

Embedded System Architecture - CSEN 701

Module 5: Communication and Networking

Lecture 11b-12: Serial Communication (2) - Inter-Integrated Circuit

Dr. Eng. Catherine M. Elias

catherine.elias@guc.edu.eg

***Lecturer, Computer Science and Engineering,
Faculty of Media Engineering and Technology, German University in Cairo***

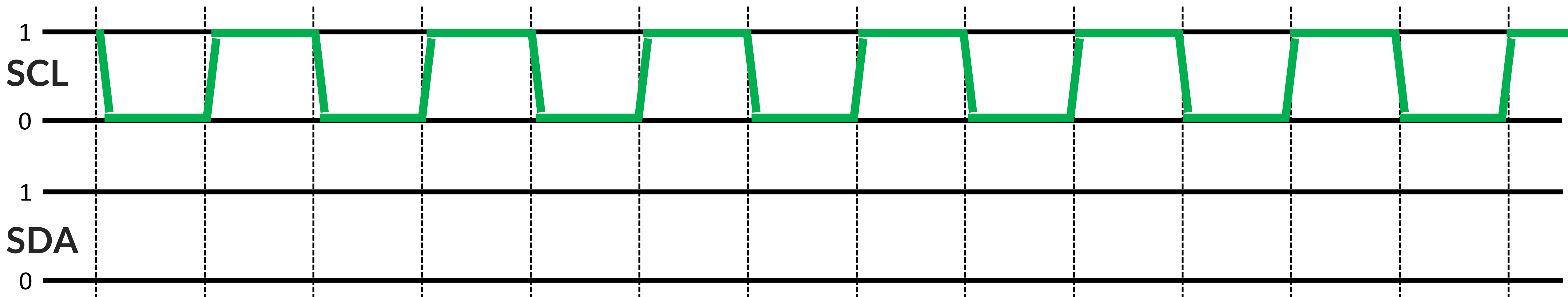
- I2C Protocol
 - Definition
 - Requirements
 - Protocol Syntax
 - Multi-Master I2C

Definition

- I2C, which stands for Inter-Integrated Circuit, is a **synchronous, multi-master, multi-slave**, packet-switched serial communication protocol.
- It was developed by Philips Semiconductors (now NXP Semiconductors) in the early 1980s and is widely used for connecting various components on a single circuit board or between different boards.
- I2C is a popular choice for communication between microcontrollers, sensors, EEPROMs (Electrically Erasable Programmable Read-Only Memory), real-time clocks, and other integrated circuits.

Requirements

- Synchronous or Asynchronous? **Synchronous**
- Two-Wire Communication
 1. **SCL (Serial Clock)**: The clock line in the I2C bus. It provides synchronization for data transmission between devices.
 2. **SDA (Serial Data)**: The data line in the I2C bus. It is **bidirectional** and used for transmitting and receiving data between devices.

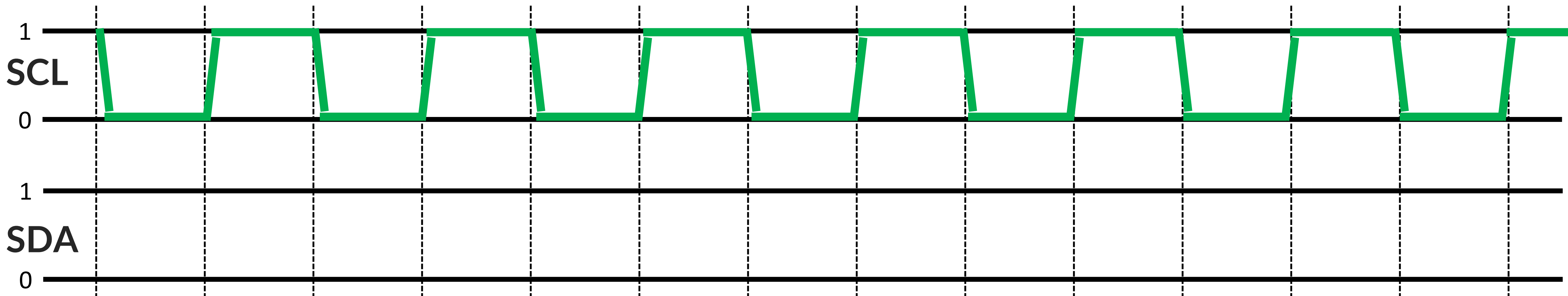


Requirements

- Two-Wire Communication

➤ Data on the I2C-bus can be transferred at rates of:

- up to 100 kbit/s in the Standard-mode,
- up to 400 kbit/s in the Fast-mode,
- up to 1 Mbit/s in Fast-mode Plus,
- or up to 3.4 Mbit/s in the High-speed mode.

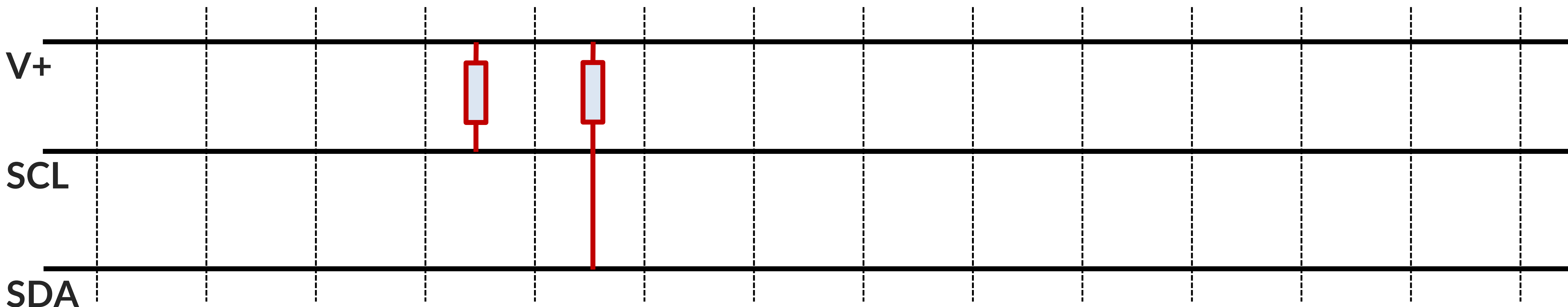


Requirements

- Two-Wire Communication

- Both SDA and SCL are **bidirectional** lines, connected to a positive supply voltage via a current-source or **pull-up resistor**
- Pull-Up Resistors are connected to the SDA and SCL lines to maintain a **high** signal level when no device is actively pulling the line low.

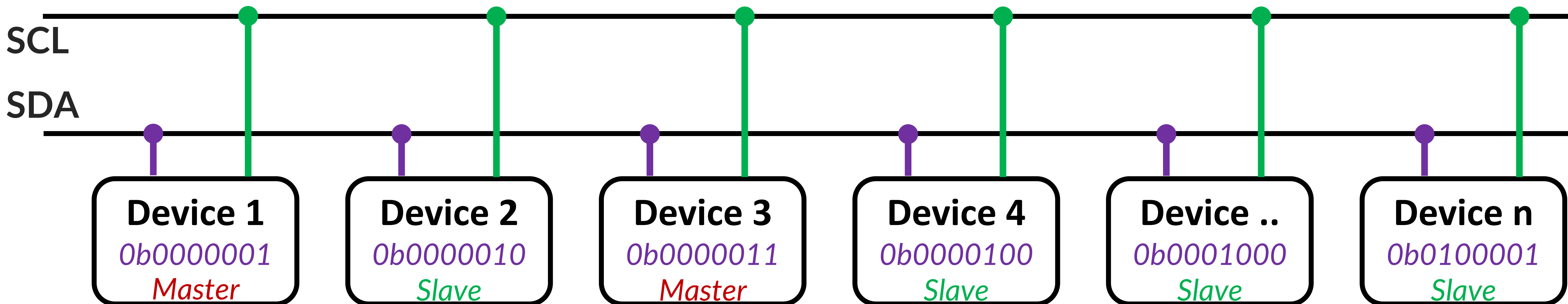
When the bus is free, both lines are HIGH.



Requirements

• Master-Slave Architecture

1. **Address:** A unique identifier assigned to each I2C device. It can be 7 bits (standard addressing) or 10 bits (extended addressing) in length.
2. **Master:** The device that initiates and controls communication on the I2C bus. The master generates start and stop conditions and controls the clock.
3. **Slave:** A device that responds to commands from the master on the I2C bus. Slaves have unique addresses and can be addressed by the master for data exchange.

