Digital Logic
Design with FPGA

FPGA Design Optimization

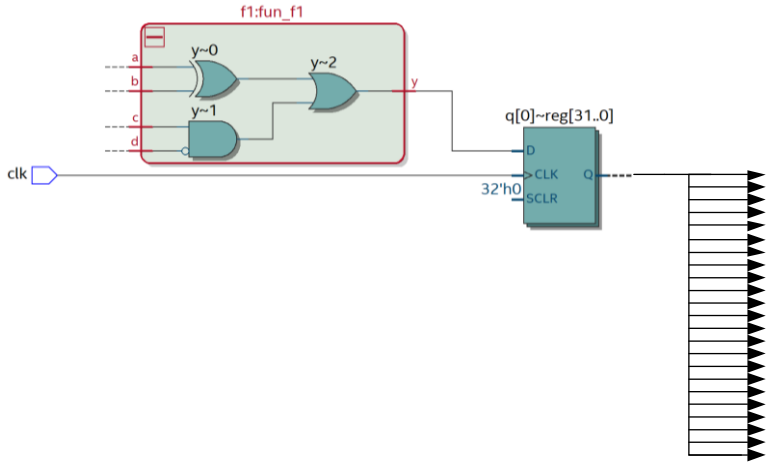
Outline

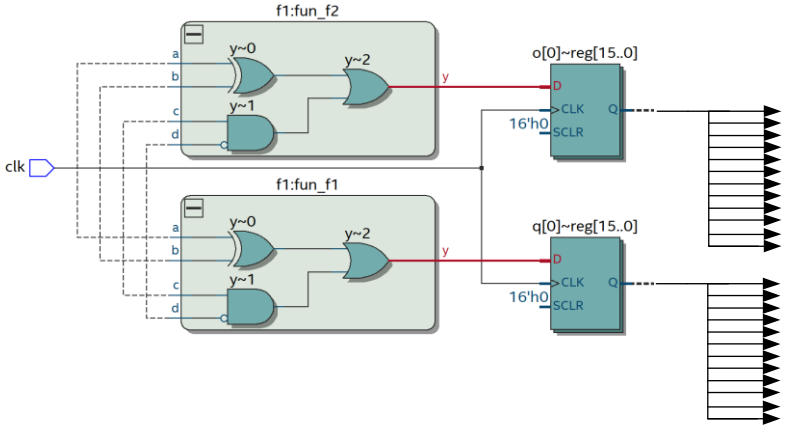
- Registers Duplication
- Pipelining
- IO Registers
- Retiming
- Design Guidelines

Registers Duplication

- Highly loaded (fanout) networks are slow and difficult to route
- Registers duplication can fix both of these problems
  - Reducing fanout will reduce network delays
  - Duplicating functions will reduce the density of connections in a given area of the chip

