Lecture 10

EE 421 / CS 425

Digital System Design

Fall 2025

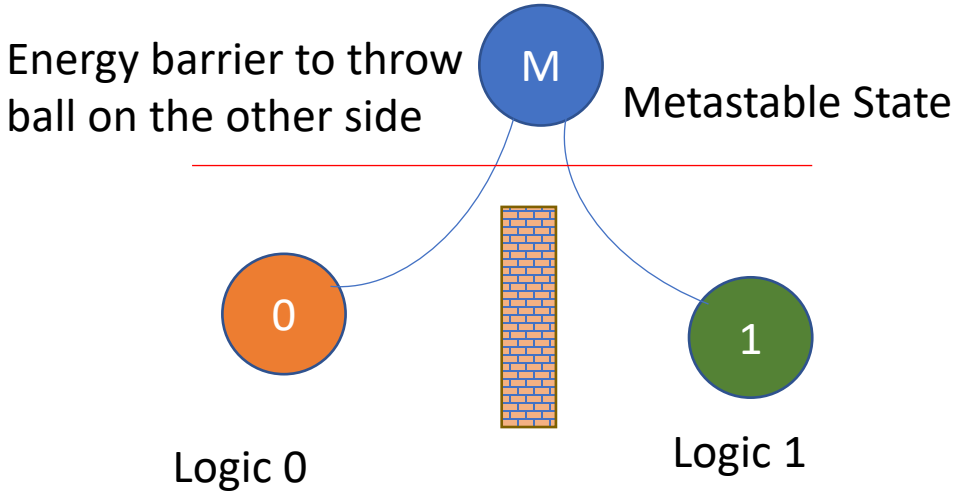
Shahid Masud

Topics

- Metastability
- Synchronizers for capturing Asynchronous Inputs
- Signal Transfer with and without Clock
- Communication between modules in Complex Digital System
- **QUIZ 2 TODAY**

Metastability, and Asynchronous Data Management

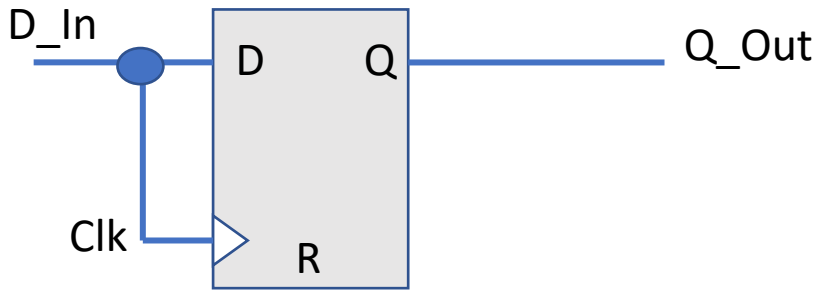
Objective – to reduce possible Metastability in capturing Asynchronous Input



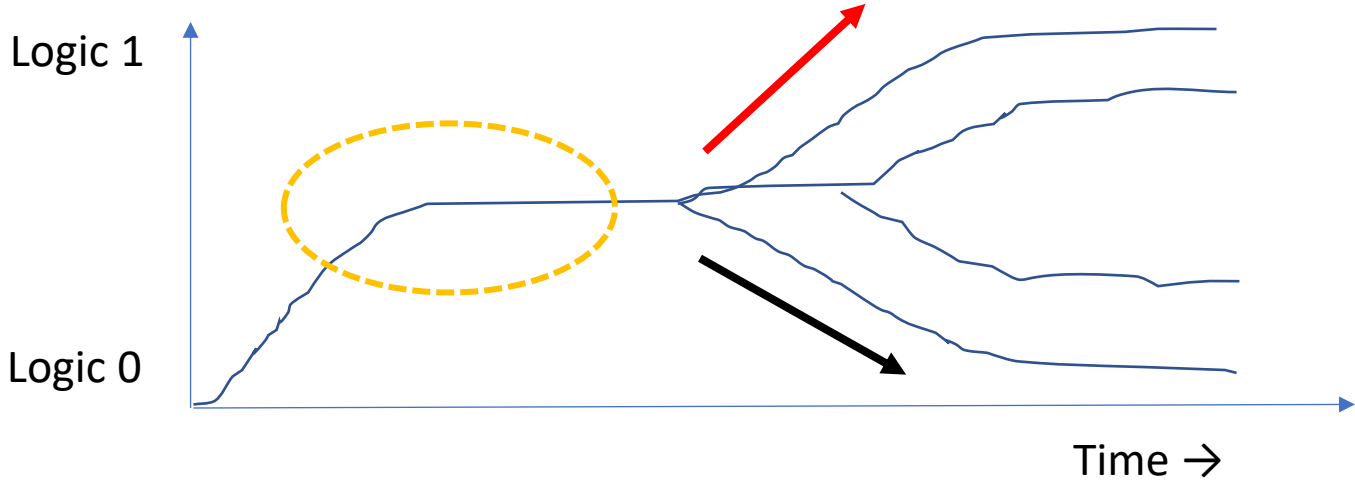
Synchronizer Failure: When flipflop hangs in a Metastable State for a long time (indefinitely)

Normally, the flipflop output would settle to a stable 0 or 1 state after some time

Output Behaviour with Metastability



DFF is connected to produce metastability
As setup time is violated



Oscilloscope trace of metastable behaviour

Eventually a stable state is reached

Problem occurs when flipflop is not stable within One clock period

Synchronizers

- Asynchronous inputs are problematic as their transitions are not predictable
- High speed digital circuits rely on synchronizers to create a time buffer for recovering from a metastable event; thus reducing the possibility that metastability will cause circuit to malfunction
- An asynchronous signal should be synchronized by one synchronizer only. If not, then multiple synchronized signals could be present in the system and one of these could be driven into metastable state

Asynchronous Inputs to a Synchronous Digital System

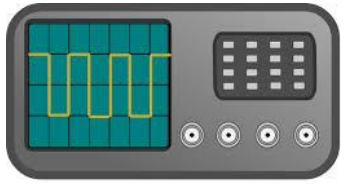
Asynchronous Domain

Asynch_In

Synchronizer (Type I or II)

