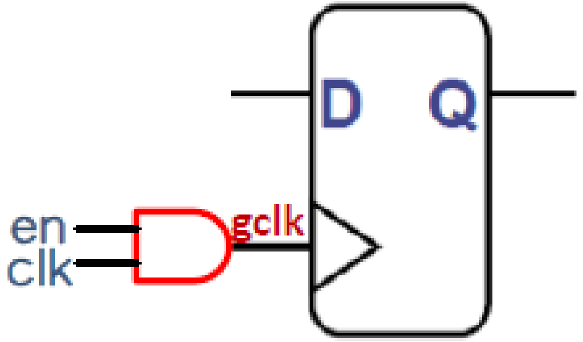


Clock gating methods

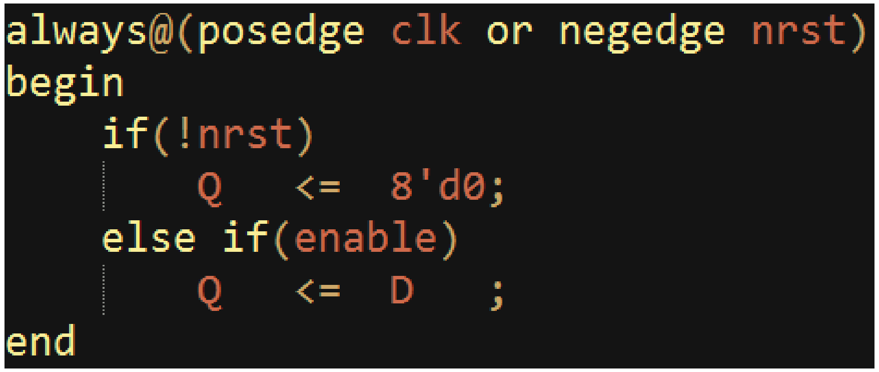
- Method 1 (unsafe gclk with possible glitches)



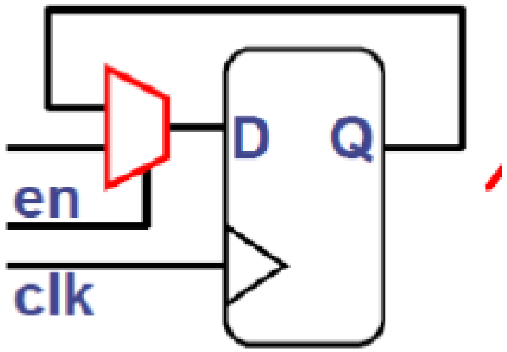
```
assign  gclk = clk && enable    ;

always@(posedge gclk or negedge nrst)
begin
    if(!nrst)
        Q   <=  8'd0;
    else
        Q   <=  D   ;
end
```



- Method2 (high switching activity of clk)

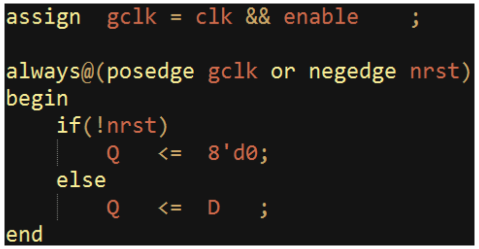


```
always@(posedge clk or negedge nrst)
begin
    if(!nrst)
        Q   <=  8'd0;
    else if(enable)
        Q   <=  D   ;
end
```

