Communication Protocols in Embedded Systems

e-Yantra Team

Embedded Real-Time Systems (ERTS) Lab
Indian Institute of Technology, Bombay

IIT Bombay
April 9, 2023

Serial Vs Parallel

Serial Vs Parallel

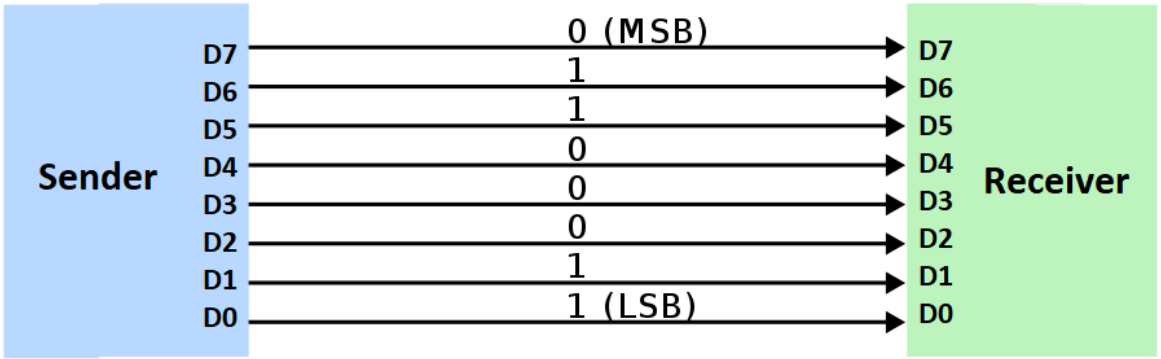


Figure: Parallel Communication

Serial Vs Parallel

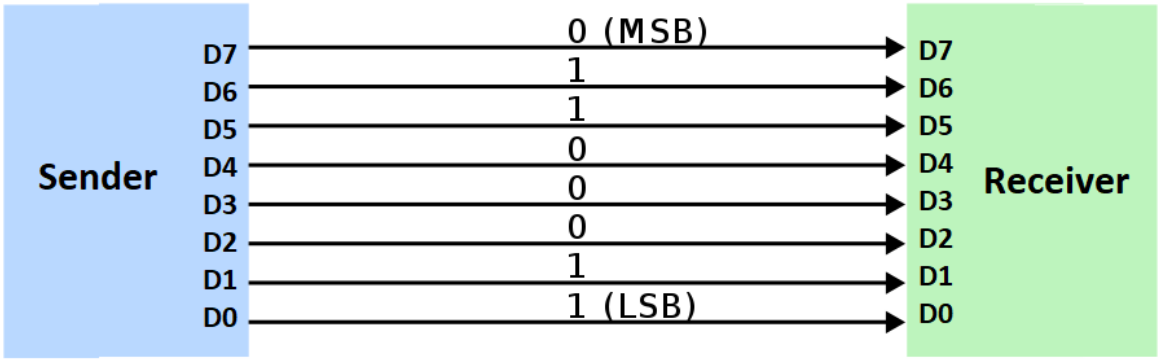


Figure: Parallel Communication

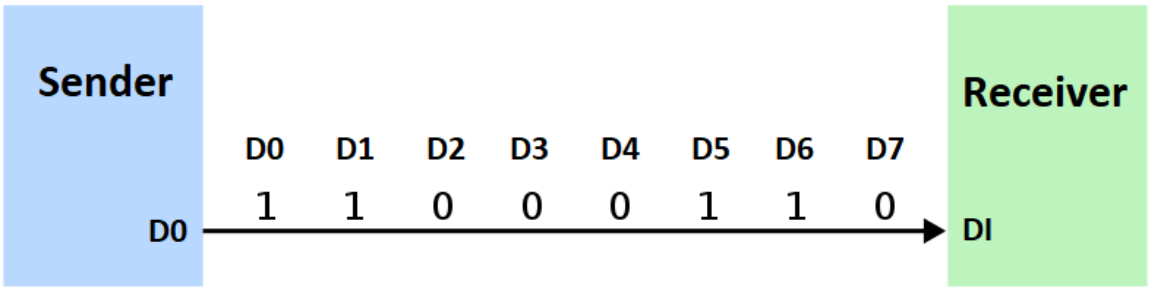


Figure: Serial Communication

Serial Vs Parallel

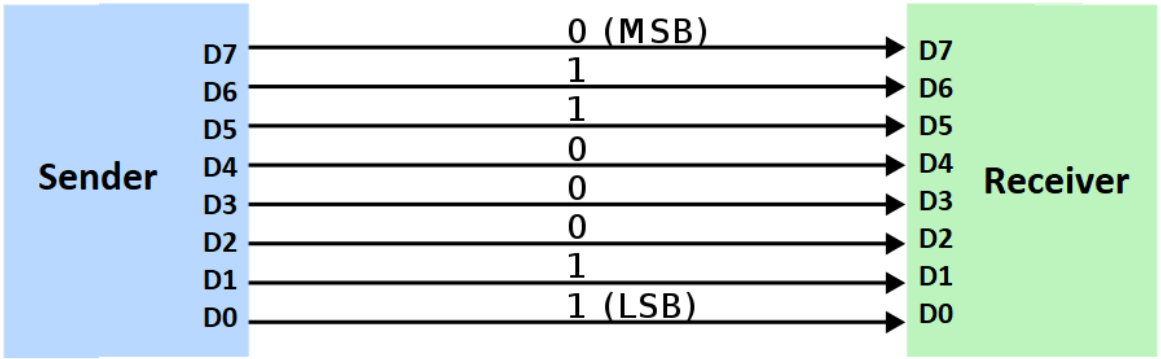


Figure: Parallel Communication

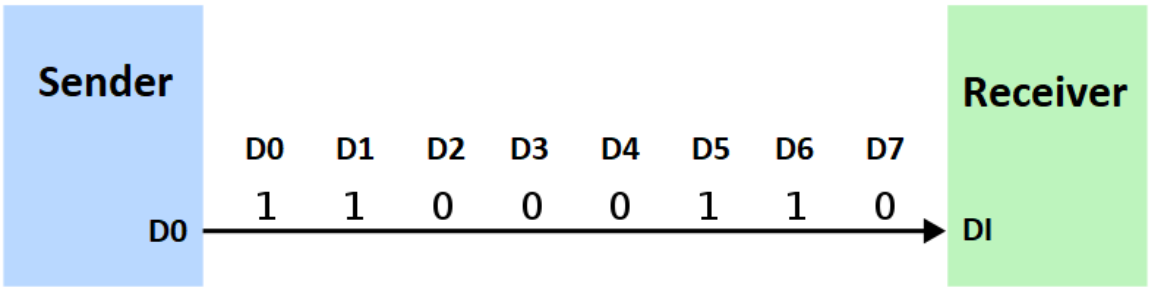
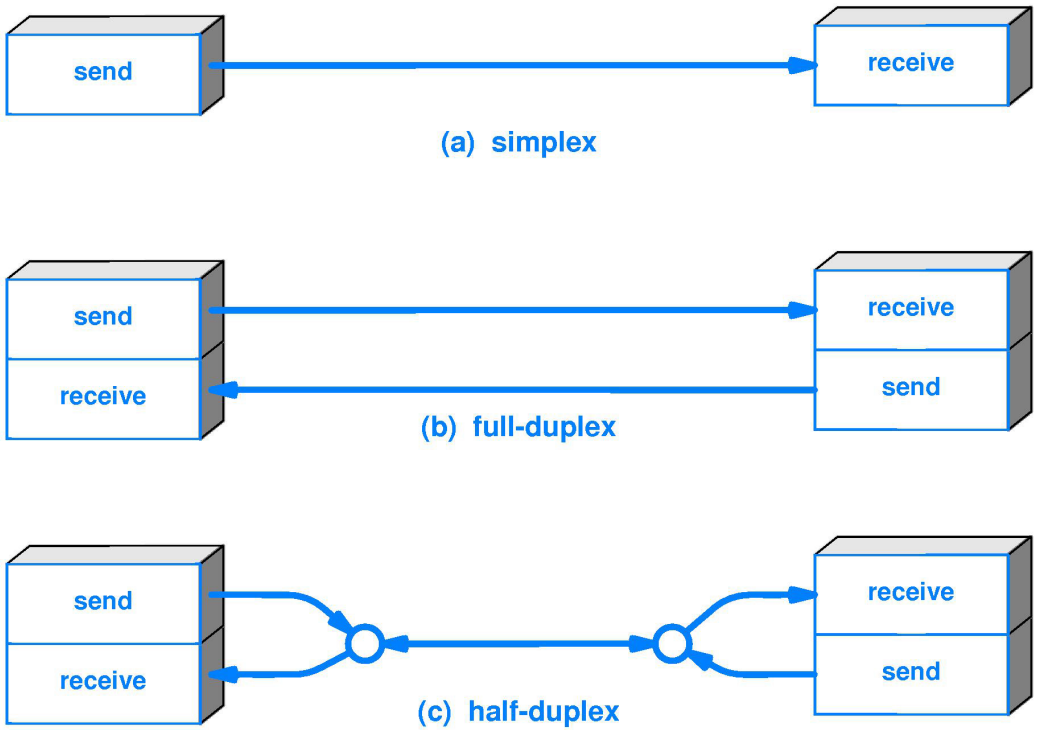


Figure: Serial Communication

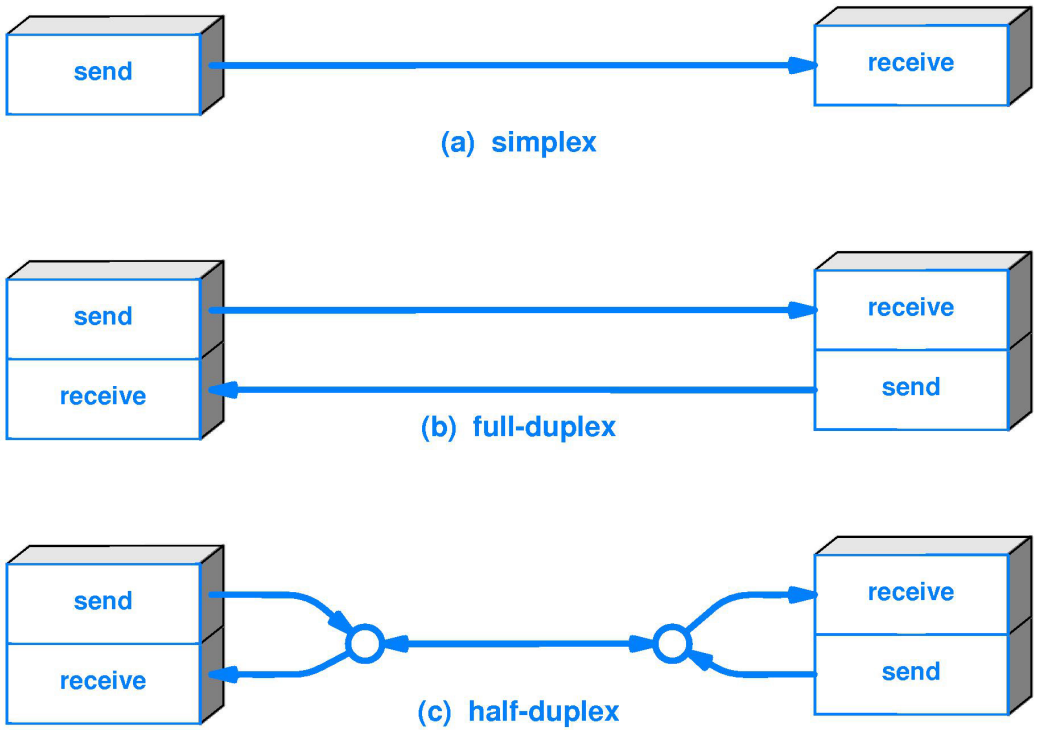
One bit is sent at a time.

