Lecture 10
EE 421 / CS 425
Digital System Design

Fall 2024

Shahid Masud

Topics

- State Reduction by Implication Charts – to complete
- ----------
- Metastability
- Synchronizers for capturing Asynchronous Inputs
- Signal Transfer with and without Clock
- Communication between modules in Complex Digital System

State Reduction using Implication Charts

- Graphical Technique for State Reduction
- Definition:
- Redundant State:
  - One state is equivalent to another (and hence redundant) if the state functions are in-distinguishable

- When all states that have (i) different next states and (ii) different outputs, have been identified, the remaining states are considered to be redundant

State Equivalence Theorem

- Two states SA and SB are equivalent iff for every possible input X sequence, the outputs are same and the next states are equivalent

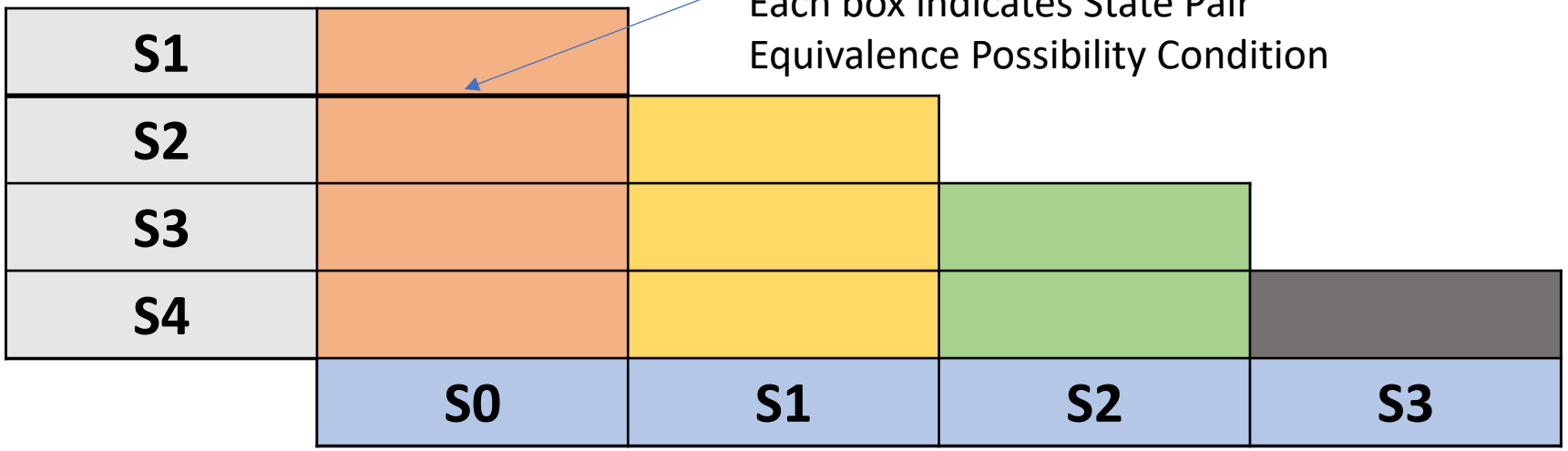
- Properties of Equivalent States:
- Symmetry: If SA=SB, then SB=SA
- Reflexivity: SA = SA for any state
- Transitivity: If SA=SB, and SB=SC, then SA=SC

Construction of Implication Charts

The Chart is constructed by listing all of the states except the last, along horizontal axis, and
Listing all of the states except the first, along the vertical axix
Intersection of the horizontal and vertical spaces indicates a possible state equivalene

Eg. Total States **S0 to S4**

Implies the possibility that S0 = S1
Each box indicates State Pair
Equivalence Possibility Condition



Example of State Reduction using Implication Charts

Given the State Table as follows:

Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S3	0	1
S1	S5	S3	0	0
S2	S4	S1	0	1
S3	S5	S1	0	0
S4	S2	S5	0	1
S5	S1	S2	0	0

Total 6 states, S0 to S5

Filling the Implication Chart

- Place X in squares where outputs are different, as such states cannot be equivalent
- For other squares, we look at State Equivalence Pairs, i.e.:
  - S0 ≡ S2, iff S1 ≡ S3

  Thus we write S1,S3 in the square at intersection of S0, S2

  S0 ≡ S4, iff S2 ≡ S3, and S3 ≡ S5

  S1 ≡ S5, iff S2 ≡ S3

  S2 ≡ S4, iff S1 ≡ S5

  S3 ≡ S5, iff S1 ≡ S5, and S1 ≡ S2

Filled Implication Chart with conditions of equivalent states

S1	X				
S2	S1,S3	X			
S3	X	S1,S3	X		
S4	S2,S4 S3,S5	X	S2,S4 S1,S5	X	
S5	X	S1,S5 S2,S3	X	S1,S5 S1,S2	X
	S0	S1	S2	S3	S4

Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S3	0	1
S1	S5	S3	0	0
S2	S4	S1	0	1
S3	S5	S1	0	0
S4	S2	S5	0	1
S5	S1	S2	0	0

One by one, examine all table entries and refer to the state transition table

X is inserted where outputs of states is different
Check conditions of equivalence in respective squares

Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S3	0	1
S1	S5	S3	0	0
S2	S4	S1	0	1
S3	S5	S1	0	0
S4	S2	S5	0	1
S5	S1	S2	0	0

S1	X				
S2	S1,S3	X			
S3	X	S1,S3	X		
S4	S2,S4 S3,S5	X	S2,S4 S1,S5	X	
S5	X	S1,S5 S2,S3	X	S1,S5 S1,S2	X
	S0	S1	S2	S3	S4

To check for: {S1,S3}, {S1,S5}, {S2,S3}, {S1,S2}

To check for: {S1,S3}, {S1,S5}, {S2,S3}, {S1,S2}

Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S3	0	1
S1	S5	S3	0	0
S2	S4	S1	0	1
S3	S5	S1	0	0
S4	S2	S5	0	1
S5	S1	S2	0	0

In {S1,S3}:
Outputs are same for S1, and S3
In Next States, When X=1, S1 goes to S3 and S3 goes to S1
Hence S1 and S3 are equivalent

In {S1,S2}, output is different when X=1, hence these cannot Be equivalent, so this square will get 'X'

In {S2,S3}, output is different when X=1, hence these cannot be equivalent, so this square will be 'X'.
This means that S1 cannot be equivalent to S5 as it is dependent on S2 and S3 as Next State

