Improving Timing,

Area, and Power

<Ref> Verilog Styles for Synthesis of Digital Systems, Smith/Franzon, Prentice-Hall 2000

- Timing Issues in Design
- Low-Power Issues in Design
- Area Issues in Design

Timing Issues in Design

Performance and Implementation area are fundamental importance in the design of a digital system.

Timing-related design issues have a big influence on both of these.

Performance measure is

- **Latency**：How long does it take to complete a particular operation ?
- **Throughput**：How many operations can be completed per second ?

- In general, the throughput of an ASIC is defined as a function of two factors:

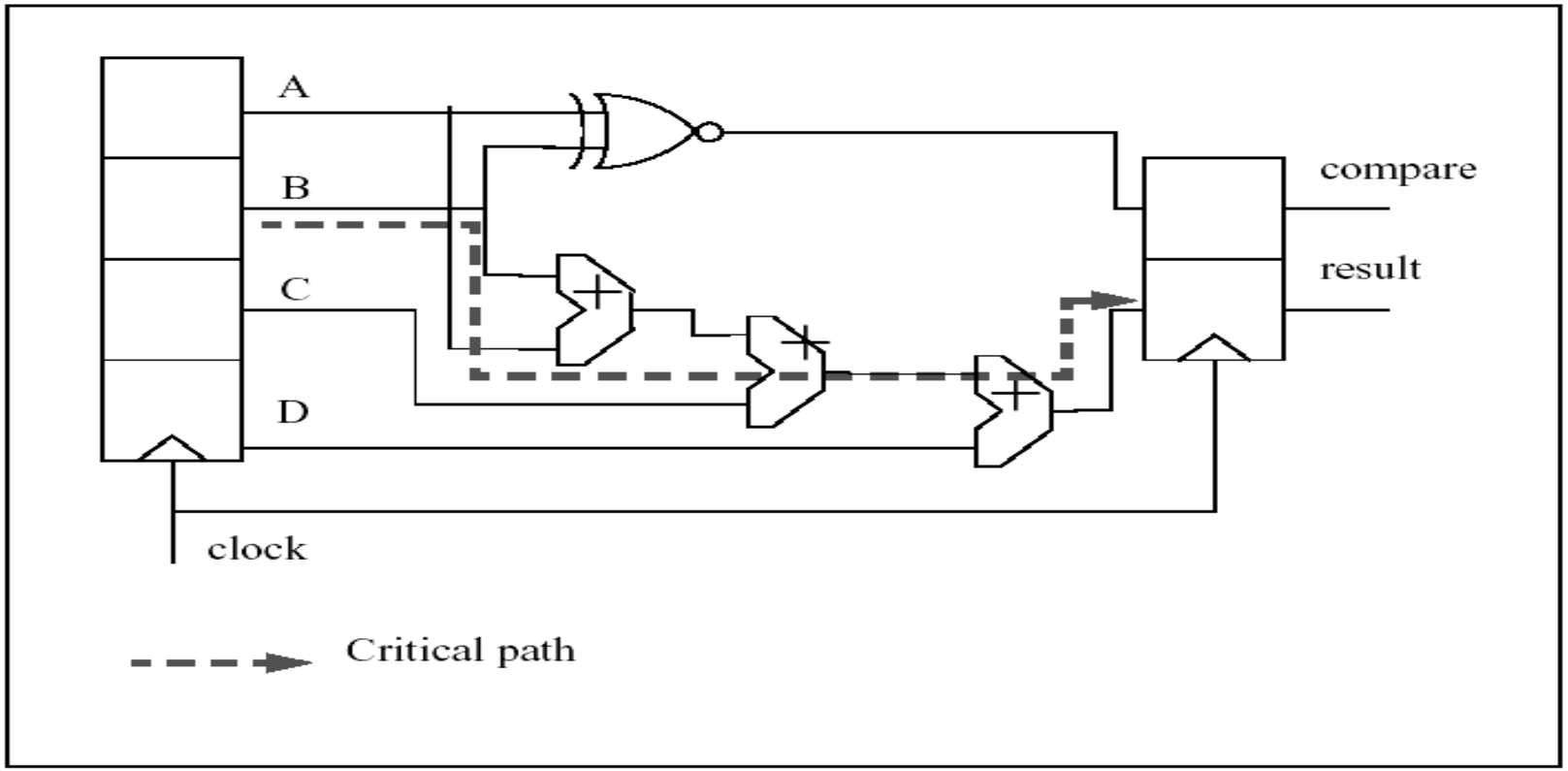
$$Throughput = f_{clock} / CPO$$

Where

- f_{clock} is the clock frequency .
- CPO is the cycles per operation.
  - how many clock ticks transpire between the completion of each operation.

