Embedded System Communication Protocols

- ➔ Communication protocols are a set of rules that control how two or more devices or systems share information through a physical connection. These rules include the meaning of data, the format in which it is sent, and the timing needed for proper communication.
- ➔ Protocols can be implemented using software, hardware, or a mix of both, which is common in embedded systems. Different protocols are used for different purposes, depending on the system's needs.
- ➔ Embedded systems rely on communication protocols because they include various sensors and peripherals connected to a microcontroller. To exchange data between these components, specific protocols are required.
- ➔ Some common communication protocols used in embedded systems are UART, USART, SPI, I2C, CAN, LIN, USB, and MODBUS. Each of these follows its own set of rules for data transmission.

Embedded System Communication Protocols

Inter – System Protocols

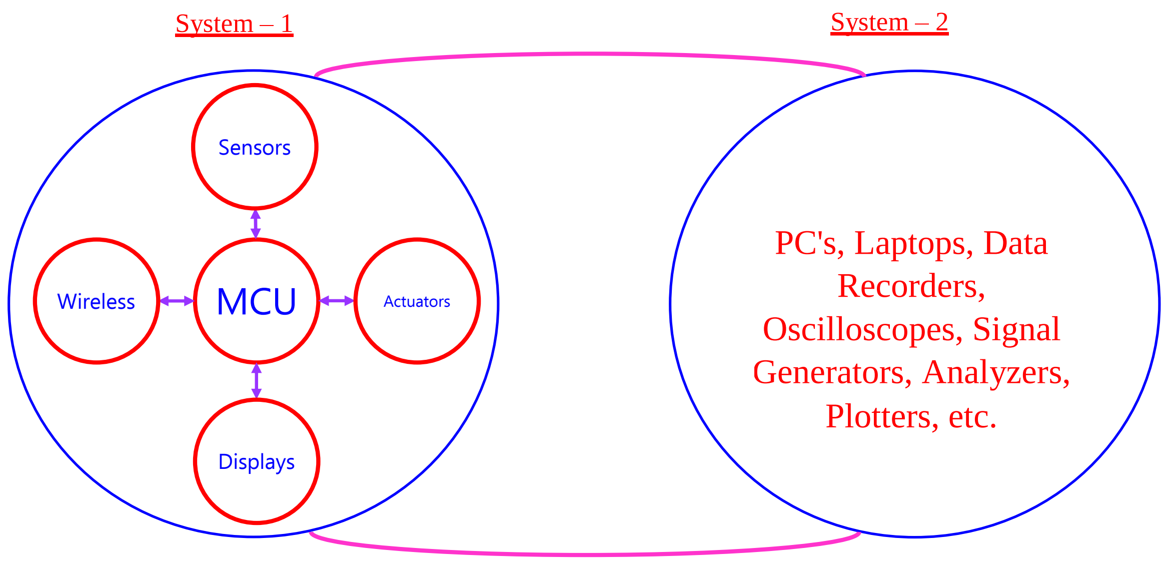
These protocols govern the communication between microcontroller and other devices like PC, laptop, recorders, analyzers, etc.

For Example – UART, USART, USB, etc.

Intra – System Protocols

These protocols govern the communication between microcontroller and sensors or any other peripherals interfaced with it.

For Example – SPI, I2C, CAN, LIN, MODBUS, etc.


System – 1
System – 2
Sensors
Wireless
MCU
Actuators
Displays
PC's, Laptops, Data Recorders, Oscilloscopes, Signal Generators, Analyzers, Plotters, etc.

In the diagram above, the pink lines show Inter-System Protocols, which are used for communication between System-1 and System-2.

The purple lines indicate Intra-System Protocols, which handle communication between different components within System-1.

● Common Terms in Embedded System Communication Protocols

1. **Master:** It is the device that controls the entire communication process. It always starts the communication by generating the clock signal and acts as the main controlling unit.
2. **Slave:** It is the device that follows commands from the master and carries out the required actions. It also receives the clock signal from the master.
3. **Full – Duplex Communication Protocol:** This protocol uses separate data lines for sending and receiving information between the master and the slave.
4. **Half – Duplex Communication Protocol:** This protocol uses a single data line to send and receive information between the master and the slave.

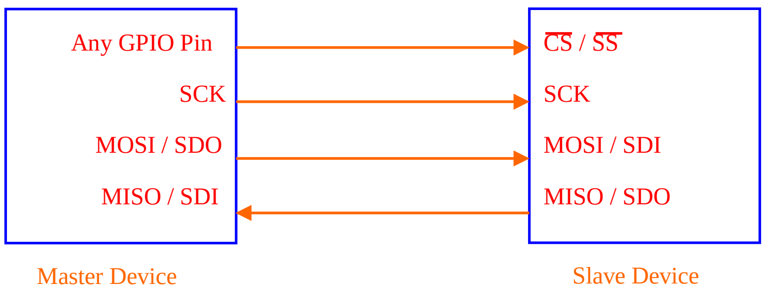
SPI and I2C are synchronous communication protocols because they need a clock signal for data transfer between the master and slave devices.

● SPI Protocol

➔ SPI, which stands for Serial Peripheral Interface, is one of the most basic communication protocols used in embedded systems. It was developed by Motorola in the early 1980s.

➔ Many sensors and peripherals use this protocol, including SD card readers, flash memories, RFID readers, graphical displays, DACs, ADCs, SRAMs, accelerometers, and special driver ICs.

➔ The SPI connection between the master and slave devices can be established in two ways: 3-wire interface and 4-wire interface. The 4-wire interface is more commonly used and is shown below.



➔ **4-Wire Interface Explanation:** In the 4-wire interface, both the master and slave devices require four pins for communication. These pins are described below:

I. **Chip Select / Slave Select ($\overline{CS}$ / $\overline{SS}$) pin:** This pin allows the master device to choose which slave device it wants to communicate with. The master controls this pin using one of its regular GPIO pins.
The $\overline{CS}$ / $\overline{SS}$ pin is active low, meaning:
- The slave is selected when the master sends a 0 (LOW) signal.
- The slave is unselected when the master sends a 1 (HIGH) signal.

In this setup, the master's GPIO pin works as an output, while the slave's $\overline{CS}$ / $\overline{SS}$ pin functions as an input.

II. **Serial Clock (SCK) pin:** The Serial Clock (SCK) pin is used to synchronize communication between the master and slave devices.
- Master: Acts as an output, generating the clock signal.
- Slave: Acts as an input, receiving the clock signal.

The master device always provides the clock signal at the required frequency to communicate with the slave. The correct clock frequency can be found in the technical documents of the slave device. The entire data transfer process between the master and slave is synchronized with this clock signal.

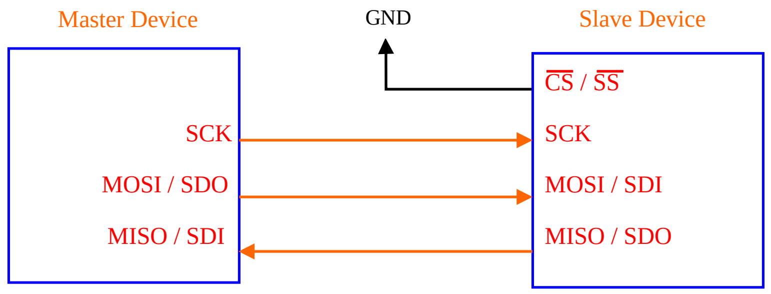
III. **Master-Out Slave-In (MOSI) pin:** The MOSI pin is used for data transfer from the master to the slave.
- Master: Sends data (acts as an output).
- Slave: Receives data (acts as an input).

