

REAL-TIME DIGITAL SYSTEMS DESIGN AND VERIFICATION WITH FPGAS

ECE 387 – LECTURE 9

PROF. DAVID ZARETSKY

DAVID.ZARETSKY@NORTHWESTERN.EDU

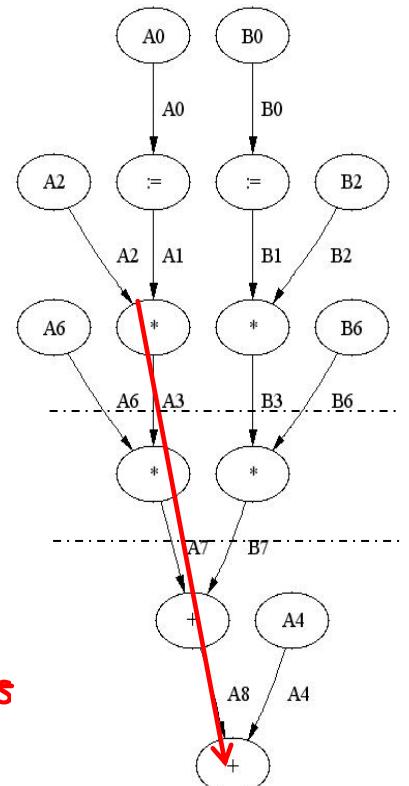
AGENDA

- Critical Paths
- Loop Unrolling
- Hardware Pipelining
- Software Pipelining

CRITICAL PATHS

- The critical path (or critical delay) affects performance
- Operation scheduling lets you arrange or chain operations in different states of an FSM
- Breaking up complex operations and large combinational paths can improve performance.

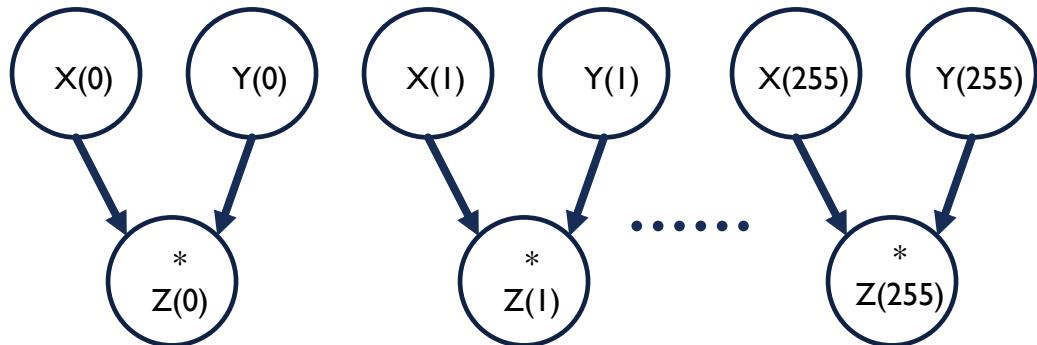
Multipliers = 5 ns
Adders = 2 ns
~~Critical delay = 14 ns~~
Critical delay = 5 ns