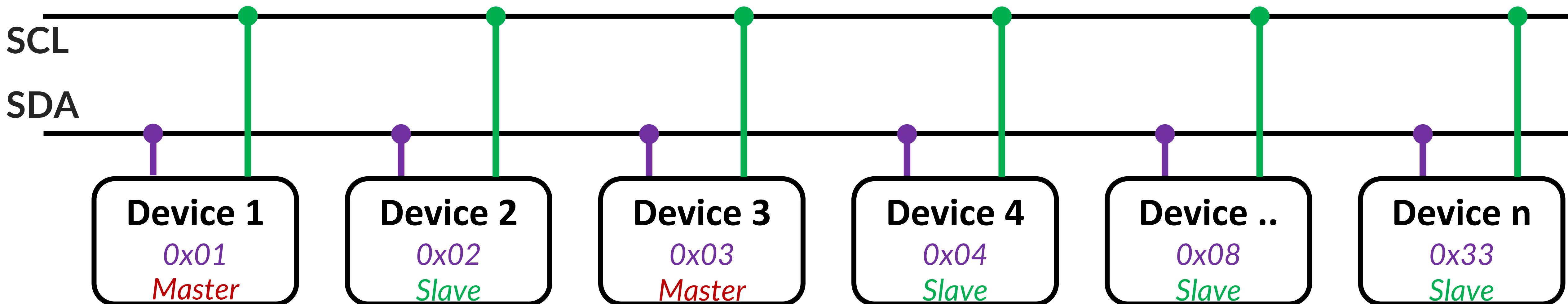


Requirements

- **Master-Slave Architecture:** Example

➤ Assuming Device 1 is the master and wants to send information to Device 4 (slave):

1. Device 1, **addresses** Device 4
2. Device 1 (master-transmitter) **sends** data to Device 4 (slave-receiver)
3. Device 1 **terminates** the transfer.

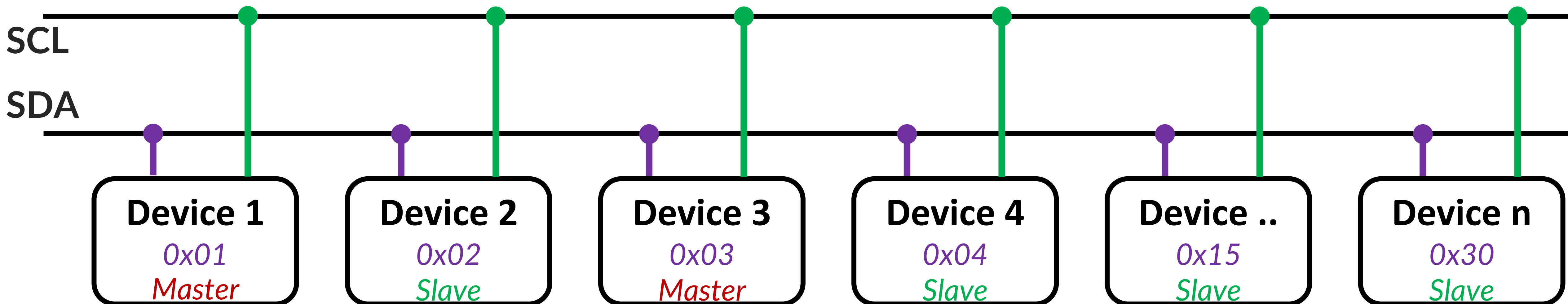


Requirements

- **Master-Slave Architecture:** Example

➤ Assuming Device 1 is the master and wants to receive information to Device 4 (slave):

1. Device 1, **addresses** Device 4
2. Device 4 (slave-transmitter) **sends** data to Device 1 (master-receiver)
3. Device 1 **terminates** the transfer.



Requirements

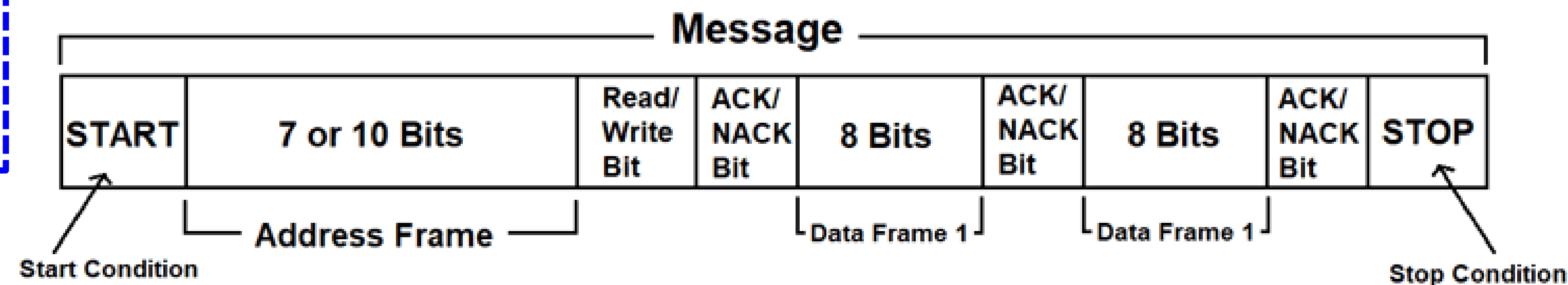
- Protocol Syntax

Start Bit (0)	Data Bits (1:8)	Ack Bit (9)	...	Stop Bit (10)
1 bit	8 bits	1 bit		1 bit

Data Frame

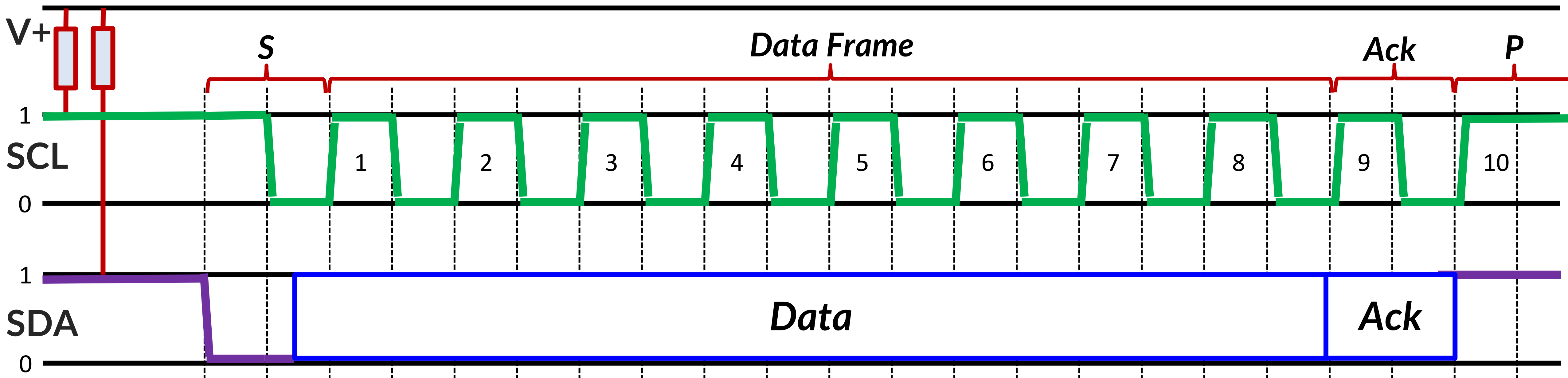
Initiation
Slave address
(7 bits) + R/W bit

