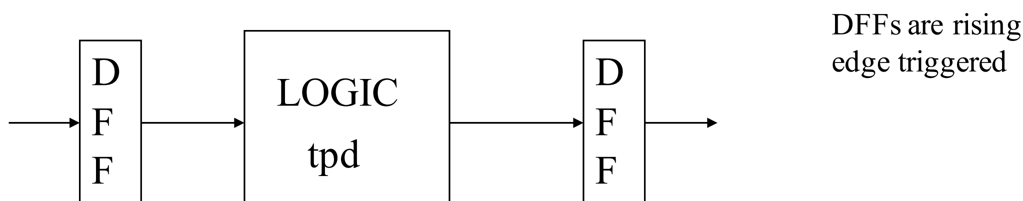
Speeding up the Clock

- The *register-to-register* delay is usually the delay path that sets the maximum clock rate
- From a design point of view, can only modify the *combinational logic* between the registers
  - Need to shorten the maximum combinational delay path
  - Setup/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*
  - Adds *latency* to the output (the number of clocks between an input value and its corresponding output result)

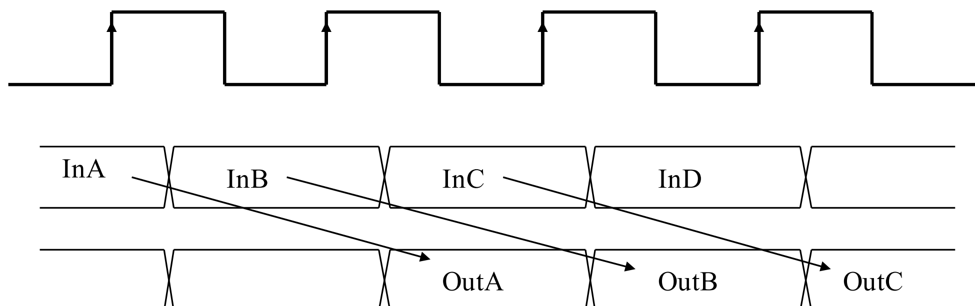
Registered Datapath



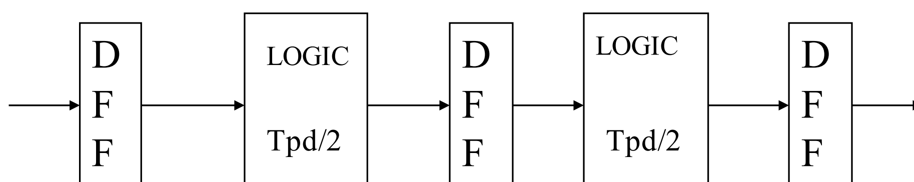
DFFs are rising edge triggered

Clk Freq = 1/ (Tclk2q + Tpd + Tsu)

Latency = 1 clk

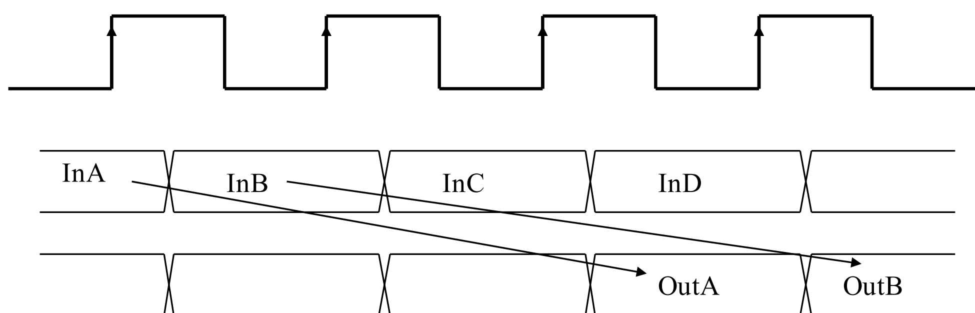


Add a pipeline stage



Clk Freq = 1/ (Tclk2q + Tpd/2 + Tsu)

