
Computer Aided Digital Systems Design - EE 4743/6743

Sherif Abdelwahed

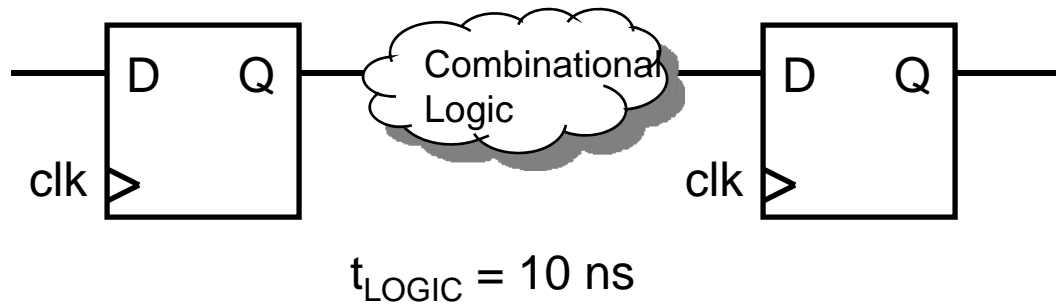
Pipelining

**Department of Electrical and Computer Engineering
Mississippi State University**

Speeding up the Clock

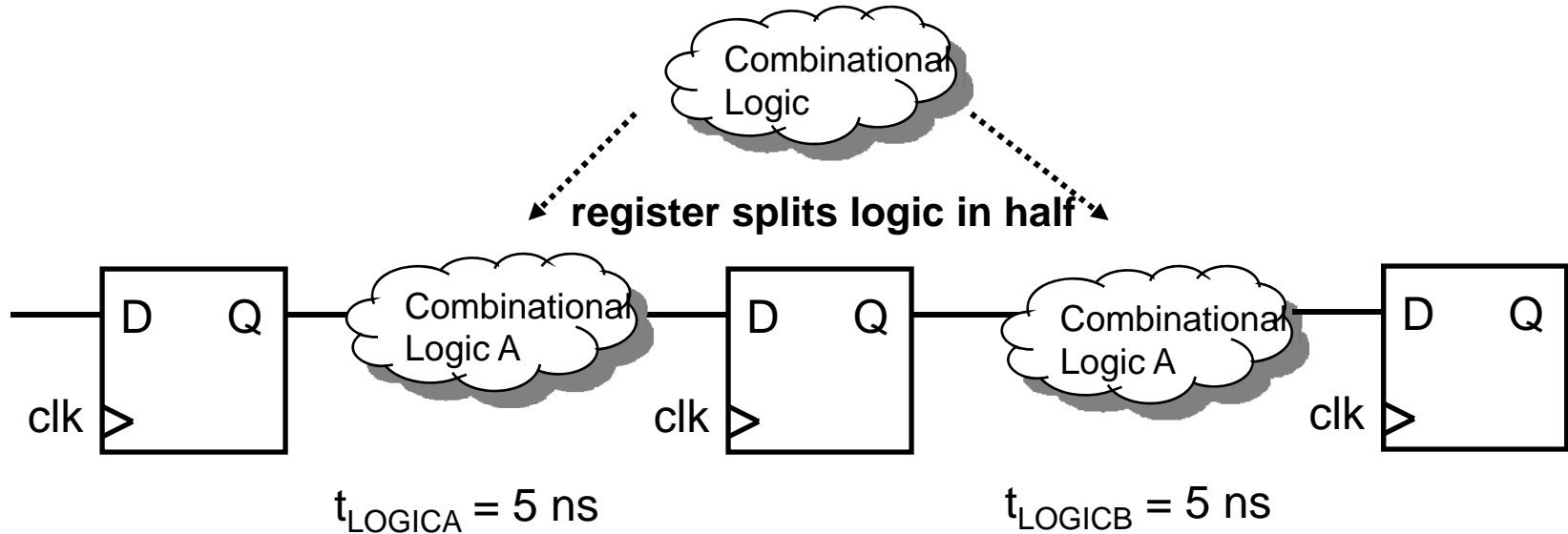
- The register-to-register delay is usually the delay path that sets the maximum clock rate
 - From a design point of view, can only affect the combinational logic between the registers
 - Need to shorten the maximum combinational delay path
 - Setup and Hold time of registers are fixed
 - Can shorten the delay by placing a register in the combinational logic to break longest delay path
 - This technique is called **pipelining**
-

Pipelining: Conceptual



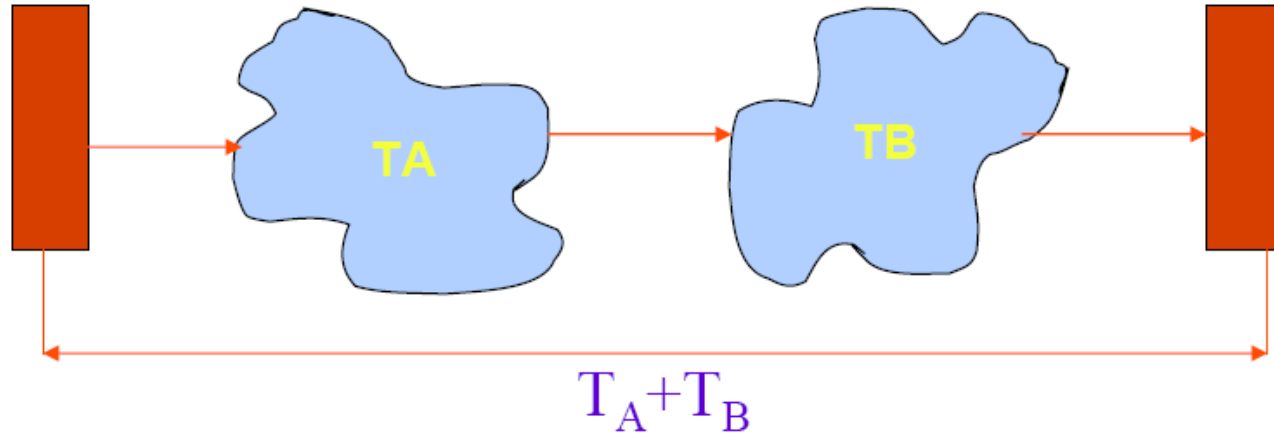
- Assuming $t_{\text{C2Q}} = t_{\text{su}} = 0 \text{ ns}$, the critical path is 10 ns, and the maximum clock frequency is 100 MHz
 - Latency = 2 cycles

Pipelining: Conceptual

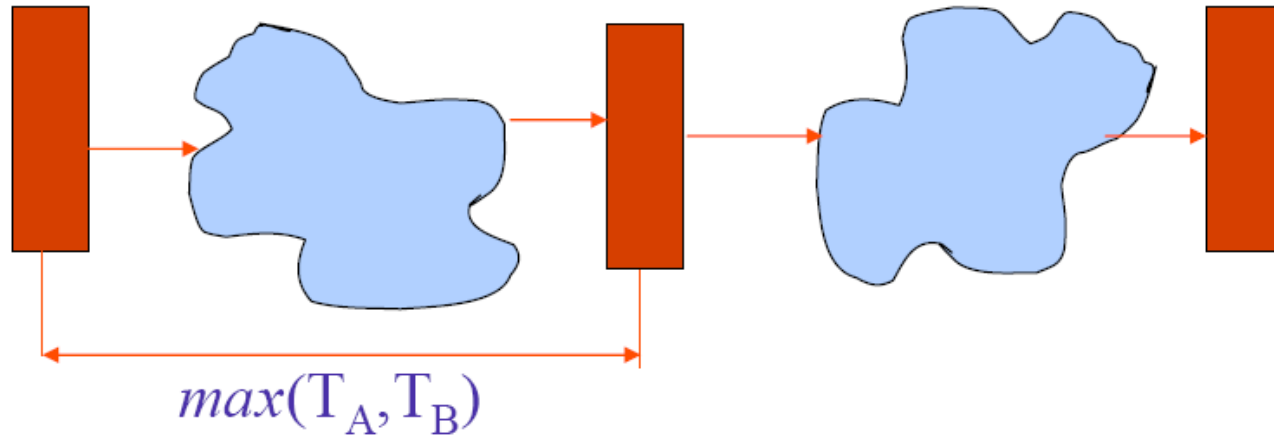


- Purpose of pipelining is to reduce the critical path of the circuit by inserting an additional register (called a pipeline register)
- Now critical path delay is 5 ns, so maximum clock frequency is 200 MHz
 - Double the clock frequency
- However, latency increases to 3 cycles (and area is increased due to additional register)
- **In general, pipelining increases throughput at the cost of increased latency, area, and power**

