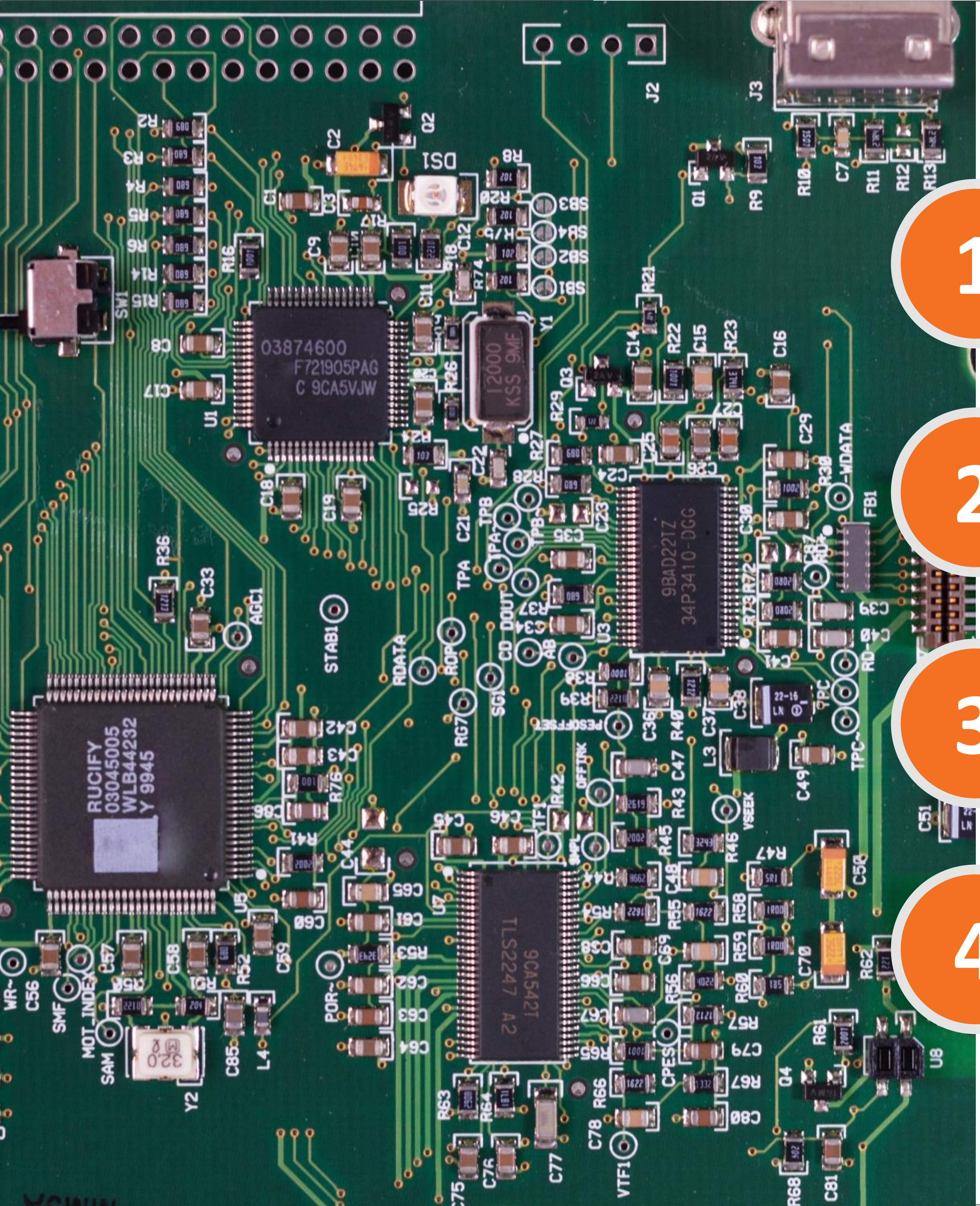
MCT-338: Embedded Systems-II

Lecture 4
Serial Communication Interfaces

Shujat Ali



1 SERIAL COMMUNICATION FUNDAMENTALS

2 UNIVERSAL ASYNCHRONOUS RECIEVER/TRANSMITTER (UART) INTERFACE

3 SERIAL PERIPHERAL INTERFACE (SPI)

4 INTER-INTEGRATED CIRCUIT (I2C) INTERFACE

SERIAL COMMUNICATION FUNDAMENTALS

Introduction to Serial Communication and it’s Classification

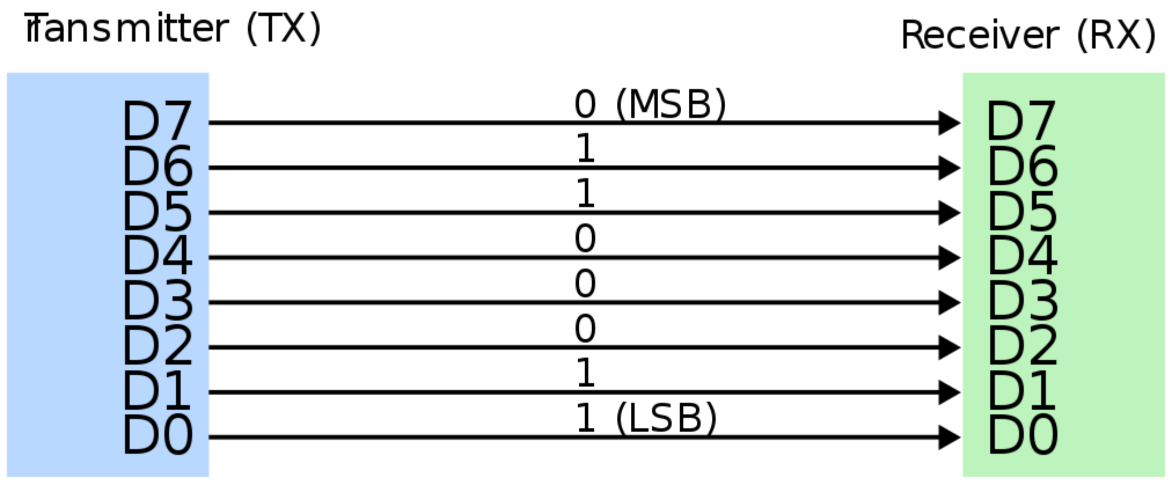
Overview

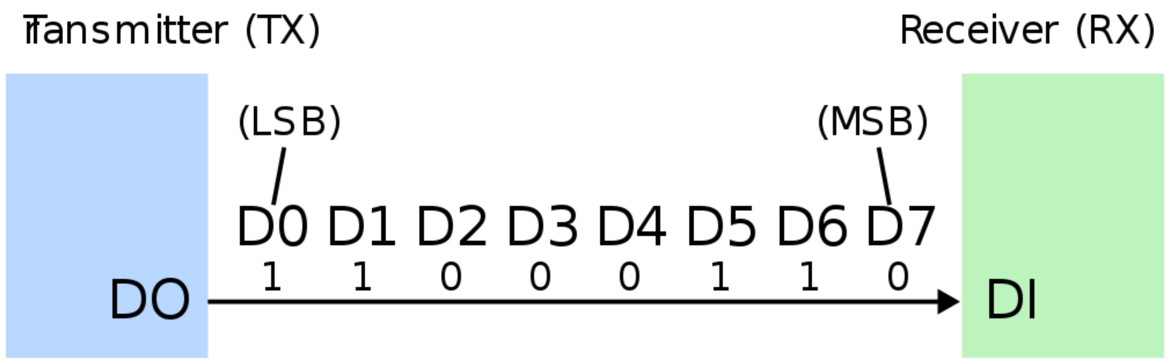
SERIAL COMMUNICATION FUNDAMENTALS

- Different serial communication protocols are widely used for data transfers among digital systems.
- Serial communication protocols can be
  1. **Synchronous**: Serial Peripheral Bus (**SPI**) and Inter-Integrated Circuit (**I2C**)
  2. **Asynchronous**: Universal Asynchronous Receive/Transmit (**UART**), Universal Serial Bus (**USB**), Ethernet, and Controller Area Network (**CAN**)
- Both types of these serial interfaces are integrated in microcontrollers, and we will discuss in detail UART and SPI interfaces in this lecture

Parallel vs. Serial Communication

SERIAL COMMUNICATION FUNDAMENTALS





Parallel Communication Interface

- Requires multiple parallel lines for and have faster data communication
- Normally used when two devices communicating are closer to each other (order of few meters)
- E.g., connecting memory, graphics display