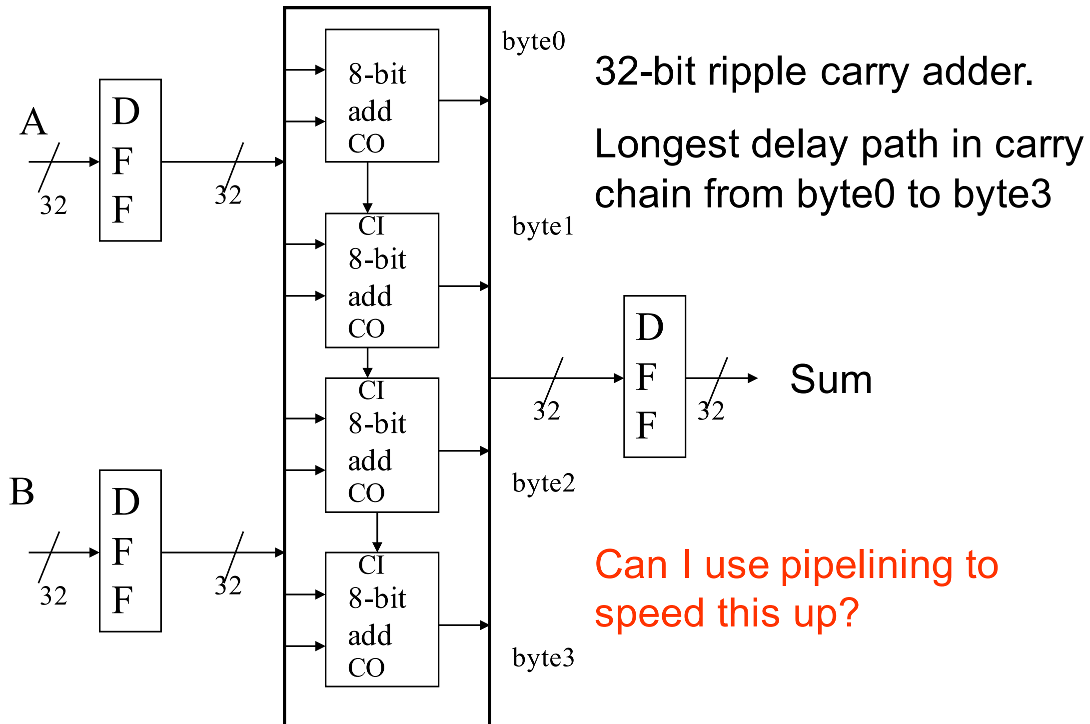
Latency = 2 clks



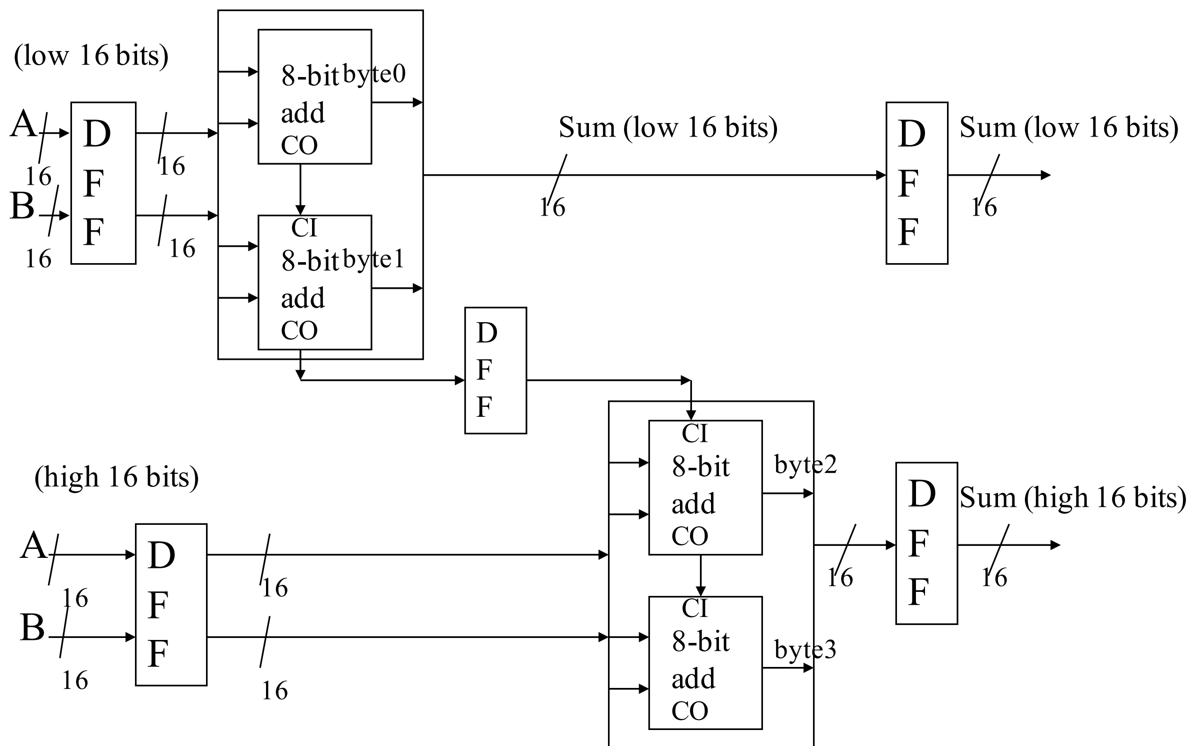
Definitions

- **<u>Initiation Rate</u>** - Rate at which new input values are accepted
  – Rate at which new computations are initiated
  – Minimum initiation rate = 1
- **<u>Latency</u>**
  – Number of *<u>clock cycles</u>* between input value and output value
  – Adding pipeline stages always increases latency
- At some point, adding more pipeline stages does not increase clock frequency because Tclk2q and Tsu dominate delay.

