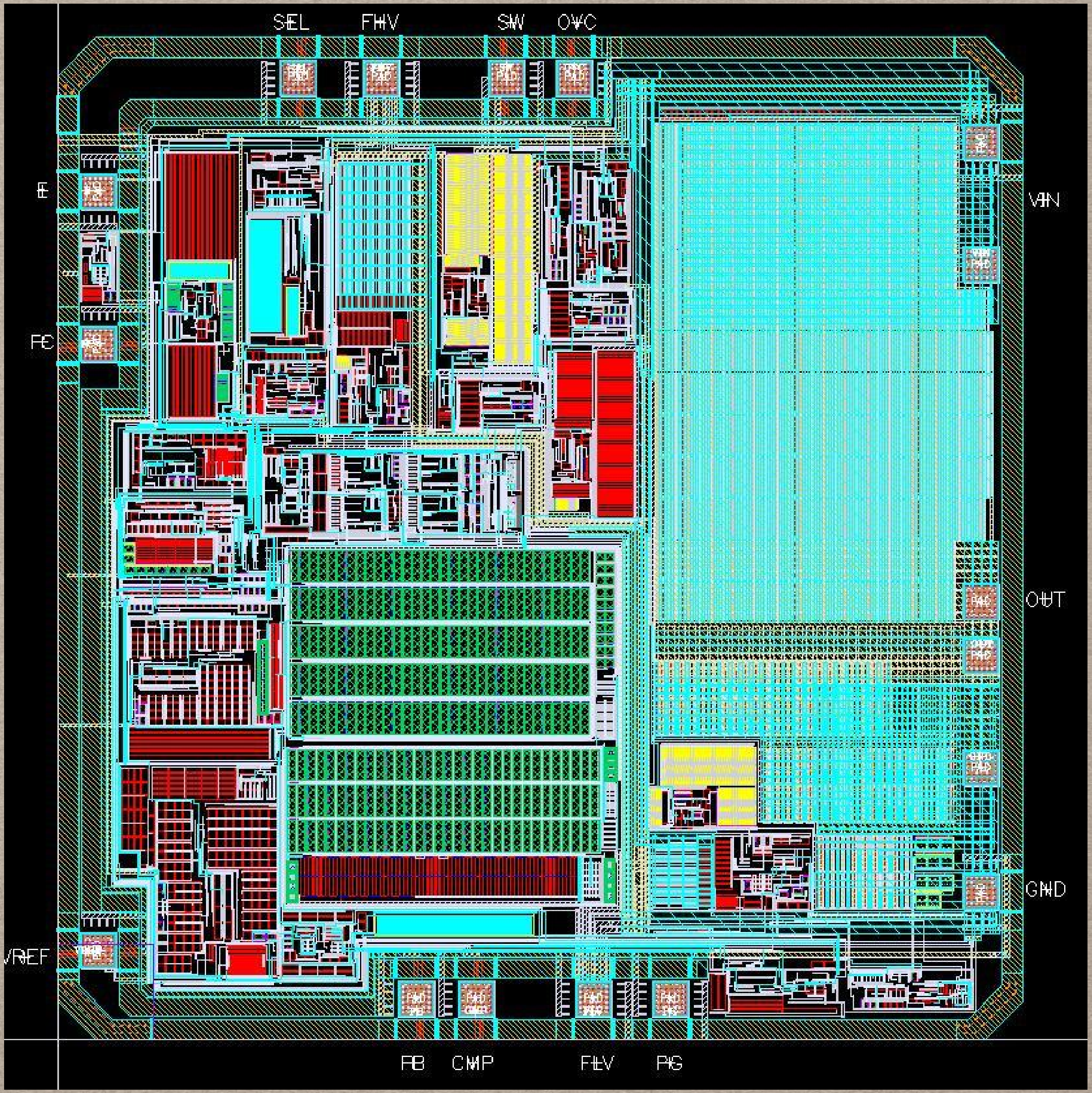
ECE4514
DIGITAL DESIGN 2

PERFORMANCE

CHAPTER 1



Going Beyond "Getting The Job Done"

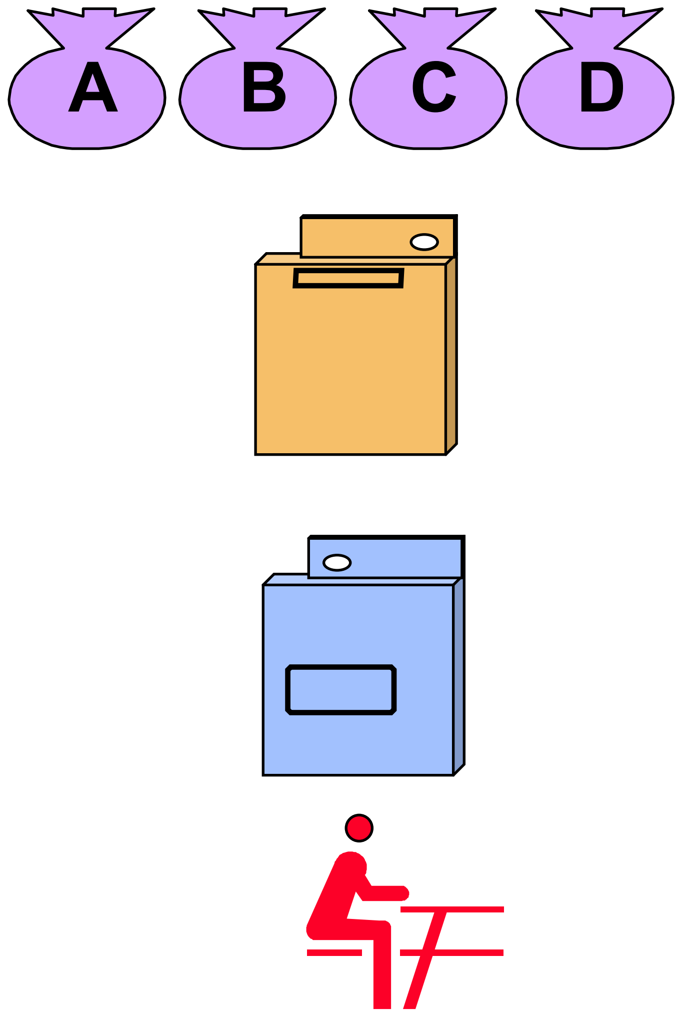
- High-Throughput Architectures
- Low-Latency Architectures
- Timing Optimizations
  - Adding Register Layers
  - Parallel Structures
  - Flatten Logic Structures
  - Register Balancing
  - Reorder Paths

Throughput vs. Latency

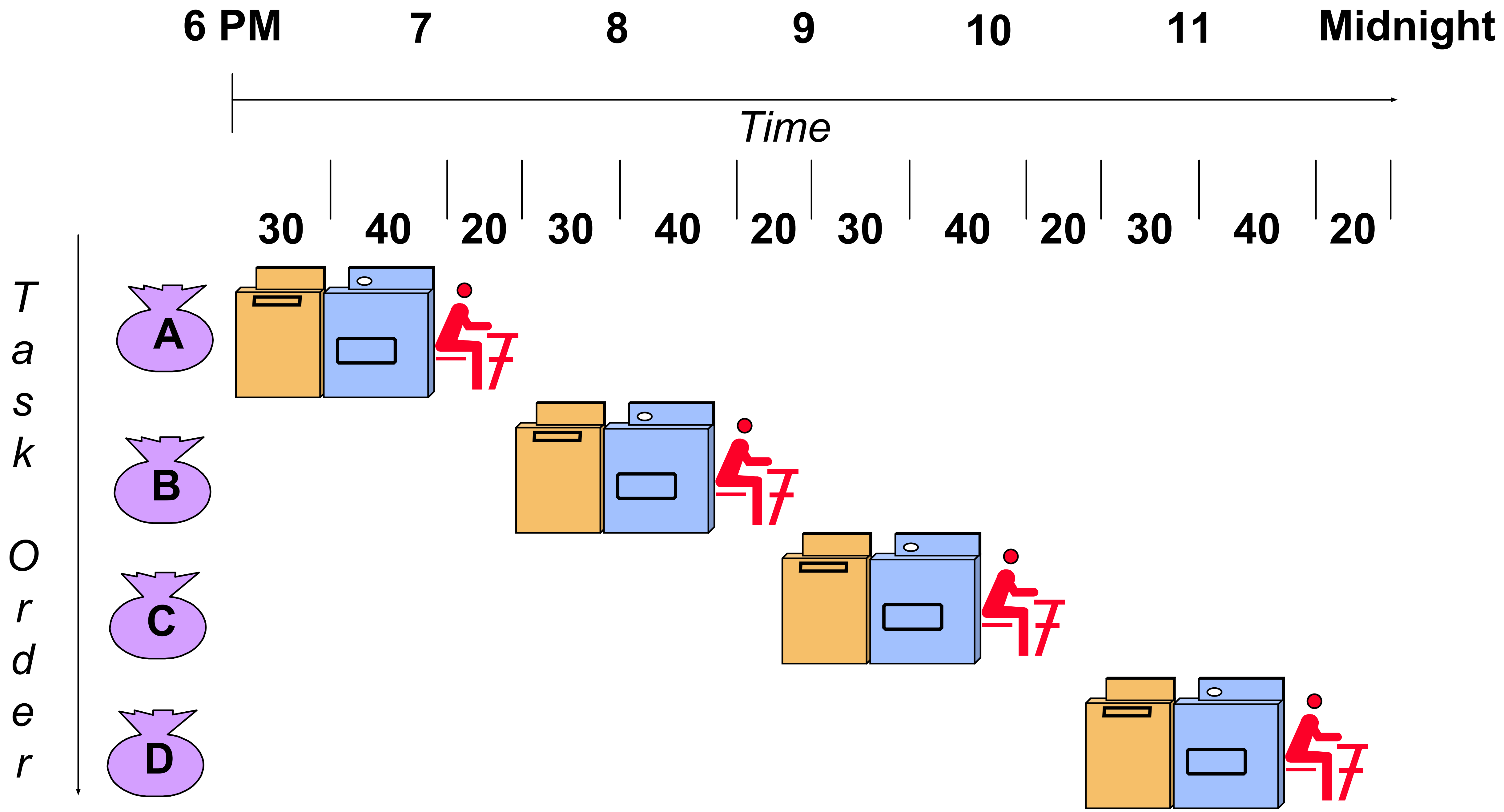
Laundry Day

Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold

- Washer takes 30 minutes
- Dryer takes 40 minutes
- Folding takes 20 minutes

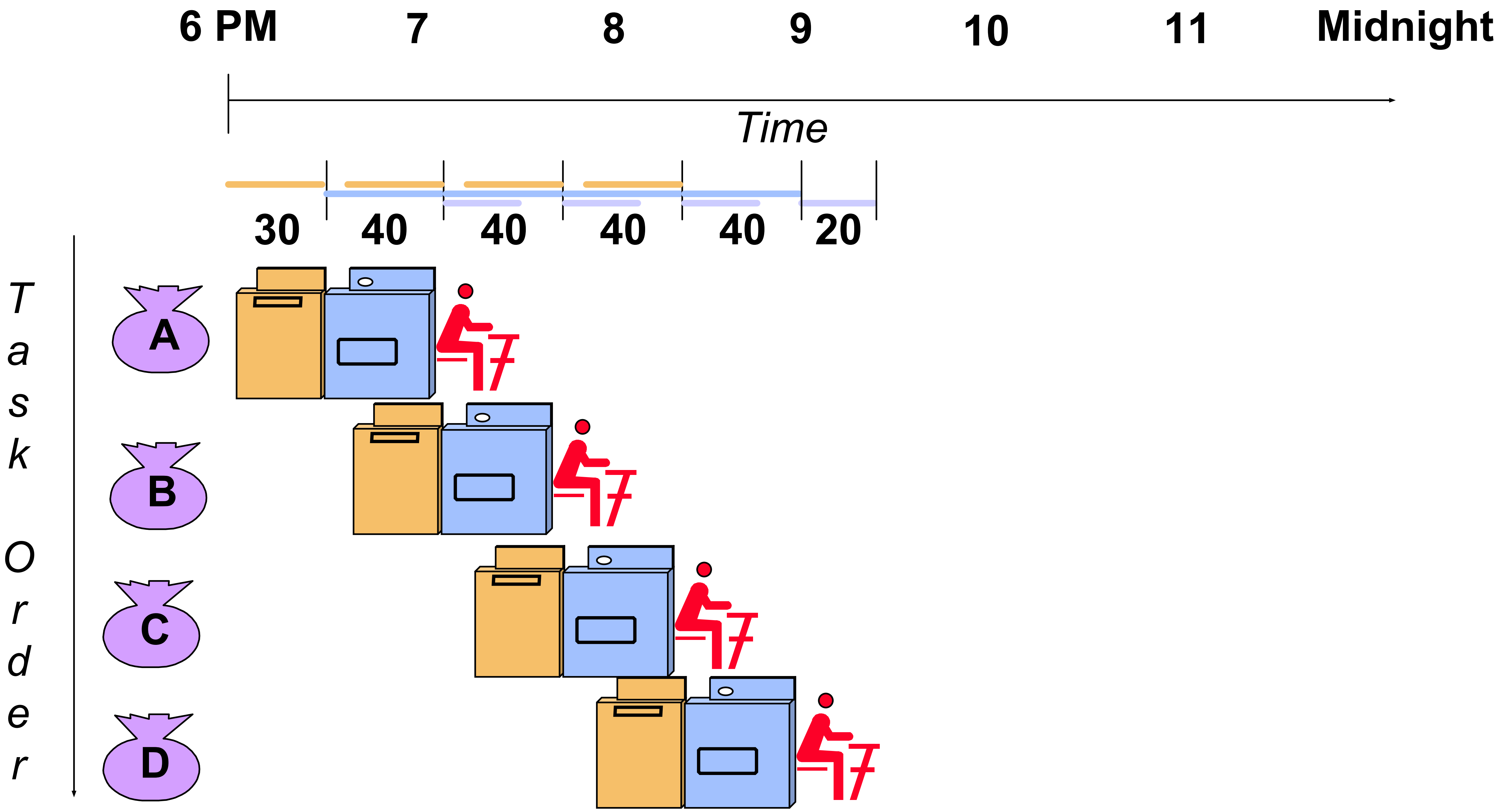


Sequential Laundry



- Sequential laundry takes 6 hours for 4 loads
- With pipelining, how long would laundry take?