Data Transfer
1 byte Data



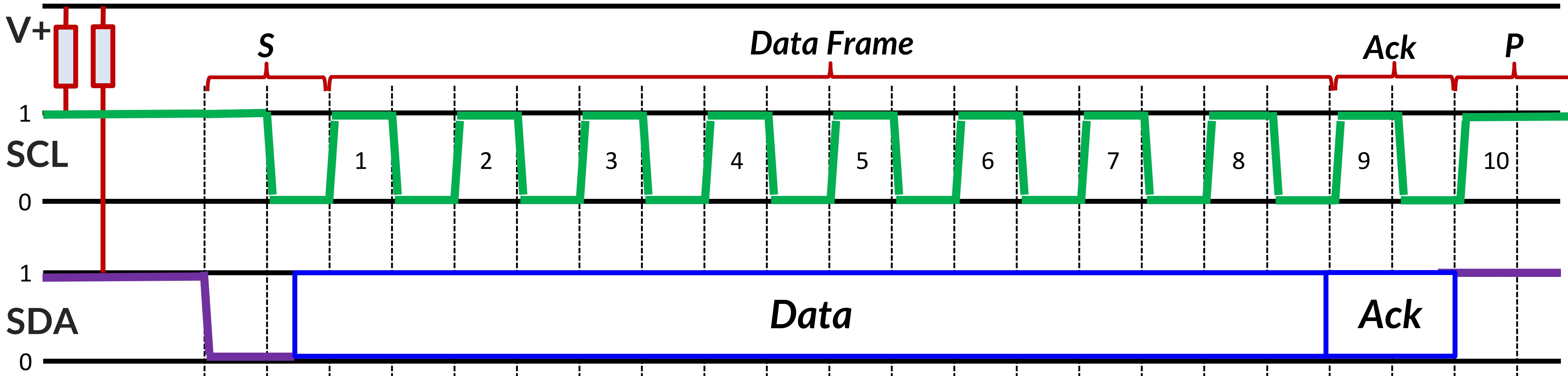
Start and Stop Conditions

- All transactions begin with a START (S) and are terminated by a STOP (P).
- **Start Condition (S):** It is initiated by the master pulling the **SDA** line **low** while the **SCL** line is **high**.
- **Stop Condition (P):** It is initiated by the master releasing the **SDA** line from **low** to **high** while the **SCL** line is **high**.



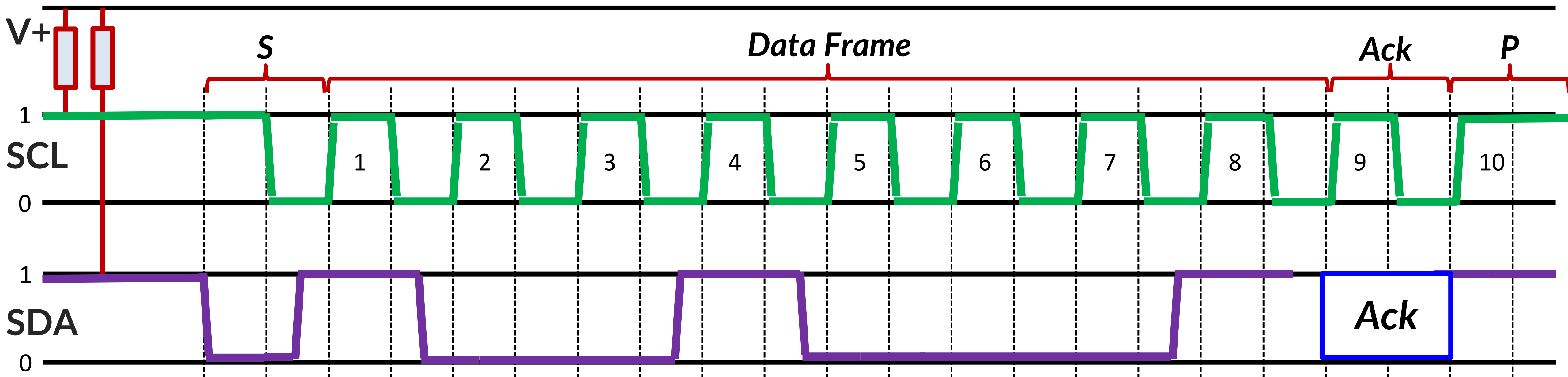
Byte Transfer

- The **number of bytes** that can be transmitted per transfer is **unrestricted**.
- **Each byte** must be **followed** by an **Acknowledge bit**.
- Data is transferred with the **MSB first**.
- New bit starts transitioning from the middle of the low half of the cycle.



Byte Transfer

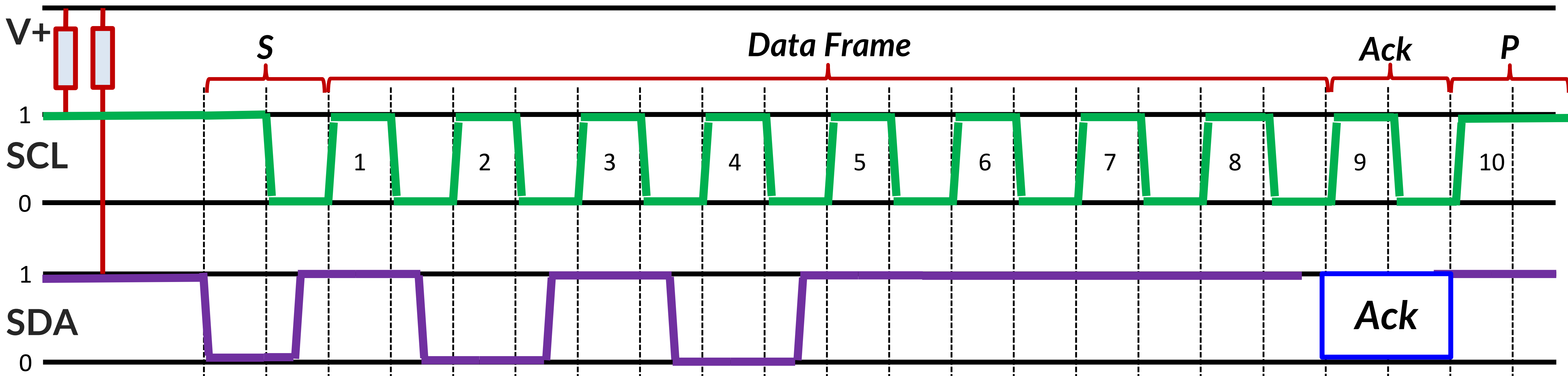
- **Slave Address + Read/Write Bit:**
 - The 7-bit or 10-bit address of the target slave device, followed by a Read/Write bit.
 - The Read/Write bit indicates whether the master intends to read (1) from or write (0) to the slave.
 - Example: Slave is Device of address 0b1001000 writes to master



Byte Transfer

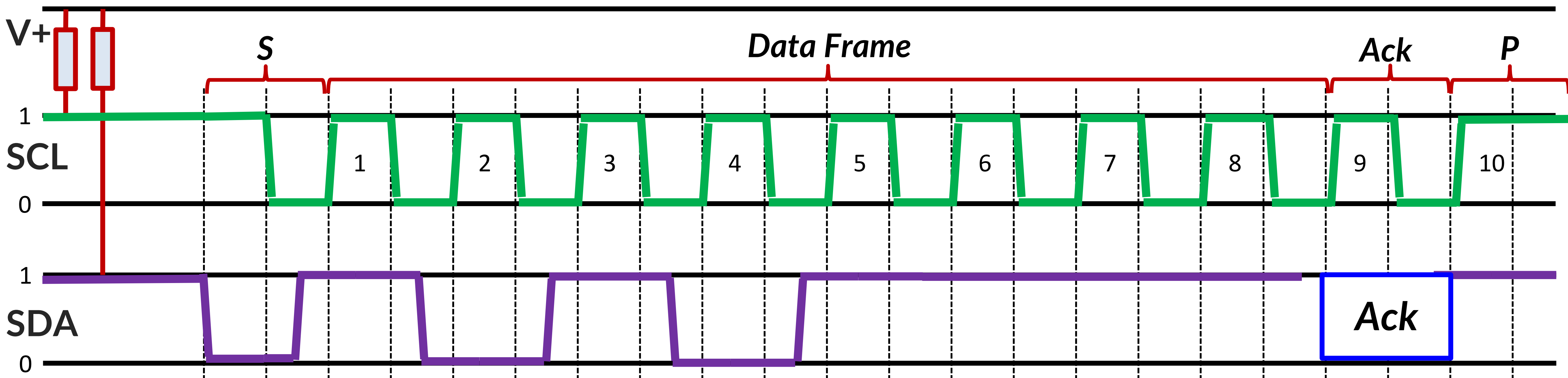
- **Data Bytes:**

- The actual data being transmitted, one byte at a time.
- Data is typically 8 bits long but can be extended to more bits if needed.
- Example: Slave address 0b1001000 writes to master a byte message of 0b10101111