Original specification fragment and corresponding critical path

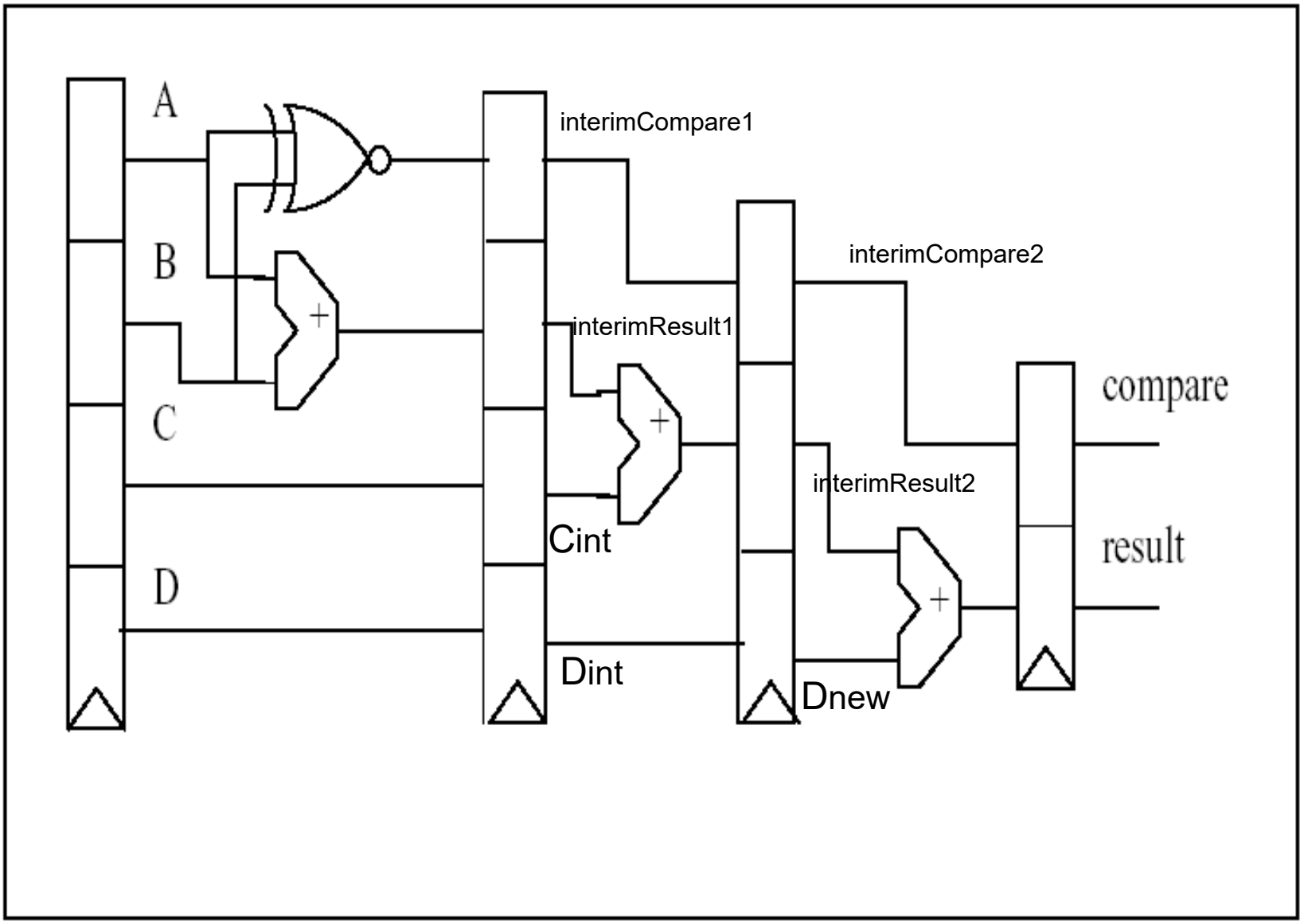
```
always@(posedge clock)
  begin
    result <= A + B + C + D;
    compare <=A ^ B;
  end
```



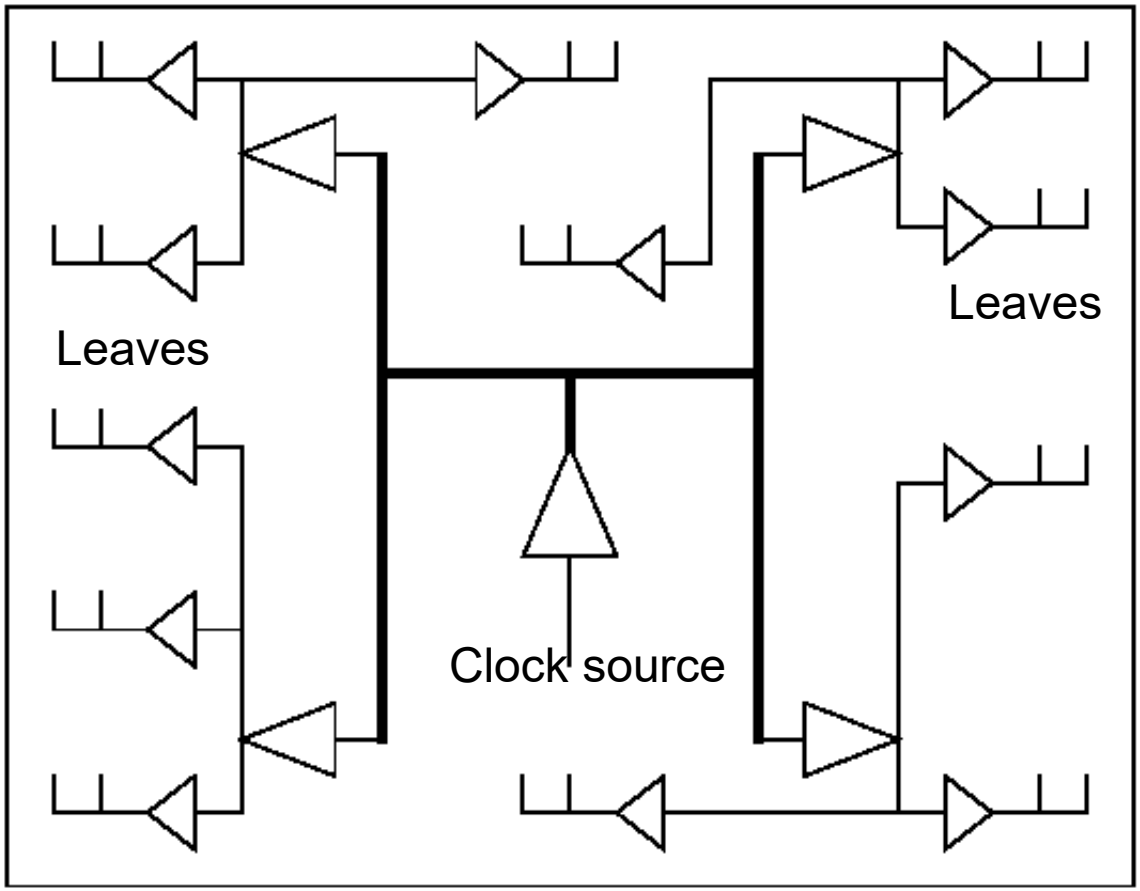
Modified specification fragment & corresponding critical path.

```
always@(posedge clock)
  begin
    interimResult1 <= A + B;
    Cnew <= C;
    Dint <= D;
    Dnew <= Dint;
    interimResult2 <= InterimResult1 + Cnew;
    Result <= InterimResult2 + Dnew;
    InterimCompare1 <= A^B;
    InterimCompare2 <= InterimCompare1;
    Compare <= InterimCompare2;
  end
```