Serial Communication Interface

- Requires single line for data transfer and have relatively slower data communication
- Used when two devices communicating are longer distance apart (order of hundreds of meters)
- Data transfer rate in serial communication is referred as **Bit Rate** or **Baud Rate**

https://en.wikipedia.org/wiki/Parallel_communication#/media/File:Serial_vs._parallel_transmission.svg

Types of Serial Communication

SERIAL COMMUNICATION FUNDAMENTALS

- There are many different types of serial communication interfaces that are in use today.
- List of selected set of serial communication interfaces available on microcontrollers:
  - **Universal Asynchronous Receiver/Transmitter** (**UART**) – simple and widely used serial communication protocols. RS232, RS422 and RS485 are example of UART based standards
  - **Serial Peripheral Interface** (**SPI**) or Synchronous Serial Interface (**SSI**) – widely used for short distance communication e.g., processor-to-processor, processor-to-devices (memories, displays, sensors, ADC, DAC, RF devices, etc.)
  - **Inter-Integrated Circuit** (**IIC** or **I2C**) **Bus** – widely used for short distance communication e.g., processor-to-processor, processor-to-devices (memories, displays, sensors, ADC, DAC, RF devices, etc.)
  - **Controller Area Network** (**CAN**) – being widely used in industry especially in automobiles industry
  - **Universal Serial Bus** (**USB**) – most used communication protocol
  - **Ethernet** – most used communication protocol for computer networks, e.g., internet

Types of Serial Communication

SERIAL COMMUNICATION FUNDAMENTALS

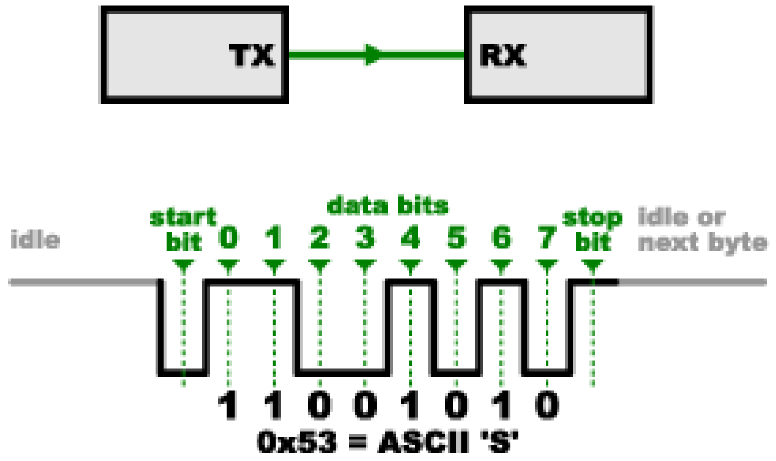
Every serial communication protocol can be classified using different attributes. Two of the most important attribute are

1. **Synchronization**: Synchronous or Asynchronous
2. **Duplexity**: Half-duplex or Full-duplex

Synchronization of Serial Interface

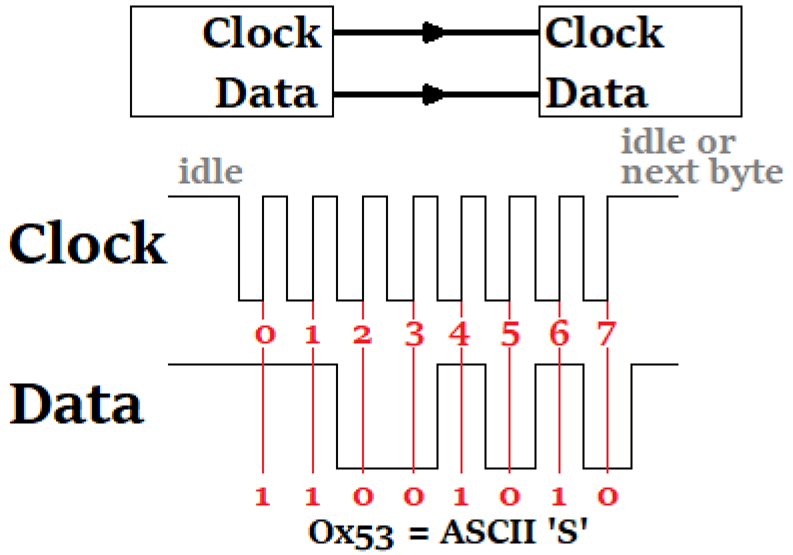
SERIAL COMMUNICATION FUNDAMENTALS

Asynchronous Serial Interface



- Only data is transmitted at a mutually agreed bit rate
- The receiver generates a local clock signal to reliably receive the transmitted data bits
- Transmitter must inform the receiver, if it want to switch to a different baud rate, before starting data transmission at new rate

Synchronous Serial Interface

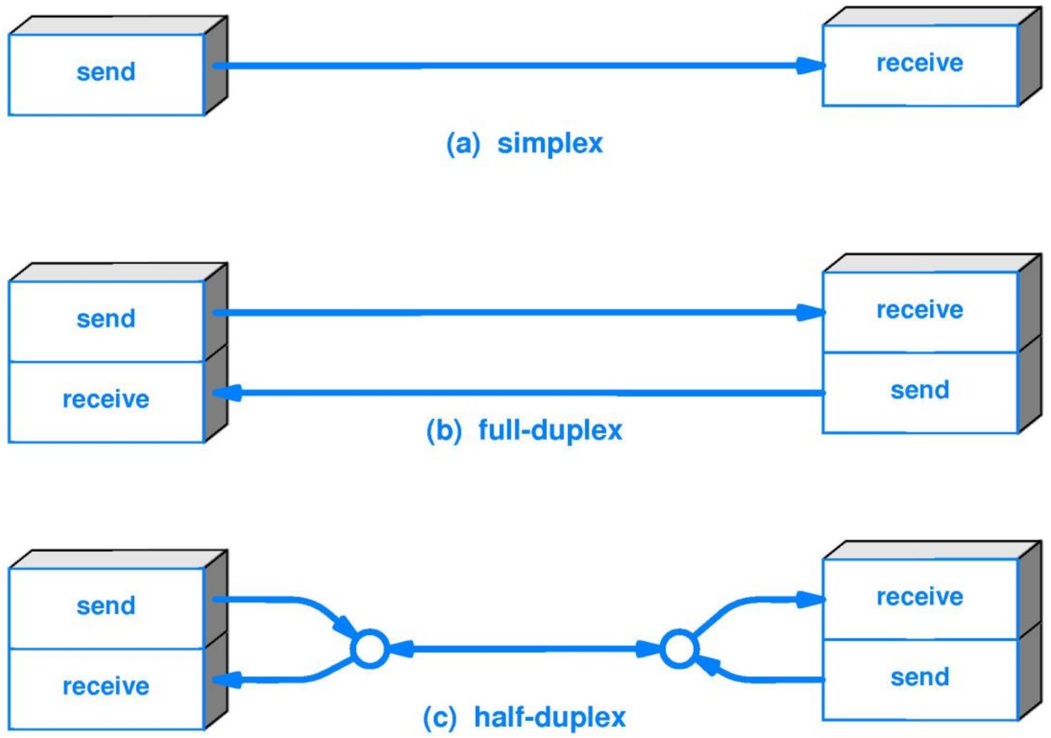


- Both data and clock signals are transmitted simultaneously
- Sender can change baud rate without informing the receiver
- I2C and Serial Peripheral Interface (SPI)

Duplexity of Serial Interface

SERIAL COMMUNICATION FUNDAMENTALS

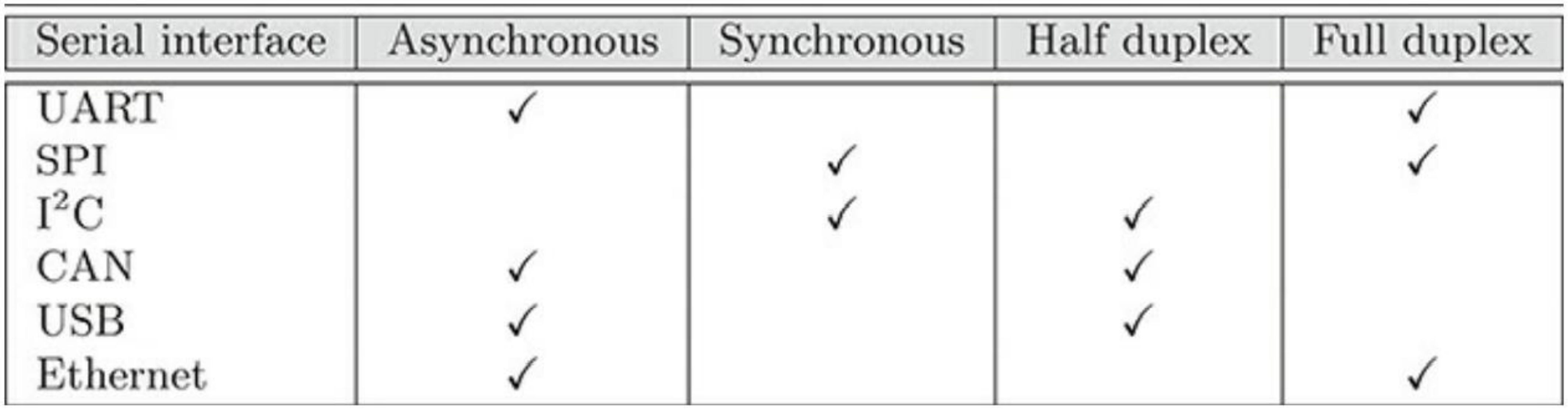
- If data can only be transmitted in one direction, then this interface is termed as **Simplex** interface.
- A **Duplex** interface implies that data can be transmitted in either direction between the communicating devices. Based on it, there are two different types:
- **Half Duplex**: data can only be transmitted in one direction at a time, e.g., I2C and CAN bus
- **Full Duplex**: data can be transmitted in both directions simultaneously, e.g., UART and SPI