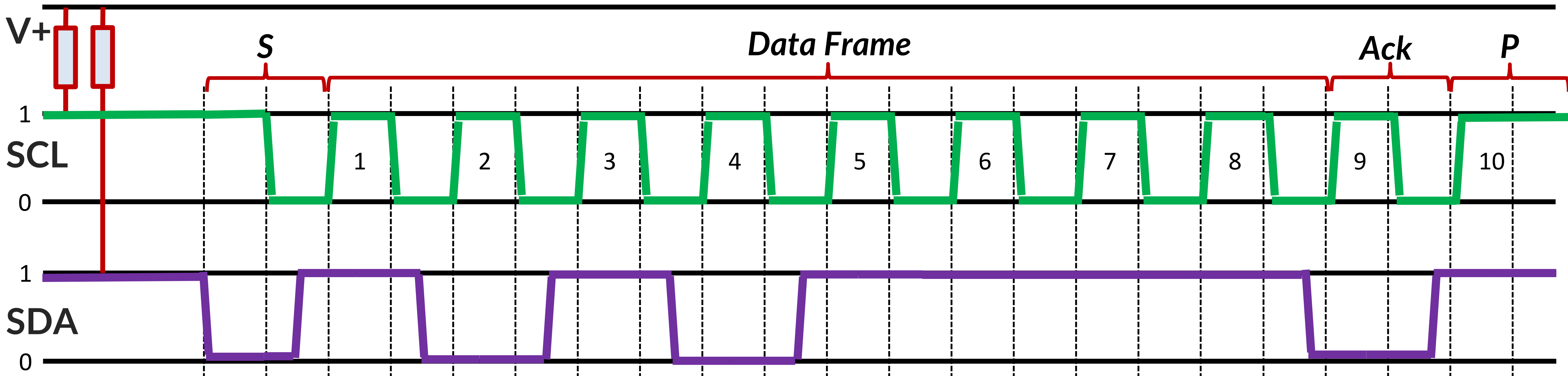
Acknowledgment

- After each byte of data, the receiving device sends an acknowledgment bit (ACK) to confirm **successful reception**.
- The absence of an acknowledgment indicates **an error** or the end of data.
- The master generates the acknowledge ninth clock pulse.



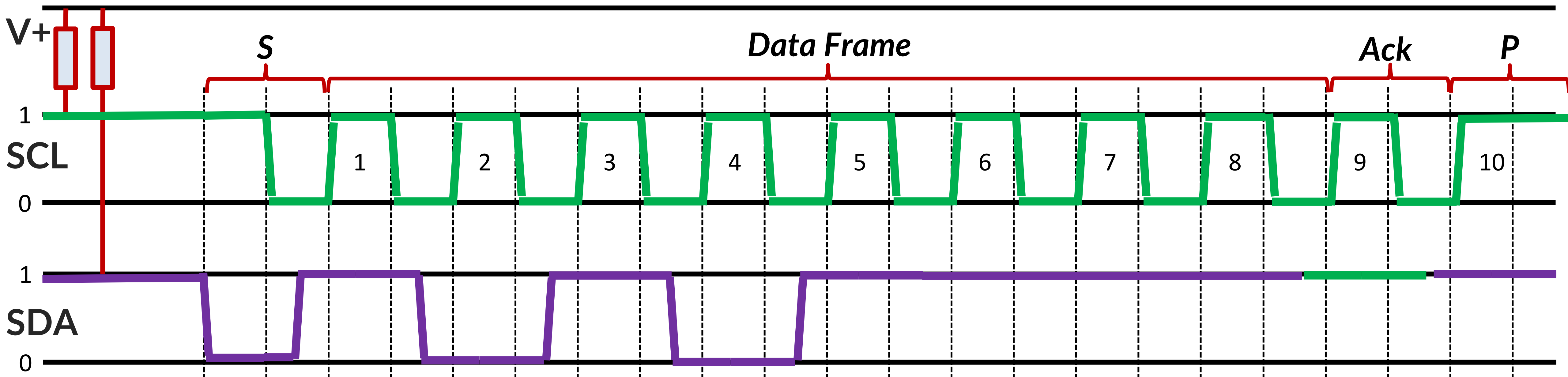
Acknowledgment

- The Acknowledge signal is defined as follows:
 - the transmitter releases the **SDA line** during the **acknowledge clock pulse**
 - An acknowledgment bit (Ack) is pulled **low** by the receiver, indicating successful reception.



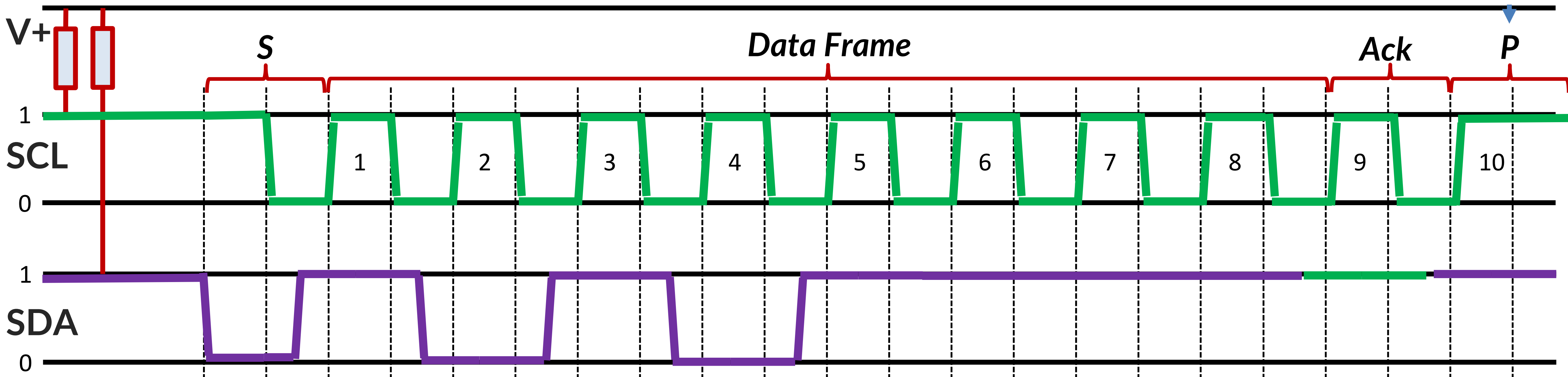
N-Acknowledgment

- The **N**-Acknowledge signal is defined as follows:
 - the transmitter releases the **SDA line** during the **acknowledge clock pulse**
 - An **non**-acknowledgment bit (**NAck**) is pulled **High** by the receiver, indicating an error in reception.



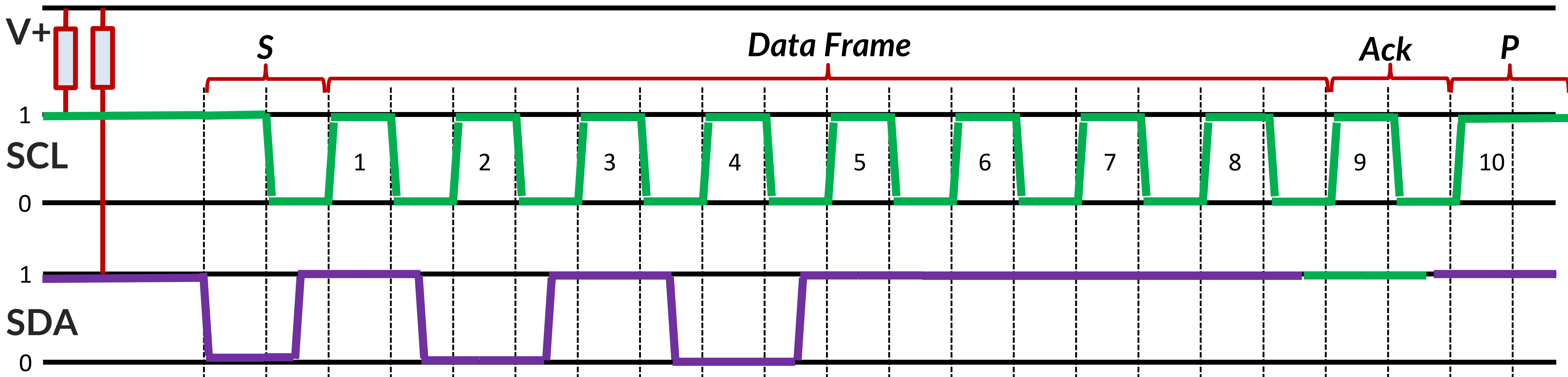
N-Acknowledgment

- In case of N-Acknowledge signal, the master can then generate either:
 - a STOP condition to abort the transfer, or
 - a repeated START condition to start a new transfer.