Synch_Out

Ref Clk

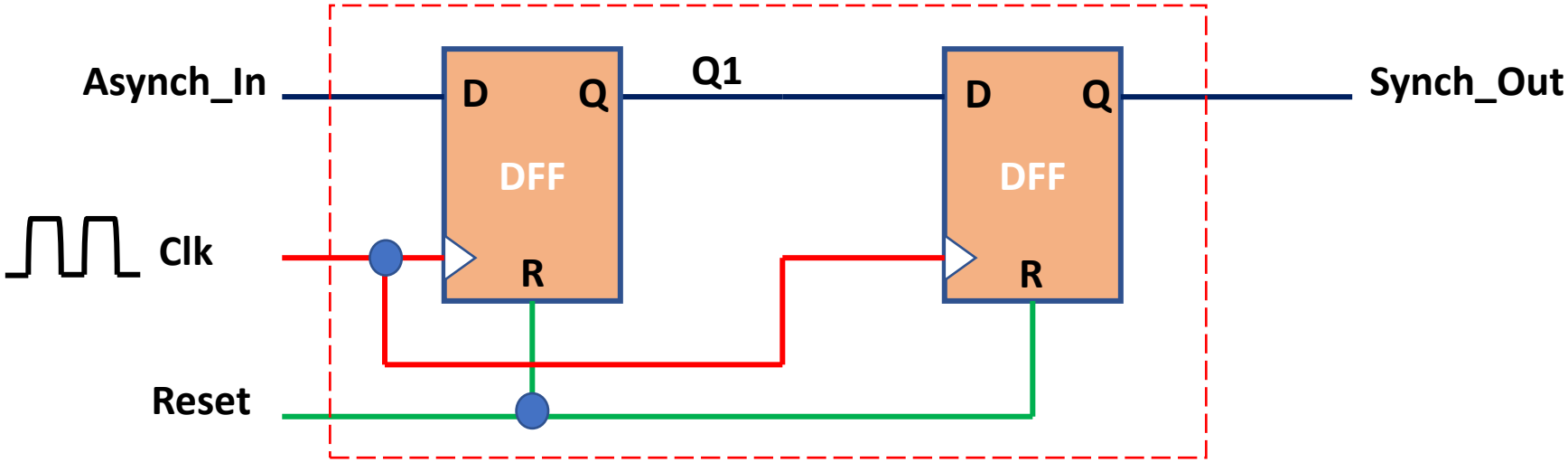


Synchronous Domain

Synchronizer 1

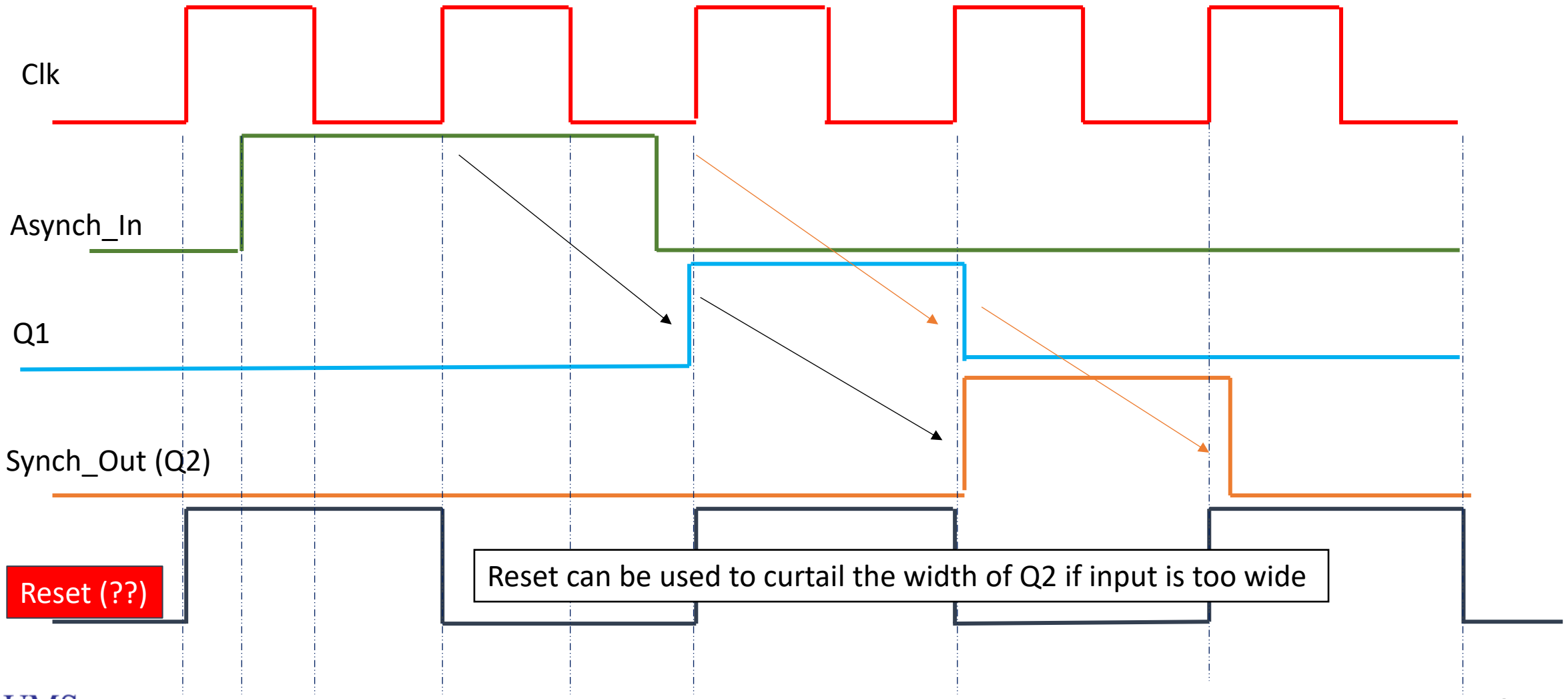
Condition: The width of asynchronous input pulse is greater than period of the clock