- Increasing performance at the expense of increasing resources used

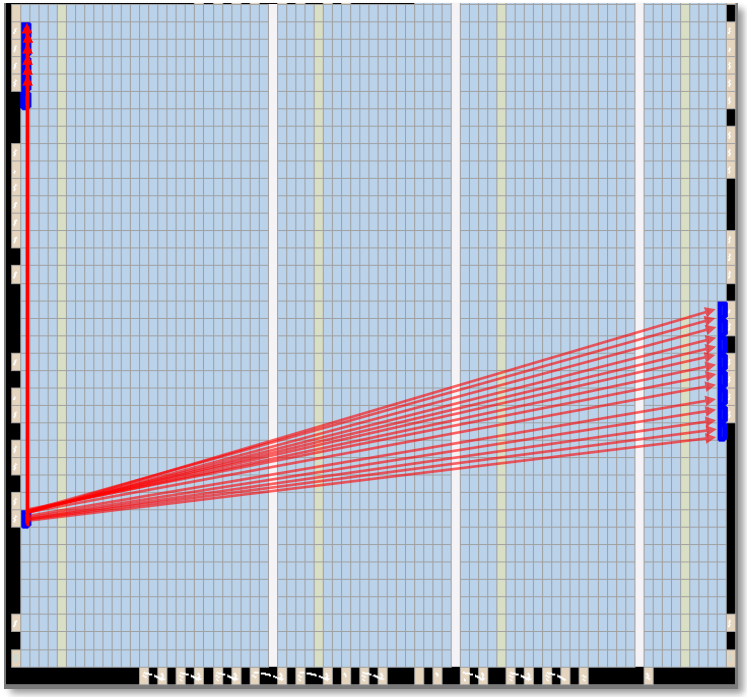




Registers Duplication

The source circuit controls two banks of registers embedded in different regions of the chip

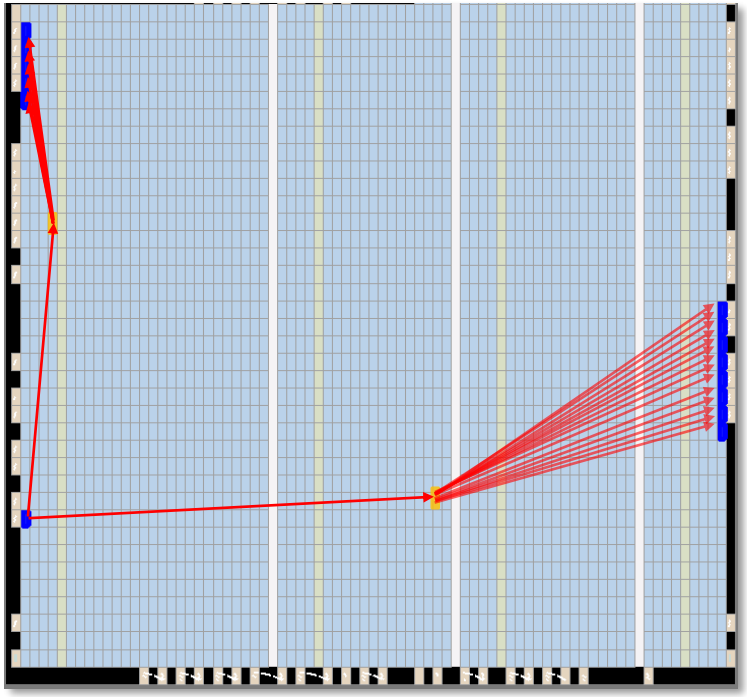
- Condition: The source flip-flop is not tightly bound to the current location
- Timing requirement for clock period (PERIOD = 6 ns)
- After implementation with default settings, the longest path was obtained = 7.4 ns (exceeds the desired clock period)



Registers Duplication

Project status after duplication of the register and control function

- Each flip-flop controls a different region of the system
  - This allows the FF to be located closer to the controlled object
  - We get shorter routing lines (and with lower capacity)
- After implementation, the longest path was obtained = 5.4 ns (the system meets the time requirements)

