Communication protocols

* Introduction :-

→ Data transmisson types :-

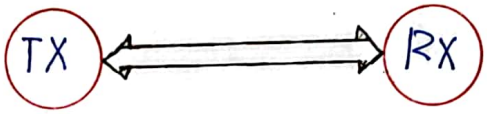
1- simplex

- one device transmits and the other devices receive
- Communication is possible in one direction only
- EX :
  radio

TX : Transmitter
RX : Receiver

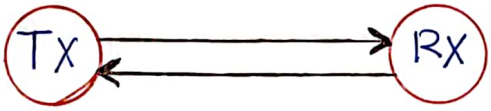
2- Half-Duplex :

- two devices can send and receive from each others.
- two way communication but only one way at a time
- EX :
  Walkie-talkie



3- Full-Duplex :

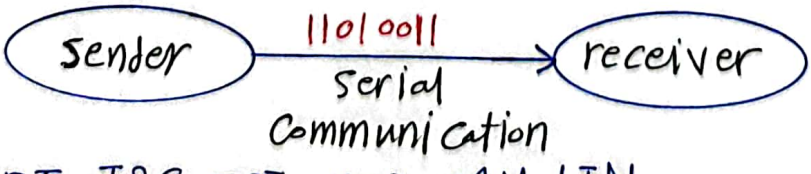
- Communication is possible in both directions, both sides can transmit and receive at the same time.
- ex : phone call



* Types of Communication :-

1- serial Communication :-

- the data will be sent (bit by bit), one bit at a time
- Less number of cables required to transmit data, as it needs one channel or wire for transmitting (low cost)
- sequentially transmission (slower)
- For correct data transmission, there has to be some form of synchronization between transmitter and receiver.
- used for all long-haul communication and most computer networks.



- ex :- UART . I2C, SPI , USB, CAN. LIN . ---

2- Parallel communication :-

- more than bit can be transmitted at the same time over a link that consists of several parallel channels.
- each bit is transmitted over a separate channel, allowing for faster data transfer compared to serial communication.
- parallel communication is typically used for short-distance transmissions due to certain limitations and challenges it faces:

1- clock skew :.
- the clock signal (sent from the clock circuit) arrives at different components (at different times)
- this happens because of wire-interconnect length, capacitive coupling, matrial imperfections (عيوب), and other environmental factors.
- it reduces the speed of every link to the slowest of all of the links

2- crosstalk:
- it occurs when the transmitted signal is adversely (بشكل) affected by another nearby signal.
- it happens when the ~~transmitted signal is also~~ electromagnetic energy from one cable or channel leaves an imprint on adjacent cables, causing interference (تداخل)

* features of parallel communication include:.

1- mostly used for short-distance communication
2- it offers fast data transfer rates (high bandwidth)
3- simplicity : it is relatively simple in terms of hardware implementation since it does not require serialization and deserialization components but it requires multiple channels or cables
4- less susceptible to noise

- ex : PCI Bus (peripheral component interconnect)

3- wireless communication :-

→ this is done by using IR (infrared) or RF (radio freq)
ex : Bluetooth, ZigBee, WIFI, 2G, 3G. ...etc

* Some of communication definitions

→ Bitrate : the number of bits transmitted per second.

→ Baud rate : the number of symbols signaled in a second

symbol = number of bits

ex :

when using a device has a set of 8 different symbols. each symbol represents 3 bits of useful data. if it transmits 300 bits per second, it gathers each 3 bits in 1 symbol. so it transmits 100 symbol per second.

