Digital Logic
Design with FPGA

Synchronous circuits design

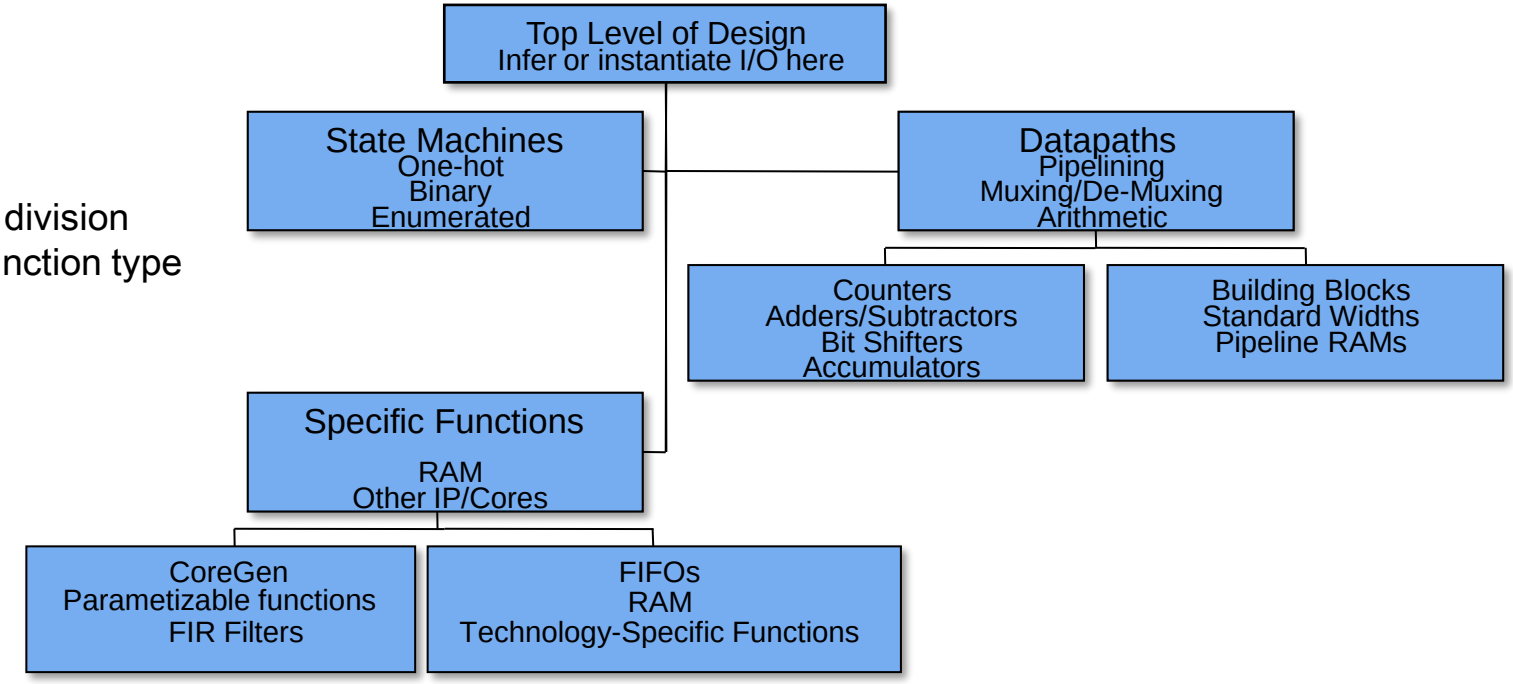
Outline

- Good design practices
- Techniques for designing synchronous systems
- Construction of synchronization circuits

Hierarchical structure of the project

- The use of hierarchy allows to increase the readability of the design
- Correct division into functional modules facilitates reuse and debugging

Example of division based on function type



Design tips: Increase the readability of your design

- Create hierarchical blocks based on:
  - Functional consistency within the block
  - Minimal routing between blocks
- Use meaningful labels for blocks and the signals
- Separate independent clock domains
  - Define clear relationships between clock signals
- Keep source file sizes reasonable
  - Makes synthesis and debugging easier

Design tips: Design for Reuse

- Define a set of core modules common to all projects
  - Register banks
  - FIFO memories
  - Other standard features
- Name library elements based on their functions and implementation technique
  - Makes it easier to find the item you want
  - Example: REG_16X8_M10 (bank of 16 8-bit registers dedicated to the MAX10 platform)
- Store defined libraries/packages/blocks in a directory independent of the EDA tools you use
  - Makes sharing easier
  - Protects against damage/destruction, e.g. when updating tools

Outline

- Good design practices
- Techniques for designing synchronous systems
- Construction of synchronization circuits

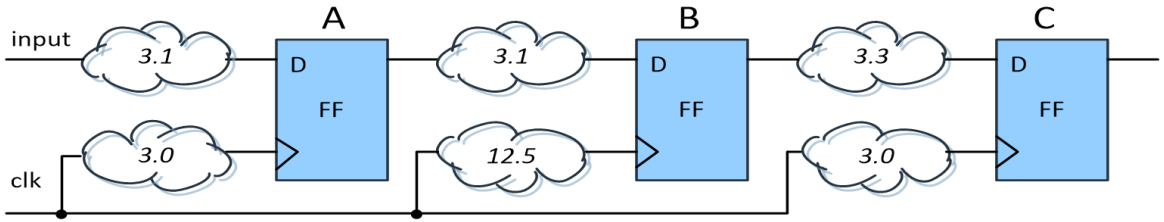
Why synchronous devices?