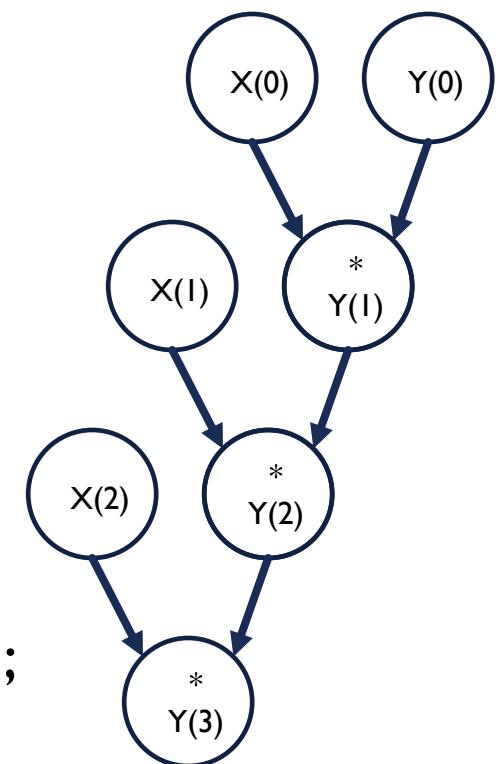
LOOP UNROLLING

- Using loop structures will replicate operations within the same cycle.
- Long delay paths may be result if there exists data-dependencies across loop iterations

```
for i in 0 to 255 loop  
    z(i) := x(i) * y(i);  
end loop;
```



```
for i in 0 to 2 loop  
    y(i+1) := x(i) * y(i);  
end loop;
```