The General Situation



Pipelined
version

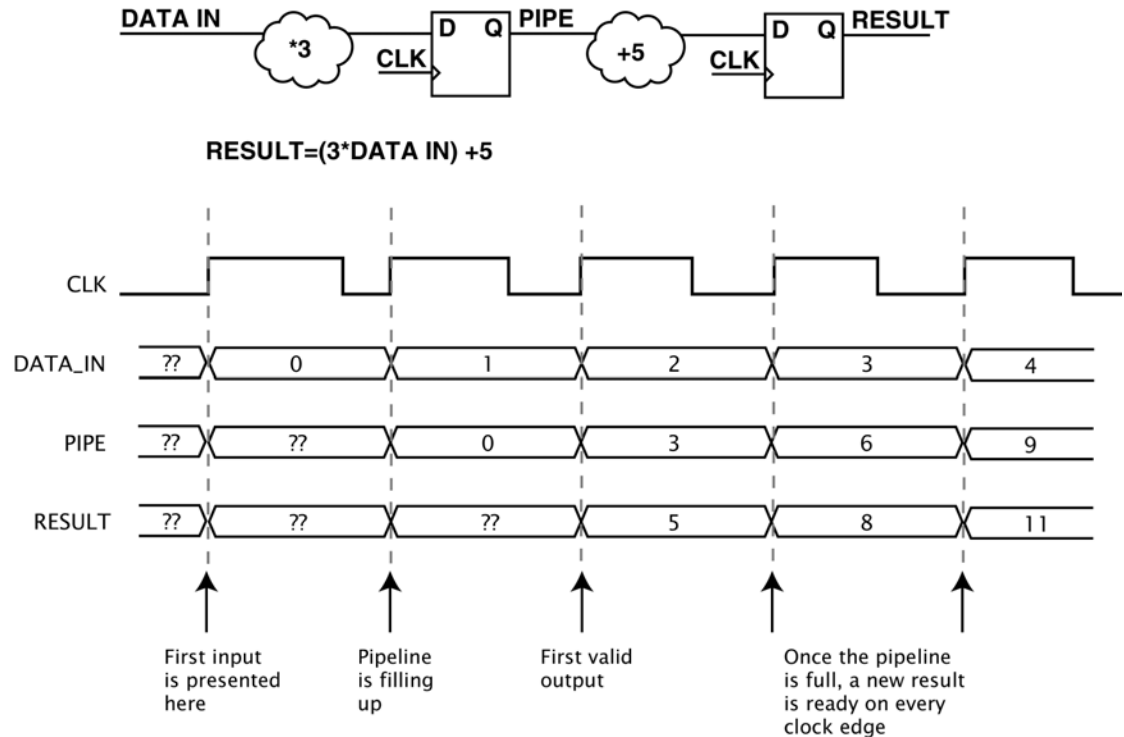


Pipelining Considerations

- Are enough flip-flops available?
 - In general, you will not run out of flip-flops
 - Are there multiple logic levels between flip-flops?
 - If there is only one logic level between flip-flops, pipelining will not improve performance
 - Can the system tolerate *latency*?
-

Latency in Pipelines

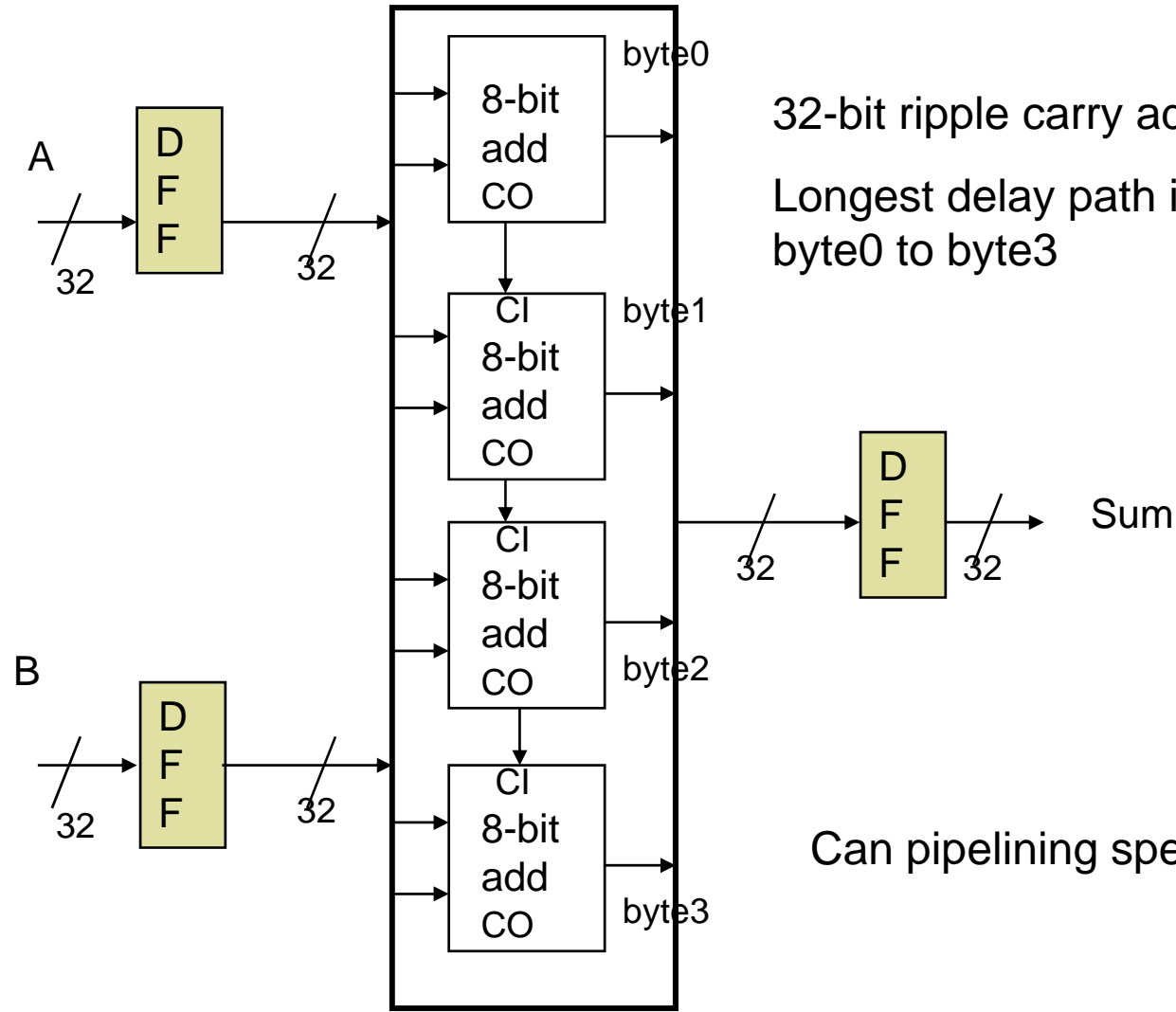
- Each pipeline stage adds one clock cycle of delay before the first output will be available
 - Also called “filling the pipeline”
- After the pipeline is filled, a new output is available every clock cycle



Timing parameters

	definition	units		pipelining
delay	time point→point	ns		
latency	time input→output	ns	↗	bad
throughput	#output bits/time unit	Mbits/s	↗	good
clock period	rising edge →rising edge of clock	ns	↘	good
clock frequency	$\frac{1}{\text{clock period}}$	MHz	↗	good