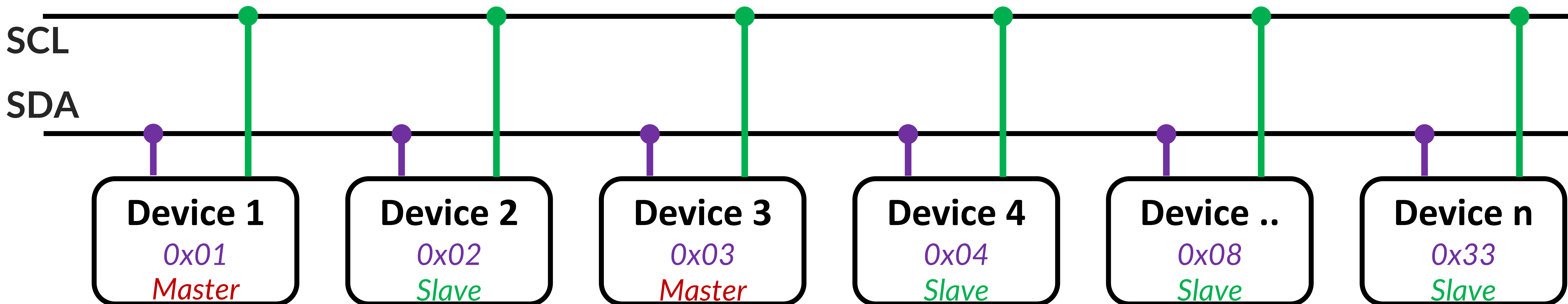
N-Acknowledgment

- There are four reasons that lead to the generation of a NACK.
 - No receiver is present** on the bus with the transmitted address **to respond** with ACK.
 - The **receiver is unable to transmit** because it is dealing with a real-time function
 - During the transfer, the receiver gets data or commands that it **does not understand**.
 - During the transfer, the receiver **cannot receive any more** data bytes.



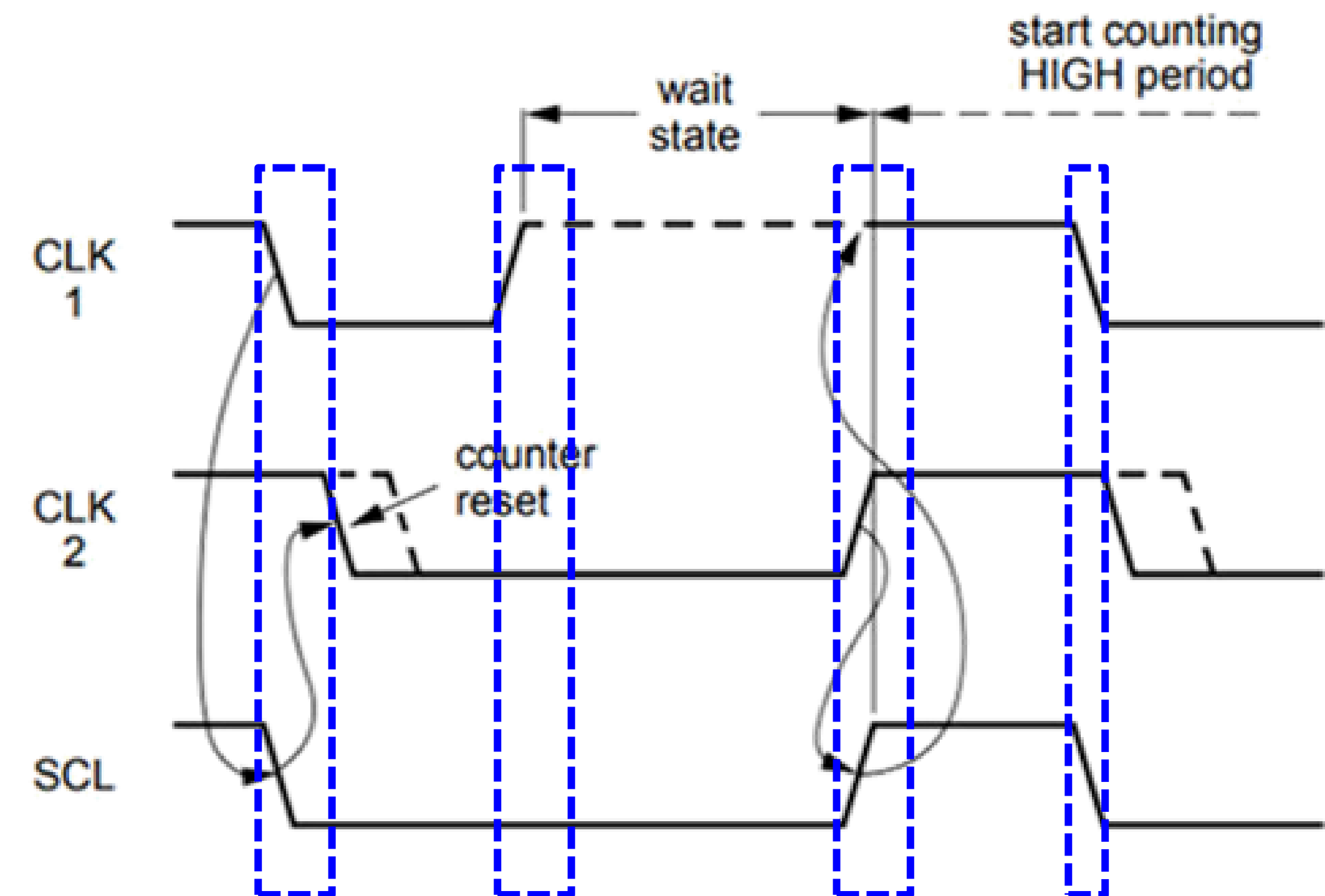
Multi-Master I2C

- Two masters can begin transmitting on a free bus at the same time.
- There must be a method for deciding *which takes control of the bus* and complete its transmission.
- This is done by clock **synchronization** and **arbitration**.
- Note that in single master systems, clock synchronization and arbitration are not needed.