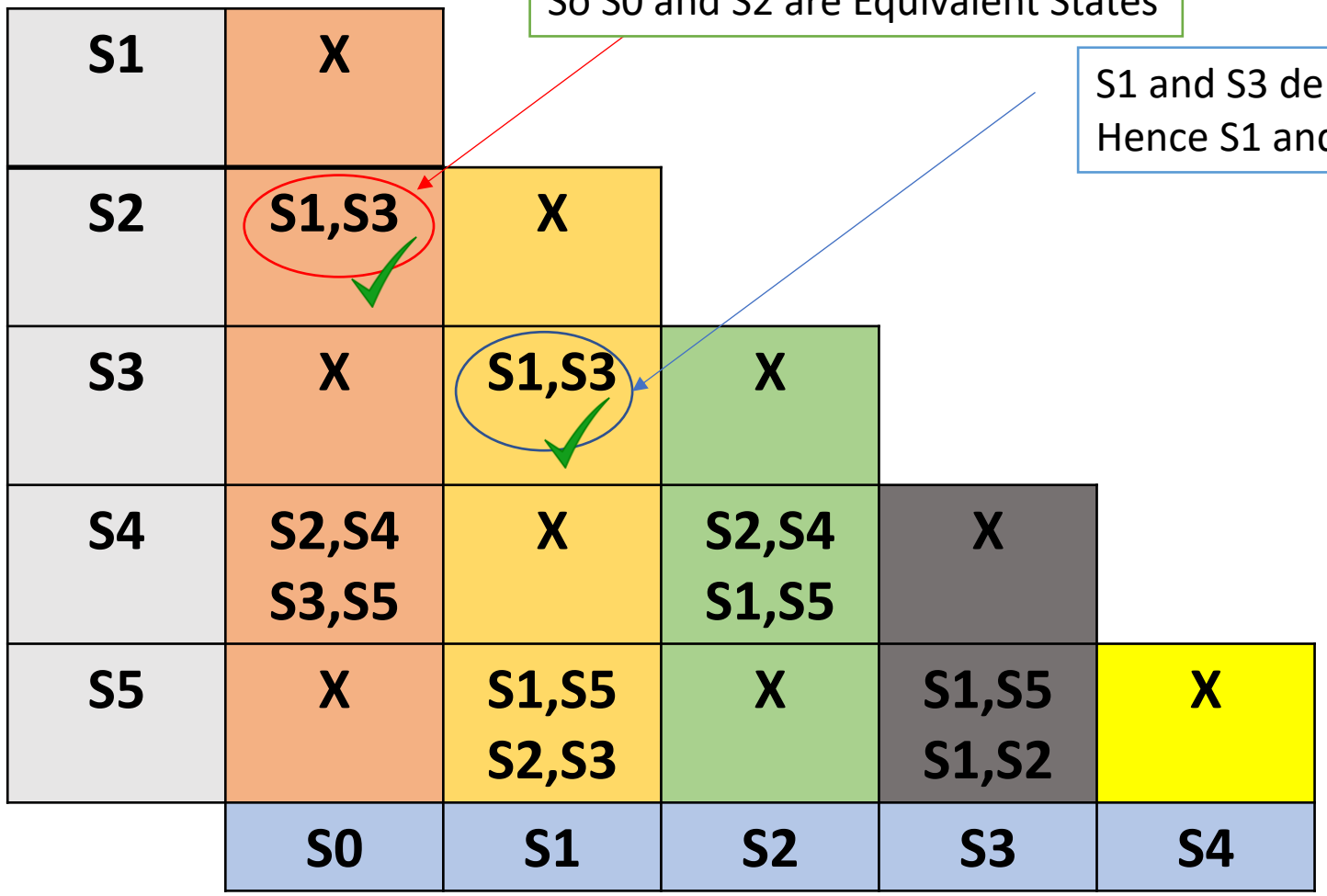
Similarly, check all conditions in all squares

Updated Implication Chart

Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S3	0	1
S1	S5	S3	0	0
S2	S4	S1	0	1
S3	S5	S1	0	0
S4	S2	S5	0	1
S5	S1	S2	0	0

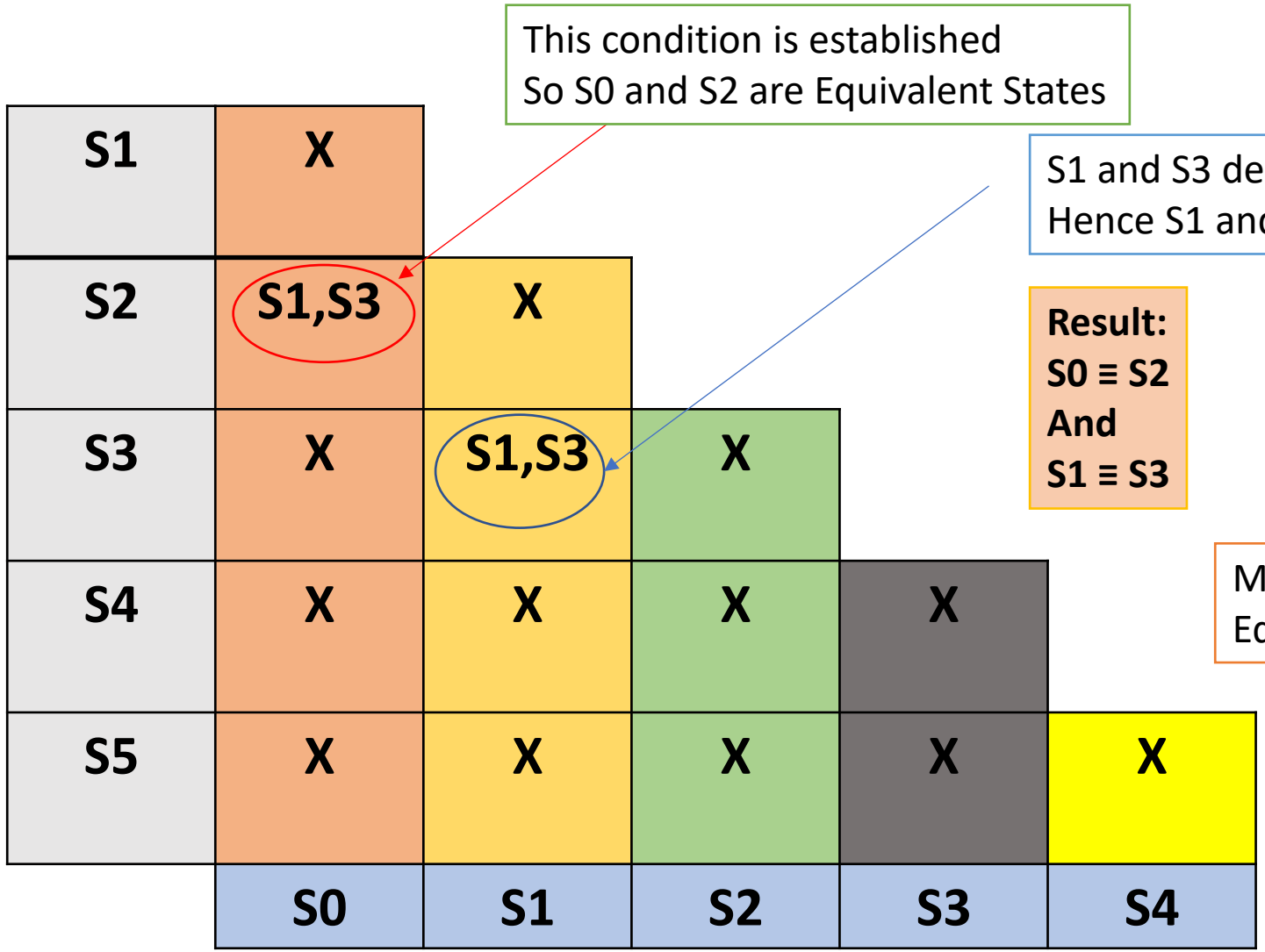
This condition is established
So S0 and S2 are Equivalent States

S1 and S3 depend on themselves
Hence S1 and S3 are Equivalent states



Result:
S0 ≡ S2
And
S1 ≡ S3

Updated further - Implication Chart



Updated State Table

Original table with six states

Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S3	0	1
S1	S5	S3	0	0
S2	S4	S1	0	1
S3	S5	S1	0	0
S4	S2	S5	0	1
S5	S1	S2	0	0