Pipeline Example

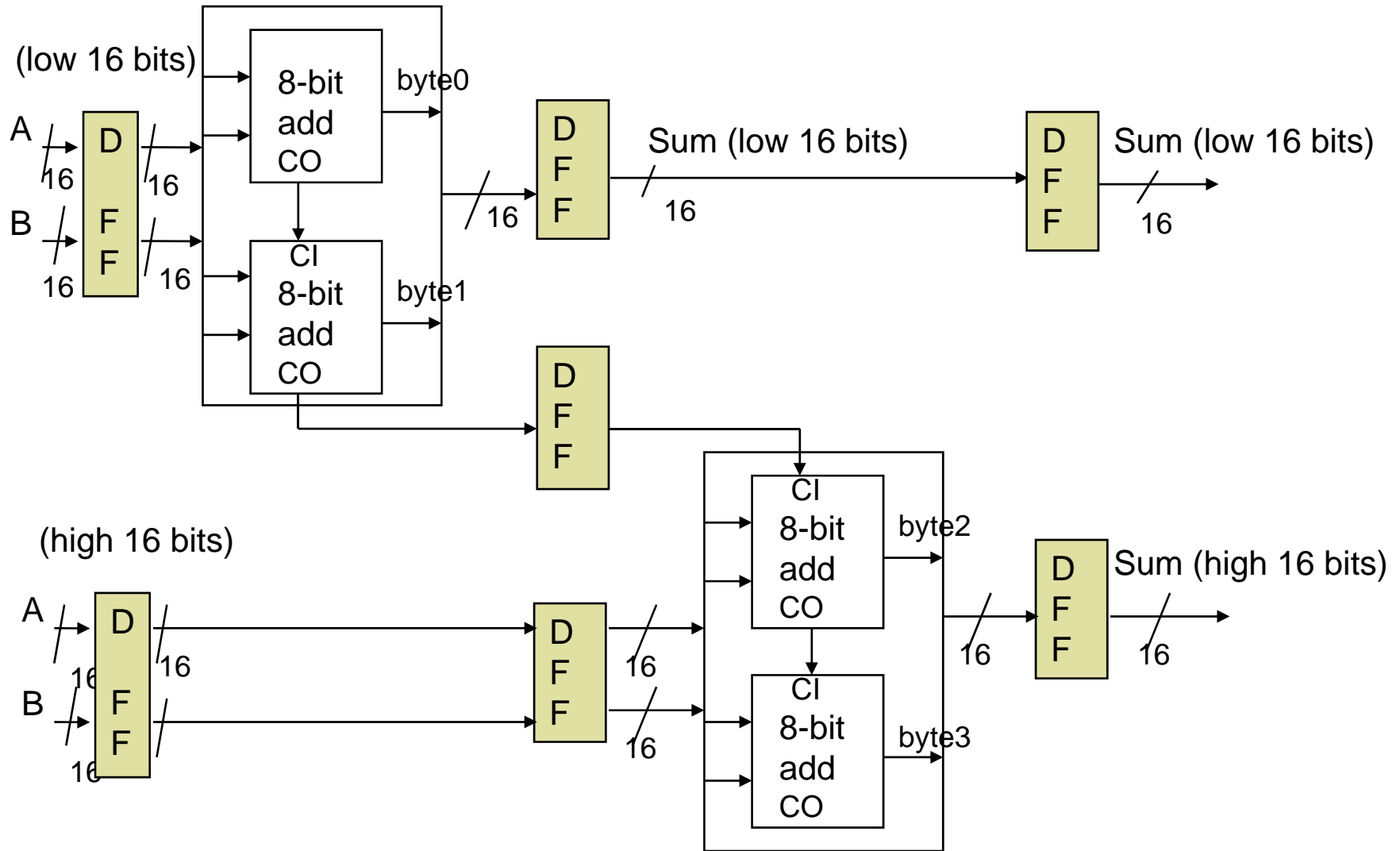


32-bit ripple carry adder.

Longest delay path in carry chain from byte0 to byte3

Can pipelining speed this up?

Insert pipeline stage between byte1 and byte2



Comments on Pipeline Example

- Note that the pipeline stage broke the carry chain into two equal paths
 - Each pipeline stage should have approximately the same combinational delay
 - Clock speed will be set by the delay of the slowest pipeline stage
 - If I inserted 2 pipeline stages, I would need to break the carry chain delay into equal thirds
 - Could insert a pipeline stage between each BIT in order to get maximum clock speed
 - Called '*bit pipelining*'
-

Latency Tolerance

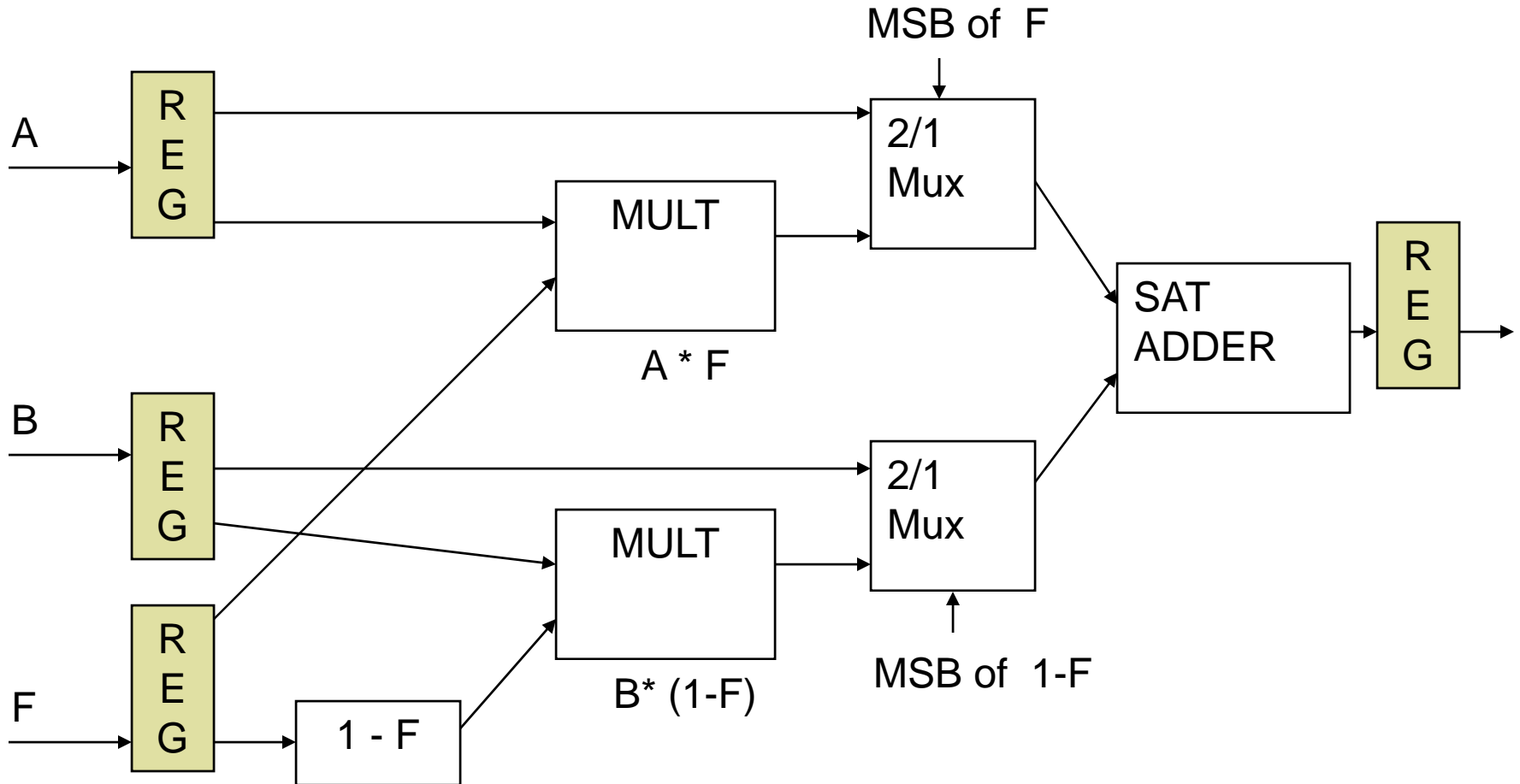
- Latency tolerance is dependent upon each application
 - Frequent flushing of a pipeline:
 - discarding partial results within the pipeline and restarting the pipeline with a new value
 - wastes time
 - makes an application latency **intolerant**.
 - Flushing of a pipeline introduces clock cycles in which the results coming out of the pipeline are ignored
 - wasted clock cycles.
 - High latency tolerance design
 - have many pipeline stages
 - whatever the number you need to meet the clock rate specification.
-

Two Applications

- Graphics hardware for processing pixels is extremely latency tolerant - not unusual to find pipelines that have 10's of stages.
 - Graphics pipelines are never flushed
 - High clock rate is EXTREMELY important because of large number of pixels (> 1 Million) that have to be supplied every screen, at > 30 updates per second
 - Microprocessor instruction pipelines are not very latency tolerant - most CPU pipelines are only about 5-10 stages.
 - Branch instructions can cause pipeline to be flushed. By the time you determine direction of branch, may have started processing instructions that should not be in the pipeline. These are flushed and the pipeline restarted.
-