Simplex vs Duplex

Simplex vs Duplex



Simplex vs Duplex



Synchronous vs Asynchronous

Synchronous vs Asynchronous

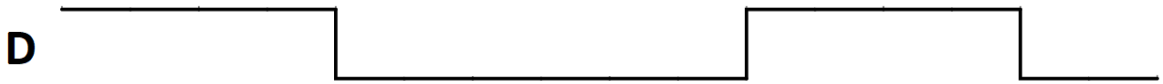


Figure: Data sent on Data Pin

Synchronous vs Asynchronous

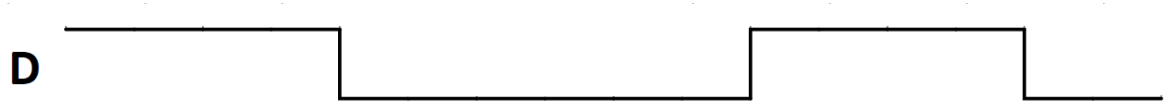


Figure: Data sent on Data Pin

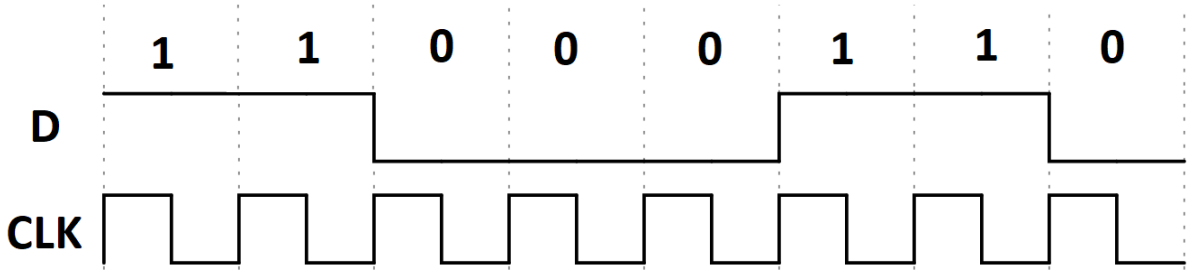


Figure: Data along with Clock

Synchronous vs Asynchronous

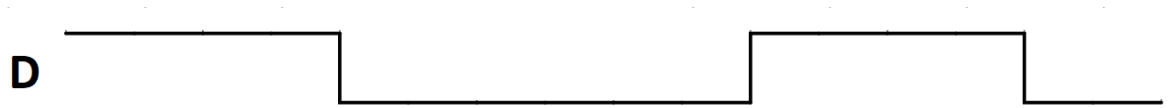


Figure: Data sent on Data Pin

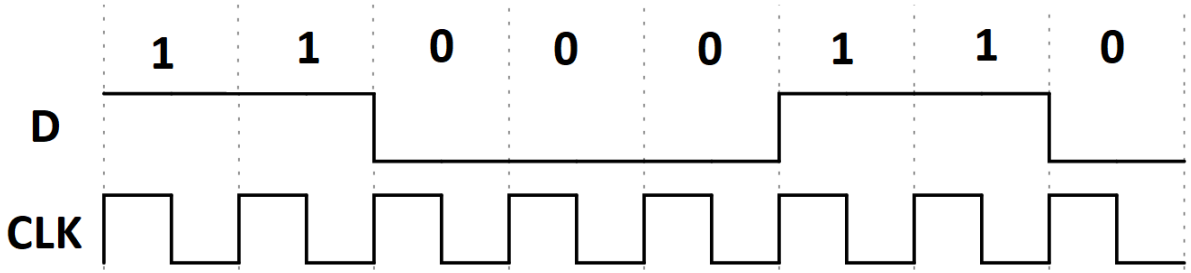


Figure: Data along with Clock

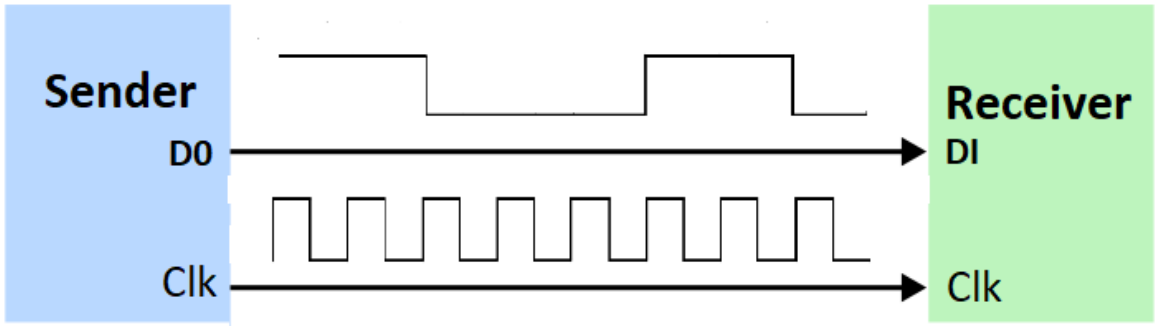


Figure: Communication between Two Devices

Synchronous v/s Asynchronous

Synchronous v/s Asynchronous

Parameters	**Synchronous**	**Asynchronous**
Clock signal	Required	Not required
Overhead bits	Not required	Required
Data transmission speed	Fast	Slow
Data Tx/Rx	Blocks or frames	Bytes or character

Different types of communication available in Embedded System

Different types of communication available in Embedded System

- UART (Universal Asynchronous Receiver Transmitter)

Different types of communication available in Embedded System

- UART (Universal Asynchronous Receiver Transmitter)
- I2C (Inter-Integrated Communication)

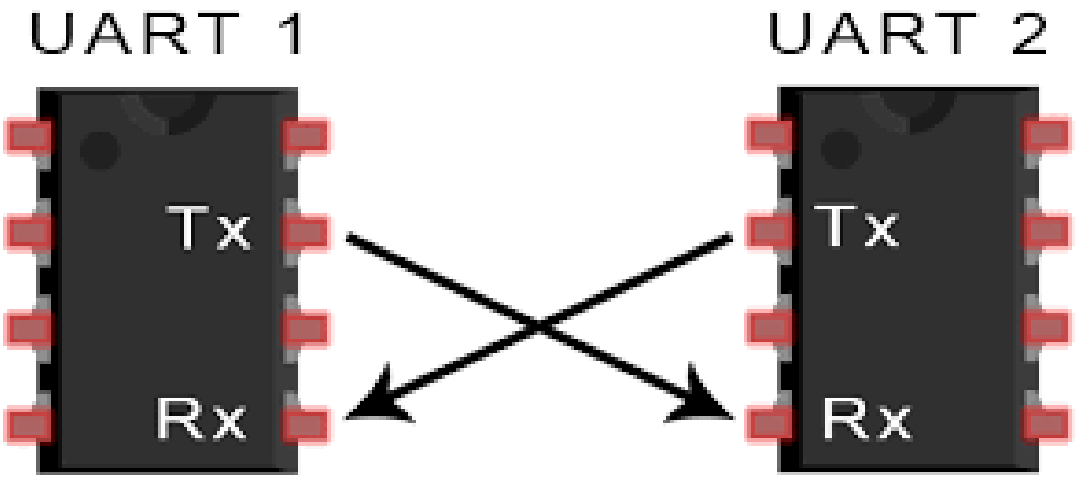
Different types of communication available in Embedded System

- UART (Universal Asynchronous Receiver Transmitter)
- I2C (Inter-Integrated Communication)
- SPI (Serial Peripheral Interface)

Different types of communication available in Embedded System

Different types of communication available in Embedded System

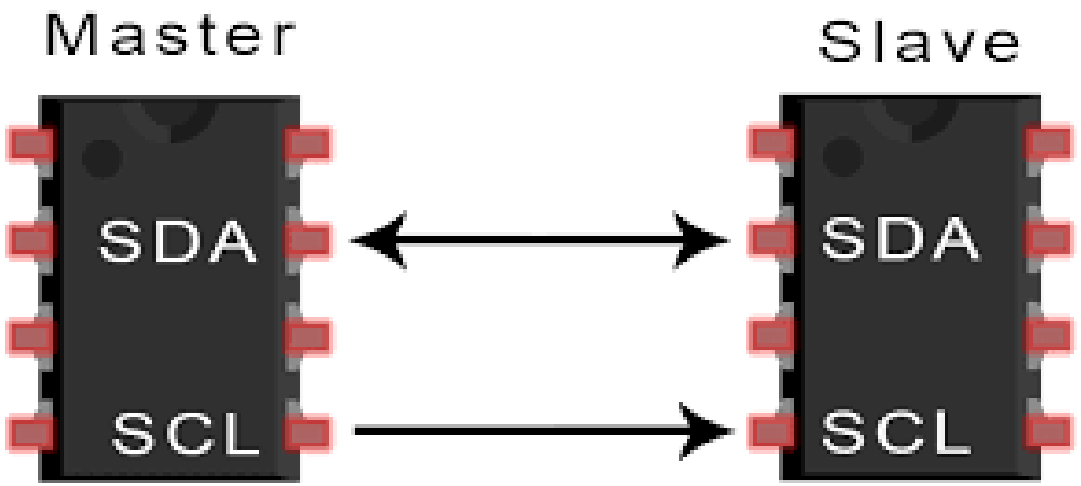
UART (Universal Asynchronous Receiver Transmitter)