Pipelined Design

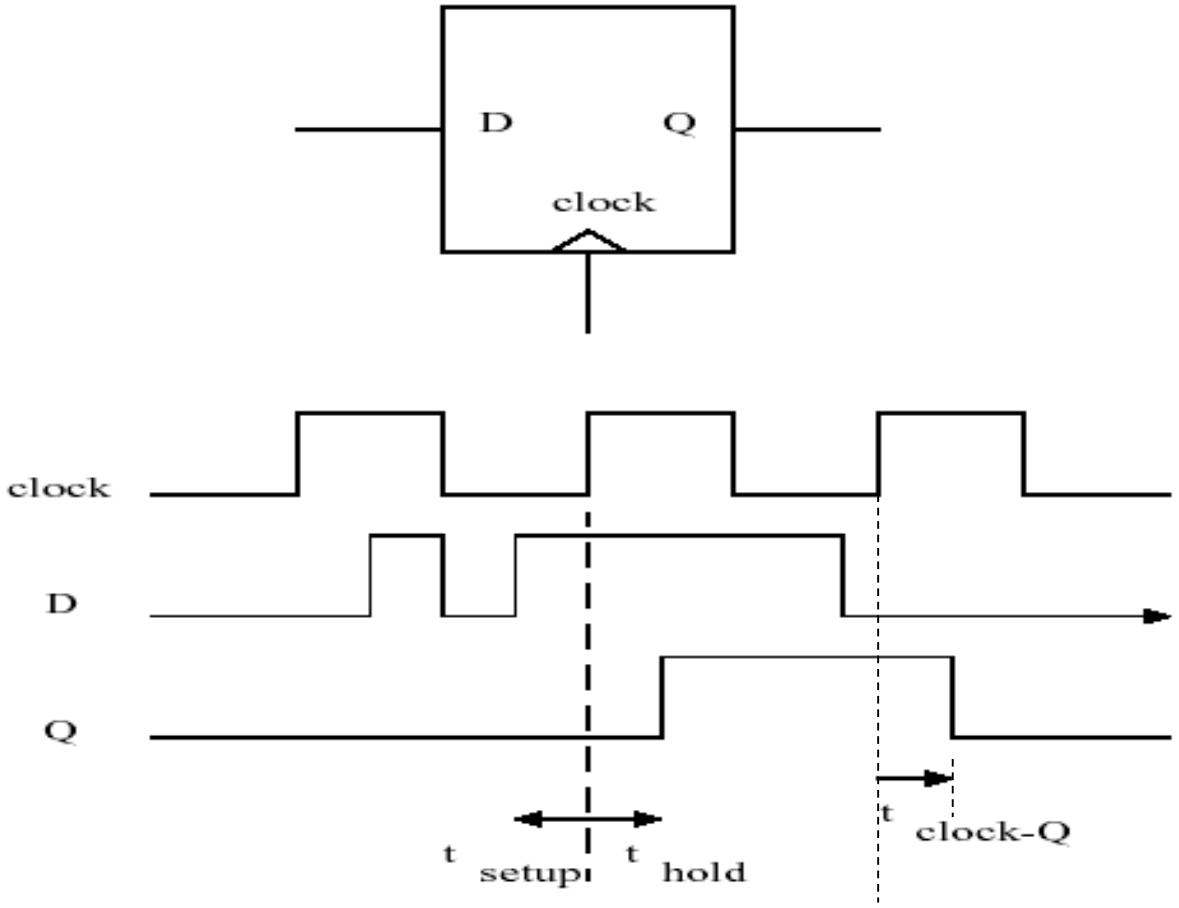


Typical Clock Distribution Scheme



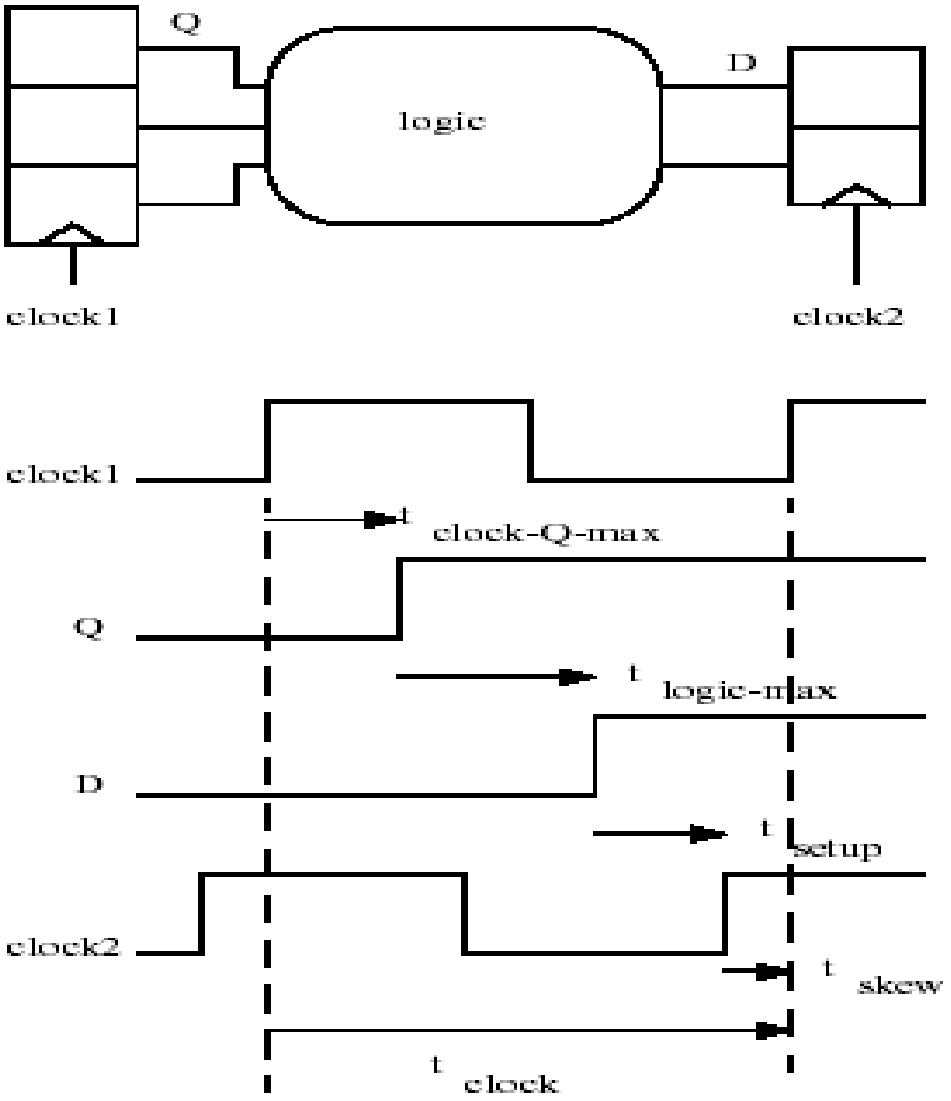
- The delay calculators are used by timing analysis tools, during and after synthesis.
- The clock tree is designed to attempt to ensure that the delay from the clock source to each of the clock leaves is identical.

Basic Flip-flop Operation



- t_{clock_Q} : D input is sampled at the rising edge of the clock and transferred to the Q output after the delay time.