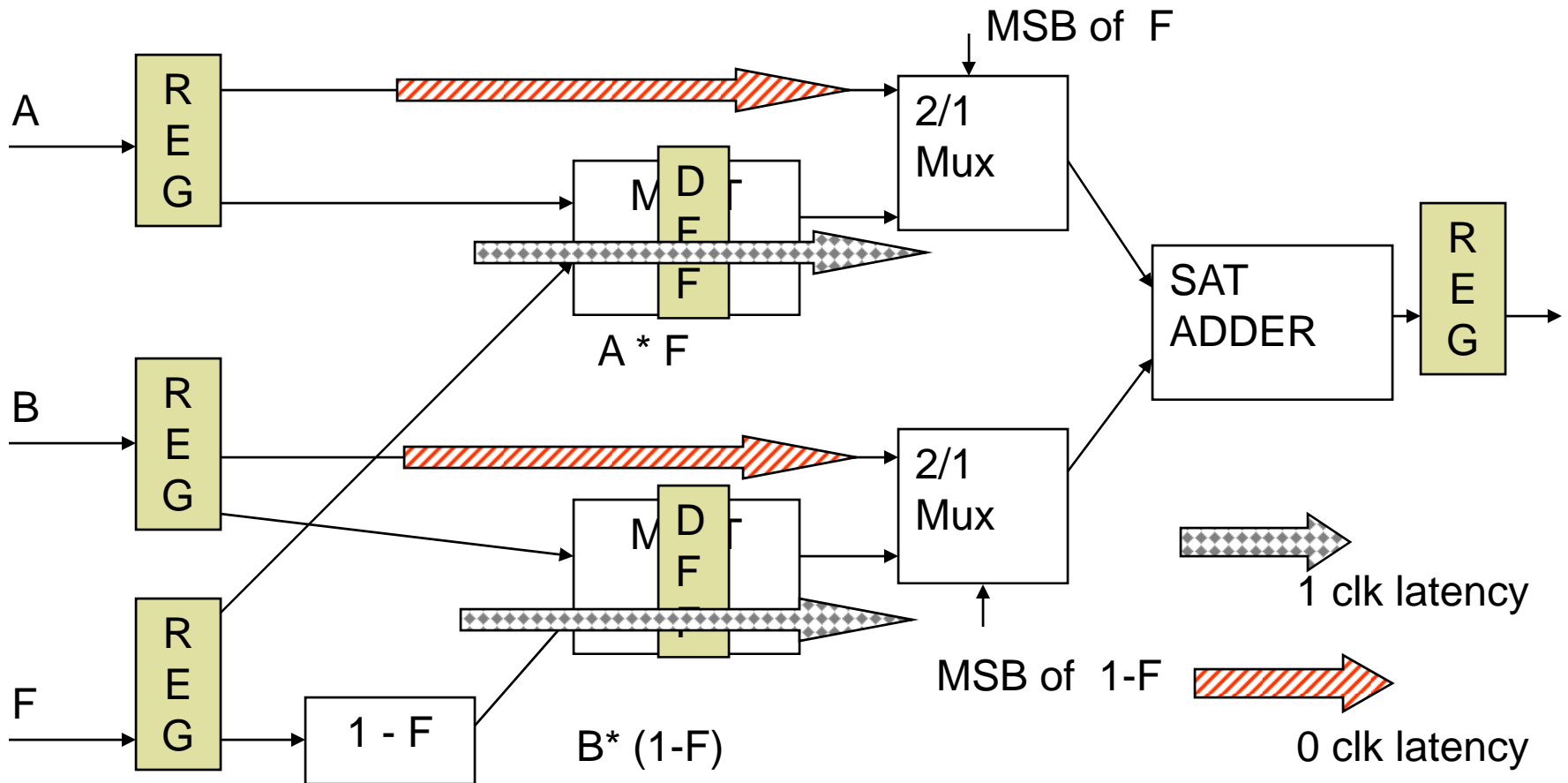
Example: Blend Operation

BLEND Datapath without Pipelining. MULT is combinational.



Example: Blend Operation

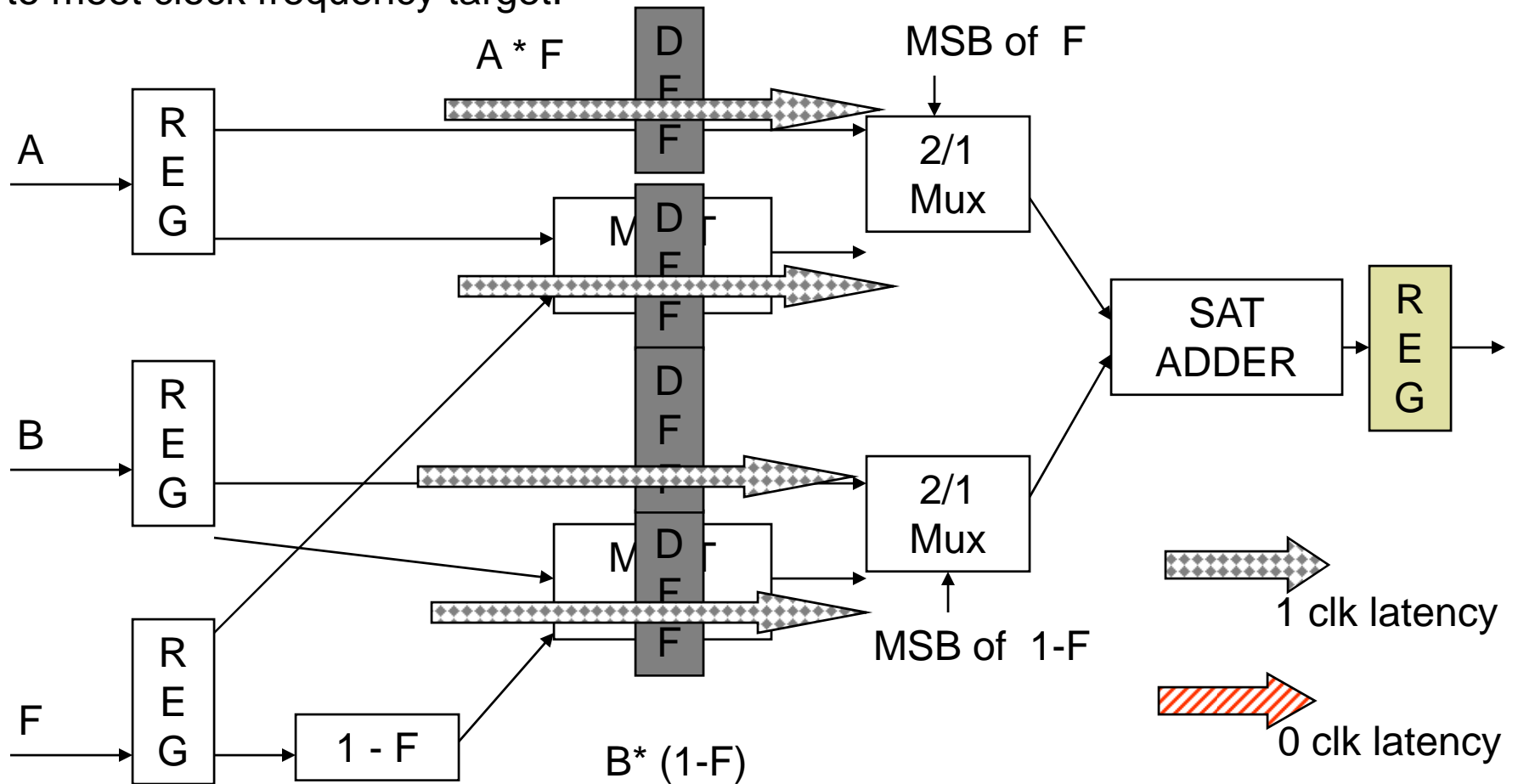
Multiplication units can be pipelined via a special "PIPELINE" parameter. Add one pipeline stage to each Multiplier. DFFs will be inserted automatically



We now have a LATENCY mismatch within our datapath!!!!

Example: Blend Operation

Correct the latency mismatch by adding DFFs in other path as well. May have to break delay paths in other places or add additional pipeline stages to the multipliers to meet clock frequency target.

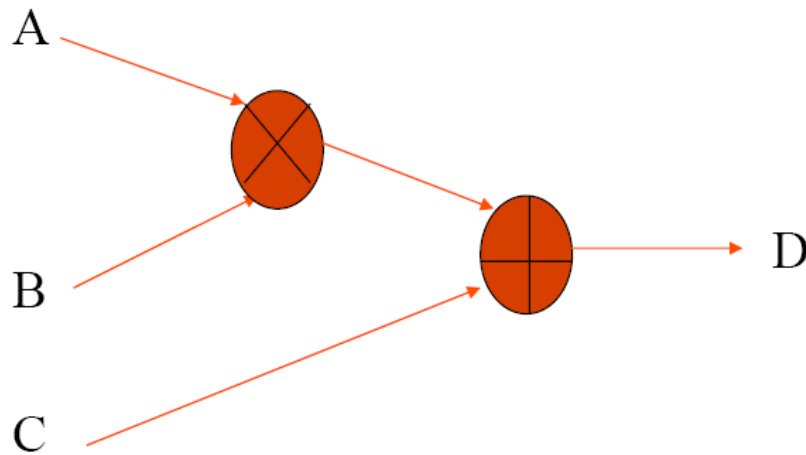


Control lines (like MUX control line), must be pipelined also.