Result:
S0 ≡ S2
And
S1 ≡ S3

Only four states are left

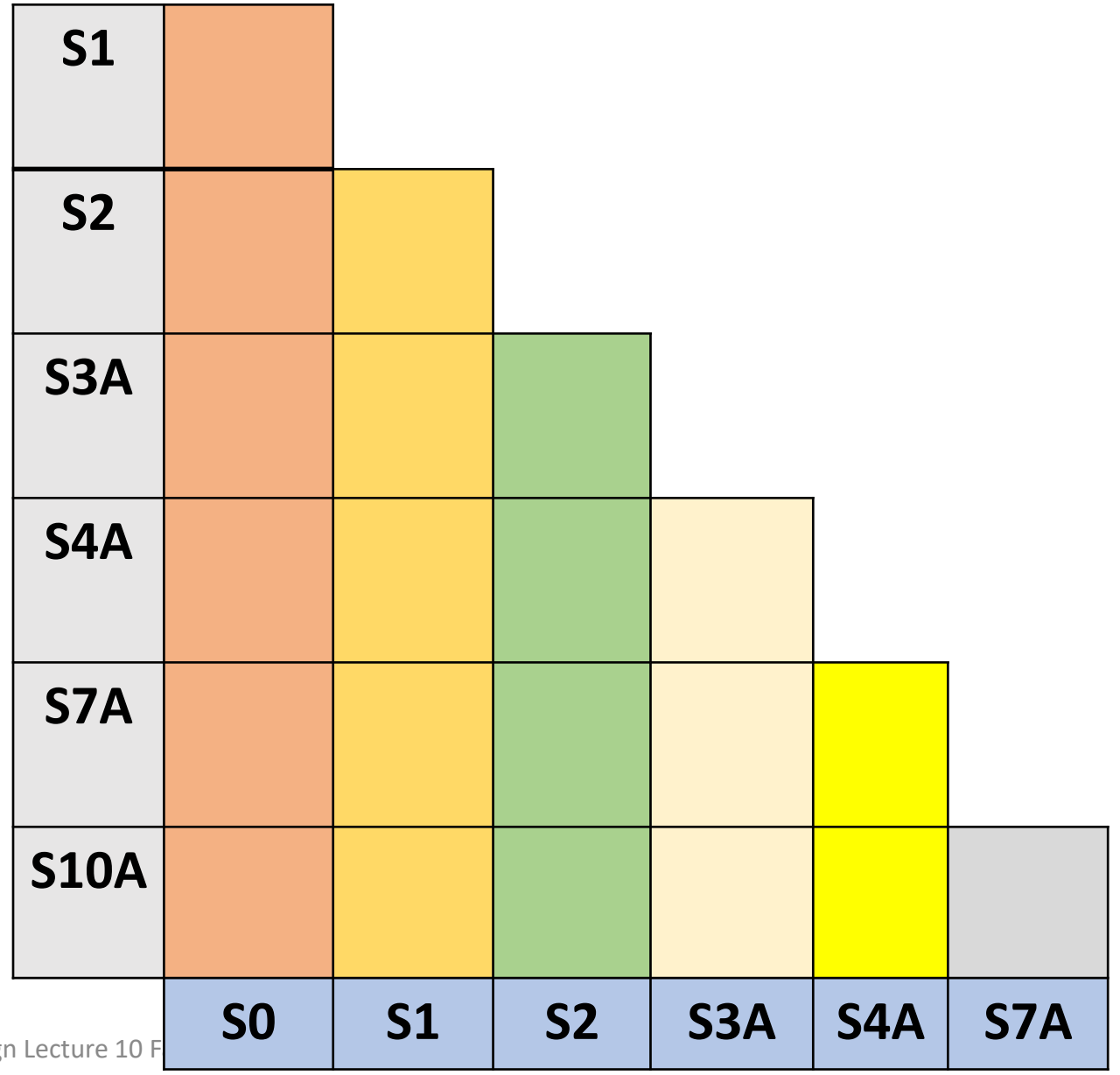
Present State	Next State		Output	
	When X=0	When X=1	When X=0	When X=1
S0	S4	S1	0	1
S1	S5	S1	0	0
~~S2~~	~~S4~~	~~S1~~	~~0~~	~~1~~
~~S3~~	~~S5~~	~~S1~~	~~0~~	~~0~~
S4	S0	S5	0	1
S5	S1	S0	0	0

Practice Example: 4-Bit Sequence Detector

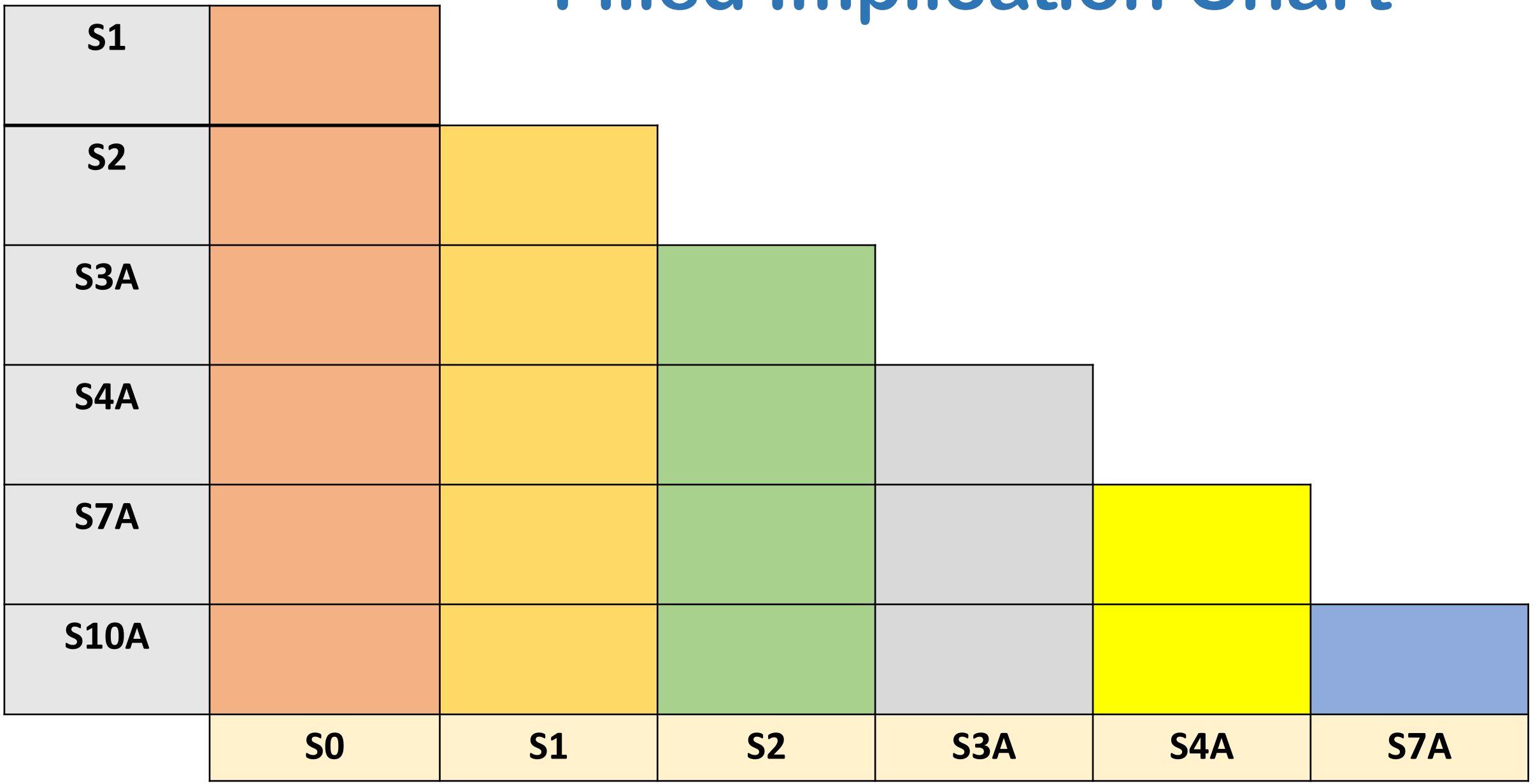
Input Sequence	Present State	Next State		Output	
		When X=0	When X=1	When X=0	When X=1
Reset	S0	S1	S2	0	0
0	S1	S3A	S4A	0	0
1	S2	S4A	S7A	0	0
00, 11	S3A	S7A	S7A	0	0
01, 10	S4A	S7A	S10A	0	0
000, 001, 010, 100, 110, 111	S7A	S0	S0	0	0
011 or 101	S10A	S0	S0	1	0

See if states can be reduced further:

Input Sequence	Present State	Next State		Output	
		When X=0	When X=1	When X=0	When X=1
Reset	S0	S1	S2	0	0
0	S1	S3A	S4A	0	0
1	S2	S4A	S7A	0	0
00, 11	S3A	S7A	S7A	0	0
01, 10	S4A	S7A	S10A	0	0
000, 001, 010, 100, 110, 111	S7A	S0	S0	0	0
011 or 101	S10A	S0	S0	1	0