Pipelined Laundry
Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

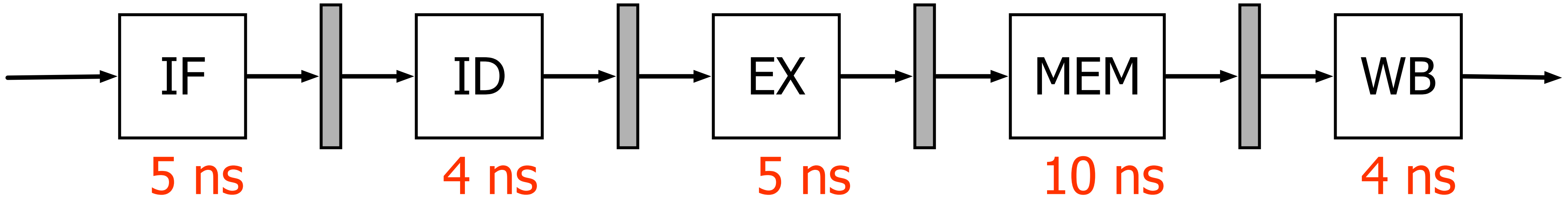
Key Definitions

Pipelining is a key implementation technique used to build fast processors. It allows the execution of multiple instructions to overlap in time.

A pipeline within a processor is similar to a car assembly line. Each assembly station is called a *pipe stage* or a *pipe segment.*

The throughput of an instruction pipeline is the measure of how often an instruction exits the pipeline.

Pipeline Throughput and Latency

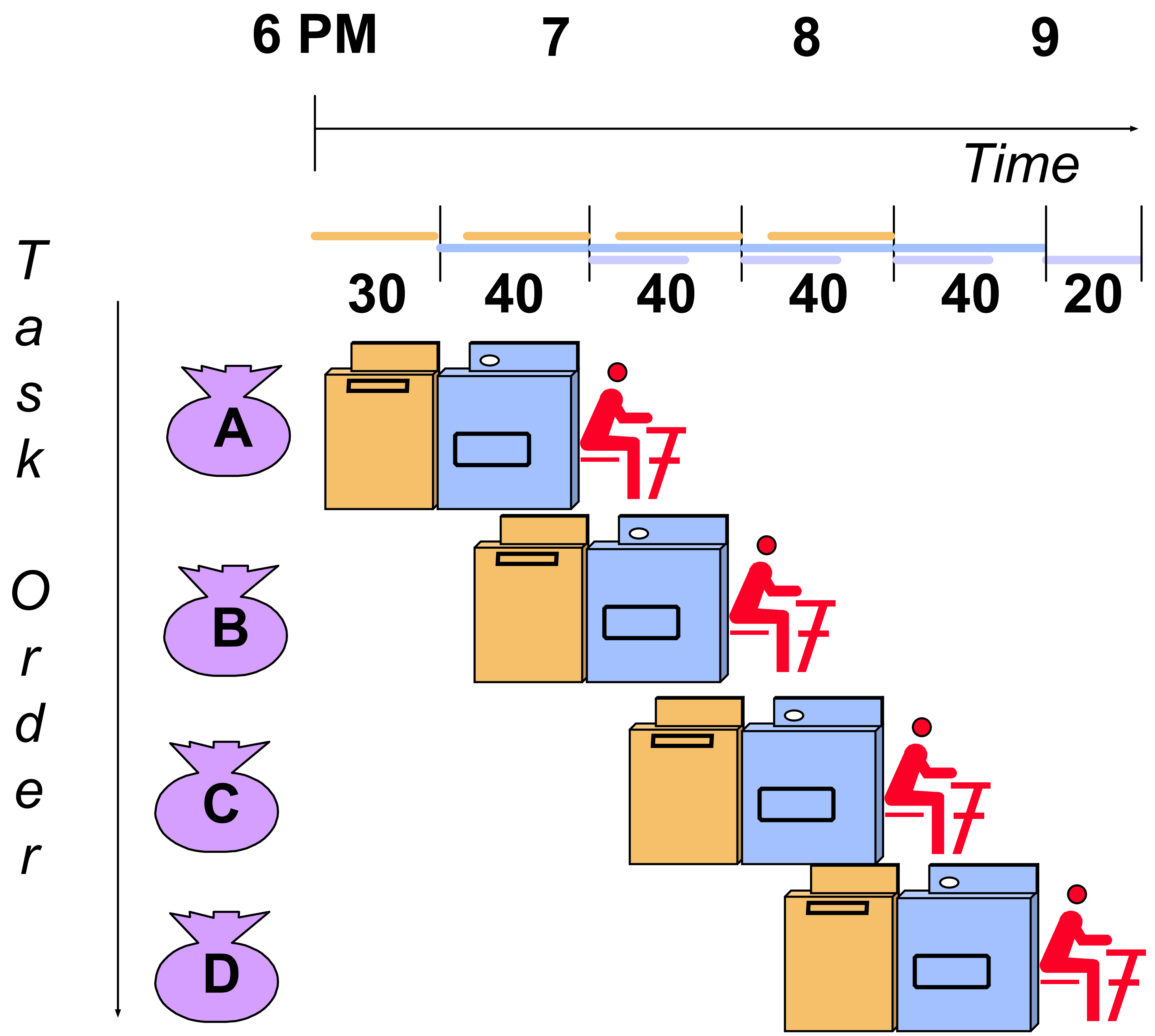


Consider the pipeline above with the indicated delays. We want to know what is the *pipeline throughput* and the *pipeline latency*.

Pipeline throughput: instructions completed per second.

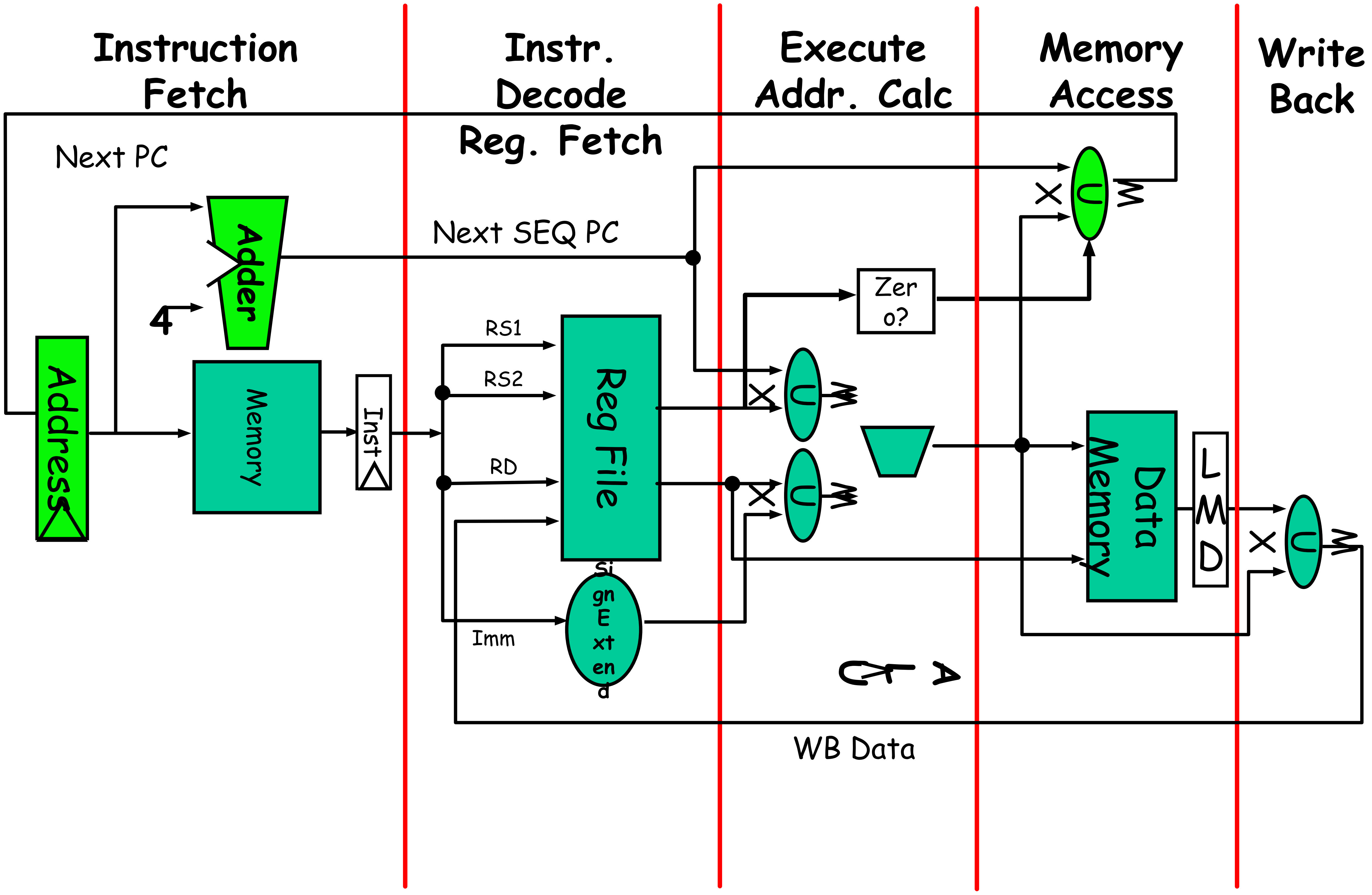
Pipeline latency: how long does it take to execute a single instruction in the pipeline.