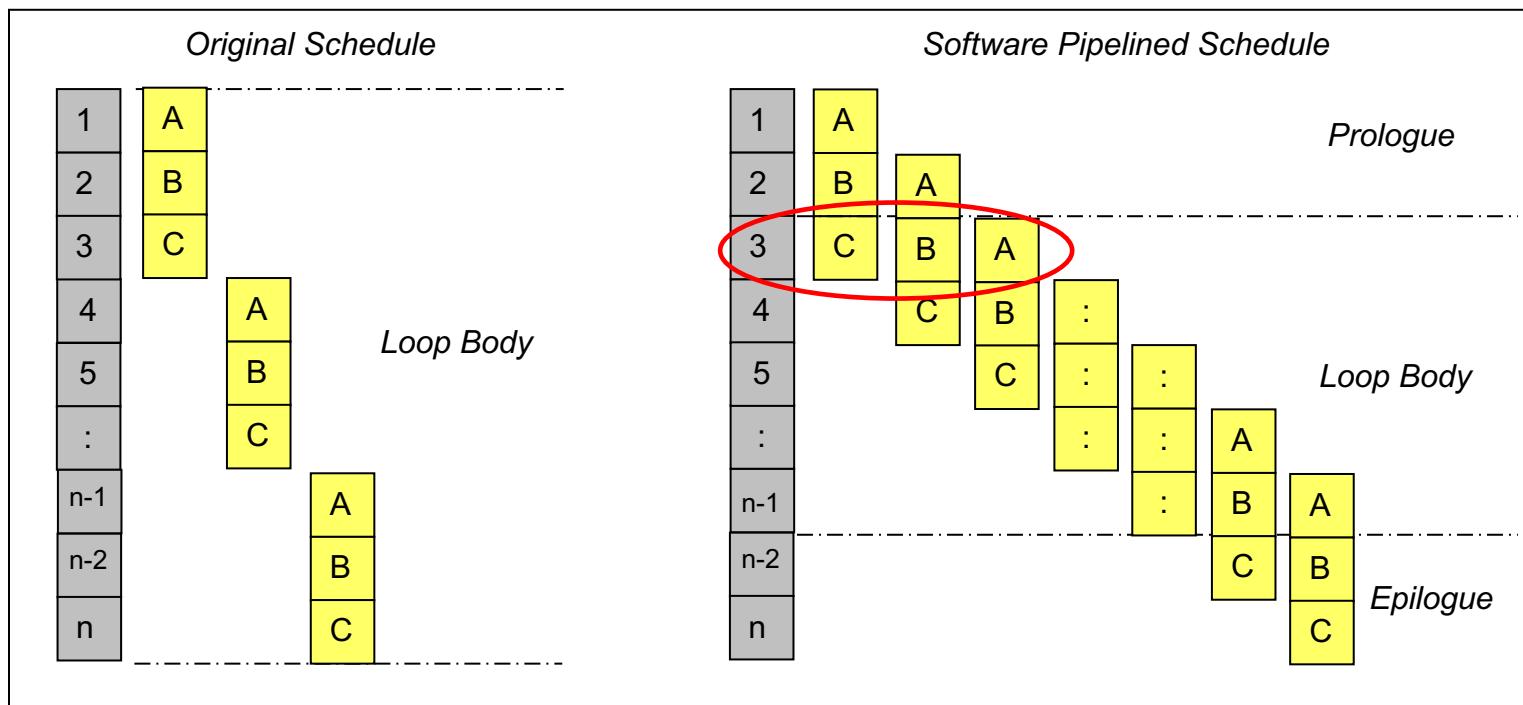


PIPELINING LOGIC TO IMPROVE SPEED

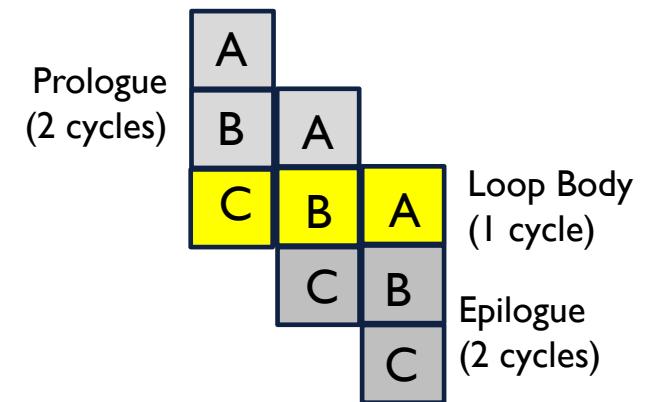
- Pipelining is most efficient for critical paths which can not be fixed by special coding schemes.
- 2 forms of pipelining:
 - Software Pipelining
 - Hardware Pipelining

SOFTWARE PIPELINING

- Overlap loop iterations to reduce the body of the loop from 3 cycles to 1 cycle.



Software Pipelined Implementation



SOFTWARE PIPELINING EXAMPLE

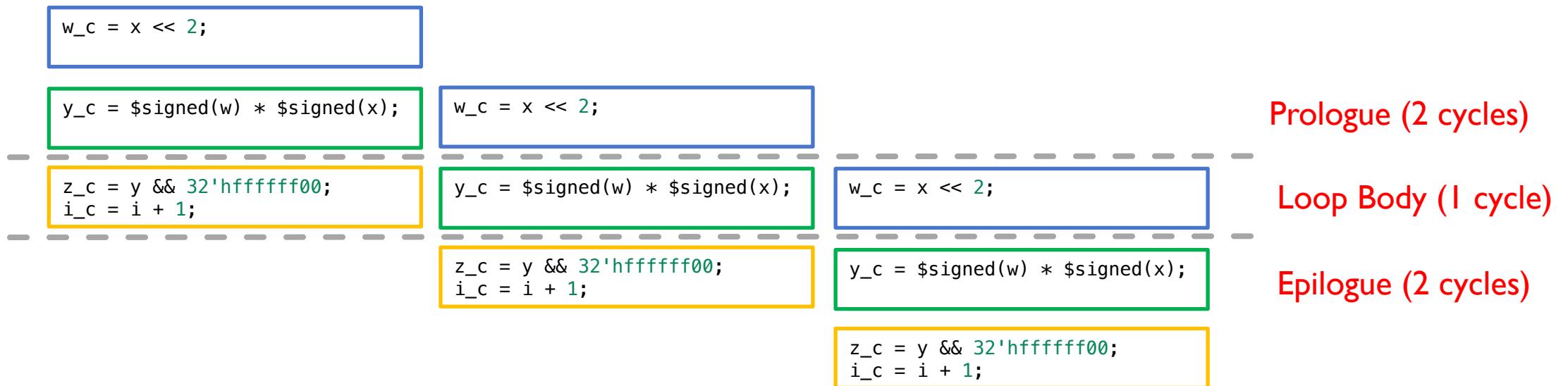
- Loop body with 3 cycles (states)
- Reduce loop body to 1 cycle using software pipelining

```
s1: begin
    w_c = x << 2;
    next_state = s2;
end

s2: begin
    y_c = $signed(w) * $signed(x);
    next_state = s3;
end

s3: begin
    z_c = y && 32'hfffff00;
    i_c = i + 1;
    if (i < 64 )
        next_state = s1;
    else
        next_state = s4;
end
end
```