→ Bit rate = 300 bit/sec

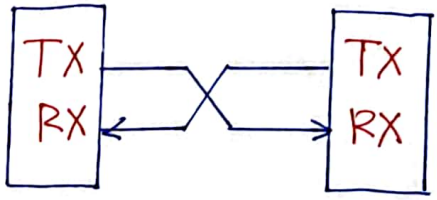
→ Baud rate = 100 symbol/sec

→ throughput : is the actual amount of data that is successfully sent/received over the communication Link (useful bits / frame bits)

$$\text{Throughput} = \frac{\text{no. of data bits}}{\text{no. of frame bits}}$$

* Types of communication systems (syncheronous / Asynchronous) :

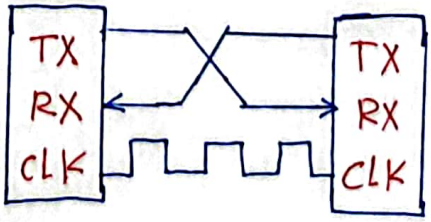
1) Asynchronous :



- the transmission of the data don't use external clock
- synchronization is done using a fixed baud rate and using synchronization bits (start and stop bits)
- the baud rate defined by the lowest one of the devices
- Byte oriented communication (data is a group of "Bytes")
- ex : UART

2) synchronous :-

- the transmission of the data uses external clock (common clock)
- the transmission is synchronized over clock
- the transmitter sends the data with the clock
- used at short distance communication
- more data to send
- faster than Asynchronous communication
- ex : SPI, I2C, USART, USB ----

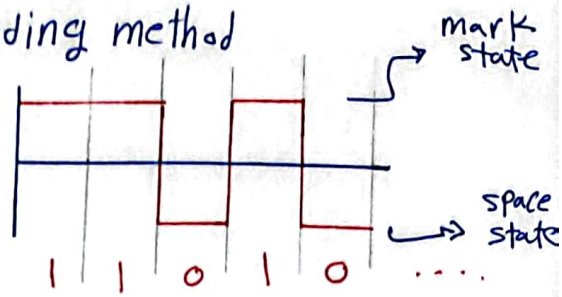


* Line encoding for data transmission

→ How to represent the data over a protocol or a wire :-

1- NRZL (non-returned-to-zero) encoding method

- at logic (1) → +ve voltage
- at logic (0) → -ve voltage

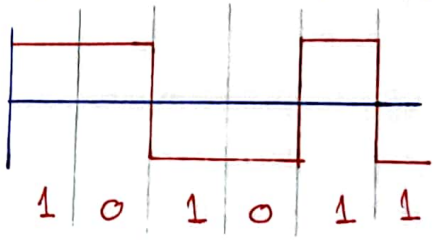


2- NRZI (non-return-to-zero-inversed)

- Logic (1) → -ve voltage
- Logic (0) → +ve voltage

3- NRZM (non-return-to zero-mark) :-

- the signal level stay constant for a 0 bit, and a transition occurs in 1-logic bit.



4- NRZS (non-return-to-zero-space) :-

- NRZS is the inverse of NRZM
- the signal level stay constant for a logic 1 bit, and a transition occurs in 0-logic bit

---

universal synchronous Asynchronous Receiver Transmitter (USART)

→ USART: is a peripheral for handling serial I/O communications

→ in PIC18f, it contains all the clock generators, shift registers and data buffers necessary to perform an input or output serial data transfer.

→ the USART, also known as a serial communication interface (SCI)

→ it can be configured as a full-duplex asynchrounous or half-duplex synchronous system.

→ the pins of the EUSART are multiplexed with portC

→ specs :-

1. full-duplex async or half-duplex sync
2. wired
3. serial
4. peer-to-peer : both devices can send and receive data from each other at any time

* full-duplex : with peripheral systems such as terminals and PC
* half-duplex : with peripheral devices such as A/D or D/A, serial EEPROM, or other microcontrollers

* In asynchronous mode :-
  - data is transmitted in the form of
    1- start bit. signal is always low
    2- Data bits. 0 to [5...9]
    3- parity bit. either odd or even
       even ← { no. of 1 is even → = 0 ; no. of 1 is odd → = 1 }
       which is used for error detection.
    4- stop bit. signal is always high [one or more stop bit]
    5- IDLE, No frame is transferred on the communication line. signal is always high in this state (there is no data transferring in this state)
  - The USART protocol uses two pins, one for transmitting (TX) and one for receiving (RX)
  - it operates at specific baud rates determined by a Baud-Rate Generator.
  - the specific implementation and features of USART can vary between different microcontrollers, making it not standard communication protocol.
  - the baud rates are:

    300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps.