Pipelining Lessons



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

5 Steps of MIPS Datapath



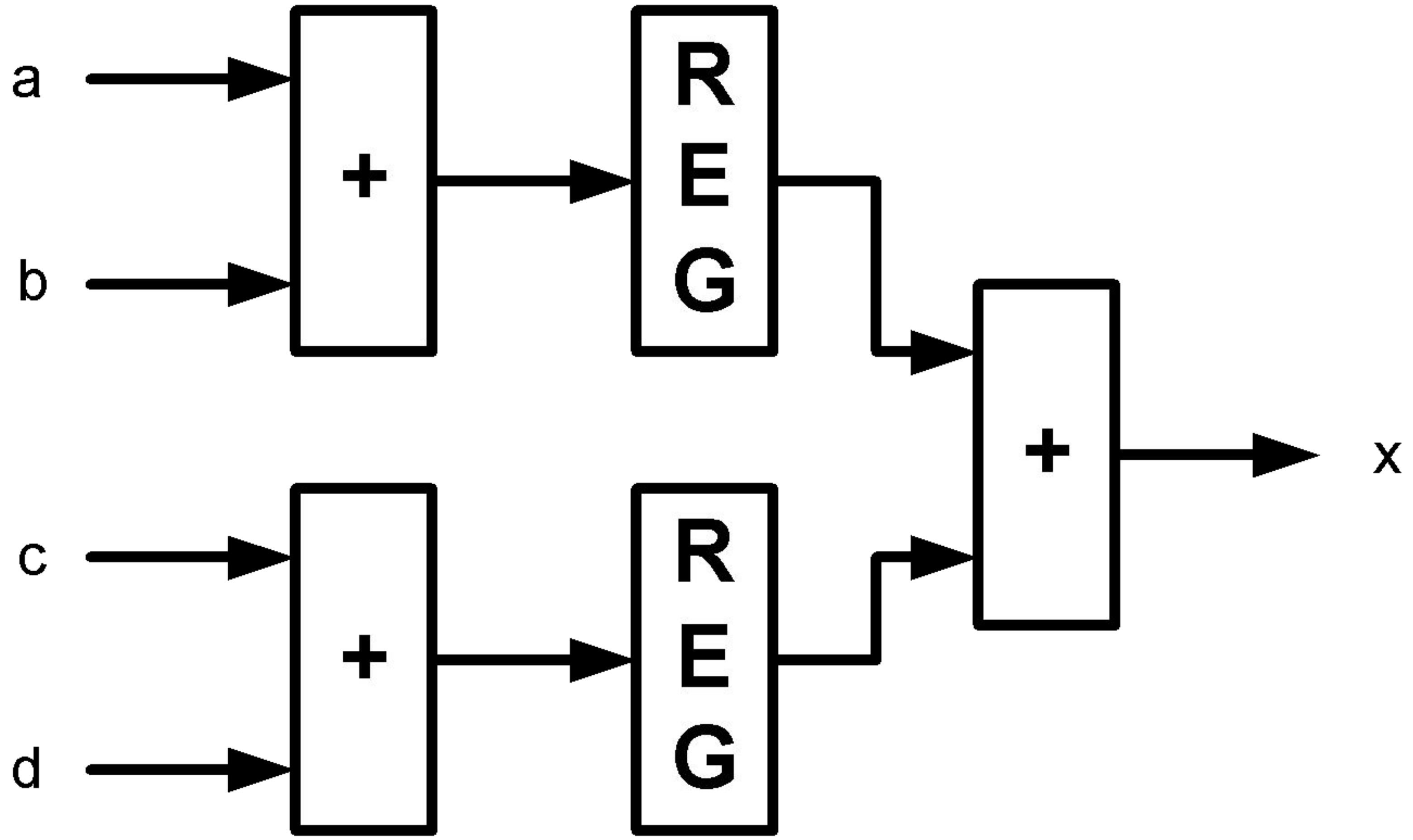
Example 1: Design for low latency (parallelism)

$$\mathbf{X = a + b + c + d}$$

All are 32-bit values

Example 1: Design for delay

- **X = a + b + c + d**



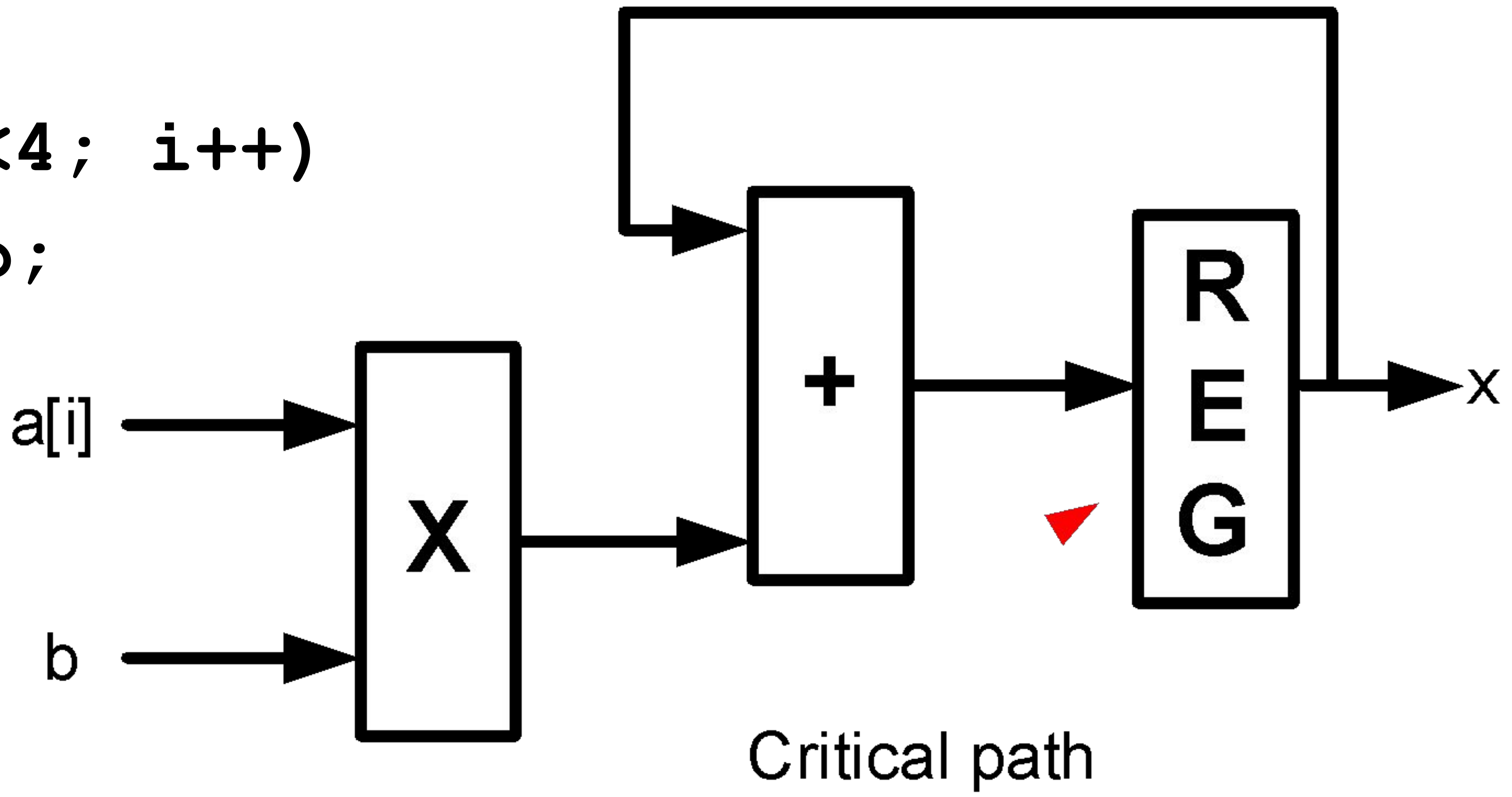
Delay = 1*add + Reg

Latency = 2 cycles

Throughput = *N* bits/clock

Example 2: Design for delay

```
x=0;
for (i=0; i<4; i++)
   x+= a[i]*b;
```



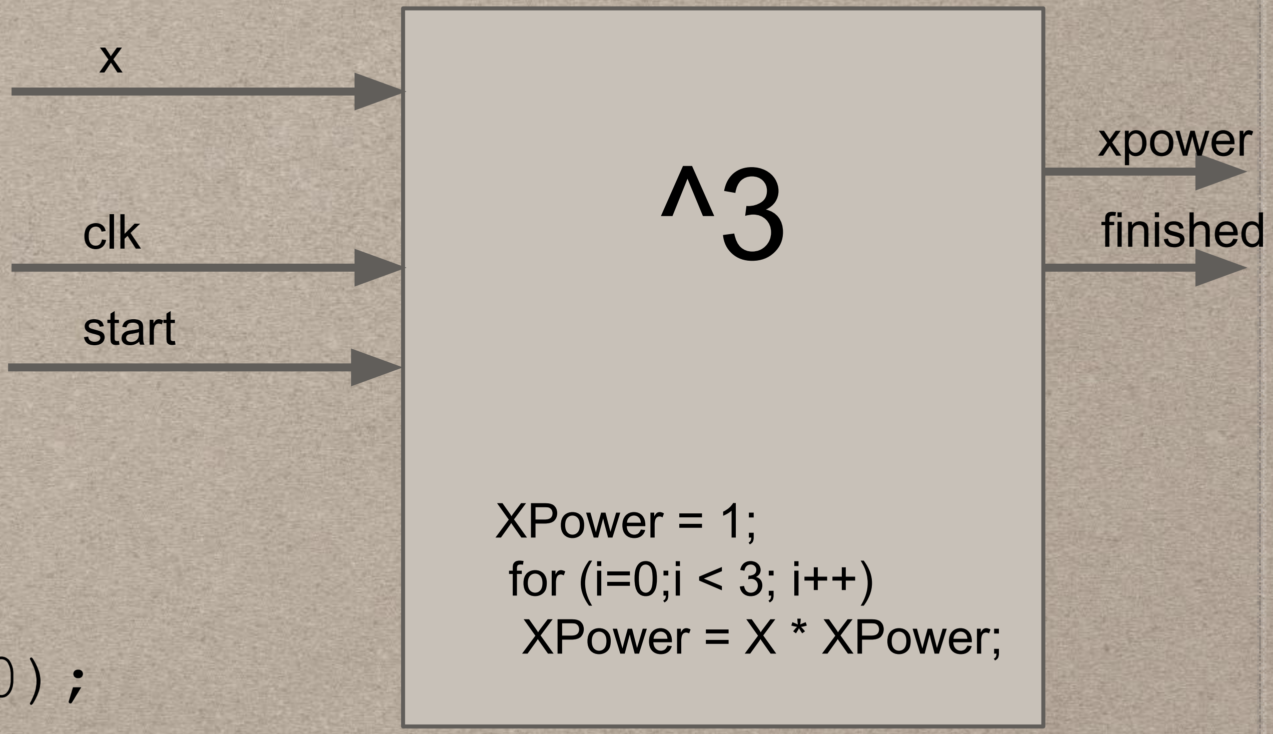
Delay: 1*Mul + 1 Add

Latency: 4 cycles

Throughput: *N* bits/4 cycles

High-Throughput Architecture -- Example 1

```
module power3(
     output reg [7:0] xpower,
     output finished,
     input  [7:0] x,
     input  clk, start);
reg    [7:0] ncount;

assign finished = (ncount == 0);
always@(posedge clk)
   if(start) begin
      xpower <= x;
      ncount <= 2;
   end
    else if(!finished) begin
      ncount <= ncount - 1;
      xpower <= xpower * x;
  end
 endmodule
```



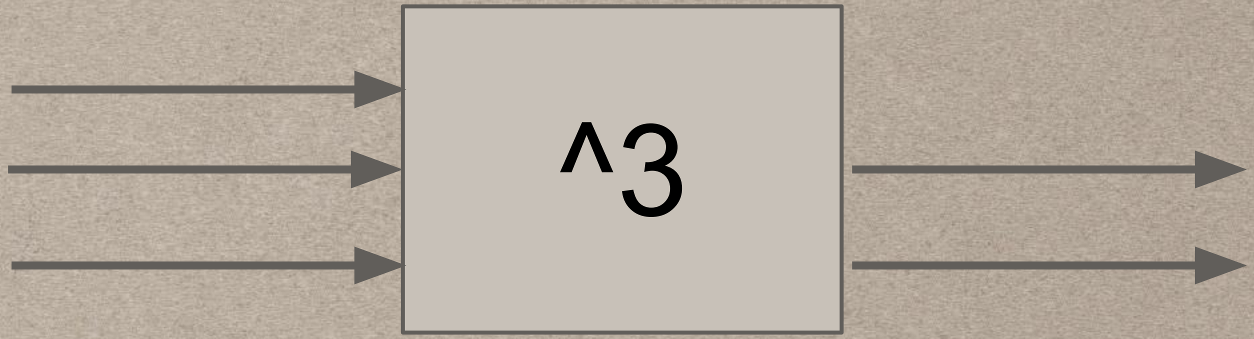
```
XPower = 1;
 for (i=0;i < 3; i++)
  XPower = X * XPower;
```