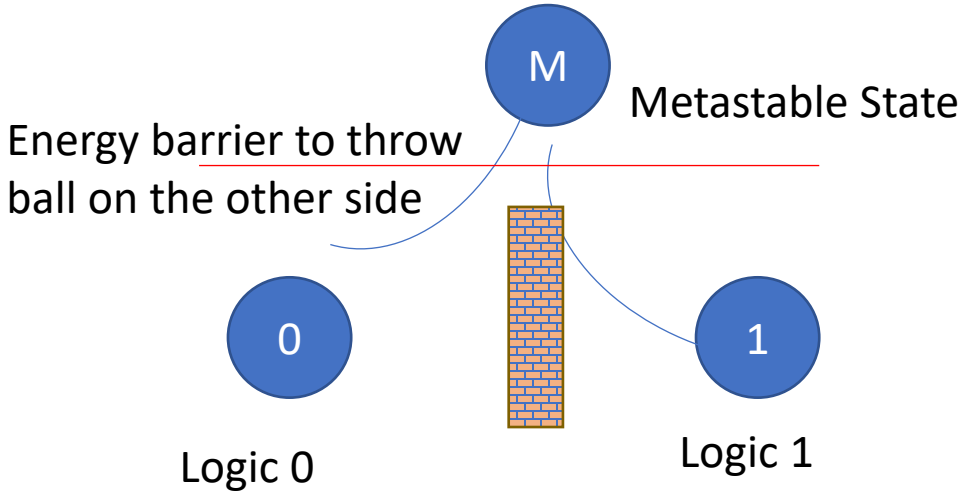


Filled Implication Chart



Metastability, and Asynchronous Data Management

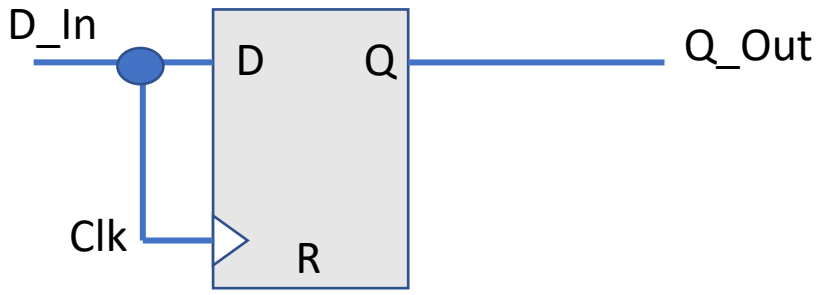
Objective – to reduce possible Metastability in capturing Asynchronous Input



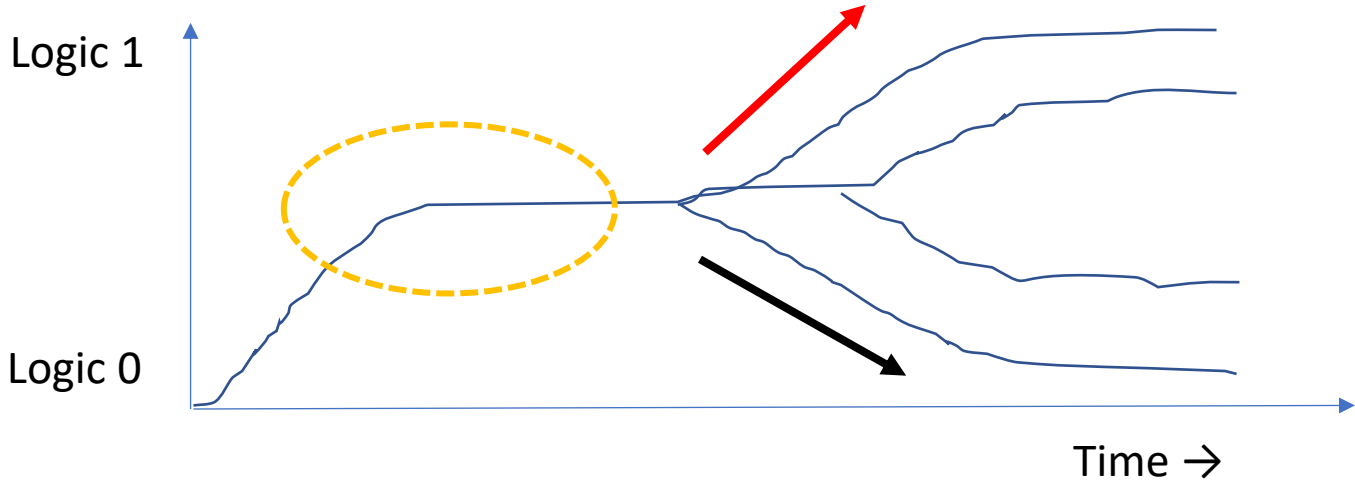
Synchronizer Failure: When flipflop hangs in a Metastable State for a long time (indefinitely)

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

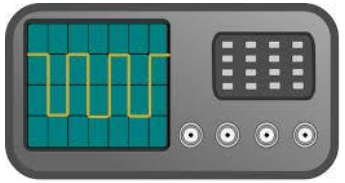
Asynchronous Domain

Asynch_In

Synchronizer (Type I or II)