Different types of communication available in Embedded System

Different types of communication available in Embedded System

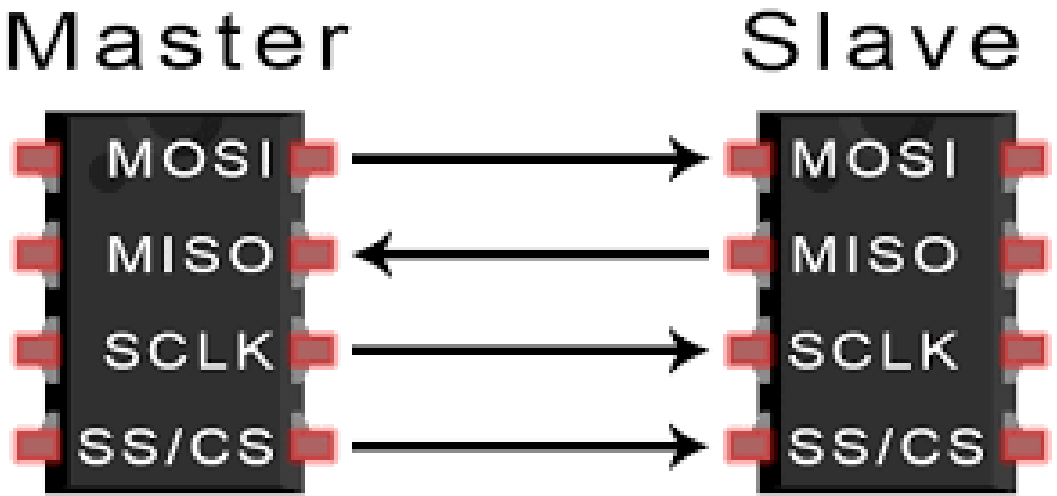
I2C (Inter-Integrated Communication)



Different types of communication available in Embedded System

Different types of communication available in Embedded System

SPI (Serial Peripheral Interface)



What is UART?

What is UART?

- **Universal Asynchronous Receiver Transmitter (UART)** is a **serial asynchronous** communication protocol used to send/receive data between microcontroller and PC or other devices.

What is UART?

- **Universal Asynchronous Receiver Transmitter (UART)** is a **serial asynchronous** communication protocol used to send/receive data between microcontroller and PC or other devices.
- UART requires two lines for communication
  1. Transmit - TX
  2. Receive - RX

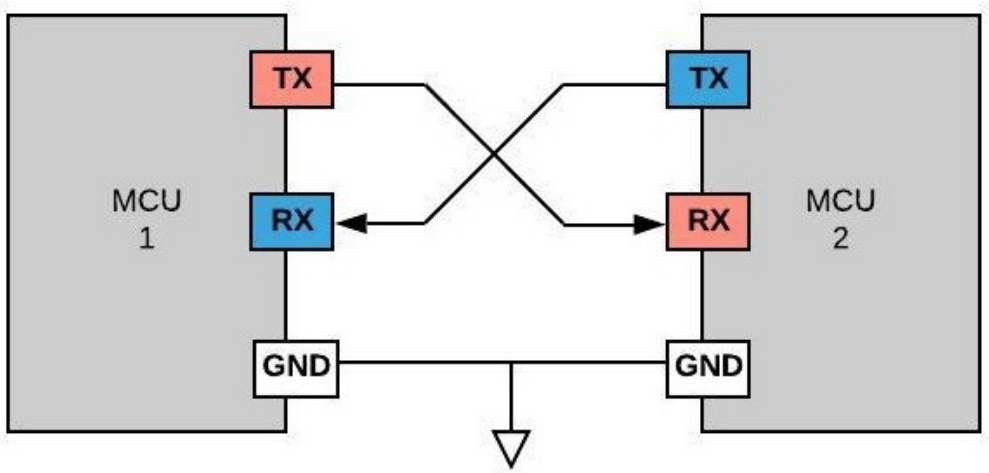


Figure: UART connection between two MCUs

Working of UART

Working of UART

- An external clock signal is not required.

Working of UART

- An external clock signal is not required.
- Extra rules or mechanisms are needed to ensure reliable, error-free sending and receiving of data, which are:

Working of UART

- An external clock signal is not required.
- Extra rules or mechanisms are needed to ensure reliable, error-free sending and receiving of data, which are:
  1. Data Packet
     - Synchronization Bits
     - Data Bits
     - Parity Bits
  2. Baud Rate

Data Packet

Data Packet

- It is a packet of Synchronization and Parity bits appended to Data bits.

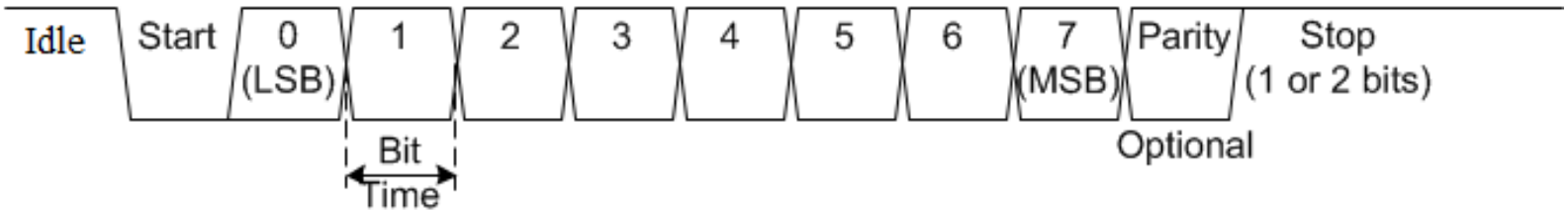


Figure: UART Data Packet

Data Packet

- It is a packet of Synchronization and Parity bits appended to Data bits.

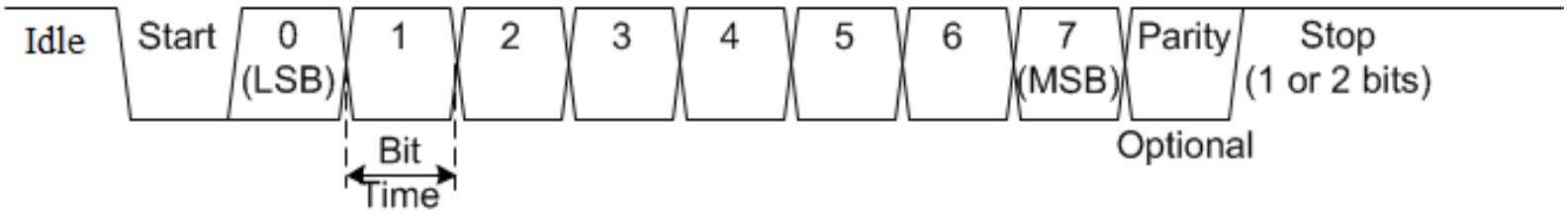


Figure: UART Data Packet

1. **Synchronization Bits**

   Overhead bits are added for each data packet:
   - Start (1 bit) - transition on idle data line from 1 to 0.
   - Stop (1-2 bit/s) - transition back to idle state, holding the line at 1.

Data Packet

Data Packet

2 Data Bits

- Number of Data bits can vary from 5 to 9, the standard data size is 8-bits.
- Data can be sent as Big Endian (MSB first) or as Little Endian (LSB first) and both communicating devices need to agree on the same endianness.
  If not stated, the default is Little-Endian.

Data Packet

2 Data Bits

- Number of Data bits can vary from 5 to 9, the standard data size is 8-bits.
- Data can be sent as Big Endian (MSB first) or as Little Endian (LSB first) and both communicating devices need to agree on the same endianness.
  If not stated, the default is Little-Endian.

3 Parity Bits

- Low-level and simple form of error checking.
- It can be odd or even.
- For example, consider Data byte = **0b10101011**
  If Parity mode = **Even**, then Parity bit = **1**.
  Similarly, if Parity mode = **Odd**, then Parity bit = **0**.

Baud Rate

Baud Rate

- It indicates the rate at which data transfer will occur between two or more devices.

Baud Rate

- It indicates the rate at which data transfer will occur between two or more devices.
- The unit is **bits-per-second (bps)**.

Baud Rate

- It indicates the rate at which data transfer will occur between two or more devices.
- The unit is **bits-per-second (bps)**.
- Baud rates can take any value; but devices must agree to operate on same rate, as communication is asynchronous.

Baud Rate

- It indicates the rate at which data transfer will occur between two or more devices.
- The unit is **bits-per-second (bps)**.
- Baud rates can take any value; but devices must agree to operate on same rate, as communication is asynchronous.
- Commonly used baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.

Baud Rate

- It indicates the rate at which data transfer will occur between two or more devices.
- The unit is **bits-per-second (bps)**.
- Baud rates can take any value; but devices must agree to operate on same rate, as communication is asynchronous.
- Commonly used baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200.
- These baud rates are achieved in microcontroller by dividing the clock frequency.

An Example

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** $\Rightarrow$ Baud Rate

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** $\Rightarrow$ Baud Rate
  2. **8** $\Rightarrow$ Number of data bits in a frame

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** $\Rightarrow$ Baud Rate
  2. **8** $\Rightarrow$ Number of data bits in a frame
  3. **N** $\Rightarrow$ No Parity bits

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** $\Rightarrow$ Baud Rate
  2. **8** $\Rightarrow$ Number of data bits in a frame
  3. **N** $\Rightarrow$ No Parity bits
  4. **1** $\Rightarrow$ 1 Stop bit

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** ⇒ Baud Rate
  2. **8** ⇒ Number of data bits in a frame
  3. **N** ⇒ No Parity bits
  4. **1** ⇒ 1 Stop bit
  5. "**H**" ⇒ ASCII value = **0b01001000** [?]