Synchronizer is a multistage shift register



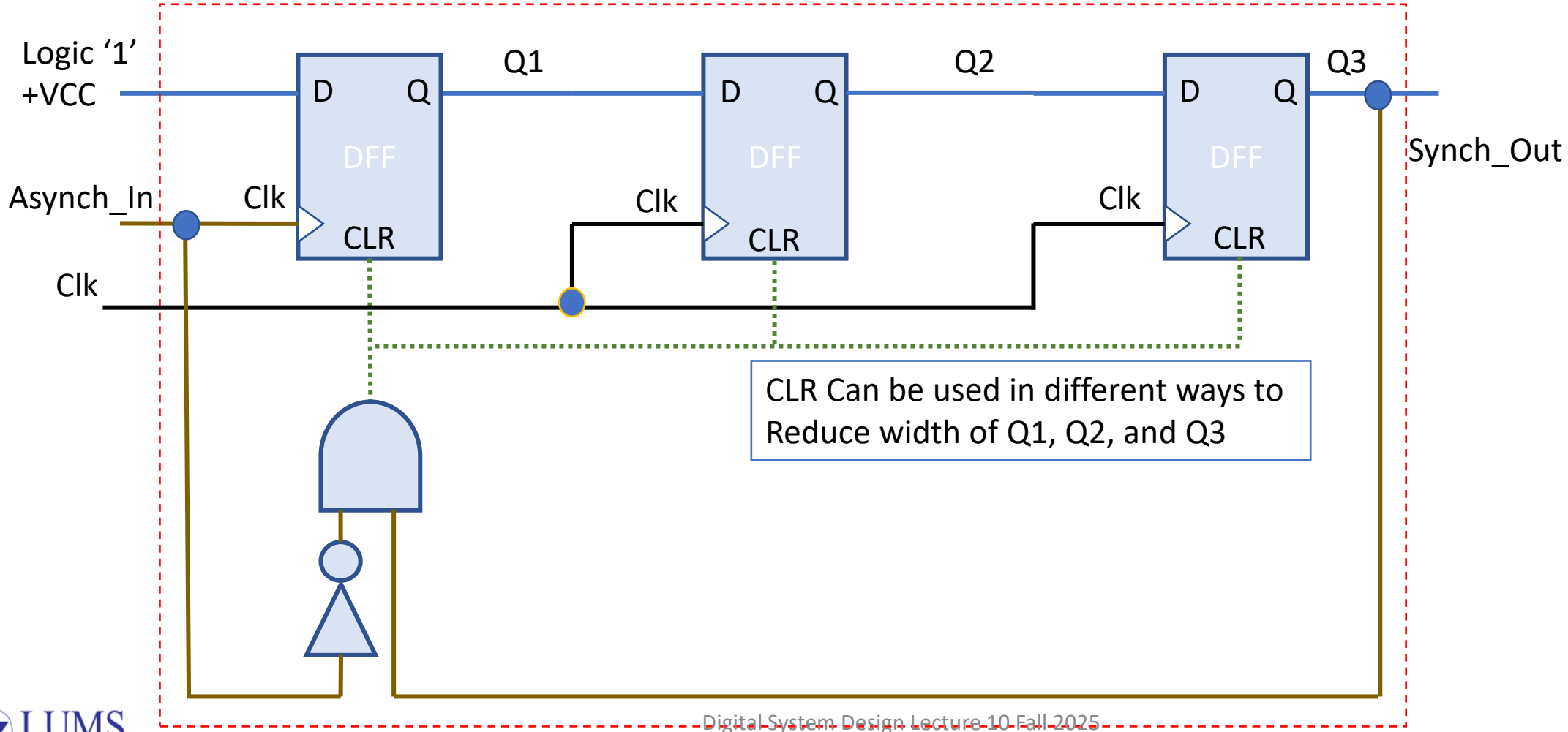
Reset Input 'R' is used as control to bring Synch_Out back to '0', as required