  - in Asynchronous mode we need to define the frame format and Baud rate "speed"
  - the point of synchronization is managed by having the same baud rate on both devices.
  - the allowable difference of baud rate is up to 10%.
  - the USART data transmission line is normally held at high voltage level when it's not transmitting data 'IDLE'
  - to start the transfer of data, the transmitting device pulls the transmission line from high to Low for one clock cycle "start bit"
  - when the receiving device detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

* Data Frame

→ in UART, the mode of transmission is in the form of a packet "frame"
  → a packet "frame" consists of a start-bit, data-bits, parity_bit and stop-bits

Start_bit (1-bit)	Data Frame 5 to 9 data bits	Parity-bits 0/1 bit	stop-bits 1/1.5/2 bits

→ the data field contains the actual data being transferred.
  * Data field size : [5:8 bits] + parity bit
  * " " " : [9 bits] → No " "

→ in most cases, the data is sent with the least significant bit (LSB) first.

ex: 'A' → ASCII Code = 65 → 0100 0001 (MSB ↝ , LSB ↝)

→ the sender & receiver must align to the size of data field "must be the same number of bits

* Parity bit :-

even Parity
- "1" bits odd → parity bit = 1
- "1" bits even → parity bit = 0

odd parity
- "1" bits odd → parity bit = 0
- "1" bits even → parity bit = 1

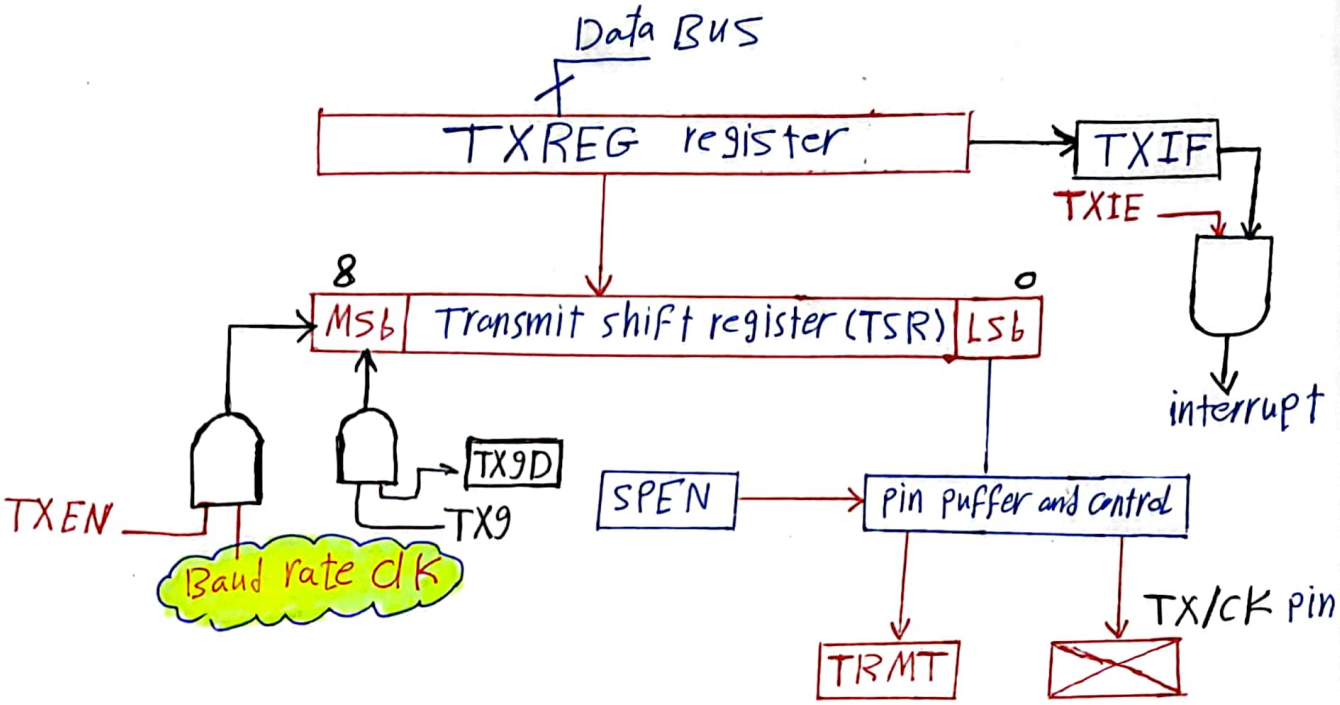
→ this bit is added to help to check that the number is received correctly.
  • bits can be changed by electromagnetic radiation, mismatched baud_rates or long distance transfers.

