Synchronous Serial Interfacing

Serial communication

- Single data wire, possibly also control and power wires
- Words transmitted one bit at a time
- Higher data throughput with long distances
  - Less average capacitance, so more bits per unit of time
- Cheaper, less bulky
- More complex interfacing logic and communication protocol
  - Sender needs to decompose word into bits
  - Receiver needs to recompose bits into word
  - Control signals often sent on same wire as data increasing protocol complexity

SPI(Serial Peripheral Interface)

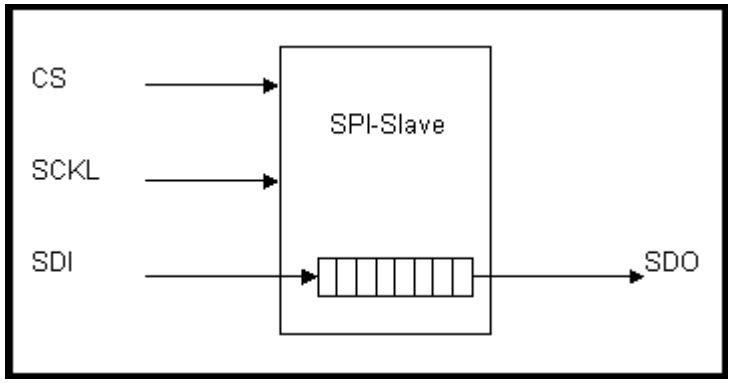
- Synchronous Serial Communication
- Developed by Motorola
- Also known as MicroWire (National Semiconductor), QSPI (Queued), MicrowirePlus
- There is no official specification for the SPI bus.
- It is necessary to consult the data sheets of the devices.
- Use to connect integrated circuits on a circuit board.

SPI(Serial Peripheral Interface)

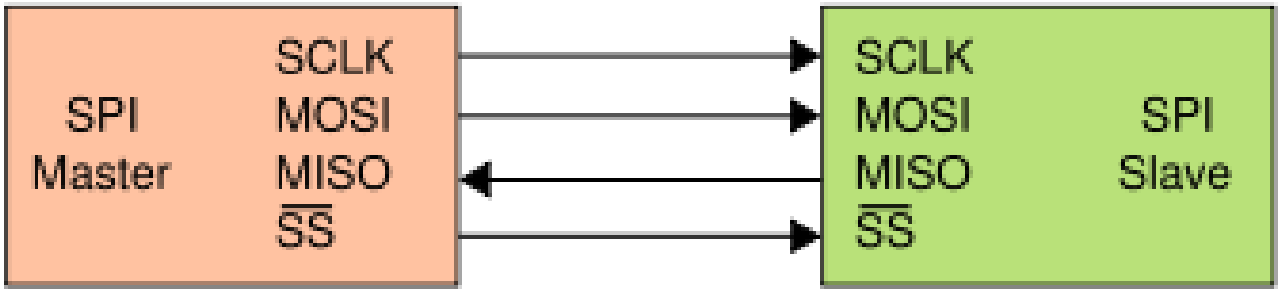
- SPI keeps the number of signal connections to a minimum and so reduces circuit board complexity.
- Many different peripheral device types available.
- Typical SPI devices are :-
  - Converters (ADC, DAC)
  - Memories (EEPROM, RAM's,Flash)
  - Sensors (Temperature, Humidity, Pressure)
  - Real Time Clocks
  - Potentiometers, LCD controllers, UART's, USB controller, CAN controller,amplifiers

SPI Configuration

- Primarily used for serial communication between a host processor and peripherals.
- Can also connect 2 processors via SPI
- SPI works in a master slave configuration with the master being the host microcontroller for example and the slave being the peripheral



SPI signals

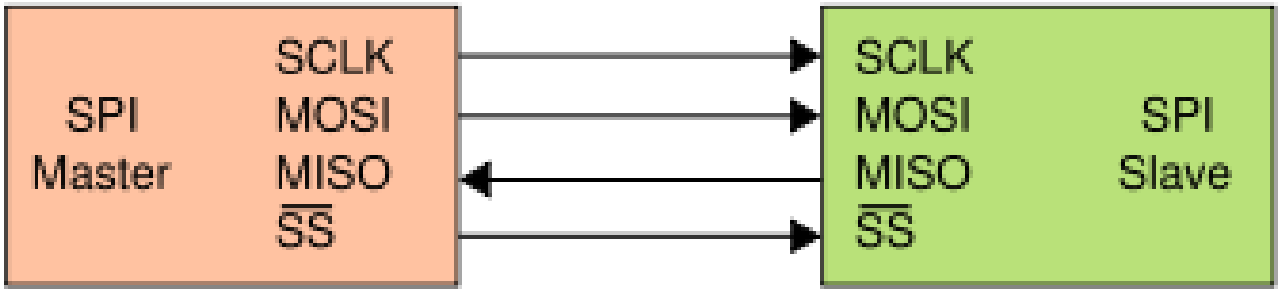


- The SPI bus specifies four logic signals.
  - SCLK - Serial Clock (output from master)
  - MOSI - Master Output, Slave Input (output from master)
  - MISO - Master Input, Slave Output (output from slave)
  - SS – Chip/Slave Select (active low; output from master)
- Alternative naming conventions
  - SCK, CLK - Serial Clock (output from master)
  - SDI, DI, SI - Serial Data In
  - SDO, DO, SO - Serial Data Out
  - SSEL - Slave Select

SPI Characteristics

- There is only one master, there number of slaves depends on the number of chip select lines of the master.
- Synchronous operation,
  - Data is only output during the rising or falling edge of SCK
  - Data is latched during the opposite edge of SCK
  - The opposite edge is used to ensure data is valid at the time of reading
- Supports Various Data Transfer Rates
- Master sends out clocks and chip selects. Activates the slaves it wants to communicate with

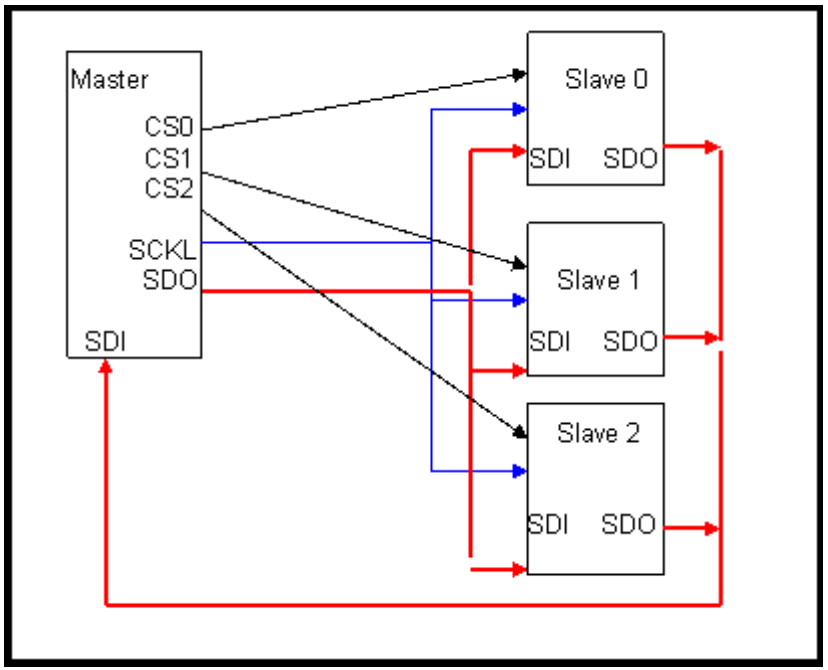
Typical SPI Configuration



1. The master pulls the slave select low and then issues clock cycles.
2. The clock frequency is not specified in the SPI protocol and can be anything from 0 up to 70MHz depending on the characteristics of the slave device.
3. The data transfer then takes place.
4. The master then de-selects the slave.