- Synchronous systems are more reliable
  - Events are analyzed only on clock edges that occur at precisely defined time intervals
  - The outputs from one level of logic have the entire clock period to propagate to the input of the next level
  - Offsets between the times of successive data arrival are allowed within the same clock period
- The problem of asynchronous devices
  - Delays with a specific value (e.g. 3.5ns) – more difficult to implement, may have a large spread (different system parameters)
  - Many signals may require specific timing (e.g. 4ns stable data before the selection signal appears, etc.)
  - Relying on propagation delays can result in possible glitches and spikes, because propagation delay varies with temperature and voltage fluctuations
  - Combinational logic is the main cause of glitches

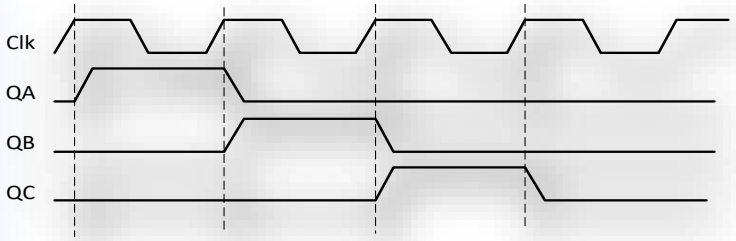
Good synchronous design practices can help you meet your design goals consistently.

Clock Skew

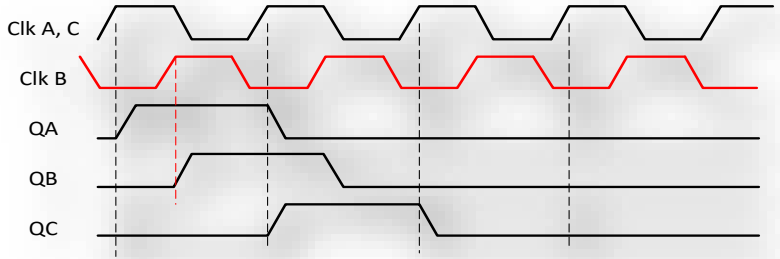
The shift register from the diagram does not work properly due to shifts in the clock path (various delays - spread in the times of adding the edge to the flip-flops)



correct waveforms

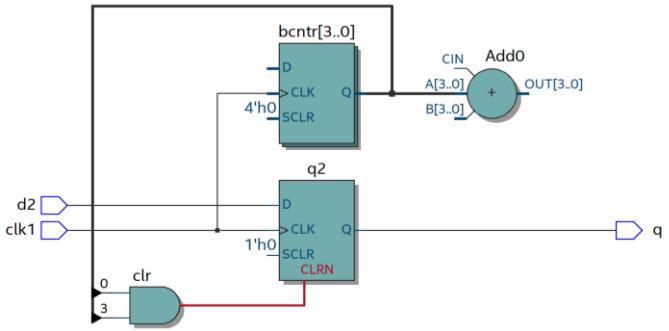


waveforms according to the times from the diagram

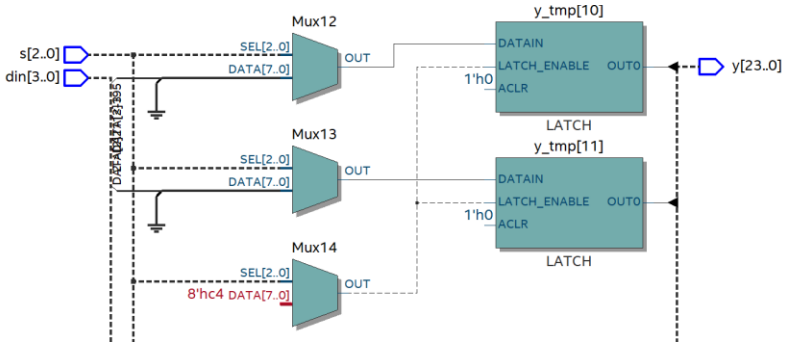


Design guidelines

- Avoid combinational loops whenever possible (feedback loops should include registers)
- A combinational loop also occurs when you feed back the output of a register to an asynchronous pin of the same register through combinational logic

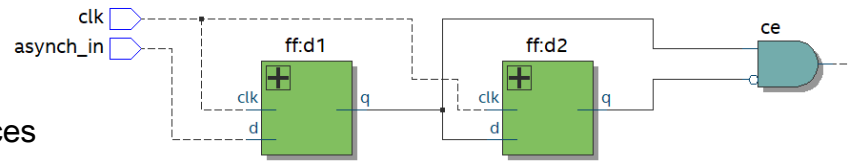


- Avoid using latches to ensure that you can completely analyze the timing performance and reliability of your design (a common mistake in HDL code is unintended latch inference)
- FPGA architecture is not optimized for latch implementation and latches generally have slower timing performance compared to equivalent registered circuitry