* Stop bit:

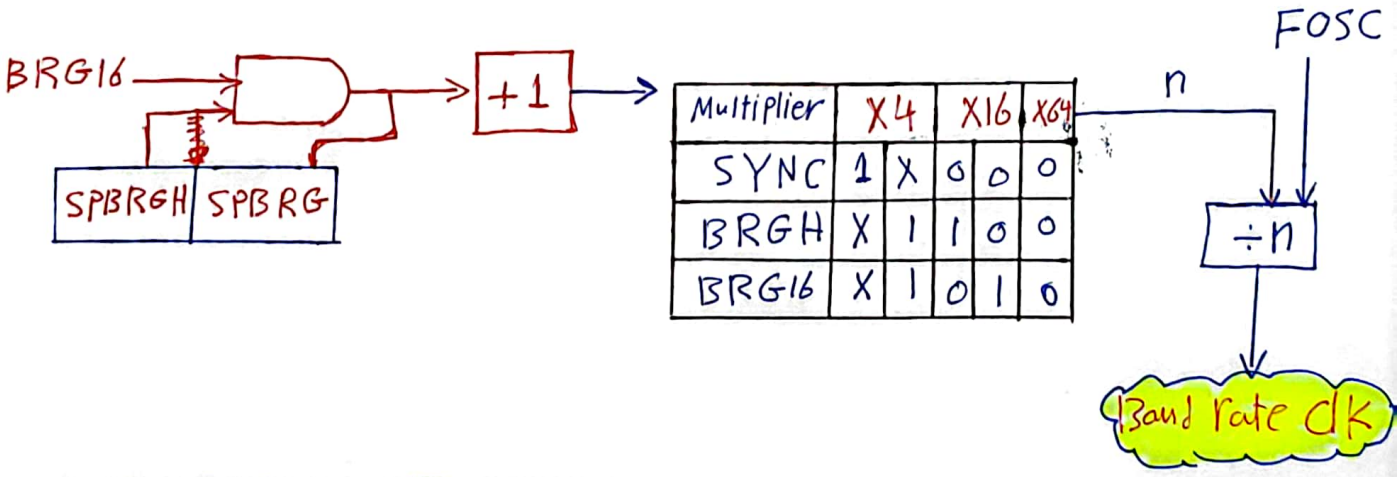
→ to signal the end of the data packet, the sender drives the data transmisson line from a low voltage to a high voltage for 1 bit or 2 bits or 1.5 bits duration

→ in PIC 18F the operation of EUSART module is controlled through 3 registers:
  • transmit status and control (TXSTA)
  • Receive " " " (RCSTA)
  • Baud rate control (BAUDCON)

*Transmit block diagram:



Baud Rate Generator (BRG)



Multiplier	X4		X16		X64
SYNC	1	X	0	0	0
BRGH	X	1	1	0	0
BRG16	X	1	0	1	0

→ the usART transmitter block diagram consists of the transmit shift register (TSR), which receives data from the transmit buffer (TXREG)
→ the TXREG register is loaded with data in software
→ the TSR register is loaded only after the previous load's stop bit has been transmitted
→ once the stop bit is transmitted, the TSR is loaded with new data from TXREG
→ when the TXREG transfer data to the TSR, the TXIF flag bit is set, indicating that the TXREG register is empty

→ the TRMT bit in the TXSTA register (transmit status and control) shows the status of the TSR register, indicating whether it is empty or not.