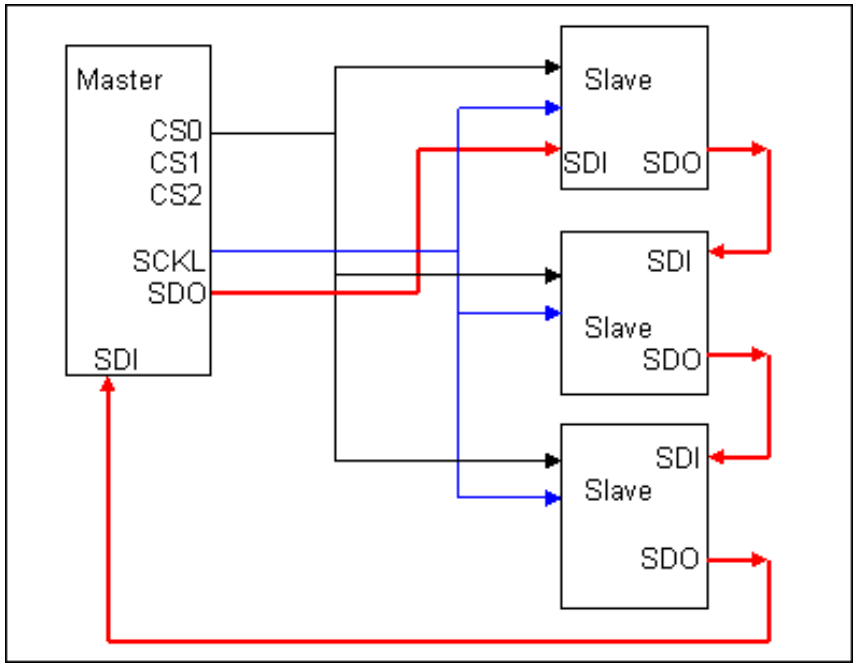
Master Slave Setup

- In this setup, there are 3 slave devices. The SDO lines are tied together to the SDI line of the master.
- The master determines which chip it is talking to by the CS lines. For the slaves that are not being talked to, the data output goes to a Hi Z state
- Multiple Independent Slave Configuration



Master Slave Setup
Multiple slave cascaded

- In this example, each slave is cascaded so that the output of one slave is the input of another. When cascading, they are treated as one slave and connecting to the same chip select



Simple master slave implementation

- During each SPI clock cycle, a full duplex data transmission occurs:
  - the master sends a bit on the MOSI line; the slave reads it from that same line
  - the slave sends a bit on the MISO line; the master reads it from that same line
- Not all transmissions require all four of these operations to be *meaningful* but they do happen.
- The number of bits transferred is not fixed but is usually a multiple of 8-bits.

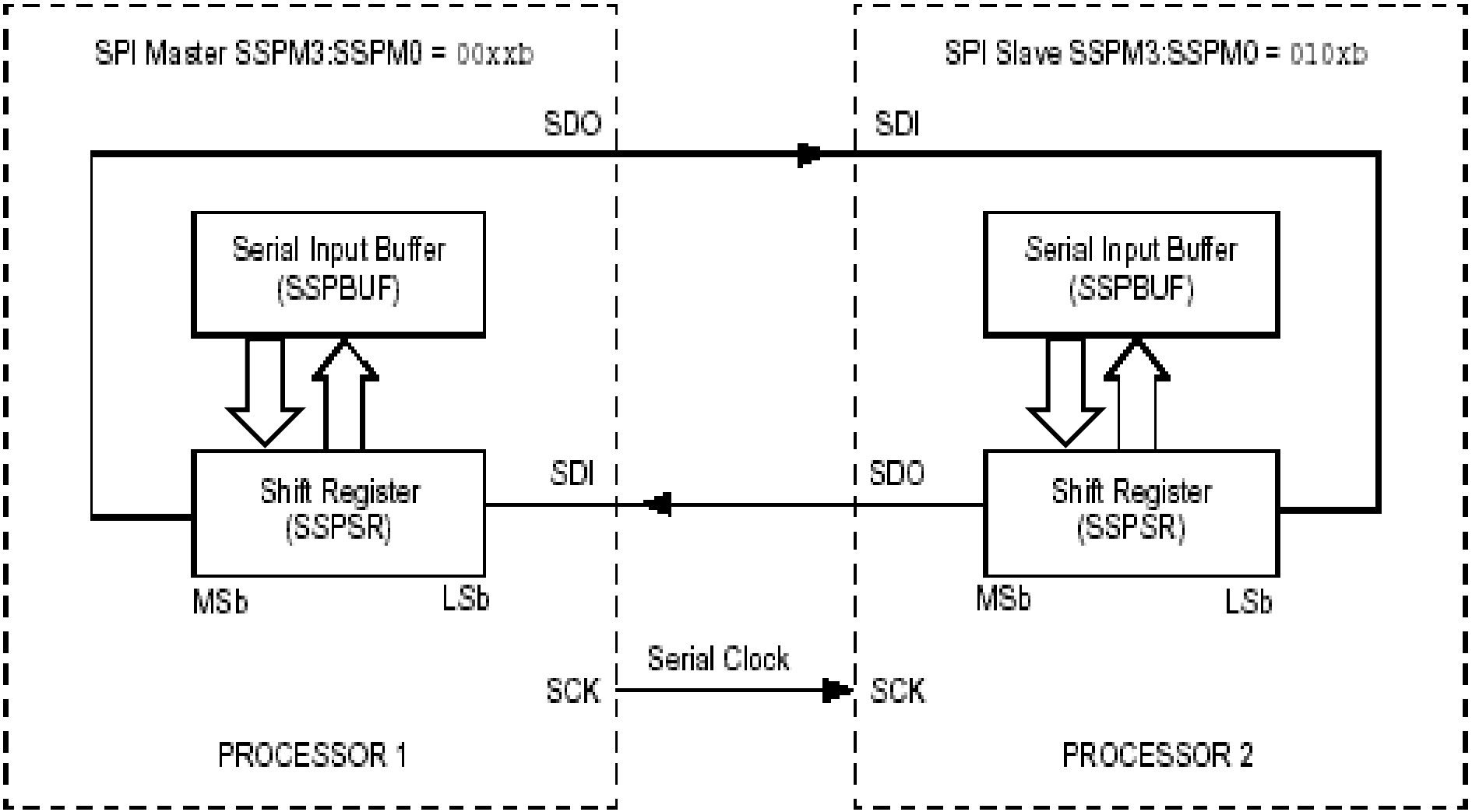
SPI Bus characteristics

- It is up to the master and slave devices to know whether a received byte is meaningful or not.
- So a device must discard the received byte in a "transmit only" frame or generate a dummy byte for a "receive only" frame.
- No Acknowledgement
- Master doesn't even know if slave is present!
- Slaves can be thought of as IO devices of the master.

