Internet of Things & Its Applications

Dr. Phanindra Thota

Unit-3

IoT Physical Devices and Endpoints - Arduino UNO

IoT Physical Devices and Endpoints - Arduino UNO

- Introduction to Arduino
- Arduino UNO
- Hardware layout
- Interfaces
  - I2C
  - SPI
  - Parallel
- Digital I/O and Analog I/O functions
- Arduino programming

Introduction to Arduino

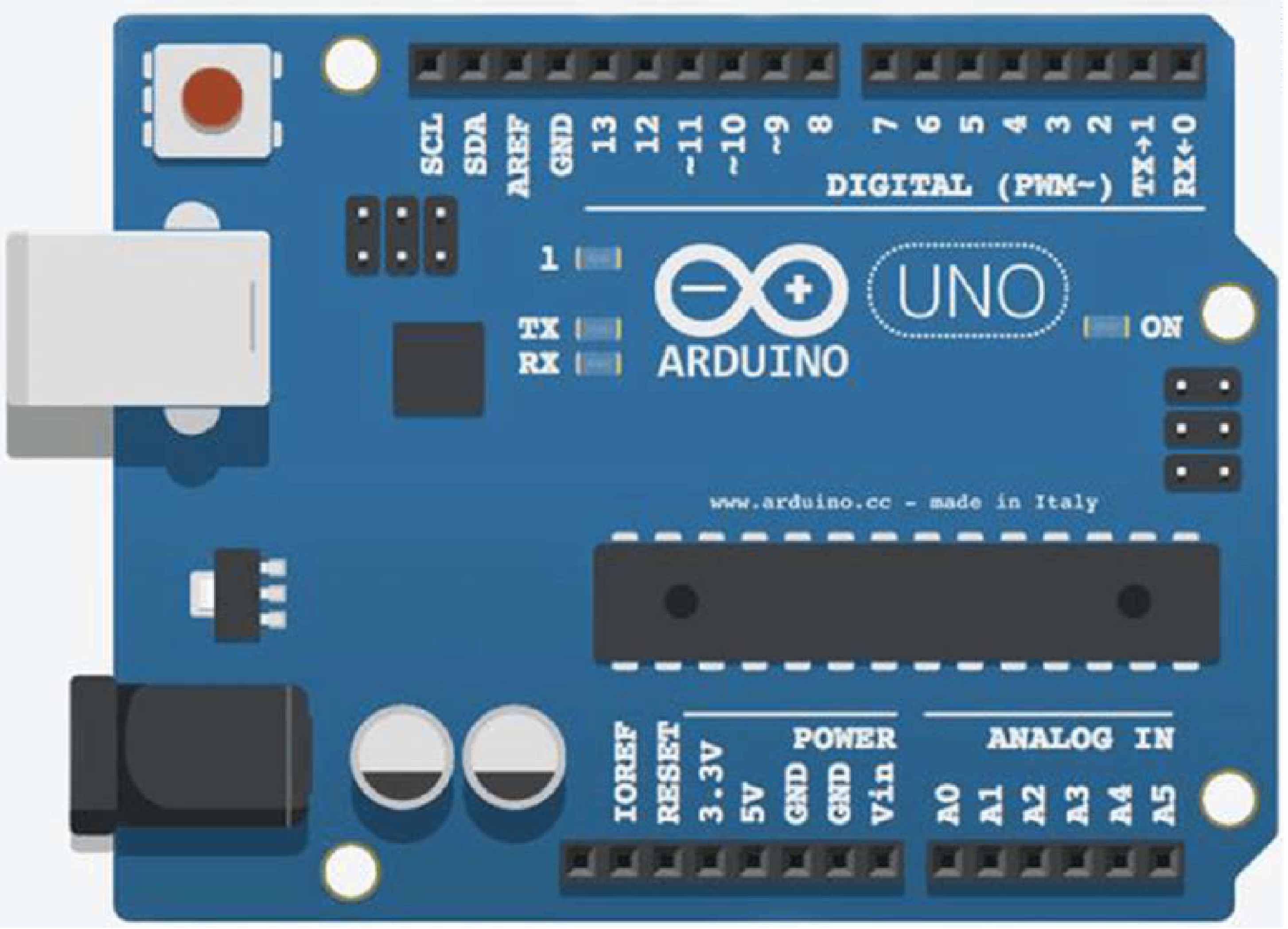
- 2005
- Massimo Banzi, David Cuartielles, Tom Igoe, GianlucaMartino, and David Mellis
- Interaction Design Institute Ivreain Ivrea, Italy
- Aneasy-to-use programmable device for interactive art design
- Easy to connect to various things (such as relays, motors, and sensors),
- Easy to program.
- Inexpensive to make it cost-effective for students and artists
- AVR family of 8-bit microcontroller (MCU or µC) devices from Atmel
- Designed a self-contained circuit board with easy-to-use connections
- Wrote bootloader firmware for the microcontroller,
- Integrated it all into a simple development environment that used programs called "sketches." Arduino.

- Arduino is an open-source microcontroller that enables programming and interaction
- It is programmed in C/C++ with an Arduino library to allow it to access the hardware
- Allows for more flexible programmability and the ability to use electronics that can interface with Arduino
- Arduino is open source –the plans for the circuits are available online for free to anyone who wants to use and create their own board based on the schematics
- Allows for considerable customizability in projects
- Users have built Arduinos of different sizes, shapes, and power levels to control their projects

Arduino is composed of two major parts:

- The **Arduino board**, which is a piece of **hardware** you work on when you build your objects.
- The **Arduino IDE**, which is a piece of **software** you run on your computer. You use the

- It is a multiplatform environment –runs on Windows, Macintosh, and Linux
- IDE –which is an easy-to-use development environment used by artists and designers
- You program it via a USB cable, not a serial port.
- It is open-source hardware and
- The hardware is cheap
- There is an active community of users
- The Arduino project was developed in an educational environment, and is therefore, great for newcomers to get things working quickly.


SCL
SDA
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
1
TX
RX
ARDUINO
UNO
ON
www.arduino.cc - made in Italy
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5

Arduino UNO

The Arduino UNO is a popular microcontroller board based on the ATmega328P. Here are some key features and specifications:

Microcontroller	: ATmega328P
Operating Voltage	: 5V
Digital I/O Pins	: 14 (of which 6 can provide PWM output)
Analog Input Pins	: 6
Flash Memory	: 32 KB (ATmega328P) of which 0.5 KB is used by the bootloader
SRAM	: 2 KB (ATmega328P)
EEPROM	: 1 KB (ATmega328P)
Clock Speed	: 16 MHz

Hardware Overview

1. Microcontroller: The heart of the Arduino UNO is the ATmega328P microcontroller, which includes all the necessary components to operate and interface with various peripherals.