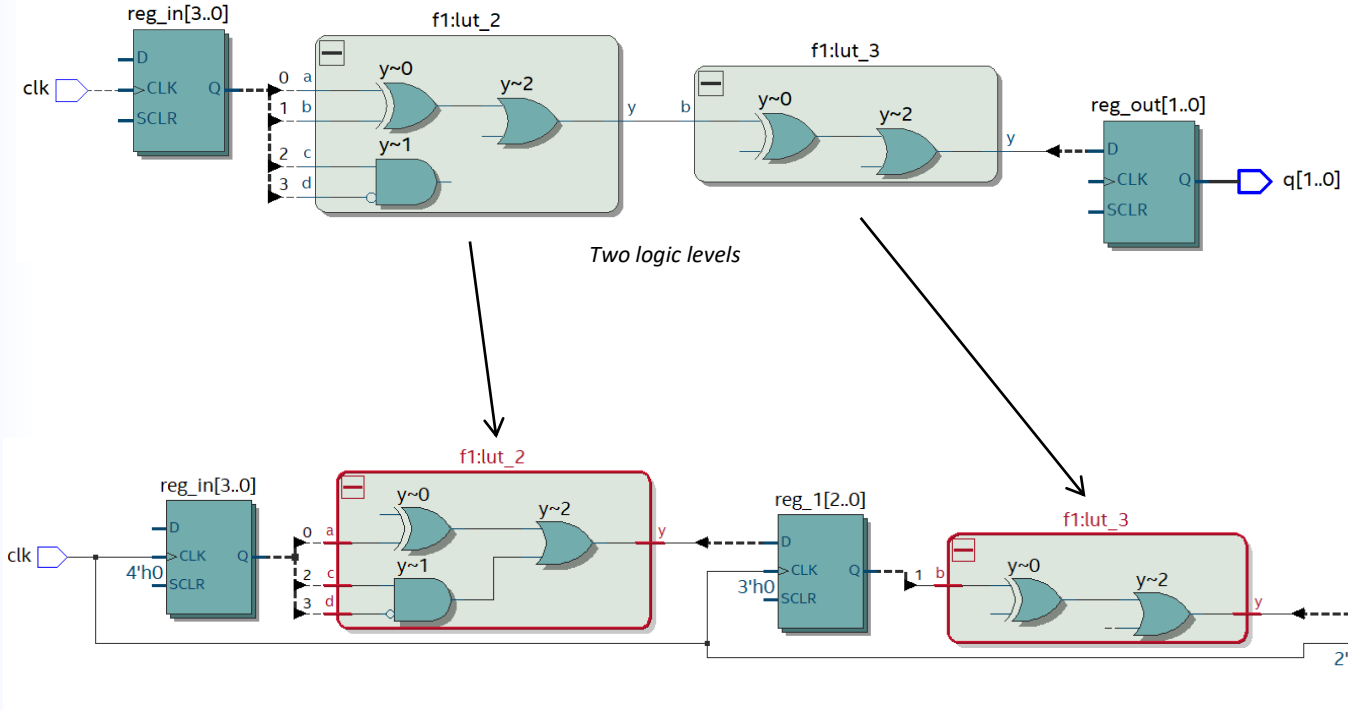


Outline

- Registers Duplication
- Pipelining
- IO Registers
- Retiming
- Design Guidelines

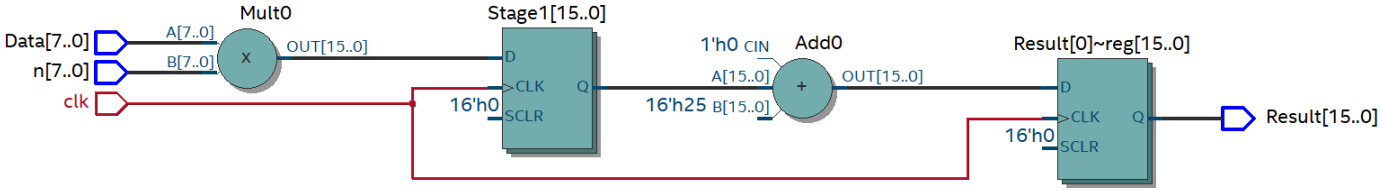
Pipelining

- Introducing additional registers in the data path increases processing efficiency