Basic serial data transfer

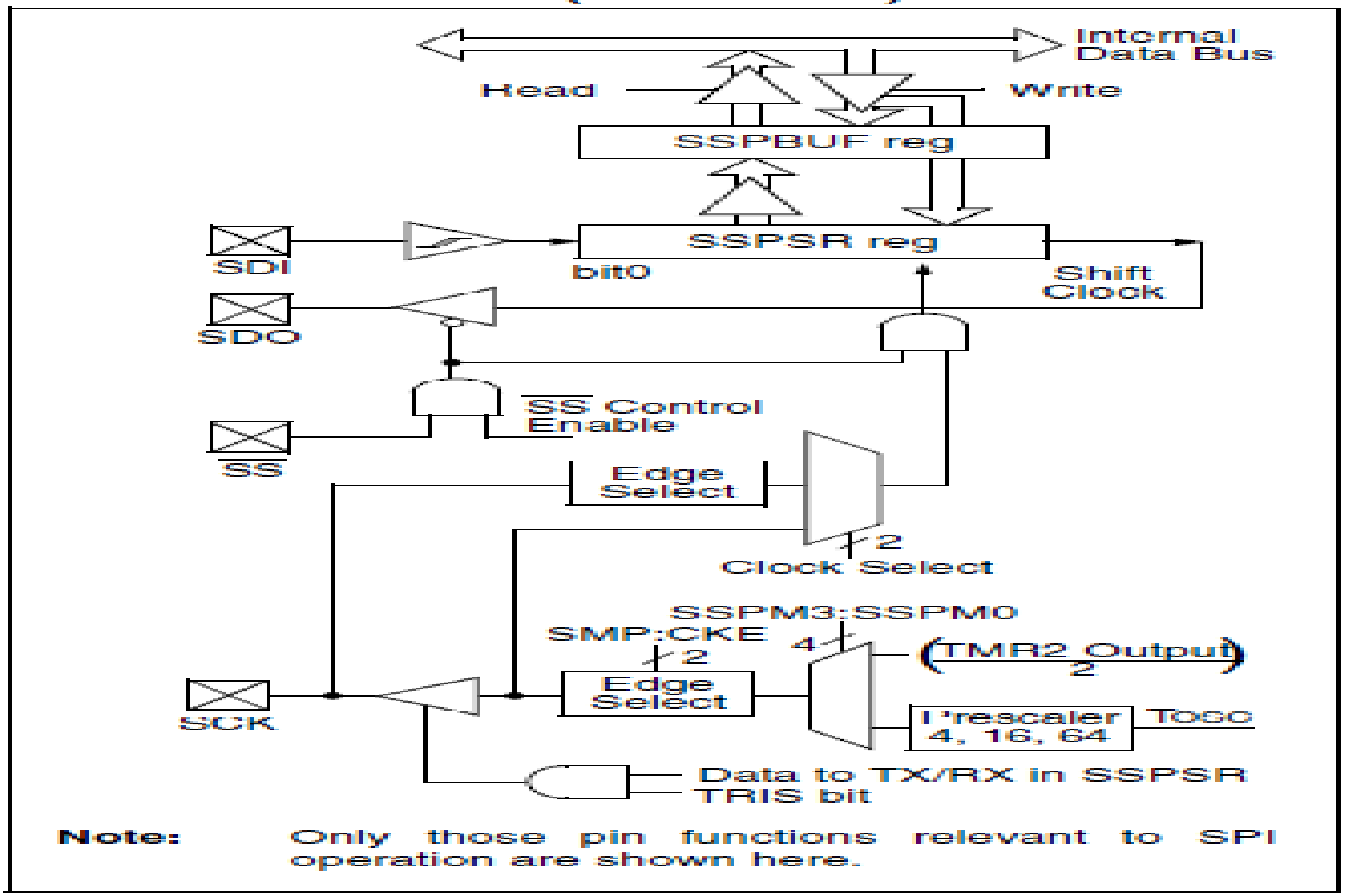
The registers within the master and slave act like shift registers shifting one bit on every cycle of the SCLK.

SPI Interconnection



Hardware description SPI

FIGURE 19-1: MSSP BLOCK DIAGRAM (SPI MODE)