2. Digital I/O Pins: There are 14 digital input/output pins (D0 to D13), six of which can be used as PWM outputs (pins 3, 5, 6, 9, 10, and 11).

3. Analog Input Pins: There are 6 analog inputs (A0 to A5), each providing 10 bits of resolution (i.e., 1024 different values).

4. Power Jack: The board can be powered via a USB connection or with an external power supply. The power source is selected automatically.

5. Reset Button: Allows you to reset the board manually.

6. USB Connection: Used for power, programming, and serial communication with the computer.

Communication

- **Serial Communication:** The UNO has a serial communication interface (UART) via digital pins 0 (RX) and 1 (TX). It also supports serial communication over USB, which is handled by the ATmega16U2 (on newer boards) acting as a USB-to-serial converter.
- **I2C Communication:** The board includes I2C (Inter-Integrated Circuit) communication on pins A4 (SDA) and A5 (SCL).
- **SPI Communication:** The board supports SPI (Serial Peripheral Interface) communication, available on pins 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK).

Arduino Sketch

- Basic function of the Arduino programming language runs in at least two parts
- setup() and loop()
- Declaration of any variables at the very beginning of the program
- setup(): A function present in every Arduino sketch Run once before the loop() function

- loop(): A function present in every single Arduino sketch.
- This code happens over and over again—reading inputs, triggering outputs, etc.
- The loop() is where(almost) everything happens and where the bulk of the work is performed.

```
void setup() {
//code goes here
}
void loop() {
//code goes here
}
```

- Digital I/O will allow us to read the state of an input pin as well as produce a logical high or low at an output pin Each pin of a port can be programmed independently

- Analog I/O refers to signals that can take a range of values. Arduino handles analog input using an ADC (Analog to Digital Converter) and analog output using PWM (Pulse Width Modulation).

Digital I/O functions

pinMode (pin, mode)

digitalWrite (pin, value)

digitalRead (pin)

Analog I/O functions

analogReference (type)

analogWrite (pin, value)

analogRead (pin)

```
pinMode (13, INPUT/OUTPUT);

digitalWrite (13, HIGH/LOW);

digitalRead (13);
```

```
analogRead (pin); (Input, 0-1023)
analogWrite (pin, value); (Output, 0-255 PWM)
```

Digital I/O functions
Digital I/O refers to signals that have two states: HIGH (ON, 5V) or LOW (OFF, 0V).

1. pinMode(pin, mode)

pinMode(pin,mode)
Parameters
pin:the number of the pin whose mode you wish to set
mode:INPUT, OUTPUT, or INPUT_PULLUP
Returns None

- pinMode(13, OUTPUT); // Set pin 13 as an output
- pinMode(2, INPUT); // Set pin 2 as an input

digitalWrite(pin, value)

digitalWrite(pin,value)
Parameters
Pin:the number of the pin you want to write
value:HIGH or LOW
Returns
None

digitalRead(pin)

digitalRead(pin)
Parameters
pin:the number of the pin you want to read (*int*)
Returns
HIGH or LOW

Example:

```
digitalWrite(13, HIGH);
```

```
digitalRead(13);
```

Analog I/O functions

1. analogReference(type)Provides the Arduino’s ADC a reference voltage other than +5 V
Effectively increases the resolution available to analog inputs that operate at some other range of lower voltages below +5 V

The function is only called once in a sketch

Must be declared before analogRead() is used for the first time –placed in the setup() function
The type specified relates to the kind of reference voltage that we want to use.

analogReference(type)

Parameters

type:which type of reference to use (DEFAULT,INTERNAL, INTERNAL1V1, INTERNAL2V56, or EXTERNAL)

Returns

None

DEFAULT: the default analog reference of 5 volts (on 5 V Arduino boards) or 3.3 volts (on 3.3 V Arduino boards)

INTERNAL: a built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (not available on the Arduino Mega)

INTERNAL1V1: a built-in 1.1 V reference (Arduino Mega only)

INTERNAL2V56: a built-in 2.56 V reference (Arduino Mega only)

EXTERNAL: the voltage applied to the AREF pin (0–5 V only) is used as the reference

2. analogRead(pin)

- Reads the value from a specified analog pin with a 10-bit resolution
- Works with the above analogy only for pins (A0–A5)
- analogRead() command will return a number including or between 0 and 1023
- It takes about 100 µs (0.0001 s) to read an analog input maximum reading rate is about 10,000 times a second
- Do not need to be first declared as INPUT nor OUTPUT

analogRead(pin)

Parameters

pin:the number of the analog input pin to read from(0-5)

Returns int(0 to 1023)

3. analogWrite(pin,value)

- The Arduino also has the capability to output a Digital signal that acts as an Analog signal
- This signal is called *pulse width modulation (PWM)*
- Digital Pins 3, 5, 6, 9, 10, and 11 have PWM capabilities
- *To output a PWM signal use the command: analogWrite()*
- You do not need to call pinMode() to set the pin as an output before calling analogWrite()

analogWrite(pin,value)

Parameters

pin:the number of the pin you want to write value: the duty cycle between 0 (always off, 0%) and 255 (always on, 100%)