In older naming conventions:
- This pin is called SDO (Serial Data Out) on the master.
- It is called SDI (Serial Data In) on the slave.

IV. **Master-In Slave-Out (MISO) pin:** The MISO pin is used for data transfer from the slave to the master.
- Slave: Sends data (acts as an output).
- Master: Receives data (acts as an input).

In older naming conventions:
- This pin is called SDO (Serial Data Out) on the slave.
- It is called SDI (Serial Data In) on the master.

➔ **3-Wire Interface Explanation:** In the 3-wire interface, the connections for SCK (Serial Clock), MOSI (Master-Out Slave-In), and MISO (Master-In Slave-Out) remain unchanged. However, instead of being controlled by the master, the $\overline{CS}$ / $\overline{SS}$ (Chip Select / Slave Select) pin of the slave is directly connected to ground, as shown below:



Limitations of the 3-Wire Interface in SPI Communication:
The 3-wire interface is generally not recommended for SPI communication. It is only suitable when:

- The master is connected to a single slave device.
- Continuous communication between the master and slave is required.

If multiple slaves are connected, the 3-wire interface causes all slaves to be selected at once, but the master can only communicate with one slave at a time. Due to this limitation, the 4-wire interface is highly recommended for SPI-based communication.

SDI and SDO Naming Convention in SPI:
The terms SDI (Serial Data In) and SDO (Serial Data Out) were used in older SPI conventions but are rarely used today because they can be confusing.

SDI pin:

- MISO (Master-In Slave-Out) from the master's perspective.
- MOSI (Master-Out Slave-In) from the slave's perspective.

SDO pin:

- MOSI (Master-Out Slave-In) from the master's perspective.
- MISO (Master-In Slave-Out) from the slave's perspective.

Since SDI and SDO refer to different lines depending on whether it's the master or slave device, the MISO and MOSI terms are now preferred to avoid confusion.

➔ **SPI Protocol: Full-Duplex and Synchronous Communication**
From the discussion above, we can conclude that SPI is a full-duplex communication protocol because it has two separate data lines:

- MOSI (Master-Out Slave-In) for data transfer from master to slave.
- MISO (Master-In Slave-Out) for data transfer from slave to master.

Additionally, SPI is a synchronous protocol, as the data exchange between the master and slave is completely synchronized with the clock signal generated by the master.

➔ **SPI Clock Frequency and Data Transfer Speed**
The SPI clock frequency does not have a fixed ideal range, but its upper limit is always half of the embedded system’s operating frequency.

- The typical SPI data transfer speed is around 10 Mbps.
- Advanced versions of SPI, such as Dual-SPI, Quad-SPI, Octo-SPI, and Extended-SPI can achieve speeds of up to 160 Mbps.

➔ **High-Speed Data Transfer in SPI Protocol**
Since the transmitting device can directly control the logic levels of the data and clock lines, switching between high and low states happens quickly. This minimizes the delay in signal changes, allowing SPI to achieve higher data transfer rates compared to other embedded communication protocols.

➔ Also, the data transfer speed depends on the number of bits transferred during each SPI clock cycle. For example:

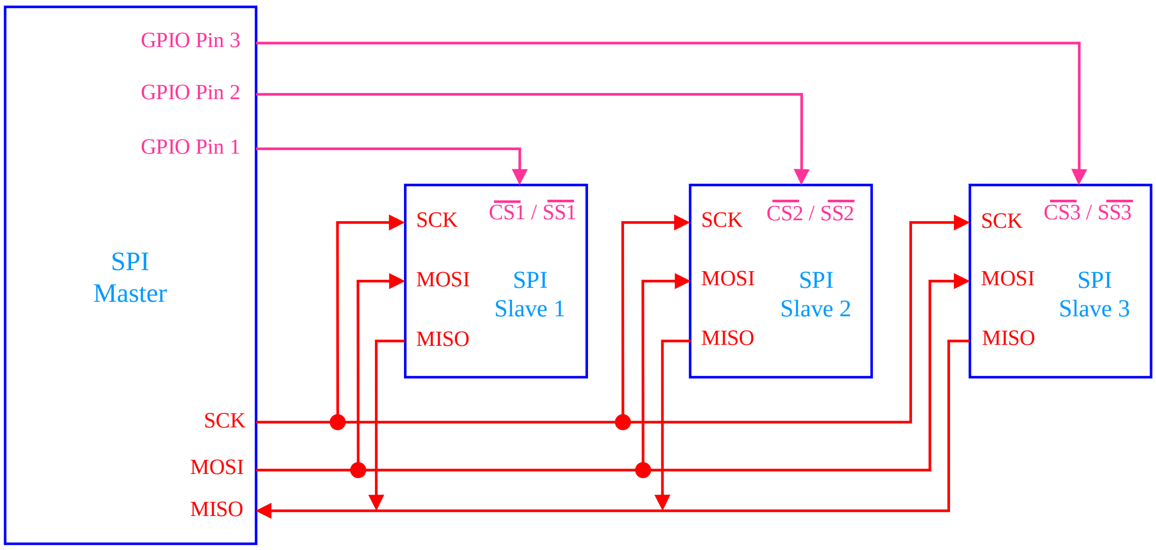
- **Standard SPI Interface** = 1 bit per SPI clock cycle i.e., 1 byte is transferred in 8 SPI clock cycles.
- **Dual – SPI Interface** = 2 bits per SPI clock cycle i.e., 1 byte is transferred in 4 SPI clock cycles.
- **Quad – SPI Interface** = 4 bits per SPI clock cycle i.e., 1 byte is transferred in 2 SPI clock cycles.
- **Octo – SPI Interface** = 8 bits per SPI clock cycle i.e., 1 byte is transferred in 1 SPI clock cycle.
- **Extended – SPI Interface** = 16 bits per SPI clock cycle i.e., 2 bytes are transferred in 1 SPI clock cycle.

➔ **SPI protocol** supports **single master and multiple slave configuration** only i.e., there can be only one master device and multiple slave devices can be interfaced with it and this can be achieved using 2 methods:

- Independent Slave Method
- Daisy – Chain Method

Independent Slave Method in SPI

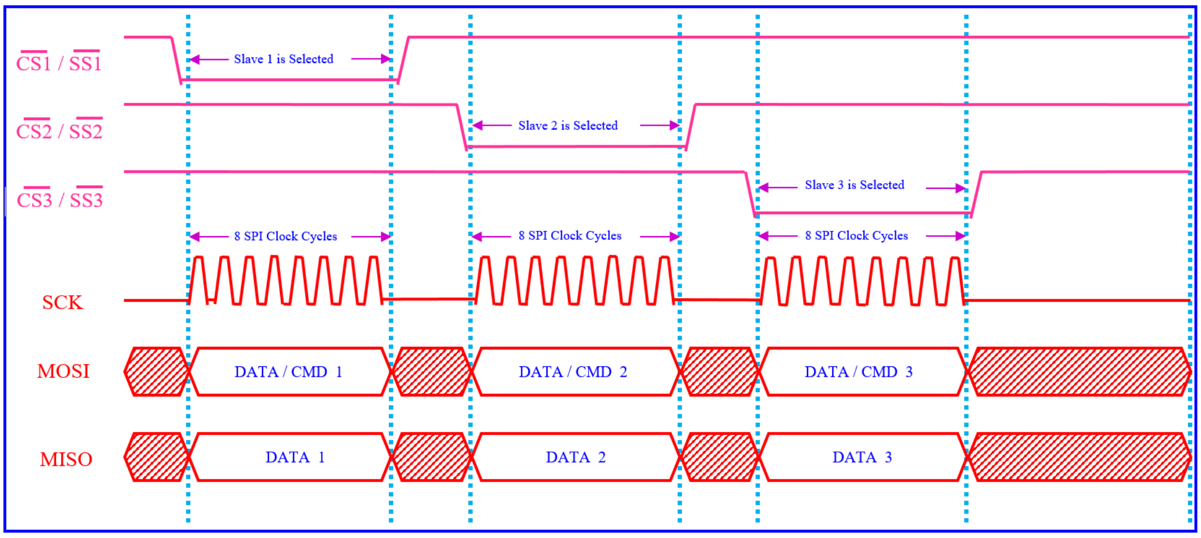
In this method, each slave is connected to the master separately, with its own Chip Select ($\overline{CS}$ / $\overline{SS}$) pin, while the SCK, MOSI, and MISO connections are shared among all slaves, as shown below:



➔ From the diagram, it is clear that the number of Chip Select ($\overline{CS}$ / $\overline{SS}$) lines corresponds to the number of connected slave devices, while the SCK, MOSI, and MISO lines are shared among all the slaves.

➔ Here, the data takes approximately the same amount of time to reach all the slave devices since the data lines have been made common to all of them. For example, if standard SPI protocol is used, then all the slave devices will receive their corresponding data or command in 8 SPI clock cycles.

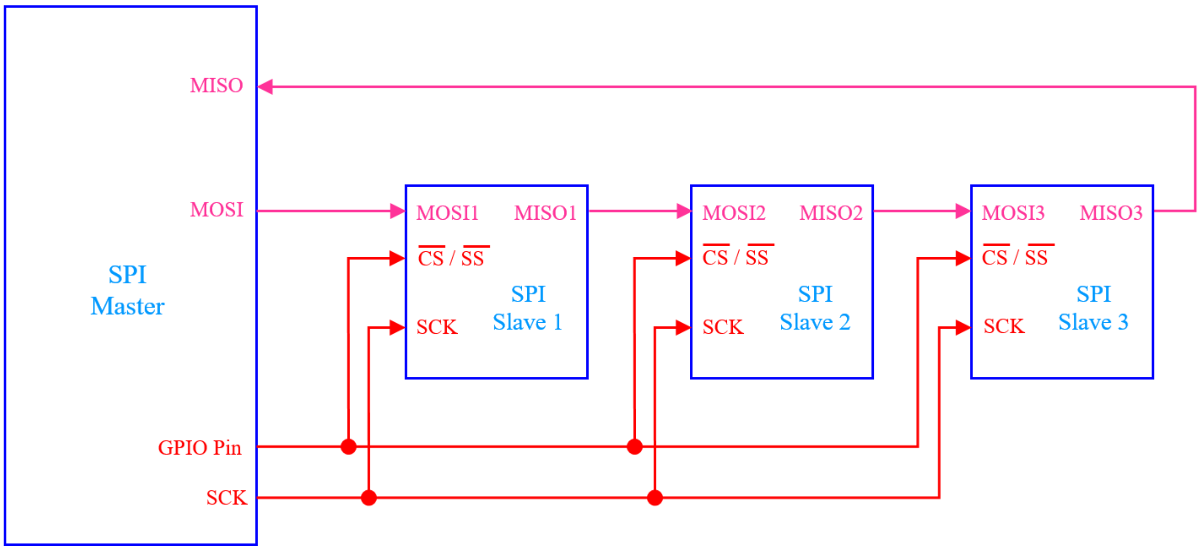
➔ Since each slave device has its own Chip Select ($\overline{CS}$ / $\overline{SS}$) line, the master can communicate with each slave independently. However, activating multiple $\overline{CS}$ / $\overline{SS}$ lines at the same time should be avoided, as it can cause data corruption on the MISO line, making it impossible for the master to determine which slave is transmitting data.



➔ The main drawback of this method is that the number of slave devices that can be connected to the master is limited. As more slaves are added, the number of Chip Select ($\overline{CS}$ / $\overline{SS}$) lines increases, leading to greater circuit complexity and potential exhaustion of the master's GPIO pins.

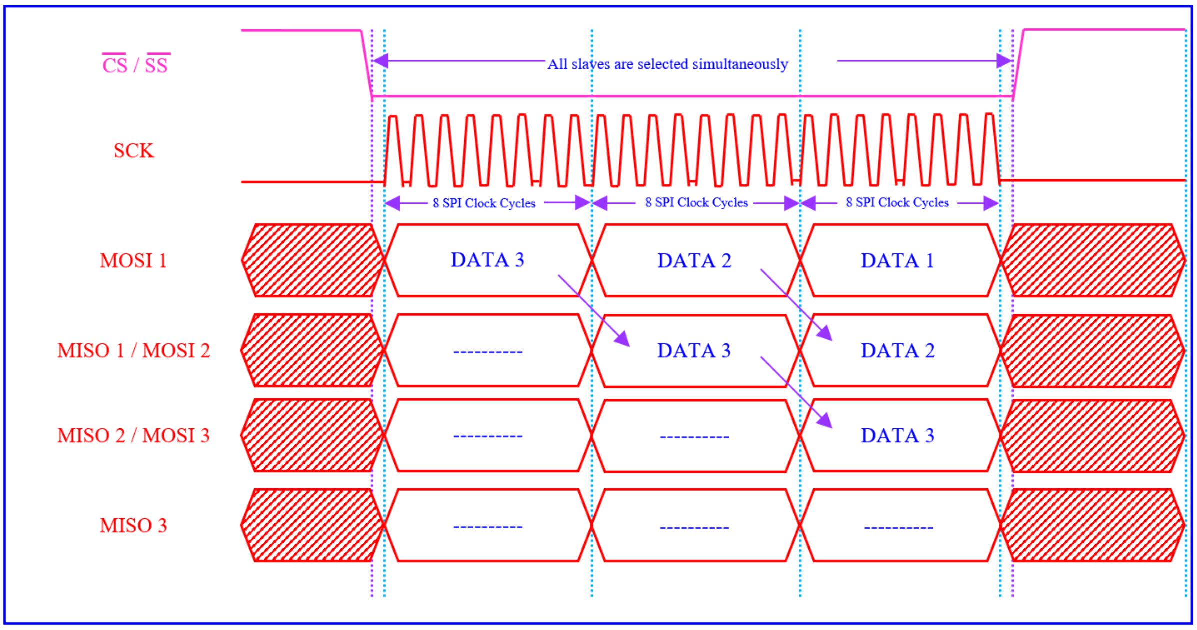
● Daisy-Chain Method in SPI

In this method, the Chip Select ($\overline{CS}$ / $\overline{SS}$) lines and Serial Clock (SCK) lines of all slaves are connected parallelly with each slaves, while the data transfer lines are linked in a cascaded manner, as shown in the diagram below:



➔ Here, all the slaves are selected and they receive the clock signal at the same time and the data is sent from master to 1st slave and then 1st slave sends the data to 2nd slave and so on but the last slave sends the data back to the master.

