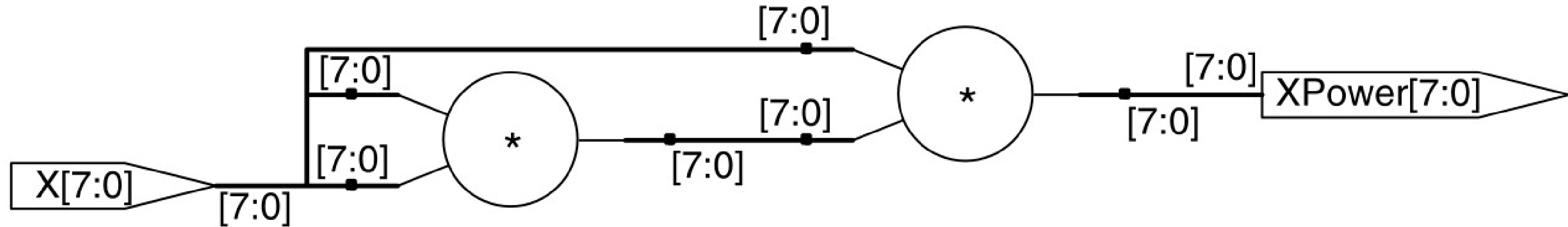
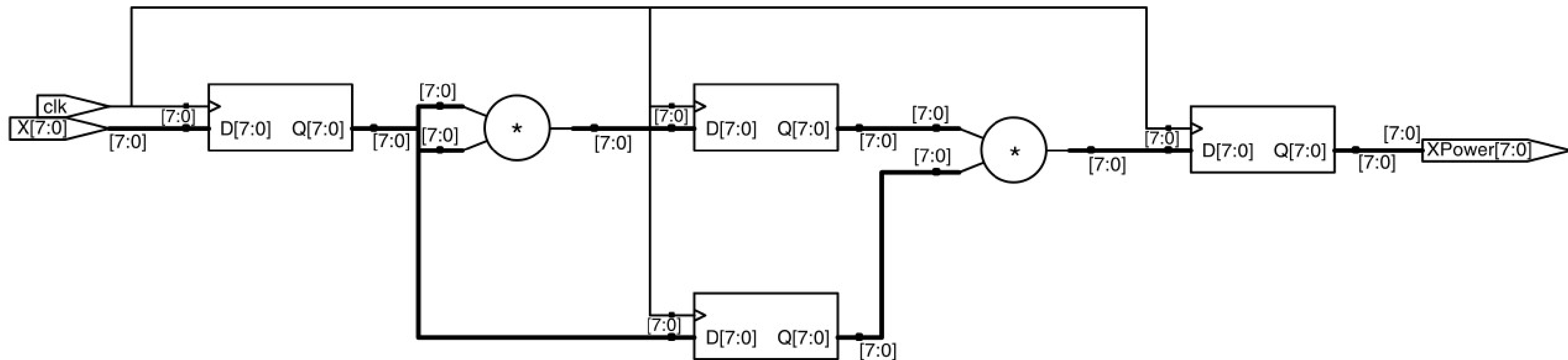
EEL 4783: HDL in Digital System Design

Final Review

Prof. Mingjie Lin

Circuit and Performance





Throughput = 8 bits/clock (assuming one new input per clock)

Latency = Between one and two multiplier delays, 0 clocks

Timing = Two multiplier delays in the critical path

Low-Latency

1. A low-latency design is one that passes the data from the input to the output as quickly as possible by minimizing the intermediate processing delays.
2. Oftentimes, a low-latency design will require parallelisms, removal of pipelining, and logical short cuts that may reduce the throughput or the max clock speed in a design.
3. No opportunity reducing latency in serial circuit, but there is in pipelined version.