Returns

None

Communication Interfaces

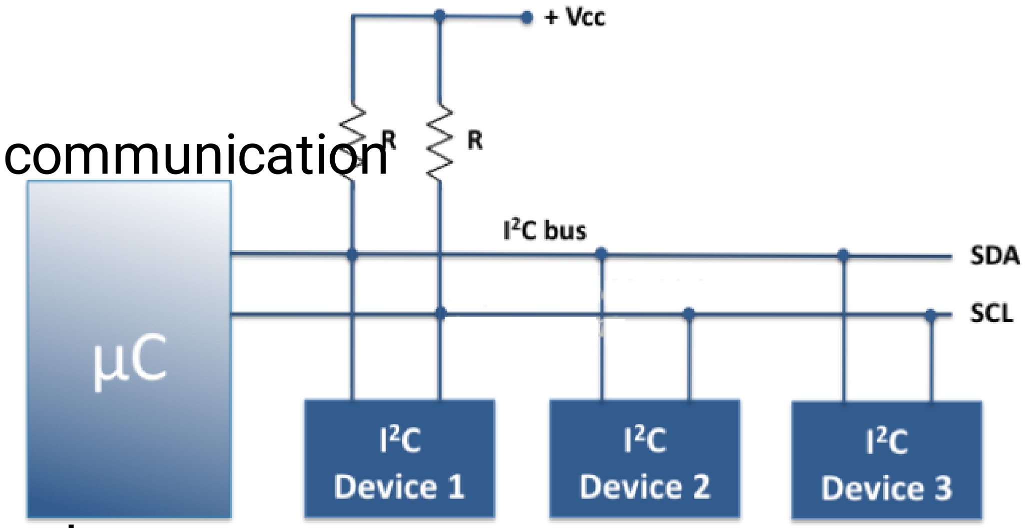
1. On board communication interfaces Inter Integrated Circuit Bus (I2C)
2. I2C is a synchronous bi-directional half duplex (one-directional communication at a given point of time) two wire serial interface bus
3. I2C bus comprise of two bus lines

   Serial Clock–SCL (For Synchronous Communication)

   Serial Data–SDA

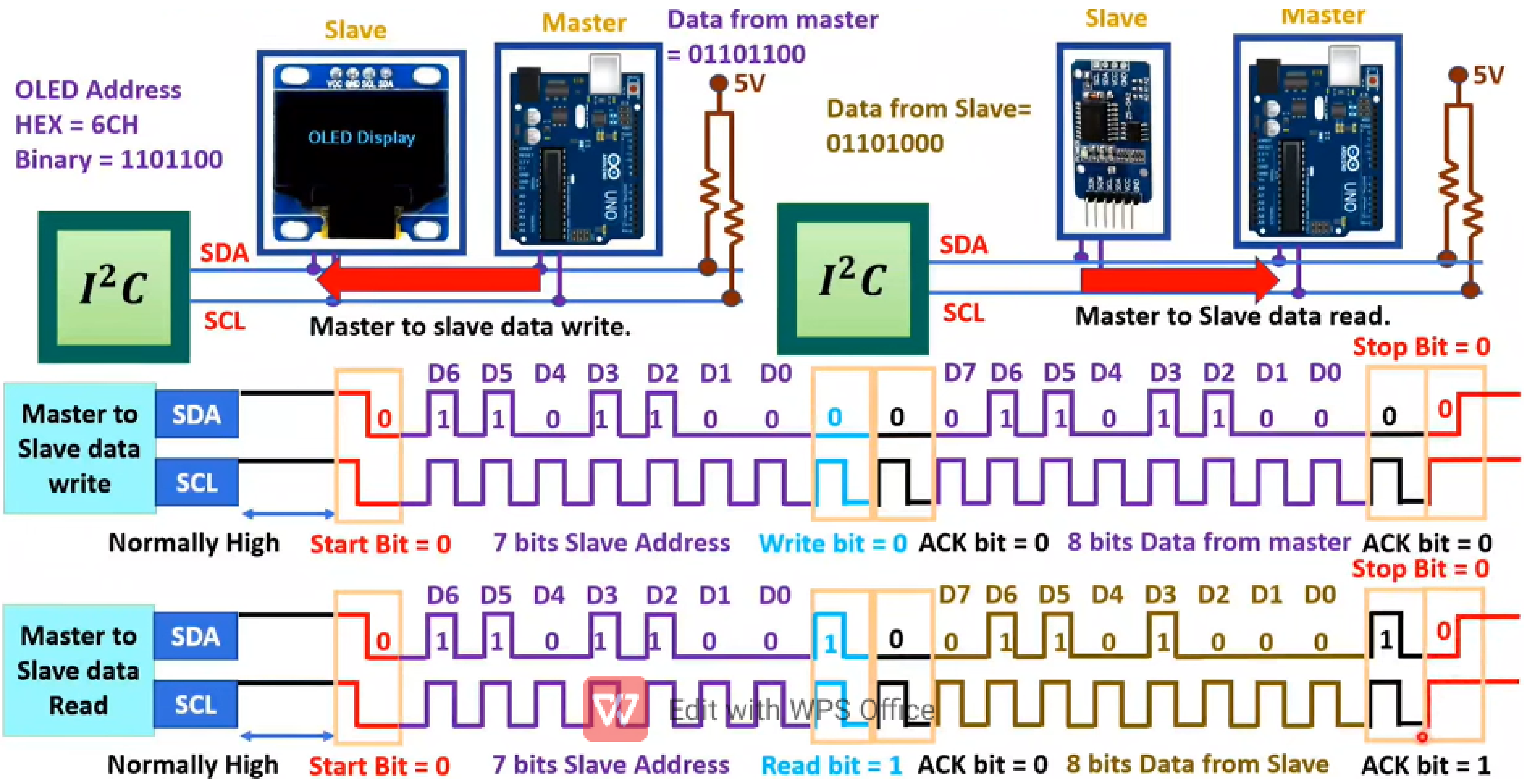
4. SCL line is responsible for generating synchronization clock pulses
5. SDA is responsible for transmitting the serial data across devices
6. I2C bus is a shared bus system to which many number of I2C devices can be
   connected.

- Inter Integrated Circuit Bus (I2C) / TWI (Two Wire Interface)
- I2C bus is a shared bus system to which many number of I2C devices can be connected
- **SDA and SCL are conned to +Vcc**
- ‘Master’ device is responsible for controlling the communication by initiating/terminating data transfer.
- Low Bandwidth protocol used for short range communication
- At max 128 devices we can connect
- I2C is Multi Master-Multi Slave Configuration
- No.of devices is also limited by the total allowed bus capacitance of 400 micro farad