Classification of Serial Communication

SERIAL COMMUNICATION FUNDAMENTALS

- The classifications of different serial communication interfaces, based on synchronization and duplexity



Serial interface	Asynchronous	Synchronous	Half duplex	Full duplex
UART	✓			✓
SPI		✓		✓
I^2C		✓	✓	
CAN	✓		✓	
USB	✓		✓	
Ethernet	✓			✓

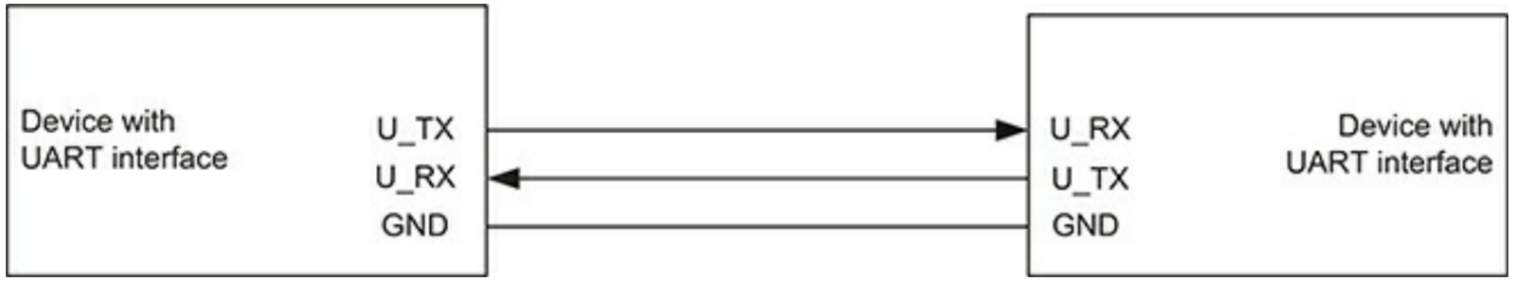
UNIVERSAL ASYNCHRONOUS RECIEVER/TRANSMITTER (UART) INTERFACE

Asynchronous & Full-duplex Serial Communication

Connections

UART INTERFACE

- Minimum connection requirements for UART interface



- For reliable data transfer through UART serial communication, **hand-shaking** or **flow control** is essential
  - Consider the scenario, where during data transmission, the receiving device runs out of buffer space, and it becomes essential to inform the transmitter to stop further data transmission
- This additional control signaling between transmitter and receiver is called **flow control** and it can be implemented by using
  1. **Hardware**: Dedicated hardware lines labeled as request to send (RTS) and clear to send (CTS) are additionally used
  2. **Software**: A communication protocol is implemented as a part of the software without any extra hardware

Communication Protocol

UART INTERFACE

Baud Rate

- Both transmitter and receiver should be configured with same predefined baud rate for reliable data transmission.

$$\text{UART Baud} = \frac{f_{\text{uart}}}{k \times \text{baud\_divisor}}$$

  - ***f*****uart** = frequency of the clock fed to the UART module
  - **baud_divisor** = baud rate divisor field defined in one of the UART configuration registers
  - ***k*** = clock divisor constant and can take different values depending on selected hardware platform

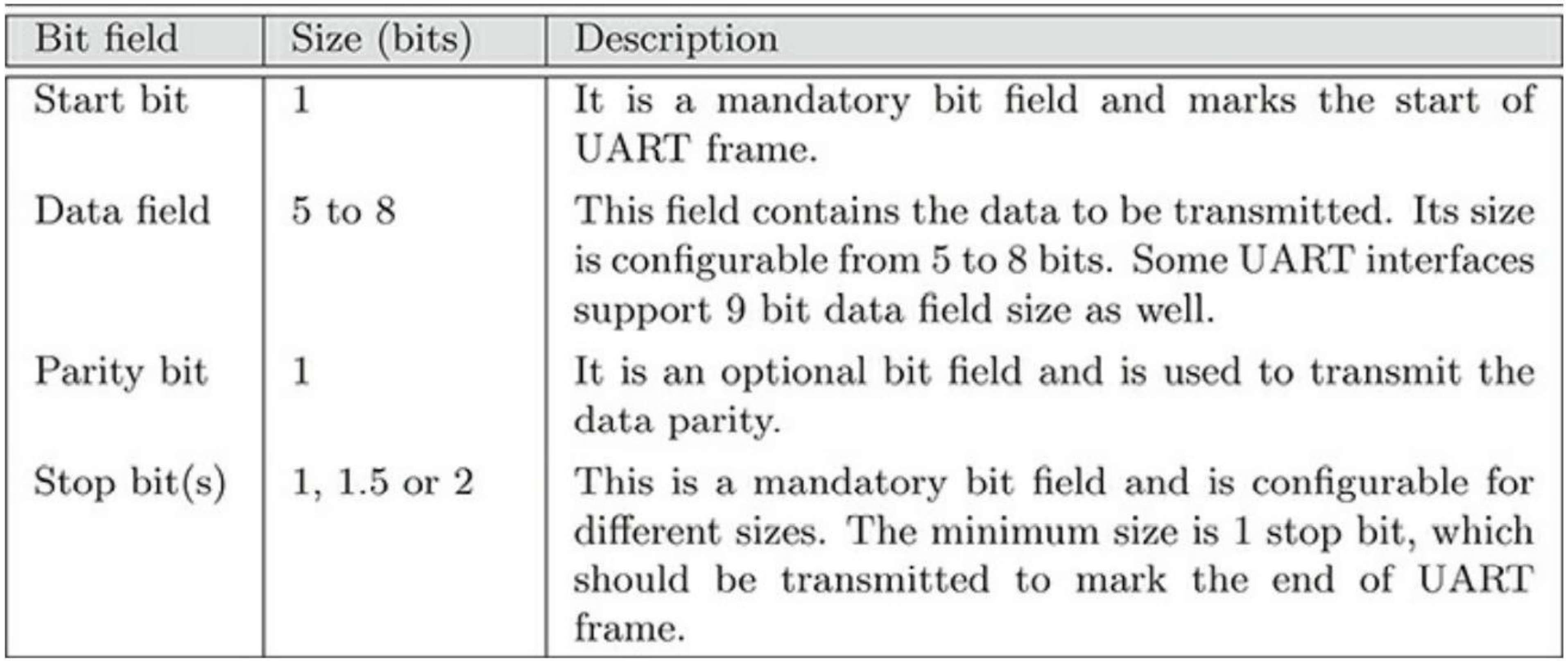
- Based on the available configuration parameters, an arbitrary baud rate can be used for data transfers. However, there are some standard baud rates that are used by different applications