Reducing the number of logic levels
(LUTs) between adjacent registers

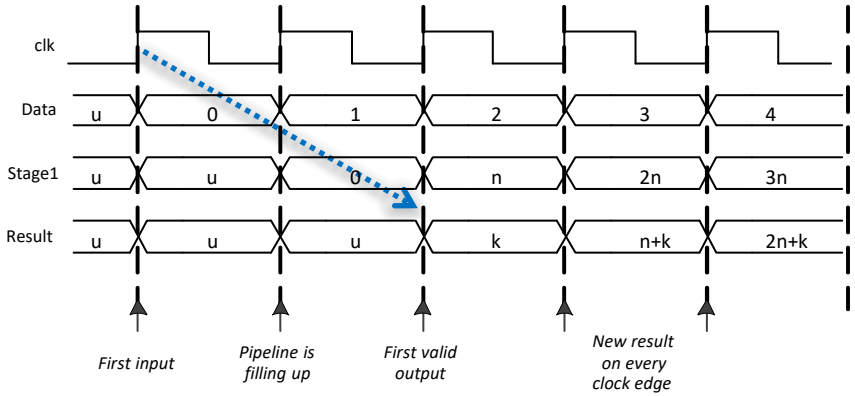
Latency in pipeline processing



- Each stage of registers in the pipeline introduces an additional clock cycle delay before the first valid data appears at the device output
- This is a "flow-filling" phenomenon. Once filled, the correct data appears at the output on each subsequent clock cycle.
- Processing latency (pipeline latency) is equal to the number of clock cycles (number of pipeline stages). It is sometimes referred to as "stream depth".

Result = (n*Data) + k

k - constant

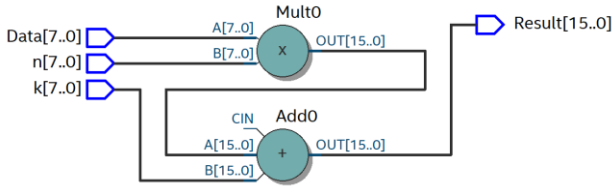


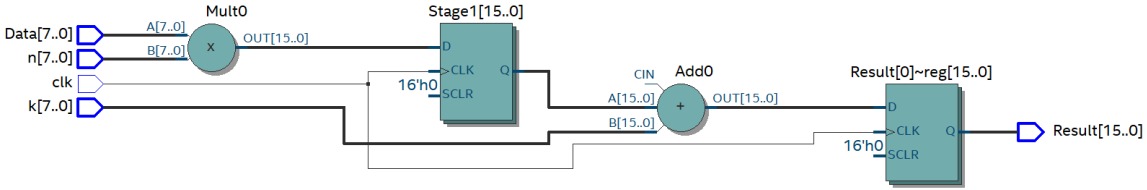
Pipeline guidelines

- Is there multi-level logic between adjacent registers?
- Does the designed system tolerate delays (latency)?
- Are the resources of the designed system sufficient to implement pipelines?
  - Are there enough free flip-flops/connections to support many additional stages of the pipeline?
  - This is a problem that rarely occurs in FPGA systems due to their construction…

Case 1

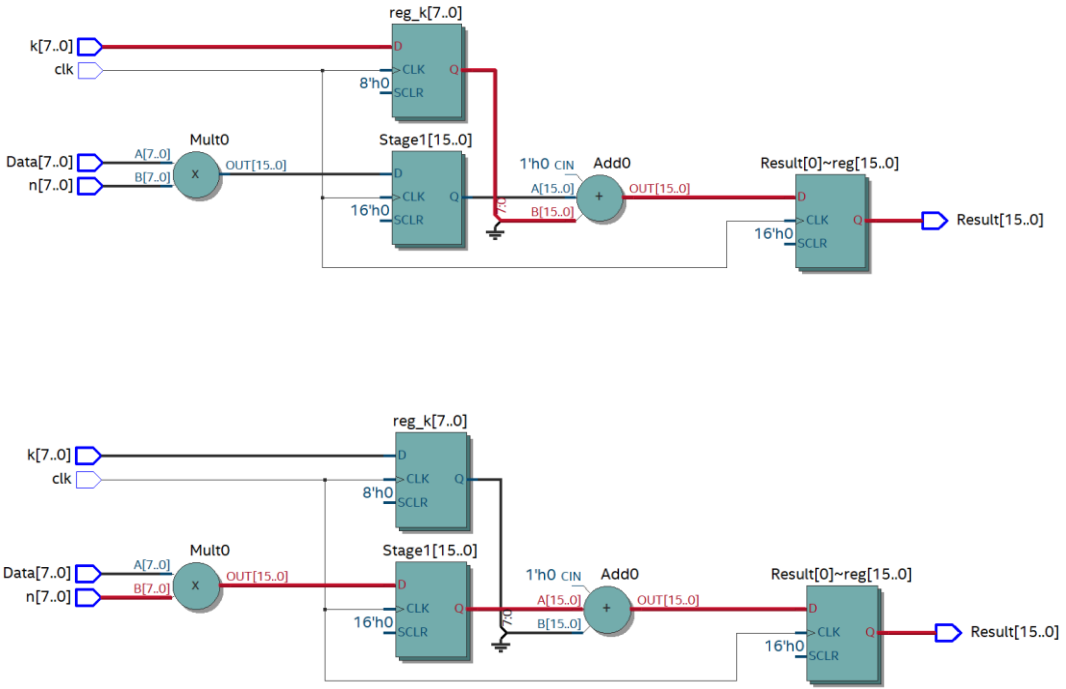
- The diagrams show two systems:
  the original one and the one with introduced pipeline registers.
- After introducing pipeline processing, the system gives incorrect results.
- Why? How can I fix it?