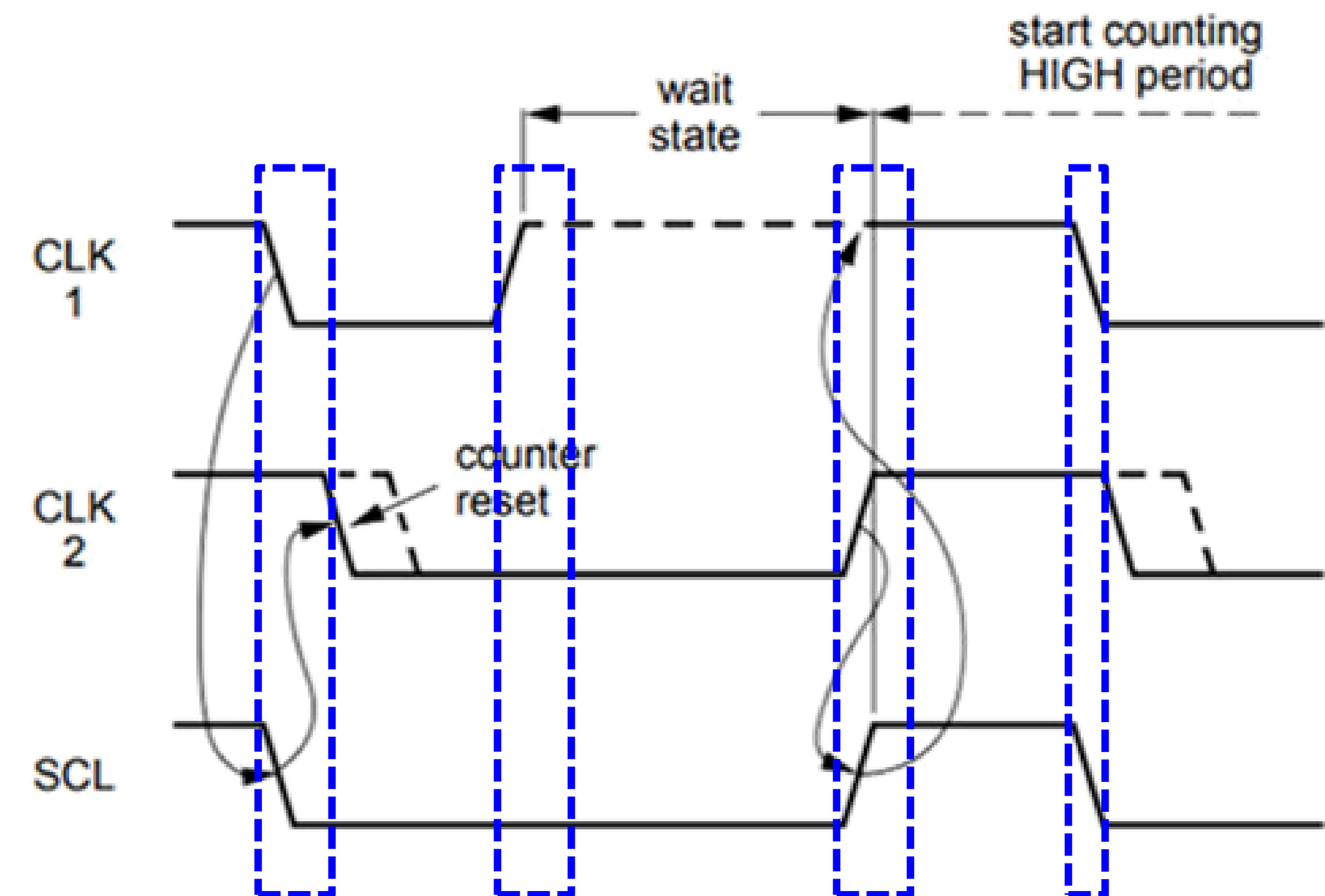
Clock Synchronization

- Clock synchronization is performed using the **wired-AND** connection of I2C interfaces to the SCL line.
- A **HIGH to LOW transition on the SCL** line causes the masters concerned to start **counting off** their LOW period.
- Once a master clock has gone LOW, it holds the SCL line in that state until the clock HIGH state is reached
- if another clock is still within its LOW period, the LOW to HIGH transition of this clock may not change the state of the SCL line.



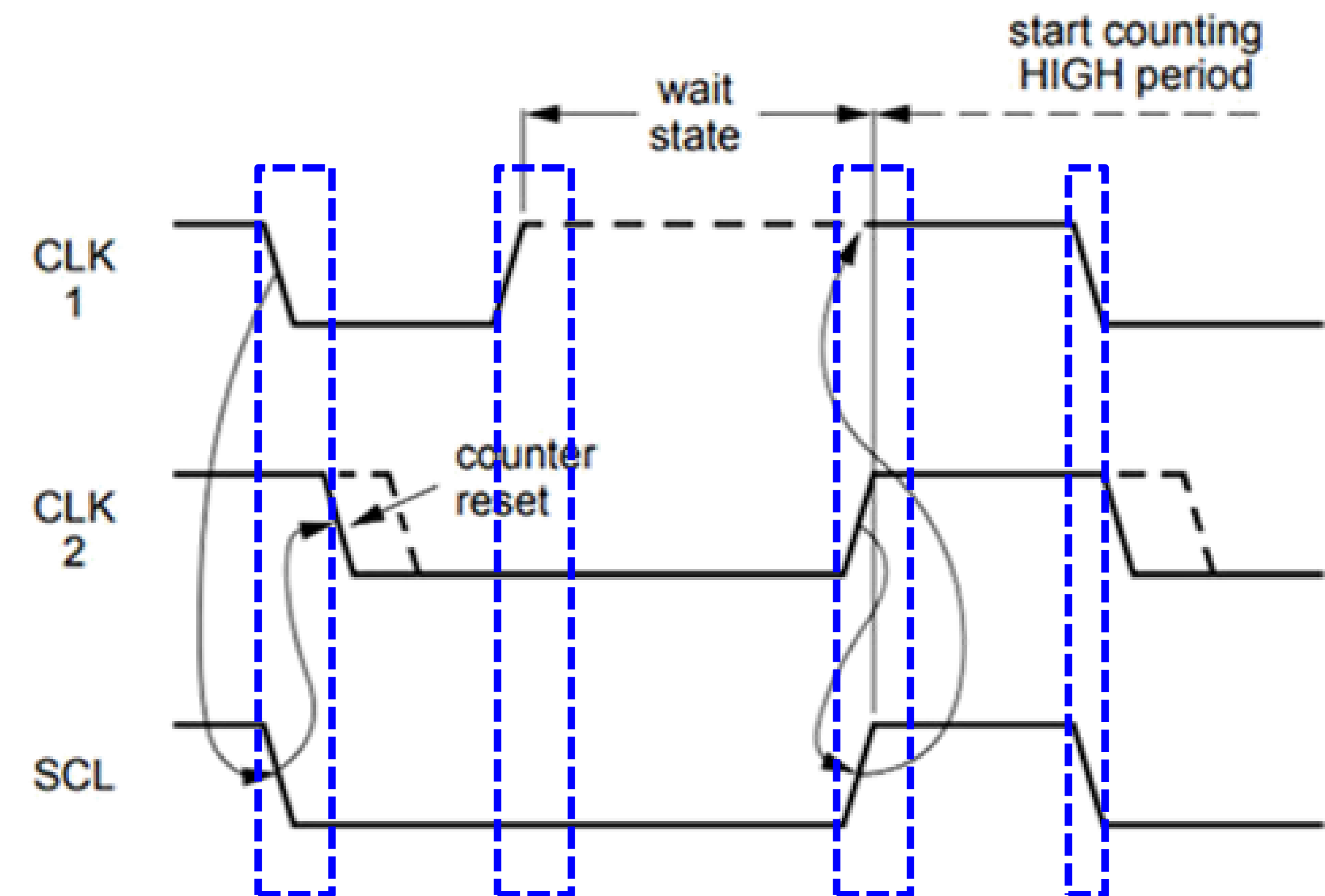
Clock Synchronization

- The **SCL line** is therefore held **LOW** by the **master with the longest LOW period**.
- Masters with shorter LOW periods enter a **HIGH wait-state** during this time.