Dataflow Example

```
input [3:0] a,b,c;  
reg [7:0] D;
```

```
-- a, b and c arrive at the same time  
assign d = a*b + c;
```

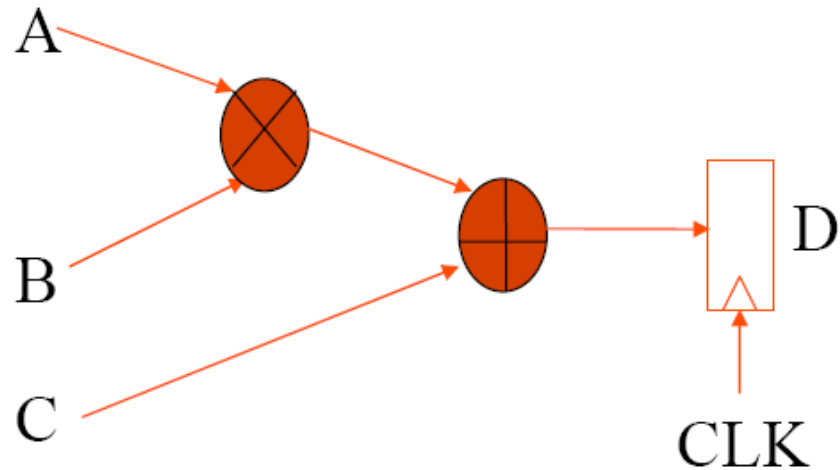


Purely
Combinational

Registered Output - Blocking

```
always @(a,b) begin
  ab = a * b;
end

always @(posedge clk)
  d <= ab + c;
```



Equivalent to

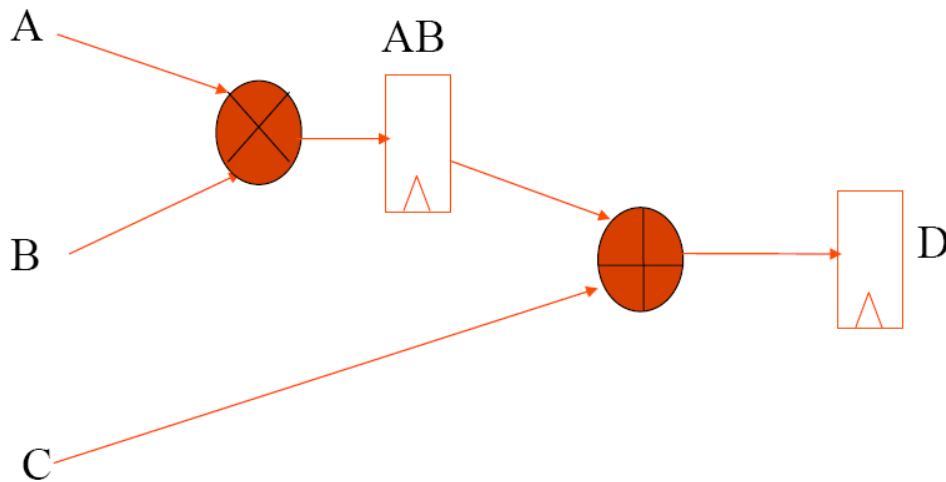
$$d = a[i]*b[i] + c[i];$$

Implications

- Addition and Multiplication operation are cascaded
- The maximum delay through the combinational logic is
$$T_{\text{ADD}} + T_{\text{MULT}}$$
- After the delay the register can latch the data
- Meanwhile the input must remain unchanged
- Next input can be given only after the delay $T_{\text{ADD}} + T_{\text{MULT}}$ and thus clock should be as wide as the sum of the delays
- The operation takes one clock cycle and you can perform one operation every clock cycle

Model with Non-blocking

```
always @(posedge clk) begin
  ab <= a * b;
  d <= ab + c;
end
```



Why?

- Register for ab
 - Assigned inside a clock statement
- Register for d
 - Also within a clock statement

Problem with the Model

- Multiplier works on current a and b
 - The result will be available only after one clock cycle
- Adder works on current c and previous ab
- The equivalent C code:

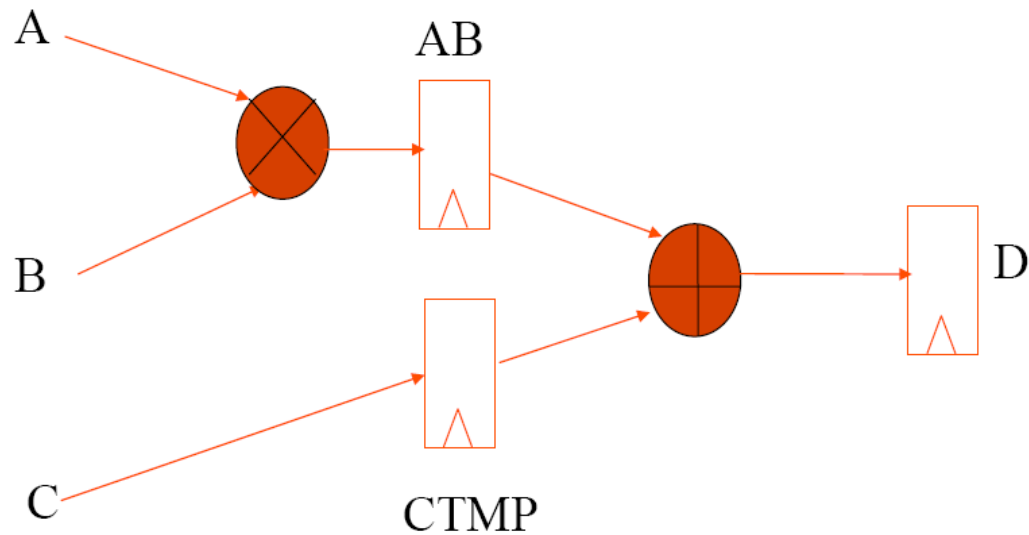
$$d = a[i-1]*b[i-1] + c[i];$$

From Simulation Point of View

- ab is a nonblocking assignment
 - Not updated till a new timing control
 - d uses the value of ab
 - Value of ab not updated immediately
 - Reg ab has memory
 - Thus previous value is used
 - Simulation and Synthesis are consistent
-

Another Model with Non-blocking

```
always @(posedge clk) begin
  ab   <= a * b;
  ctmp <= c;
  d    <= ab + ctmp;
end
```



Analysis of the Model

- New reg ctmp copies c
- All the regs ab, ctmp and d get a register
- When ab is computed, c is just copied to ctmp
- Adder always looks at the previous value of ab and ctmp (previous data)
- all data inputs pass through same number of registers and hence consistent results
- Equivalent C code :

$$d = a[i-1]*b[i-1] + c[i-1];$$

From Simulation Point of View

- ab is assigned only at the end
 - ctmp is also assigned only at the end
 - Both ab and ctmp are regs and thus retain the old value
 - d looks at the values of ab and ctmp from the previous assignment
 - Consistent with the synthesis model
-

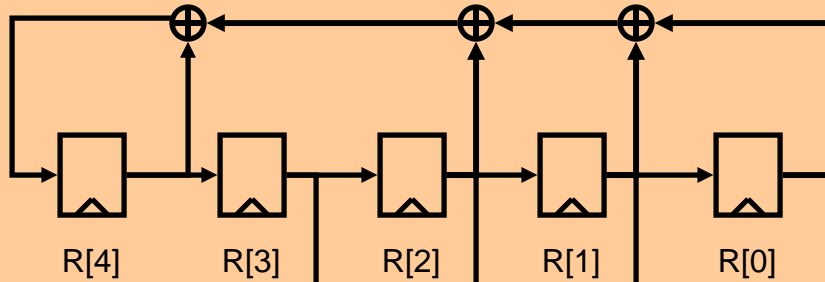
More Analysis

- Unlike the model with blocking assignments, results are not available immediately. They are delayed by one clock cycle.
- The clock can now be $\max(T_{ADD}, T_{MULT})$ instead of $T_{ADD} + T_{MULT}$
 - Faster clock
- You can supply data, once every clock cycle
- You get the results once every clock cycle (except for the very first data)

Feedforward vs. Feedback Pipelining

- In general, pipelining only possible on feedforward paths, not feedback paths
 - Feedback path pipelining requires special examination to retain the same functionality
 - Feedforward paths can be pipelined with no change in functionality (only a change in latency)
-

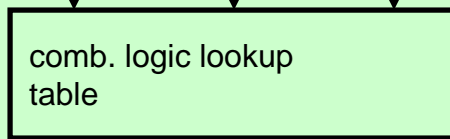
Feedforward Path versus Feedback Path



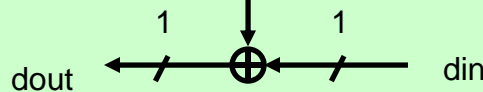
$$R4(n+1) = R4(n) + R2(n) + R1(n) + R0(n)$$

FEEDBACK SECTION

-generally cannot be pipelined
(unless certain properties exist)