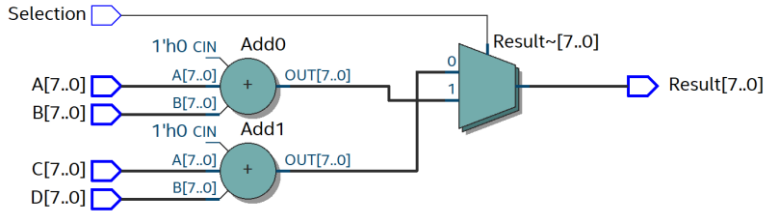
Case 1

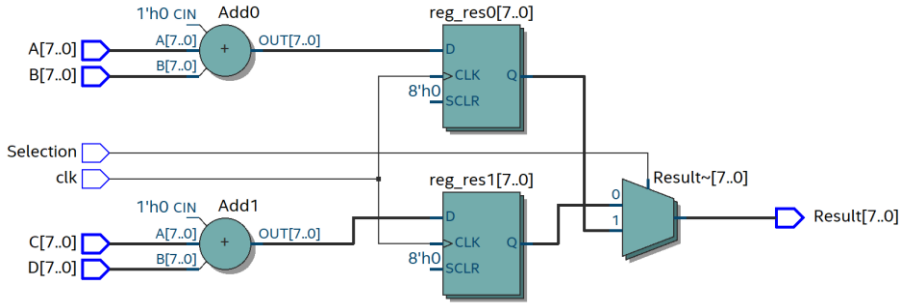
- The delay in each path between the input and output of the circuit must be equal
- Failure to follow this rule leads to processing errors (mixing samples from different times)



Case 2

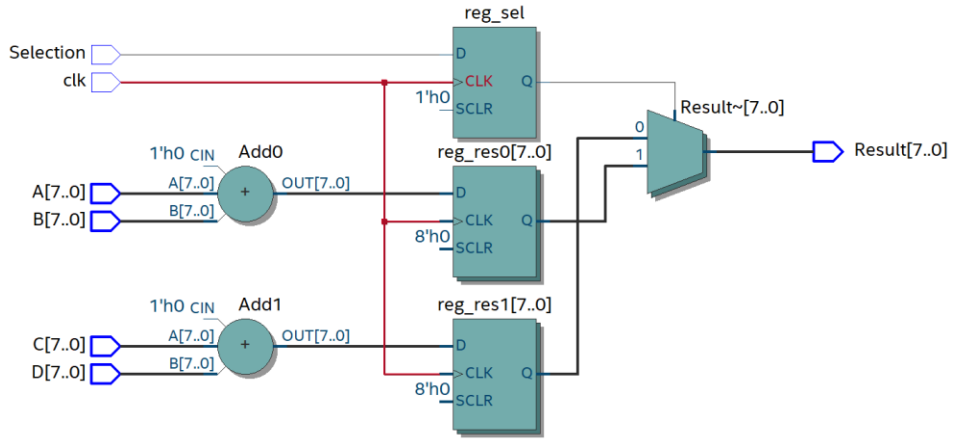
- The diagrams show two systems:
  the original one and the one with introduced pipeline registers.
- After introducing pipeline processing, the system gives incorrect results.
- Why? How can I fix it?