Synch_Out

Ref Clk

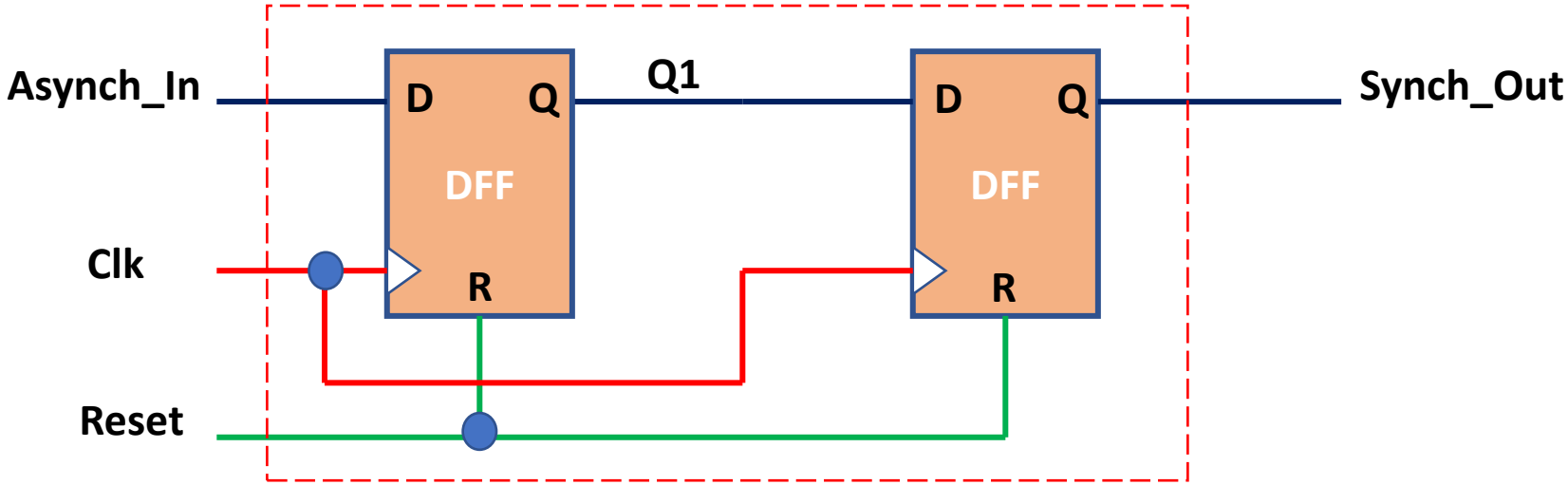


Synchronous Domain

Synchronizer 1

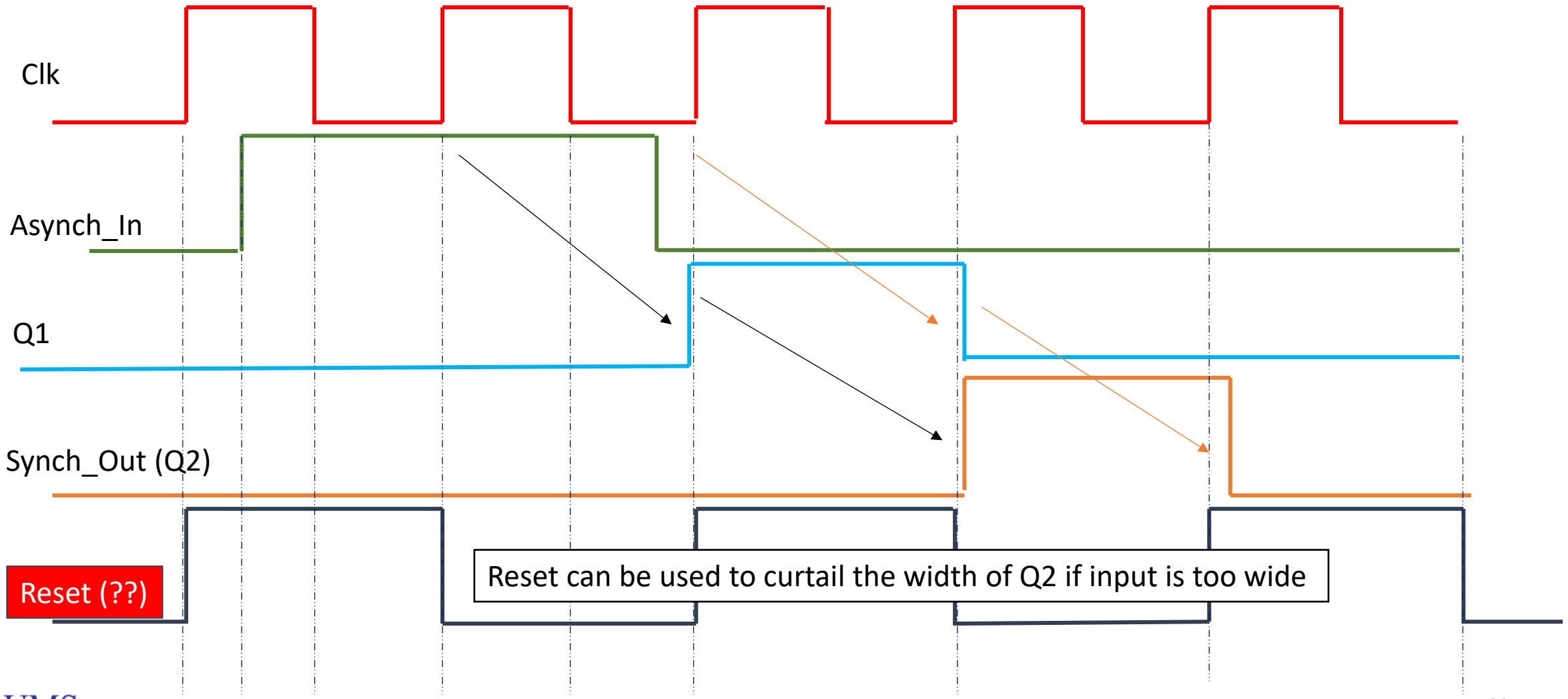
Condition: The width of asynchronous input pulse is greater than period of the clock

Synchronizer is a multistage shift register



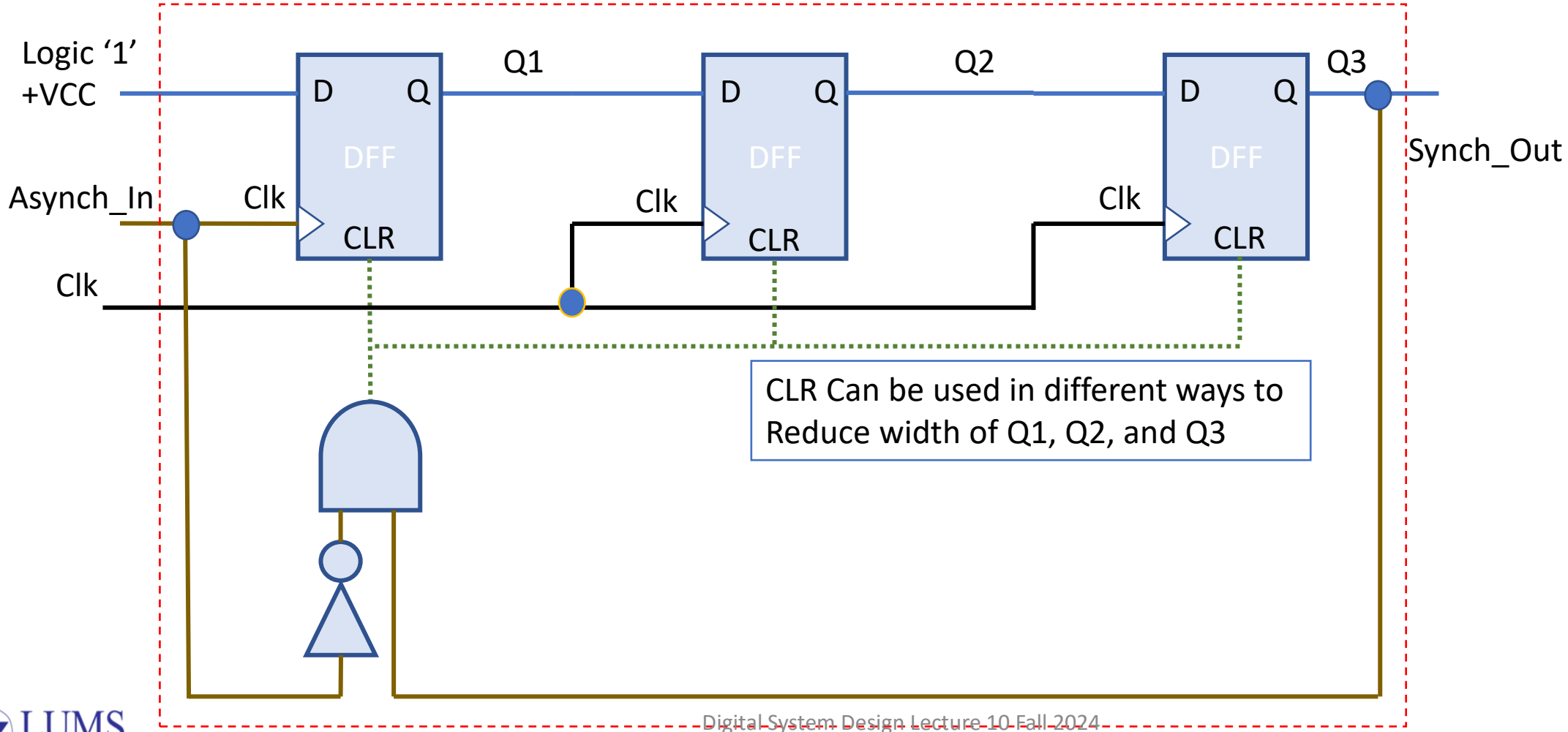
Reset Input 'R' is used as control to bring Synch_Out back to '0', as required