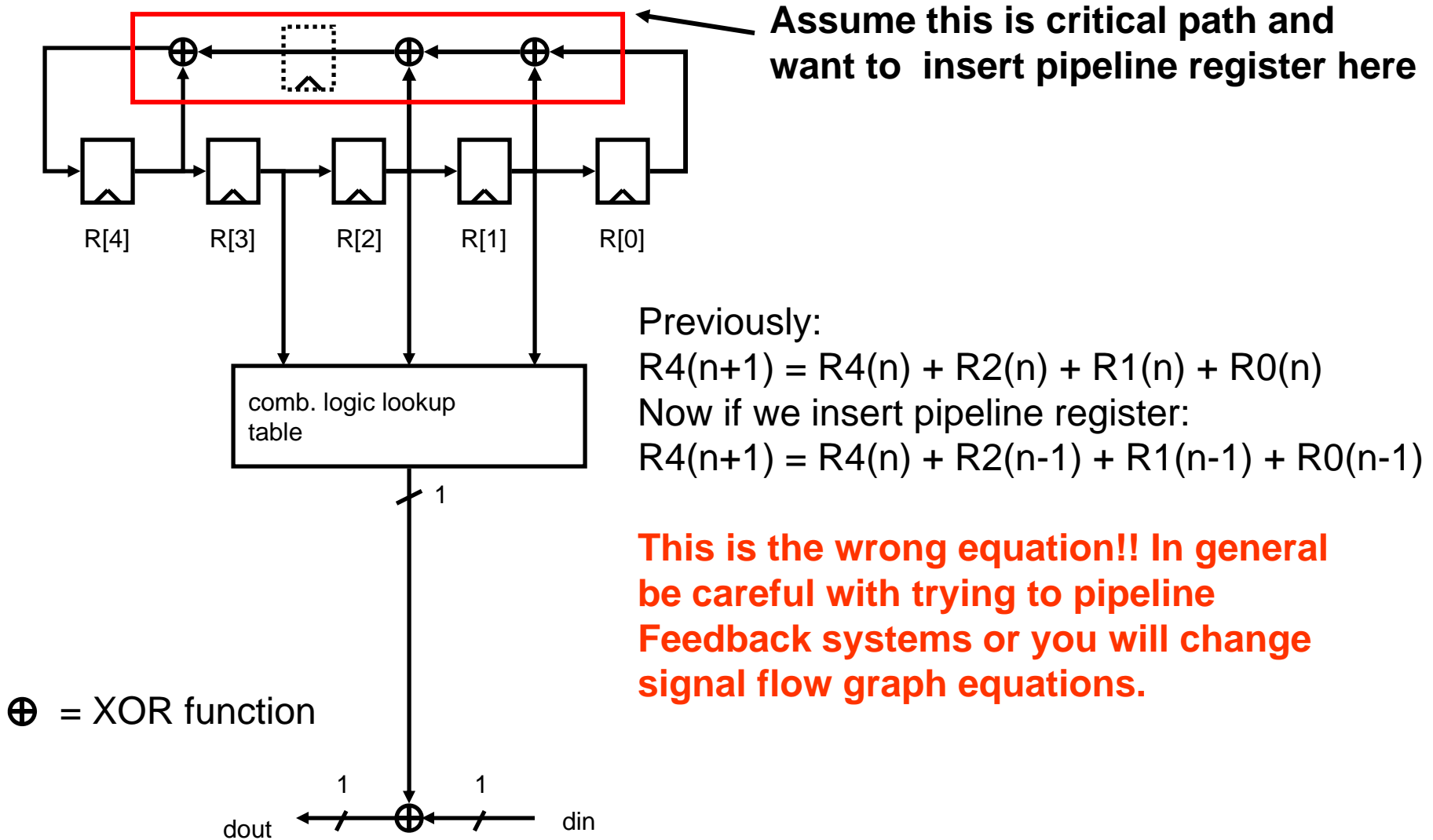
\oplus = XOR function



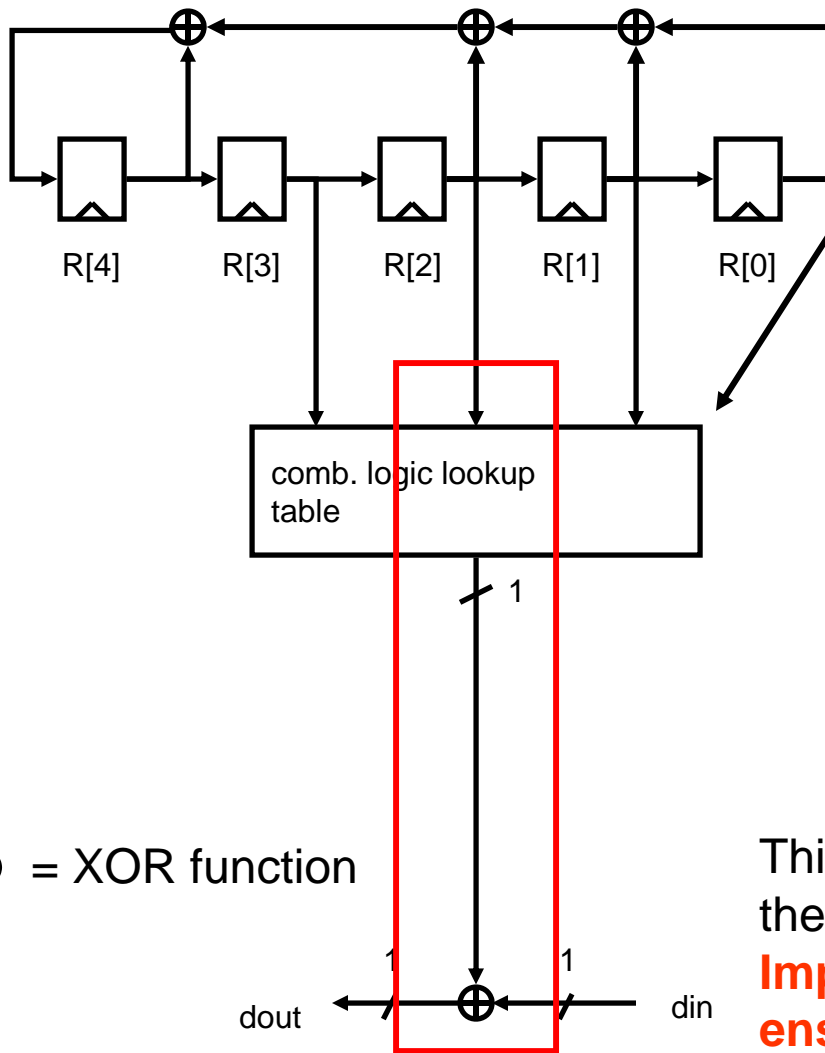
FEEDFORWARD SECTION

-generally can always be pipelined

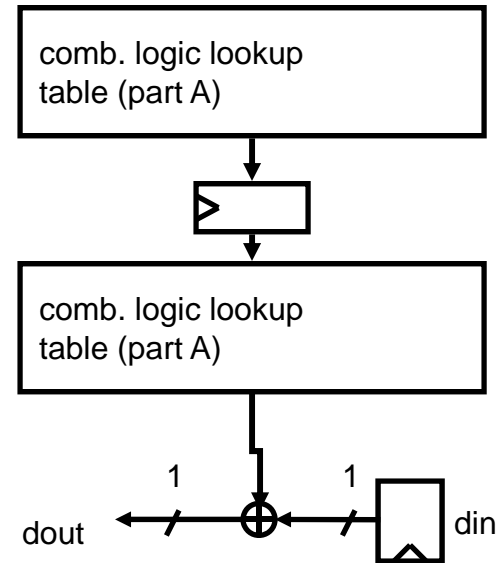
Pipelining Feedback Path



Pipelining Feedforward path



Assume this is critical path and want to insert pipeline register here to make:



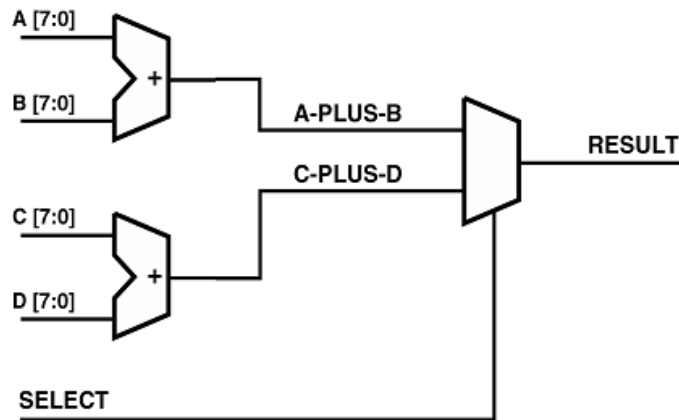
\oplus = XOR function

This does not change the functionality, only the latency so pipelining is fine

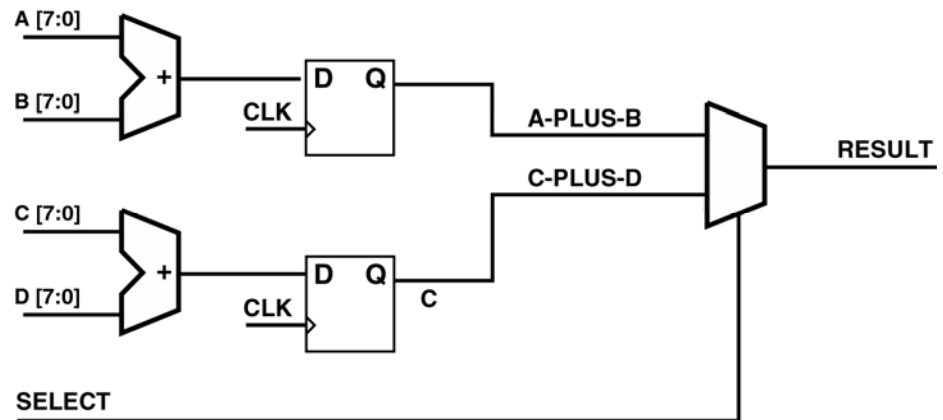
Important: Be sure to add register to din to ensure timing between paths is the same!