Case 2

- The main problem is the incorrect introduction of pipelines
- Input data is transferred to the output with a different delay (latency mismatch)
- Delay on lines A,B,C,D = 1, delay on Selection line = 0, this mixes new data with data from the previous processing cycle
- This is a common mistake designers make.
  Pipeline registers must be entered in both the data path and the control path.

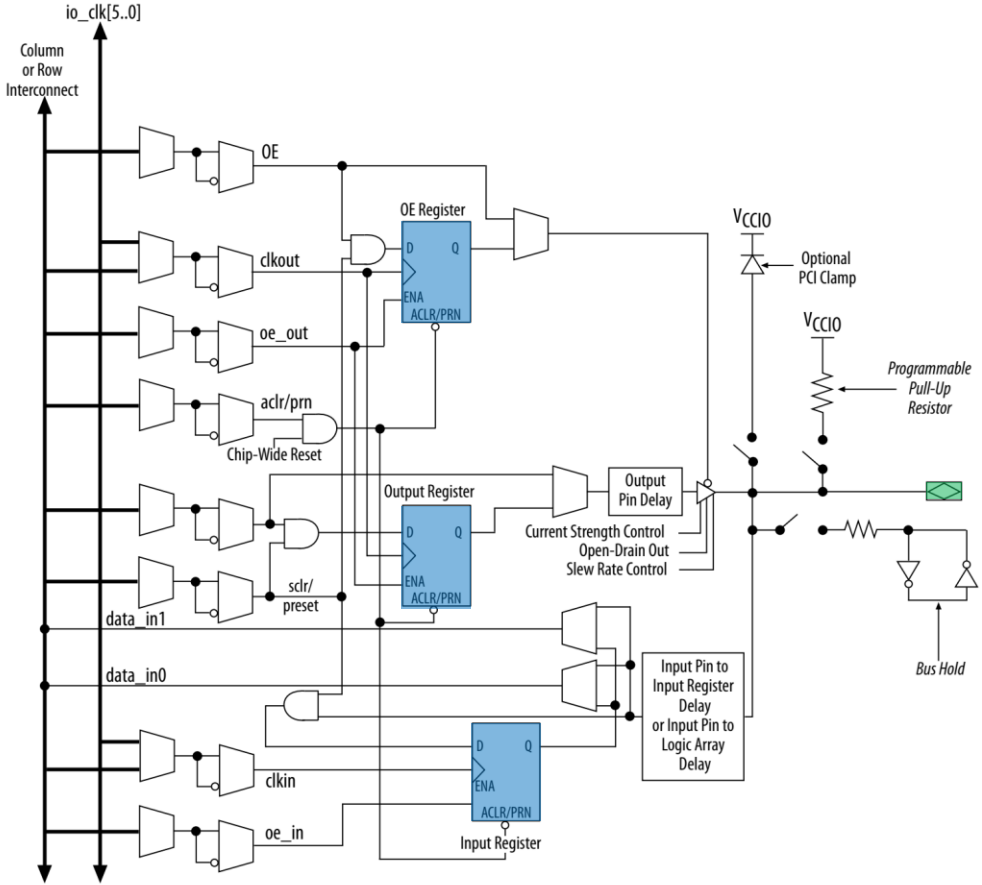


Outline

- Registers Duplication
- Pipelining
- IO Registers
- Retiming
- Design Guidelines

I/O elements (IOEs)

- MAX 10 I/O elements (IOEs) contain a bidirectional I/O buffer and five registers for registering input, output, output-enable signals
- Transfer mode: bidirectional single data rate (SDR) and double data rate (DDR)
- All I/O block flip-flops have guaranteed setup time and hold time
  (and clock-to-out times if the clock signal is passed through the global clock buffer)



source: Intel® MAX® 10 General Purpose I/O User Guide, https://www.intel.com/content/www/us/en/docs/programmable/683751/22-1/i-o-overview.html