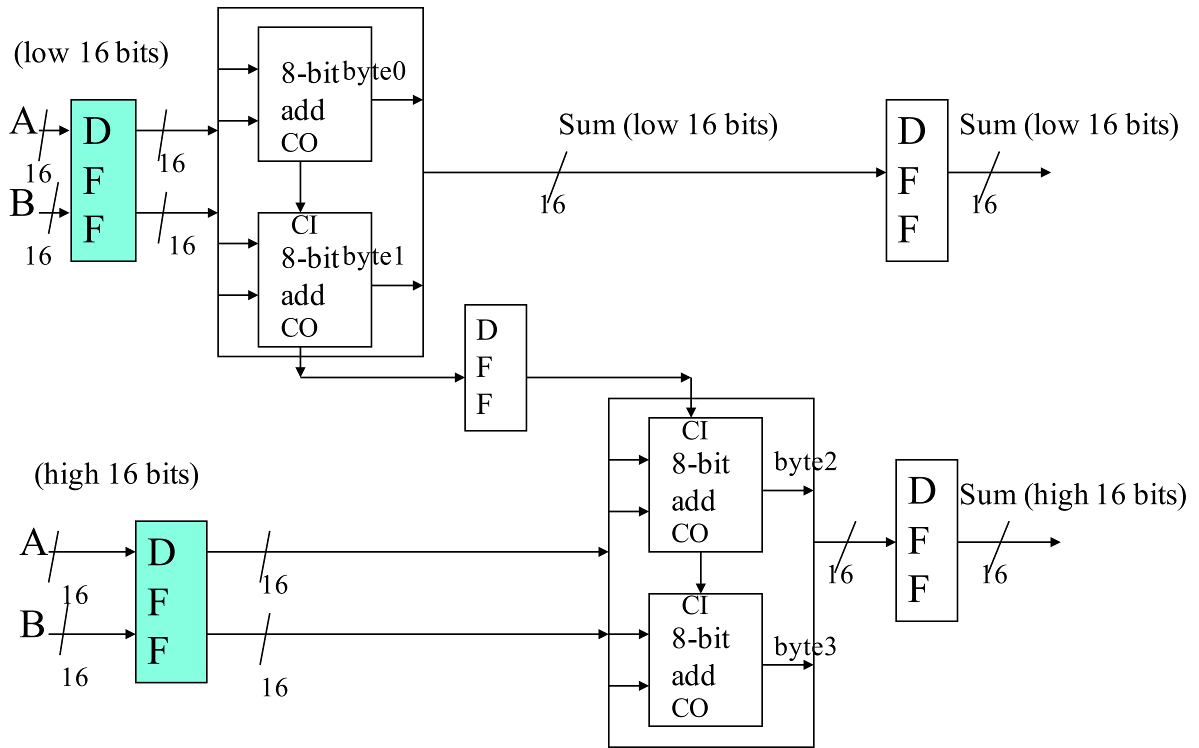
Pipeline Example

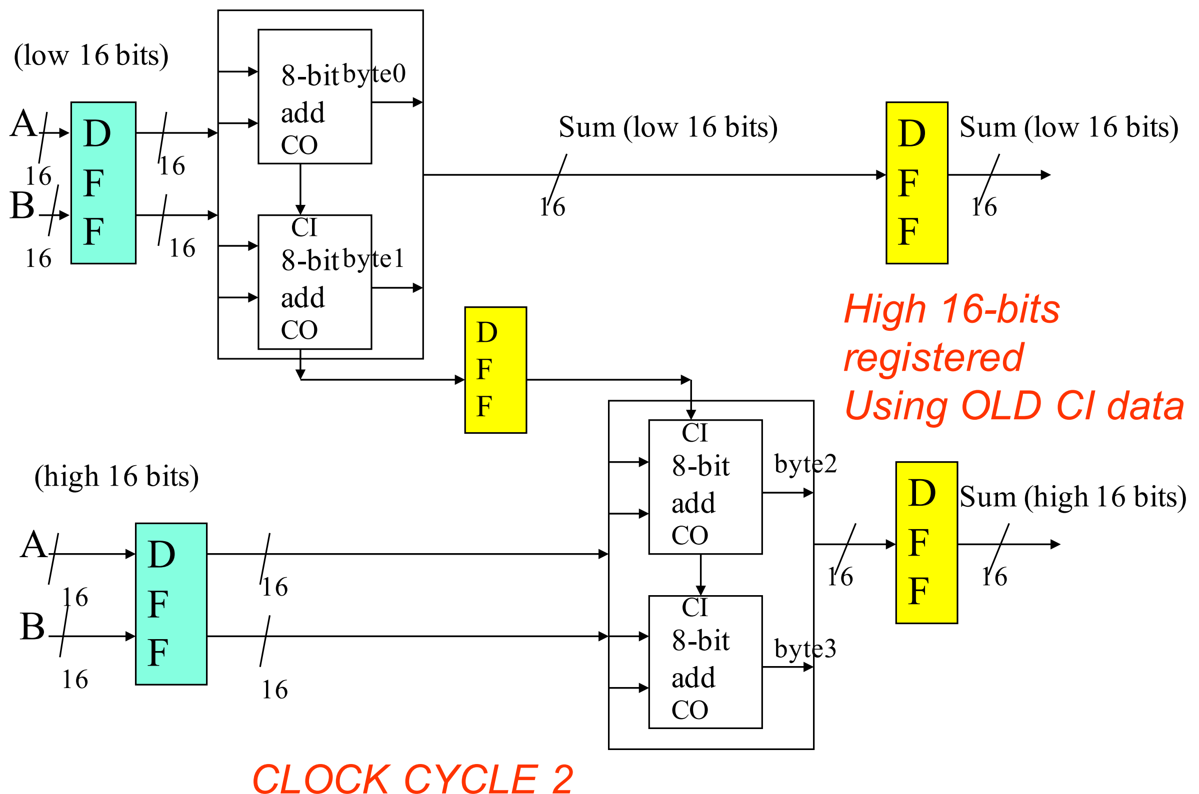