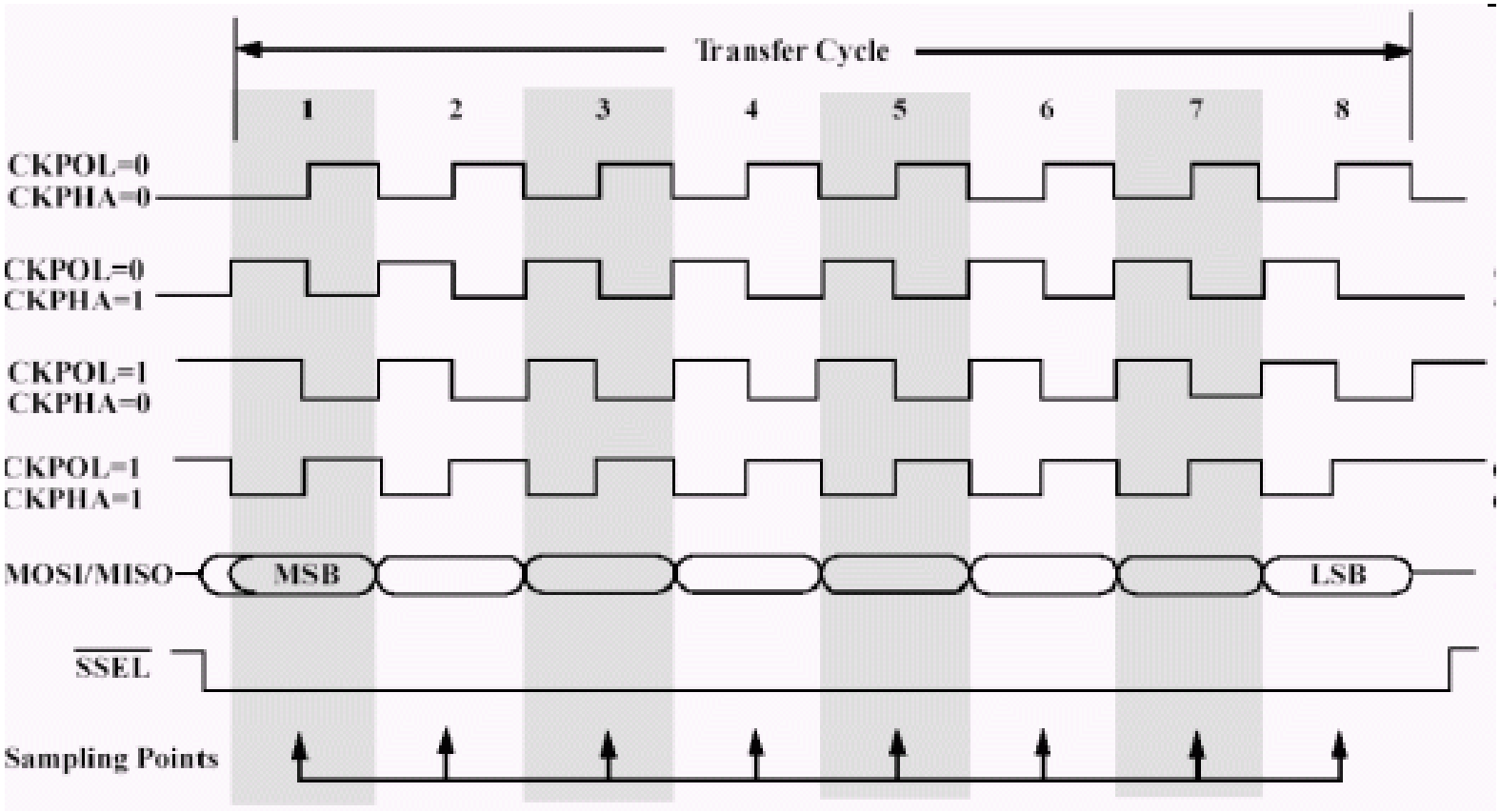


Note: Only those pin functions relevant to SPI operation are shown here.

Data transfer details

- Most SPI interfaces have two configuration bits, called clock polarity (CPOL) and clock phase (CPHA).
- CPOL will determine if the clock idles high or low:
  - CPOL = 0 SCK will idle low
  - CPOL = 1 SCK will idle high
- If CPOL=0 data are shifted out ,
  - on the falling edge of the clock if CPHA=0
  - on the rising edge of the clock if CPHA=1
- If CPOL=1, data are shifted out
  - on the rising edge of the clock if CPHA=0
  - on the falling edge of the clock if CPHA=1

SPI Data Transfer Modes



Data transfer details

- As each bit has two states, this allows for four different combinations, all of which are incompatible with each other.
- For two SPI devices to talk to each other, they need to be set to use the same clock polarity and phase settings.

SPI-mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

- Modes 0 and 3 are the most common.
- With SPI modes 0 and 3, data is always latched in on the rising edge of SCK and always output on the falling edge of SCK.

Serial protocols: I^2C

- I^2C (Inter-IC)
  - Two-wire serial bus protocol developed by Philips Semiconductors
  - Enables peripheral ICs to communicate using simple communication hardware
  - Common devices capable of interfacing to I^2C bus:
    - EPROMS, Flash, and some RAM memory, real-time clocks, watchdog timers, and microcontrollers

What is I^2C used for?

- **Data transfer between ICs and systems at relatively low rates**
  - "Classic" I^2C is rated to 100K bits/second
  - "Fast Mode" devices support up to 400K bits/second
  - A "High Speed Mode" is defined for operation up to 3.4M bits/second
- **Reduces Board Space and Cost By:**
  - Allowing use of ICs with fewer pins and smaller packages
  - Greatly reducing interconnect complexity
  - Allowing digitally controlled components to be located close to their point of use

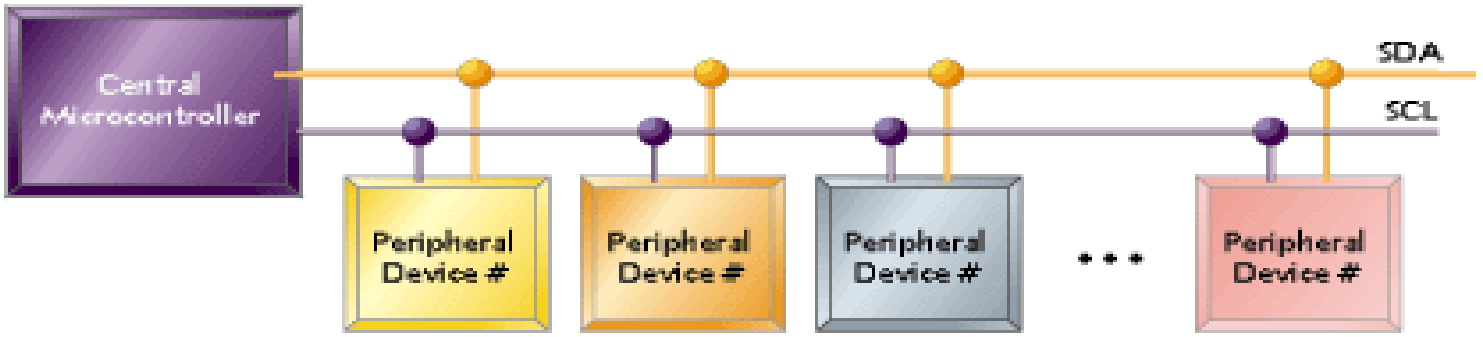
I^2C Bus Characteristics

- Two wire serial data & control bus implemented with the serial data (SDA) and clock (SCL) lines
  - For reliable operation, a third line is required: Common ground
- Unique start and stop condition
- Slave selection protocol uses a 7-Bit slave address
  - The bus specification allows an extension to 10 bits
- Bi-directional data transfer
- Acknowledgement after each transferred byte
- No fixed length of transfer
- Multiple Devices Connected On Bus