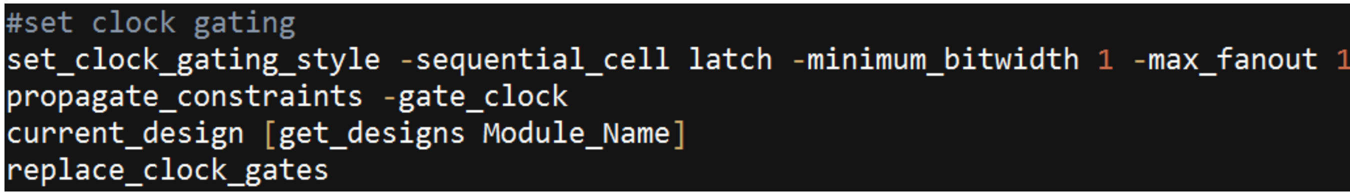


Synthesize lock gating with Synopsys

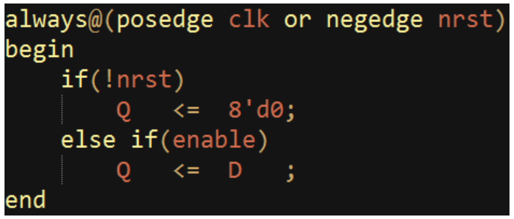
- use Synopsys synthesis script ***set_clock_gating_style*** (or other commands) which automatically find the gated clock signal and generate the safe clock-gating design



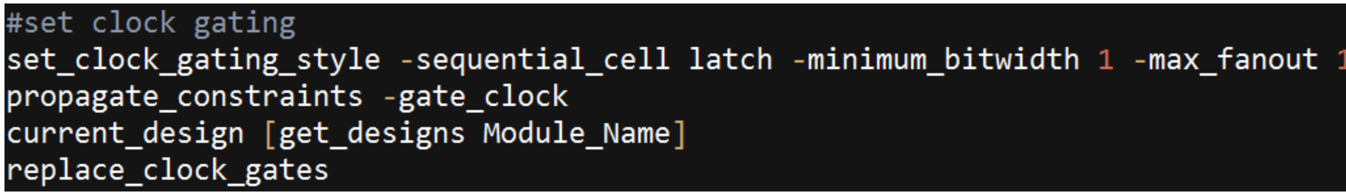
```
assign  gclk = clk && enable    ;

always@(posedge gclk or negedge nrst)
begin
    if(!nrst)
        Q   <=  8'd0;
    else
        Q   <=  D   ;
end
```



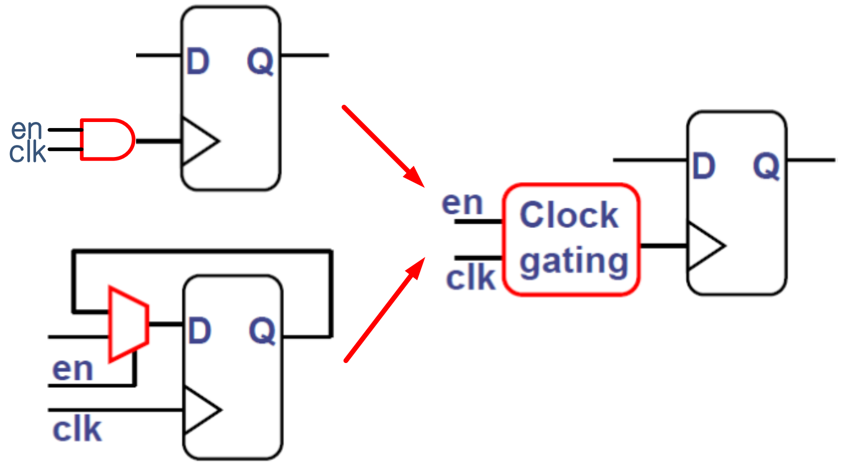
```
#set clock gating
set_clock_gating_style -sequential_cell latch -minimum_bitwidth 1 -max_fanout 1
propagate_constraints -gate_clock
current_design [get_designs Module_Name]
replace_clock_gates
```