Review Questions

- Given the original circuit, what is wrong with the pipelined circuit?
- How can the problem be corrected?



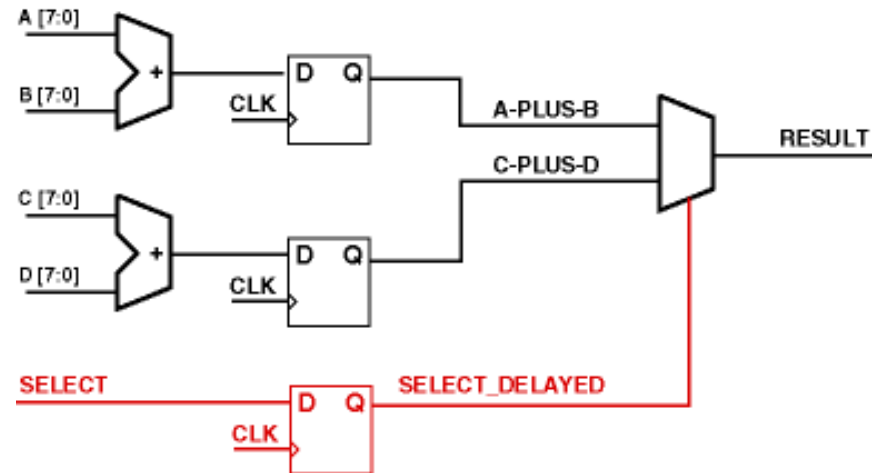
Original Circuit



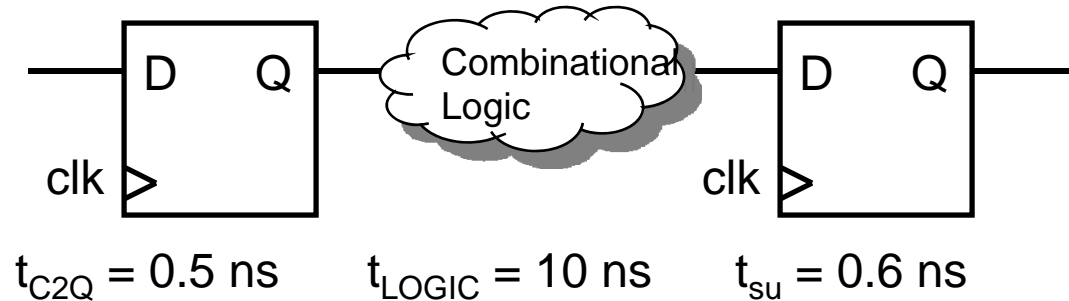
Piplined Circuit

Answers

- What is wrong with the pipelined circuit?
 - Latency mismatch
 - Older data is mixed with newer data
 - Circuit output is incorrect
- How can the problem be corrected?
 - Add a flip-flop on *SELECT*
 - All data inputs now experience the same amount of latency

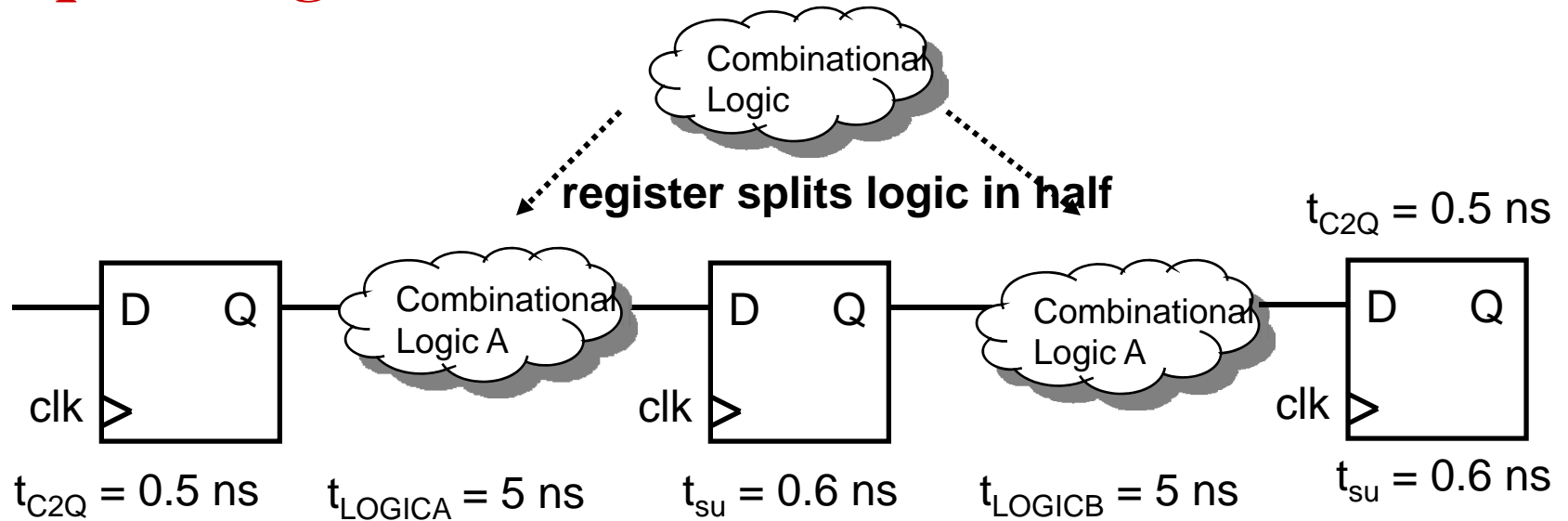


Pipelining: Real Life Issues



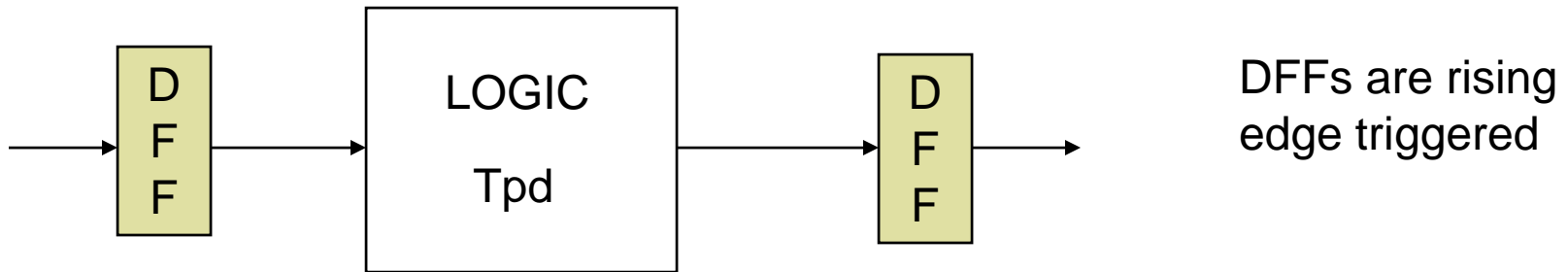
- In real designs cannot ignore t_{C2Q} and t_{su}
 - This circuit has $t_{critical_path} = 11.1 \text{ ns} \rightarrow$ maximum clock frequency = 90 MHz
 - Latency = 2 cycles