→ ~~to enable~~ note: the TSR register is not mapped in data memory, so it is not available to the user

2: when the TXEN bit is set, the TXIF flag bit will also be set since the transmit buffer is not yet full (can move transmit data to the TXREG register)

→ to enable the transmission, the TXEN enable bit in the TXSTA reg should be set.

→ the actual transmission will occur when the TXREG register has data and the baud rate Generator (BRG) has produced a shift clock.

→ the transmission can also be started by loading the TXREG register and then setting the TXEN enable bit

→ if 9-bit transmission is desired, the TX9 bit in the TXSTA register should be set

→ the ninth bit should be written to the TX9D bit before writing the 8-bit data to the TXREG register.

→ steps to follow when setting up an Asynchronous transmission:

1. initialize the SPBRG register for the desired baud rate. if a high speed baud rate is desired, set the SPBRGH bit.
2. enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit
3. if interrupts are desired, then set the TXIE, GIE/GIEH and PEIE/GIEL bits and specify the interrupt priority if required
4. set the TX9 bit if 9-bit transmission is desired
5. enable the transmission by setting the TXEN bit, which also sets the TXIF bit
6. if 9-bit transmission is selected, load the 9th bit in the TX9D bit
7. load data into the TXREG register to start the transmission.

* USART Baud Rate Generator (BRG):

→ BRG: it is a dedicated 8-16 bit baud rate generator

→ the SPBRG register controls the period of a free running 8-bit timer which operates independently and continuously without being triggered by an external event

→ the SPBRGH : SPBRG register pair determines the baud rate timer period.

→ in asynchronous mode, the multiplier of the baud rate period is determined by both the BRGH and BRG16 bits

→ in synchronous mode, the BRGH bit is ignored.

→ using high baud rate (BRGH = 1) or 16-bit BRG (BRG16 = 1) reduces baud rate error.

→ writing a new value to SPBRGH : SPBRG register pair resets the BRG timer.

* Baud Rate Calculation

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit / Asynchronous	Fosc / (64 (n+1))
0	0	1	8-bit / Asynchronous	Fosc / (16 (n+1))
0	1	0	16-bit / Asynchronous	Fosc / (16 (n+1))
0	1	1	16-bit / Asynchronous	Fosc / (4 (n+1))
1	0	X	8-bit / synchronous	Fosc / (4 (n+1))
1	1	X	16-bit / synchronous	Fosc / (4 (n+1))

* SYNC: EUSART mode select bit
  - 1 = synchronous mode
  - 0 = Asynchronous mode

* BRG16 : Baud rate 16-bit register enable
  - 1 = 16-bit BRG
  - 0 = 8-bit BRG

* BRGH : High baud rate select bit

→ Asynchronous mode:
  - 1 = High speed
  - 0 = low speed

→ synchronous mode:
  - unused in this mode

ex:

For a device with Fosc of 16MHZ, desired baud rate of 9600, Asynchronous mode, 8-bit BRG

desired baud rate = Fosc / (64 (n+1)) n = [SPBRGH:SPBRG]