Multiple devices on common I²C bus

OLED Address
HEX = 6CH
Binary = 1101100



I2C - Five operating speed categories

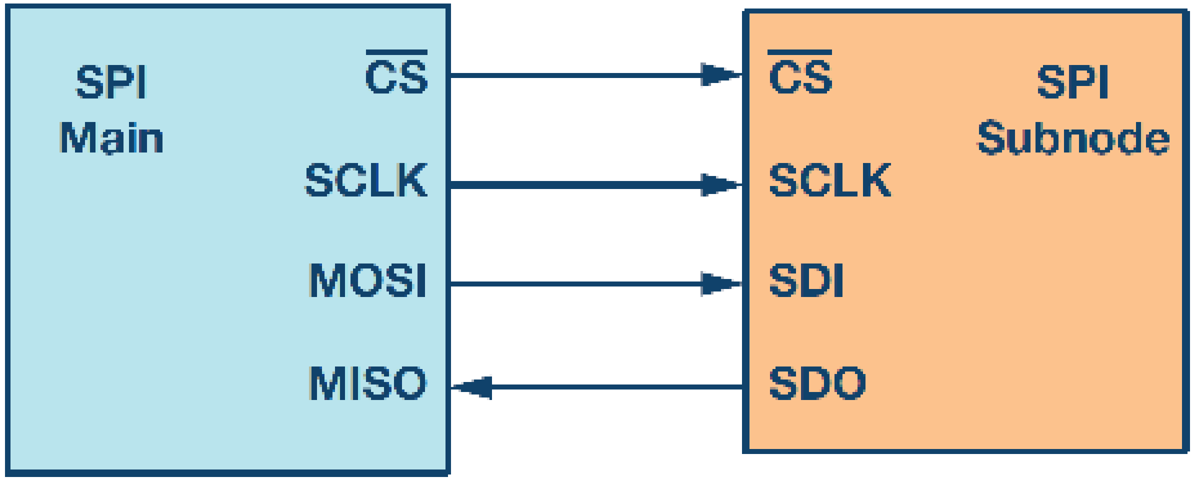
1. **Standard mode** (Sm - Data rate up to 100kbit/sec)
2. **Fast mode** (Fm - Data rate up to 400kbit/sec)
3. **Fast mode Plus** (Fm+ - Data rate up to 1Mbit/sec)
4. **High-speed mode** (Hs-mode - Data rate up to 3.4Mbit/sec)
5. **Ultra Fast-mode** (UFm), with a bit rate up to 5 Mbit/s

IEEE Standard 1355-1995 – for serial communication

Serial Peripheral Interface (SPI) Bus

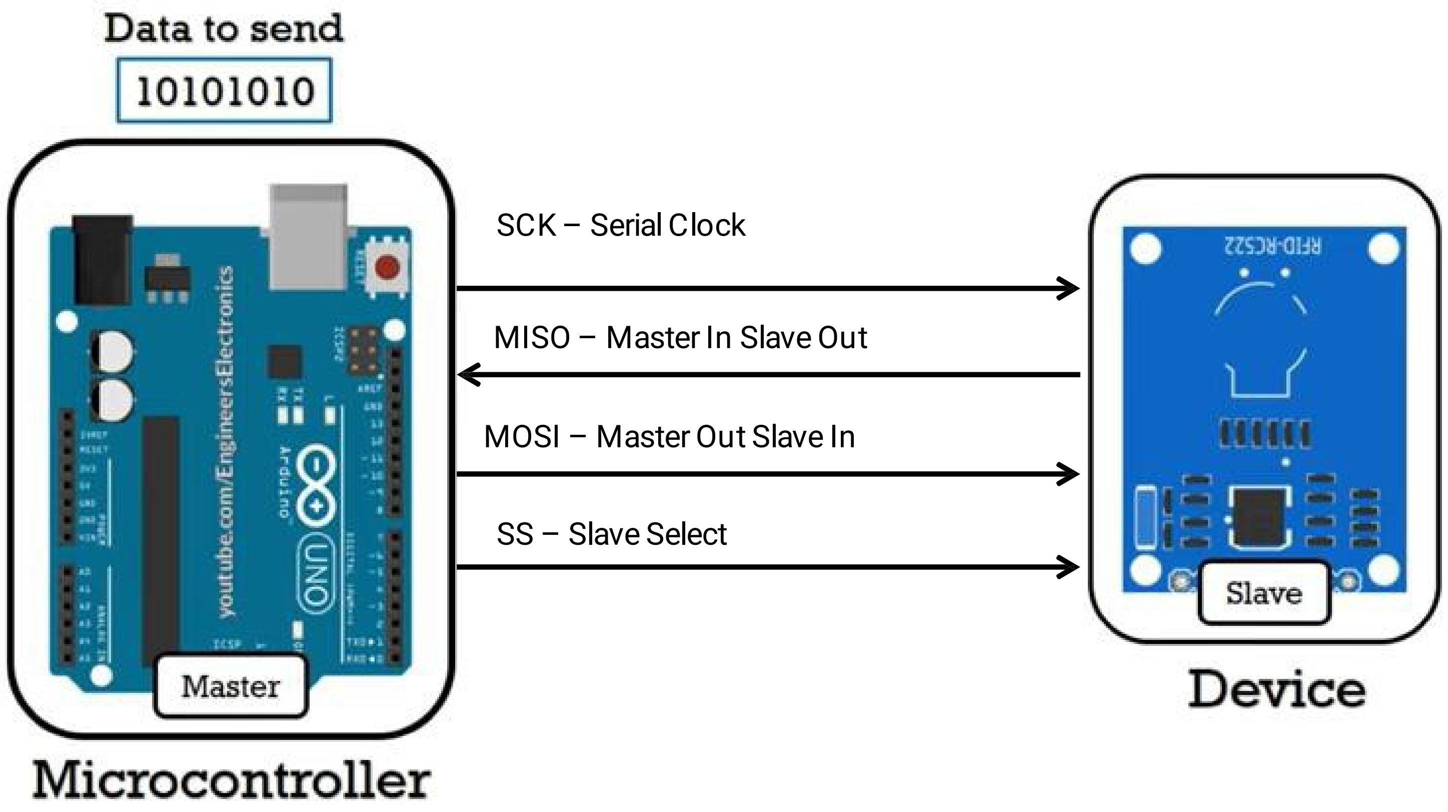
- SPI is a synchronous bi-directional full duplex four-wire serial interface bus
- SPI was introduced by Motorola
- It is a ***single master*** ***multi-slave*** system
- SPI requires four signal lines for communication
- SPI devices support much higher clock frequencies compared to I2C interfaces.