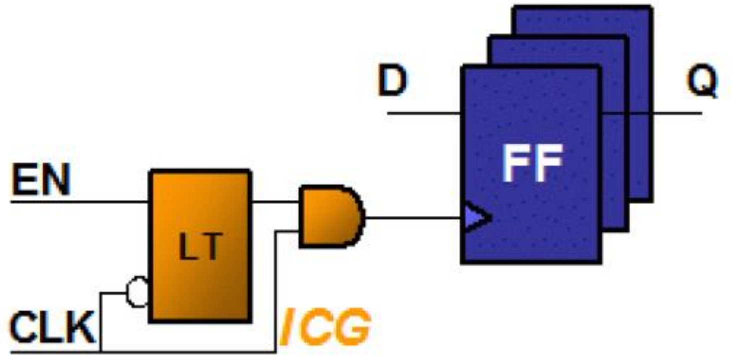


```
always@(posedge clk or negedge nrst)
begin
    if(!nrst)
        Q   <=  8'd0;
    else if(enable)
        Q   <=  D   ;
end
```



```
#set clock gating
set_clock_gating_style -sequential_cell latch -minimum_bitwidth 1 -max_fanout 1
propagate_constraints -gate_clock
current_design [get_designs Module_Name]
replace_clock_gates
```





Static vs. Dynamic Analysis

- static timing analysis
  - e.g. reports from Synopsys Design Compiler
  - no need to provide inputs
  - possible false path delay
  - could not measure power due to clock gateing at some condition of input signals