Design guidelines

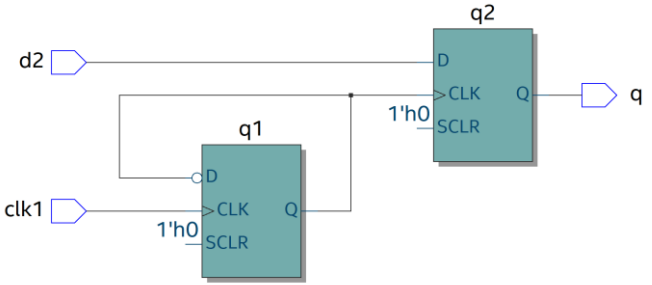
- Use synchronous techniques to design pulse
- The pulse width is always equal to the clock period
- This pulse generator is predictable, can be verified with timing analysis, and is easily moved to other devices



- Avoid using internally generated clocks (other than PLLs) wherever possible because they can cause functional and timing problems in the design
- Clocks generated with combinational logic can introduce glitches that create functional problems
- Use global device-wide, low-skew dedicated routing for all internally-generated clocks, instead of routing clocks on regular routing lines
- Always register the output of combinational logic before you use it as a clock signal

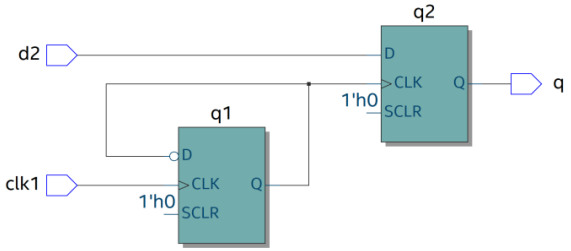
- Avoid ripple counters (cascaded registers can cause problems because the counter creates a ripple clock at each stage)

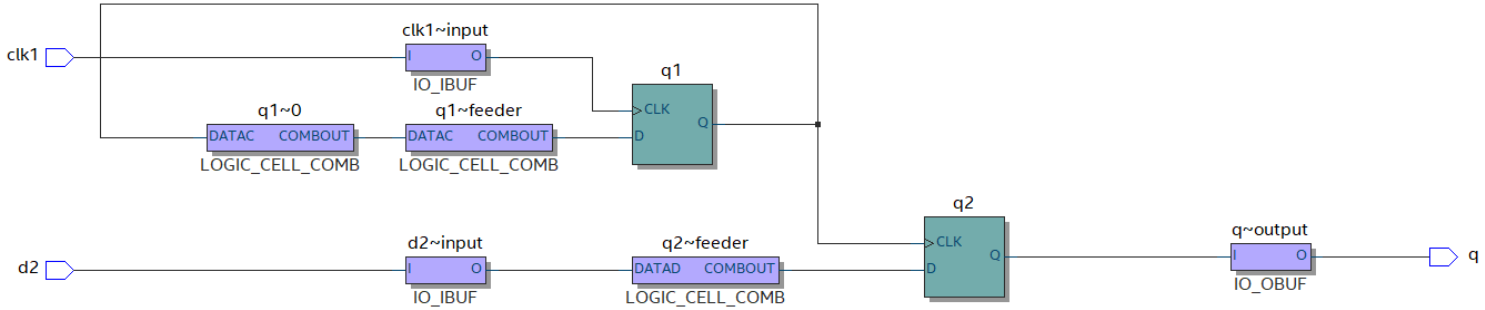
- Use dedicated hardware to perform clock gating rather than an AND or OR gate



Internally generated clocks

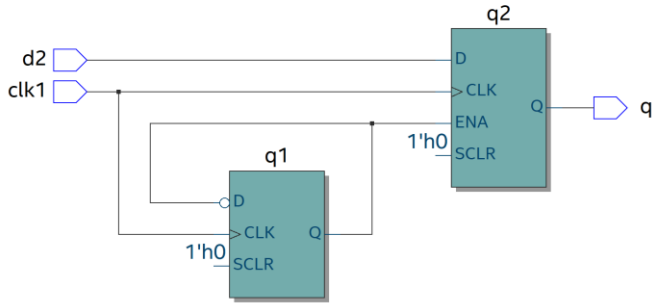
- The use of a locally generated signal in the clock path introduces an additional edge shift between clk1 and clk2 (q2 input)
- Not recommended