Algorithmic Perspective of Pipeline

1. Unrolling the loop
   a) Iterative

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

```
module power3(
  output [7:0] XPower,
  output       finished,
  input  [7:0] X,
  input        clk, start); // the duration of start is a
                              single clock
  reg    [7:0] ncount;
  reg    [7:0] XPower;

  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
```

Algorithmic Perspective of Pipeline

1. Unrolling the loop
   a) Iterative

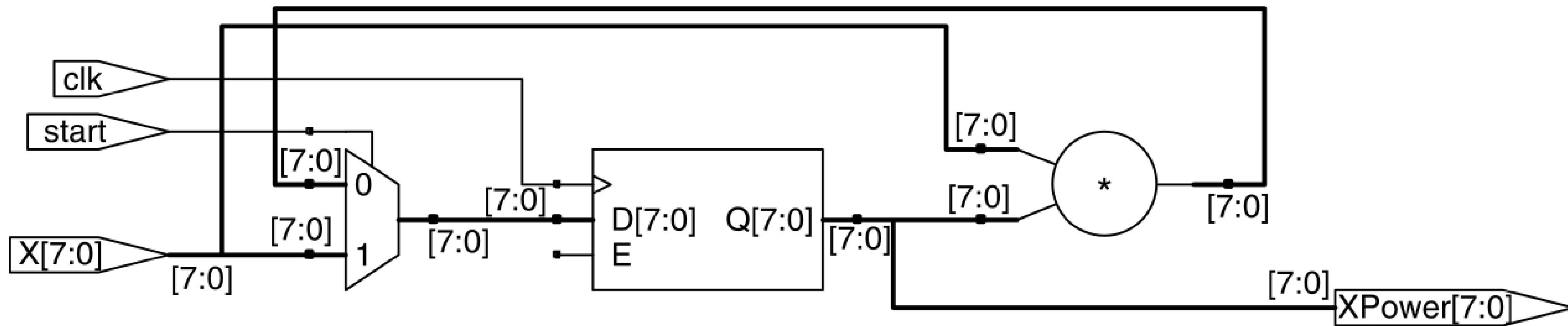
```
XPower = 1;
for (i=0;i < 3; i++)
  XPower = X * XPower;
```

```
module power3(
  output [7:0] XPower,
  output       finished,
  input  [7:0] X,
  input        clk, start); // the duration of start is a
                               single clock
  reg    [7:0] ncount;
  reg    [7:0] XPower;

  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
```

Performance



$$\text{Throughput} = 8/3, \text{ or } 2.7 \text{ bits/clock}$$

$$\text{Latency} = 3 \text{ clocks}$$

$$\text{Timing} = \text{One multiplier delay in the critical path}$$

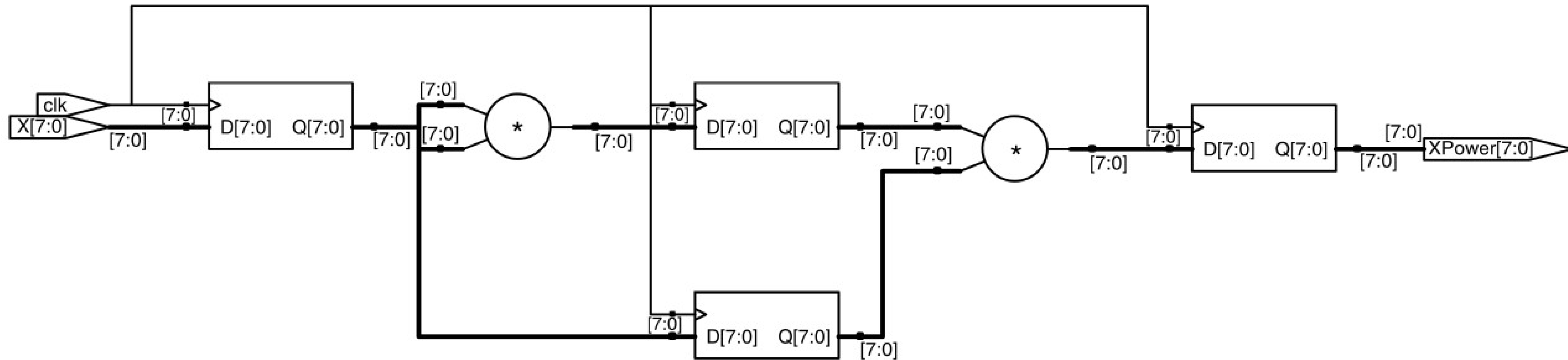
Pipelined Version

```
module power3(
  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
```

Circuit and Performance



Throughput = 8/1, or 8 bits/clock

Latency = 3 clocks

Timing = One multiplier delay in the critical path

How to Improve Timing? 1

1. The first strategy for architectural timing improvements is to add intermediate layers of registers to the critical path.
2. This technique should be used in highly pipelined designs where an additional clock cycle latency does not violate the design specifications, and the overall functionality will not be affected by the further addition of registers.