Timing Diagram – Synchronizer 1

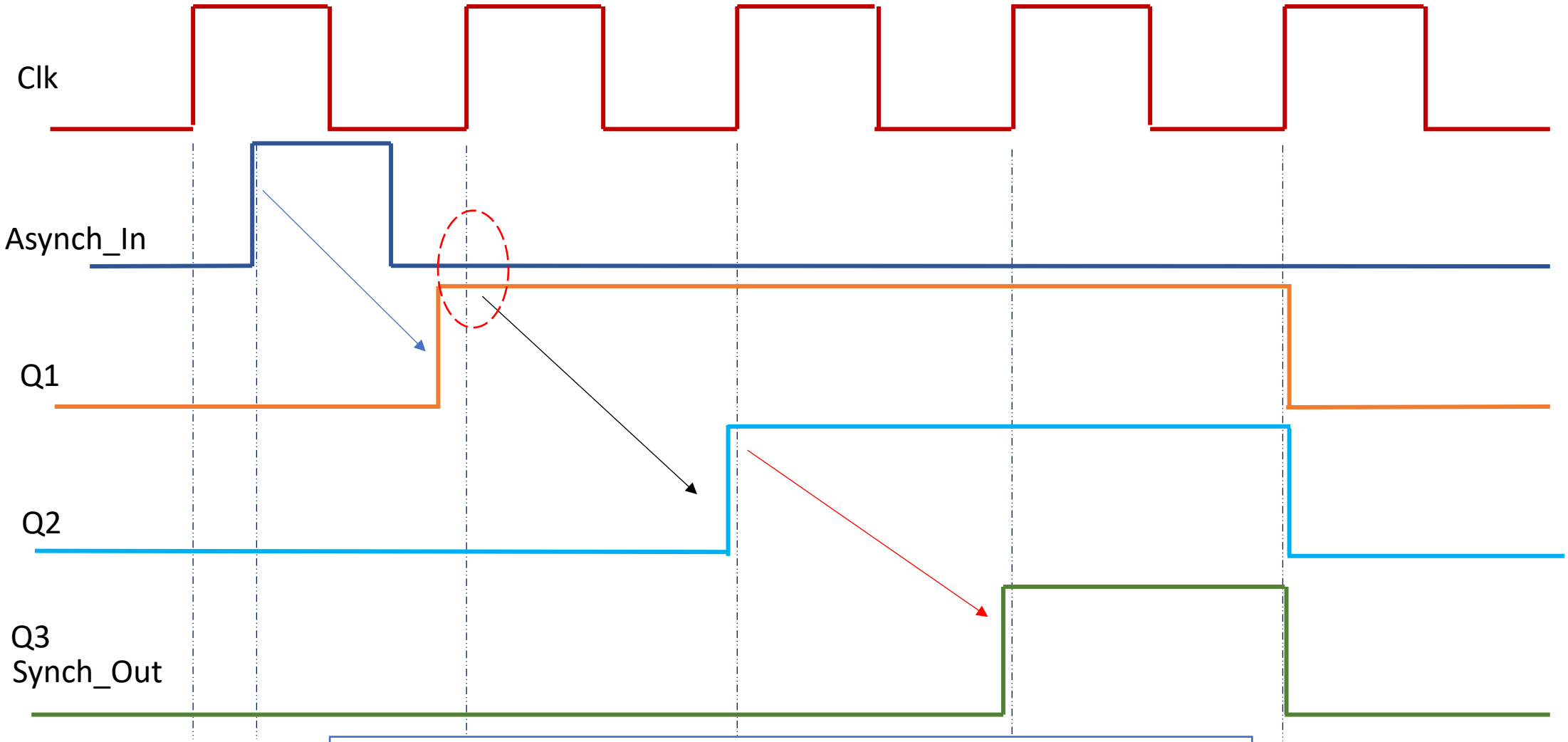


Synchronizer 2

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



Timing Diagram – Synchronizer 2

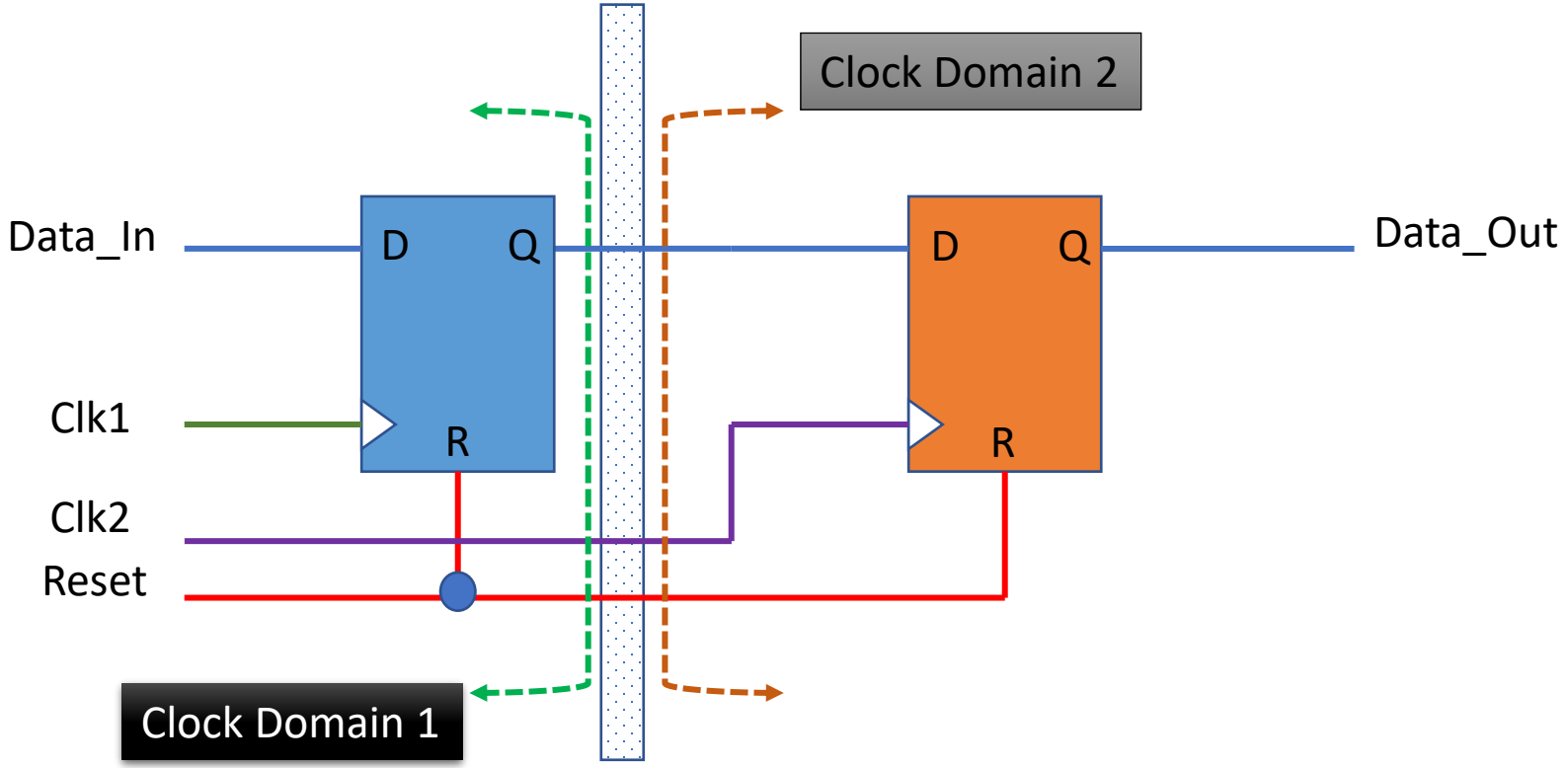


CLR input at DFF1 is used to bring back Q1 and then Q2 to Zero

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

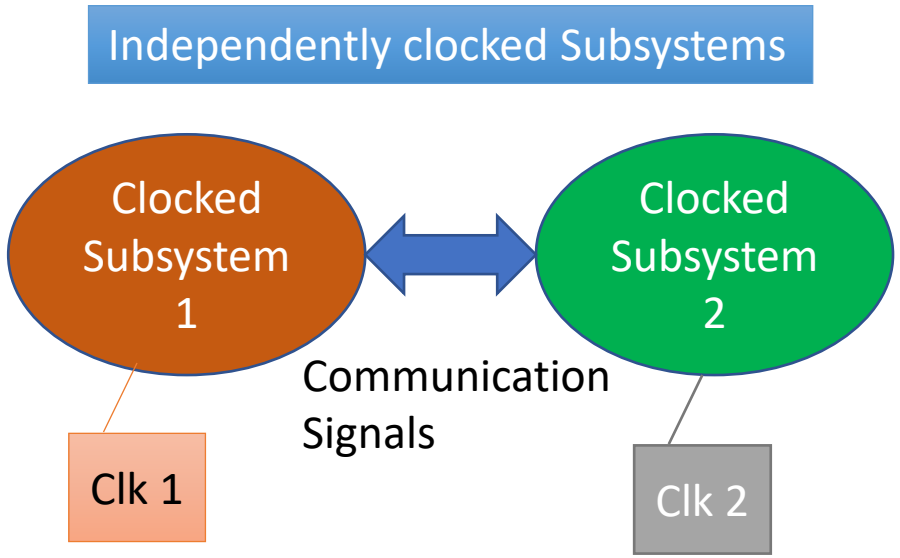
Communication across modules in Complex SoC