- dynamic timing analysis
  - e.g., Synopsys PrimeTime
  - need to provide input test patterns
  - maximum delay among all the provided inputs
  - could measure actual power due to clock gating

summary table

- synthesis results
  - delay and power from DC and PrimeTime might be different

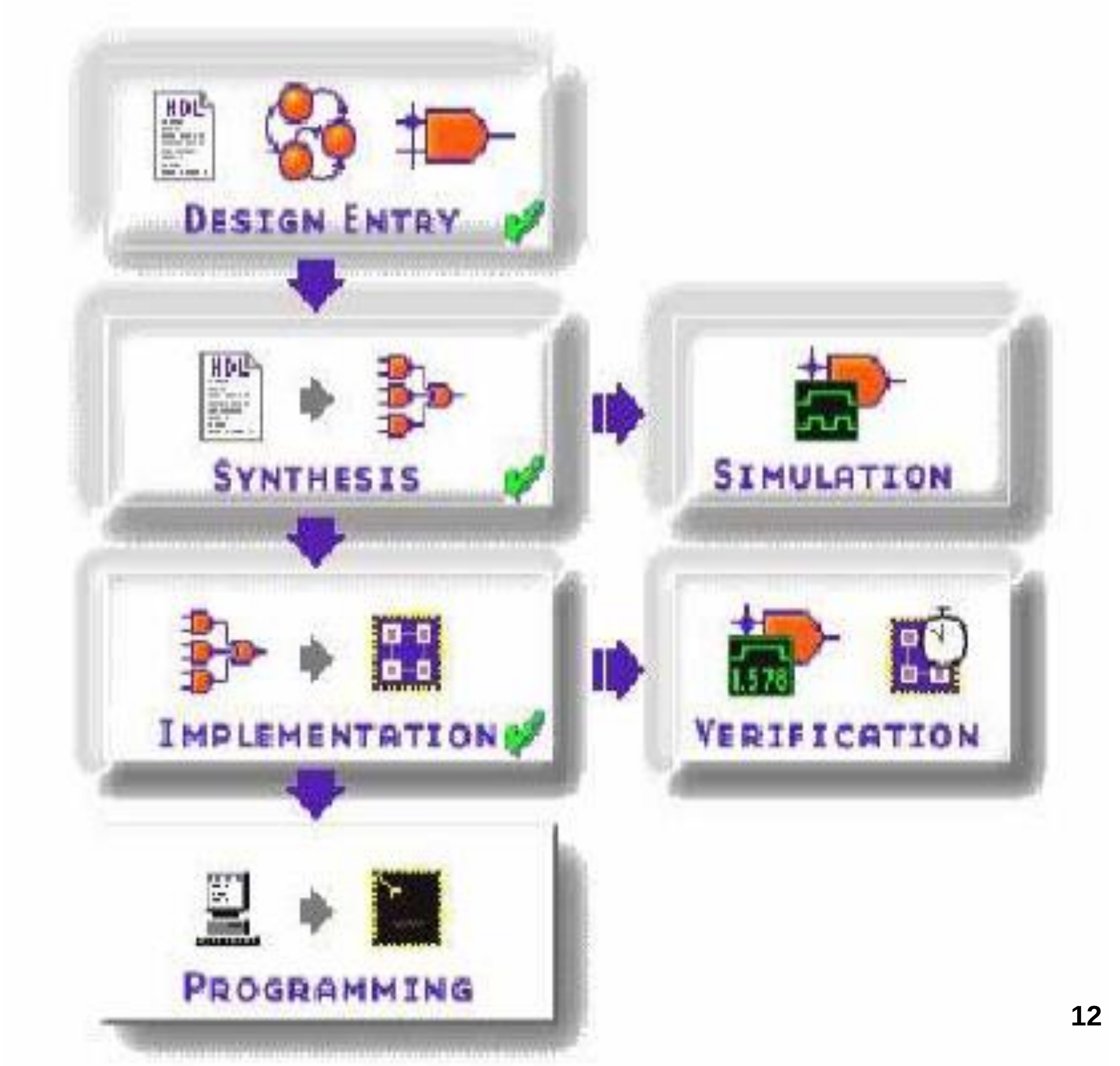
	Area (um^2)			Delay (ns)	Latency (ns)	Power (W)		
	CL	SL	Total			dynamic	leakage	total
Non-pipelined (DC)								
Non-pipelined (PrimeTime)								
Pipelined (DC)								
Pipelined (PtimeTime)								
Clock-gated (DC)								
Clock-gated (PrimeTime)								