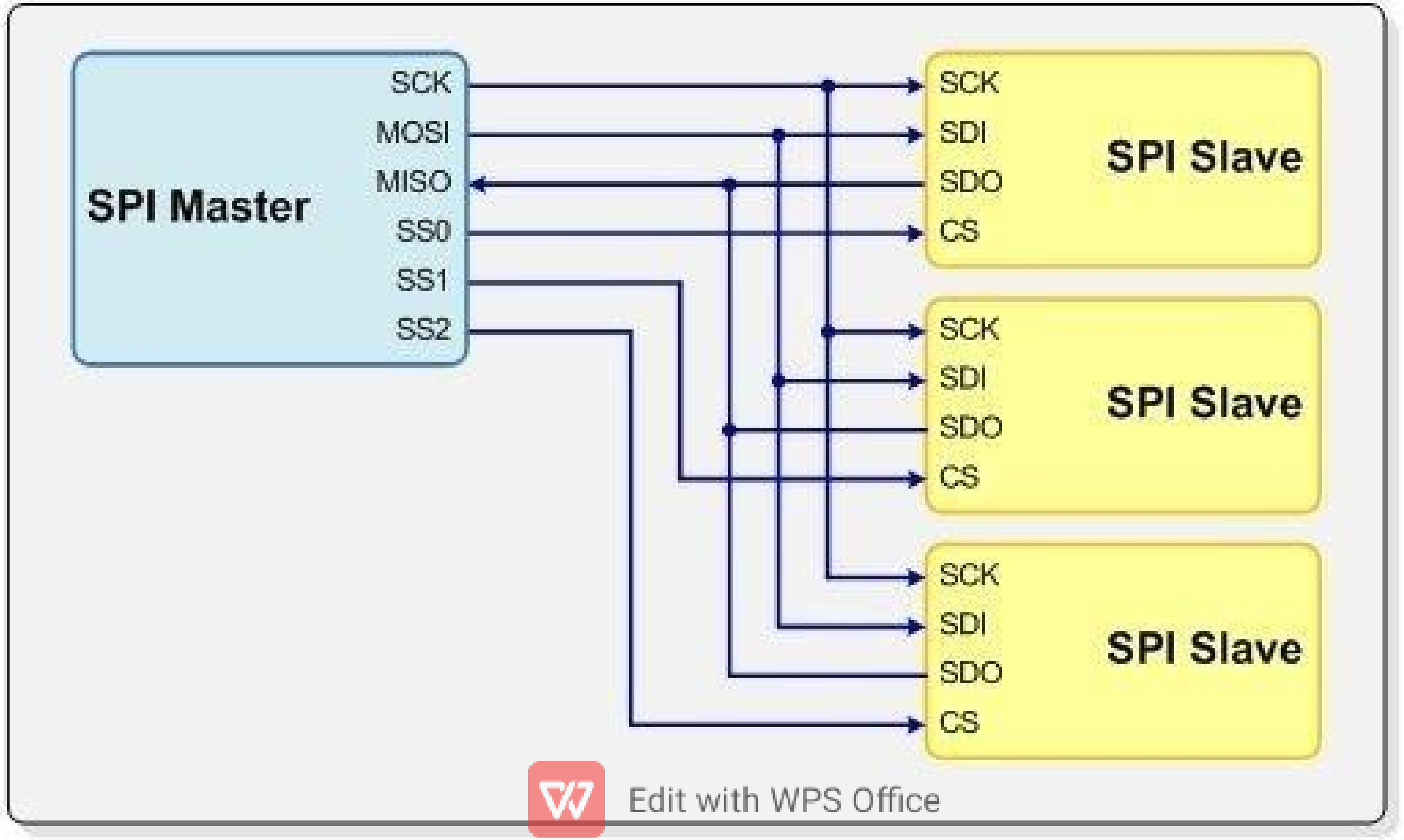
- **Master Out Slave In (MOSI):** Signal line carrying the data from master to slave device. It is also known as Slave Input/Slave Data In (SI/SDI).
- **Master In Slave Out (MISO):** Signal line carrying the data from slave to master device. It is also known as Slave Output (SO/SDO).
- **Serial Clock (SCLK):** Signal line carrying the clock signals.
- **Slave Select (SS):** Signal line for slave device select.



Interface

Serial Peripheral Interface (SPI) Bus – Connection Diagram




SPI Master
SCK
MOSI
MISO
SS0
SS1
SS2
SCK
SDI
SDO
CS
SPI Slave
SCK
SDI
SDO
CS
SPI Slave
SCK
SDI
SDO
CS
SPI Slave

Data Transmission in SPI

- The main must send the clock signal and select the Salve by enabling the CS signal.
- Usually chip select is an active low signal; hence, the main must send a logic 0 on this signal to select the Salve .
- SPI is a **full-duplex interface**; both main and Salve can send data at the same time via the MOSI and MISO lines respectively.
- During SPI communication, the data is simultaneously transmitted (**shifted out serially onto the MOSI/SDO bus**) and received (**the data on the bus (MISO/ SDI) is sampled or read in**).
- The serial clock edge synchronizes the shifting and sampling of the data.

- Clock polarity (CPOL) and Clock Phase(CPHA) are the two parameters that are used to form four unique modes to provide flexibility in communication between master and slave.

- The SPI interface provides the user with f lexibility to select the rising or falling edge of the clock to sample and/or shift the data.

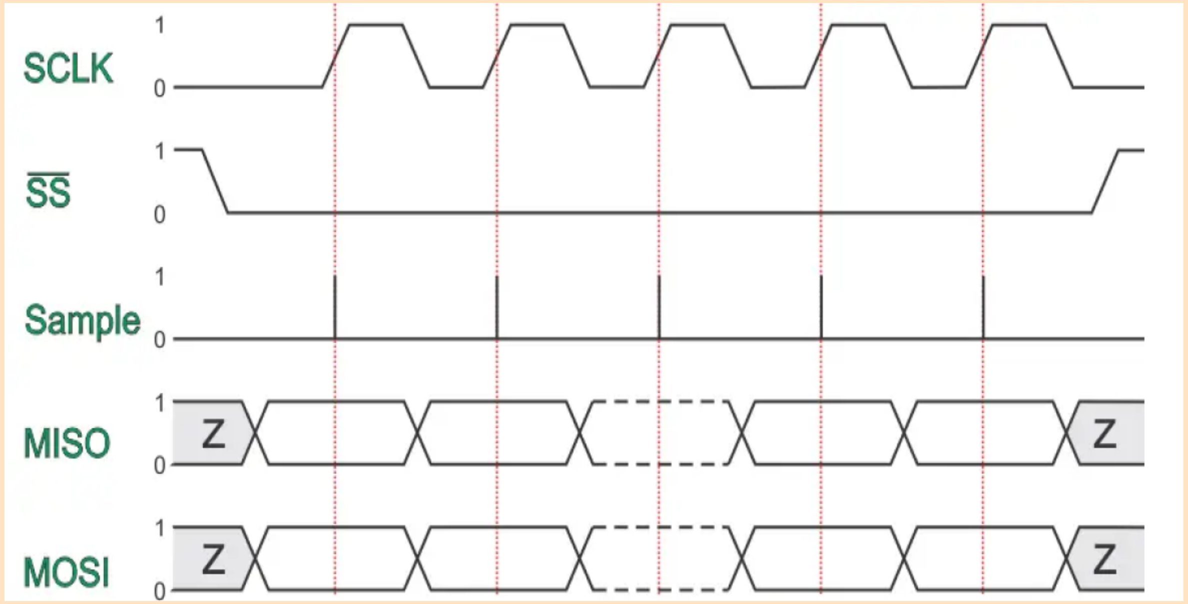
- The CPHA bit selects the clock phase. Depending on the CPHA bit, the rising or falling clock edge is used to sample and/or shift the data.