Pipelining: Real Life Issues



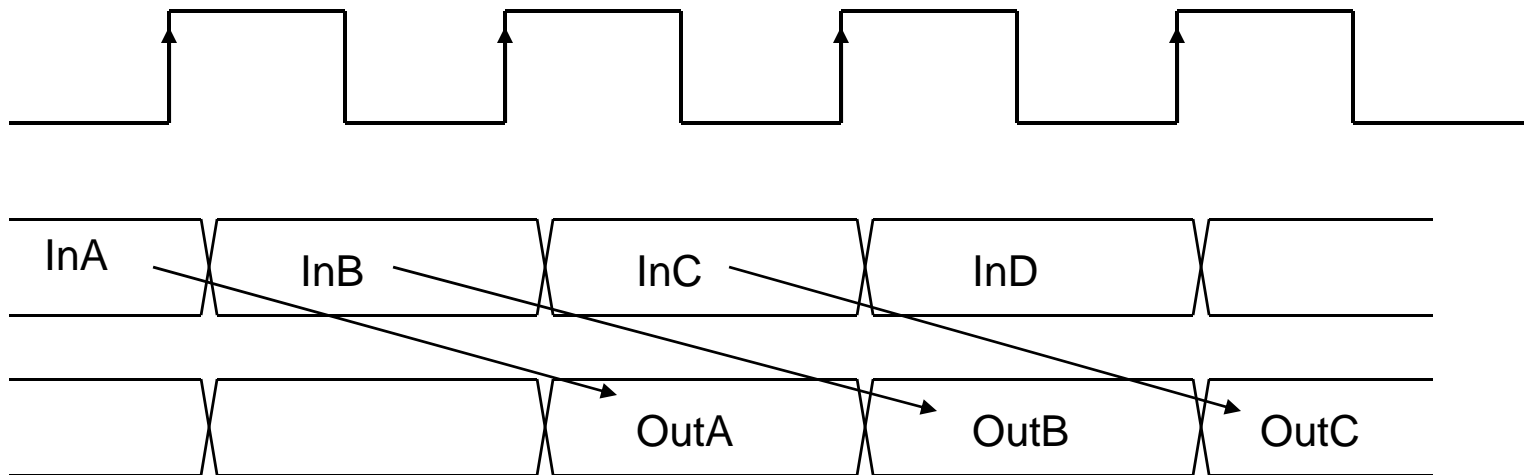
- This circuit has $t_{\text{critical_path}} = 6.1 \text{ ns} \rightarrow$ maximum clock frequency = 163 MHz
 - This is less than double the non-pipelined version due to t_{C2Q} and t_{su} (1.82x faster instead of "ideal" 2x faster)
 - Latency = 3 cycles
- Splitting the combinational logic into three "equal" logic sections of 3.33 ns (Latency = 4 cycles)
 - Maximum clock frequency = 227 MHz (2.27 times faster instead of "ideal" 3 times faster)
- **Important: t_{C2Q} and t_{su} cause real overhead!**
 - At some point, adding more pipeline stages does not increase clock frequency because t_{C2Q} and t_{su} dominate delay.
 - This is one reason pipelining does not solve all problems (Intel moving to multi-core)
- In real designs, usually LOGICA and LOGICB do not conveniently equal exactly half of LOGIC. Sometimes they are more, sometimes they are less depending on gates, wire delays, etc.
 - The worst-case path of t_{LOGICA} and t_{LOGICB} determines critical path

Registered Datapath

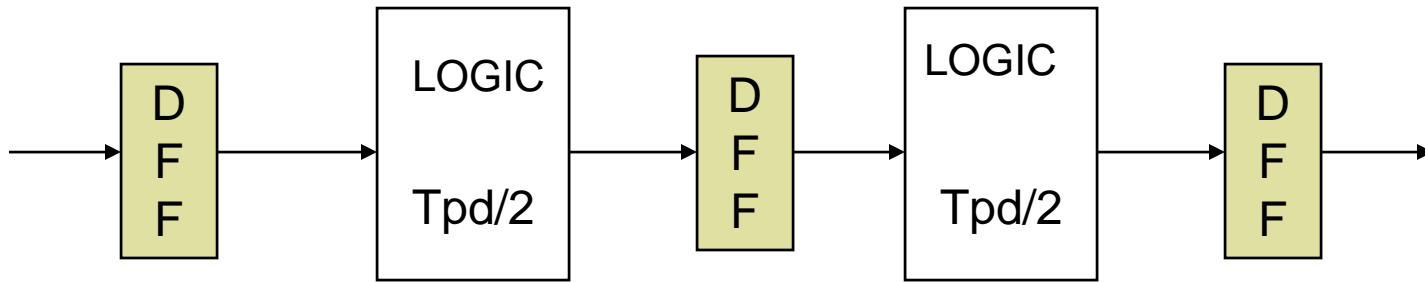


$$\text{Clk Freq} = 1 / (T_{c2q} + T_{pd} + T_{su})$$

Latency = 1 clk

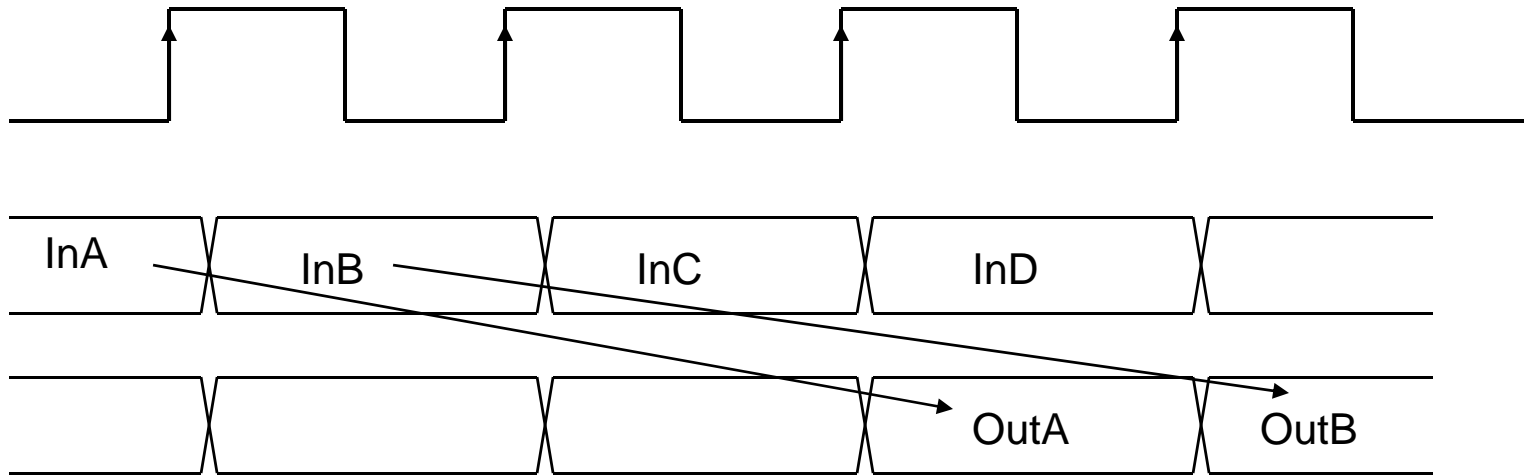


Add a Pipeline Stage



$$\text{Clk Freq} = 1 / (T_{c2q} + T_{pd}/2 + T_{su})$$

Latency = 2 clks



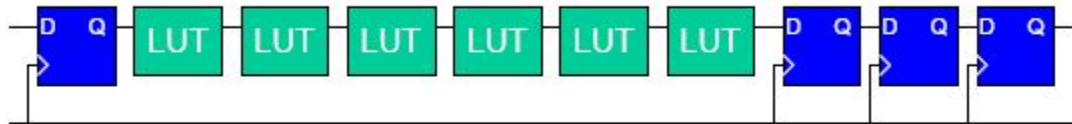
Pipelining Effect on Controller ASM

- Pipelining directly affect controller function ASM and the timing of the control signals
 - Make sure any pipeline stages added in your datapath are reflected in your ASM
 - If you are not careful, you may enable a register or counter one cycle too soon or too late!!
-

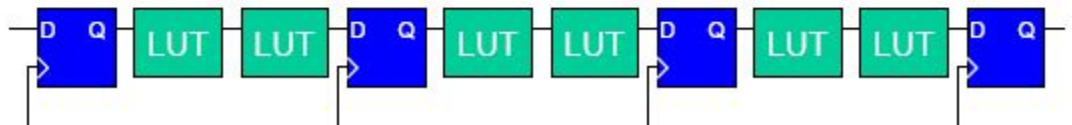
Synthesis Tool Support for Pipelining

- You can add pipeline registers manually or have the tool assist you
- Some tools have a pipelining option
 - This means you can leave a number of pipeline registers at the end of a combinational logic function and the tool will move the registers to achieve the best critical path delay

Before



After



Pipelining Summary

- In general, pipelining increases throughput at the cost of increased latency and area/power
 - Important: t_{C2Q} and t_{su} cause real overhead!
 - For beginners, do not pipeline feedback paths; pipeline feedforward paths only
 - If you add a pipeline register to one path, make sure all paths which require the same timing (converging or diverging paths) have pipeline registers as well to maintain correct timing
 - Also make sure your controller ASM reflects any pipeline registers added in your datapath else you may enable a counter one cycle too early or late, or your input/output data may be one cycle too early or late!
-