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** $\Rightarrow$ Baud Rate
  2. **8** $\Rightarrow$ Number of data bits in a frame
  3. **N** $\Rightarrow$ No Parity bits
  4. **1** $\Rightarrow$ 1 Stop bit
  5. "**H**" $\Rightarrow$ ASCII value = **0b01001000** [**?**]
  6. "**i**" $\Rightarrow$ ASCII value = **0b01101001** [**?**]

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.
  1. **9600** ⇒ Baud Rate
  2. **8** ⇒ Number of data bits in a frame
  3. **N** ⇒ No Parity bits
  4. **1** ⇒ 1 Stop bit
  5. "**H**" ⇒ ASCII value = **0b01001000** [**?**]
  6. "**i**" ⇒ ASCII value = **0b01101001** [**?**]
  7. **Endianness** ⇒ Little-endian, by default

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.

1. **9600** ⇒ Baud Rate
2. **8** ⇒ Number of data bits in a frame
3. **N** ⇒ No Parity bits
4. **1** ⇒ 1 Stop bit
5. "**H**" ⇒ ASCII value = **0b01001000** [**?**]
6. "**i**" ⇒ ASCII value = **0b01101001** [**?**]
7. **Endianness** ⇒ Little-endian, by default
8. **10 bits** per packet (1-Start, 8-Data and 1-Stop)

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.

1. **9600** ⇒ Baud Rate
2. **8** ⇒ Number of data bits in a frame
3. **N** ⇒ No Parity bits
4. **1** ⇒ 1 Stop bit
5. "**H**" ⇒ ASCII value = **0b01001000** [**?**]
6. "**i**" ⇒ ASCII value = **0b01101001** [**?**]
7. **Endianness** ⇒ Little-endian, by default
8. **10 bits** per packet (1-Start, 8-Data and 1-Stop)
9. With Baud Rate = 9600 bps, **960 packets** are sent per sec

An Example

- Send the data "**Hi**" with UART configuration **9600-8N1**. Determine the number of packets transferred per second.

1. **9600** ⇒ Baud Rate
2. **8** ⇒ Number of data bits in a frame
3. **N** ⇒ No Parity bits
4. **1** ⇒ 1 Stop bit
5. "**H**" ⇒ ASCII value = **0b01001000** [?]
6. "**i**" ⇒ ASCII value = **0b01101001** [?]
7. **Endianness** ⇒ Little-endian, by default
8. **10 bits** per packet (1-Start, 8-Data and 1-Stop)
9. With Baud Rate = 9600 bps, **960 packets** are sent per sec

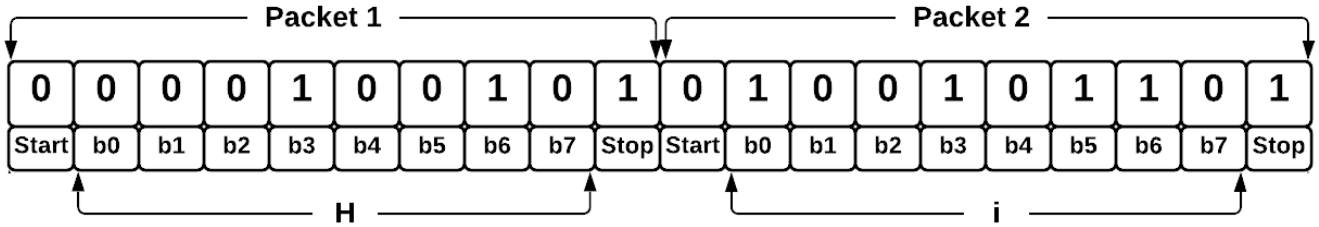


Figure: Data Frame format with two packets

What is I2C?

What is I2C?

Inter-Integrated Circuit, I^2C or I2C or TWI

What is I2C?

Inter-Integrated Circuit, I^2C or I2C or TWI

- Serial and synchronous communication Protocol.

What is I2C?

Inter-Integrated Circuit, I^2C or I2C or TWI

- Serial and synchronous communication Protocol.
- Master-Slave, half duplex protocol.

What is I2C?

Inter-Integrated Circuit, I^2C or I2C or TWI

- Serial and synchronous communication Protocol.
- Master-Slave, half duplex protocol.
- Can be multi-master.

What is I2C?

Inter-Integrated Circuit, I^2C or I2C or TWI

- Serial and synchronous communication Protocol.
- Master-Slave, half duplex protocol.
- Can be multi-master.
- Ensures transmission by acknowledgment.

Master-Slave configuration:

Master-Slave configuration:

Master is responsible for initiating a communication.

Master-Slave configuration:

Master is responsible for initiating a communication.
Clock should be generated by Master only.

Master-Slave configuration:

Master is responsible for initiating a communication.
Clock should be generated by Master only.

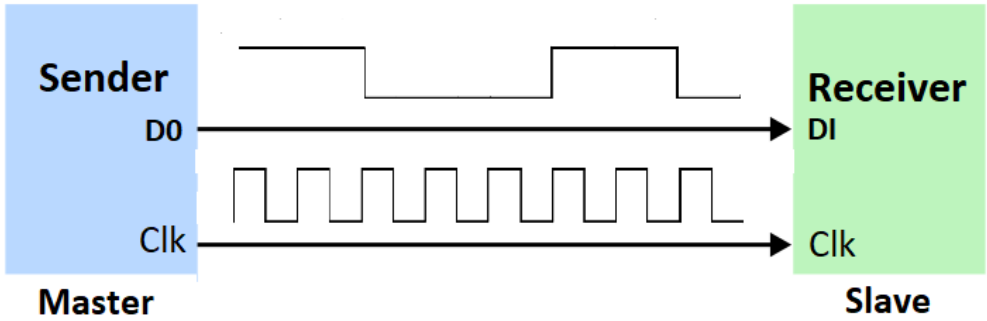


Figure: Master Transmitter Slave Receiver

Master-Slave configuration:

Master is responsible for initiating a communication.
Clock should be generated by Master only.

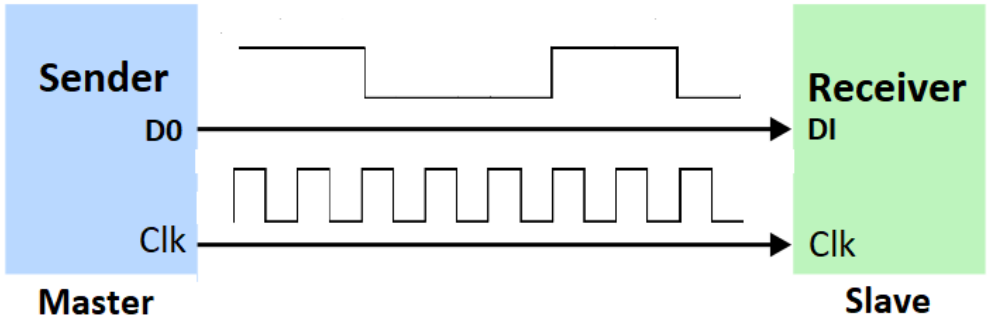


Figure: Master Transmitter Slave Receiver

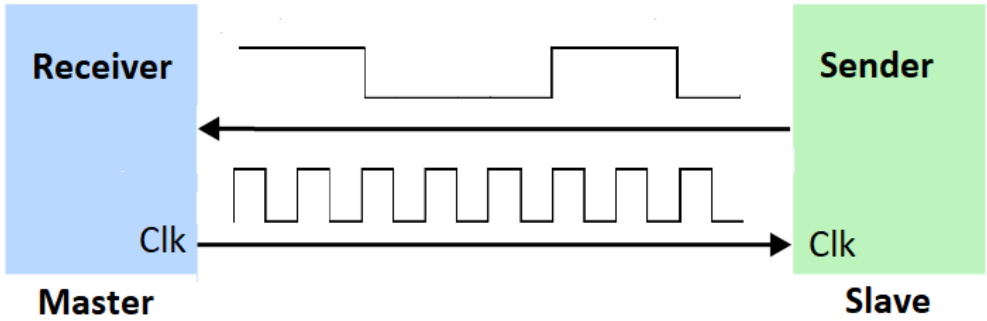


Figure: Master Receiver Slave Transmitter

Master-Slave configuration:

Master is responsible for initiating a communication.
Clock should be generated by Master only.

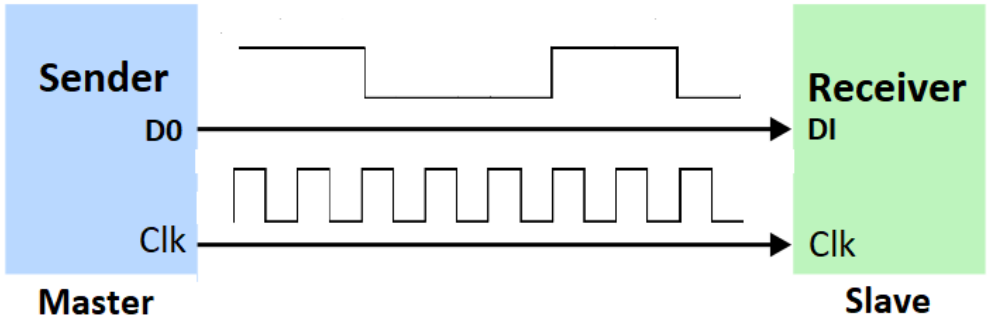


Figure: Master Transmitter Slave Receiver

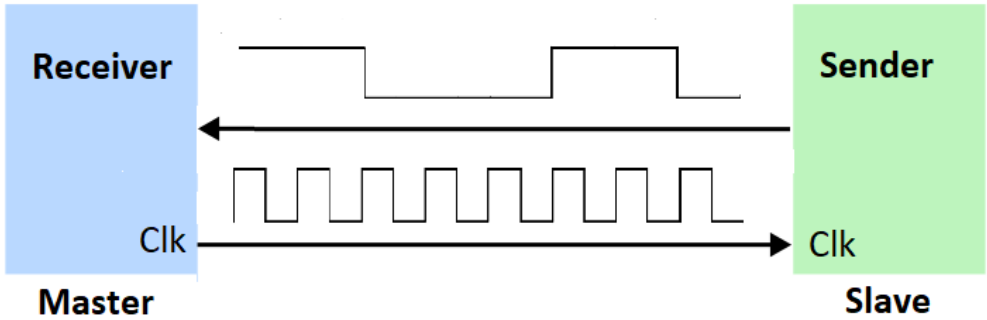


Figure: Master Receiver Slave Transmitter

I2C is a Half-Duplex communication.