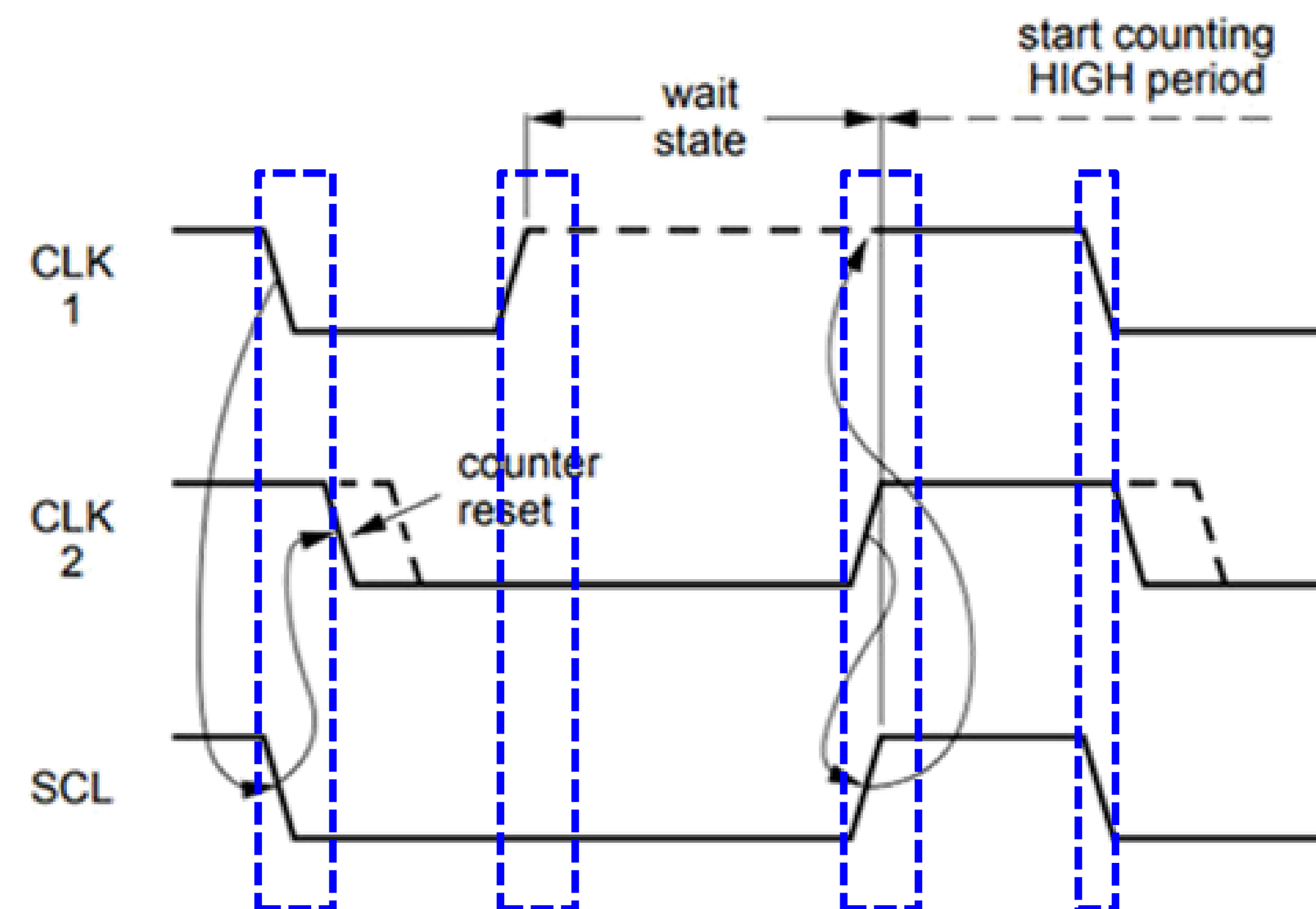
Clock Synchronization

- When all masters concerned have counted off their LOW period, the clock line is released and goes HIGH.
- There is then no difference between the master clocks and the state of the SCL line, and all the masters start counting their HIGH periods.
- The first master to complete its HIGH period pulls the SCL line LOW again.



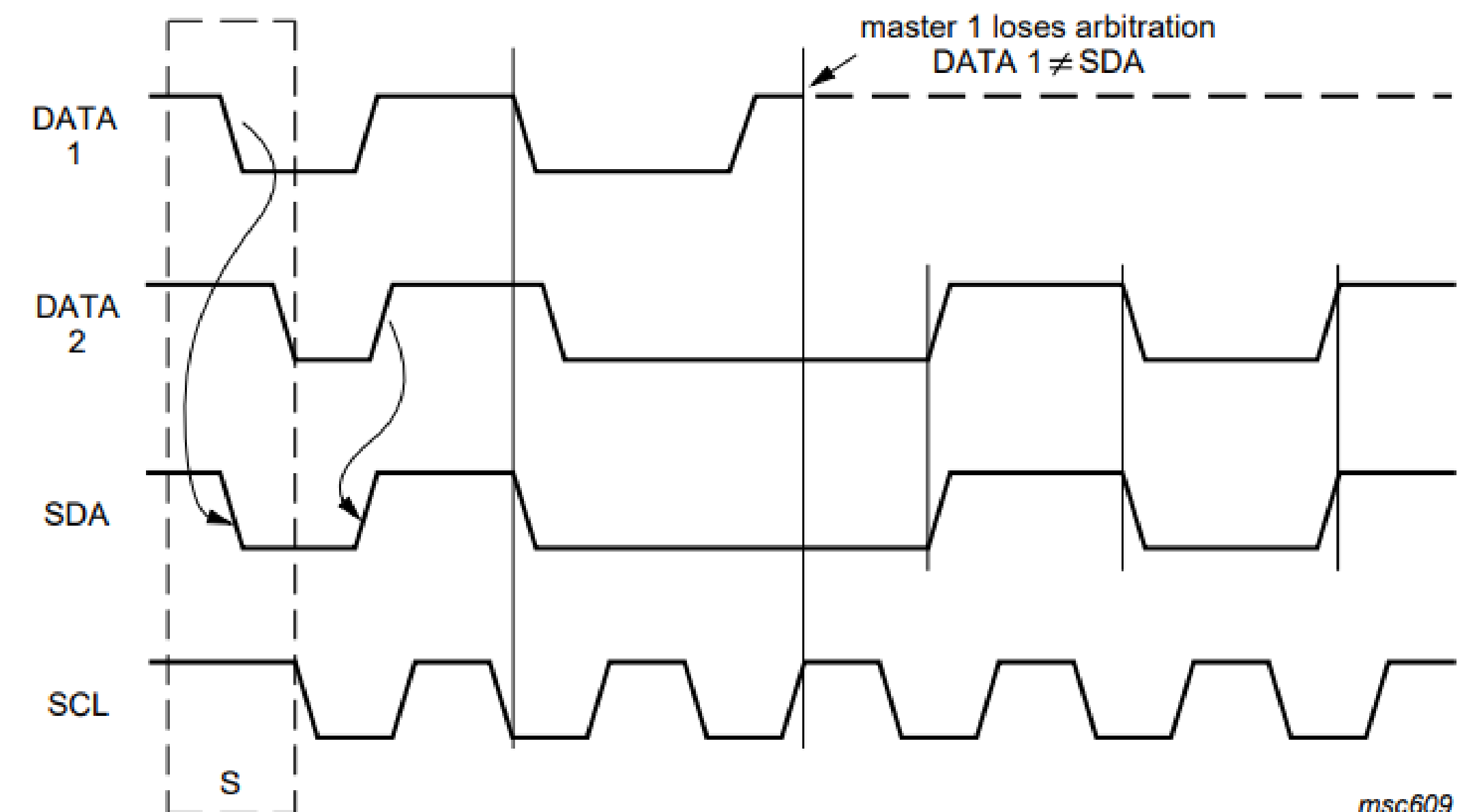
Clock Synchronization

- Example:
 - if one master clock goes LOW for 200ms & then goes in HIGH state.
 - another master goes LOW for 400ms then goes in high state.
 - the complete LOW period for the SCL will be 400ms only and the master with shorter LOW clock period has to wait in HIGH state.