SOFTWARE PIPELINING EXAMPLE



SOFTWARE PIPELINING EXAMPLE

```
s1: begin  
    w_c = x << 2;  
    next_state = s2;  
end  
  
s2: begin  
    y_c = $signed(w) * $signed(x);  
    next_state = s3;  
end  
  
s3: begin  
    z_c = y && 32'hfffff00;  
    i_c = i + 1;  
    if (i < 64 )  
        next_state = s1;  
    else  
        next_state = s4;  
    end  
end
```

Non-pipelined loop
has 3 cycles.

```
s1: begin  
    w_c = x << 2;  
    next_state = s2;  
end  
  
s2: begin  
    y_c = $signed(w) * $signed(x);  
    w_c = x << 2;  
    next_state = s3;  
end  
  
s3: begin  
    z_c = y && 32'hfffff00;  
    y_c = $signed(w) * $signed(x);  
    w_c = x << 2;  
    i_c = i + 1;  
    if (i < 62 ) next_state = s3;  
    else next_state = s4;  
end  
  
s4: begin  
    z_c = y && 32'hfffff00;  
    y_c = $signed(w) * $signed(x);  
end  
  
s5: begin  
    z_c = y && 32'hfffff00;  
end
```

Prologue (2 cycles)

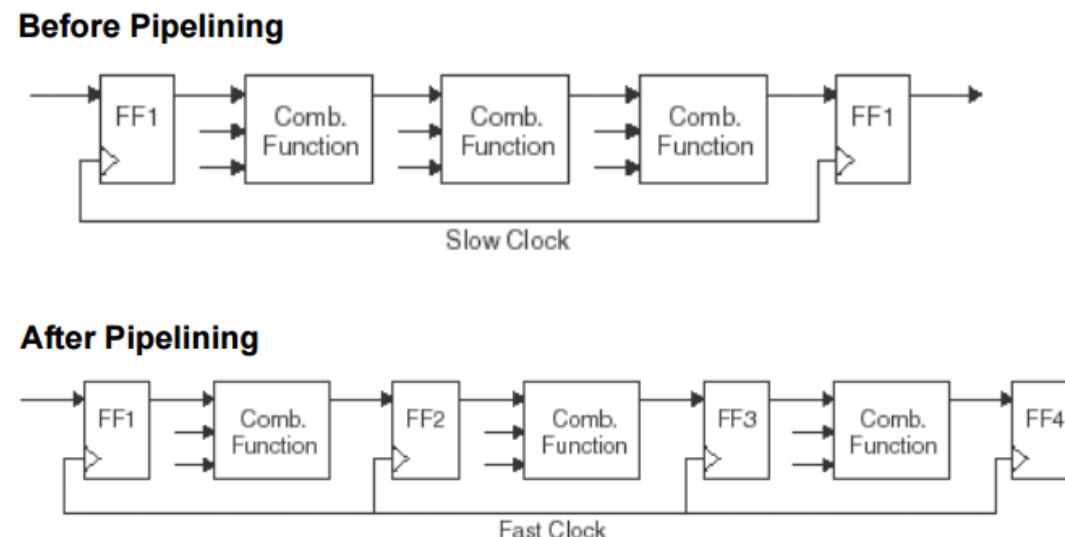
Loop Body (1 cycle)

Pipelining resulted in loop body reduced from
3 to 1 cycle – 3X SPEEDUP!

Epilogue (2 cycles)

HARDWARE PIPELINING

- Pipelining improves performance by restructuring long data paths with several levels of logic over multiple clock cycles.
- Produces faster clock cycles by relaxing the clock-to-output and setup time requirements between the registers.
- Pipelining multipliers will allow you to reach 250MHz+ frequencies.