How to Improve Timing? 2

1. The second strategy for architectural timing improvements is to reorganize the critical path such that logic structures are implemented in parallel.
2. This technique should be used whenever a function that currently evaluates through a serial string of logic can be broken up and evaluated in parallel.

How to Improve Timing? 3

1. The third strategy for architectural timing improvements is to flatten logic structures.
2. This is closely related to the idea of parallel structures, but applies specifically to logic that is chained due to priority encoding. Typically, synthesis and layout tools are smart enough to duplicate logic to reduce fanout, but they are not smart enough to break up logic structures that are coded in a serial fashion, nor do they have enough information relating to the priority requirements of the design.

How to Improve Timing? 4

1. The fourth strategy is called register balancing.
2. Conceptually, the idea is to redistribute logic evenly between registers to minimize the worst-case delay between any two registers.
3. This technique should be used whenever logic is highly imbalanced between the critical path and an adjacent path. Because the clock speed is limited by only the worst-case path, it may only take one small change to successfully rebalance the critical logic.

How to Improve Timing? 5

1. The fifth strategy is to reorder the paths in the data flow to minimize the critical path.
2. This technique should be used whenever multiple paths combine with the critical path, and the combined path can be reordered such that the critical path can be moved closer to the destination register.
3. With this strategy, we will only be concerned with the logic paths between any given set of registers.

Digital Circuits: Area=Cost

- Choosing correct topology
  - Higher-level organization of the design
  - Not device specific

- Circuit-level reduction
  - Performed by the synthesis and layout tools
  - Minimization of the number of gates
  - May be device specific.

- Dilemma
  - Good topology for area
  - Reuse logic resources as much as possible

Main Points

- Rolling up the pipeline to reuse logic resources in different stages of a computation.
- Controls to manage the reuse of logic when a natural flow does not exist.
- Sharing logic resources between different functional operations.
- The impact of reset on area optimization.
    - Impact of FPGA resources that lack reset capability.
    - Impact of FPGA resources that lack set capability.
    - Impact of FPGA resources that lack asynchronous reset capability.
    - Impact of RAM reset.
    - Optimization using set/reset pins for logic implementation.

Rolling up the Pipeline

- "Rolling up" is direct opposite to "unrolling the loop"
- Unrolling a loop
  - Increase performance
  - Increase area → require more resource to hold immediate results and replicate computational structures for parallel
- Rolling up a pipeline
  - Optimize the area of pipelined designs with duplicated logic in the pipeline stages.

```
1  initial begin
2    mema              = 0; // Illegal syntax - Attempt to write to entire array
3    arrayb[1]         = 0; // Illegal syntax - Attempt to write to elements [1][255]...[1][0]
4    arrayb[1][31:12] = 0; // Illegal syntax - Attempt to write to multiple elements
5    mema[1]           = 0; // Assigns 0 to the second element of mema
6    arrayb[1][0]      = 0; // Assigns 0 to the bit referenced by indices [1][0]
7  end
8
9  // Generate loop with arrays of wires
10 generate
11 genvar gi;
12   for (gi=0; gi<8; gi=gi+1) begin : gen_array_transform
13     my_example_16_bit_transform_module u_mod (
14       .in  (y[gi]),
15       .out (z[gi])
16     );
17   end
18 endgenerate
19
20 // For loop with arrays
21 integer index;
22 always @(posedge clk, negedge rst_n) begin
23   if (!rst_n) begin
24     // reset arrayb
25     for (index=0; index<256; index=index+1) begin
26       mema[index] <= 8'h00;
27     end
28   end
29   else begin
30     // out of reset functional code
31   end
32 end
```

Important Topics

- The impact of clock control on dynamic power consumption
- Problems with clock gating
- Managing clock skew on gated clocks
- Input control for power minimization
- Impact of the core voltage supply
- Guidelines for dual-edge triggered flip-flops
- Reducing static power dissipation in terminations