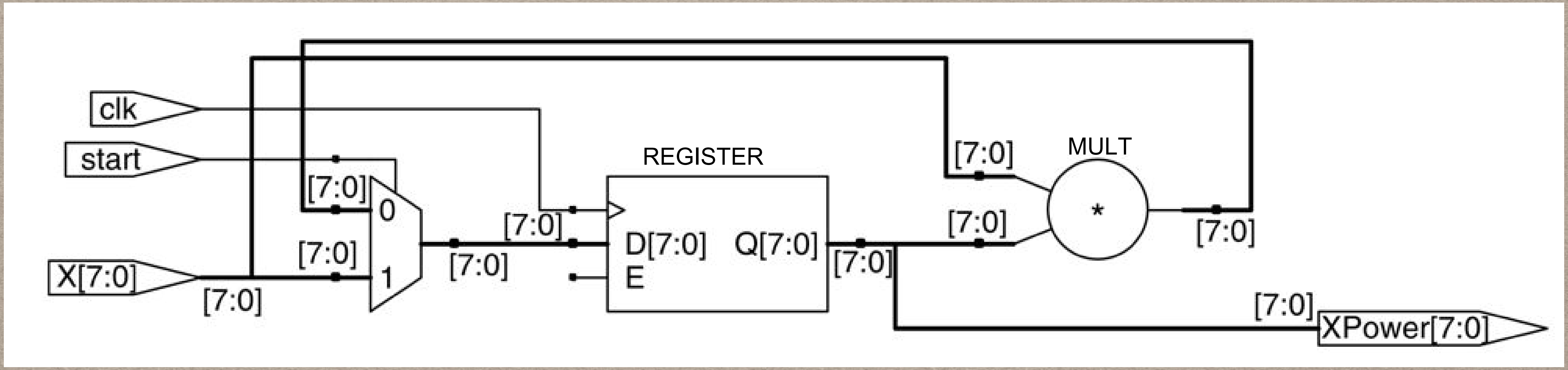
THROUGHPUT =

LATENCY =

CYCLE TIME =

High-Throughput Architecture -- Example 1



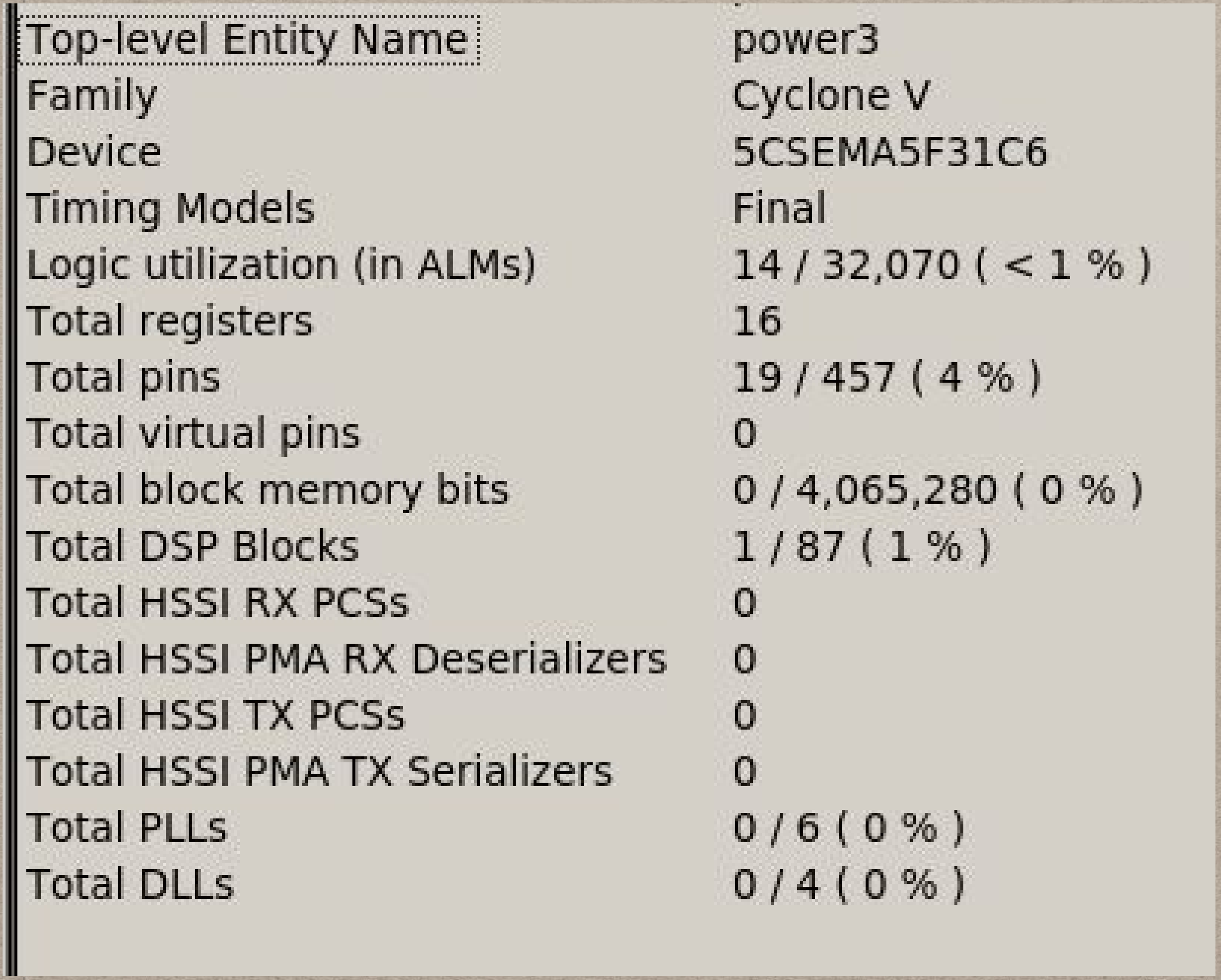


THROUGHPUT =

LATENCY =

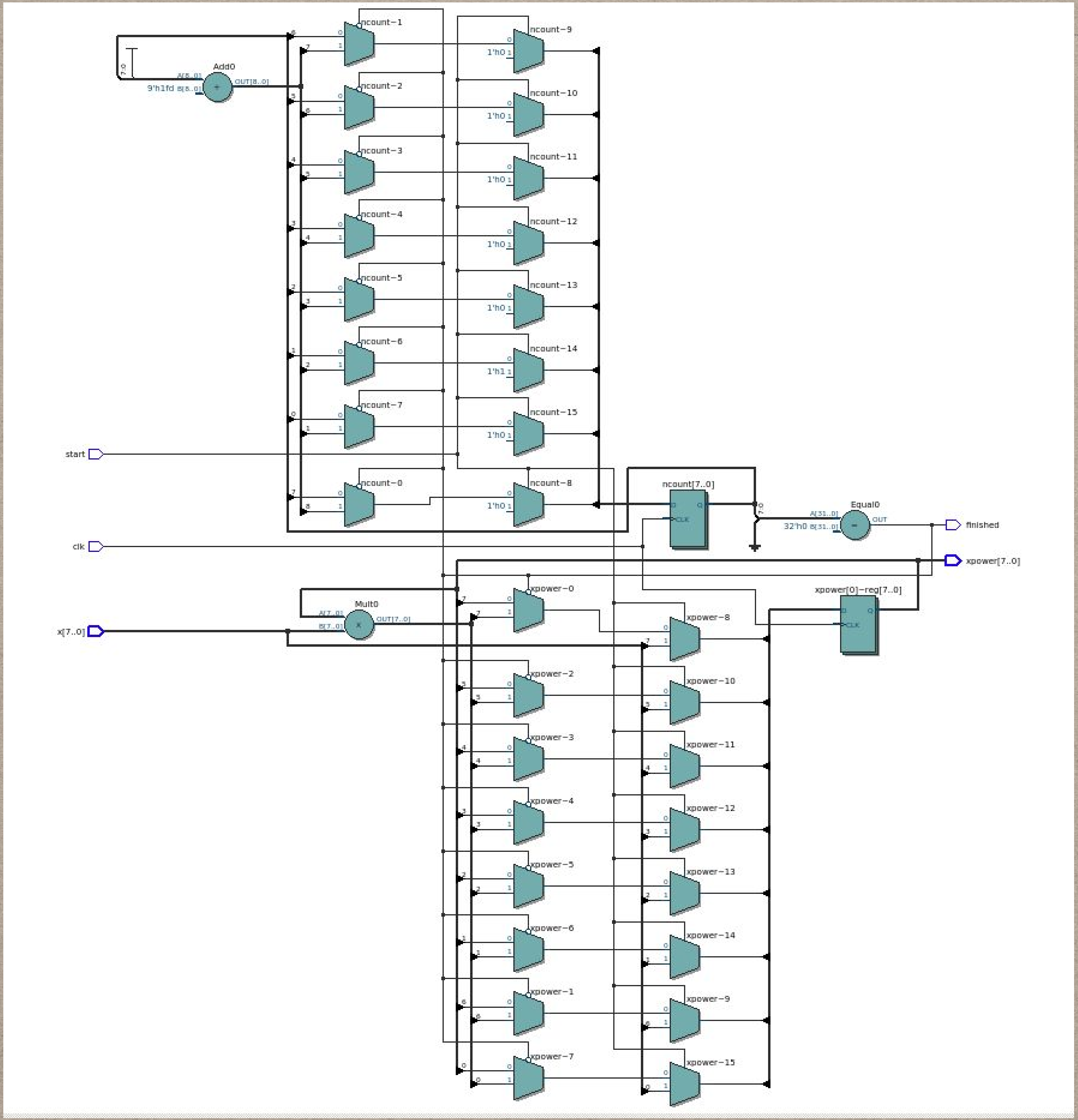
CRITICAL PATH =

ALTERA *FLOW SUMMARY* REPORT



Top-level Entity Name	power3
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	14 / 32,070 (< 1 %)
Total registers	16
Total pins	19 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	1 / 87 (1 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

- Adaptive Logic Modules (ALMs)
- **DSP block 9-bit elements** -- digital signal processing block 9-bit element
- **HSSI RX PCSs** -- high speed serial interface physical coding sub-layer receiver channels
- **HSSI PMA RX Deserializers** -- high speed serial interface physical media attachment receiver channels
- **HSSI TX PCSs** -- high speed serial interface physical coding sub-layer transmitter channels
- **HSSI PMA TX Serializers** -- high speed serial interface physical media attachment transmitter channels.
- **PLLs**
- **DLLs**



Altera RTL Mapping of Power3

High-Throughput Architecture -- Example 2

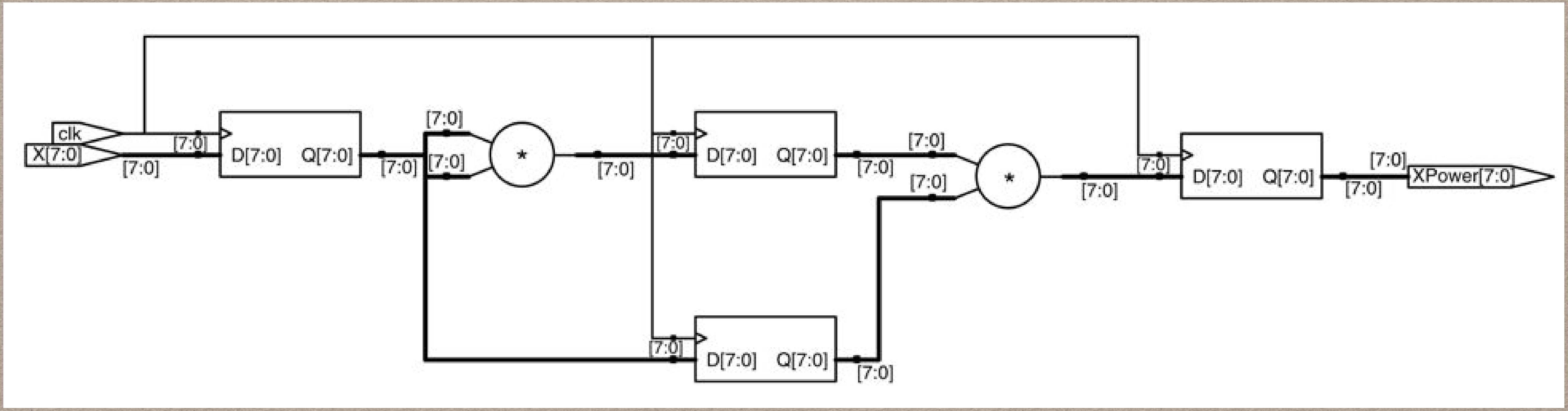
```
module power3b(
     output reg [7:0] XPower,
     input            clk,
     input      [7:0] X
     );
     reg        [7:0] XPower1, XPower2;
     reg        [7:0] X1, X2;
     always @(posedge clk) begin
       // Pipeline stage 1
       X1      <= X;
       XPower1 <= X;
       // Pipeline stage 2
       X2      <= X1;
       XPower2 <= XPower1 * X1;
       // Pipeline stage 3
       XPower <= XPower2 * X2;
     end
endmodule
```