HARDWARE PIPELINING EXAMPLE

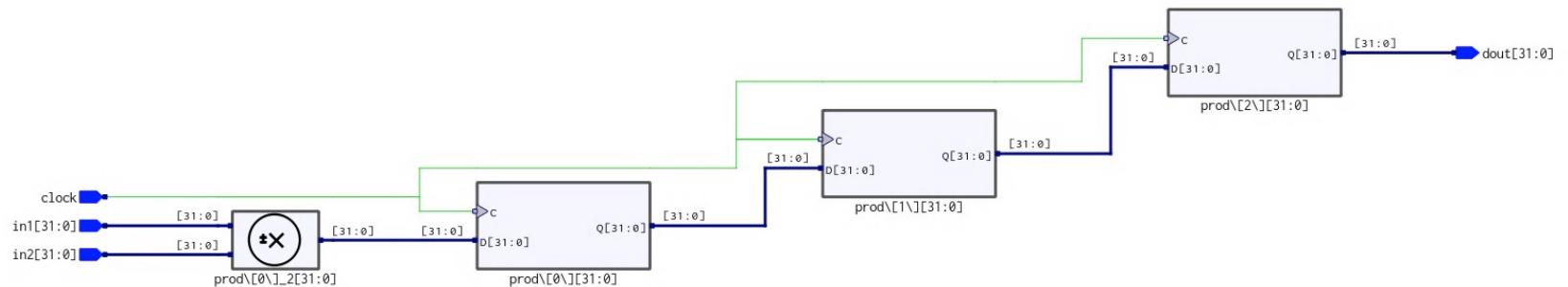
```
module mult_pipe
#(parameter STAGES = 3)
(
    input logic clock,
    input logic [31:0] in1,
    input logic [31:0] in2,
    output logic [31:0] dout
);

logic [0:STAGES-1] [31:0] prod;

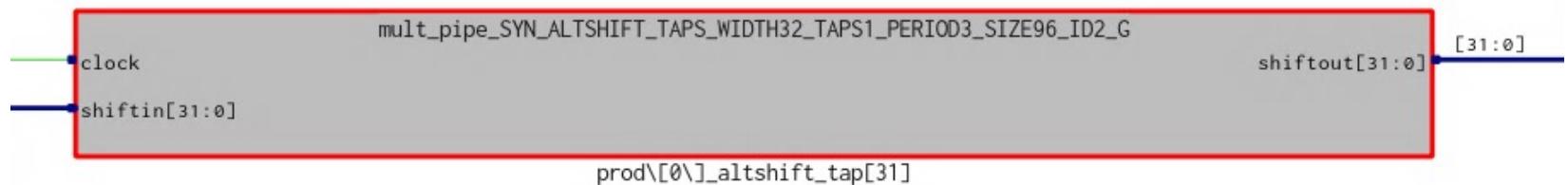
generate if (STAGES > 1)
    always_ff @(posedge clock) begin
        prod[1:STAGES-1] = prod[0:STAGES-2];
    end
endgenerate

always_ff @(posedge clock) begin
    prod[0] <= $signed(in1) * $signed(in2);
end

assign dout = prod[STAGES-1];
endmodule
```



Synthesized shift-register for multiply



LOOP UNROLLING VS. PIPELINING

- Loop Unrolling
 - Increases number of operations per cycle
 - Reduces loop iterations
 - Does not affect number of cycles per loop iteration
 - Increases area and resource utilization
 - Does not affect frequency
- Software Pipelining
 - Marginally increases number of operations per cycle
 - Reduces number of cycles per loop iteration
 - Does not affect loop iterations
 - Increases area and resources
 - Does not affect frequency
- Hardware Pipelining
 - Does not affect number of operations per cycle
 - Increases the number of cycles per loop iteration
 - Does not affect loop iterations
 - Reduces area and resources
 - Increases Frequency

GENERAL ADVICE

- Pipelining
 - Avoid hardware pipelining in loops because they increase loop cycles
 - Use hardware + software pipelining together whenever possible to reduce loop cycles
- Loop Unrolling
 - Use loop unrolling for shifts, adders, and other simple constructs
 - Be careful when using loop unrolling with multipliers (resource limited)
- Always check mapped technology to make sure the resulting instantiated components are what you expected
- Determine what constructs are supported by the target FPGA architecture
 - Shift registers
 - MACs
 - Pipelining

NEXT...

- Network Packet Processing
- Homework 5: UDP