- The main must select the clock polarity and clock phase, as per the requirement of the Salve .

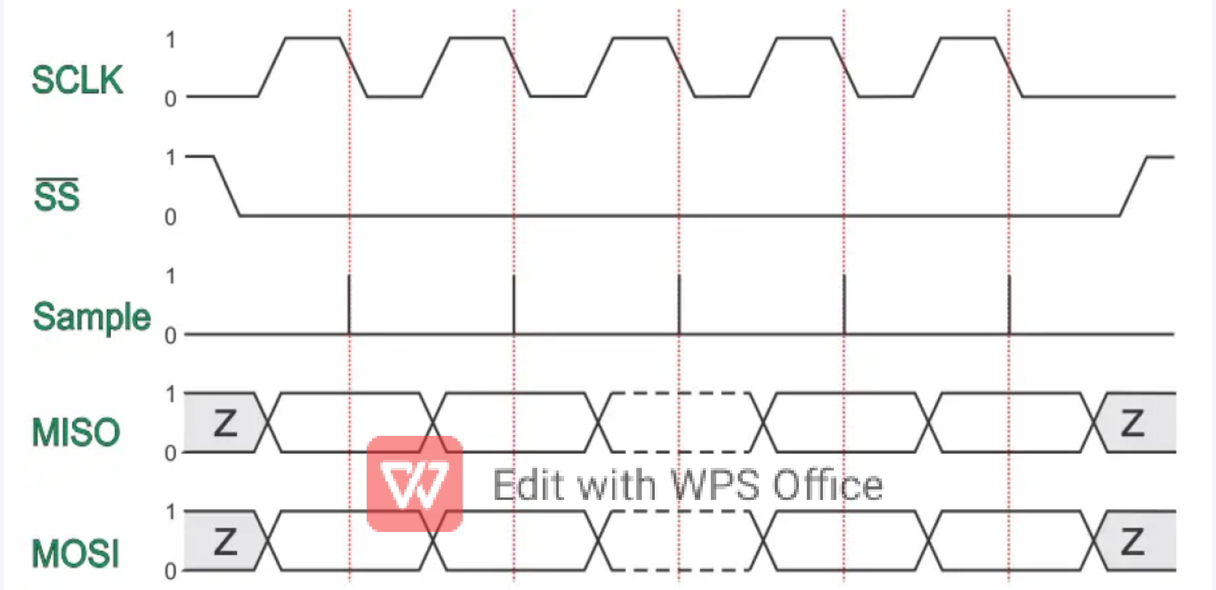
SPI Modes with CPOL and CPHA

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge
3	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge

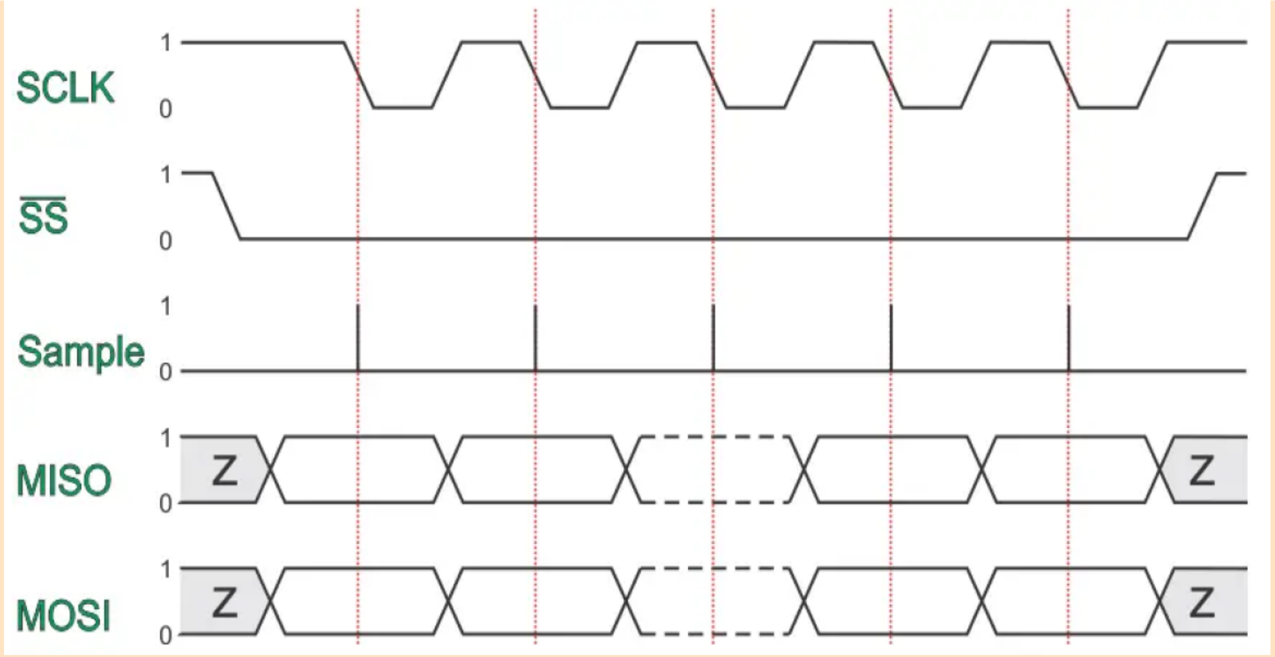
MODE- 0


SCLK
1
0
SS
1
0
Sample
1
0
MISO
1
0
Z
Z
MOSI
1
0
Z
Z

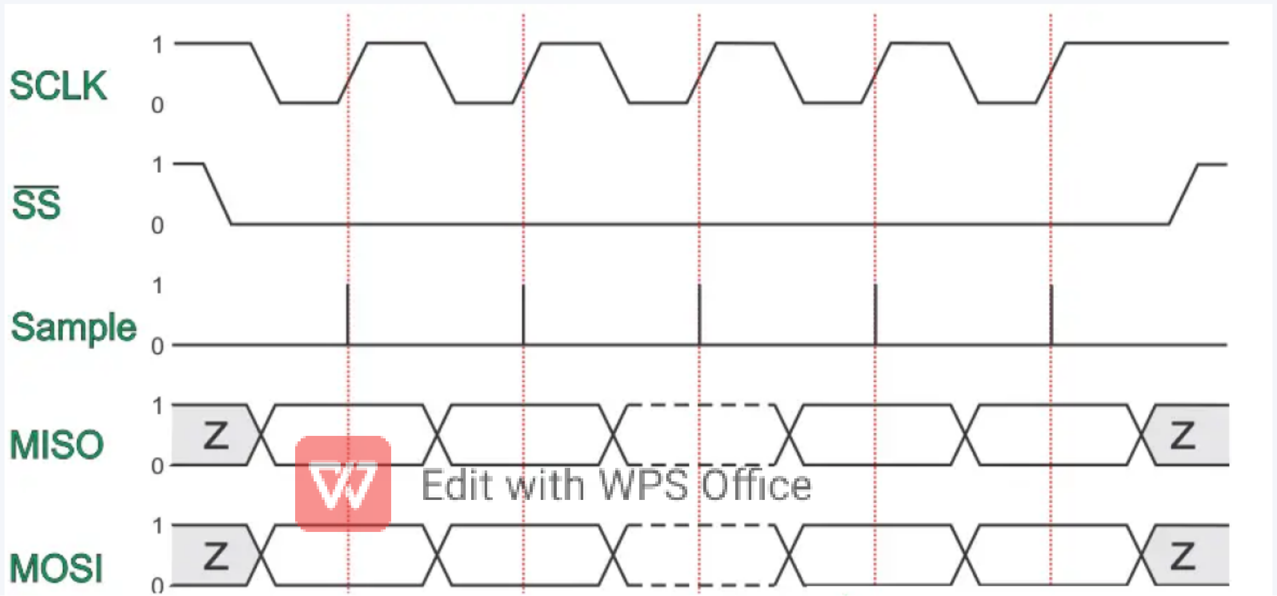
MODE- 1


SCLK
1
0
SS
1
0
Sample
1
0
MISO
1
0
Z
Z
MOSI
1
0
Z
Z

MODE- 2


SCLK
1
0
SS
1
0
Sample
1
0
MISO
1
0
Z
Z
MOSI
1
0
Z
Z

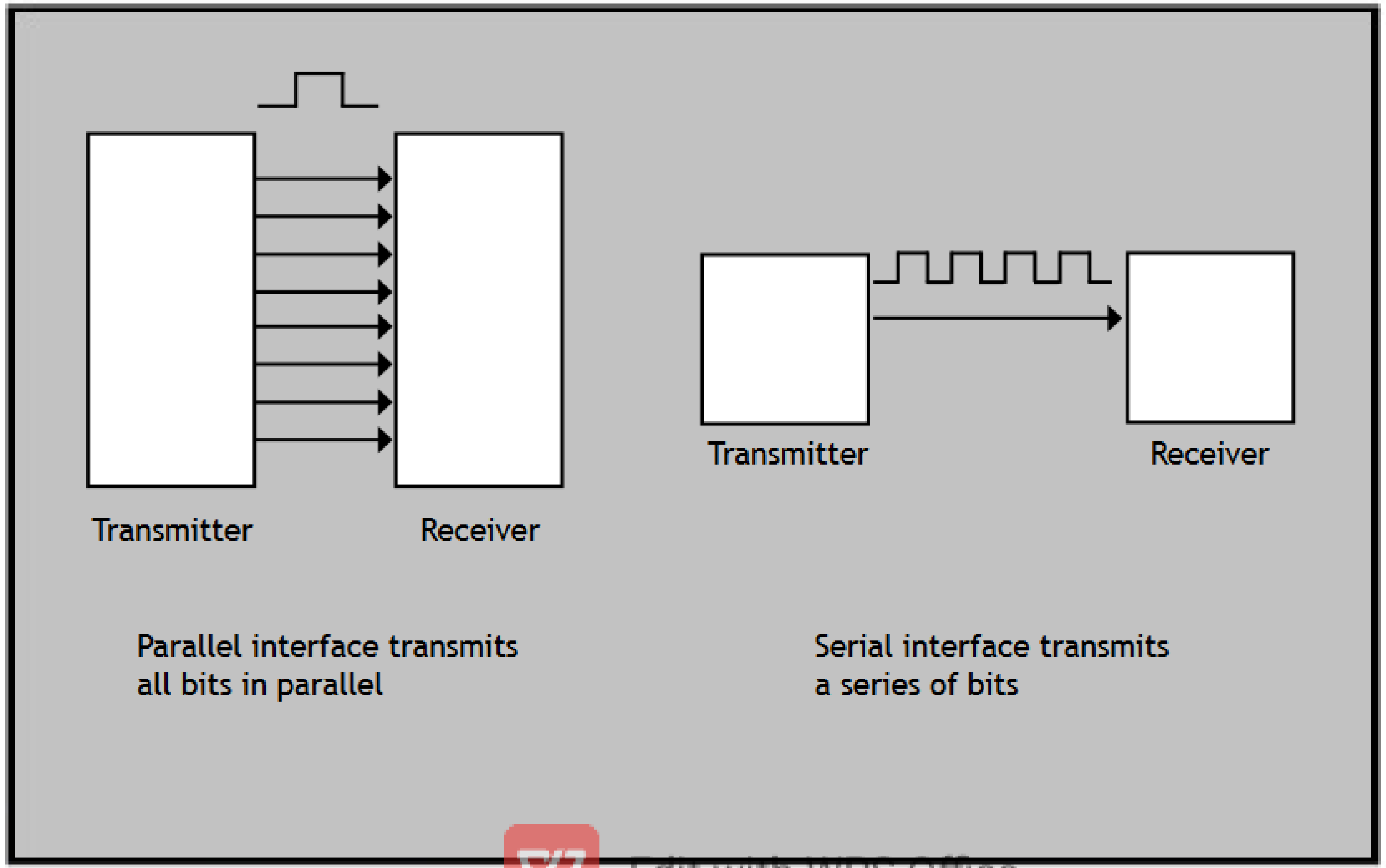
MODE- 3


SCLK
1
0
SS
1
0
Sample
1
0
MISO
1
0
Z
Z
MOSI
1
0
Z
Z

I2C	SPI
• It supports multi masters.	• It supports only one master.
• It needs a maximum of 2 wires.	• It needs a minimum of 4 wires and the maximum is based on the number of slaves.
• It is half-duplex communication as it has only one wire for Tx and Rx.	• It is full-duplex communication as it has separate wires for Rx and Tx.
• It is slower than the SPI.	• It is faster than the I2C.
• I2C has an acknowledgment mechanism.	• This doesn't have an acknowledgment mechanism.
• In this protocol, the slave has to have a unique address.	• In this protocol, the slave doesn't need an address. Master will use the SS/CS wire to select the slaves.
• Due to Start, Stop, Address bits, it has overhead.	• But it will directly start the data transfer. Start and Stop are not required. So, no overhead here.
• I2C is better for long-distance.	• SPI is better for a short distance.

On board communication interfaces