Timing Diagram – Synchronizer 1

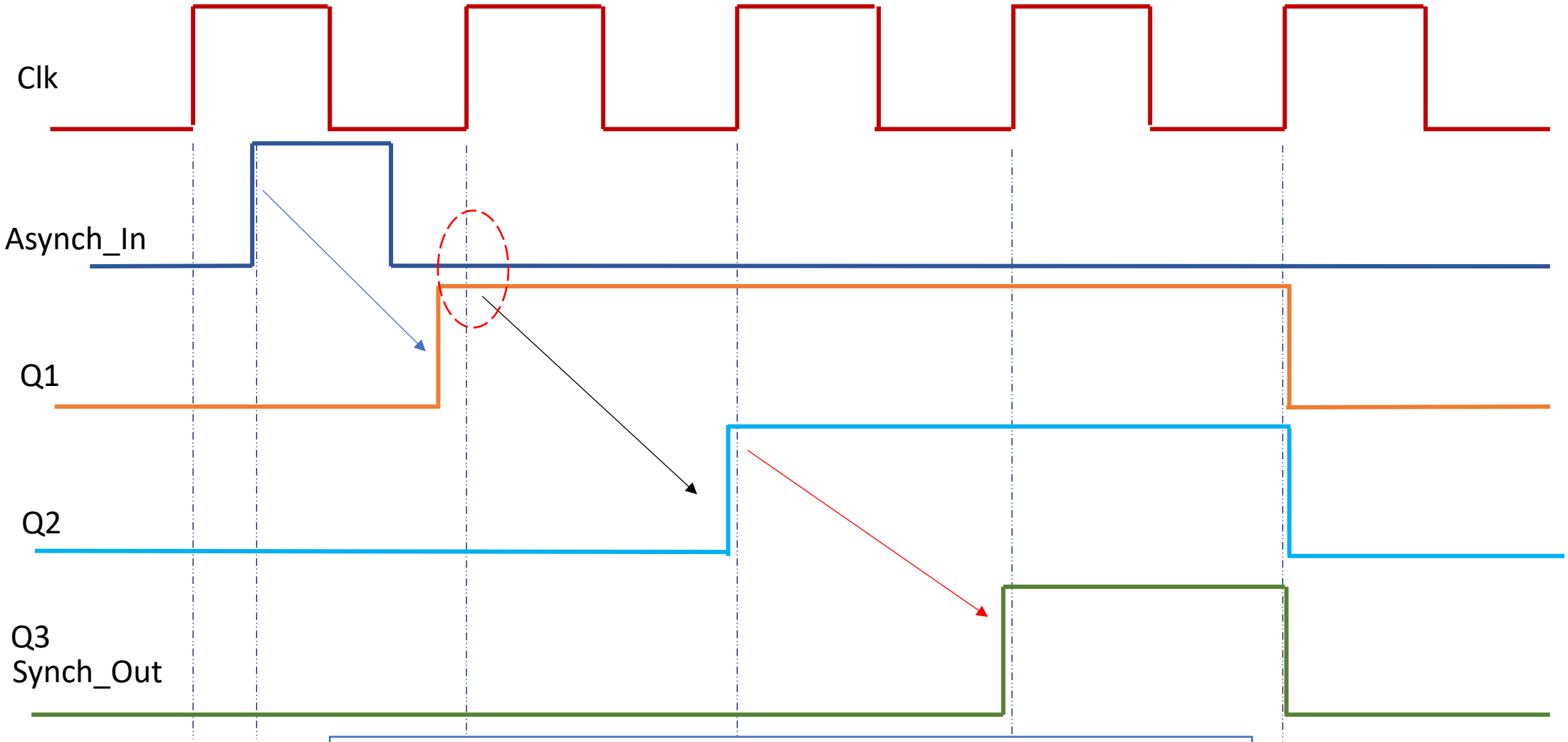


Synchronizer 2

Condition: Width of the asynchronous input pulse is less than the period of the clock



Timing Diagram – Synchronizer 2

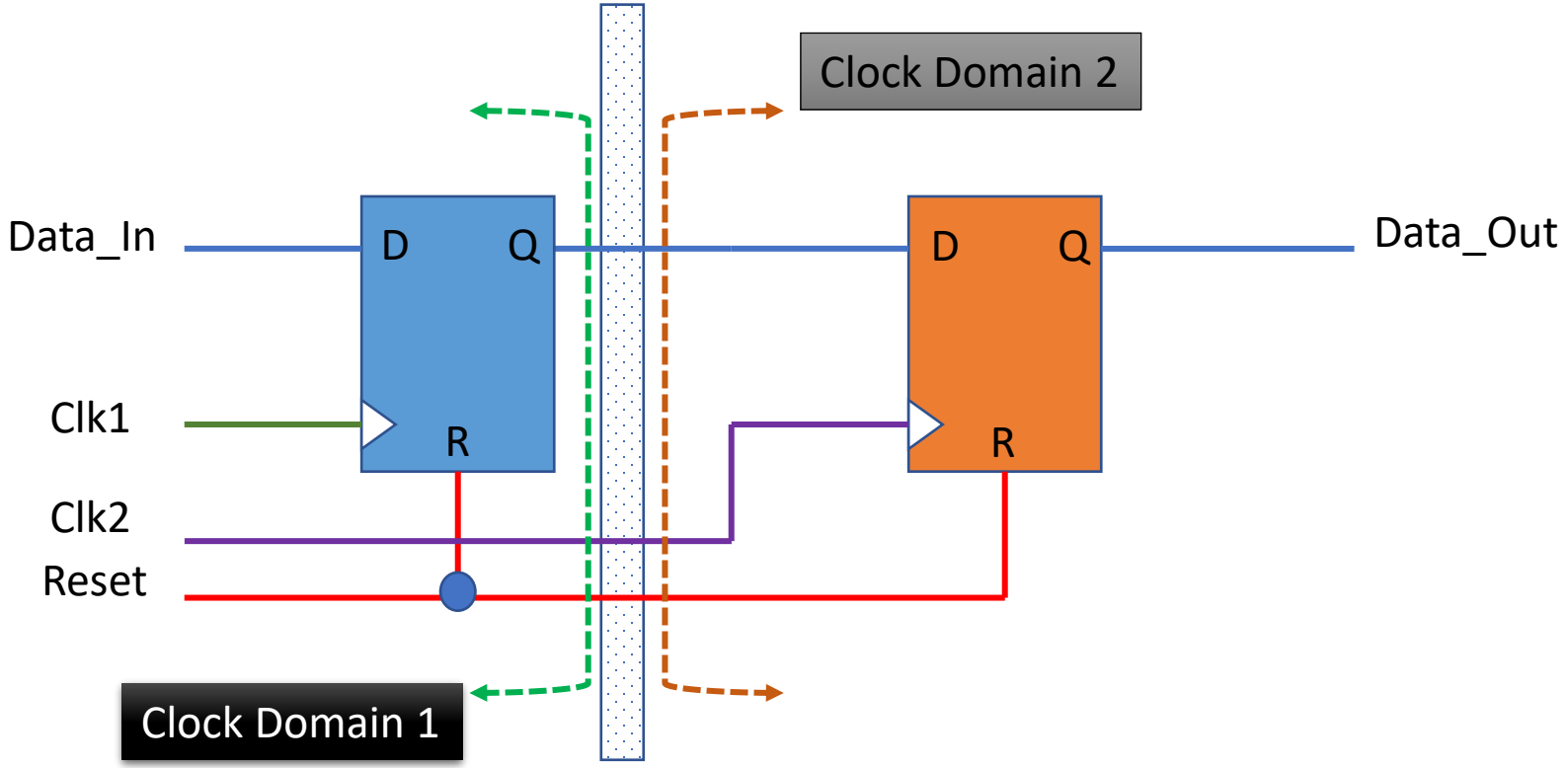


CLR input at DFF1 is used to bring back Q1 and then Q2 to Zero

Self-Timed and Speed Independent Circuits

- Having a completed Synchronous system is at times too challenging for a complex and fast digital system
- The limiting problem becomes how to distribute a single global clock without introducing intolerable clock skew
- The alternate is to partition the digital system into locally clocked pieces that communicate with each other using delay-insensitive signaling techniques (i.e. local clock for local communication)
- Each block proceeds at its own speed without the need for a global clock, synchronizing local communication whenever needed
- Usually a Request-Response Signalling method is employed