FPGA

- Xilinx Vivado synthesis
  - similar to Synopsys DC logic synthesis
  - convert RTL to gate netlists for a selected FPGA chip
- Implementation
  - similar to placement-and-routing in ASIC cell-based design flow
  - map to the selected FPGA
- utilization rate of FPGA resources
  - LUT
  - DSP
  - BRAM
  - I/O
  - …

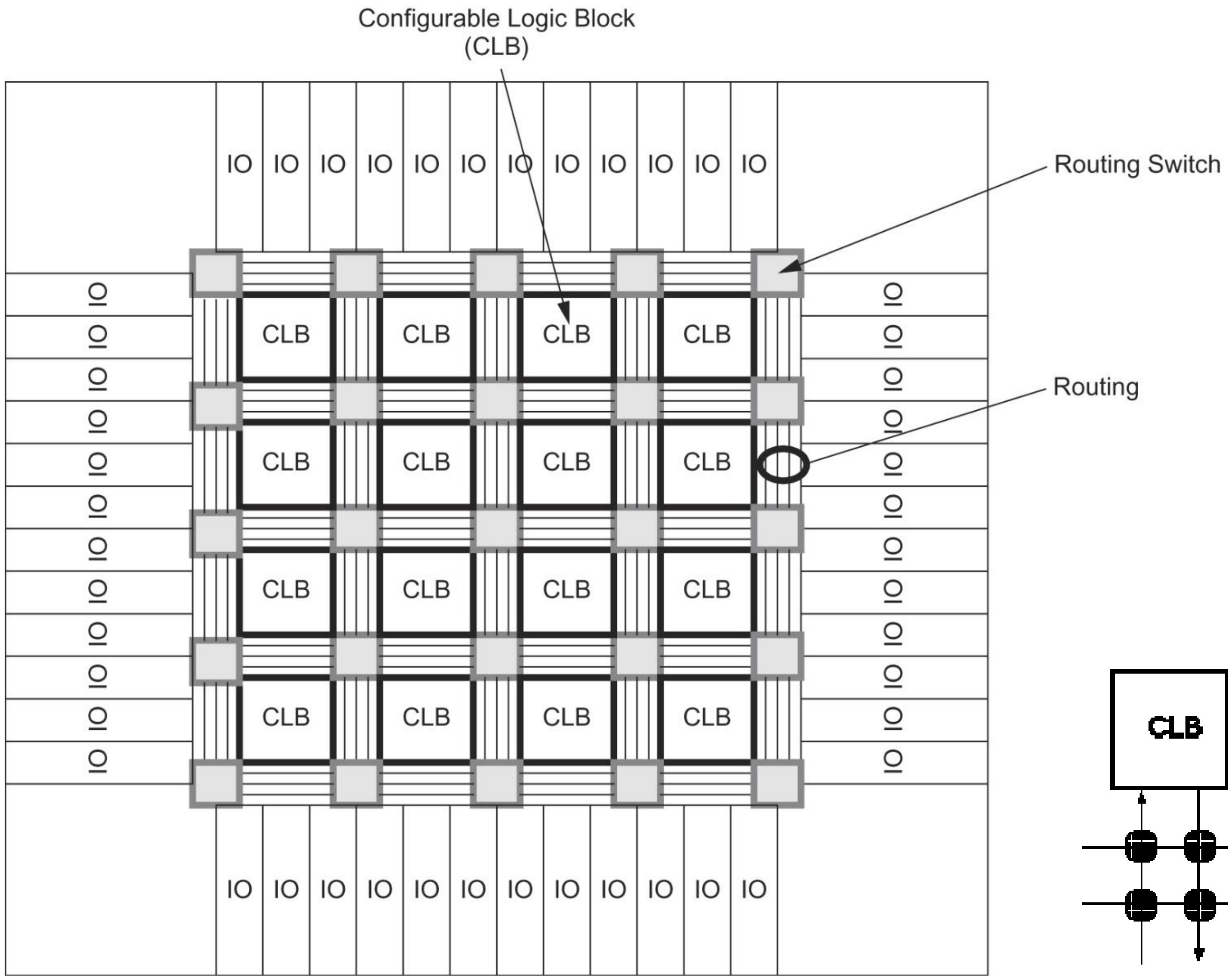
Xilinx design flow

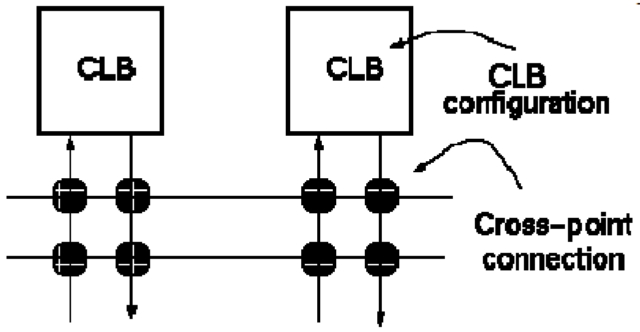
- synthesis
- implementation
- programming