THROUGHPUT =

LATENCY =

CRITICAL PATH =

High-Throughput Architecture -- Example 2

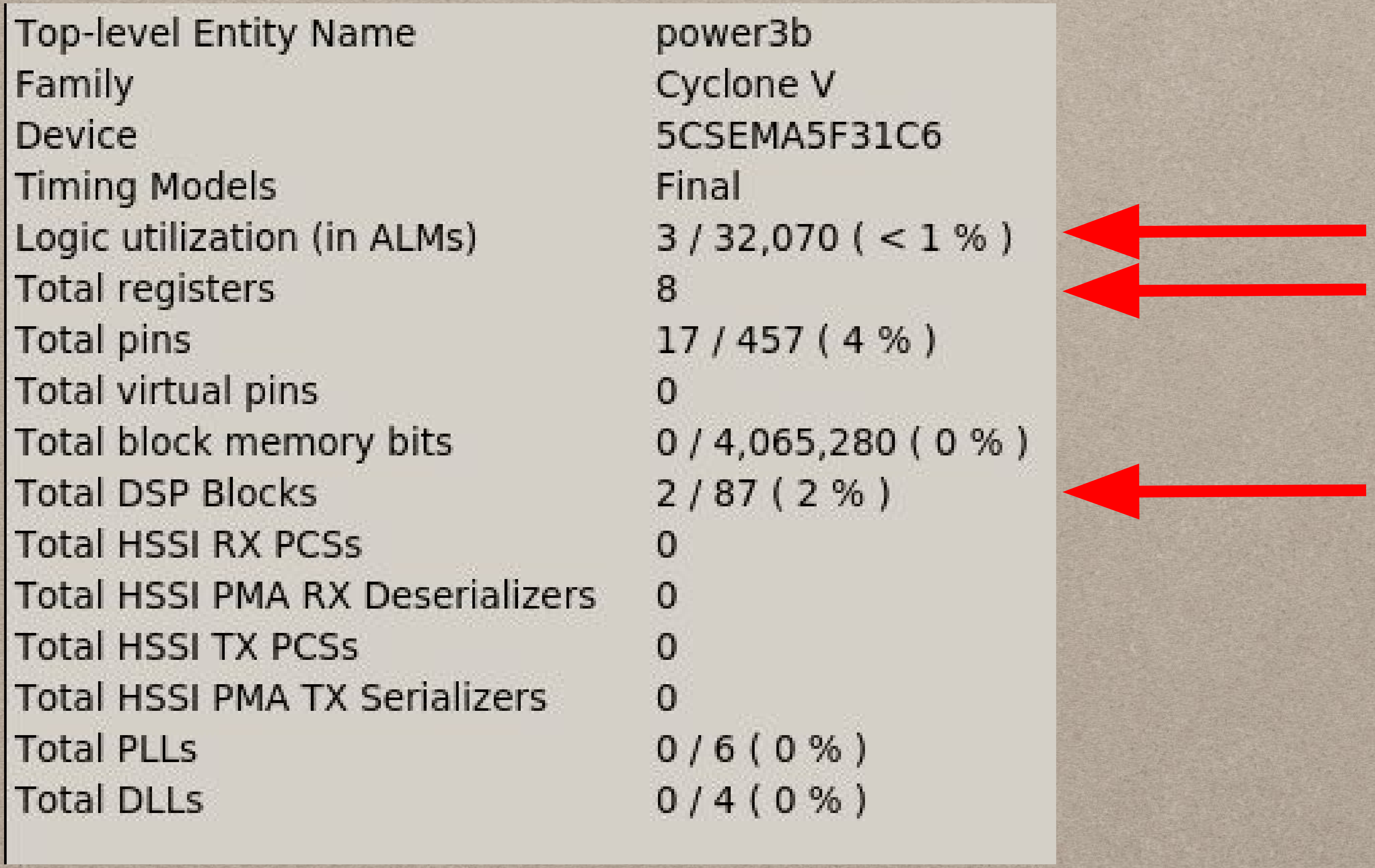


THROUGHPUT =

LATENCY =

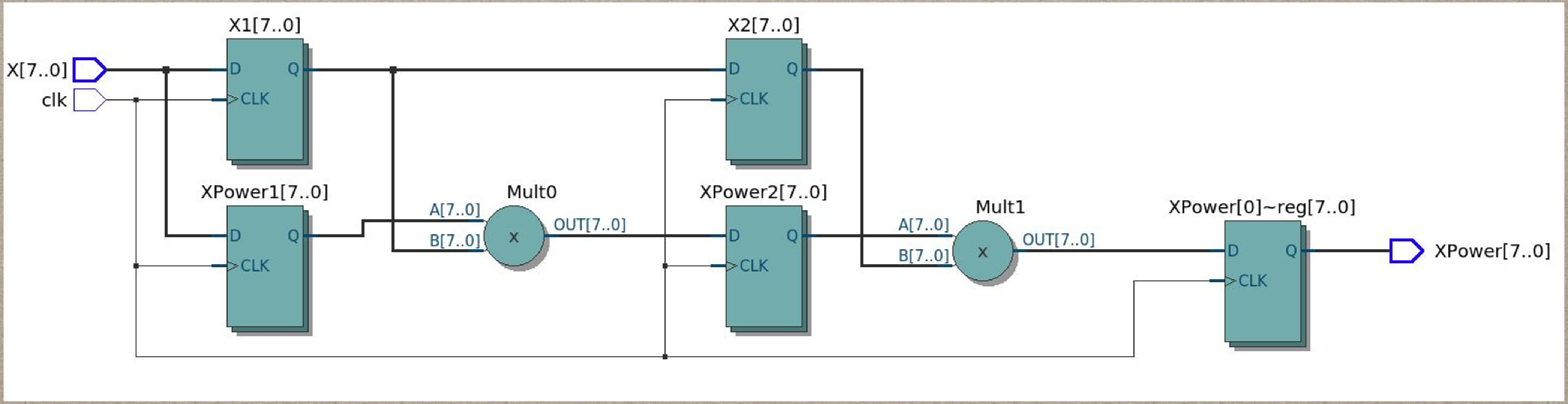
CYCLE TIME =

Altera *Flow Summary* Report

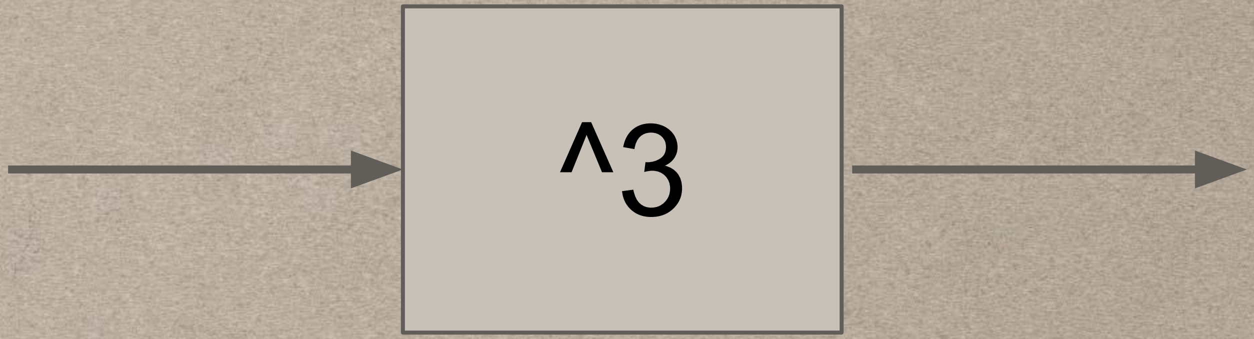


Top-level Entity Name	power3b
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	3 / 32,070 (< 1 %)
Total registers	8
Total pins	17 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	2 / 87 (2 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Altera Mapping of Power3b



Low Latency Architecture -- Example 3



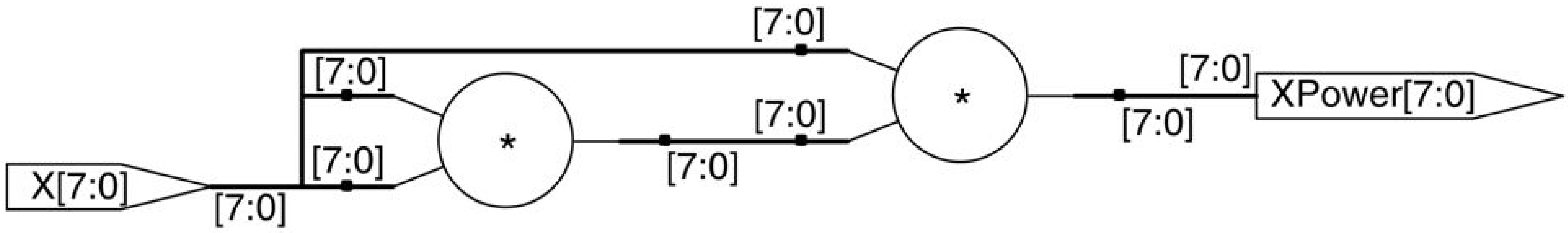
```
module power3(
     output [7:0] XPower,
     input  [7:0] X
     );
reg    [7:0] XPower1, XPower2;
reg    [7:0] X1, X2;
   assign XPower = XPower2 * X2;
   always @*
   begin
     X1 =X; XPower1 = X;
   end
   always @* begin
     X2      = X1;
     XPower2 = XPower1*X1;
   end
endmodule
```

THROUGHPUT =

LATENCY =

CRITICAL PATH =

High-Throughput Architecture -- Example 3

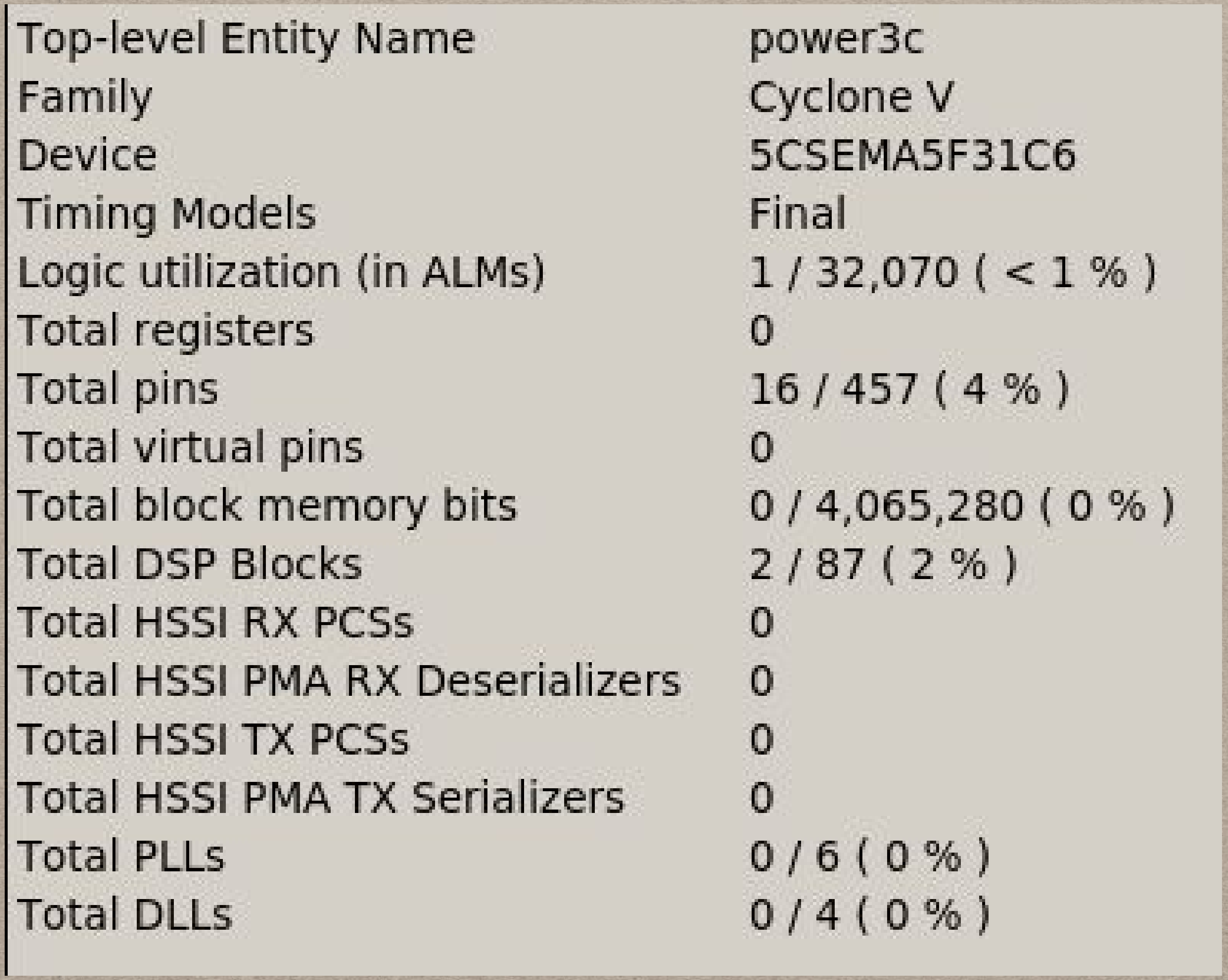


THROUGHPUT =

LATENCY =

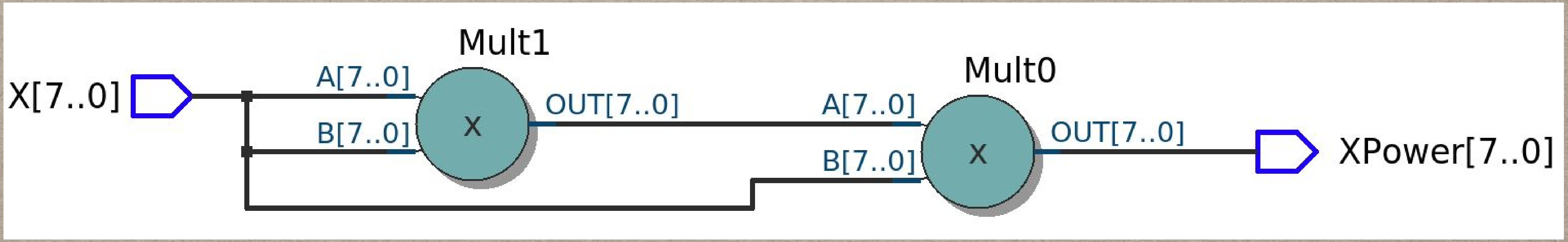
CYCLE TIME =

Altera *Flow Summary* Report



Top-level Entity Name	power3c
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0
Total pins	16 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	2 / 87 (2 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Altera Mapping of Power3c