Data Reading across two Clock Domains



frequency of Clk1 is less than Clk2
Otherwise, the Synchronizer-2 is versatile and can be used here

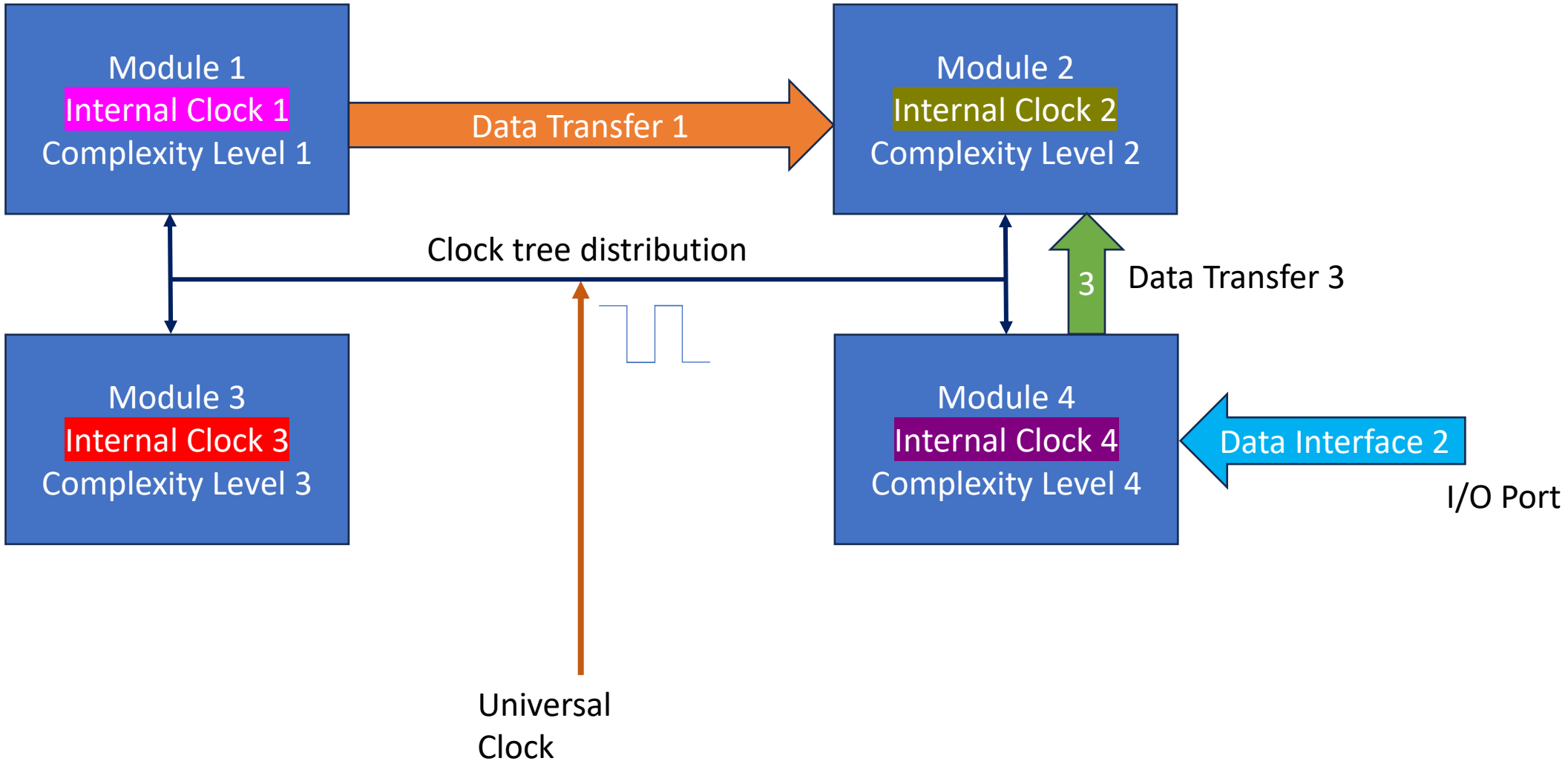
Communication across modules in Complex SoC

- Communication Signaling across modules at different clock speeds
- Self-timed circuits

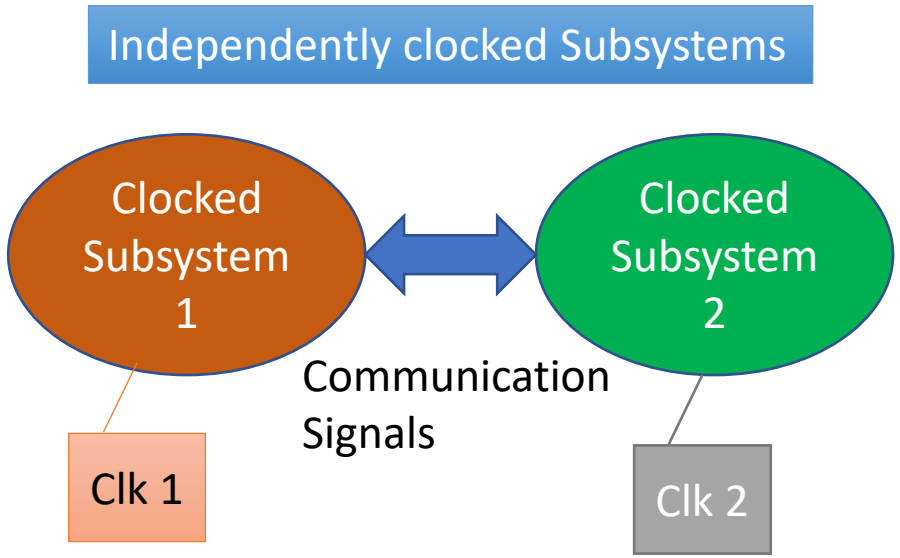
Self-Timed and Speed Independent Circuits