I^2C Bus Definitions

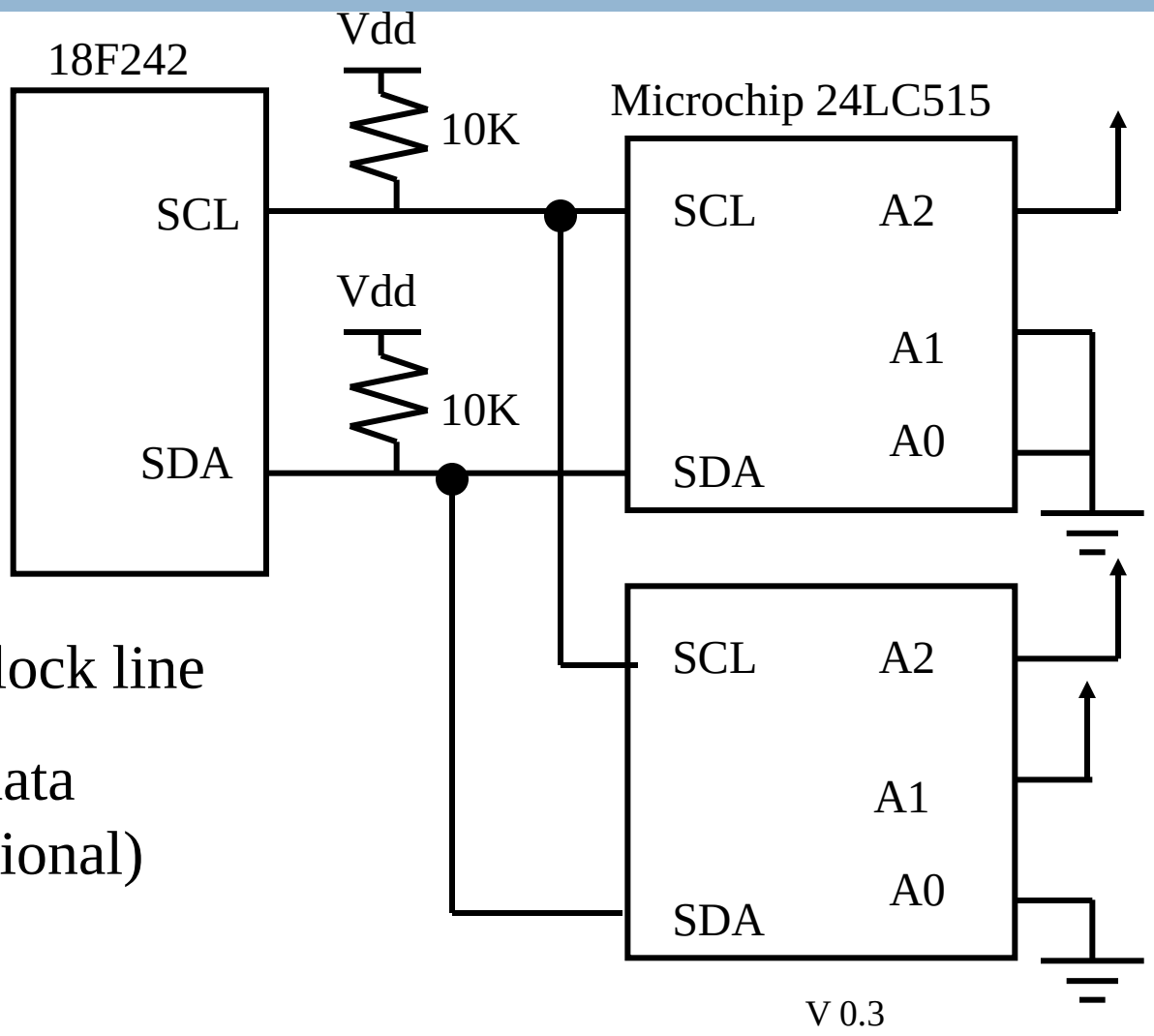
- **Master:**
  - Initiates a transfer by generating start and stop conditions
  - Generates the clock
  - Transmits the slave address
  - Determines data transfer direction
- **Slave:**
  - Responds only when addressed
  - Timing is controlled by the clock line

I²C Bus Configuration



- 2-wire serial bus – Serial data (SDA) and Serial clock (SCL)
- Half-duplex, synchronous, multi-master bus
- No chip select or arbitration logic required

I²C (Inter-Integrated-Circuit) Bus



SCL – clock line

SDA – data (bidirectional)

I^2C Features

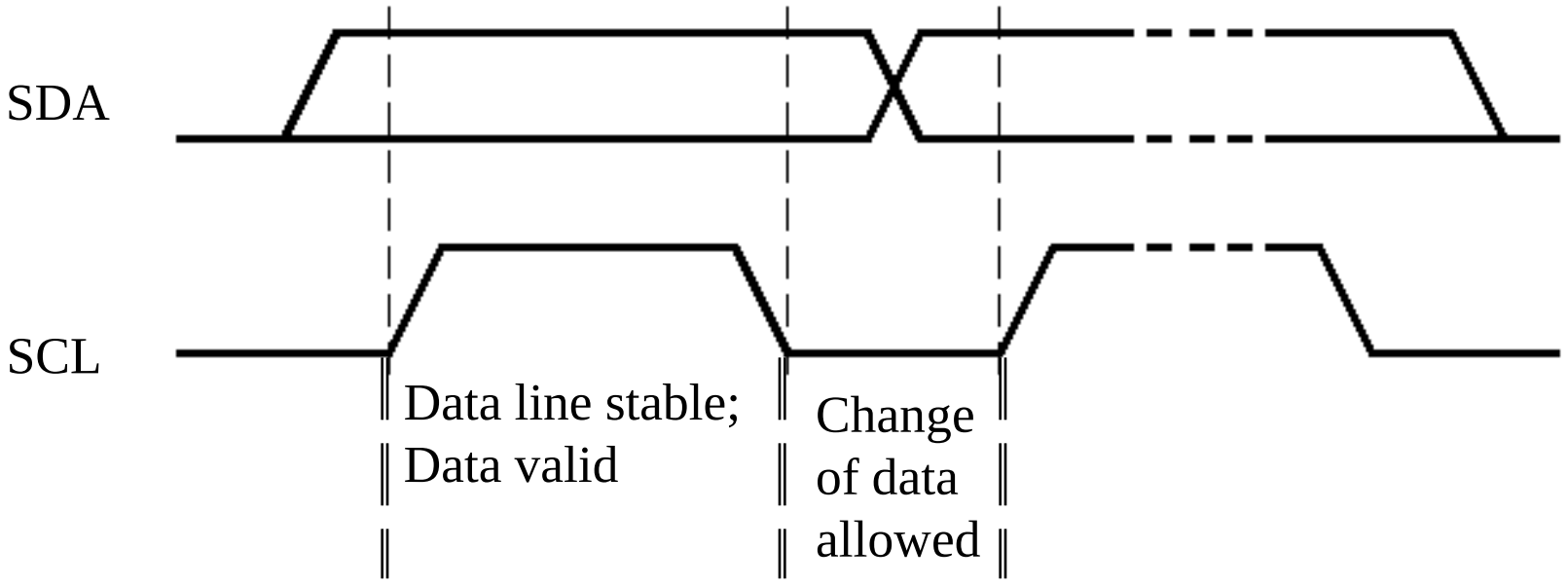
- Multiple receivers do not require separate select lines At start of each I^2C transaction a 7-bit device address is sent
  - Each device listens – if device address matches internal address, then device responds
- SDA (data line) is bidirectional, communication is half duplex
- SDA, SCLK are open-drain, require external pullups
  - Allows multiple bus masters .

I2C clock

- Not a “traditional” clock
- Normally is kept “high” using a pull-up
- Pulsed by the master during data transmission
  - Master could be either the transmitter or receiver
- Slave device can hold clock low if needs more time
  - Allows for flow control