Access to I/O registers

- During synthesis
  - Timing-driven synthesis can force the boundary flip-flops to be moved to I/O elements
- During fitting (implementation phase)
  - 'Optimize IOC Register Placement for Timing' 'Normal' option is enabled by default: the Fitter will opportunistically pack registers into I/Os that should improve I/O timing

Optimize IOC Register Placement for Timing	Pack All IO Registers

- The location of the I/O registers is checked in the report after Fitting (Partition Statistics).
  - 'Total registers' section

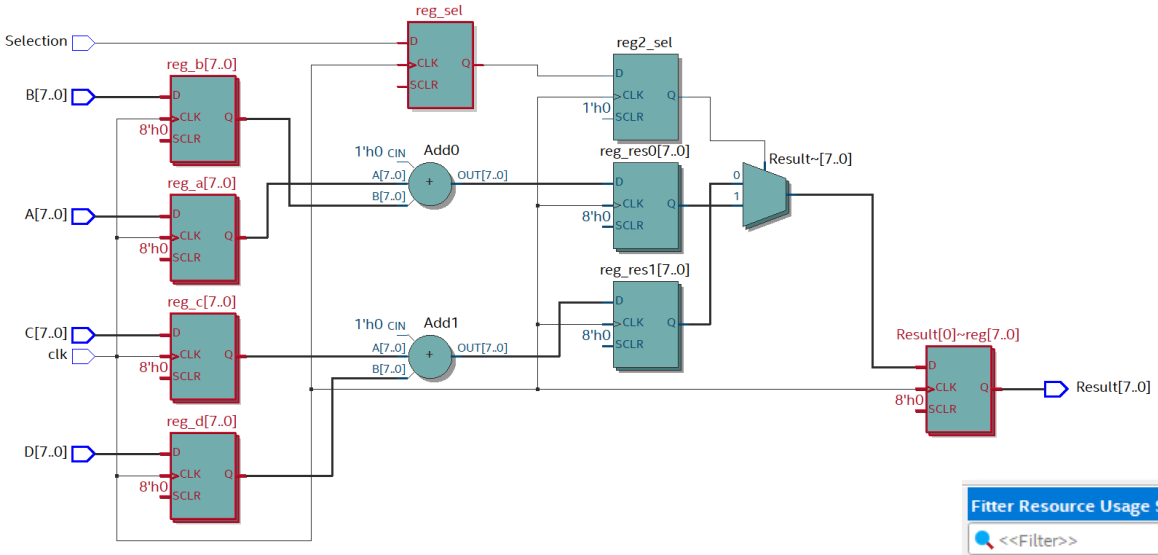
Fitter Partition Statistics

<<Filter>>

	Statistic	Top	hard_block:auto_generated_inst
9	Total registers	16	0
1	-- Dedicated logic registers	16 / 8064 (< 1 %)	0 / 8064 (0 %)
2	-- I/O registers	0	0

Case 3

- Registering input and output signals increases the depth of the pipeline
- Moving the edge registers to the I/O blocks allows for shortening the delay times at the inputs of the integrated circuit



Fitter Resource Usage Summary

<<Filter>>

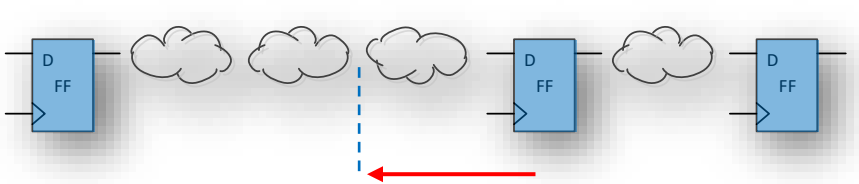
	Resource	Usage
7	Total registers*	58 / 9,287 (< 1 %)
1	-- Dedicated logic registers	17 / 8,064 (< 1 %)
2	-- I/O registers	41 / 1,223 (3 %)