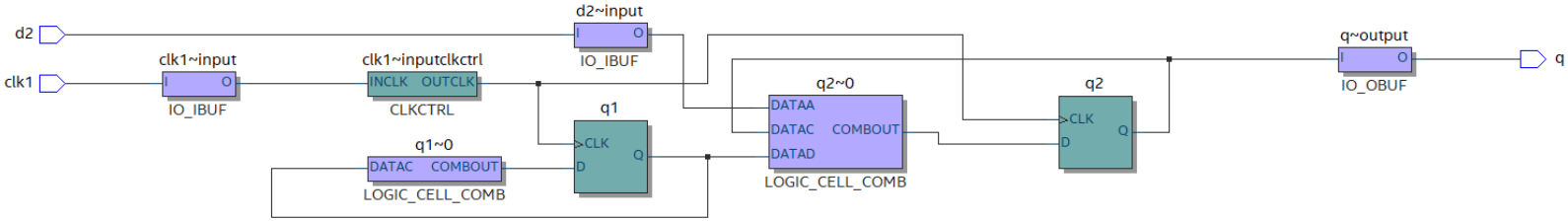




Internally generated clocks

- No clock shift between flip-flops
- Controlled by the same clock signal
- Using the CE input (clock enable)
- Global clock path routing

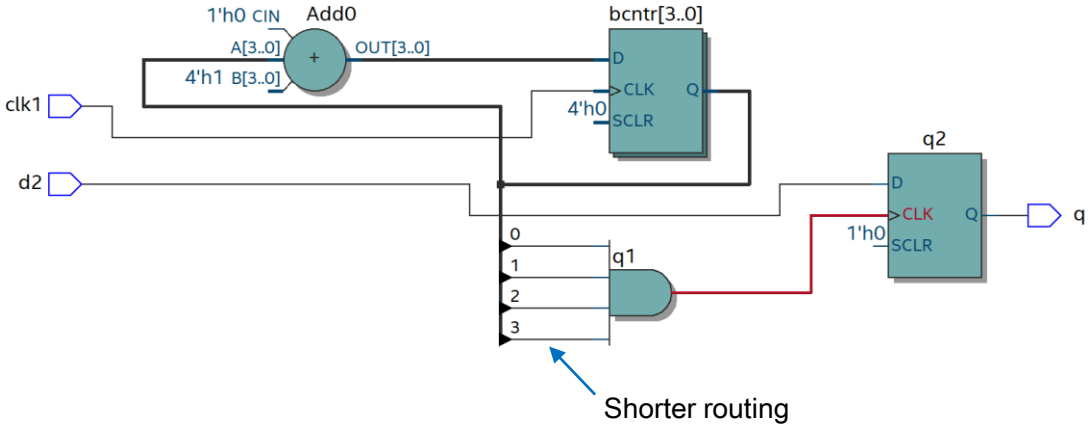




Unwanted pulses in the clock path

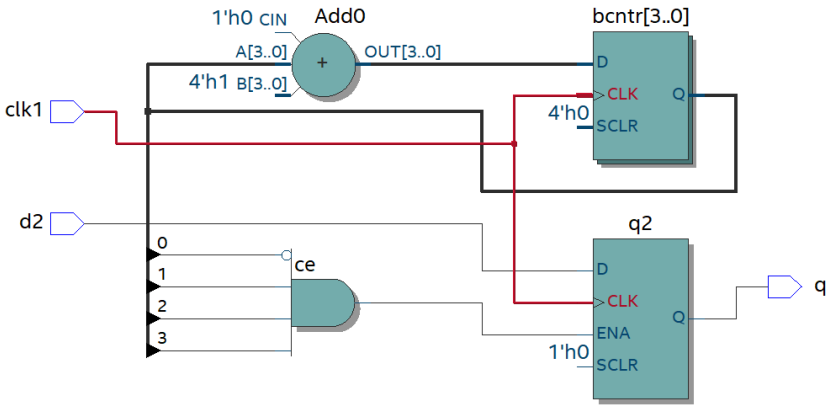
- Due to the manufacturing technology, modern FPGAs can respond even to very narrow pulses (glitches) appearing on the clock input
- !Never force the clock input with a combination signal (a technique known as "clock gating“)