Communication Protocol

UART INTERFACE

Frame Format

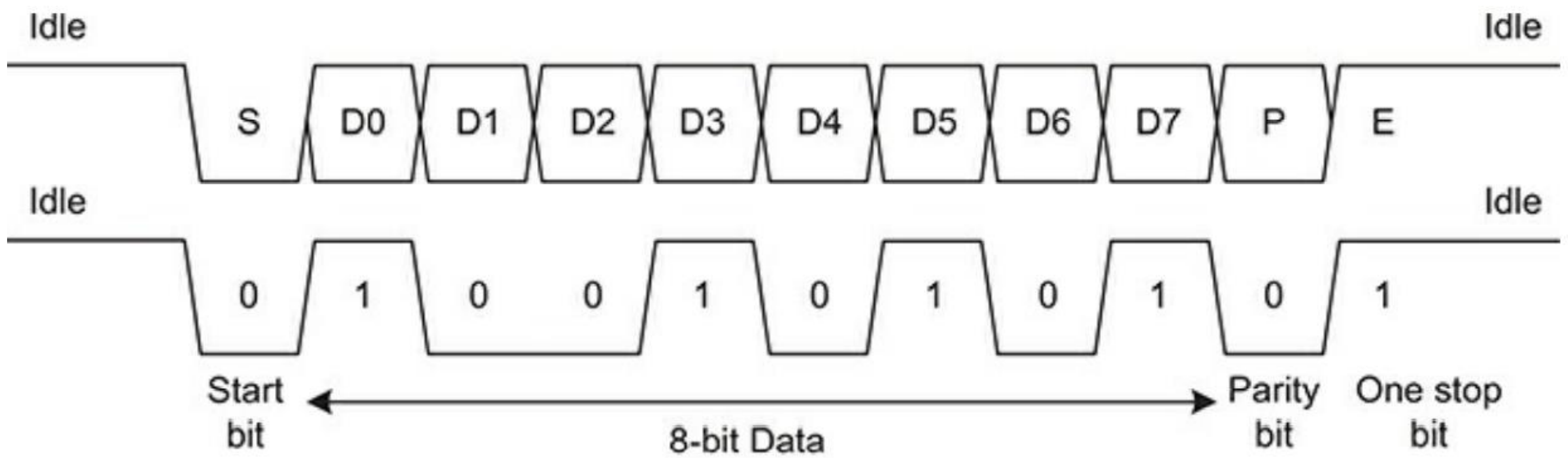
- A data byte is transmitted by the UART module, by encapsulating it in a **UART frame**.



Bit field	Size (bits)	Description
Start bit	1	It is a mandatory bit field and marks the start of UART frame.
Data field	5 to 8	This field contains the data to be transmitted. Its size is configurable from 5 to 8 bits. Some UART interfaces support 9 bit data field size as well.
Parity bit	1	It is an optional bit field and is used to transmit the data parity.
Stop bit(s)	1, 1.5 or 2	This is a mandatory bit field and is configurable for different sizes. The minimum size is 1 stop bit, which should be transmitted to mark the end of UART frame.

Communication Protocol

UART INTERFACE



Frame Format

- The UART module is idle before the beginning of transmission, i.e., UT_TX pin is kept at logic level high
- The start of UART frame transmission is marked by a logic high to low transition for one bit duration (Start bit) on U_TX pin
- Next, the first data bit (corresponding to LSB of data) is transmitted followed by bit-by-bit transmission of entire data field.
- After the transmission of last data bit, the optional parity bit, if configured, is transmitted
- Finally, one or two stop bits are transmitted to mark the end of UART frame transmission
- The transmission of next UART frame can start immediately after the transmission of stop bit(s) corresponding to previous UART frame.

Communication Protocol

UART INTERFACE

Receiver Synchronization

- Since no clock signal is transmitted by the UART transmitter, the receiver needs to synchronize itself to the incoming data locally.
- For that purpose, the UART receiver uses a clock signal that is 16 times the UART baud rate.
- When the transmission of a new UART frame is initiated by Start bit (i.e., a high to low transition), the UART receiver resets a local counter on the reception of this falling edge.
- When the counter count equals 8, it corresponds to the mid point of the Start bit. The midpoints of subsequent bits in the UART frame are expected to occur every 16 cycles thereafter.
- The higher clock signal used by the receiver increases correct bit detection capability by improving the receiver synchronization.

Communication Protocol

UART INTERFACE

Error Types

There are four different types of errors that can occur in case of UART communication.

1. **Parity Error**: When the received parity bit does not match with the parity calculated from the received data, a parity error is generated.
2. **Framing Error**: When a Start bit is received but the receiver does not get the corresponding stop bit then a framing error is generated.
3. **Overrun Error**: When the UART receive buffer is full and new data is received, then the newly received data overwrites the previous data. This situation leads to overrun error.
4. **Break Error**: When the receiver detects a break condition, it results in break error. A break condition is detected when UART RX pin is held low for more than one UART frame transmission time.

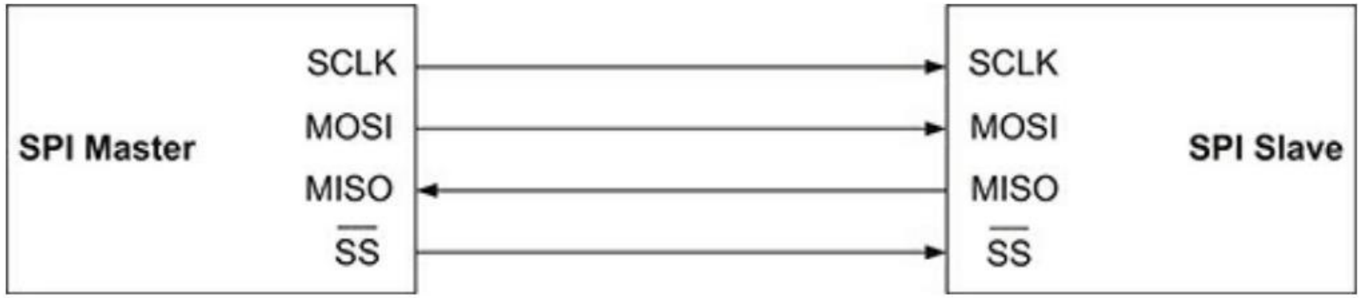
SERIAL PERIPHERAL INTERFACE (SPI)

Synchronous & Full-duplex Serial Communication

Introduction

SERIAL PERIPHERAL INTERFACE

- Serial Peripheral Interface (SPI) is originally developed by Motorola
- The SPI serial bus implements full duplex communication using master-slave paradigm