➔ Since the data propagates from one slave to another, all the slave devices will not receive the data or command in the same number of clock cycles. For example, if standard SPI protocol is used, then 1st slave will receive data after 8 SPI clock cycles, 2nd slave will receive data after 16 SPI clock cycles and 3rd slave will receive data after 24 SPI clock cycles.



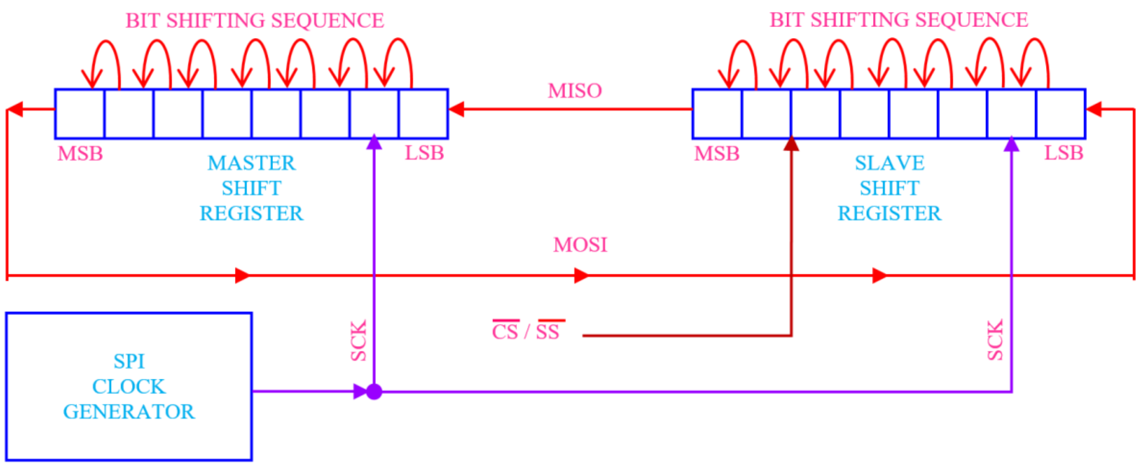
➔ One advantage of this method is that the circuit complexity will be much less as compared to independent slave method because the chip / slave select line is common to all the slaves.

➔ However, the number of slaves that can be connected using this method is also limited. As the number of slaves increases, the data propagation time increases depending on the slave's position in the chain, which is a disadvantage. Additionally, very few devices support this method in practice.

● Step-by-Step Working of SPI Protocol

1. **Identify Master and Slave –** Decide which devices will act as the master and slave.
2. **Check Slave Specifications –** Refer to the technical document of the slave device to find the required SPI clock phase, clock polarity, and clock frequency.
3. **Configure the Master –** Set up the master device to generate the clock signal based on the required parameters.
4. **Select the Slave –** Activate the slave by setting the Chip Select ($\overline{CS}$ / $\overline{SS}$) line to logic 0.
5. **Send Data –** The master transmits the command or data to the slave.
6. **Receive Data –** The slave sends data back to the master, which stores it for further processing.
7. **Deselect the Slave –** The master deactivates the slave by setting the Chip Select ($\overline{CS}$ / $\overline{SS}$) line to logic 1.

The data is transferred between master and slave in a serial manner using shift registers as shown in the following figure:



- ➔ Here, the data is shifted out from the master's shift register and shifted into the slave's shift register, one bit at a time per clock pulse of the SPI clock generator.
- ➔ As per the direction of the red arrows in the above diagram, the MSB of the data byte will be shifted out 1st and then the subsequent bits (conventional method). If the LSB of the data byte would be shifted out 1st and then the subsequent bits, then the direction of the red arrows will get reversed.
- ➔ If there are multiple slaves present, then the overall SPI clock frequency will be decided by the slave device that has the lowest value of SPI clock frequency.
- ➔ The datasheet of the slave device contains information about the various commands that master needs to send to the slave for data extraction.
- ➔ When data transfer begins either from master to slave or vice – versa, the master device will decide whether the MSB or the LSB of the data byte will be shifted out 1st from the shift registers and the configurations to be done for the same will be available in the master's datasheet.

● SPI Modes and the role of Clock Phase and Polarity

- ➔ Proper configuration of the SPI clock frequency is essential for successful communication between the master and slave devices. However, the clock frequency alone is not the only factor that determines successful data exchange. Other parameters, such as clock phase and clock polarity, also play a crucial role.
- ➔ Clock phase and clock polarity are crucial for establishing successful SPI communication between the master and slave. However, these two parameters are often overlooked during programming, which can lead to communication errors.
- ➔ Clock Polarity (CPOL) determines the idle state of the SPI clock, while Clock Phase (CPHA) defines which clock edge (rising or falling) will be used for sampling and shifting data during communication.
- ➔ The idle state of the clock refers to the logic level on the SCK line when the Chip Select ($\overline{CS}$ / $\overline{SS}$) line is at logic 1, meaning the slave is not selected.

➔ Clock Polarity (CPOL) and Clock Phase (CPHA)

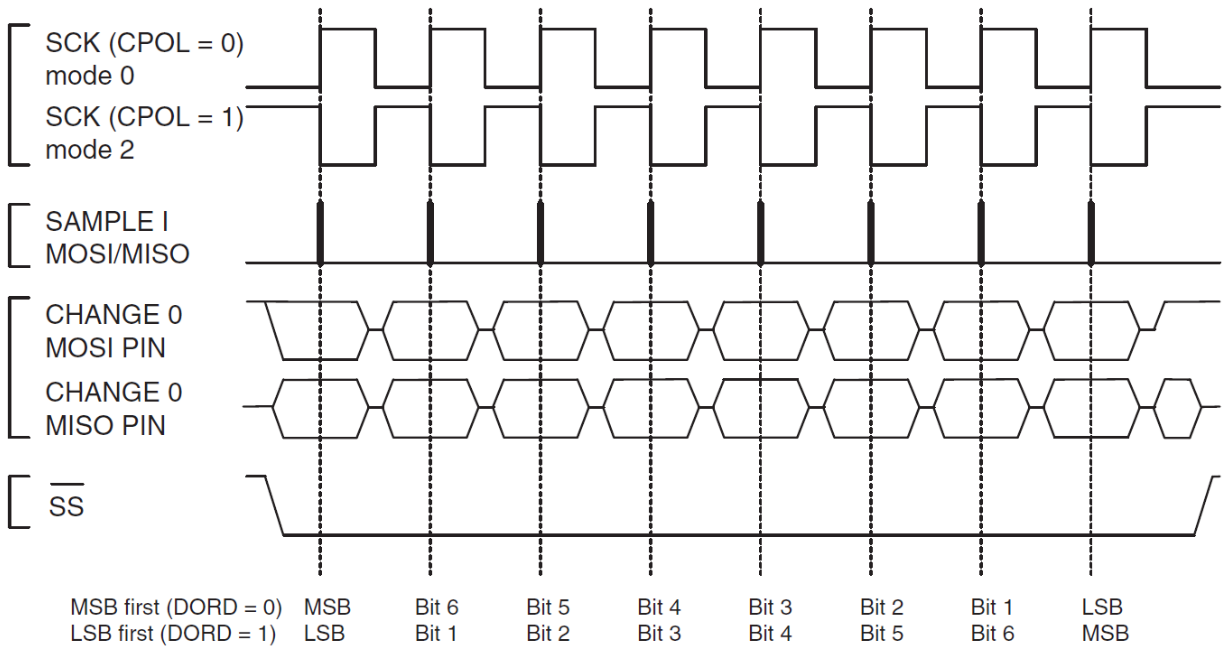
$$CPOL = \begin{cases} 0 \rightarrow \mathit{Idle\ state\ of\ SPI\ clock\ is\ logic}\ 0 \\ 1 \rightarrow \mathit{Idle\ state\ of\ SPI\ clock\ is\ logic}\ 1 \end{cases}$$