$$n = ((Fosc / \text{desired baud rate}) / 64) - 1$$

$$= \frac{Fosc / \text{desired baud rate}}{64} - 1$$

$$= \frac{16M / 9600}{64} - 1 = 25.042 \simeq 25$$

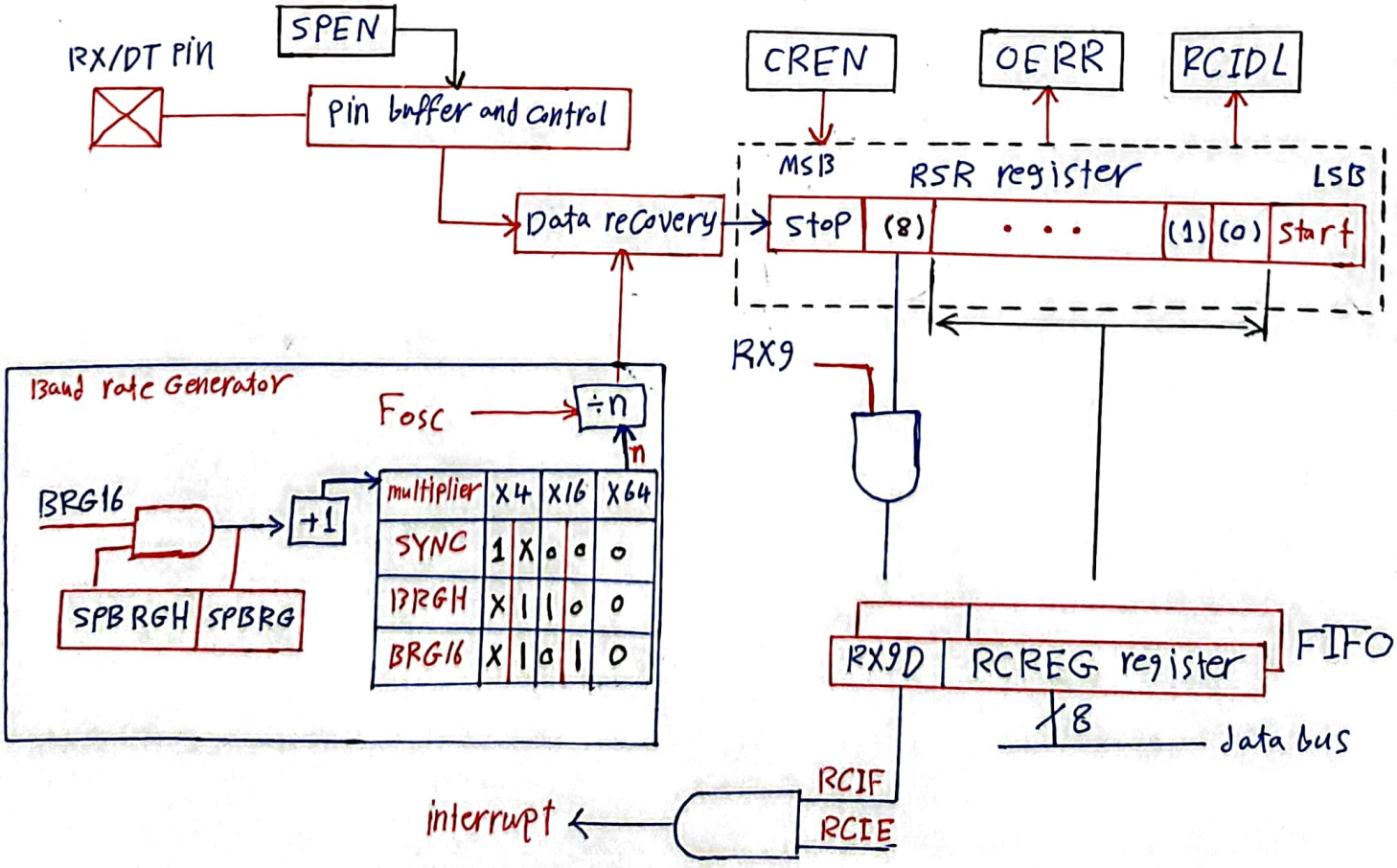
Calculating for actual baud rate:

$$\text{Calculated baud rate} = \frac{16M}{64\,(25+1)} = 9615$$

$$\text{Error} = \frac{9615 - 9600}{9600} = 0.0016 = 0.16\%\ \text{err}$$

---

* USART Asynchronous Receiver:



→ the data is received on the RX/DT pin and drives the data recovery block, which operates at X16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at Fosc
→ setting the RX9 bit in the RCSTA register enables 9-bit reception and places the ninth bit in the RX9D status bit
→ once Asynchronous mode is selected, reception is enabled by setting the CREN (continuous receive enable bit)
→ the RSR is the receive shif register and holds the received data. it is transferred to the RCREG register if empty, and the RCIF Flag is set.
→ the RCIE enable bit enables / disables the interrupt for received data
→ the RCIF flag bit is a read-only bit that is cleared by the hardware, it is cleared when the RCREG register has been read and is empty.
→ the RCREG register is double-buffered, allowing two bytes to be received and transferred while a third byte is shifting to the RSR register.
→ on the detection of the stop bit of the third byte, if the RCREG register is still full, the overrun error bit, OERR, will be set
→ the word in the RSR will be lost
→ the RCREG register can be read twice to retrieve (نسترجع) the two bytes in the fifo (first in first out)
→ the OERR bit has to be cleared in software, this is done by resetting the receive logic (the CREN bit is cleared and the set)
→ if the OERR bit is set, transfers from the RSR register to the RCREG register are inhibited
→ the framing error bit "FERR" is set if a stop bit is detected as a Low level
→ the FERR bit and the 9th receive bit are buffered the same way as the receive data.
→ reading the RCREG will load the RX9D and FERR bits with new values, so it's important to read the RCSTA register first to avoid losting old information in those bits.