Timing Calculations for Clock Period and Setup Violations



$$t_{clock} \geq t_{clock\_Q\_\max} + \sum t_{\log ic\_\max} + t_{setup\_\max} + t_{skew}$$

where

$t_{clock\_Q\_\max}$: is the maximum delay from the clock edge on the left-side FF to the Q output changing.

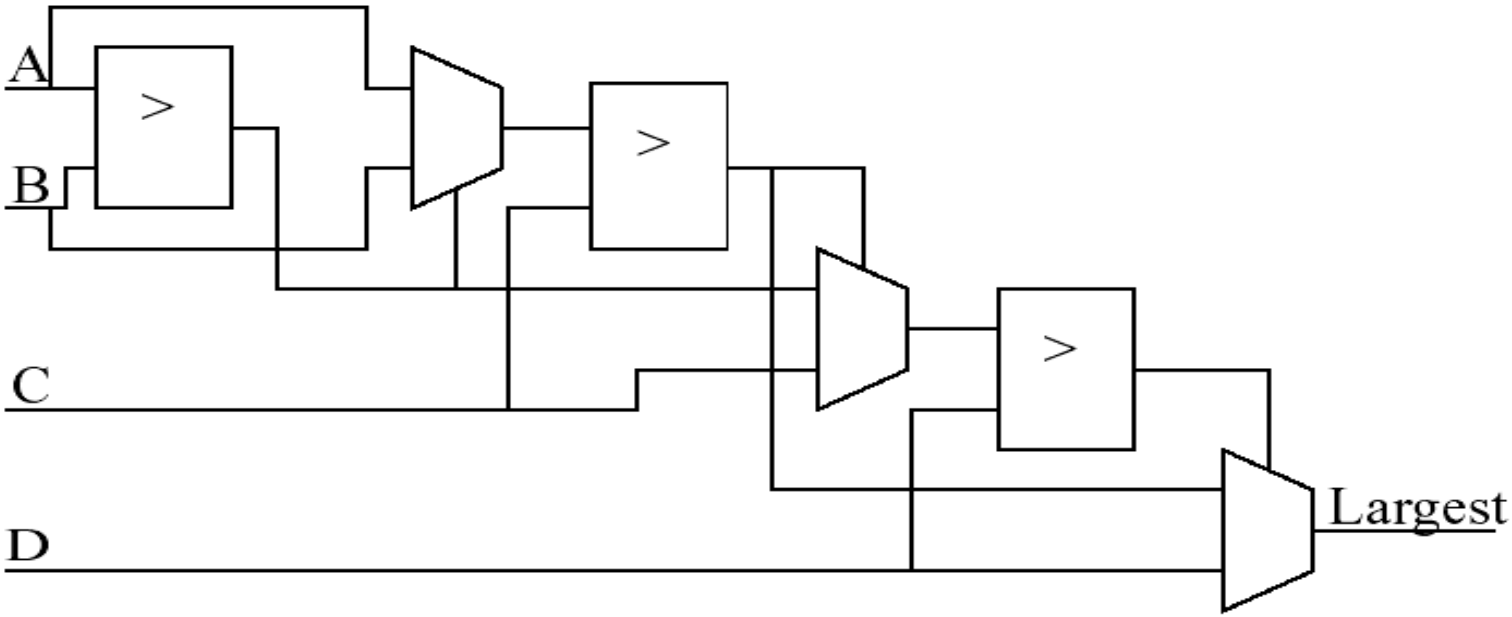
$\Sigma t_{\log ic\_\max}$: is the sum of the logic delays of the gates between successive FFs.