$$CPHA = \begin{cases} 0 \rightarrow Data\ is\ \mathbf{Read\ at\ Leading\ Edge}\ and\ data\ is\ \mathbf{Shifted\ Out\ at\ Trailing\ Edge}\ of\ \mathbf{SCK} \\ 1 \rightarrow Data\ is\ \mathbf{Read\ at\ Trailing\ Edge}\ and\ data\ is\ \mathbf{Shifted\ out\ at\ Leading\ Edge}\ of\ \mathbf{SCK} \end{cases}$$

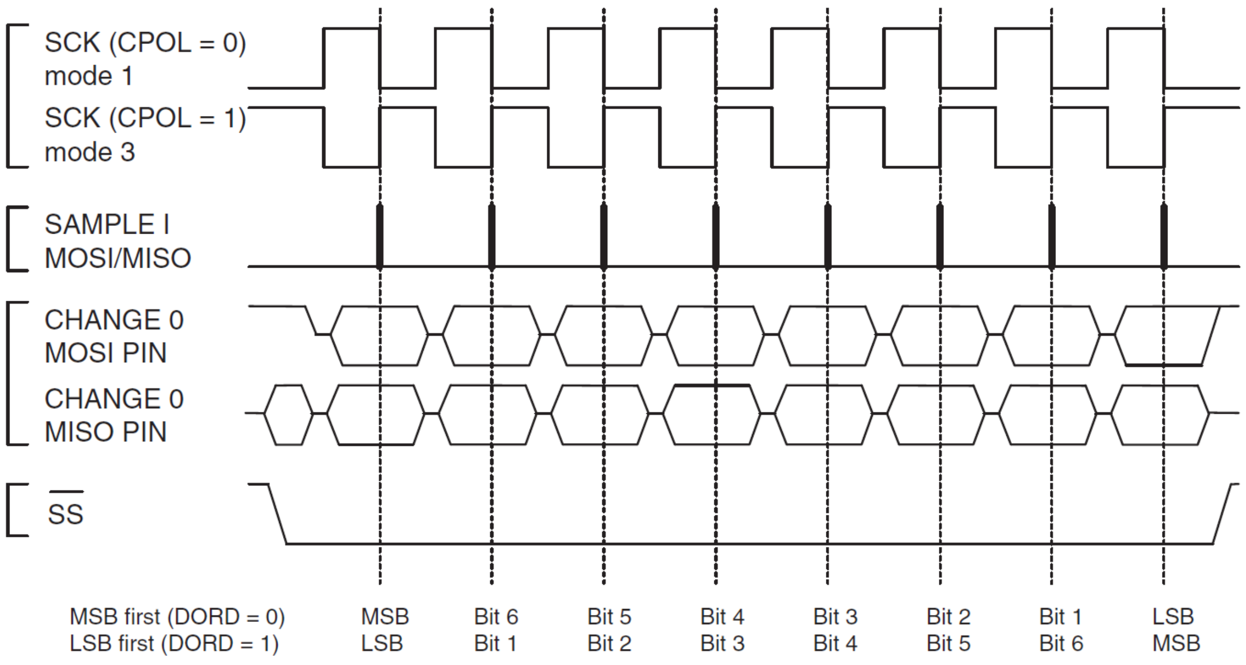
Note:

- ***Read= Sample*** and ***Shift Out= Change= Setup***
- **Always Read the data first then Shift Out. That is why for CPHA=1, the leading edge of the first clock is ignored.**