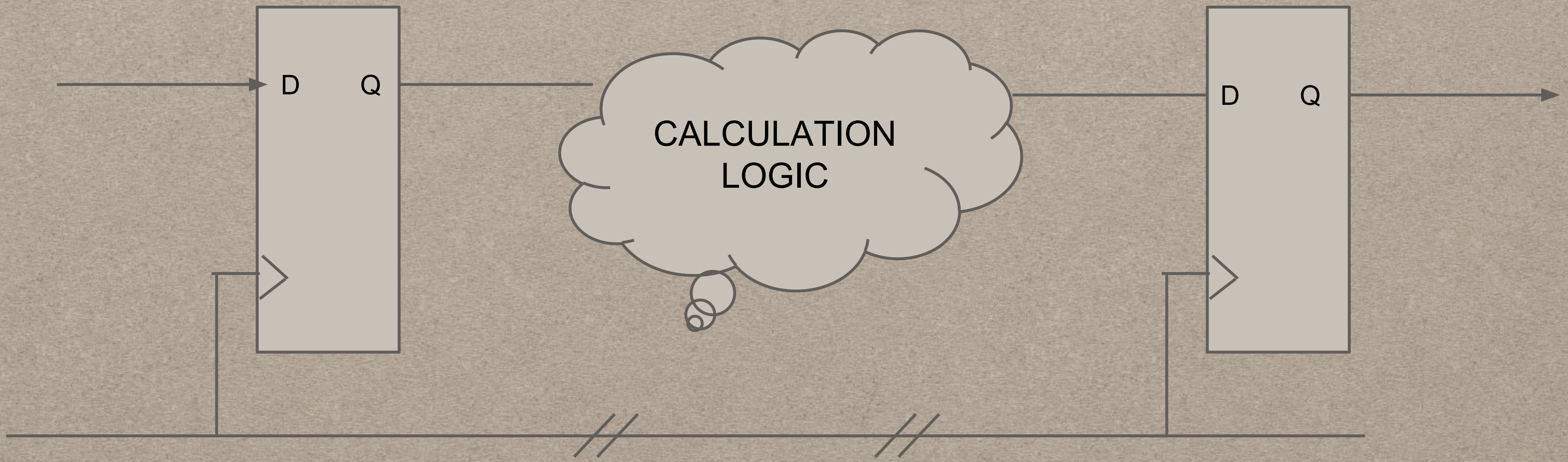


Results

DESIGN	FMAX	ALM	DSP	REGS	THRU-PUT
power3a (1st example)	217 MHz	14	1	16	578 Mbps
power3b (2nd example)	257 MHz	3	2	8	2056 Mbps
power3c (no clock)	---	1	2	0	----
power3d (3rd example)	178 MHz	1	2	0	1424 Mbps
power3e (2nd redone with piped mults)	469 MHz	5	2	16	3752 Mbps

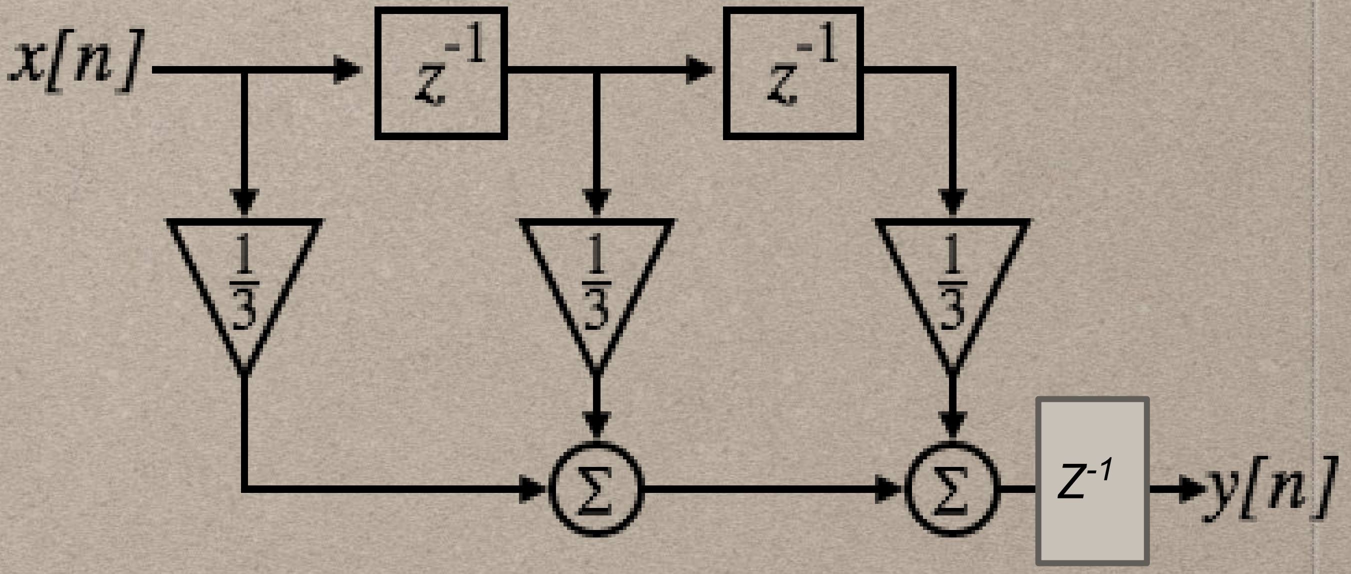
TIMING

$$F_{\max} = \frac{1}{T_{\text{clk}-q} + T_{\text{log ic}} + T_{\text{routing}} + T_{\text{setup}} - T_{\text{skew}}}$$



TIMING -- EXAMPLE: FINITE IMPULSE FILTER (FIR)

```
module fir(
     output [7:0] Y,
     input  [7:0] A, B, C, X,
     input        clk,

input        validsample);
  reg    [7:0] X1, X2, Y;
  always @(posedge clk)
    if(validsample) begin
      X1 <= X;
      X2 <= X1;
      Y <= A* X+B* X1+C* X2;
    end
endmodule
```



THROUGHPUT =

LATENCY =

CRITICAL PATH =

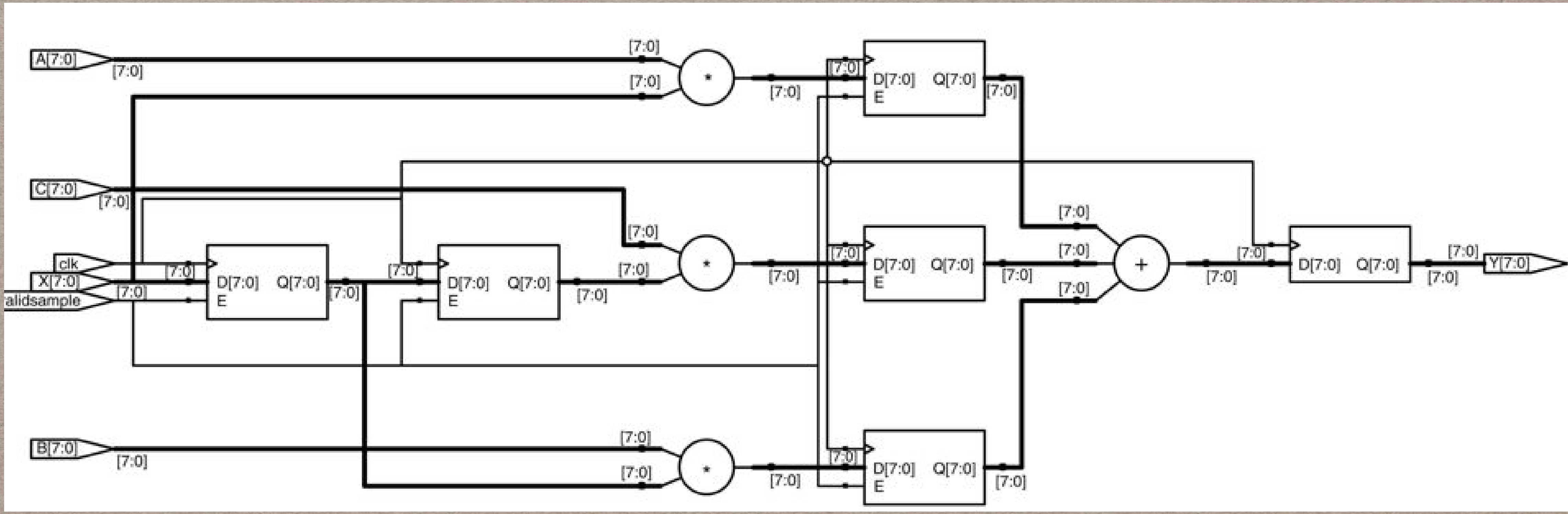
Timing -- Add Register Layers

```
module fir( ---- );
     reg    [7:0] X1, X2, Y;
     reg    [7:0] prod1, prod2, prod3;
     always @ (posedge clk) begin
       if(validsample) begin
         X1 <= X;
         X2 <= X1;
         prod1 <= A * X;
         prod2 <= B * X1;
         prod3 <= C * X2;
       end
       Y <= prod1 + prod2 + prod3;
     end
endmodule
```

THROUGHPUT =

LATENCY =

CRITICAL PATH =



Timing -- Add More Register Layers

```
module fir( ---- );
     reg    [7:0] X1, X2, Y;
     reg    [7:0] prod1, prod2, prod3, add1;
     always @ (posedge clk) begin
       if(validsample) begin
         X1 <= X;
         X2 <= X1;
         prod1 <= A * X;
         prod2 <= B * X1;
         prod3 <= C * X2;
          add1 <= prod1 + prod2;
       end
       Y <= add1 + prod3;
     end
endmodule
```

THROUGHPUT =

LATENCY =

CRITICAL PATH =

Timing -- Parallel Structures

```
module power3(
   output [7:0] XPower,
   input  [7:0] X,
   input        clk);
reg    [7:0] XPower1;
// partial product registers
reg    [3:0] XPower2_ppAA, XPower2_ppAB,
       XPower2_ppBB;
reg    [3:0] XPower3_ppAA, XPower3_ppAB,
       XPower3_ppBB;
reg    [7:0] X1, X2;
wire   [7:0] XPower2;
// nibbles for partial products (A is MS
// nibble, B is LS nibble)
wire   [3:0] XPower1_A = XPower1[7:4];
wire   [3:0] XPower1_B = XPower1[3:0];
wire   [3:0] X1_A      = X1[7:4];
wire   [3:0] X1_B      = X1[3:0];
wire   [3:0] XPower2_A = XPower2[7:4];
wire   [3:0] XPower2_B = XPower2[3:0];
wire   [3:0] X2_A      = X2[7:4];
wire   [3:0] X2_B      = X2[3:0];
```