Xilinx Virtex FPGA

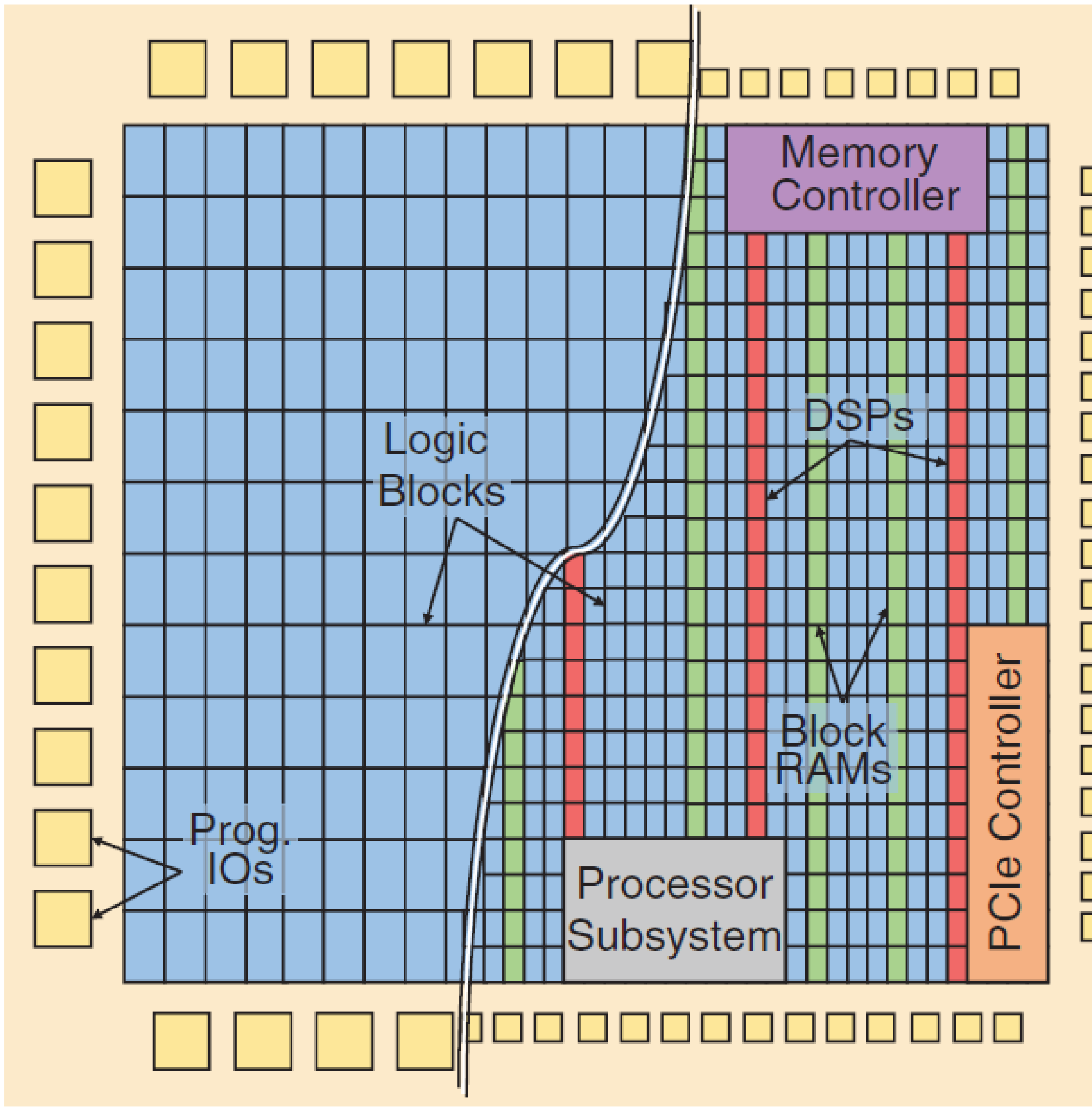
- CLB (configurable logic block)