0111 → 1111 → 1000 due to faster MSB



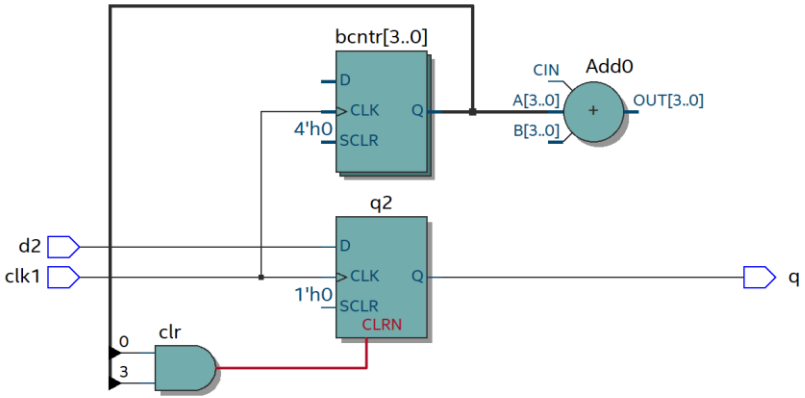
Unwanted pulses in the clock path

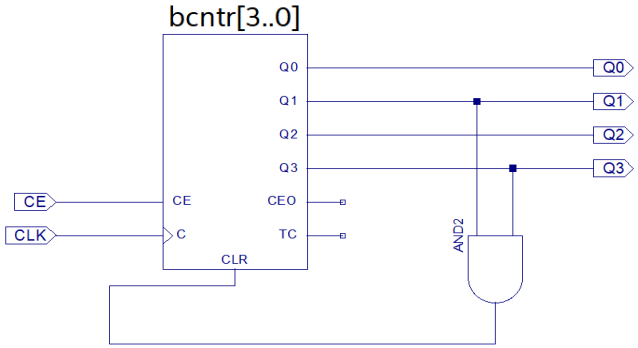
- Changing the control method increases the reliability of the system
- The function of the system is identical to that previously presented
- The difference is the use of the level-triggered **ce** (ENA) input and the use of a common clock signal **clk1**
- A solution that is insensitive to glitches appearing at the gate output
- Note the change in the function of the control gate (not LSB)

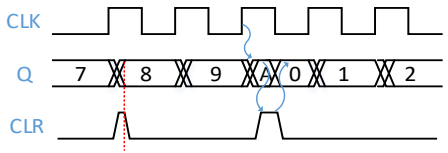


Glitches in the Set/Reset path

- Disturbances in the asynchronous set and reset signals can lead to incorrect operation of the device
- This phenomenon may be difficult to detect at the simulation stage (even post-route)
- In the diagram below, a problem may occur when switching between adjacent states

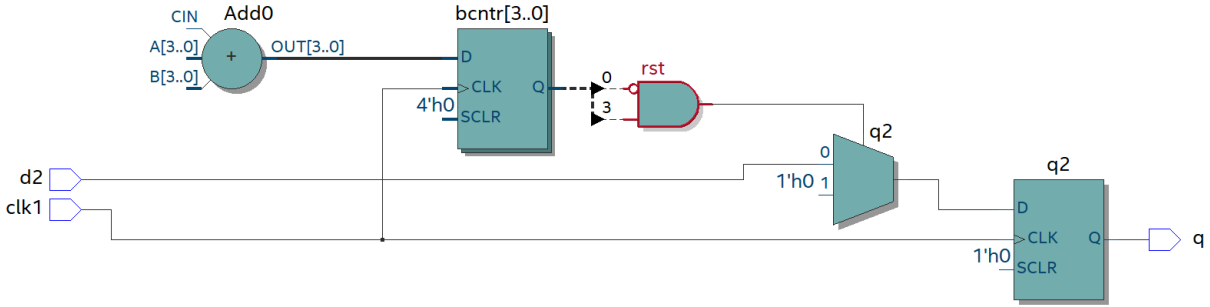


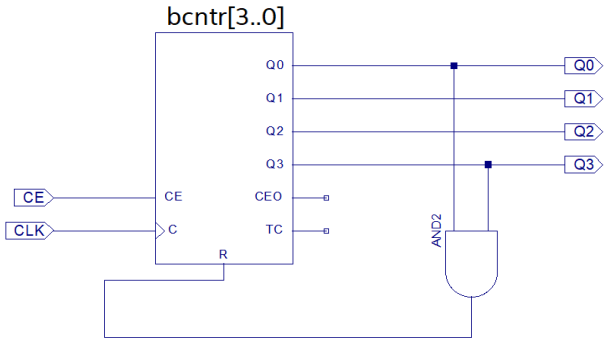


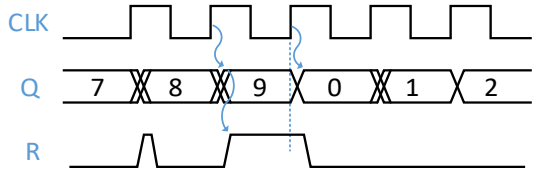


Glitches in the Set/Reset path

- The problem can be avoided by replacing asynchronous control with synchronous control
- This also involves changing the control function (negation to LSB)







Outline

- Good design practices
- Techniques for designing synchronous systems
- Construction of synchronization circuits