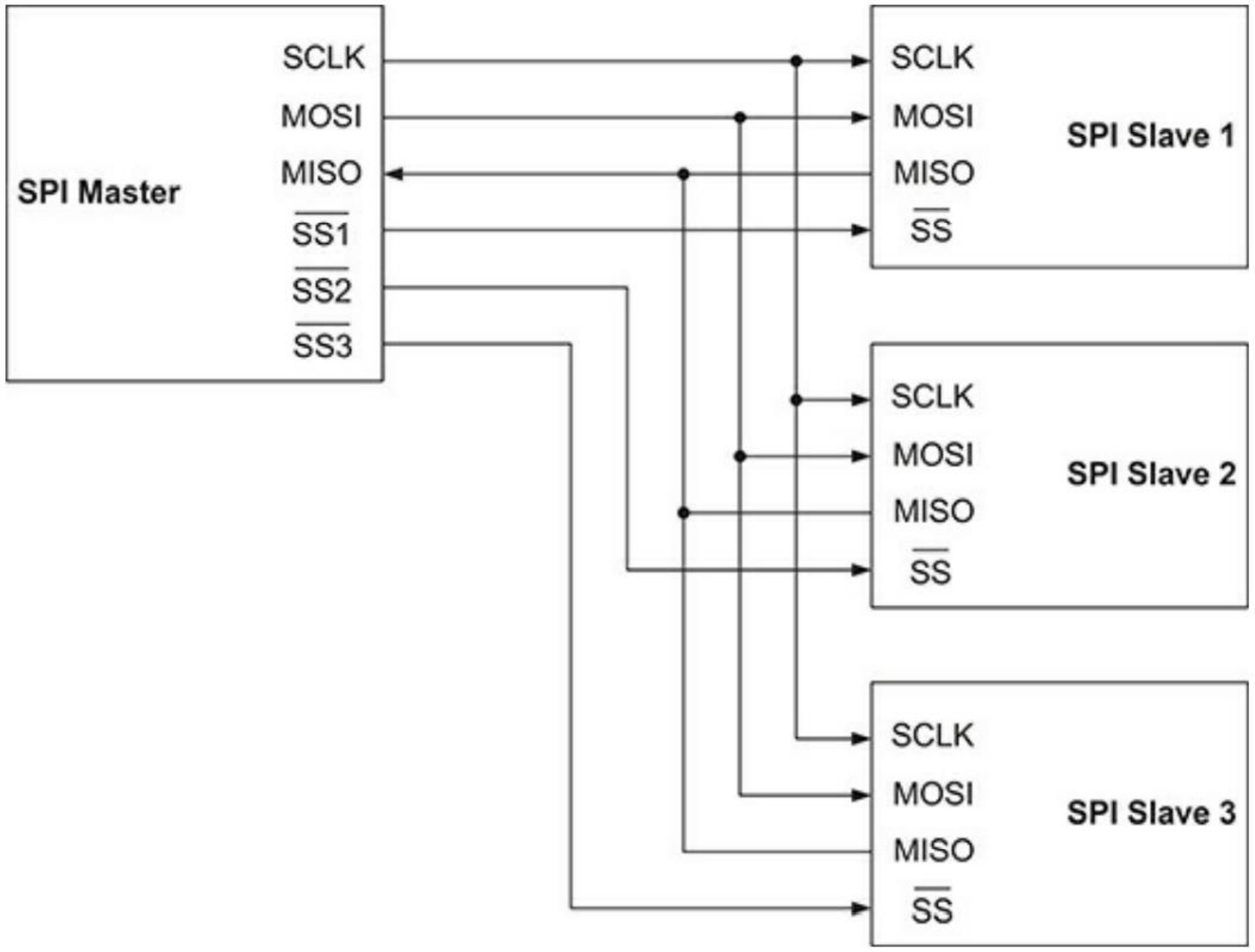


- **SCLK**: This is a serial clock signal that is transmitted by the bus master to the slave device(s)
- **SS**: This signal is used to select or enable a slave device for SPI communication. This signal is active low i.e., a logic low on this line enables the device for SPI communication.
- **MOSI**: The data from a master to a slave device, is transmitted using Master-Out Slave-In (MOSI) data line
- **MISO**: The data to a master device from a slave, is transmitted using Master-In Slave-Out (MISO) data line.

Connections

SERIAL PERIPHERAL INTERFACE

- In case of multiple devices, a separate slave select (SS) signal is used by the bus master, to enable the desired slave device.
- If n number of slave devices are connected to the bus master, then the number of master device lines required for SPI bus connectivity are n + 3.
- There are no standard specifications for SPI protocol and as a result different variants of this protocol exist.
- For instance, there is Freescale SPI, TI SPI and another closely related communication protocol, named as Microwire.



Modes of Operation

SERIAL PERIPHERAL INTERFACE

SPI modes of operation are defined by using two SCLK signal based parameters

1. **Serial Clock-polarity** (*SPOL*): sets the polarity of the clock signal during idle state ($\overline{SS}$ = High)
2. **Serial Clock-phase** (*SPHA*): defines the edge that is used to sample and shift the data

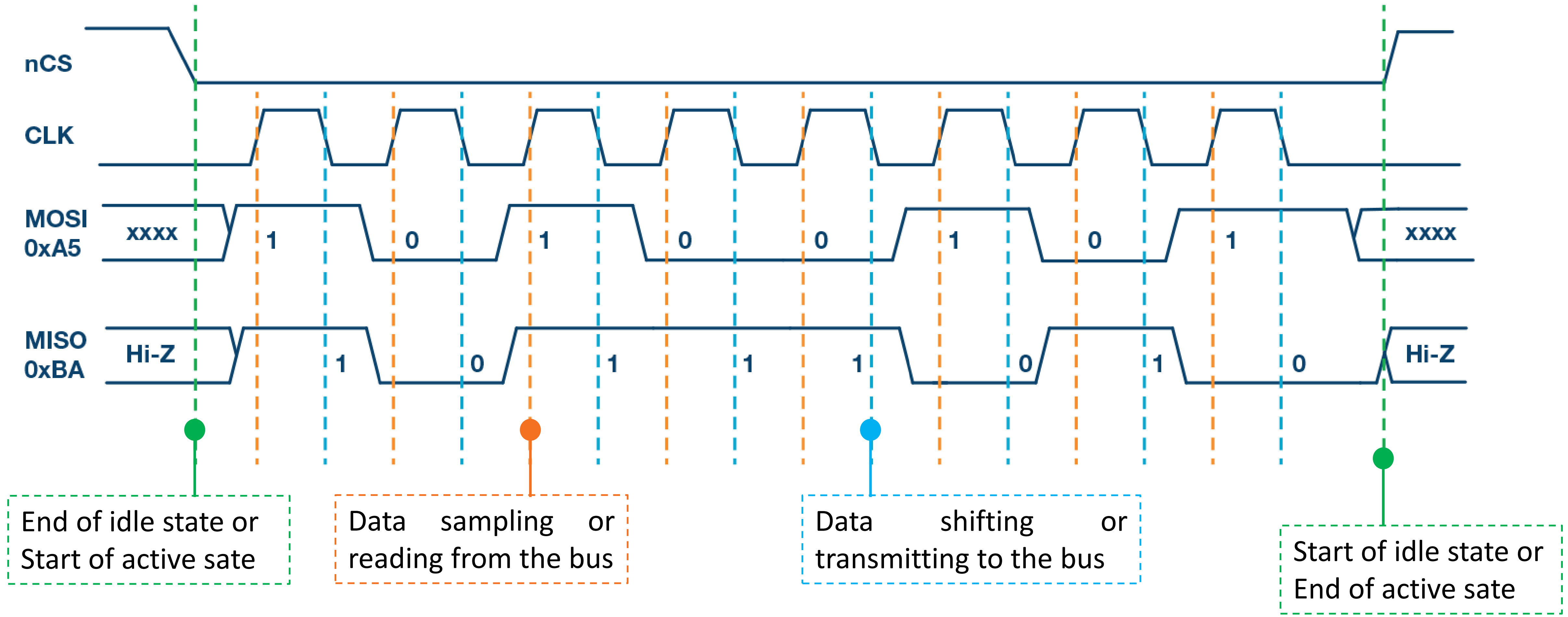
SPI protocol supports four different modes of communication, namely

- **SPI Mode 0**: *SPOL* bit = 0, *SPHA* bit = 0 (CLK Low, Data sampled on rising edge and shifted out on falling edge)
- **SPI Mode 1**: *SPOL* bit = 0, *SPHA* bit = 1 (CLK Low, Data sampled on falling edge and shifted out on rising edge)
- **SPI Mode 2**: *SPOL* bit = 1, *SPHA* bit = 0 (CLK High, Data sampled on falling edge and shifted out on rising edge)
- **SPI Mode 3**: *SPOL* bit = 1, *SPHA* bit = 1 (CLK High, Data sampled on rising edge and shifted out on falling edge)

SPI Mode 0

SERIAL PERIPHERAL INTERFACE

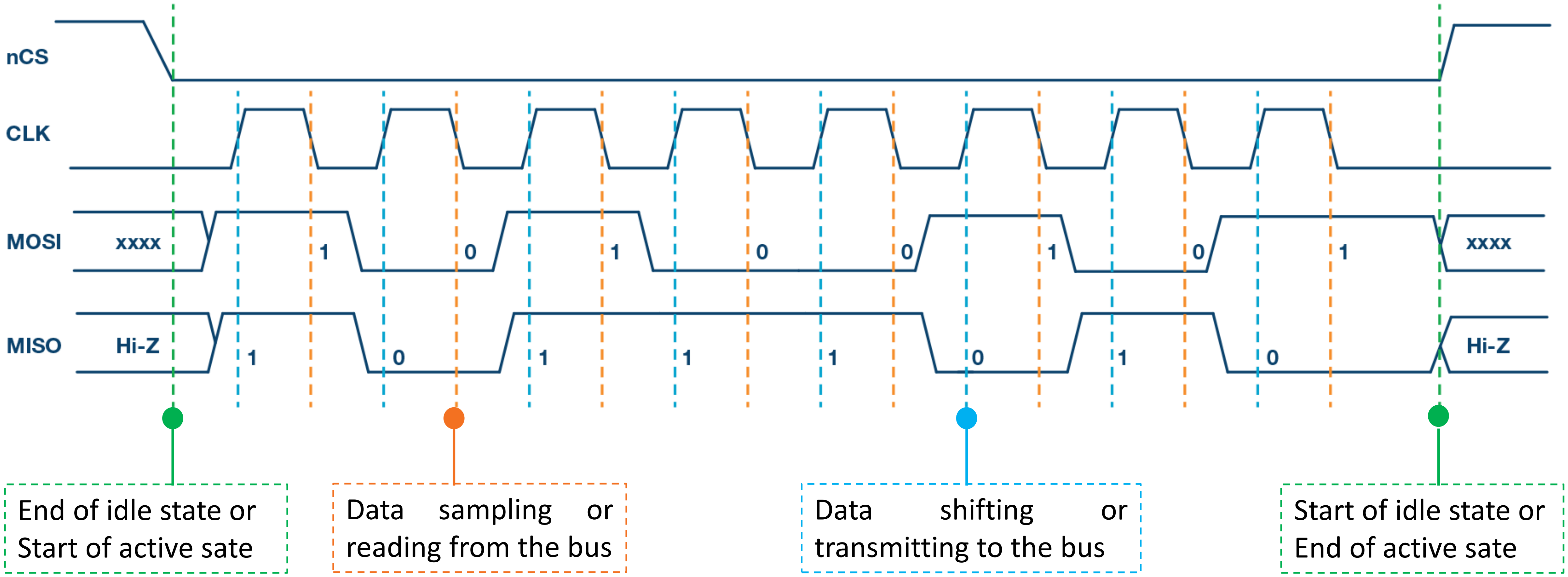
- *SPOL* bit = 0, *SPHA* bit = 0 (CLK Low, Data sampled on rising edge and shifted out on falling edge)


nCS
CLK
MOSI
0xA5
xxxx
1
0
1
0
0
1
0
1
xxxx
MISO
0xBA
Hi-Z
1
0
1
1
1
0
1
0
Hi-Z
End of idle state or Start of active sate
Data sampling or reading from the bus
Data shifting or transmitting to the bus
Start of idle state or End of active sate