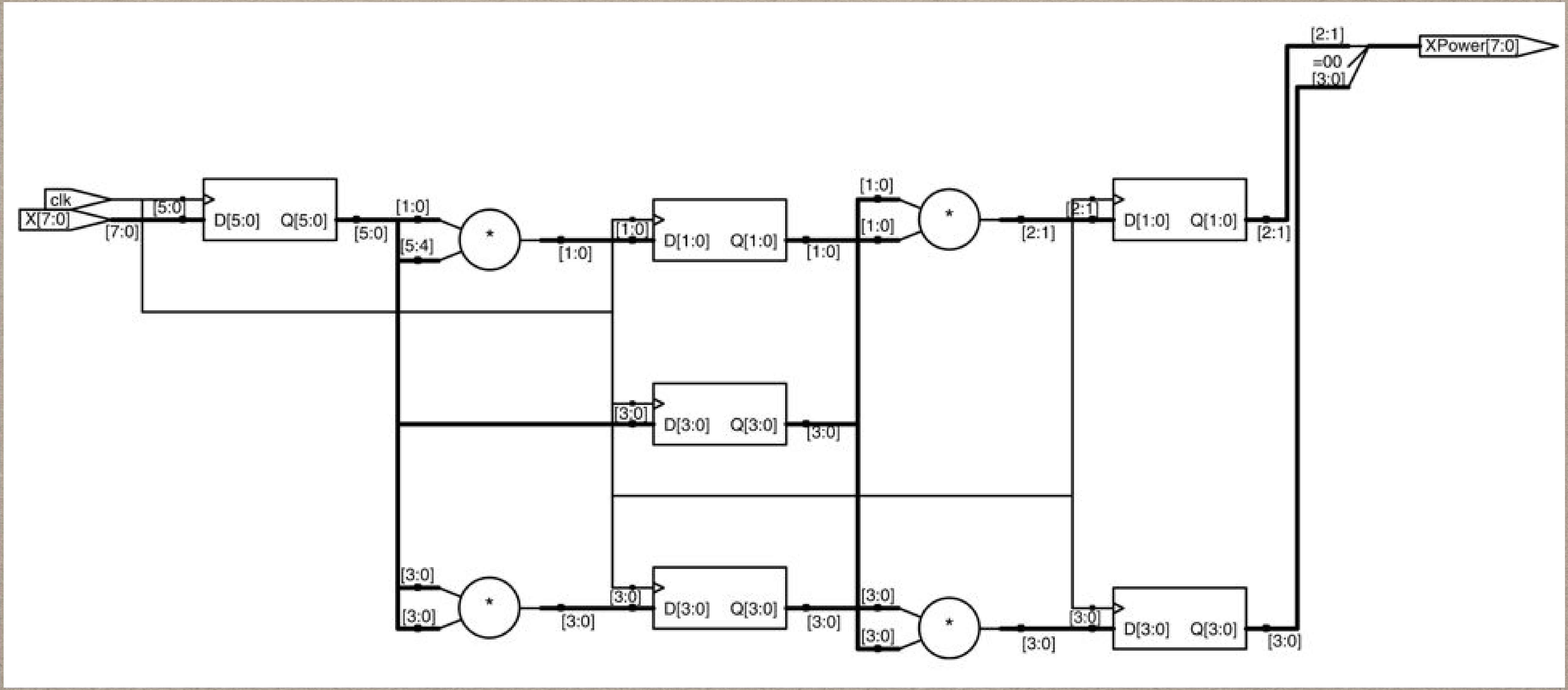
```
// assemble partial products
assign XPower = (XPower2_ppAA << 8)+
  (2*XPower2_ppAB << 4)+ XPower2_ppBB;
assign XPower2 = (XPower3_ppAA << 8)+
  (2*XPower3_ppAB << 4)+ XPower3_ppBB;
always @(posedge clk) begin
  // Pipeline stage 1
  X1           <= X;
  XPower1      <= X;
  // Pipeline stage 2
  X2           <= X1;
  // create partial products
  XPower2_ppAA <= XPower1_A * X1_A;
  XPower2_ppAB <= XPower1_A * X1_B;
  XPower2_ppBB <= XPower1_B * X1_B;
  // Pipeline stage 3
  // create partial products
  XPower3_ppAA <= XPower2_A * X2_A;
  XPower3_ppAB <= XPower2_A * X2_B;
  XPower3_ppBB <= XPower2_B * X2_B;
end
endmodule
```

TIMING -- PARALLEL STRUCTURES

THROUGHPUT =

LATENCY =

CRITICAL PATH =



TIMING -- FLATTEN LOGIC STRUCTURES

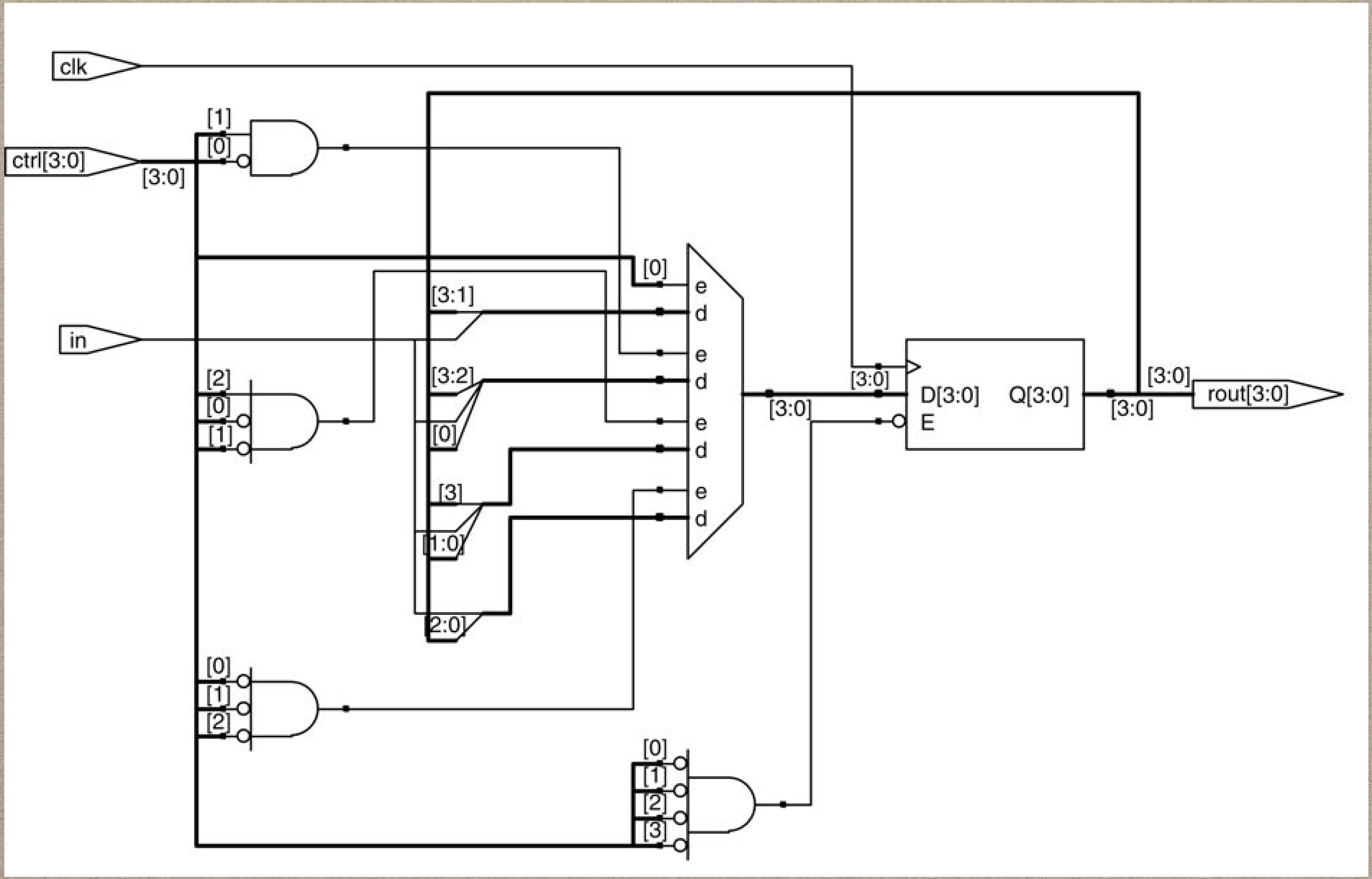
DEMULTIPLEXING:
DIRECT ‘in’ TO ONE OF FOUR REGISTERS
DEPENDING UPIN ‘ctrl’

ASSUMED :
COUT IS *ONE-HOT ENCODED*

```
module regwrite(
     output reg [3:0] rout,
     input            clk, in,
     input      [3:0] ctrl);

    always @(posedge clk)
       if(ctrl[0])      rout[0] <= in;
       else if(ctrl[1]) rout[1] <= in;
       else if(ctrl[2]) rout[2] <= in;
       else if(ctrl[3]) rout[3] <= in;
endmodule
```

NOTE THAT THE CONDITIONALS AND OUTPUT
ASSIGNMENTS ARE INDEPENDENT OF EACH
OTHER FOR EACH
IF STATEMENT

Timing -- Flatten Structures



PRIORITY ENCODER!

Timing -- Flatten Logic Structures

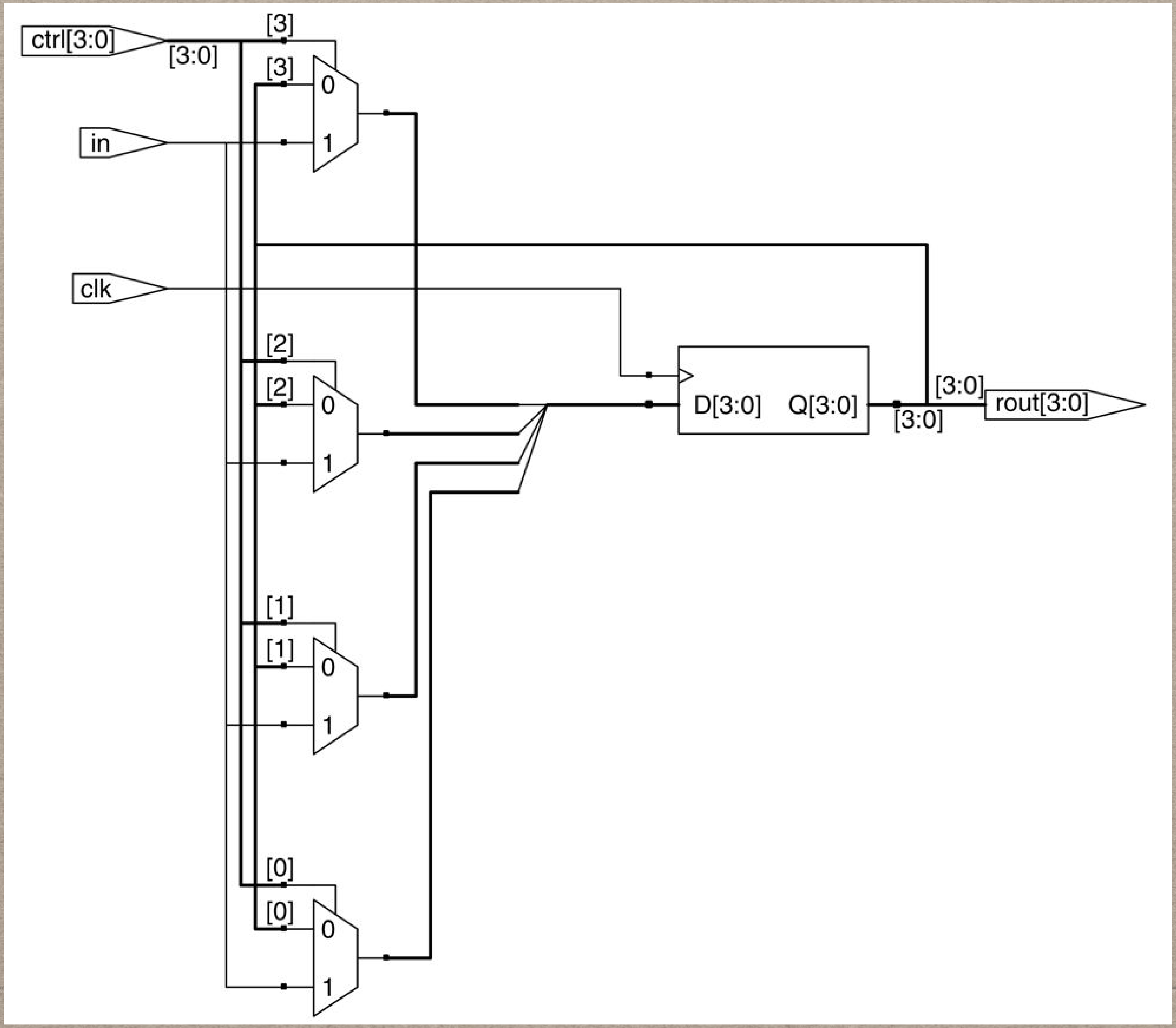
```
module regwrite(
     output reg [3:0] rout,
     input            clk, in,
     input      [3:0] ctrl);

    always @(posedge clk) begin
       if(ctrl[0]) rout[0] <= in;
       if(ctrl[1]) rout[1] <= in;
       if(ctrl[2]) rout[2] <= in;
       if(ctrl[3]) rout[3] <= in;

    end
endmodule
```