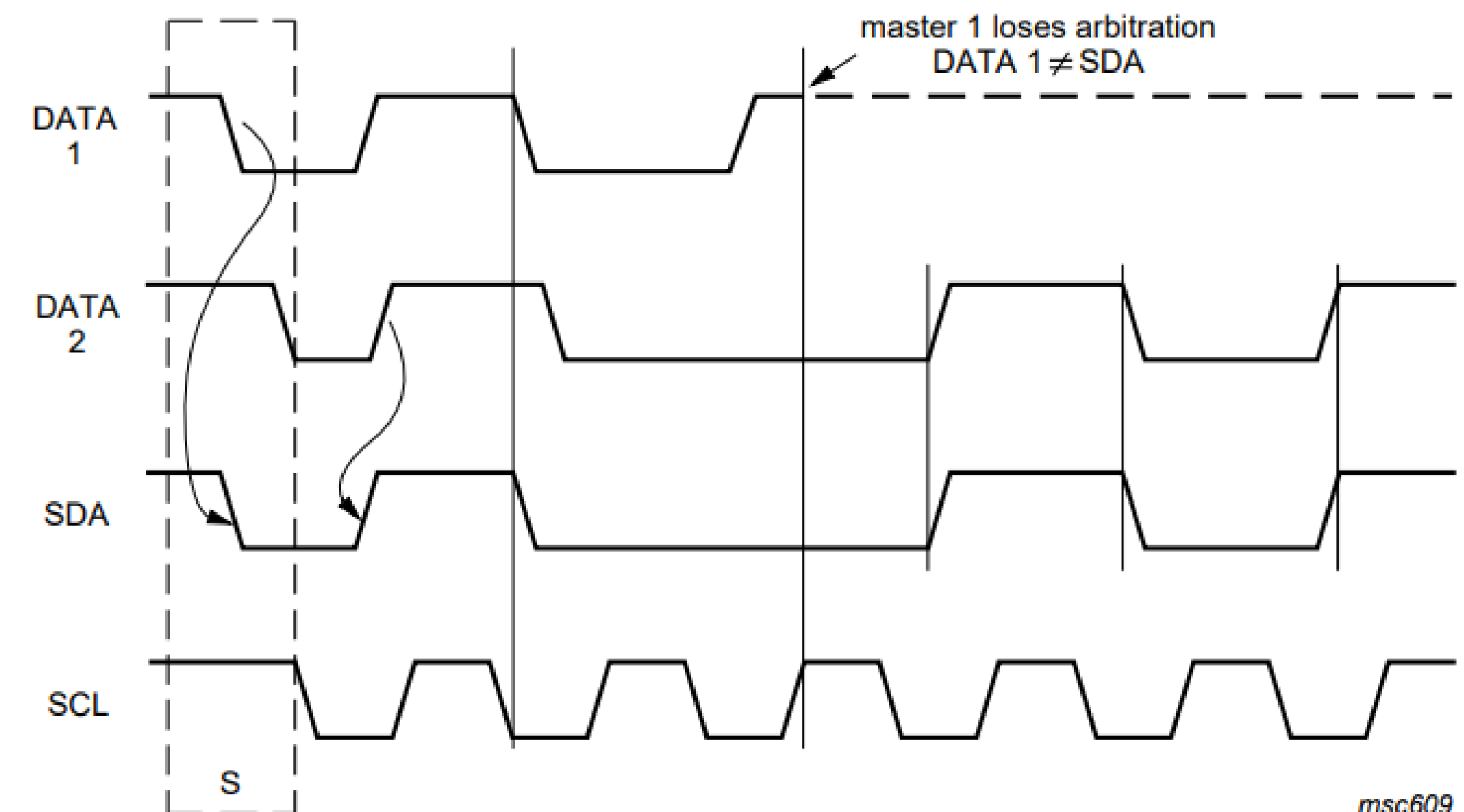
Clock Arbitration

- A master may start a transfer only if the bus is free.
- Two masters may generate a START condition within the minimum hold time of the START condition which results in a valid START condition on the bus.
- Arbitration is then required to determine which master will complete its transmission.

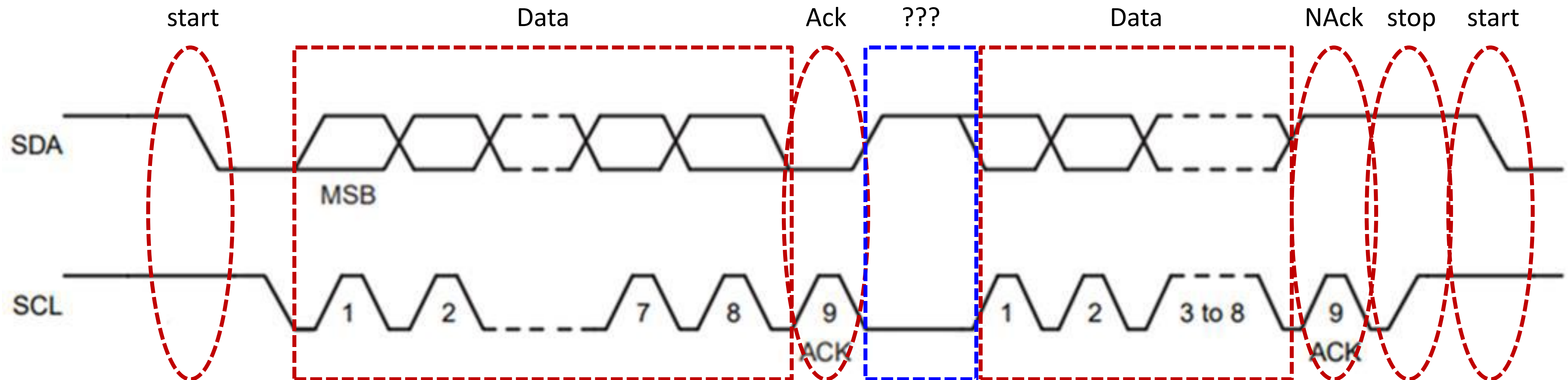


Clock Arbitration

- Arbitration proceeds bit by bit. During every bit, while SCL is HIGH, each master checks to see if the SDA level matches what it has sent.
- This process may take many bits. Two masters can actually complete an entire transaction without error, as long as the transmissions are identical.
- The first time a master tries to send a HIGH, but detects that the SDA level is LOW, the master knows that it has lost the arbitration and turns off its SDA output driver. The other master goes on to complete its transaction.



Situation Analysis



Clock Stretching

Clock Stretching

- Clock stretching allows a master or slave to hold the SCL line low to slow down the clock if it needs more time to process data.
- If a slave cannot receive or transmit another complete byte of data **until it has performed some other function** (e.g. servicing an internal interrupt) it can hold the clock line SCL LOW to force the master into a **wait state**.
- Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL.
- In a multi-master environment, a master may use clock stretching strategically to gain control during arbitration or to avoid conflicts.



ES Modeling & Design (2 Modules)

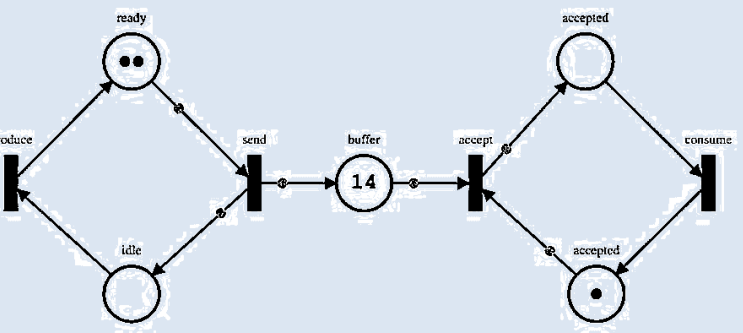
System Modeling



Design Considerations

Power management and optimization

Reliability and fault tolerance



The Real-time Embedded System

ES Hardware Components (4 Modules)

Microcontroller
Fundamentals



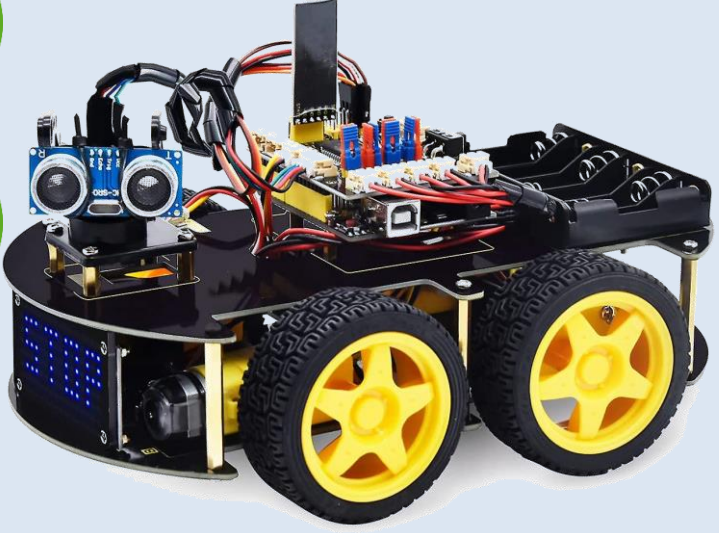
Embedded programming
languages



Embedded Hardware



Communication and
Networking



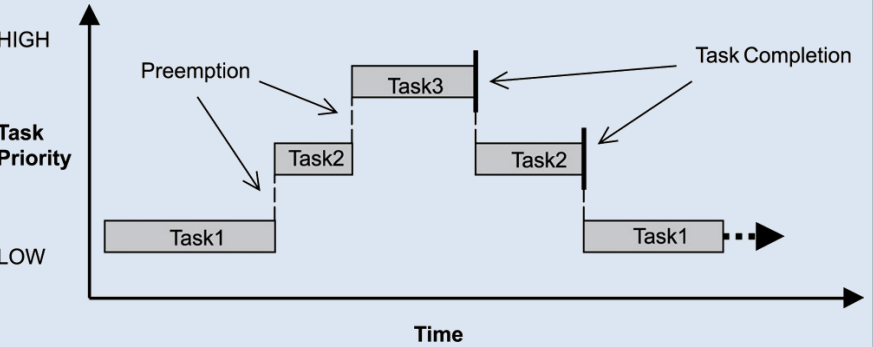
ES Software Components (1 Module)

Real-Time Systems

Multi-tasking

Scheduling

Resource Management



Embedded System Tools & Software Development (2 Modules)


Debugging techniques

Interrupts and exception
handling

Memory management

Let's go



For Further Inquiries, Please
 *send an email*

Catherine.elias@guc.edu.eg,
Catherine.elias@ieee.org

Thank you for your attention!

See you next time 😊