SPI Mode 1

SERIAL PERIPHERAL INTERFACE

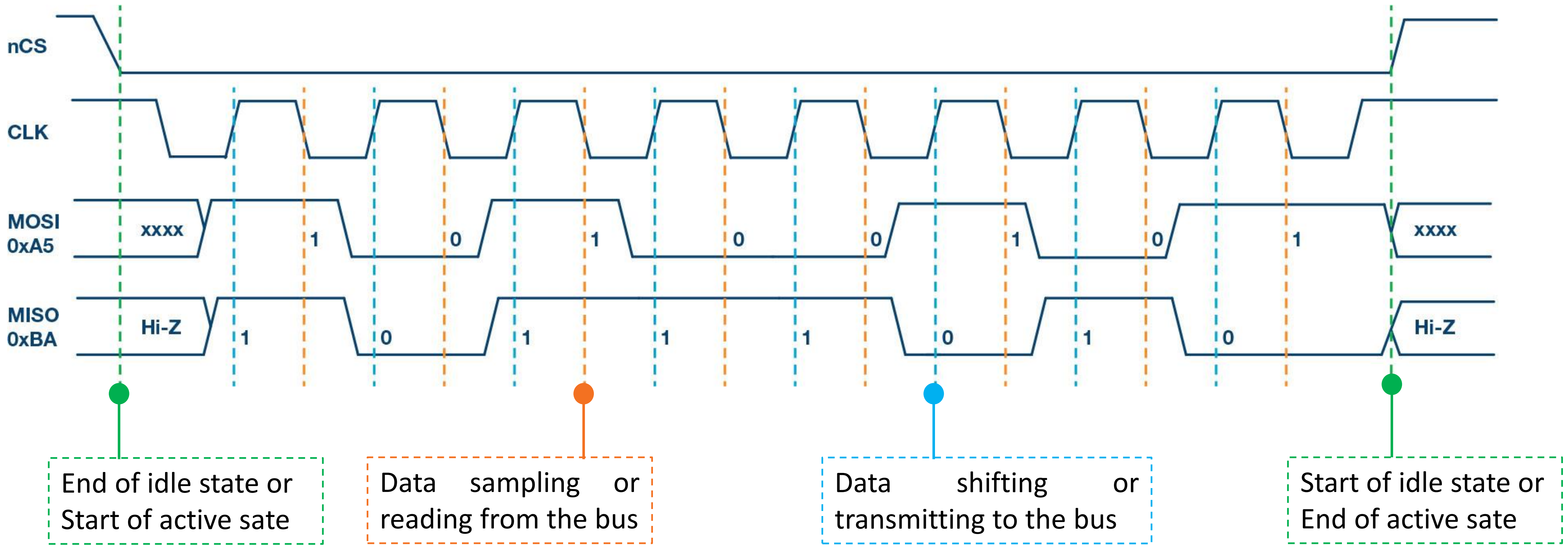
- *SPOL* bit = 0, *SPHA* bit = 1 (CLK Low, Data sampled on falling edge and shifted out on rising edge)


nCS
CLK
MOSI
xxxx
1
0
1
0
0
1
0
1
xxxx
MISO
Hi-Z
1
0
1
1
1
0
1
0
Hi-Z
End of idle state or Start of active sate
Data sampling or reading from the bus
Data shifting or transmitting to the bus
Start of idle state or End of active sate

SPI Mode 2

SERIAL PERIPHERAL INTERFACE

- *SPOL* bit = 1, *SPHA* bit = 0 (CLK High, Data sampled on falling edge and shifted out on rising edge)



End of idle state or Start of active sate

Data sampling or reading from the bus

Data shifting or transmitting to the bus

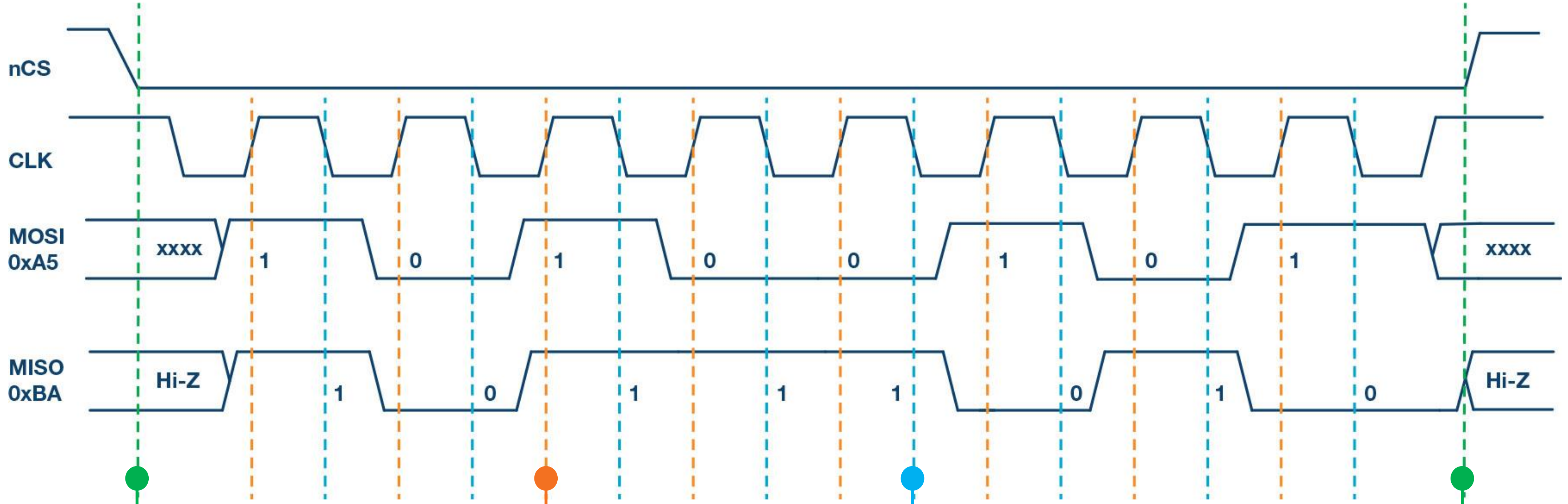
Start of idle state or End of active sate

https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html

SPI Mode 3

SERIAL PERIPHERAL INTERFACE

- *SPOL* bit = 1, *SPHA* bit = 1 (CLK High, Data sampled on rising edge and shifted out on falling edge)



End of idle state or Start of active sate

Data sampling or reading from the bus

Data shifting or transmitting to the bus

Start of idle state or End of active sate

Limitations

SERIAL PERIPHERAL INTERFACE

- For proper communication between a master and a slave device, **same parameter values** (for SCLK frequency, SPHA and SPOL) should be configured by both the devices
- Since SPI protocol **does not have standard specifications**, which leads to certain limitations, e.g., there is neither an acknowledgment mechanism for data reception confirmation, nor any flow control
- To integrate these features, a higher layer protocol is required
- In addition, the **maximum data rate** for SPI interface is not defined and it mostly depends on the I/O capabilities of the microcontroller used for SPI communication