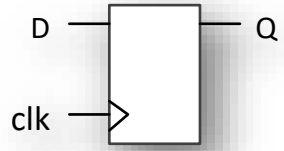
Synchronization circuits

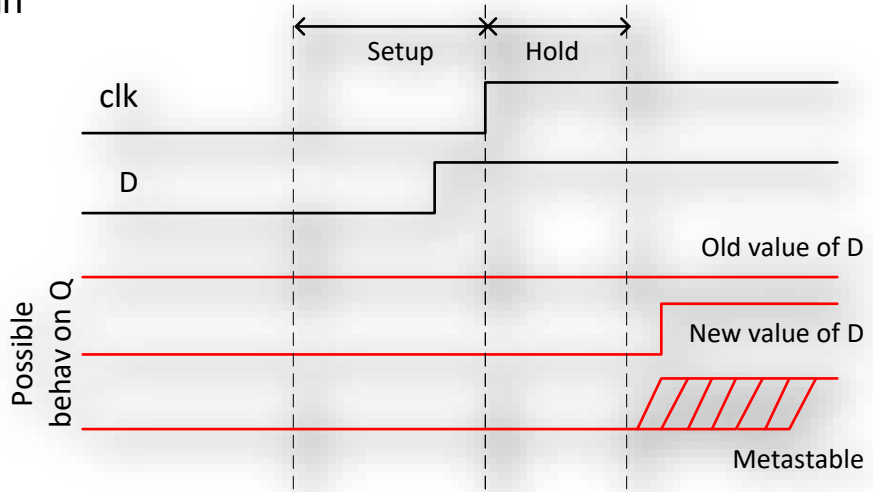
- How does the synchronization circuit work?
  - It takes an asynchronous input signal and outputs it when the clock edge arrives
- What is the synchronization circuit used for?
  - Prevents violations of the flip-flop set and hold time
  - Increases the reliability of device operation
- Where should a synchronization circuit be used?
  - When a signal passes between independent clock domains
  - When collecting asynchronous signals from outside the integrated circuit

Setup/hold time

- Before an active clock edge, you must ensure that the data input has been **stable for at least the setup time** of the register
- After an active clock edge, you must ensure that the data input remains **stable for at least the hold time** of the register

- When you **violate the setup or hold time** of a register, you might oscillate the output, or set the output to an intermediate voltage level between the high and low levels called a **metastable state**





Metastable state

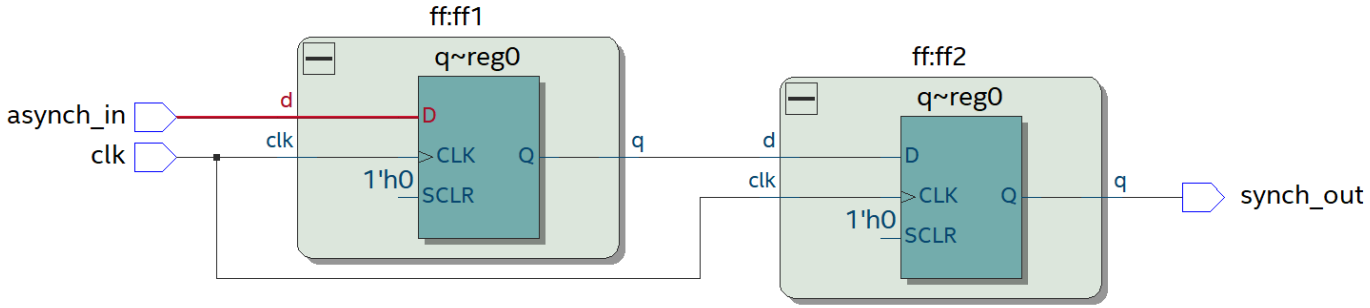
- The output of the flip-flop enters a transient state
  - It is neither a valid logical 0 nor a valid logical 1 (by some loads it may be interpreted as 0, by others as 1)
  - The output remains in this state for an unknown period of time before it reaches the correct logical state (0 or 1)
  - Small perturbations such as noise in power rails can cause the register to assume either the high or low voltage level, resulting in an unpredictable valid state

- Due to the statistical nature of the phenomenon, the occurrence of a metastable state may be reduced, not eliminated
- MTBF (Mean Time Between Failure) parameter is exponentially dependent on the length of time the flip-flop has to exit from a potential metastable state
  - A few extra **ns** of recovery time can significantly reduce the chances of a metastable event

The circuits shown next can achieve recovery times on the order of a full clock period

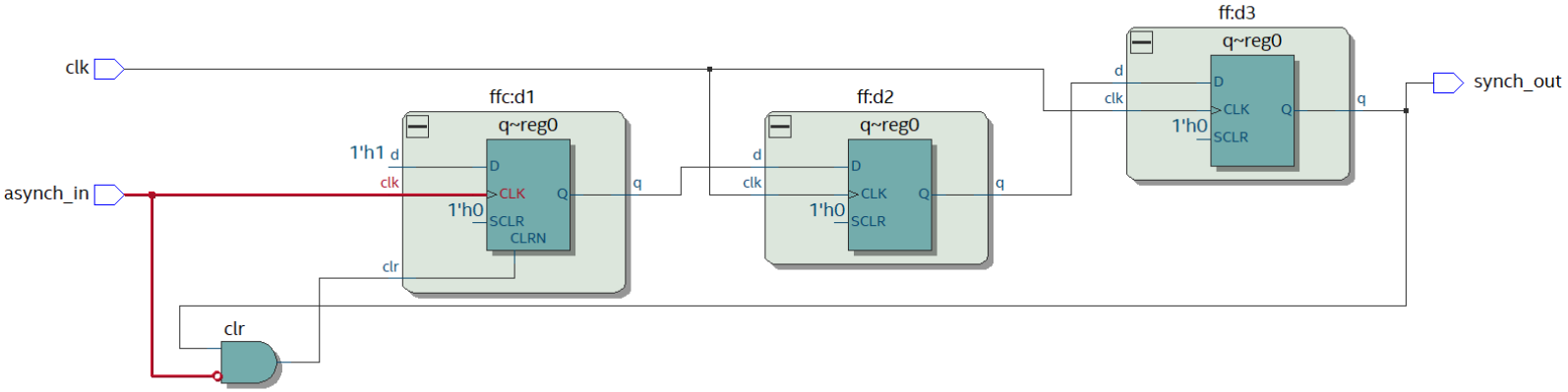
Synchronization circuit 1

- Circuit used when asynchronous input signals last at least one system clock period (clk)
- The additional FF2 flip-flop allows for full recovery time (guards against metastability)