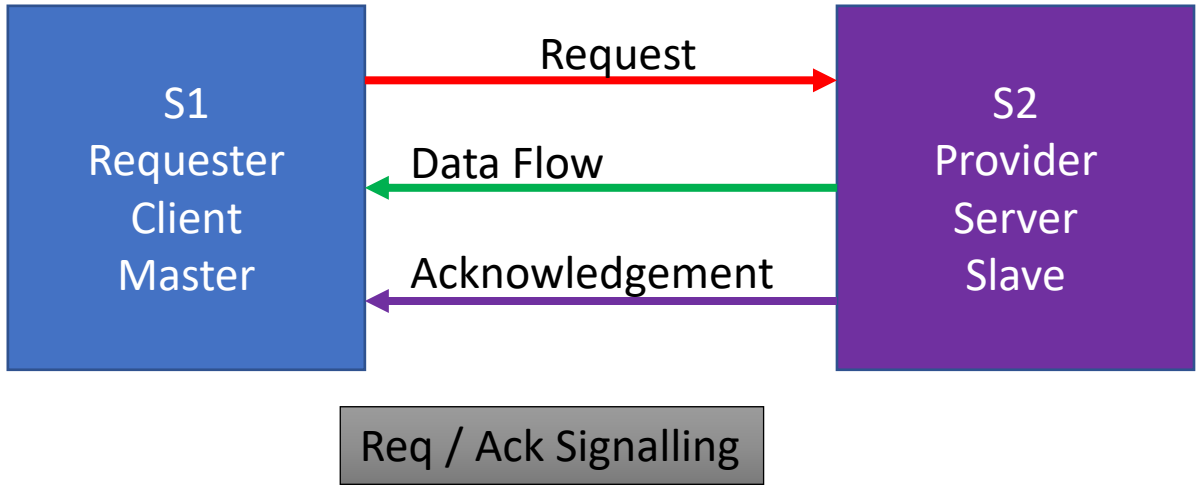
- Having a completed Synchronous system is at times too challenging for a complex and fast digital system
- The limiting problem becomes how to distribute a single global clock without introducing intolerable clock skew
- The alternate is to partition the digital system into locally clocked pieces that communicate with each other using delay-insensitive signaling techniques (i.e. local clock for local communication)
- Each block proceeds at its own speed without the need for a global clock, synchronizing local communication whenever needed
- Usually a Request-Response Signaling method is employed