INTER-INTEGRATED CIRCUIT (I2C) INTERFACE

Synchronous & Half-duplex Serial Communication

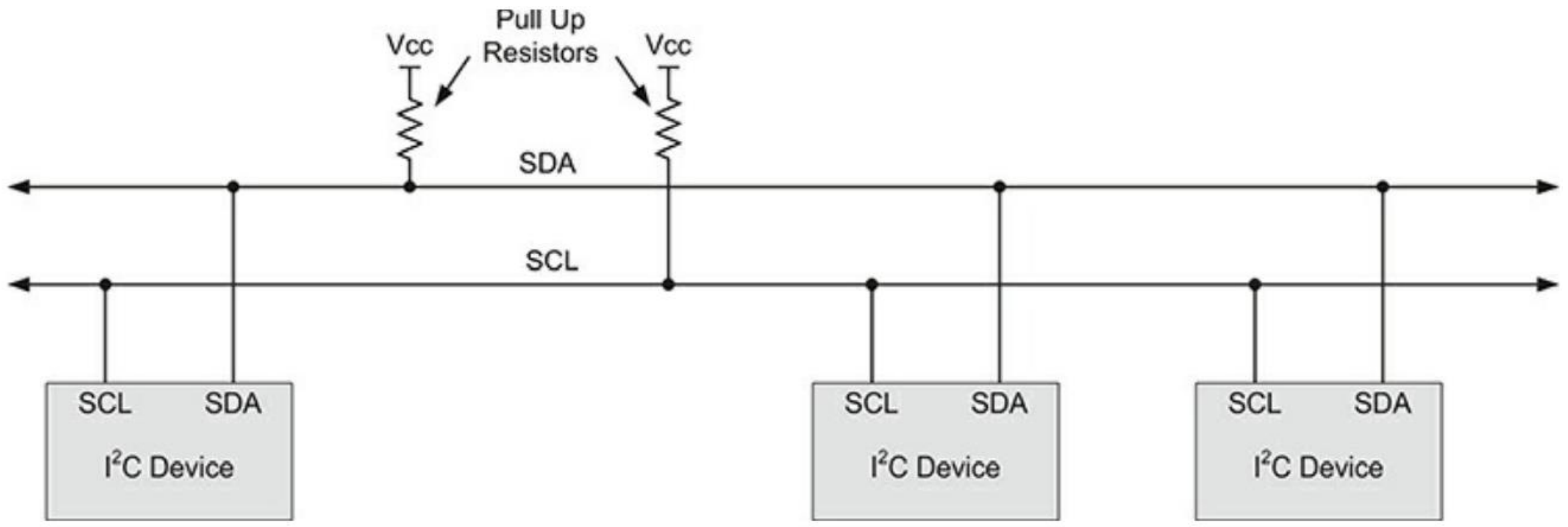
Introduction

I2C INTERFACE

- The Inter-Integrated Circuit, IIC or I2C is developed by Philips
- Multiple devices can be connected to I2C bus using only two wires.
- The I2C protocol specifications define multiple bus speeds, including
  - standard mode at 100 kbps,
  - fast mode at 400 kbps,
  - fast plus mode at 1 Mbps
  - high-speed mode at 3.33 Mbps.
- Each communicating device can be in one of the two possible modes, the master mode or slave mode.
- In addition, a device can switch between master and slave modes, making the I2C bus truly a multi-master serial bus interface.
- A master initiates communication with a slave device. The data can be either requested from or sent to the slave device.

Bus Protocol

I2C INTERFACE



- I2C is a multi-master serial bus protocol requiring two signal lines: **serial clock** (SCL) & **serial data** (SDA).
- Each device connected to an I2C bus is expected to have a unique address assigned to it.
- This address is used by the master device to communicate with a specific slave device.

Bus Protocol

I2C INTERFACE

The key attributes associated with I2C bus protocol are listed below.

- Each slave device has a 7-bit unique address assigned to it.
- The two signal lines, SCL and SDA, are bidirectional.
- Data is transmitted as a sequence of 8-bit bytes.
- In addition to data transmission, following control signals are also shared through the same pair of lines
    - communication start/stop
    - data send/receive control as well as its acknowledgment
    - slave device address communication

Bus Protocol

I2C INTERFACE

I2C Signaling Sequence for Data Transfer

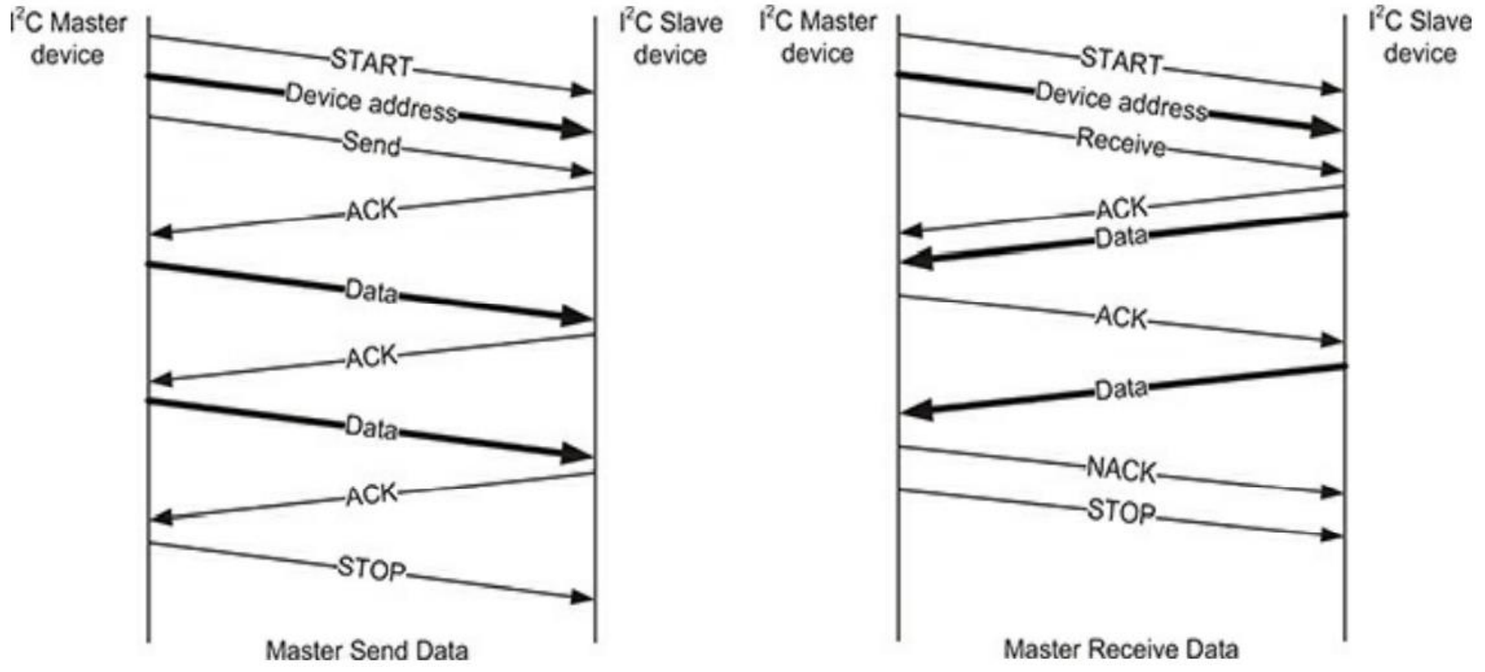
- A master device initiates data transfer on I2C bus by generating a START condition
- After the START condition, the master device transmits the ADDRESS of intended slave device followed by the Receive/Send (R/S) control signal.
    - The Send/Receive control signal informs the slave device whether it should expect incoming data, or it should respond with data.
- After receiving the slave address, all the devices will compare the slave address with their own address and slave device with matched address will respond with an acknowledge (ACK) signal
- On the other hand, those devices for which the slave address does not match, they continue waiting till the bus become idle after issuance of STOP condition.

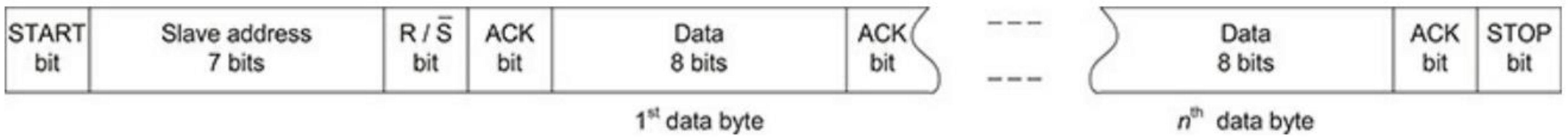
Bus Protocol

I2C INTERFACE

I2C Signaling Sequence for Data Transfer

- Each data transfer requires a sequence of nine bits involving 8 bits of data and 1 ACK bit
- After the START condition, the first byte is always transmitted by the master device.
    - The most significant 7 bits of this byte are the slave address, while the LSB of this byte is the R/$\bar{S}$ signaling bit