Multi-master and multi-slave

Multi-master and multi-slave

We can connect upto 128 devices on I2C bus.

Multi-master and multi-slave

We can connect upto 128 devices on I2C bus.

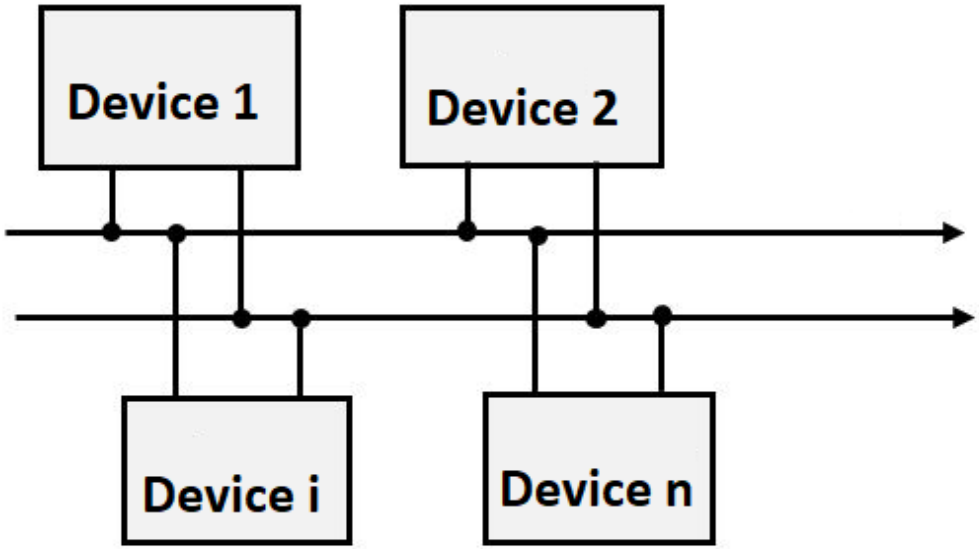


Figure: I2C Communication

Multi-master and multi-slave

We can connect upto 128 devices on I2C bus.

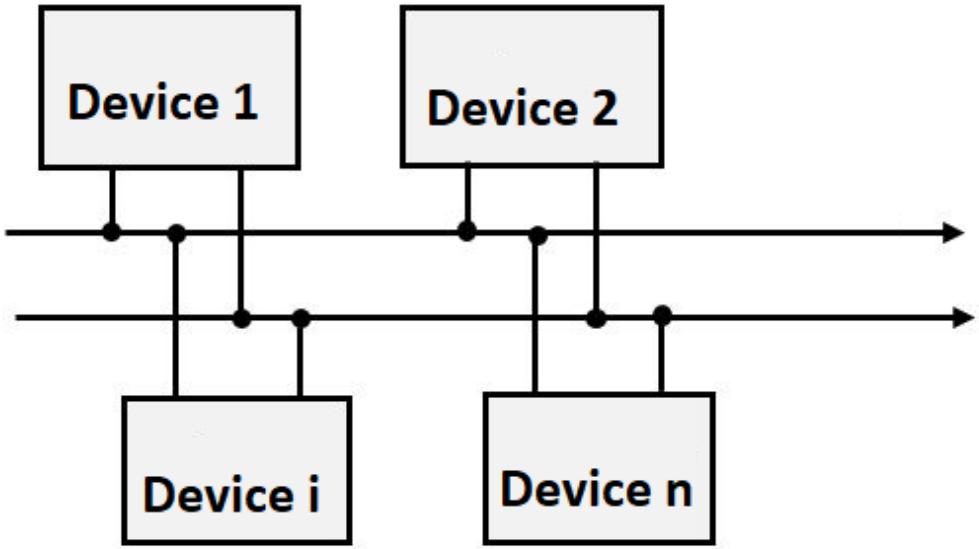


Figure: I2C Communication

Hence n can be maximum 128

Multi-master and multi-slave

We can connect upto 128 devices on I2C bus.

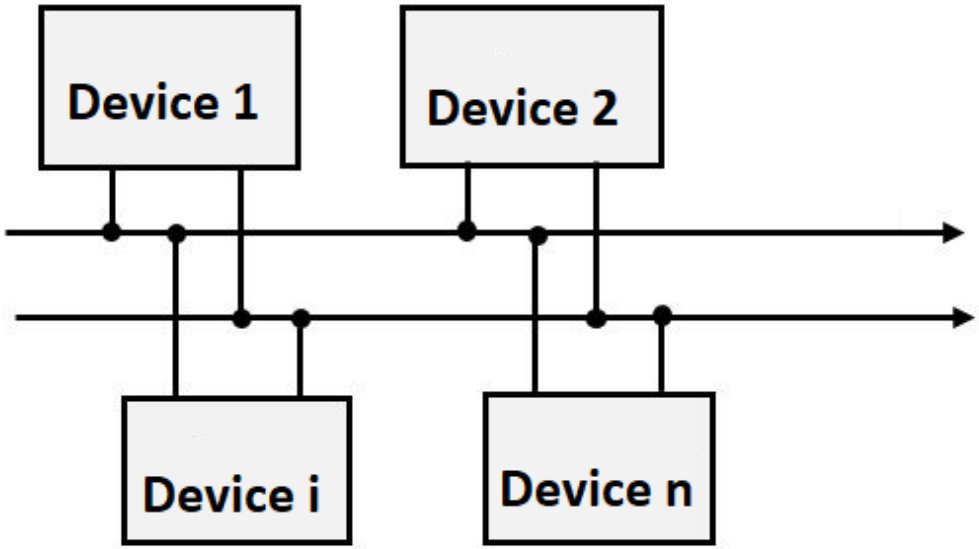


Figure: I2C Communication

Hence n can be maximum 128
At a time only one device will act as a master

Multi-master and multi-slave

We can connect upto 128 devices on I2C bus.

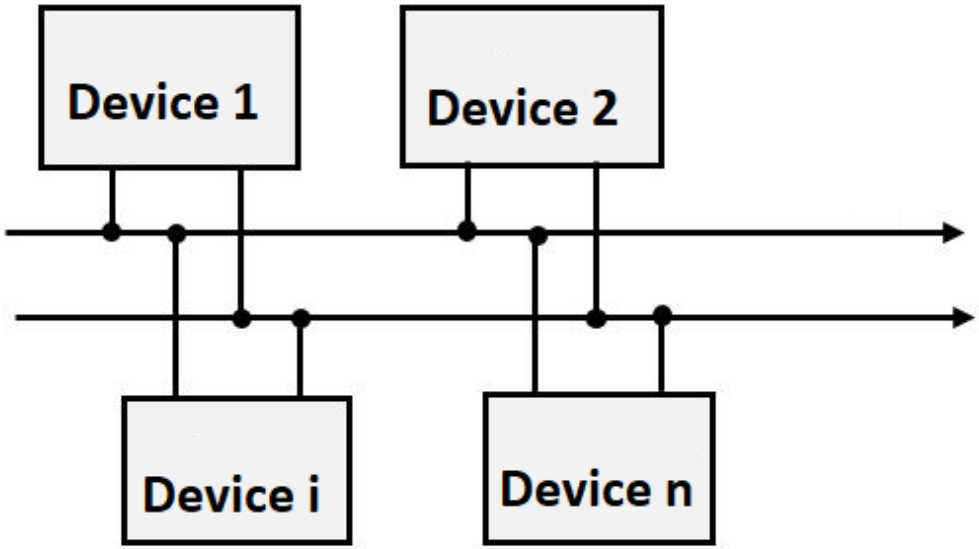
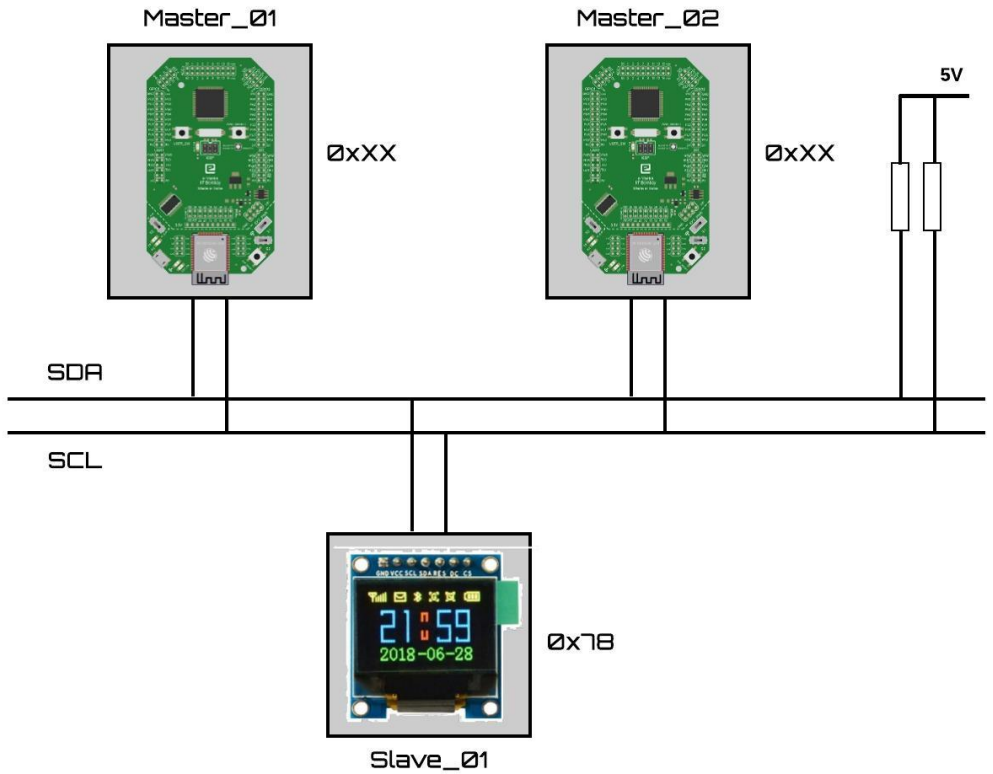