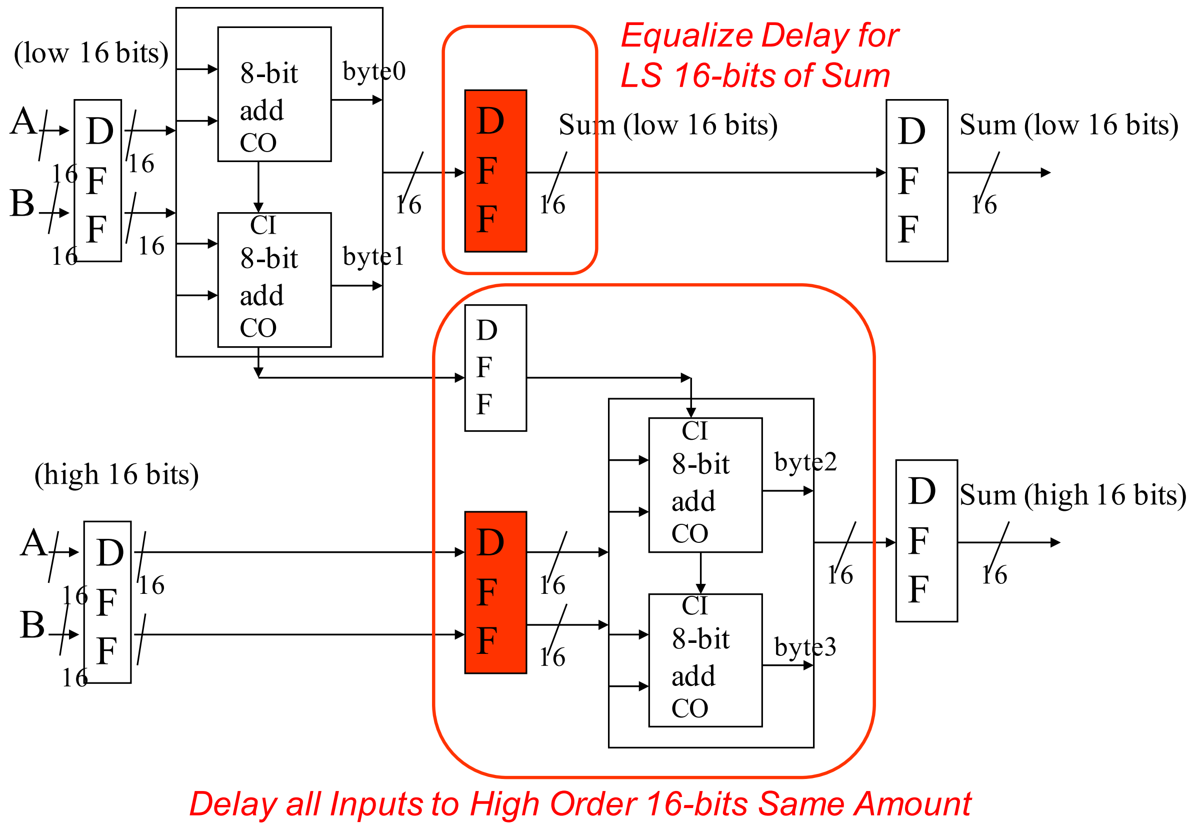
Insert pipeline stage between byte1 and byte2