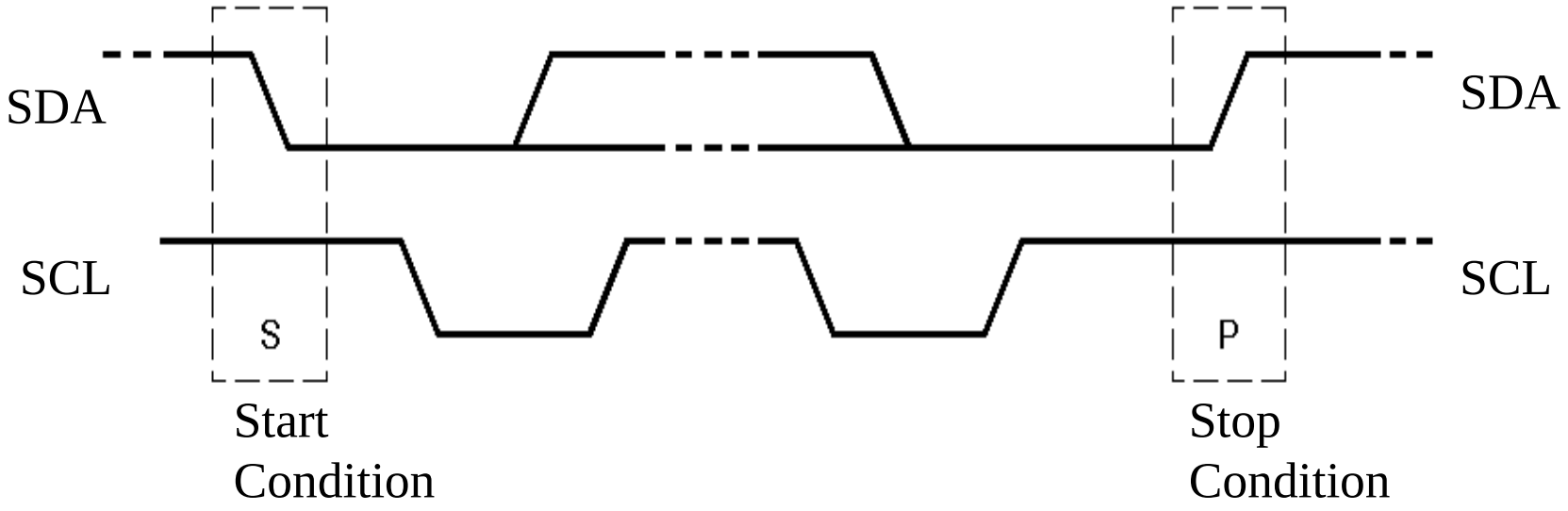
Bit Transfer on the I^2C Bus

- In normal data transfer, the data line only changes state when the clock is low



Start and Stop Conditions

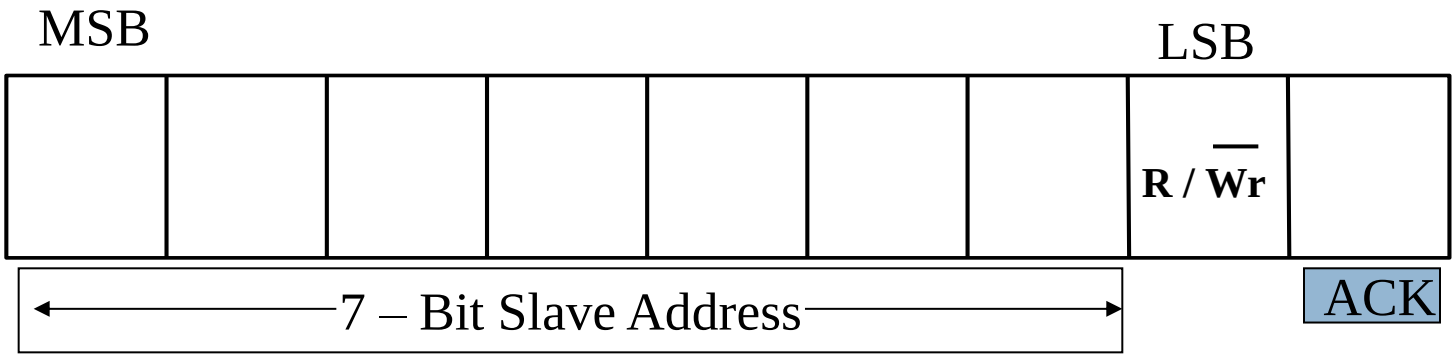
- A transition of the data line while the clock line is high is defined as either a start or a stop condition.
- Both start and stop conditions are generated by the bus master
- The bus is considered busy after a start condition, until a stop condition occurs



I²C Addressing

- Each node has a unique 7 (or 10) bit address
- Peripherals often have fixed and programmable address portions
- Addresses starting with 0000 or 1111 have special functions:-
  - 0000000 Is a General Call Address
  - 0000001 Is a Null (CBUS) Address
  - 1111XXX Address Extension
  - 1111111 Address Extension – Next Bytes are the Actual Address

First Byte in Data Transfer on the I^2C Bus



R/Wr

0 – Slave written to by Master

1 – Slave read by Master

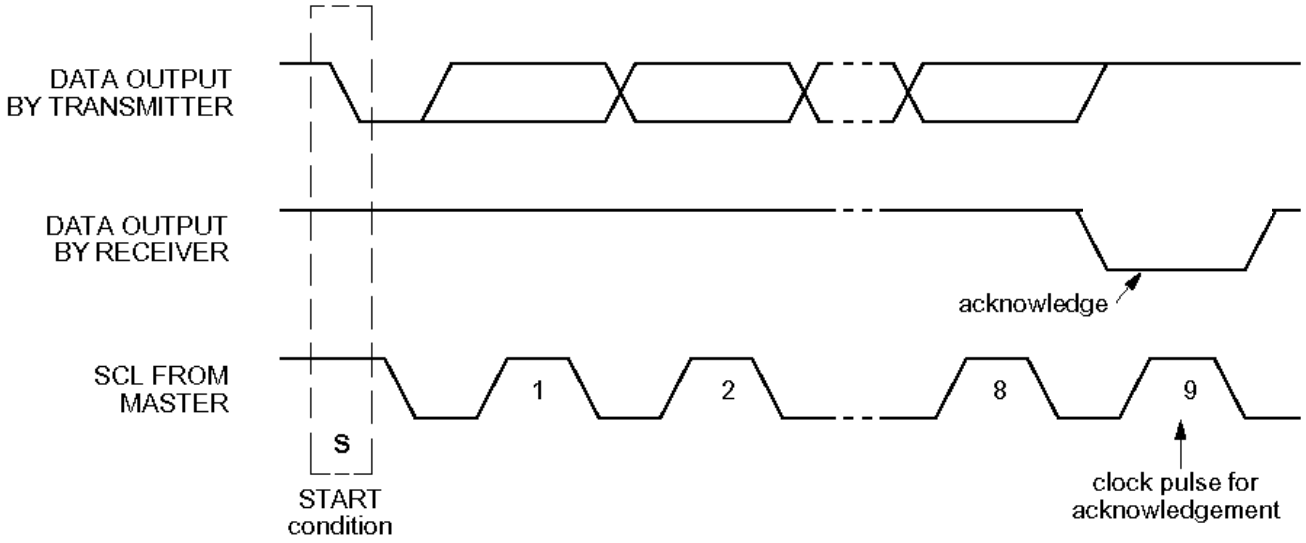
ACK – Generated by the slave whose address has been output.