Figure: I2C Communication

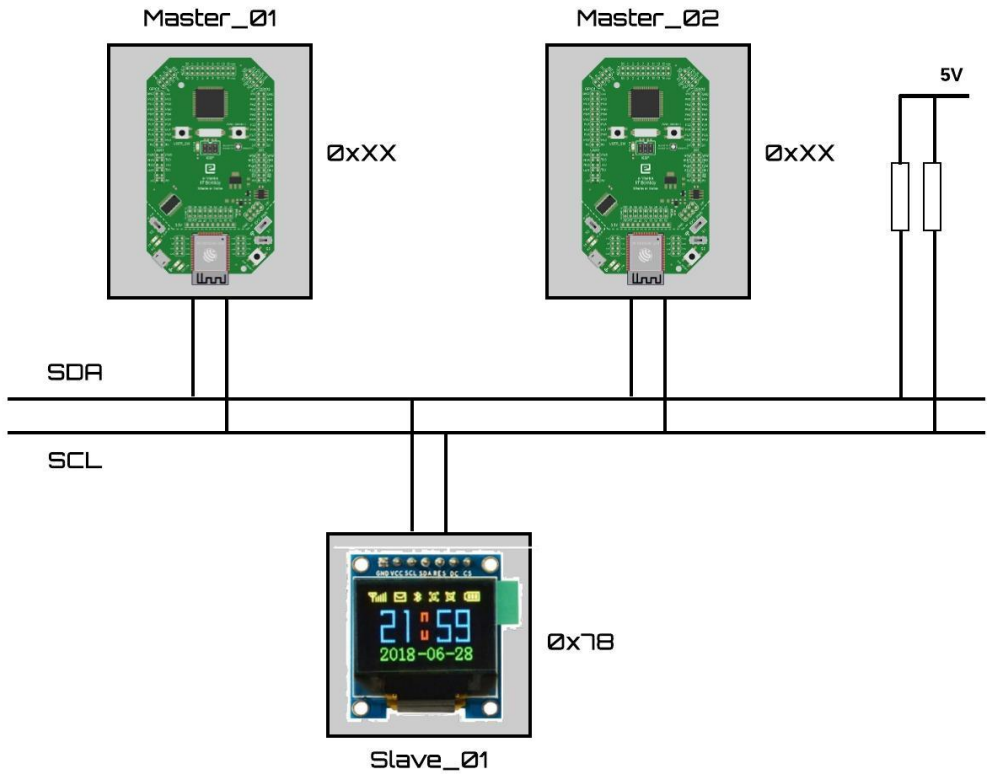
Hence n can be maximum 128
At a time only one device will act as a master
Each device in I2C is addressed by its unique address

Connection Diagram:

Connection Diagram:



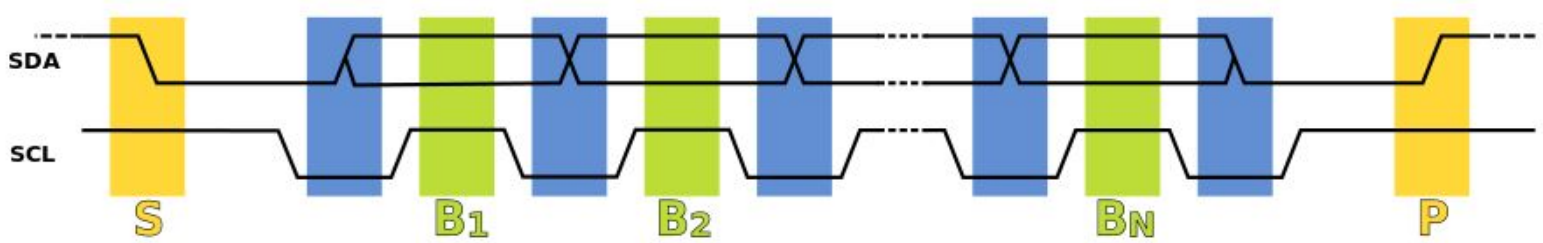
Connection Diagram:



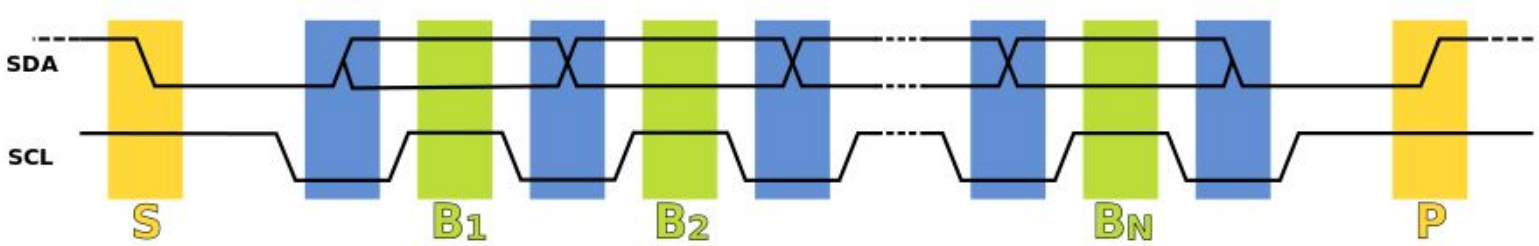
- Pins required for I2C:
  1. SDA: Serial Data Line - To send and receive information.
  2. SCL: Serial Clock Line - To synchronize the communication.

I2C Communication Steps:

I2C Communication Steps:

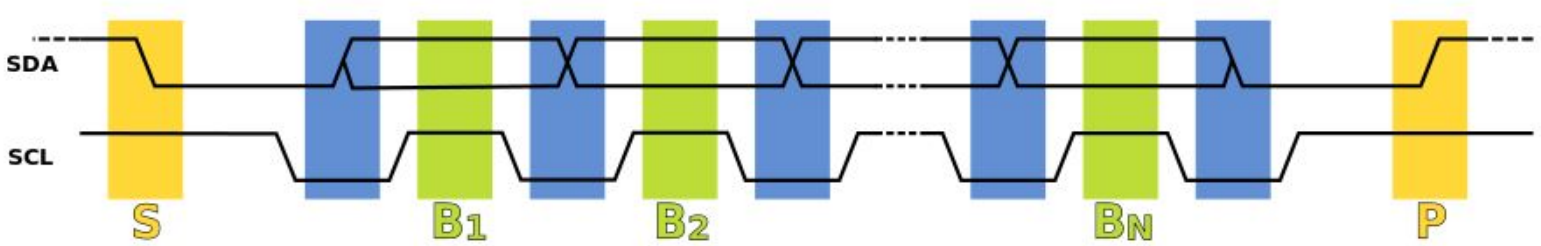


I2C Communication Steps:



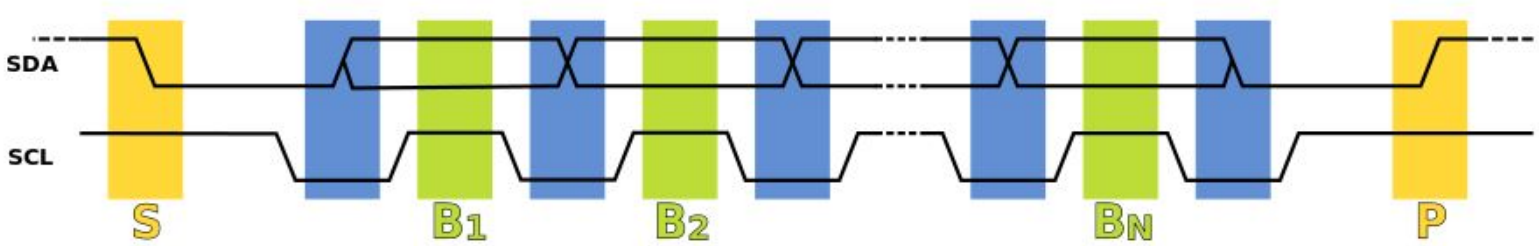
1. Initiate data transfer with a start bit (S) - SDA being pulled low while SCL stays high.

I2C Communication Steps:



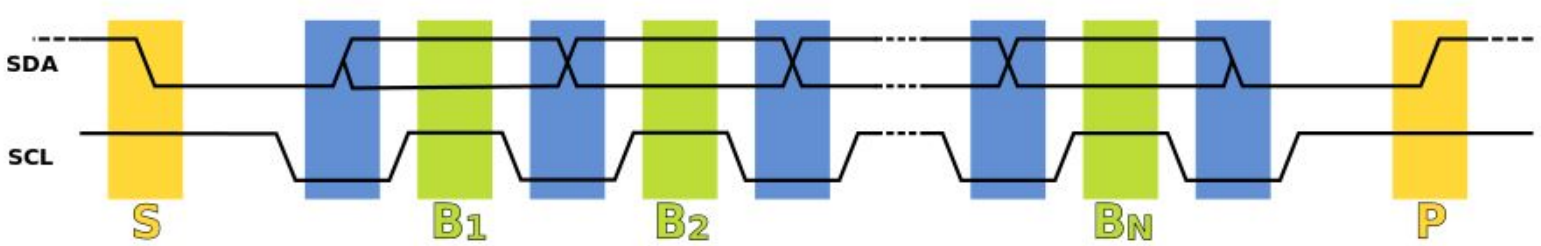
1. Initiate data transfer with a start bit (S) - SDA being pulled low while SCL stays high.
2. Send data - SCL is pulled low, and SDA sets the first data bit level.

I2C Communication Steps:



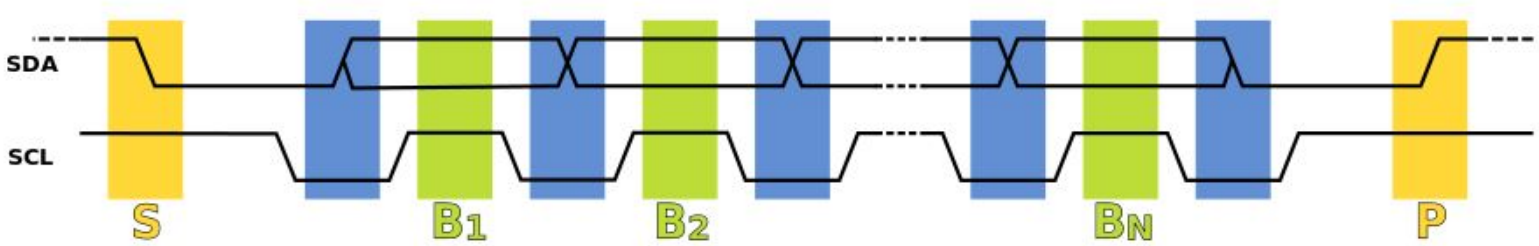
1. Initiate data transfer with a start bit (S) - SDA being pulled low while SCL stays high.
2. Send data - SCL is pulled low, and SDA sets the first data bit level.
3. Receive data - SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge.