serial peripheral interface (SPI)

→ the master synchrounous serial port (MSSP) module is a serial interface, useful for communicating with other peripheral or mcu devices

→ these peripheral devices may be serial EEPROMs, shift registers, A/D converters, etc..

→ the MSSP module can operate in one of two modes:
- serial peripheral interface (SPI)
- inter-integrated circuit (I2C)

* SPI: specifications

→ synchronous serial communication protocol

→ master slave topology (commonly single master multi slaves)

→ used for short distance communications

→ full & half duplex modes :-
- single master - single slave
- single master - multi slaves
- multi master (depend on the mcu ex. STM32F407)

→ the master provides the communication clock (SCK)(CLK)

→ requires (4 or 3 pins) → disadvantage
- MOSI : master output slave input (from master to slave)
- MISO : master input slave output (from slave to master)
- SCLK : serial clock (driven by master)
- SS or CS: slave or chip select

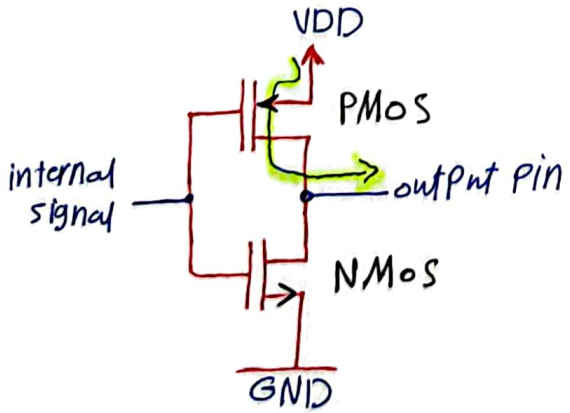
* for half duplex
  - Data I/o - clk - SS
    ↳ same pin used for transmitting and receiving

→ the SS pin is used by the master to select the slave that it wishes to communicate with.
  → one SS pin is required per slave
  → so, if "n" slaves are connected to a master in an SPI bus configuration, "n" SS pins are required on the master.

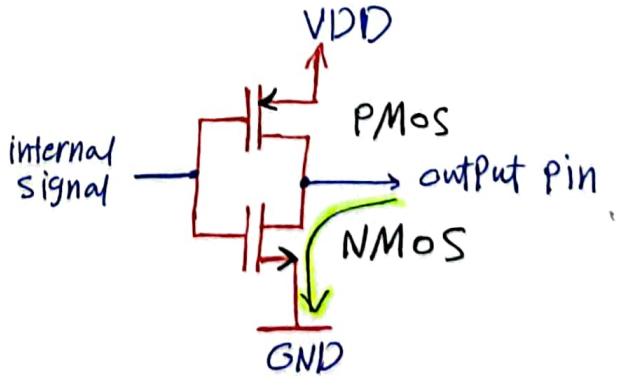
→ For a bus configuration where 'n' slaves are connected to a master, (3+n) pins will be required on the master
→ there is no standard set of speeds defined for the SPI protocol (depends on the clock frequency → 10, 20, 50 Mbs) → High speed
→ Higher throughput → since there is no overhead added by the protocol such as addressing and flow control
→ lower power requirements than I2C protocol → it used in battery operated applications than I2C protocol.
→ master controls all communications
→ No flow control, master and slave must determine the communication speed
→ GPIO → Push-Pull configuration → Advantage