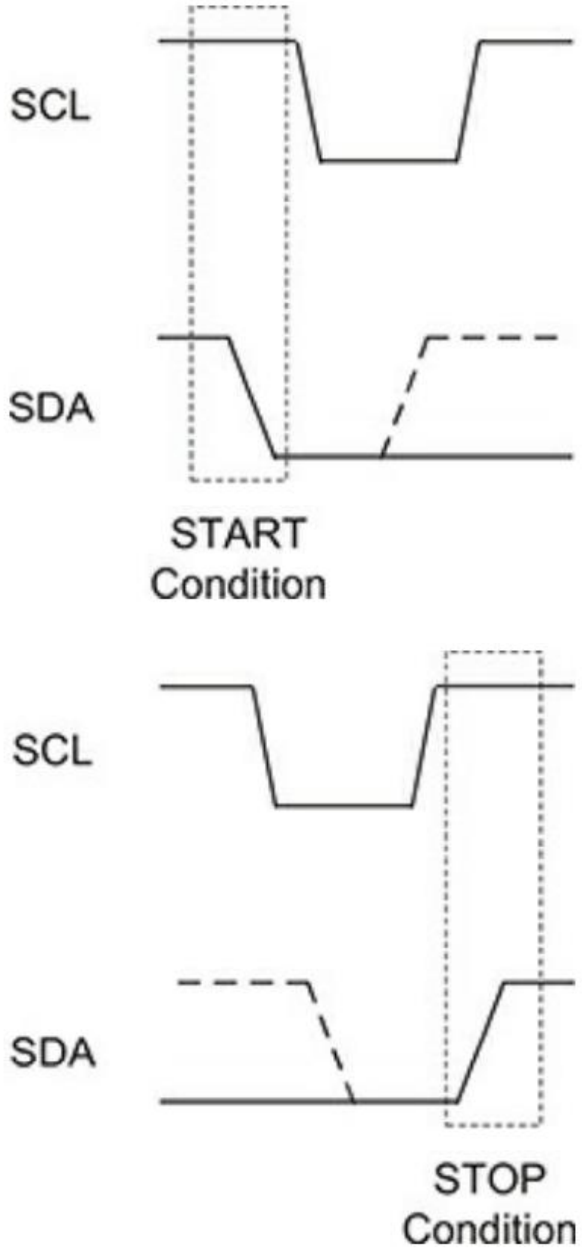


Bus Protocol

I2C INTERFACE

START STOP Conditions

- The START and STOP conditions are generated uniquely for proper differentiation from other possible transitions on the I2C bus.
- To generate a START condition, a high to low transition is generated on SDA line, during the positive half cycle of SCL line.
- Similarly, a STOP condition is generated by low to high transition on SDA line, during the positive half cycle of SCL line.

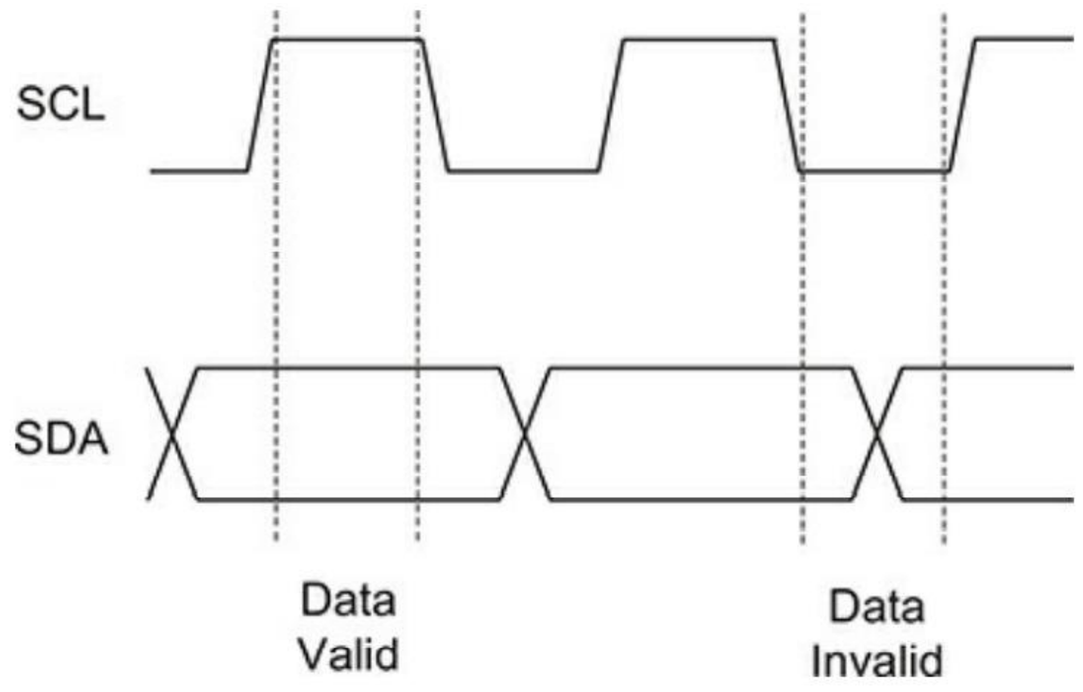


Bus Protocol

I2C INTERFACE

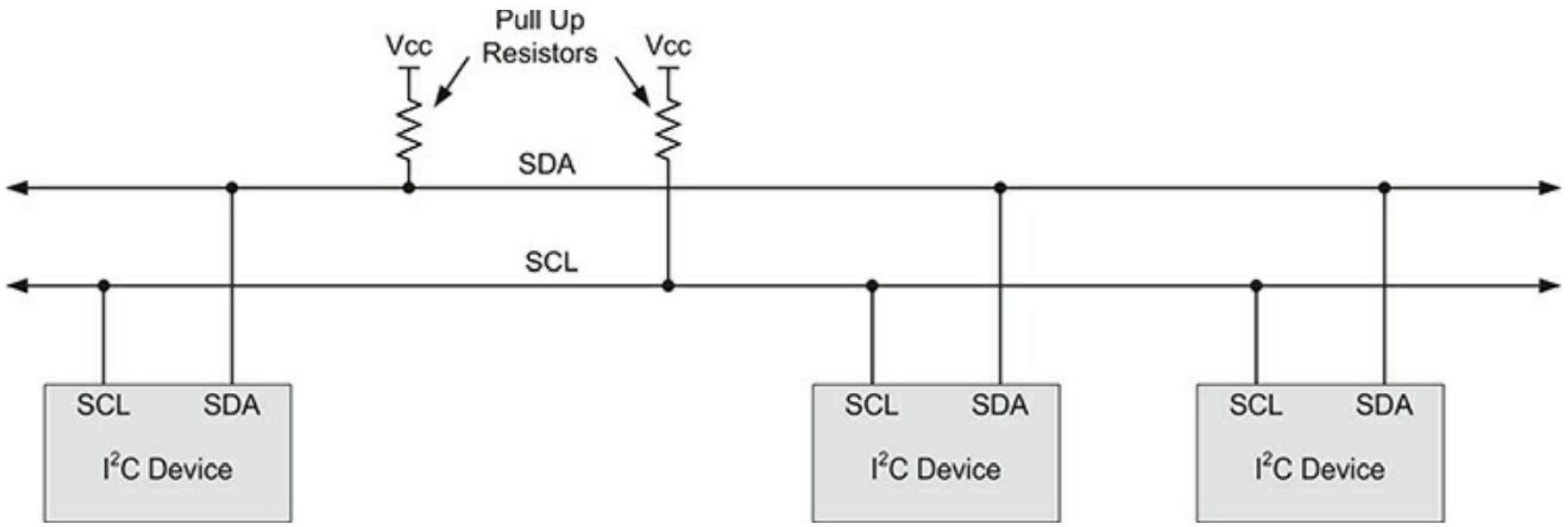
Data Validity

- When the I2C bus is idle, both SCL and SDA lines are in logic high state.
- According to I2C bus specifications, data changes on SDA line can only happen during the low half cycle of SCL signal.
- Any data changes on the SDA line during the positive half cycle of SCL signal are invalid
- For proper data transfer it is required that data remains stable on SDA line, while the SCL line is in high state



Bus Connectivity

I2C INTERFACE



- The SCL and SDA lines are configured as open drain with pull-up resistors installed
- The logic low and high levels are generated physically by pulling the line to ground or releasing the line.
- Effectively, a device connected to an I2C bus only drives logic low levels.
- All the devices that are connected to I2C bus, they listen to it continuously.
  - A master device aiming to initiate transmission on I2C bus, when detects a START condition, it will wait till a STOP condition is detected, before attempting bus access again.
  - Similarly, the slave devices on I2C bus will match the device address received after the START condition to find out whether they are the intended devices for data transfer.
  - A slave device that is not addressed, waits for the STOP condition before it listens to the bus again.

THANK YOU

Any Questions???