$t_{setup\_\max}$: is the longest possible setup time requirement.

t_{skew} : is the worst-case clock skew.

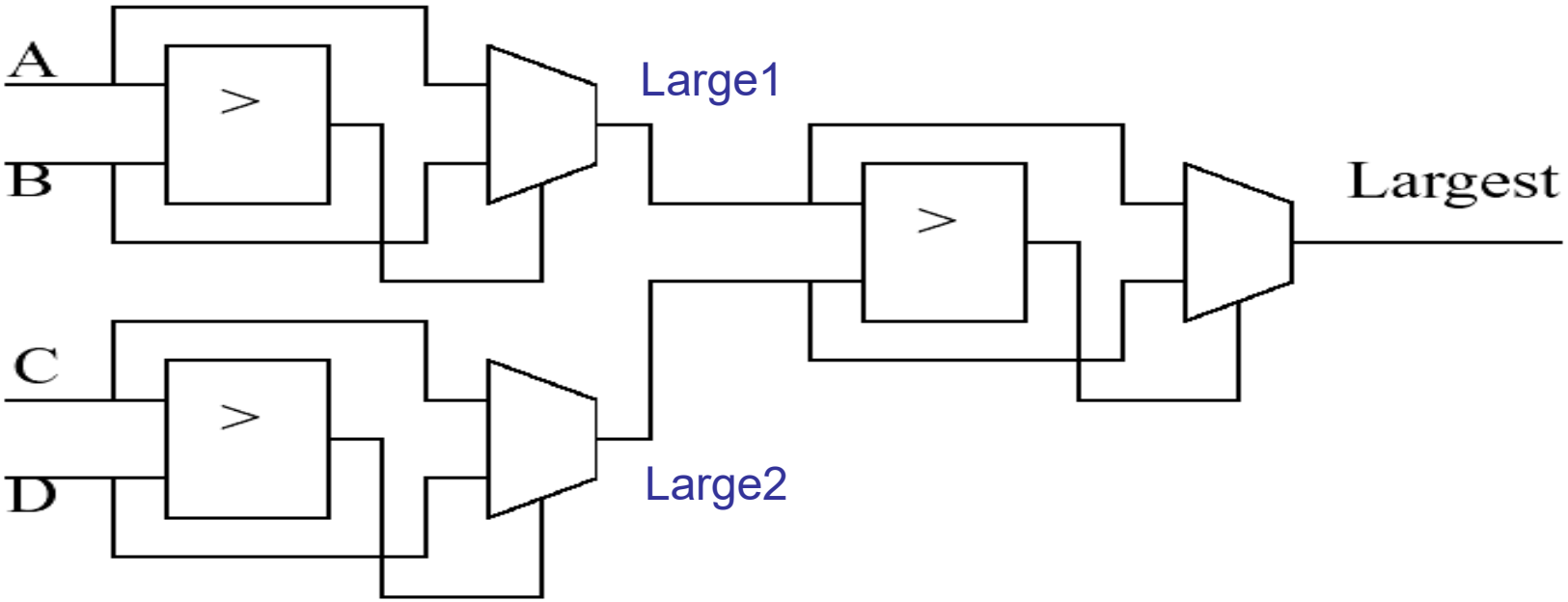
Timing aware design - slow linear search:

```
reg [15:0] A, B, C, D, Largest;

always@(A or B or C or D)
begin
  if (A>B) Largest = A;
  else Largest = B;
  if (C > Largest) Largest = C;
  if (D > Largest) Largest = D;
end
```



Fast binary search structure

```
reg [15:0] A, B, C, D;
wire [15:0] Large1, Large2, Largest;

assign Large1 = (A>B) ? A : B;
assign Large2 = (C>D) ? C : D;
assign Largest = (Large1 > Large2) ? Large1 : Large2;
```



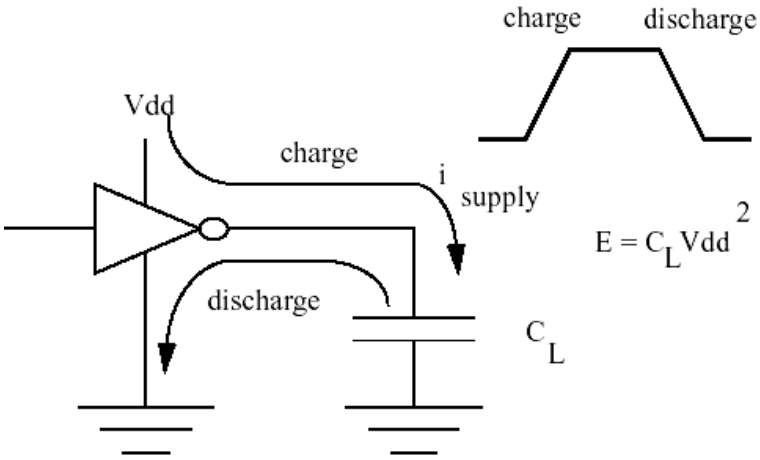
Low Power Issues in Design

- Power Consumption in CMOS Circuits

$$i_{\sup ply} = C_L \cdot \frac{dV}{dt}$$

V : the instantaneous voltage across the capacitance.