Data Transfer in Complex Modular Design



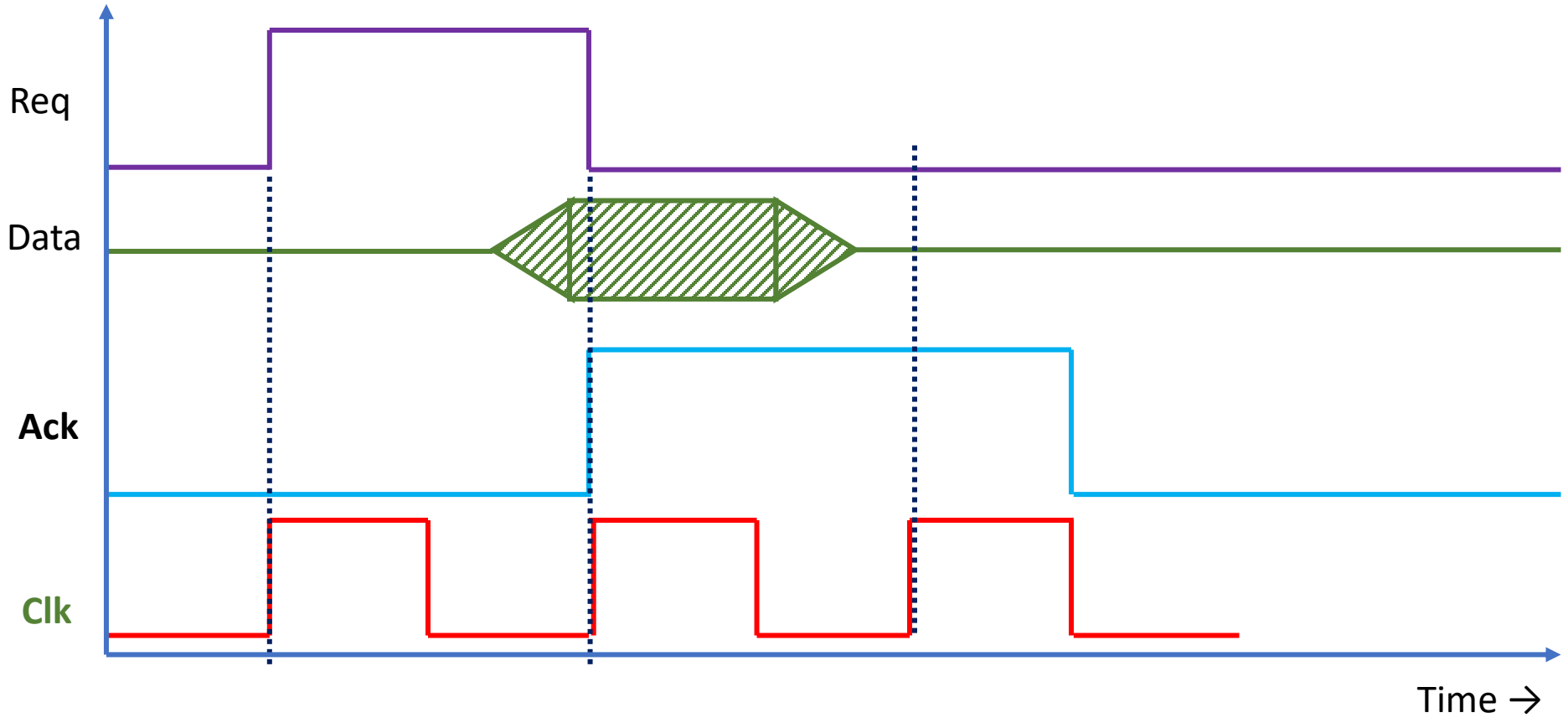
Request / Acknowledge Signaling





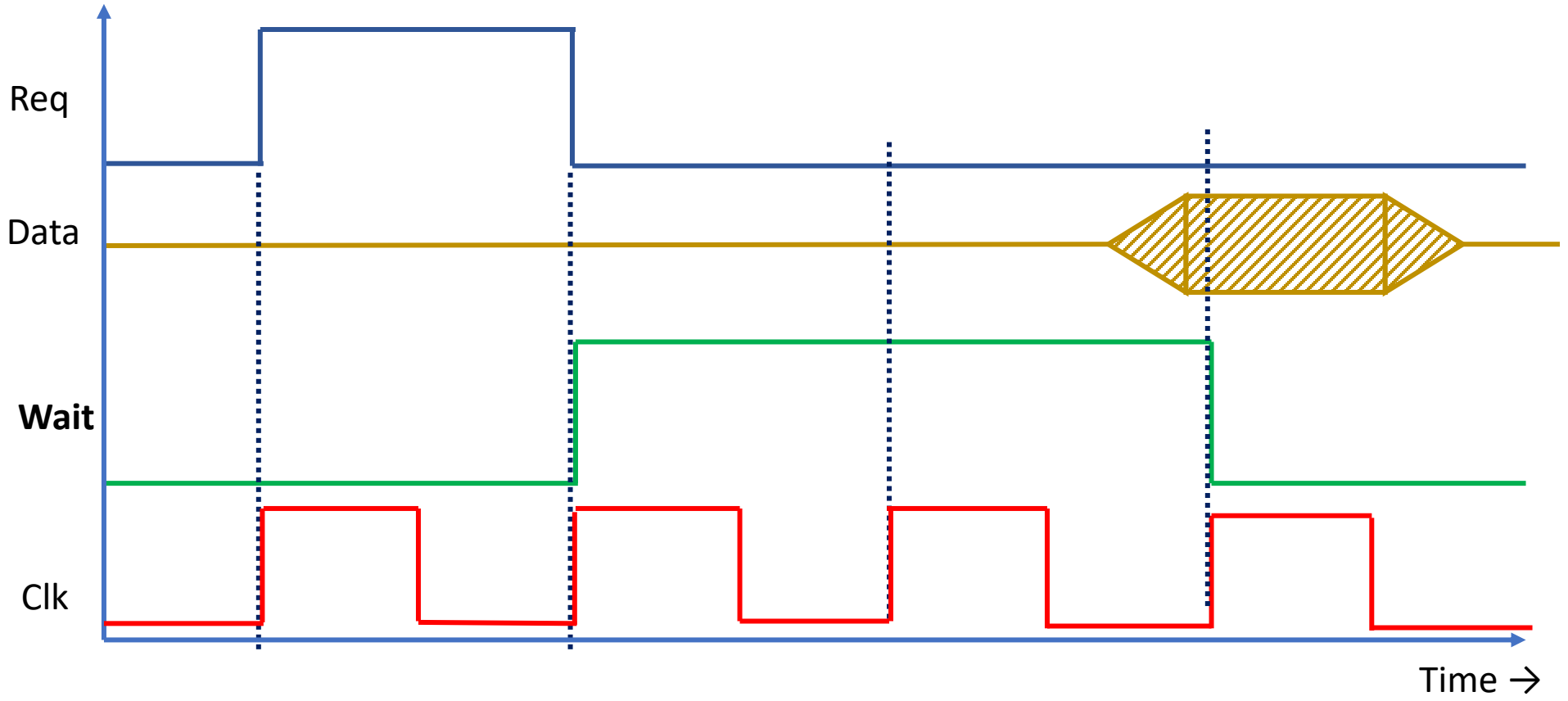
Req / Ack Signalling

Synchronous Req / Ack Signaling



Req and Ack signals are synchronized to **Clk**

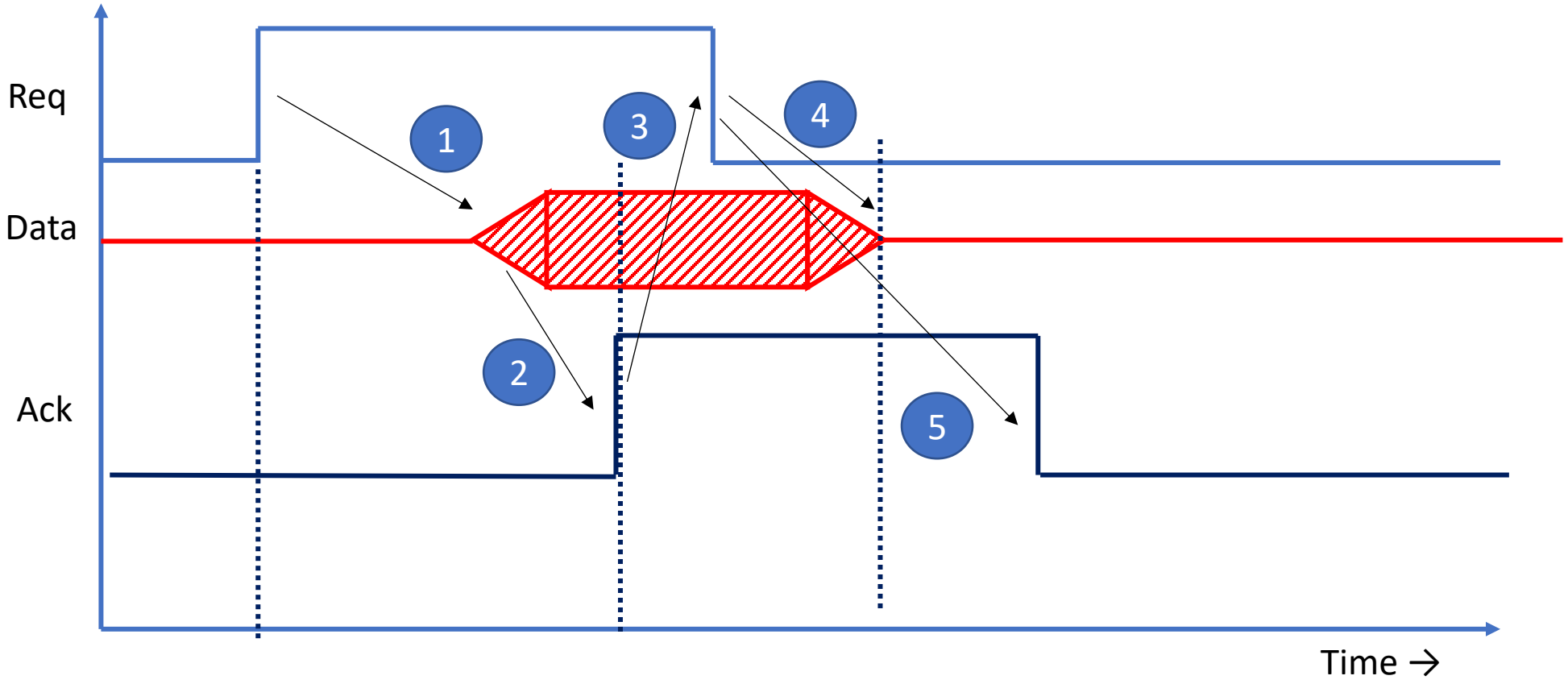
Synchronous Req with Wait Signaling



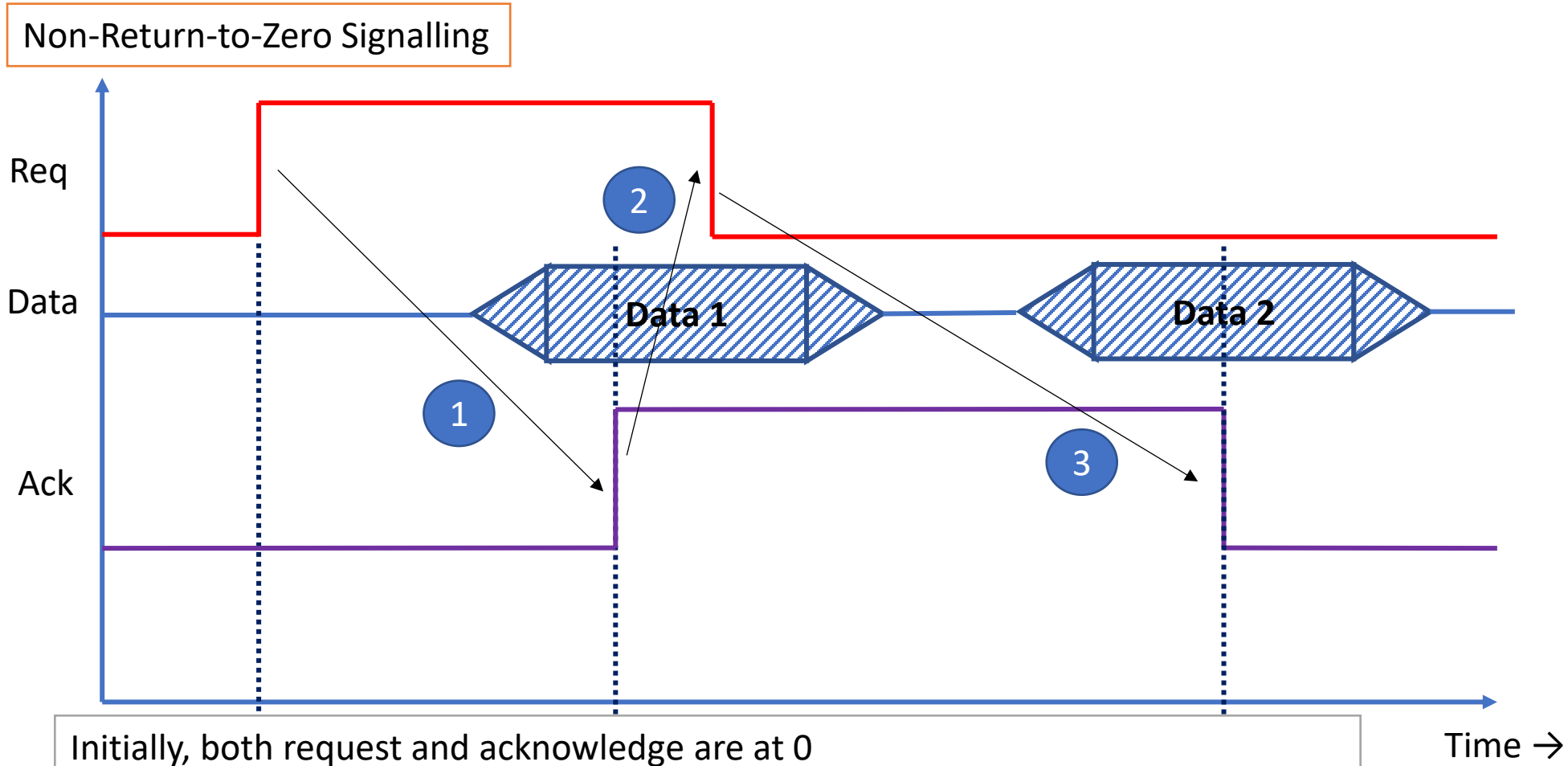
Slave can delay the Master by asserting Wait signal as it prepares the data and needs more clock cycles
When the slave un-asserts Wait signal, it acknowledges that data is now available for the Master to read
All interface signals are synchronized with Clock edge

Four Cycle Asynchronous Signaling

RTZ – Return to Zero Signaling with No Clock



Two Cycle Asynchronous Signaling



Initially, both request and acknowledge are at 0
Request line is complemented (to 1)
Slave notices change in Req, provides data, and complements the Ack line (to 1)
Further request and acknowledgement is by complementing the respective state

Self-Timed Circuits

- A self-timed circuit can determine on its own when a request has been serviced by mimicking the worst-case propagation delay path by using special logic to delay the request signal.
- This guarantees sufficient time to compute the correct output.