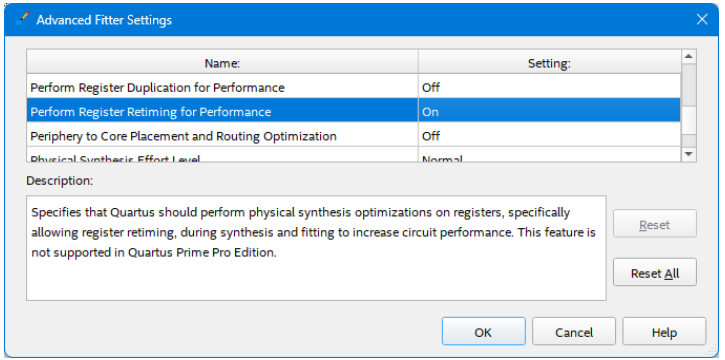
Outline

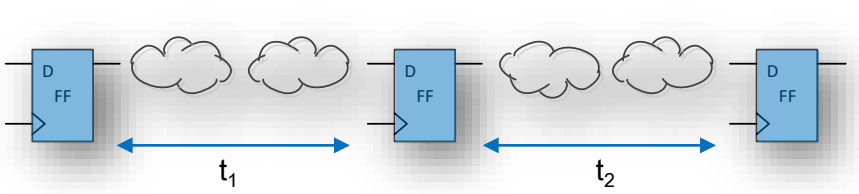
- Registers Duplication
- Pipelining
- IO Registers
- Retiming
- Design Guidelines

Retiming

- A technique of moving registers to improve timing efficiency
- Synthesis tools can shift the register to balance the delay of the combinational path seen from its input and output
- In some tools, a technique called 'Register Balancing'
- It can reduce or increase the number of registers used in the project (retiming can create multiple registers at the input of a combinational block from a register at the output of a combinational block)







$t_1 \sim t_2$

Outline

- Registers Duplication
- Pipelining
- IO Registers
- Retiming
- Design Guidelines

Design Guidelines

- Use pipeline processing

  increases the bandwidth

- Use synchronous control signals

  increases reliability

- Use inference in resource placement
  - Multiplexers
  - Shift registers
  - Memories (block and distributed)
  - DSP blocks
  - …

  resource optimization / Design for Reuse

Instantiation vs. Inference

- Allow synthesis tools to infer FPGA resources wherever possible
  - Code written this way becomes more portable (DFR)
- In some cases, the synthesis tool cannot 'infer' or incorrectly infers resources
  - In such situations, you must explicitly instantiate the desired component
- Many design tools offer ready-made components (IP-cores) that must be instantiate in the project
  - Available as a netlist and embedding pattern of the generated component in various HDL languages

Design guidelines: HDL description

- Avoid high-level syntax
  - Results from synthesis tools may be suboptimal
- Avoid repeatedly entering if-then-else structures
  - Most tools implement them in parallel
  - Executing if-then-else multiple times can lead to the creation of priority encoders
- Use the case structure to build decoders and finite state machines
  - Better results than if-then-else
- Order and group arithmetic/logic operations
  - A <= B + C + D + E; should be: A <= (B + C) + (D + E)
- Avoid entering unwanted latches
  - Specify the value of all outputs in all possible branches
    (e.g. by substituting default values before if-then-else and case)

Design tips: synthesis

- Use time restrictions
  - Define strict (but realistic!) limits on the clock signal
  - Place independent clock signals in different groups

- Use the appropriate synthesis tool options
  - Disable resource sharing
  - Move boundry registers closer to logic (or introduce additional I/O registers)
  - Enable FSM optimization
  - Use time path balancing (retiming)

Summary

- Performance-boosting techniques:
  - Duplication of registers
  - Adding pipeline stages
  - Use of input/output registers
  - Balancing delays
- Side effects:
  - Register duplication increases the device size
  - Piping introduces processing latency and increases resource consumption
  - Retiming changes the number of registers used in the design