Circuit model for power dissipated in a CMOS gate.



$$P = i_{\sup ply} Vdd = Vdd \cdot C_L \cdot \frac{dV}{dt}$$

$$E = \int_0^T P(t)dt = 2\int_0^{Vdd} C_L dV = C_L \cdot Vdd^2$$

$$Power = \sum_{nodes} N_{node} \cdot C_L Vdd^2 \cdot f_{clock}$$

N_{node} : the fraction of clock cycle; switching activity in each node.

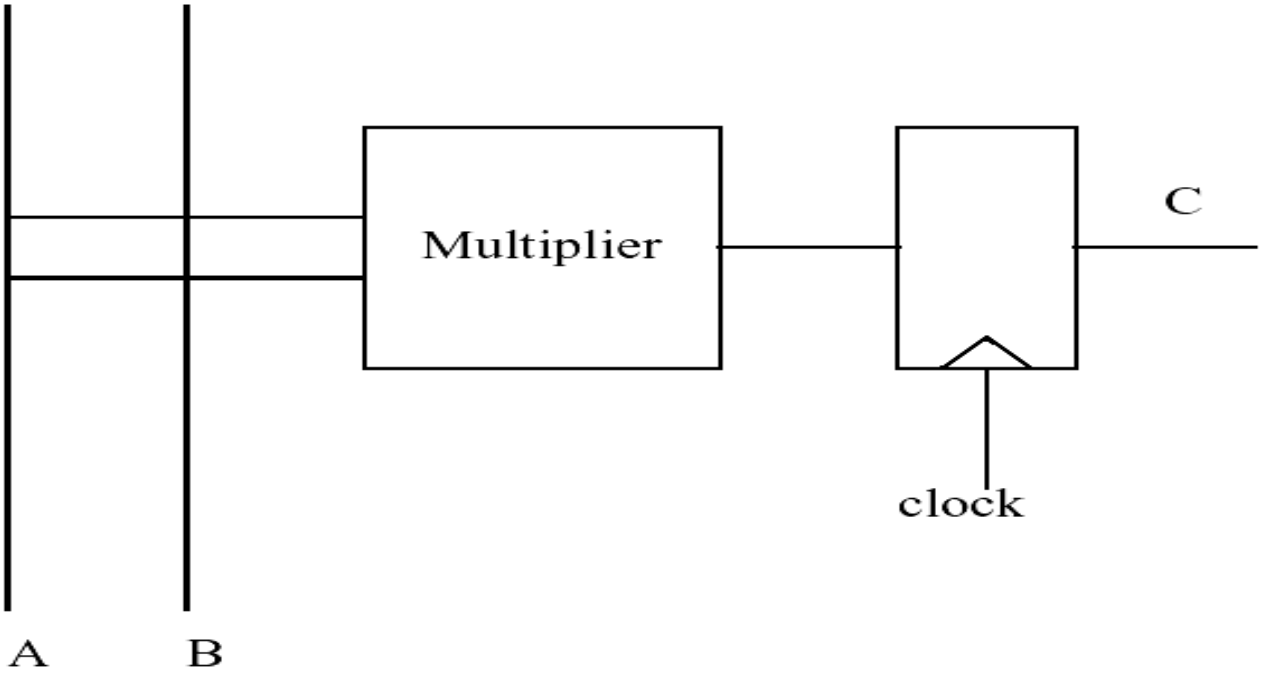
f_{clock} : the clock frequency

Potentially power-inefficient multiplier design.

```
reg [31:0] C;
reg [15:0] A,B;

always@(posedge clock)
  C = A * B;
```

• Let's assume that the output of multiplier is only actually used 10% clock cycle, When A or B change, the multiplier calculates a new result A *B

⇒ For 90% cycle time, the calculation is not used and the power consumption is wasted!