SPI Transfer Format with CPHA = 0



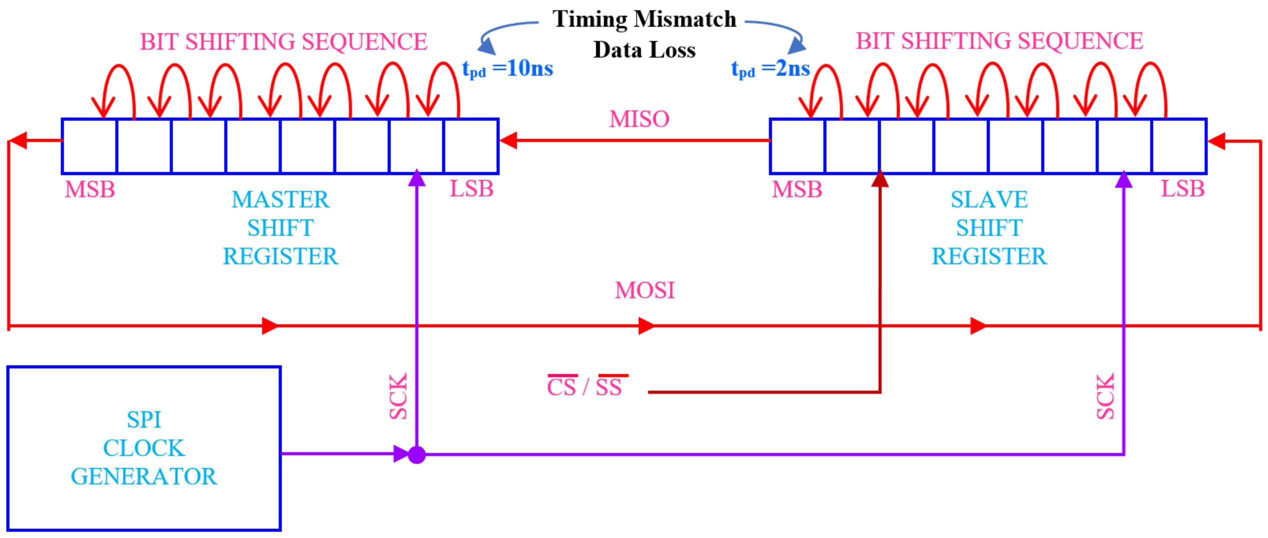
SPI Transfer Format with CPHA = 1



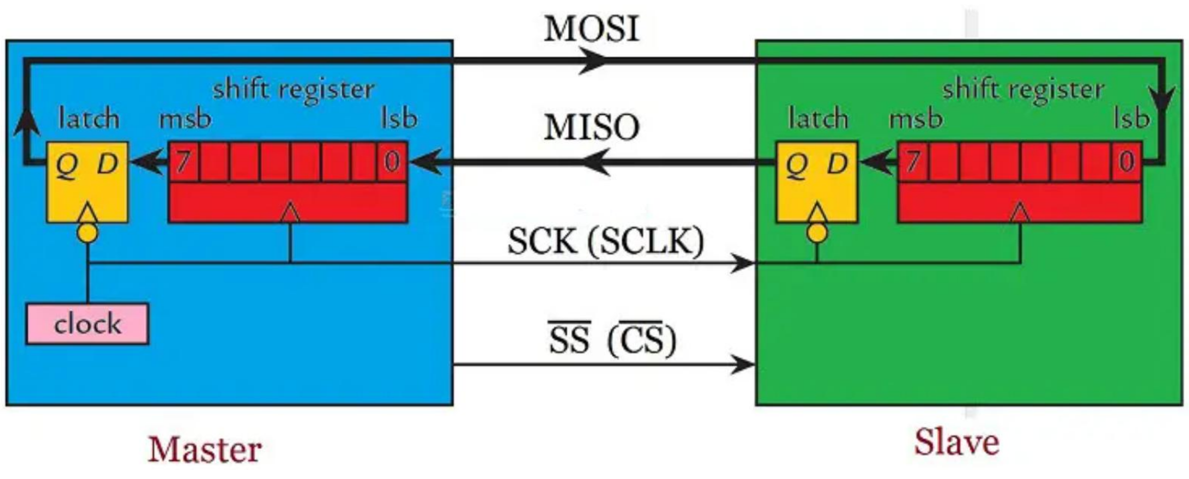
- **In SPI there are Master Shift Register and Slave Shift Register then why are Data Read and Data Shift Out required? Is it not possible to use like simple shift register cascaded together?**

Analogy: Think of a shift register as a conveyor belt (data being shifted), and the sampling process as a worker picking items off the belt. Even if the conveyor moves smoothly, the worker needs to know the exact timing to pick up an item (sampling). Without this timing, the worker might grab an item before fully delivered of the previous one.

Timing mismatches in Semiconductor: Not all semiconductor devices operate at the same speed. While the shift registers shift data in and out based on the clock signal, relying purely on this for data transmission is risky due to potential timing mismatches (Time Delay, Setup Time, Hold Time). That is why it is not possible to use like simple shift register cascaded together.



Another Latch is required with each shift register to overcome the above timing mismatches.



From the above image, the Master device includes a shift register, a data latch, and a clock generator to control data transfer. Similarly, the Slave device has a shift register and a data latch. Both shift registers are connected in a loop, allowing data to be exchanged between the Master and Slave simultaneously.

During the positive edge of the clock signal, both the Master and Slave read an input bit into the least significant bit (LSB) of their shift registers. During the negative edge, they shift out a bit from the most significant bit (MSB). This means that for every clock cycle, one bit is transferred in each direction—Master to Slave and Slave to Master. Since a byte consists of 8 bits, it takes 8 clock cycles to fully exchange one byte of data between the devices.
That is why Clock Polarity (CPOL) and Clock Phase (CPHA) come into the picture.

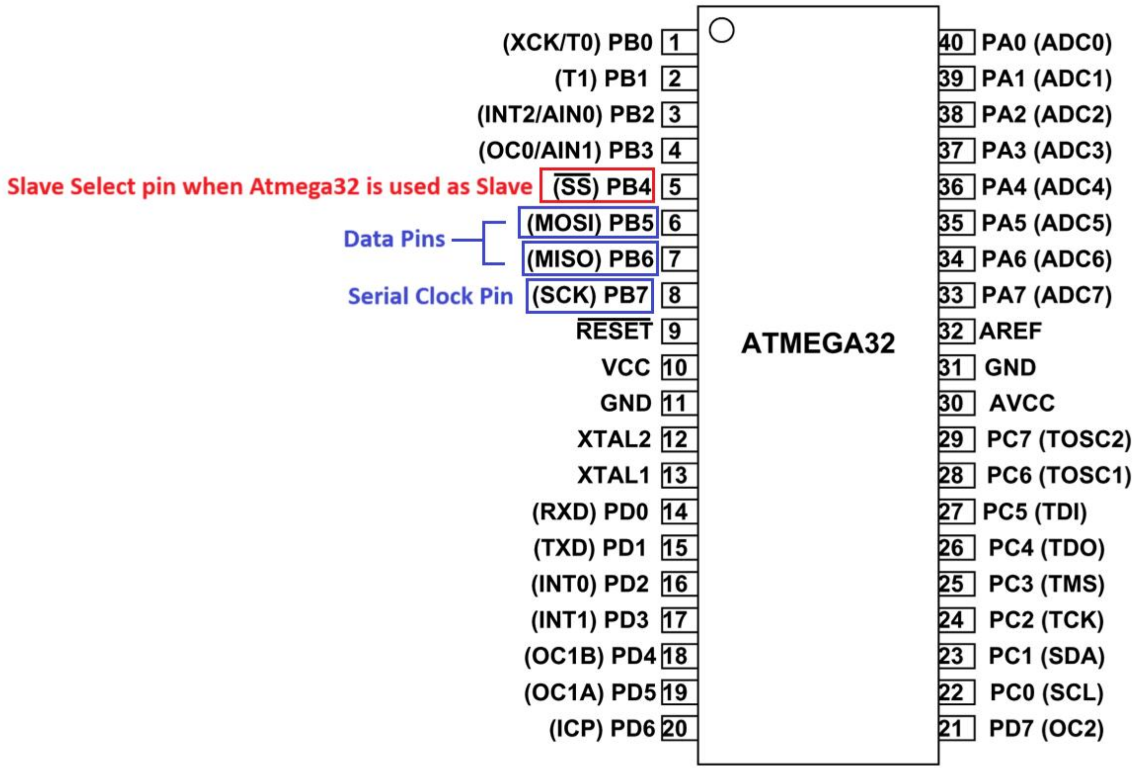
- ## Merits/ Advantages of SPI Protocol

➔ **Speed of data transfer** – Very high (10 Mbps to 100 Mbps or more).
➔ **Full-duplex communication** – Separate data lines for sending and receiving.
➔ **Simple and efficient** – No extra or unused information bits.
➔ **Single master device** – Prevents conflicts.
➔ **Low power consumption** – No need for extra hardware like pull-up resistors.
➔ **No slave addresses needed** – Uses a chip/slave select line.
➔ **Adjustable clock speed** – Master sets speed based on the slave device.
➔ **Flexible data length** – Can transfer any amount of data (limited by master and slave capacity).

- ## Demerits/ Disadvantages of SPI Protocol

➔ **Increased circuit complexity** – More slaves require individual chip/slave select lines.
➔ **Speed limitation** – Data transfer speed is limited by the slowest slave device.
➔ **Higher noise risk** – More wires lead to increased noise and crosstalk.
➔ **No acknowledgment** – No confirmation if data is received correctly.
➔ **Master-controlled communication** – Slaves cannot communicate with each other.
➔ **No error checking** – No built-in mechanism to detect transfer errors.
➔ **Shorter communication range** – Compared to UART and RS-485 protocols.
➔ **No official standards** – Used in application-specific implementations.

- ## Atmega32 Pins Associated with SPI Communication



- ## Simplex, Half-Duplex and Full-Duplex transmission

1. SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 – SPIE: SPI Interrupt Enable
This bit causes the SPI interrupt to be executed if the SPIF bit in the SPSR Register is set and if the global interrupt enable bit in SREG is set.