I2C Communication Steps:



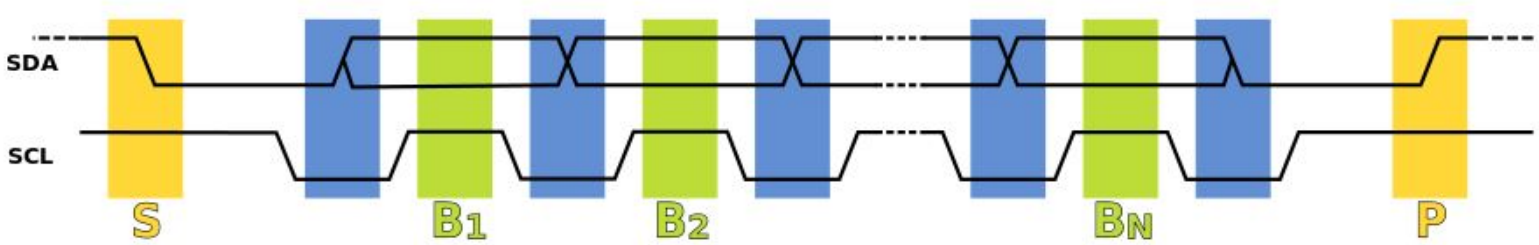
1. Initiate data transfer with a start bit (S) - SDA being pulled low while SCL stays high.
2. Send data - SCL is pulled low, and SDA sets the first data bit level.
3. Receive data - SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge.
4. Process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high.

I2C Communication Steps:



1. Initiate data transfer with a start bit (S) - SDA being pulled low while SCL stays high.
2. Send data - SCL is pulled low, and SDA sets the first data bit level.
3. Receive data - SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge.
4. Process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high.
5. Stop data transfer - SDA is pulled low in preparation for the stop bit.

I2C Communication Steps:



1. Initiate data transfer with a start bit (S) - SDA being pulled low while SCL stays high.
2. Send data - SCL is pulled low, and SDA sets the first data bit level.
3. Receive data - SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge.
4. Process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high.
5. Stop data transfer - SDA is pulled low in preparation for the stop bit.
6. A stop bit (P) is signaled when SCL rises, followed by SDA rising.

I2C Protocol

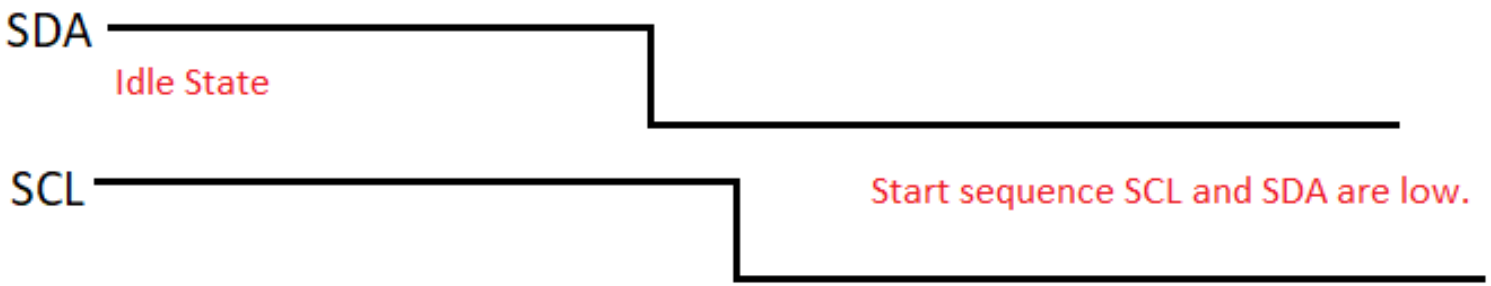
I2C Protocol

Start + Slave Addressing + Ack + Data transfer + Ack + Stop

I2C Protocol

Start + Slave Addressing + Ack + Data transfer + Ack + Stop

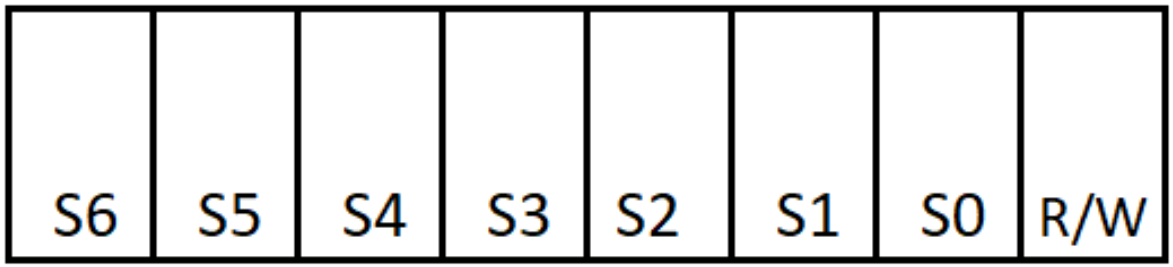
- Start condition marks the start of the protocol.
- SCL line is pulled down by lowering the voltage.



I2C Protocol

Start + Slave Addressing + Ack + Data transfer + Ack + Stop

- Slaves are selected by sending 7 or 10 bit along data line.



- R/W bit decides the read or write operation.

I2C Protocol

Start + Slave Addressing + Ack + Data transfer + Ack + Stop

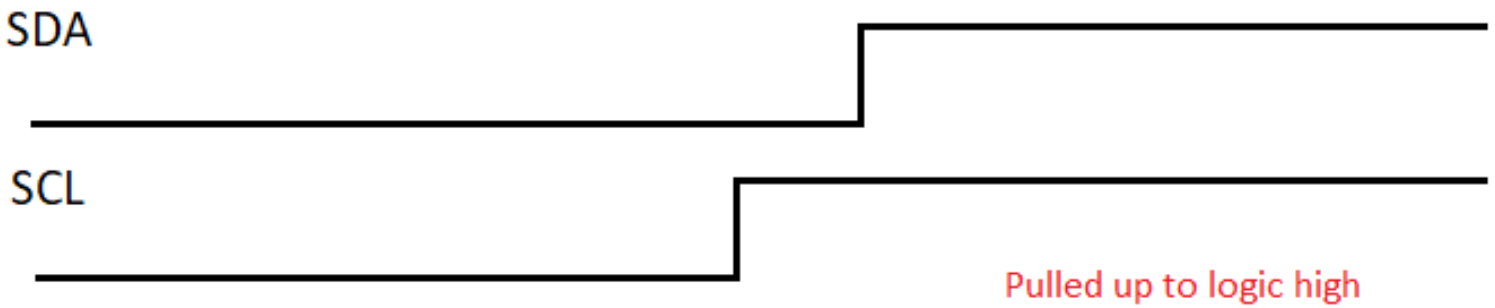
- Data is transferred (Read or Write) b/w master and selected.

- The direction of data transfer is determined by R/W bit.

I2C Protocol

Start + Slave Addressing + Ack + Data transfer + Ack + Stop

- Stop condition marks the End of the protocol.



- SCL and SDA lines are released.