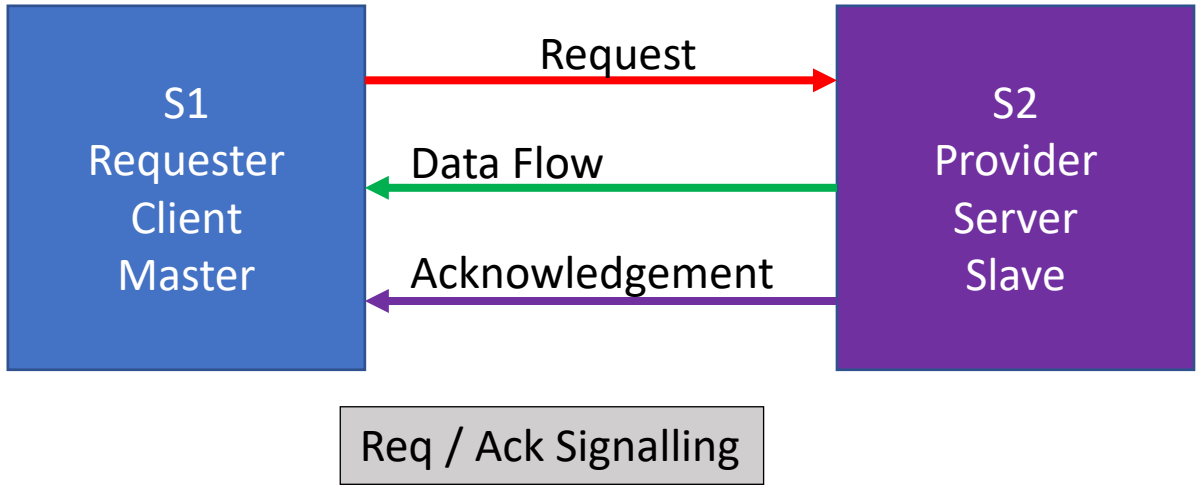
- Communication Signaling across modules at different clock speeds
- Self-timed circuits