• Bit 6 – SPE: SPI Enable
When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

• Bit 5 – DORD: Data Order
When the DORD bit is written to one, the LSB of the data word is transmitted first. When the DORD bit is written to zero, the MSB of the data word is transmitted first.

• Bit 4 – MSTR: Master/Slave Select
This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If $\overline{SS}$ is configured as an input and is driven low while MSTR is set,

MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master mode.

• Bit 3 – CPOL: Clock Polarity

When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle. The CPOL functionality is summarized below:

Table: CPOL Functionality

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

• Bit 2 – CPHA: Clock Phase

The settings of the Clock Phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK. The CPHA functionality is summarized below:

Table: CPHA Functionality

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

• Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0

These two bits control the SCK rate of the device configured as a Master. SPR1 and SPR0 have no effect if the Atmega32 is configured as Slave. The relationship between SCK and the Oscillator Clock frequency f_{osc} is shown in the following table:

Table: Relationship Between SCK and the Oscillator Frequency

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

2. SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	**WCOL**	–	–	–	–	–	**SPI2X**	**SPSR**
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 – SPIF: SPI Interrupt Flag

When a serial transfer is complete, the SPIF Flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled. If SS is an input and is driven low when the SPI is in Master mode, this will also set the SPIF Flag. SPIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, the SPIF bit is cleared by first reading the SPI Status Register with SPIF set, then accessing the SPI Data Register (SPDR).

• Bit 6 – WCOL: Write COLlision Flag

The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) are cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

Do You Need to Check WCOL After a Successful Transmission (SPIF = 1)?

Answer: NO, You Don't Need to Check WCOL If SPIF = 1

If SPIF = 1, it means that the previous data transmission has successfully completed, and the SPI shift register is now ready for new data. In this case, WCOL will not be set, so you don't need to check or handle WCOL.

• Bit 5..1 – Reserved Bits

These bits are reserved bits in the ATmega32 and will always read as zero.

• Bit 0 – SPI2X: Double SPI Speed Bit

When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode. This means that **the minimum SCK period will be two CPU clock periods**. When the **SPI is configured as Slave**, the SPI is only **guaranteed to work at $f_{osc}/4$ or lower**.

3. SPI Data Register – SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							**LSB**	**SPDR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

The SPI Data Register is a read/write register used for data transfer between the Register File and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

• SPI Communication between two Atmega32 Microcontrollers

Experiment 1: One ATmega32 works as a Master, and the other as a Slave in SPI communication. Each microcontroller has a common cathode seven-segment display connected to PORTA. The Master stores digit values (0-9) in its flash memory and sends them to the Slave via SPI to display on its seven-segment display. When the Master sends a new digit, the previous digit is received back through MISO and shown on the Master's display. Use CPOL=0 and CPHA=0.

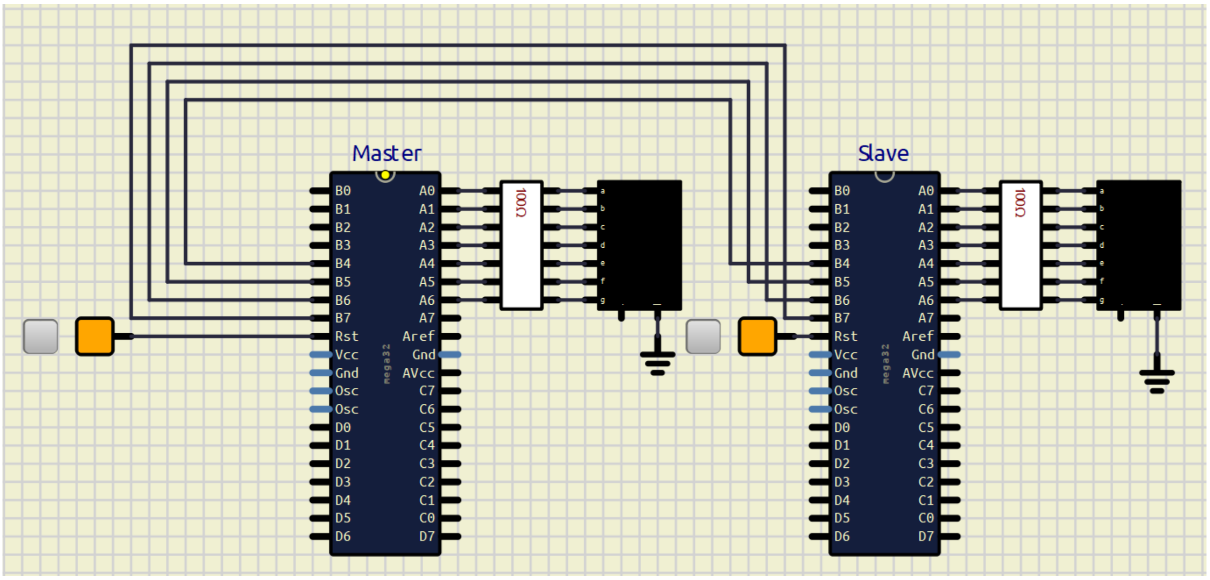


Fig: Circuit Diagram of Experiment 1

Master Code:

```
/* Atmega32 to Atmega32 SPI Communication */
// Master: Oscillator Frequency 1MHz

.INCLUDE "M32DEF.INC"
.ORG 0x0000

// Stack declaration
LDI R16,HIGH(RAMEND)
OUT SPH,R16
LDI R16,LOW(RAMEND)
OUT SPL,R16

// Data direction registers for 7 segment display
LDI R16,0XFF
OUT DDRA,R16
LDI R16,0X00
OUT PORTA,R16 // 7 segment initially turn off

//Data Direction Registers of SPI pins
SBI DDRB,PINB4 // Slave select pin
SBI PORTB,PINB4 // Slave not selected
SBI DDRB,PINB5 // MOSI Pin
CBI DDRB,PINB6 // MISO Pin
SBI PORTB,PINB6 // Internal Pull up resistor activated
SBI DDRB,PINB7 // SCK Pin

// SPI Control Register Settings
LDI R16,0x50
```