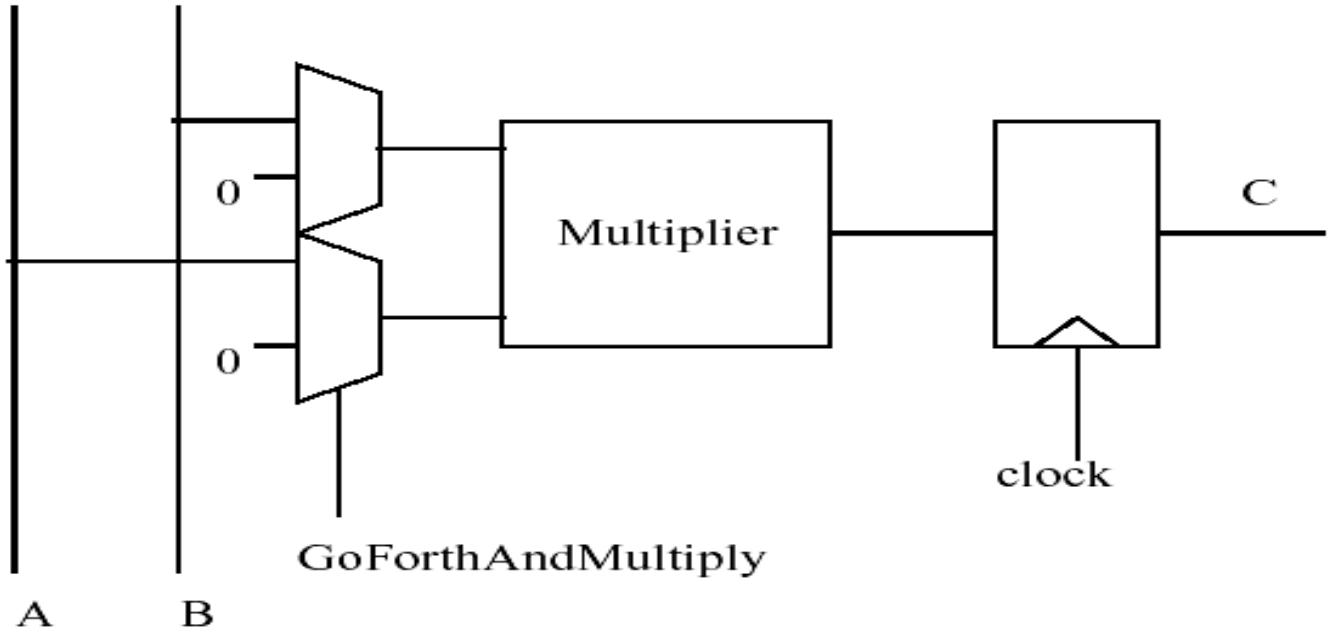


More power-efficient multiplier designs:

```
reg [31:0] C;
reg [15:0] A,B;
wire [15:0] Ain, Bin;
assign Ain = GoForthAndMultiply ? A : 16'h0;
assign Bin = GoForthAndMultiply ?  B : 16'h0;
always@(posedge clock)
    C = Ain * Bin;
```

- The input signal change only when needed.
- Tri-state z units could be used, but makes production debug and test difficult.

=> Then, replace by 0 states.



More power-efficient multiplier designs:

```
reg [31:0] C;
reg [15:0] A,B;
wire [15:0] Ain, Bin;
always@(posedge clock)
  begin
    C <= Ain * Bin;
    If (GoForthAndMultiply == 1)
        begin  Ain <= A; Bin <= B; end
  end
```

- **CAD tools in Low-power design**

There is now a variety of power design tools from CAD vendors .

- Transistor and switch level
- Gate level
- Architectural and RTL level

Area Issues in Design

Area-aware design: multiplier:

```
wire [15:0] A,B,C;
wire [7:0]  In1, In2, In3;
wire        MultSelect;

assign A = In1 * In2;
assign B = In1 * In3;
assign C = MultSelect ? A : B;
```

- Two multipliers are built, and only one of the multipliers is being used at a time.

```
wire [15:0] C;
wire [7:0]  In1, In2, In3, InSelect;
wire        MultSelect;

assign InSelect = MultSelect ? In2 : In3;
assign C = In1 * InSelect;
```

- Only one multiplier is built.

Area-aware design: register file:

```
module RegFile (clock, WE, WriteAddress, ReadAddress, WriteBus, ReadBus);

  input  clock, WE;
  input  [4:0] WriteAddress, ReadAddress;
  input  [15:0] WriteBus;
  output [15:0] ReadBus;

  reg [15:0]   RegisterFile  [0:31];   // thirty-two 16-bit registers

  assign ReadBus  =  RegisterFile [ReadAddress];

  always@(posedge clock)
      if (WE) RegisterFile [WriteAddress] <= WriteBus;

endmodule
```

WE 與 WriteAddress 無關

- In the above HDL codes, the synthesis tools are not very good at implementing the implies address decoders.

Modify the above HDL codes, then

```
module RegFile (clock, WE, WriteAddress, ReadAddress, WriteBus, ReadBus);
  input  clock, WE;
  input  [4:0] WriteAddress, ReadAddress;
  input  [15:0] WriteBus;
  output [15:0] ReadBus;
  reg [15:0]   RegisterFile  [0:31];   // thirty-two 16-bit registers
  // provide one Write Enable line per register
  wire [0:31] WElines;
  integer i;
  // explicitly build individual Write Enable lines by shifting WE to correct line
  assign WElines = (WE << WriteAddress);
  assign ReadBus  =  RegisterFile [ReadAddress];

  always@(posedge clock)
      for (i=0; i<=32; i=i+1)
      if (WEline[i]) RegisterFile [i] <= WriteBus;
endmodule
```

WE 與 WriteAddress 有關

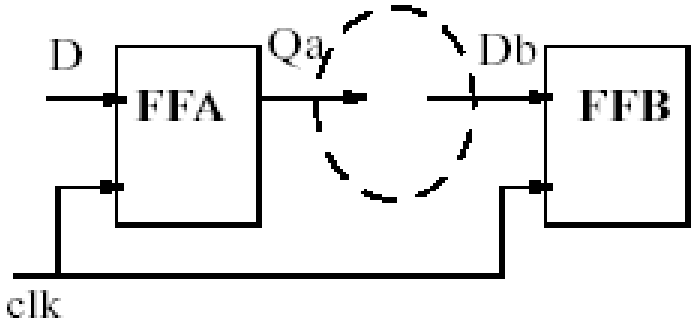
- In the above HDL codes, the design is with explicit address decode logic (as captured in the signals “WElines”).