Parallel interface Bus


Transmitter
Receiver
Transmitter
Receiver
Parallel interface transmits all bits in parallel
Serial interface transmits a series of bits

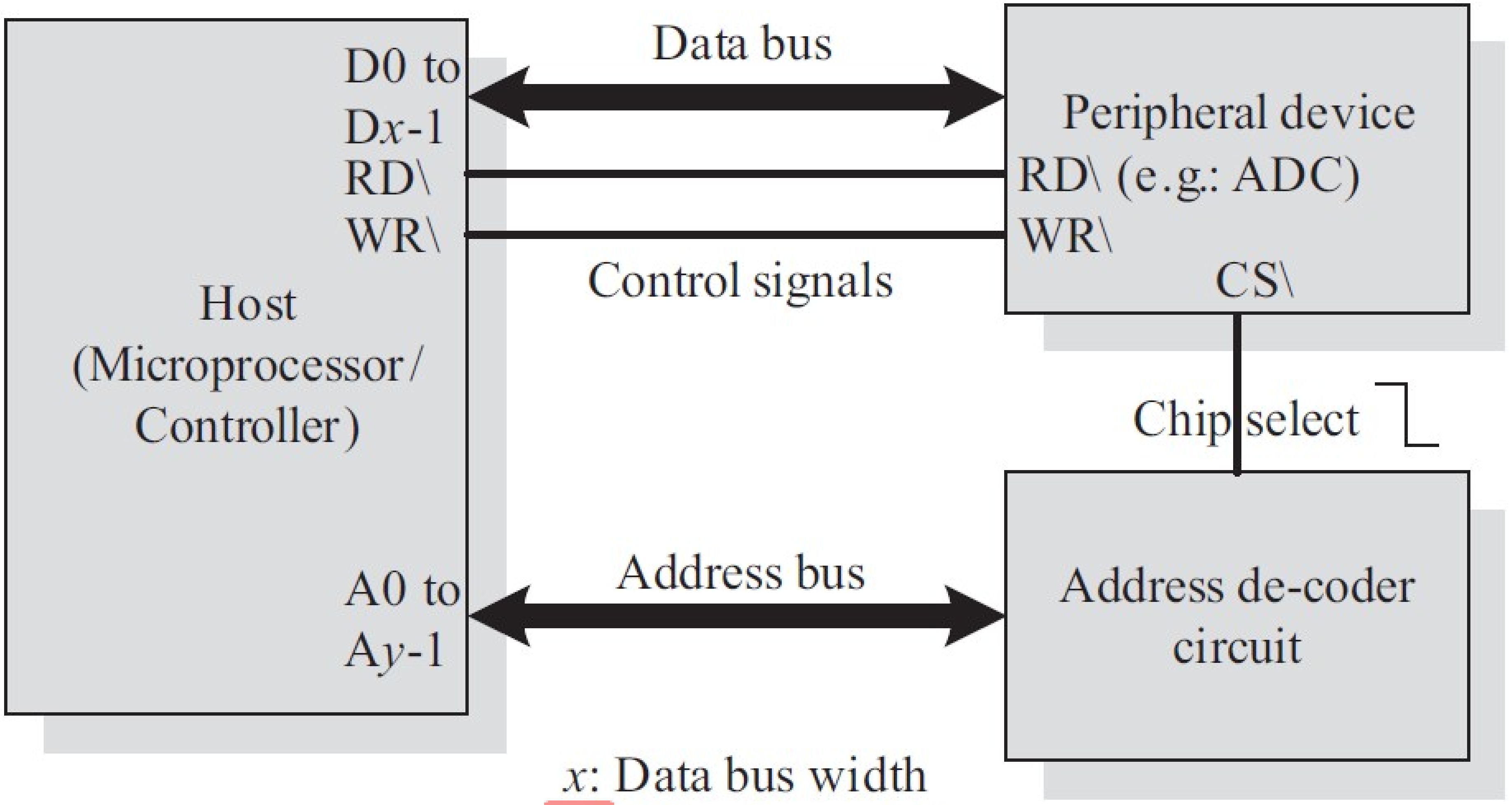
Parallel interface Bus

On-board parallel interface is normally used for communicating with peripheral devices which are memory mapped

- The host processor/controller of the embedded system contains a parallel bus
- Devices which supports parallel bus can be directly connected to this bus system
- The communication through the parallel bus is controlled by the control signal interface between the device and the host

  1. **Read/Write signal**
  2. **Device select signal**

- Device becomes active only when Device select line is asserted by the host processor
- The direction of data transfer is controlled using the control signal lines '**Read**' and '**Write**'
- The device is normally memory mapped to the ***host processor and a range of address*** is assigned to it
- An ***address decoder*** circuit is used for generating the chip select signal for the device
- When the address selected by the processor is within the range assigned for the device, the decoder circuit activates the chip select line and thereby the device becomes active



x: Data bus width
y: Address bus width