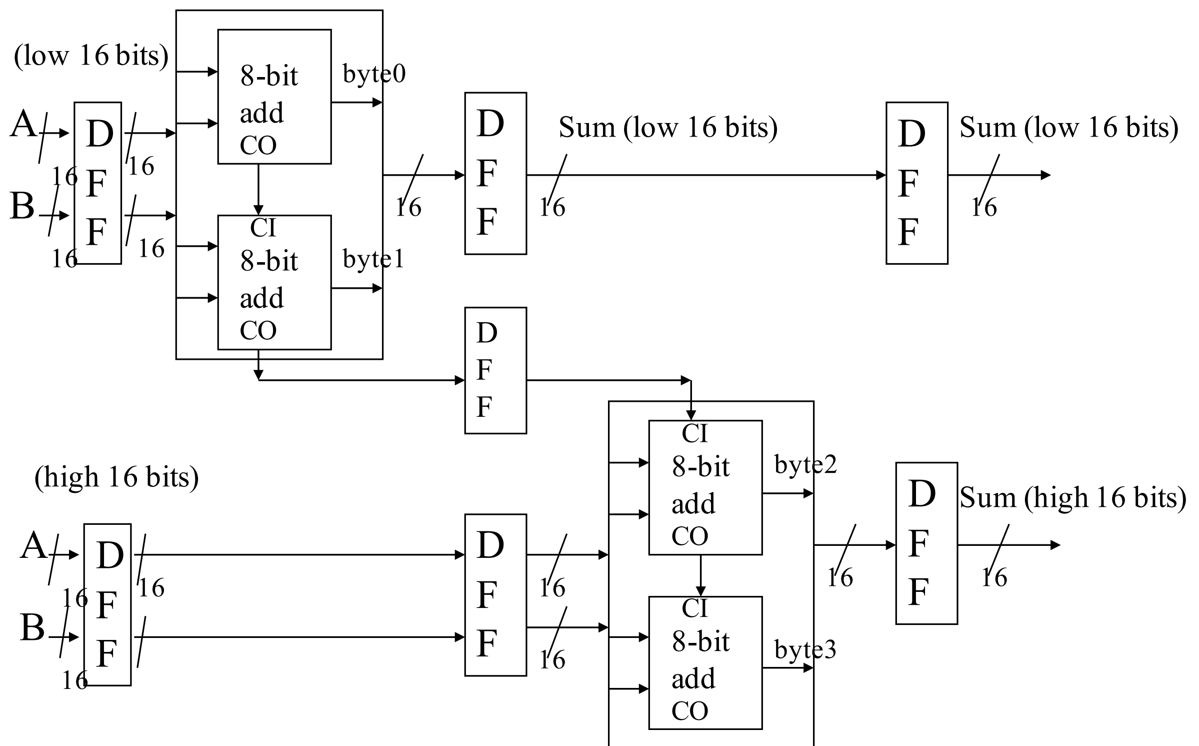


WILL THIS WORK PROPERLY?


Insert pipeline stage between byte1 and byte2
(low 16 bits)
A
B
16
16
DFF
16
16
8-bit add CO
byte0
CI
8-bit add CO
byte1
Sum (low 16 bits)
16
DFF
Sum (low 16 bits)
16
DFF
(high 16 bits)
A
B
16
16
DFF
16
16
CI
8-bit add CO
byte2
CI
8-bit add CO
byte3
16
DFF
Sum (high 16 bits)
16
CLOCK CYCLE 1


Insert pipeline stage between byte1 and byte2
(low 16 bits)
A
B
16
16
DFF
16
16
8-bit add CO
byte0
CI
8-bit add CO
byte1
Sum (low 16 bits)
16
DFF
Sum (low 16 bits)
16
High 16-bits registered Using OLD CI data
DFF
(high 16 bits)
A
B
16
16
DFF
16
16
CI
8-bit add CO
byte2
CI
8-bit add CO
byte3
16
DFF
Sum (high 16 bits)
16
CLOCK CYCLE 2


Insert pipeline stage between byte1 and byte2
Equalize Delay for
LS 16-bits of Sum
(low 16 bits)
A
B
16
16
DFF
8-bit add CO
CI 8-bit add CO
byte0
byte1
Sum (low 16 bits)
Sum (low 16 bits)
(high 16 bits)
CI 8-bit add CO
byte2
byte3
Sum (high 16 bits)
Delay all Inputs to High Order 16-bits Same Amount


Insert pipeline stage between byte1 and byte2
(low 16 bits)
A
B
16
16
DFF
8-bit add CO
CI 8-bit add CO
byte0
byte1
Sum (low 16 bits)
Sum (low 16 bits)
(high 16 bits)
CI 8-bit add CO
byte2
byte3
Sum (high 16 bits)