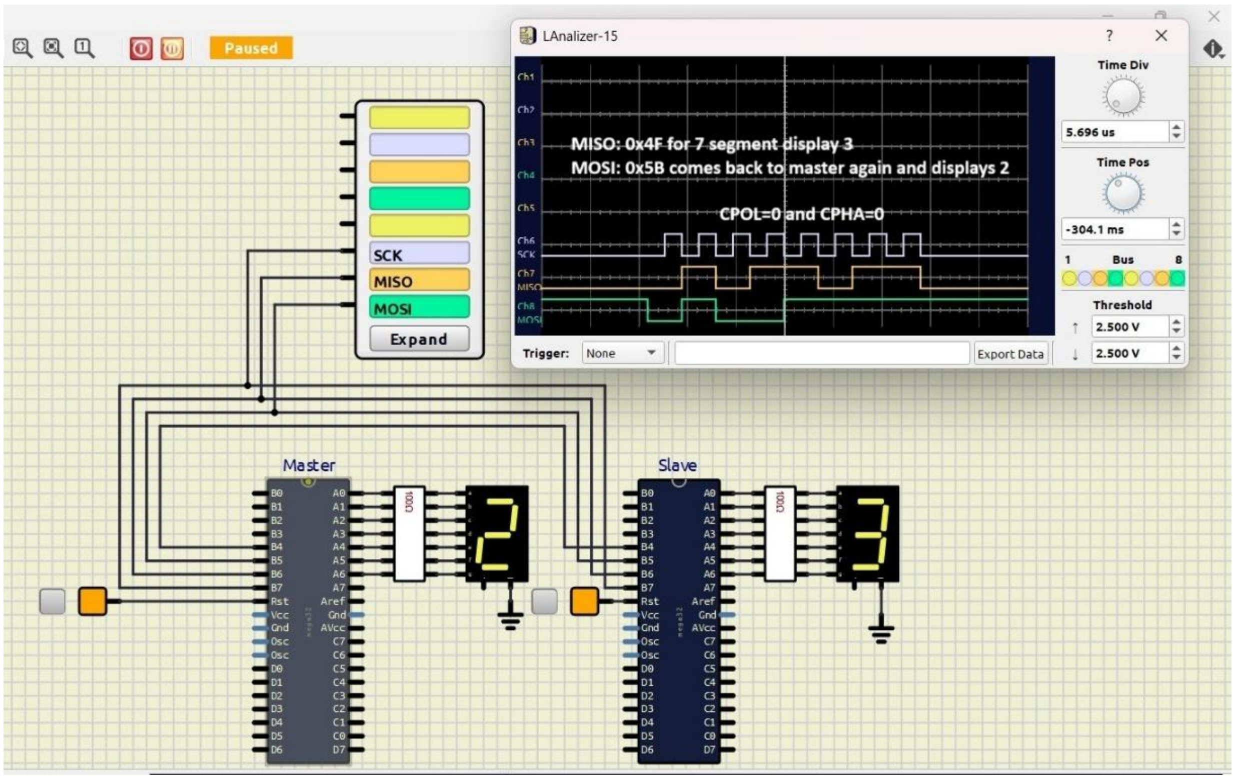
```
OUT SPCR,R16
//Select the slave(Active Low)
CBI PORTB,PINB4

// Initialization Timer1
LDI R16,0x04 // CTC Mode: Count upto 1024 times
OUT OCR1AH,R16
LDI R16,0x00
OUT OCR1AL,R16 // First Set Output Compare value
LDI R16,0x00 // Normal Mode and timer clock frequency= oscillator frequency/1024
OUT TCCR1A,R16
LDI R16,0x0D
OUT TCCR1B,R16 // At last activate the timer counter clock

LDI R31,HIGH(seven_segment<<1) // For higher byte of register pair Z
LDI R30,LOW(seven_segment<<1) // For lower byte of register pair Z

LOOP:          LPM R16,Z+
               OUT SPDR,R16
               SPI_Successful:            SBIS SPSR,SPIF
                                          RJMP SPI_Successful
               IN R17,SPDR
               OUT PORTA,R17
               CALL Delay //Make Delay of 1 Second
               CPI R30,0x0A
               BRNE LOOP
               LDI R30,LOW(seven_segment<<1)
               JMP LOOP

Delay:         LDI R17,0
               OUT TCNT1H,R17
               OUT TCNT1L,R17
               LDI R17,0x10
               OUT TIFR,R17
               LOOP_WAIT:           IN R18,TIFR
                                    ANDI R18,0x10
                                    BREQ LOOP_WAIT
               RET

.ORG 0x0400
seven_segment: .dB 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F
```

Slave Code:

```
/* Atmega32 to Atmega32 SPI Communication */
// Slave: Oscillator Frequency 8MHz

.INCLUDE "M32DEF.INC"
.ORG 0x0000

// Stack declaration
LDI R16,HIGH(RAMEND)
OUT SPH,R16
LDI R16,LOW(RAMEND)
OUT SPL,R16

// Data direction registers for 7 segment display
LDI R16,0XFF
OUT DDRA,R16
LDI R16,0X00
OUT PORTA,R16 // 7 segment initially turn off

//Data Direction Registers of SPI pins
```

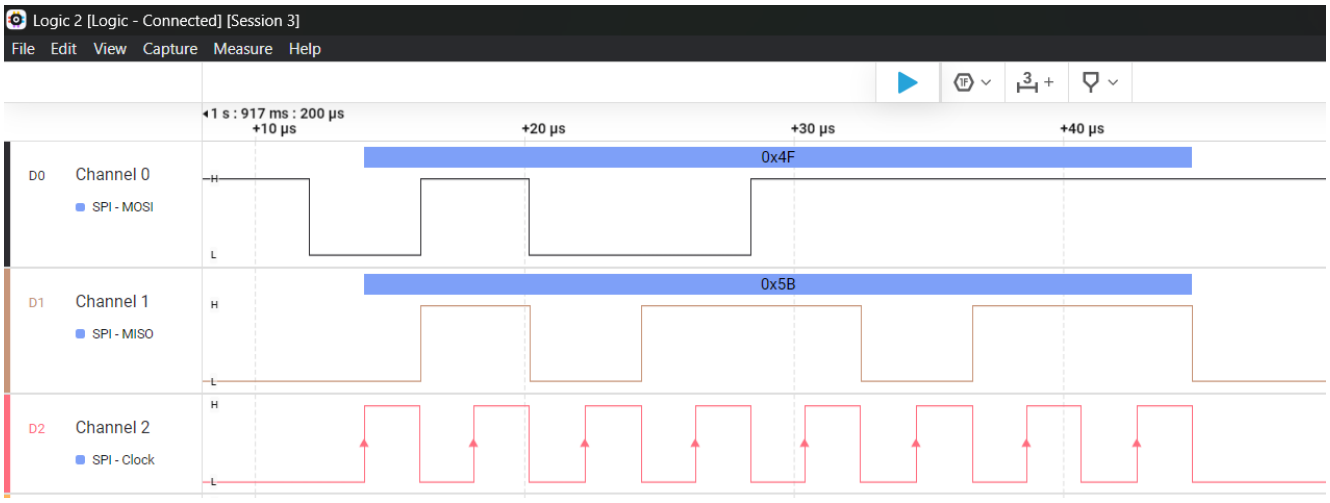
```
CBI DDRB,PINB4 // Slave select pin
SBI PORTB,PINB4 // Internal pull up resistor activated
CBI DDRB,PINB5 // MOSI pin
SBI PORTB,PINB5 // Internal pull up resistor activated
SBI DDRB,PINB6 // MISO pin
CBI DDRB,PINB7 // SCK pin
SBI PORTB,PINB7 // Internal pull up resistor activated

// SPI Control Register Settings
LDI R16,0x40
OUT SPCR,R16

SPI_Successful:             SBIS SPSR,SPIF
                            RJMP SPI_Successful
                            IN R17,SPDR
                            OUT PORTA,R17
                            JMP SPI_Successful
```

Verify the SPI data format as below:



MISO: 0x4F for 7 segment display 3
MOSI: 0x5B comes back to master again and displays 2
CPOL=0 and CPHA=0

Make the above circuit with hardware and verify the SPI data format with logic analyzer.



Class Assignment1: One ATmega32 works as a Master, and the other as a Slave in SPI communication. Each microcontroller has a common cathode seven-segment display connected to PORTA. The Master stores digit values (0-9) in its flash memory and sends them descending order to the Slave via SPI to display on its seven-segment display. When the Master sends a new digit, the previous digit is received back through MISO and shown on the Master's display. Use CPOL=1 and CPHA=1. Verify the SPI data format with logic analyzer.