TIMING -- FLATTEN STRUCTURES

THROUGHPUT =

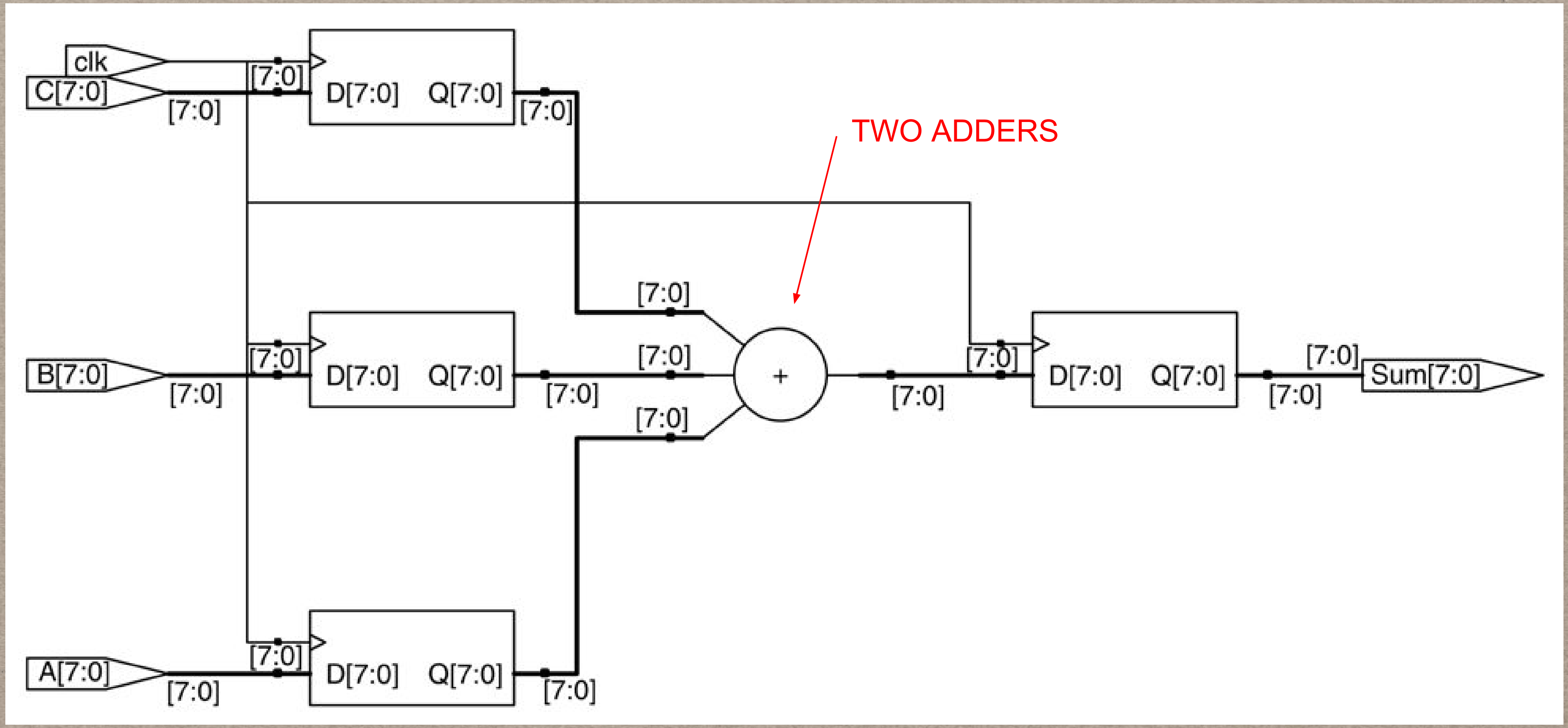
LATENCY =

CRITICAL PATH =



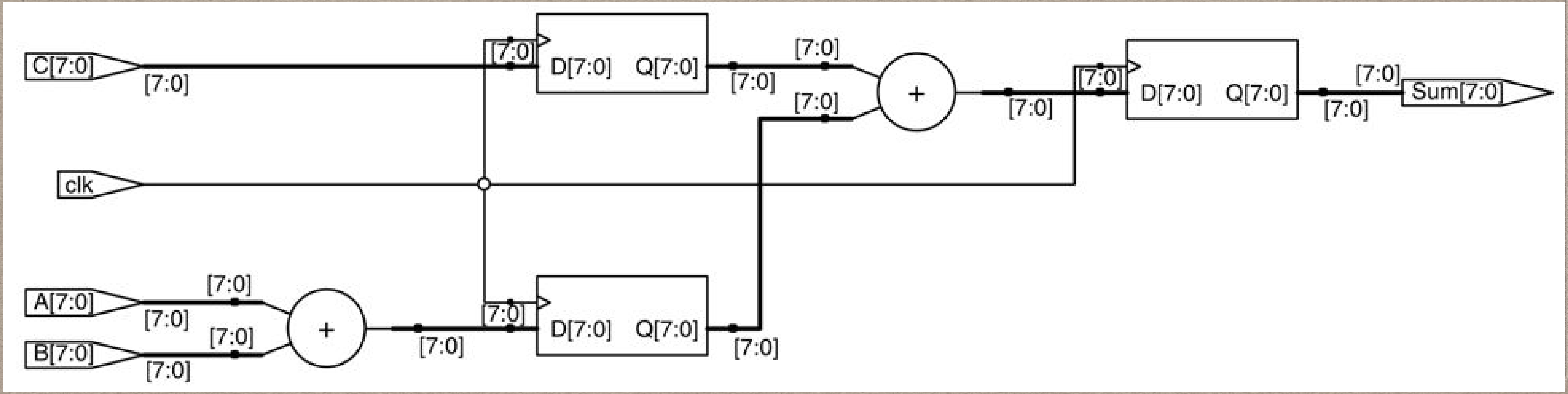
Timing -- Register Balancing

```
module adder(
     output reg [7:0] Sum,
     input      [7:0] A, B, C,
     input            clk);
     reg        [7:0] rA, rB, rC;
always @(posedge clk) begin
  rA <=A;
  rB <=B;
  rC <=C;
  Sum <=rA+rB+rC;
end
endmodule
```



Timing -- Register Balancing

```
module adder(
     output reg [7:0] Sum,
     input      [7:0] A, B, C,
     input            clk);
     reg        [7:0] rA, rB, rC;
   always @(posedge clk) begin
      rABSum <= A + B;
      rC     <= C;
      Sum    <= rABSum + rC;
   end
endmodule
```



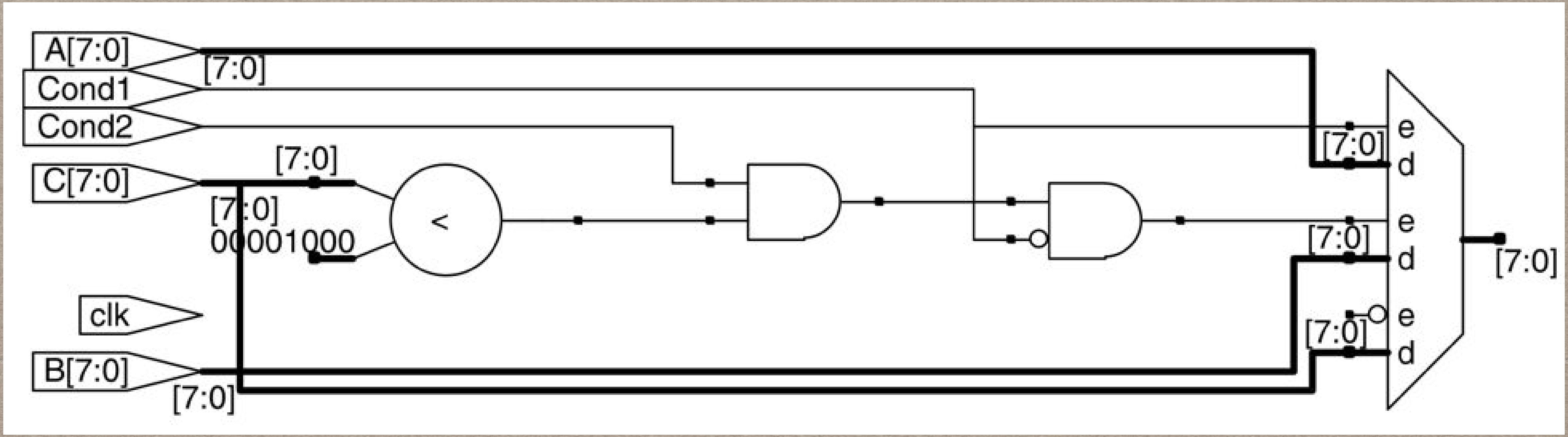
Timing -- Reorder Paths

THROUGHPUT =

LATENCY =

CRITICAL PATH =

```
module randomlogic(
     output reg [7:0] Out,
     input      [7:0] A, B, C,
     input            clk,
     input            Cond1, Cond2);
   always @(posedge clk)
       if(Cond1)
          Out <= A;
       else if(Cond2 && (C < 8))
          Out <= B;
       else
          Out <= C;
endmodule
```



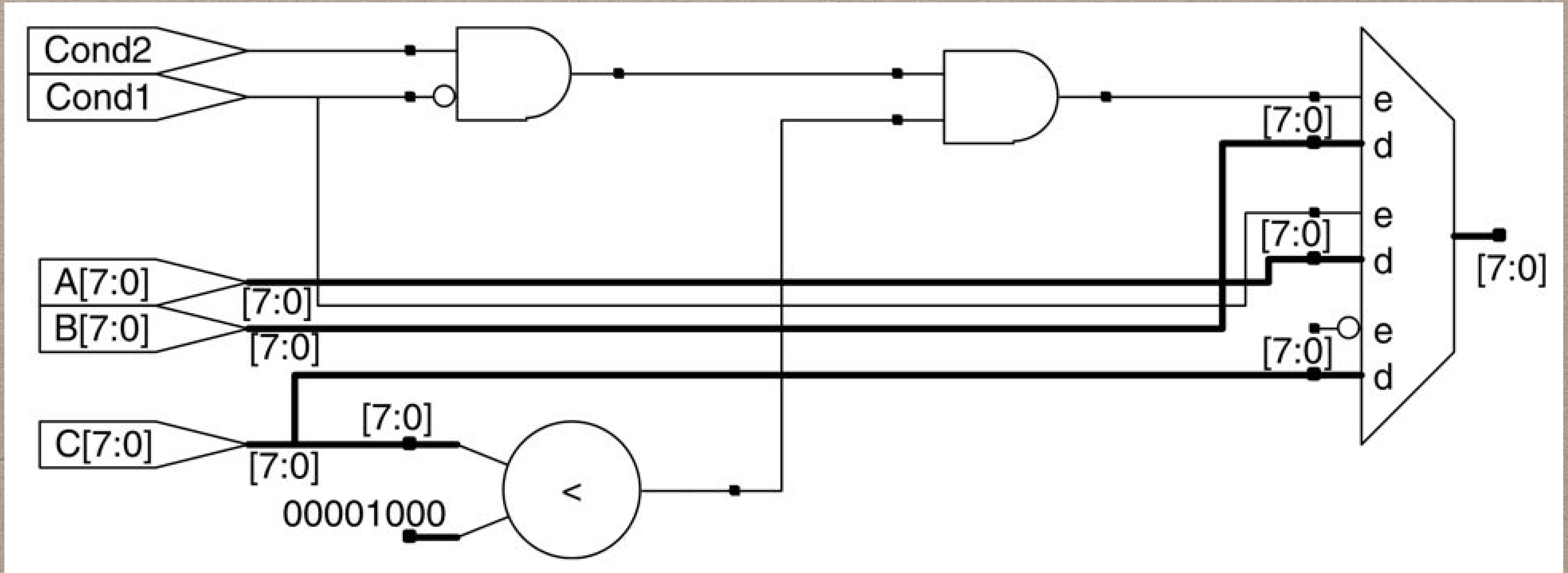
Timing -- Reorder Paths

THROUGHPUT =

LATENCY =

CRITICAL PATH =

```
module randomlogic(
     output reg [7:0] Out,
     input      [7:0] A, B, C,
     input            clk,
     input            Cond1, Cond2);
wire CondB = (Cond2 & !Cond1);
always @(posedge clk)
       if(CondB && (C < 8))
         Out <= B;
       else if(Cond1)
         Out <= A;
       else Out <= C;
endmodule
```



THE END