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'*

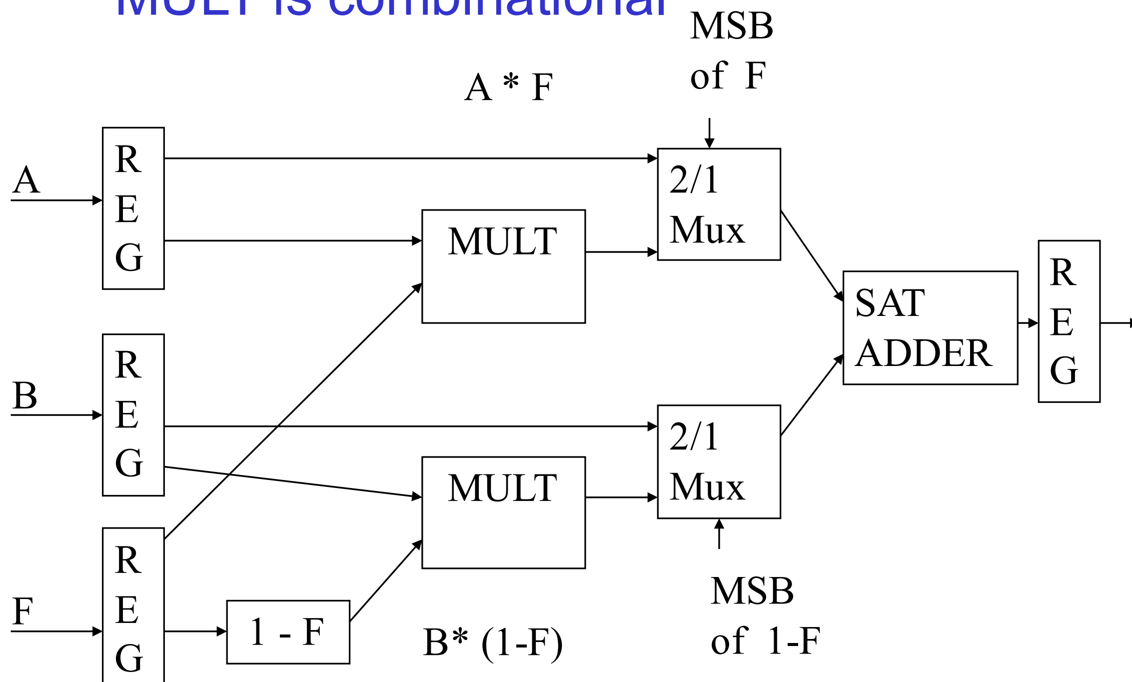
Latency Tolerance

- 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 and makes an application latency intolerant.
- Flushing of a pipeline introduces clock cycles in which the results coming out of the pipeline are ignored -- these are wasted clock cycles.
- High Latency tolerance means that you can have many pipeline stages, whatever the number you need to meet the clock rate specification.

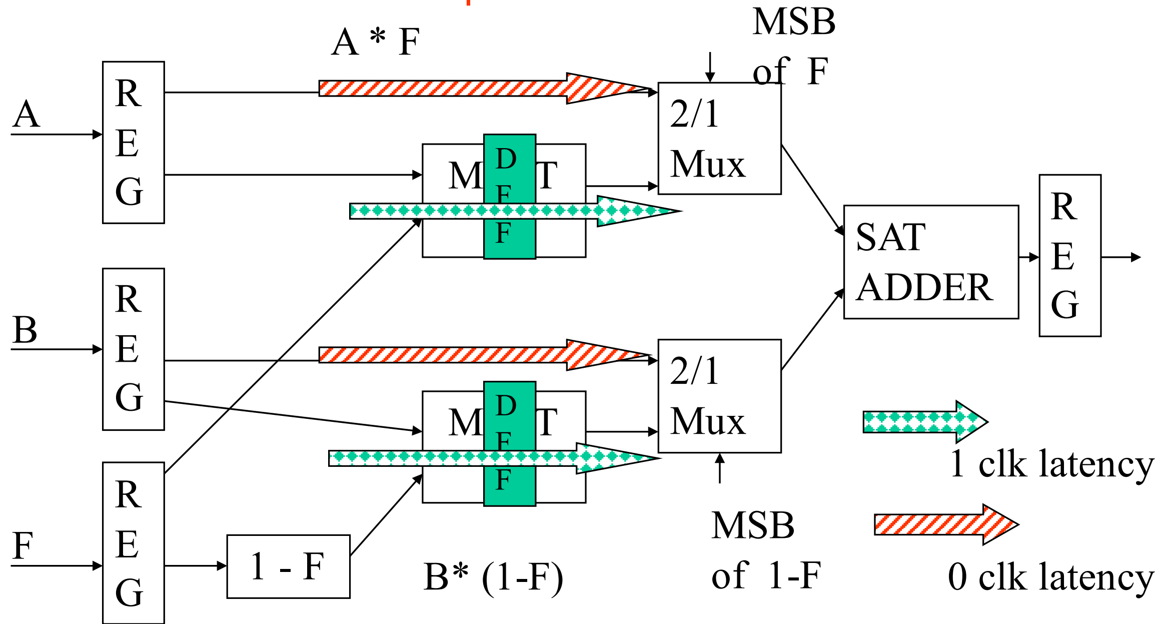
Two Applications

- 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.

BLEND Datapath without Pipelining. MULT is combinational



LPM_Mult can be pipelined via LPM_PIPELINE parameter. Add one pipeline stage to LPM Mult. DFFs inserted automatically in LPM_MULT. We now have a LATENCY mismatch within our datapath!!!!



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 LPM_mult to meet clock frequency target.