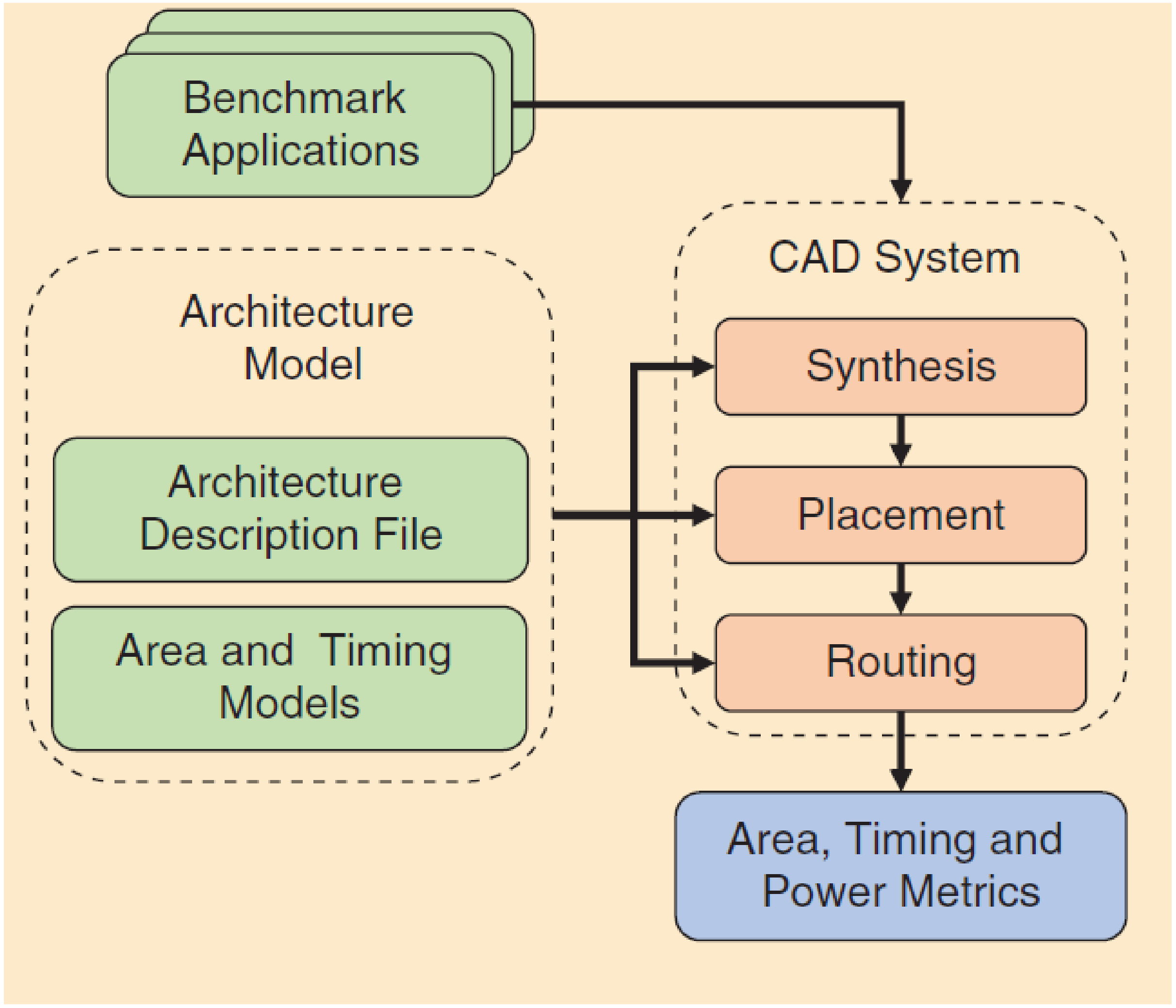
FPGA (early vs. modern)

- early FPGA
  - programmable array logic (PAL)
  - I/O
- modern FPGA
  - CLB, LE, LB, ..
  - I/O
  - block RAM (BRAM)
  - DSP
  - other hard blocks
- all connected by bit-level routing



FPGA Evaluation Flow

- Similar to cell-based design flow
  - logic synthesis
  - placement and routing
- But FPGA can be implemented on real FPGA chips
  - quick prototyping
  - reconfigurable
  - cell-based needs tape out to foundry and non-reconfigurable
- area in utimization rate of a particular FPGA chip
  - CLB. LE
  - I/O
  - LUT
  - DSP
  - BRAM
  - …


Benchmark Applications
Architecture Model
Architecture Description File
Area and Timing Models
CAD System
Synthesis
Placement
Routing
Area, Timing and Power Metrics