I^2C Bus Connections

- **Masters can be**
  - Transmitter only
  - Transmitter and receiver
- **Slaves can be**
  - Receiver only
  - Receiver and transmitter

Acknowledgements

- Master/slave receivers pull data line low for one clock pulse after reception of a byte
- Master receiver leaves data line high after receipt of the last byte requested
- Slave receiver leaves data line high on the byte following the last byte it can accept

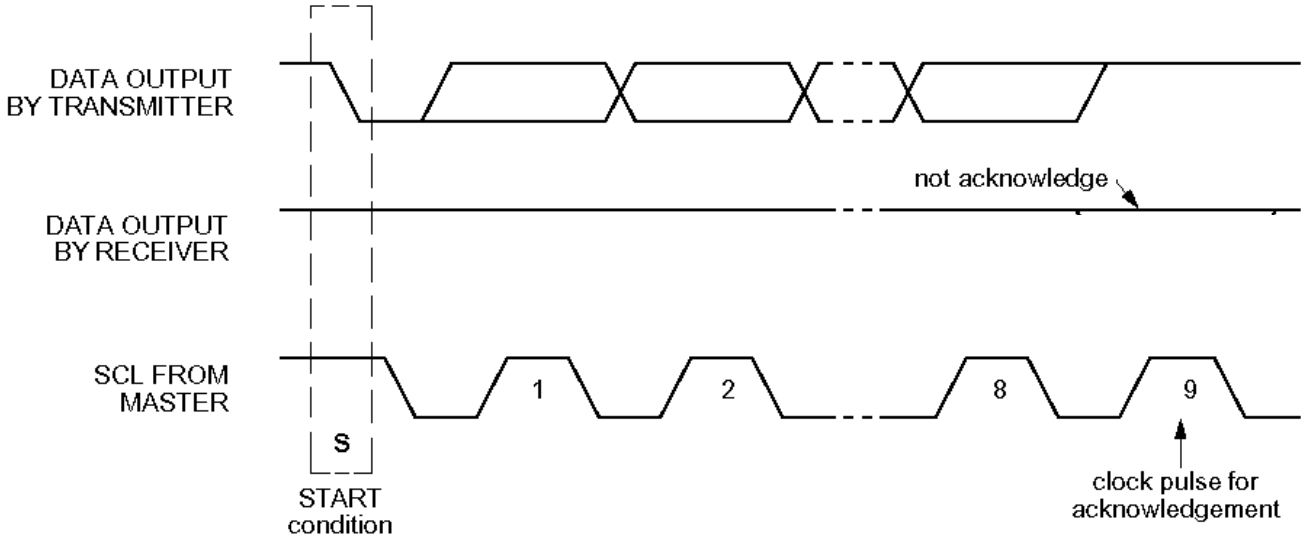


Acknowledgements

- **From Slave Receiver to Master Transmitter:**
  - After address received correctly
  - After data byte received correctly
- **From Slave Transmitter to Master Receiver:**
  - Never (Master Receiver generates ACK)
- **From Master Transmitter to Slave Receiver :**
  - Never (Slave generates ACK)
- **From Master Receiver to Slave Transmitter :**
  - After data byte received correctly

Negative Acknowledge

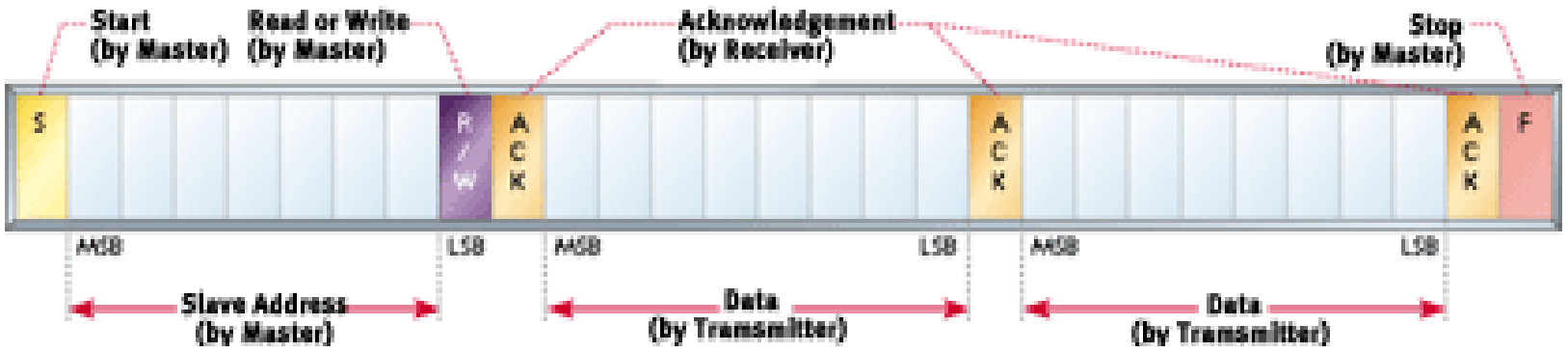
- Receiver leaves data line high for one clock pulse after reception of a byte



Negative Acknowledge (Cont'd.)

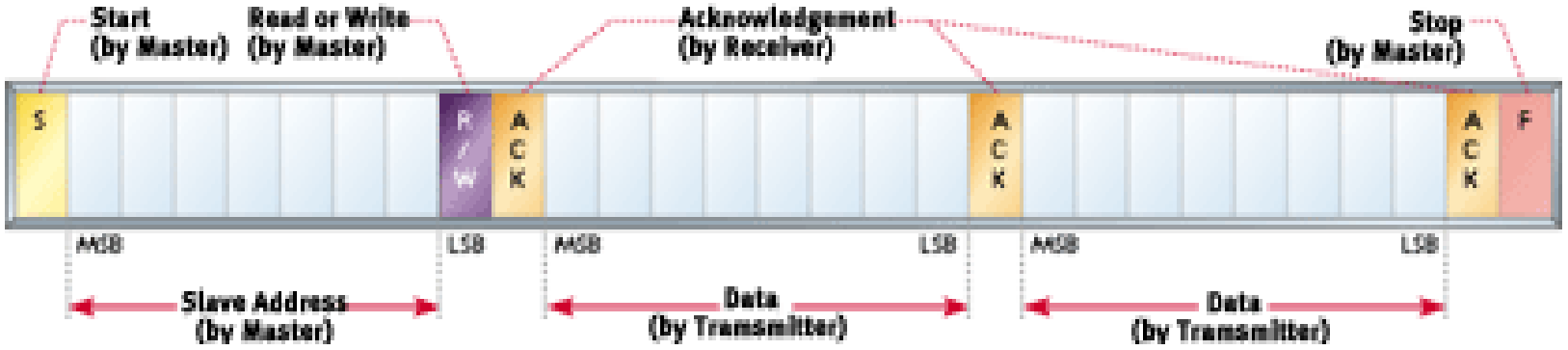
- **From Slave Receiver to Master Transmitter:**
  - After address not received correctly
  - After data byte not received correctly
  - Slave Is not connected to the bus
- **From Slave Transmitter to Master Receiver:**
  - Never (Master Receiver generates ACK)
- **From Master Transmitter to Slave:**
  - Never (Slave generates ACK)
- **From Master Receiver to Slave Transmitter :**
  - After last data byte received correctly

I²C Protocol



1. Master sends start condition (S) and controls the clock signal
2. Master sends a unique 7-bit slave device address
3. Master sends read/write bit (R/W) – 0 - slave receive, 1 - slave transmit
4. Receiver sends acknowledge bit (ACK)
5. Transmitter (slave or master) transmits 1 byte of data

I²C Protocol (cont.)