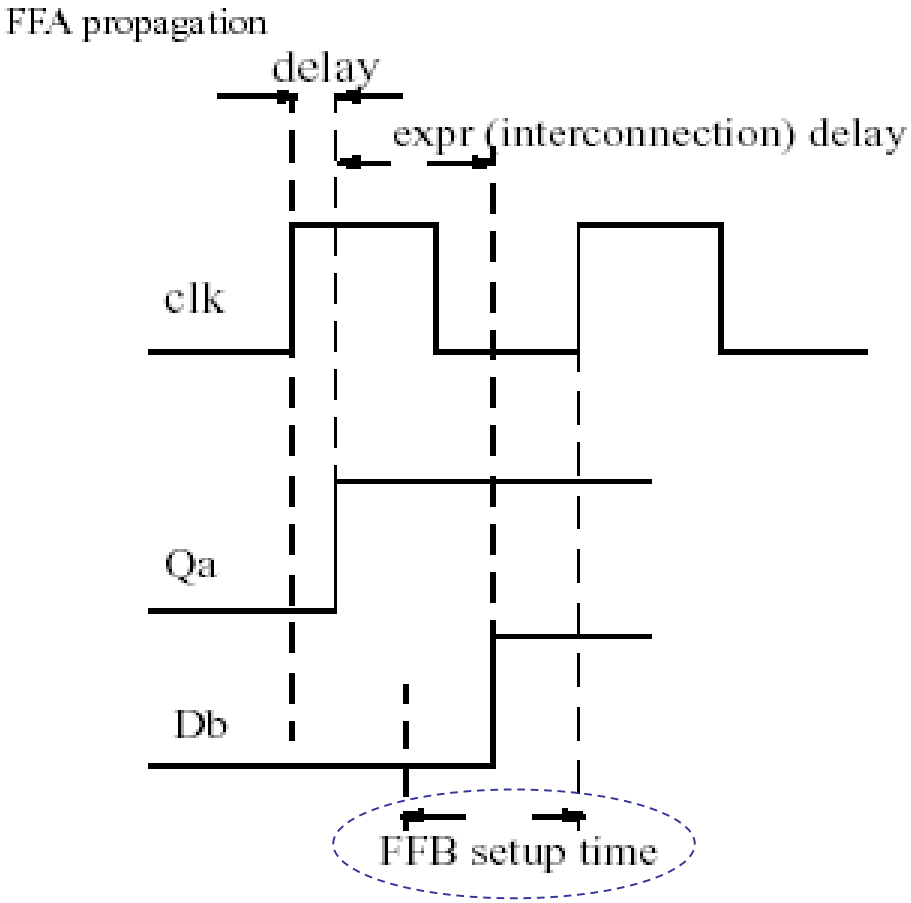
- When synthesized, this design is 20% smaller and 4% faster than the simpler, implicit design (with cell-based design).

Some Usual Suspects

Setup Violation

```
always @(posedge clk)
  begin
    FFA <= D;
    FFB <= expr(FFA);
  end
```



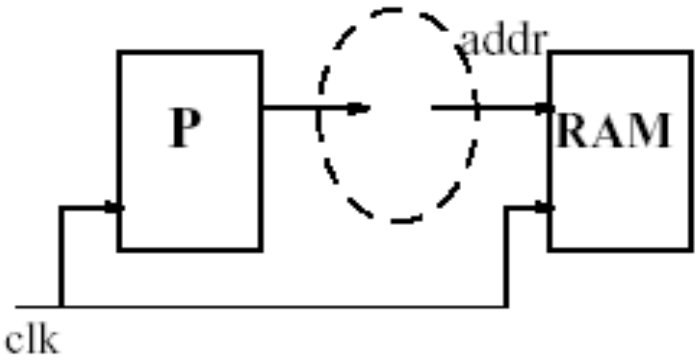


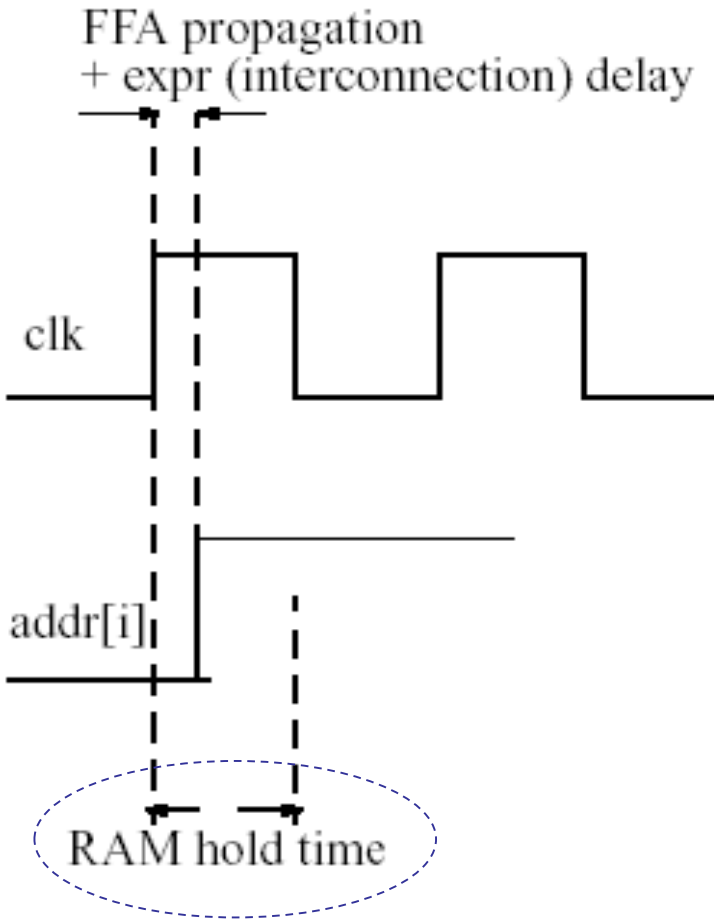
<Ref> Verilog Styles for Synthesis of Digital Systems, Smith/Franzon, Prentice-Hall 2000 pp 245 ~ 246

Some Usual Suspects

Hold Violation

```
always @(posedge clk)
  begin
     RAM.select <= 1;
     expr(P.incr) <= 1;
  end
```





<Ref> Verilog Styles for Synthesis of Digital Systems, Smith/Franzon, Prentice-Hall 2000 pp 245 ~ 246