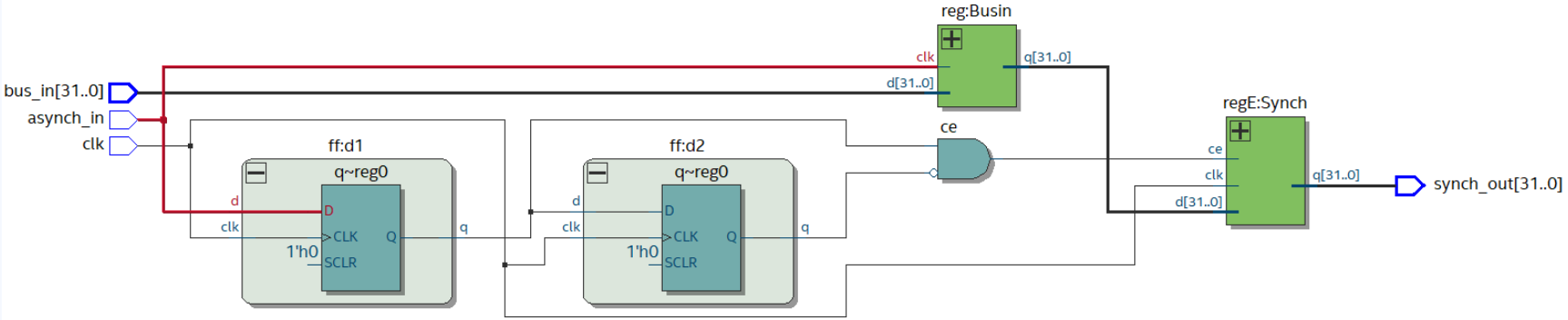
Synchronization circuit 2

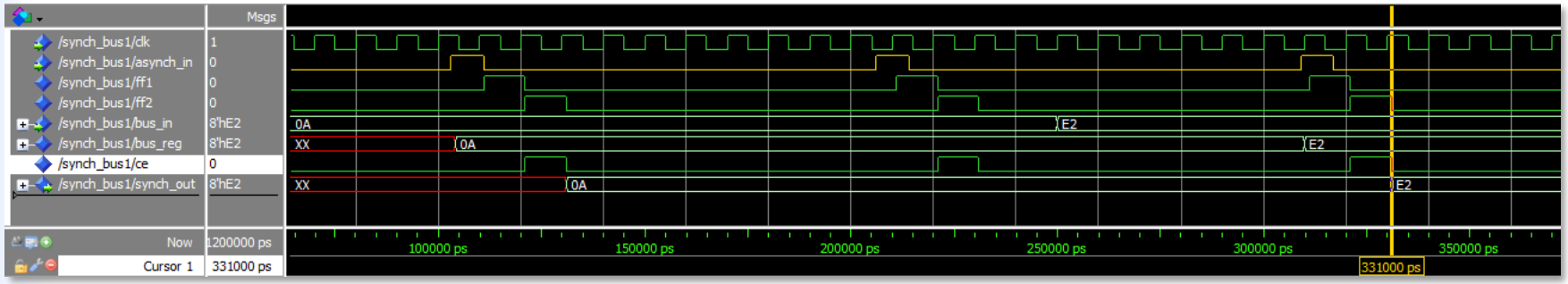
- Circuit used when input pulses may be shorter than one system clock period (clk)
- Register D1 (with asynchronous clear) is used to capture short pulses
- The AND2B1 gate protects against deleting the contents of D1 as long as the asynch_in pulse lasts
- The additional D3 flip-flop allows for full recovery time (guards against metastability)



Bus synchronization circuit 1

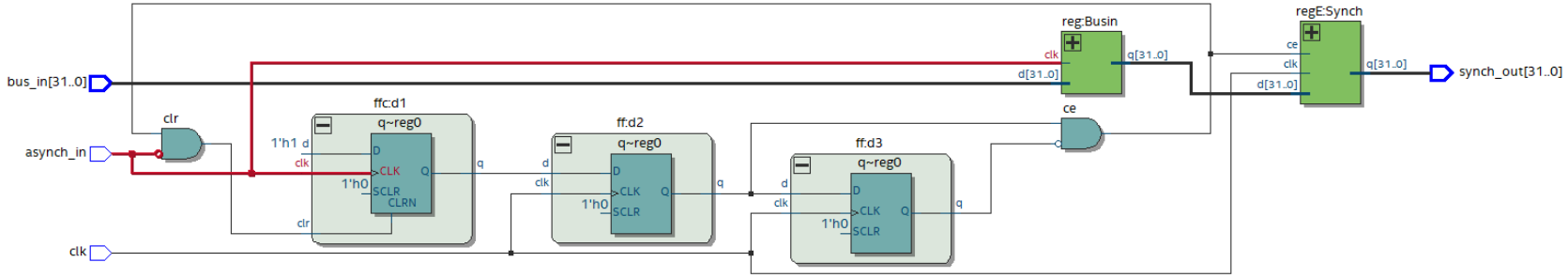
- Circuit used for trigger signals lasting **at least 1 CLK** clock cycle
- Registering the data bus (reg:Busin) with an asynchronous signal
- Rising edge detection system composed of ff:d2-and2B1 elements
- Bus data synchronized to the system clock in the regE:Synch register