I2C Protocol

I2C Protocol

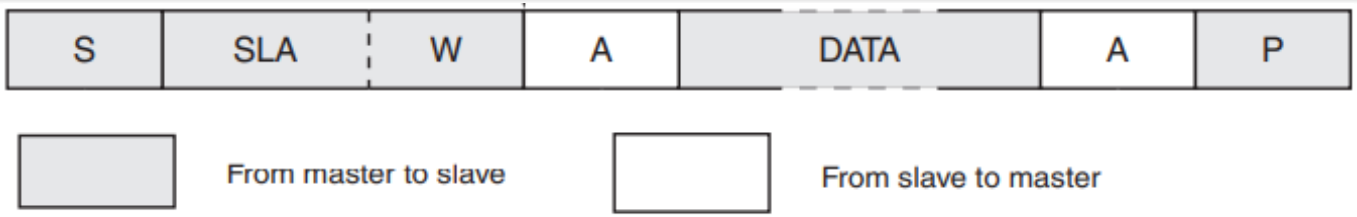


Figure: Master Transmitter

I2C Protocol

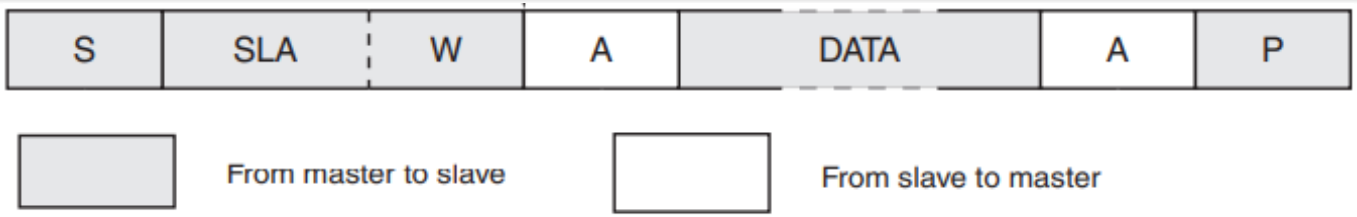


Figure: Master Transmitter

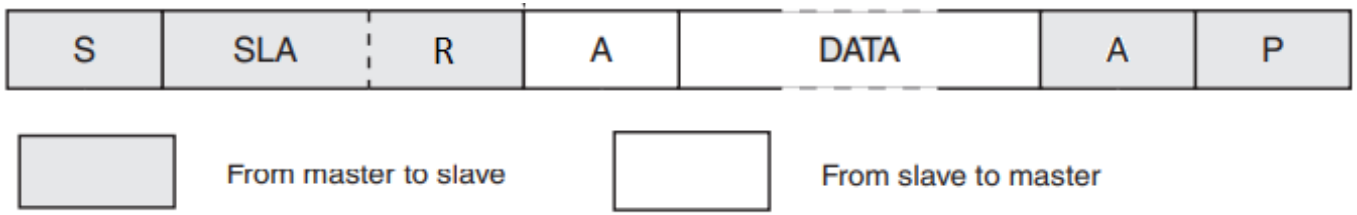


Figure: Master Receiver

I2C Protocol

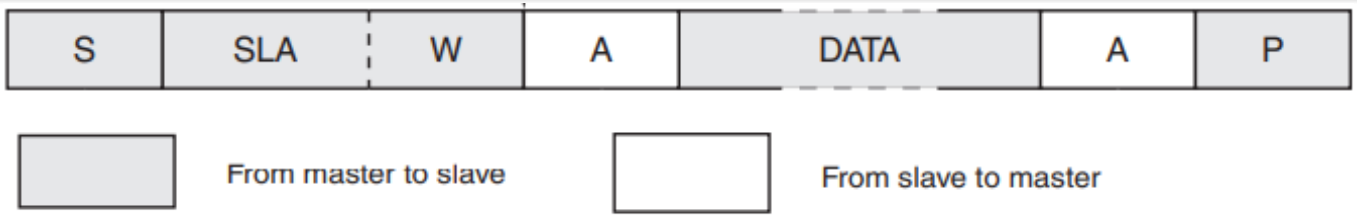


Figure: Master Transmitter

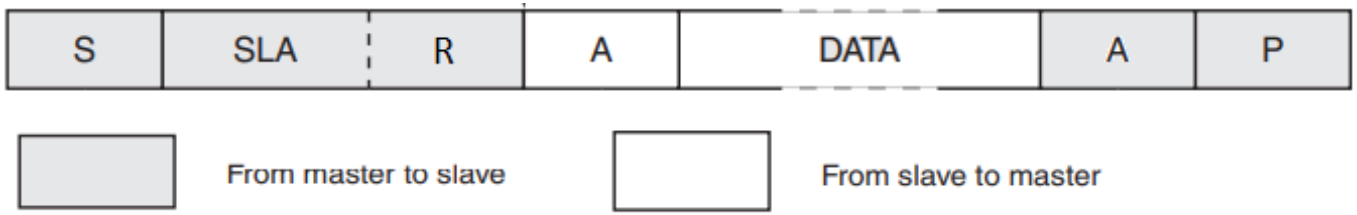


Figure: Master Receiver

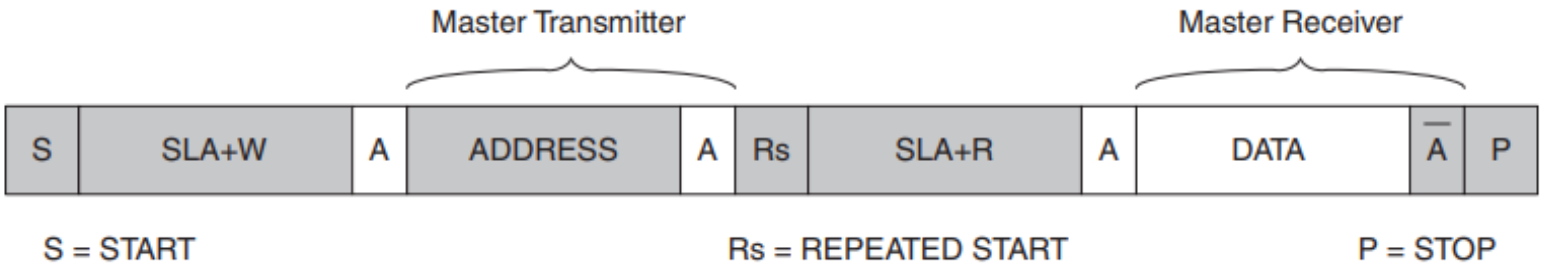
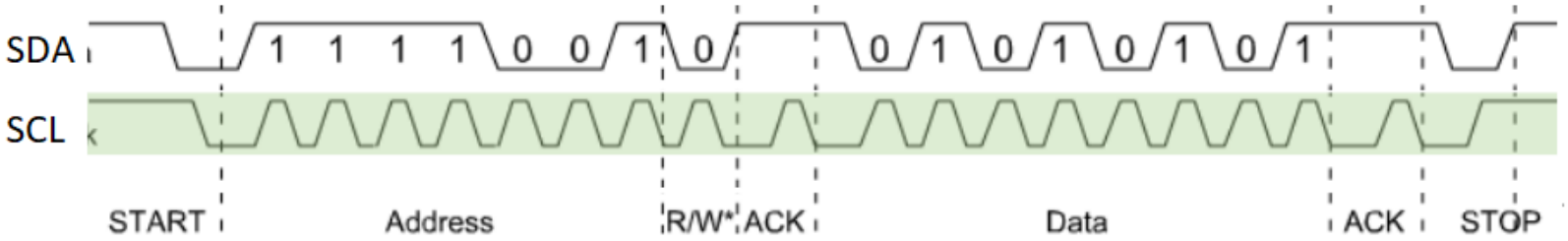


Figure: Master Transmitter and Receiver

I2C Protocol

I2C Protocol



Features of I2C:

- Bus Arbitration: When multiple devices initiates a communication
- Clock Stretching: When slave wants to take control of the clock

What is SPI?

What is SPI?

Serial Peripheral Interface

What is SPI?

Serial Peripheral Interface

- Serial and synchronous communication Protocol.

What is SPI?

Serial Peripheral Interface

- Serial and synchronous communication Protocol.
- Master-Slave, full duplex protocol.

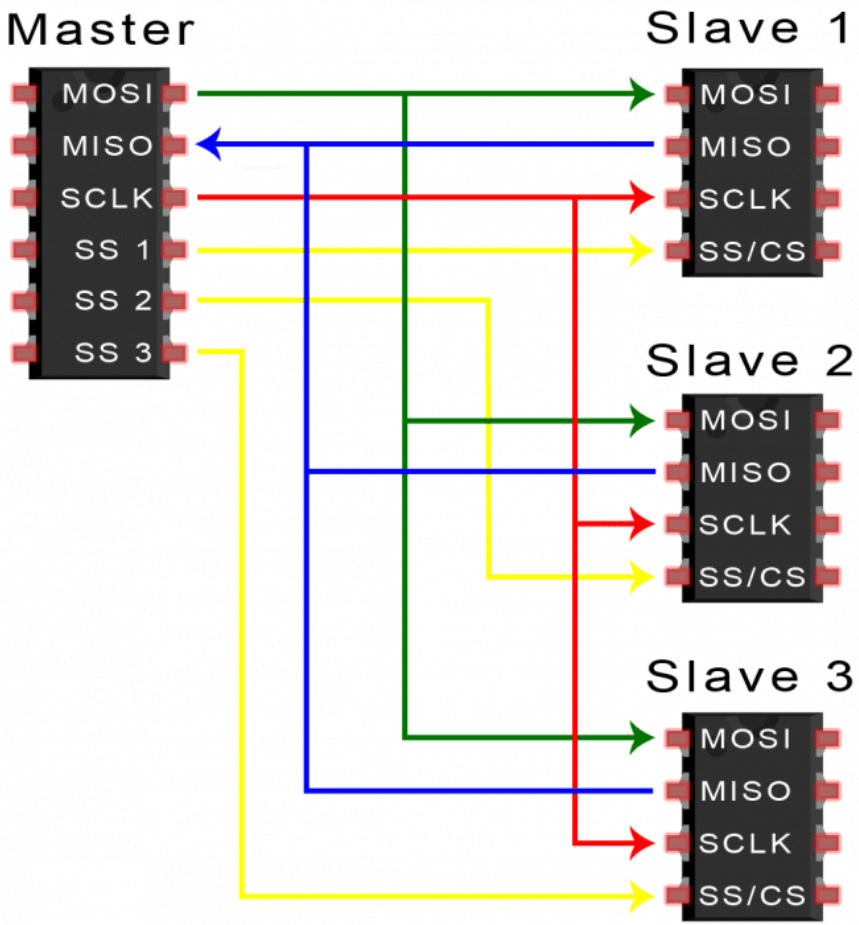
What is SPI?

Serial Peripheral Interface

- Serial and synchronous communication Protocol.
- Master-Slave, full duplex protocol.
- Only Single master.

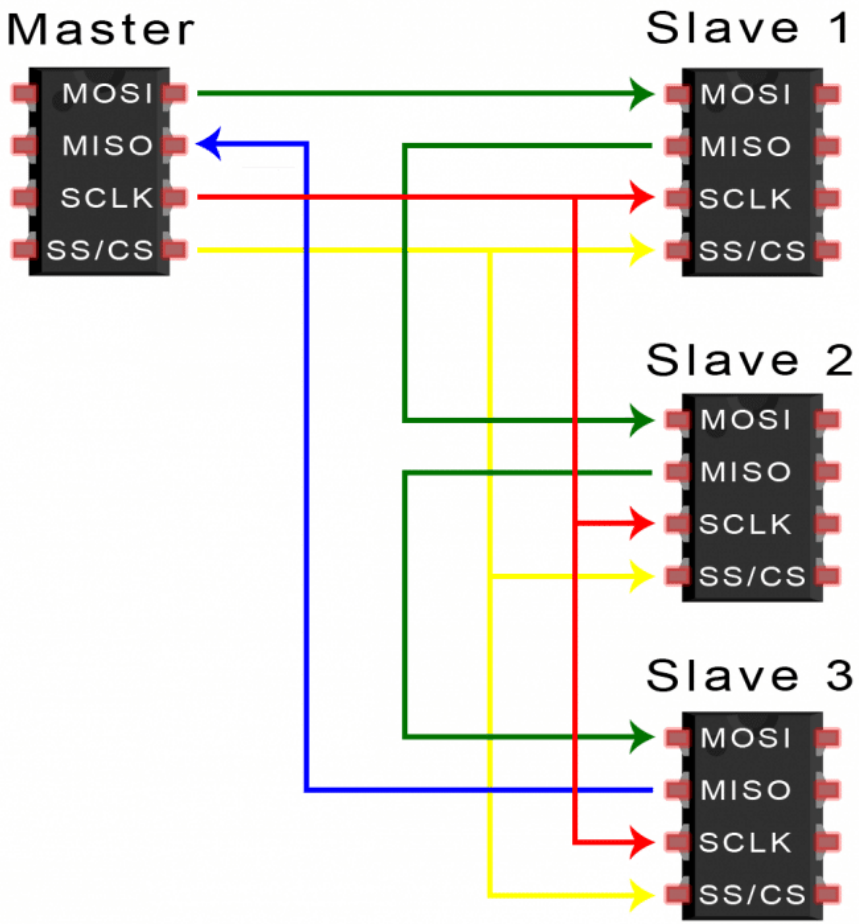
Multiple Slaves

Multiple Slaves



Multiple Slaves

Multiple Slaves



Thank You!

Post your queries on: helpdesk@e-yantra.org