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

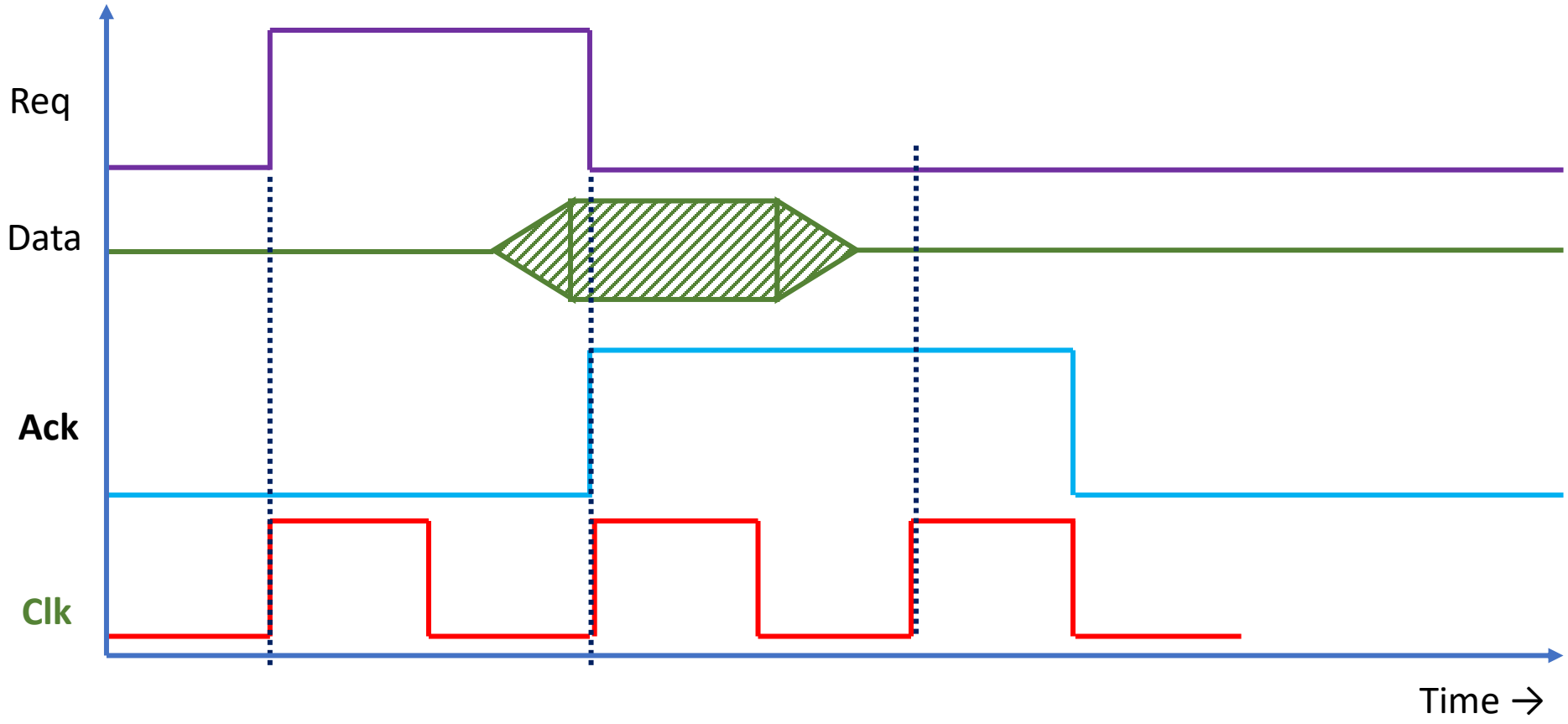
Request / Acknowledge Signaling





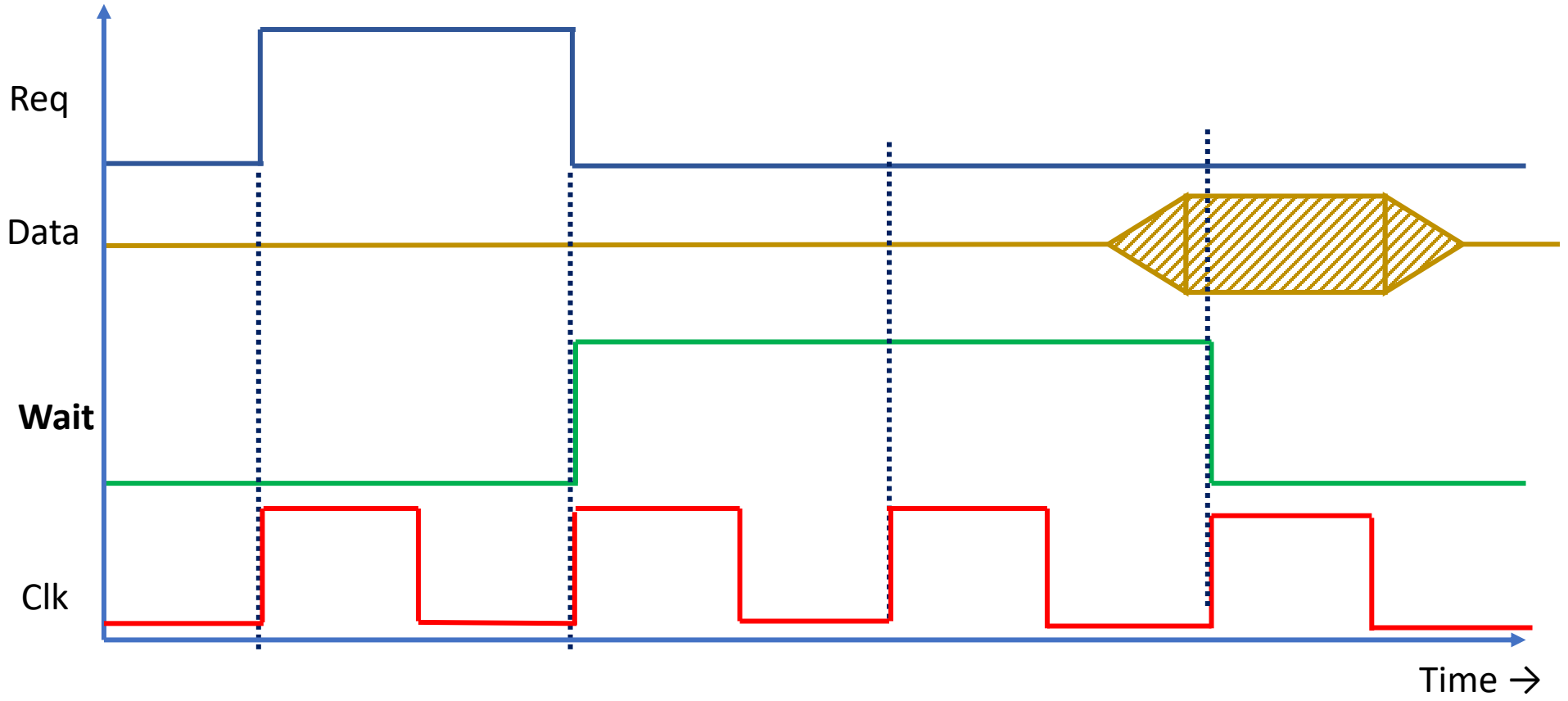
Req / Ack Signalling

Synchronous Req / Ack Signaling



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

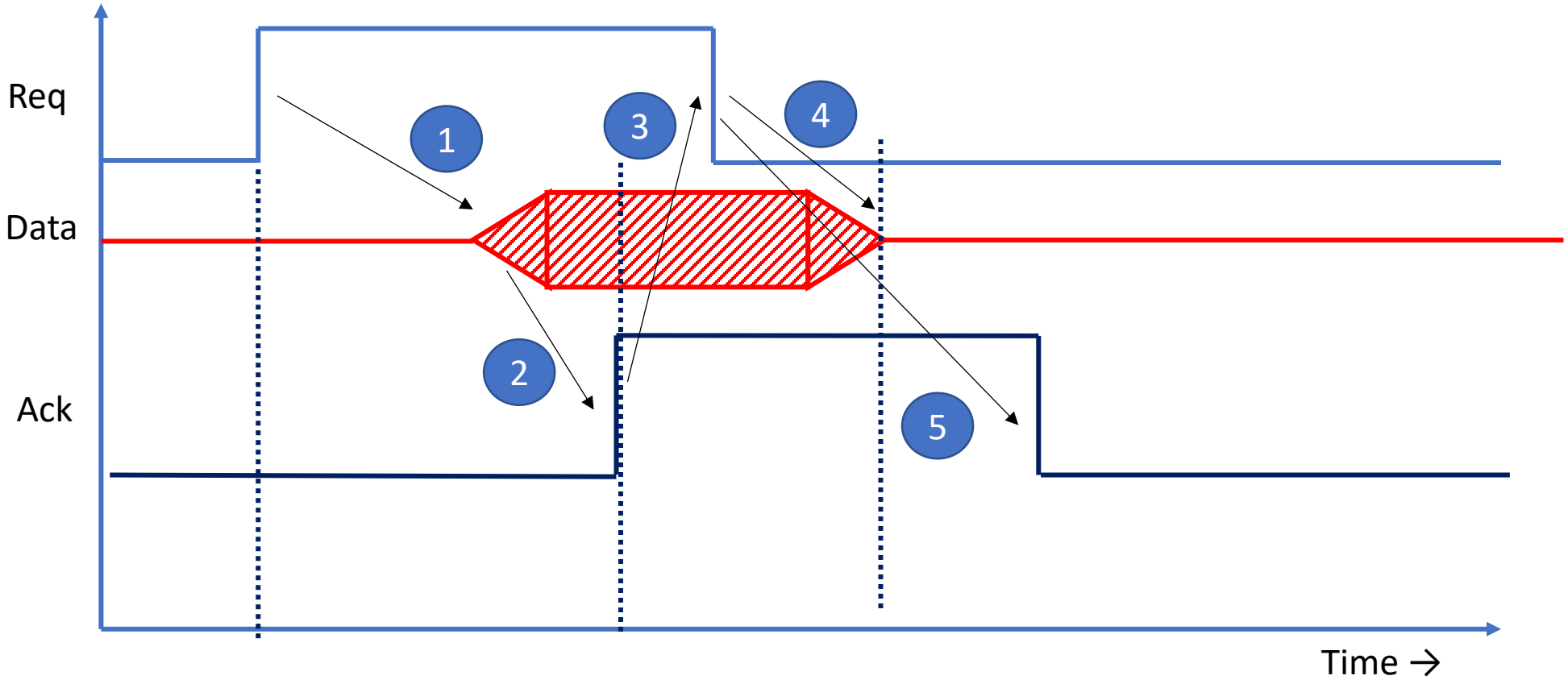
Synchronous Req with Wait Signaling



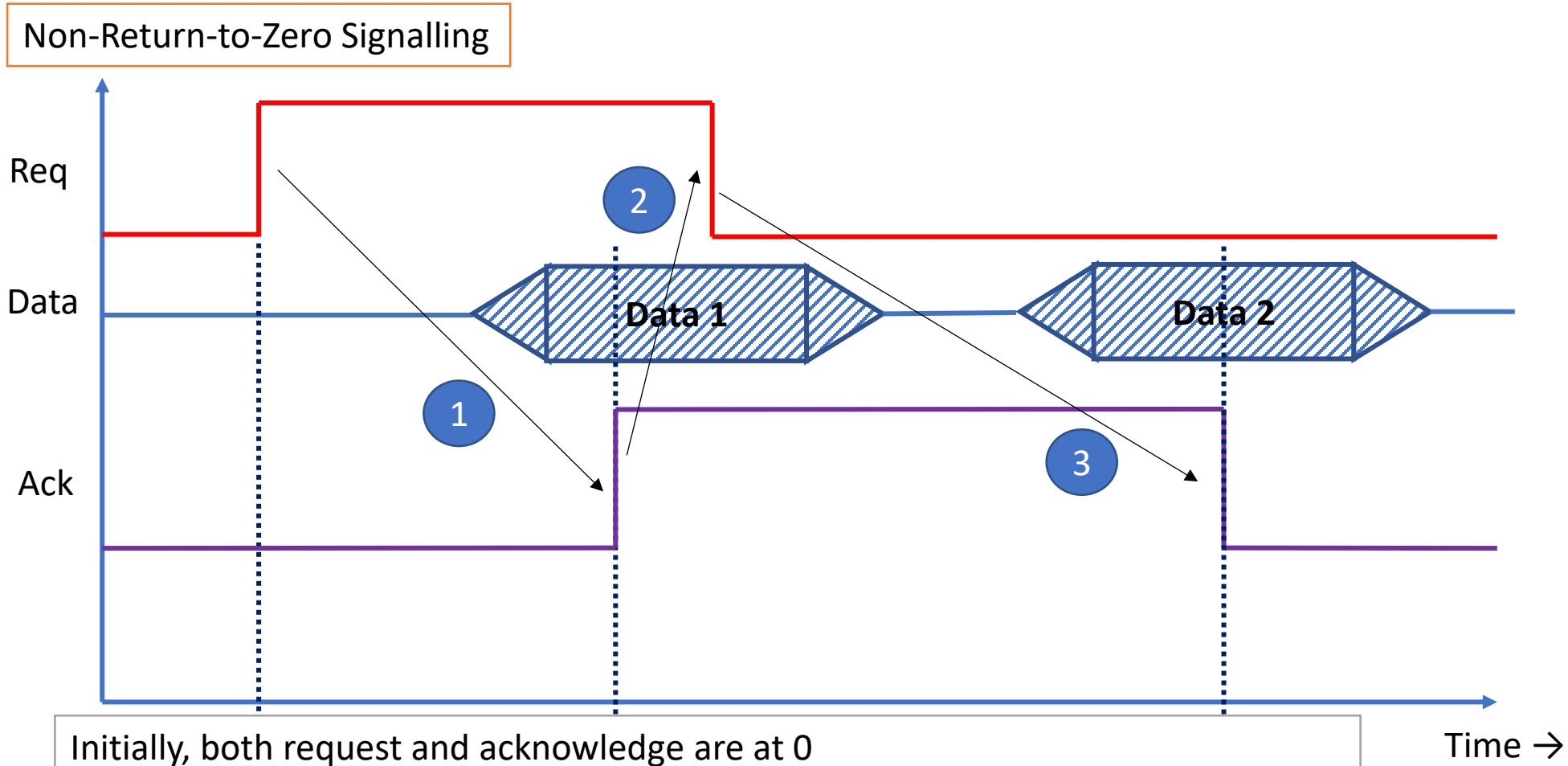
Slave can delay the Master by asserting Wait signal as it prepares the data and needs more clock cycles
When the slave un-asserts Wait signal, it acknowledges that data is now available for the Master to read
All interface signals are synchronized with Clock edge

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.