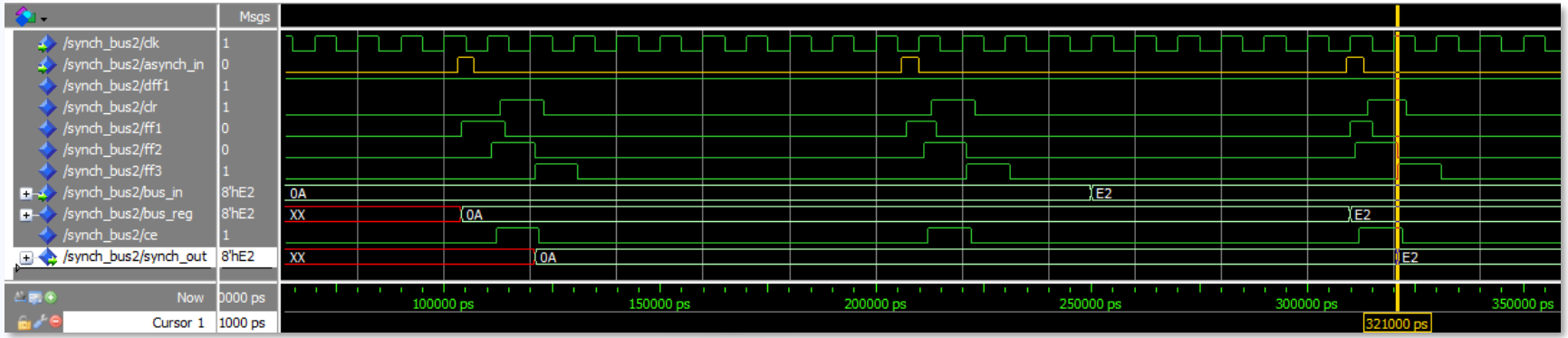




Bus synchronization circuit 2

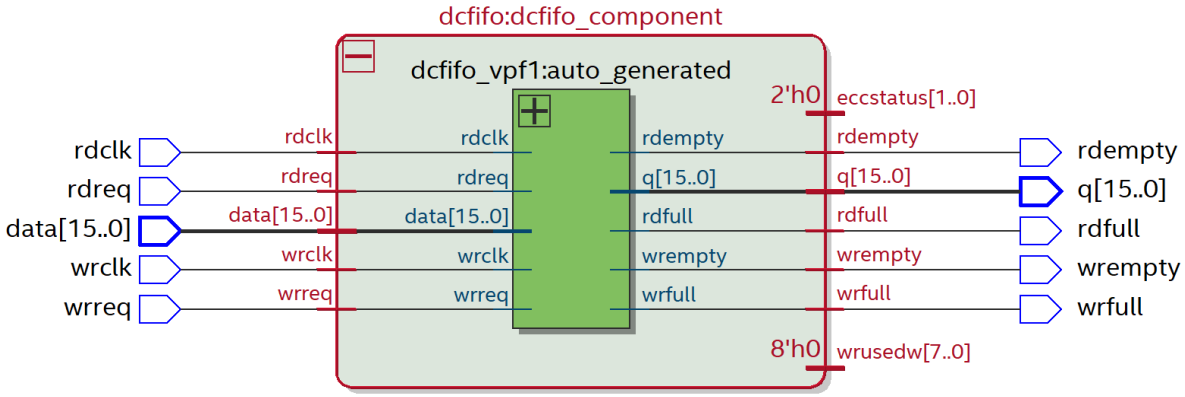
- The system is used when trigger signals may last **less than 1 CLK** clock cycle
- Registering the data bus (reg:Busin) with an asynchronous signal
- Rising edge detection system composed of ff:d3-and2B1 elements
- Bus data synchronized to the system clock in the regE:Synch register





Bus synchronization circuit 3

- A circuit used to synchronize data transfer between independent clock domains
- Use of FIFO memory with separated data writing and reading clocks



Summary

- Proper use of a hierarchical structure in your design facilitates debugging and module reuse (DFR)
- Synchronous projects are more reliable than asynchronous ones
- The use of global buffers and PLL circuits improves the timing and shape of the clock signals
- Avoid short bursts (glitches) in the clock path, asynchronous set and reset signals
- Use a clock enable as an alternative to gating a clock
- The use of synchronizing circuits increases the reliability of the device