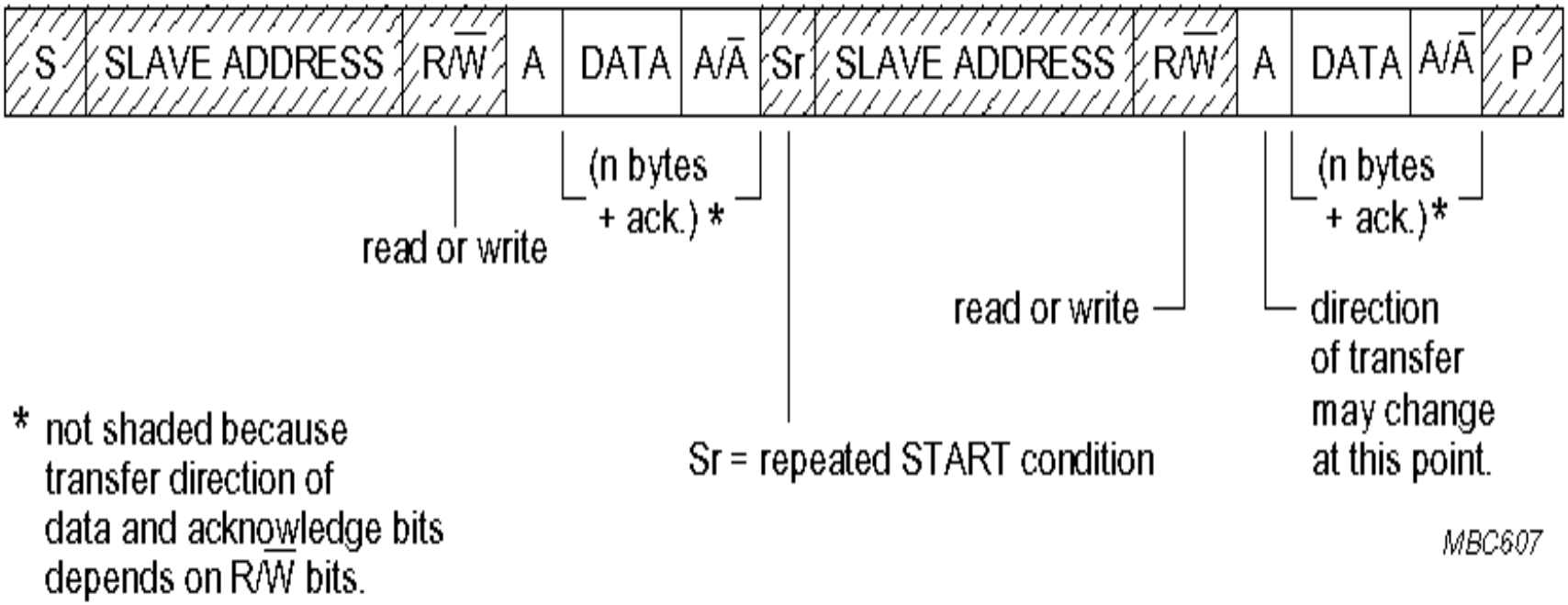
6. Receiver issues an ACK bit for the byte received

7. Repeat 5 and 6 if more bytes need to be transmitted.

8.a) For write transaction (master transmitting), master issues stop condition (P) after last byte of data.

8.b) For read transaction (master receiving), master does not acknowledge final byte and issues stop condition (P) to tell the slave the transmission is done

Repeated start

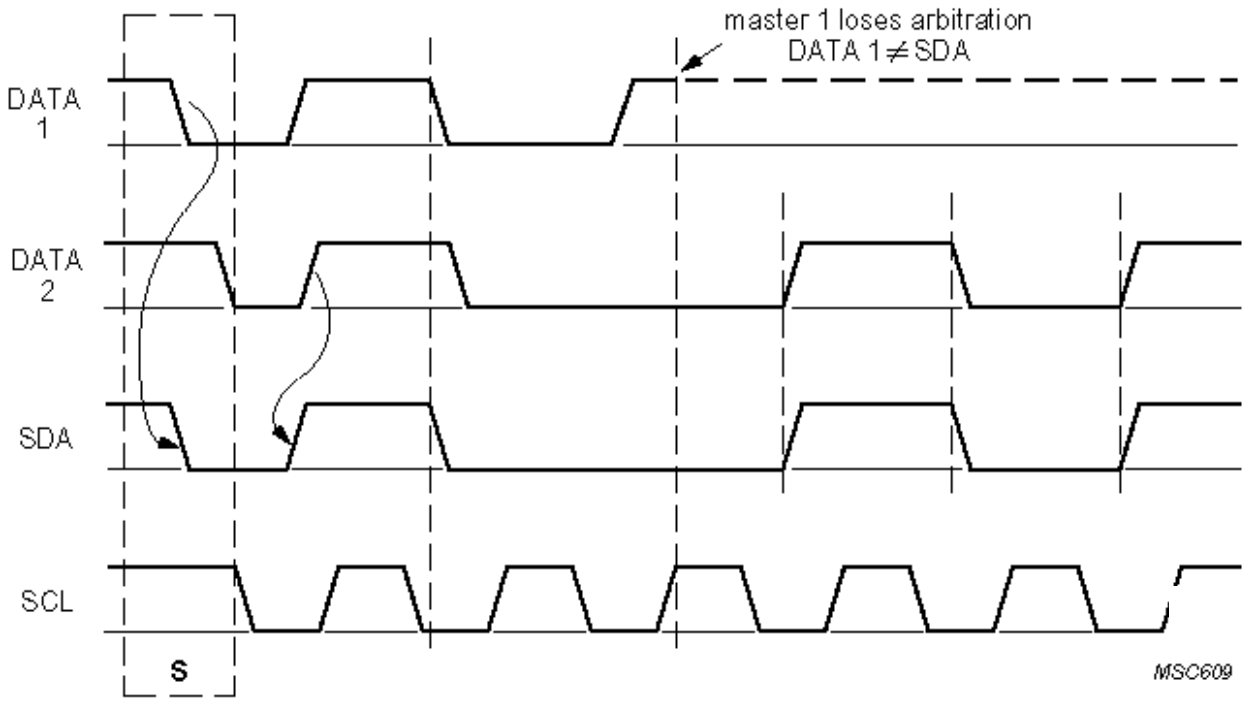


- A **repeated start** avoids releasing the bus and therefore prevents another master from taking over the bus

Multi-master I²C Systems

- **Multimaster situations require two additional features of the I²C protocol**
- **Arbitration:**
  - Arbitration is the procedure by which competing masters decide final control of the bus
  - I2C arbitration does not corrupt the data transmitted by the prevailing master
  - Arbitration is performed bit by bit until it is uniquely resolved
  - Arbitration is lost by a master when it attempts to assert a high on the data line and fails

Arbitration Between Two Masters



- As the data line is like a wired AND, a ZERO address bit overwrites a ONE
- The node detecting that it has been overwritten stops transmitting and waits for the Stop Condition before it retries to arbitrate the bus