* Push-Pull output Vs. open drain output.
→ they are two common types of output configuration for digital output in microcontrollers

1- Push-Pull
→ push-pull is the most common output configuration
→ push-pull output is capable of driving two output levels. one is pull to ground (pull / sink current from the load) and the other is push to power supply voltage (push / source current to the load)
→ the push-pull output can be implemented using a pair of switches. the practical implementation involves the use of Mos transistors.
→ in the following figure, we can see a push-pull output implementation using a PMos and an NMos transistor.



push phase
current source



pull phase
current sink

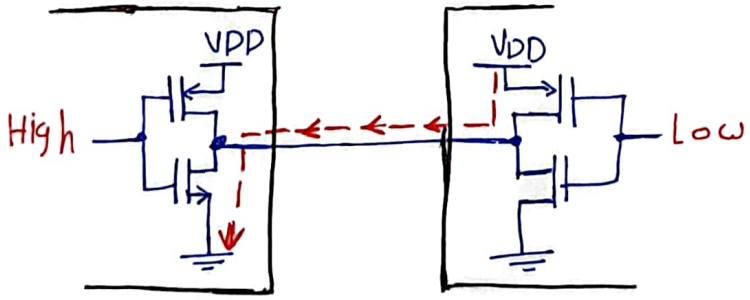
*Push-phase: when the internal signal is set to a low logic level (logic 0), the P-MOS transistor is activated and current flows through it from the VDD to the output pin. N-MOS transistor is inactive (open) and not conducting current.

→ internal signal → = 0 → output = 1 → push
→ = 1 → output = 0 → pull

*Pull-phase: when the internal signal is set to a high logic level, the N-MOS transistor is activated (closed) and current starts to flow through it from the output pin to the GND. at the same time, the P-MOS transistor is inactive and not conducting current.

→ this type of output doesn't allow connecting multiple devices together in a bus configuration, like the open drain output.

→ push-pull configuration is most commonly used in interfaces that have unidirectional lines (transmission on the line is only in a single direction - SPI, UART etc--), this is because the current will flow freely from VDD to the ground if one is logic high and the other is logic low, resulting in short circuit, which will damage the devices involved in communication



* you cannot directly connect two push-pull output

2- open-drain

→ we only have an N-MOS, the P-MOS is removed and replaced by a resistor external to the microcontroller called pull-up resistor

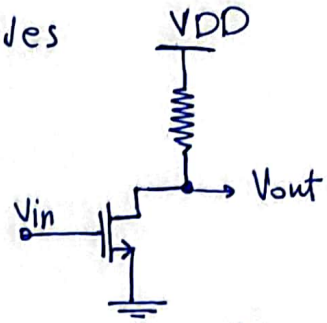
→ thus we can get either logic 0 at the output or it will be (float or is in open state)

→ if we need to get logic 1 we need to add an external pull-up resistor

→ so, we get logic 0 when N-MOS is conducting and logic 1 when it isn't.

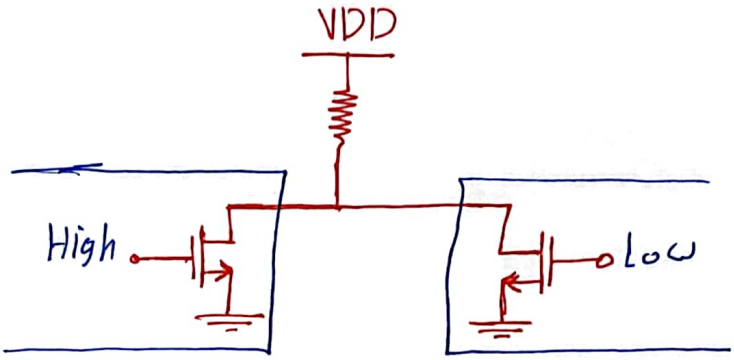
→ if the input is high, the N-Mos transistor provides a low-impedance connection to ground

→ if the input is low, the N-Mos looks like an open circuit, which means that the output gets pulled up to VDD through the external resistor



open-drain with external pull-up

* Bi-Directional communication



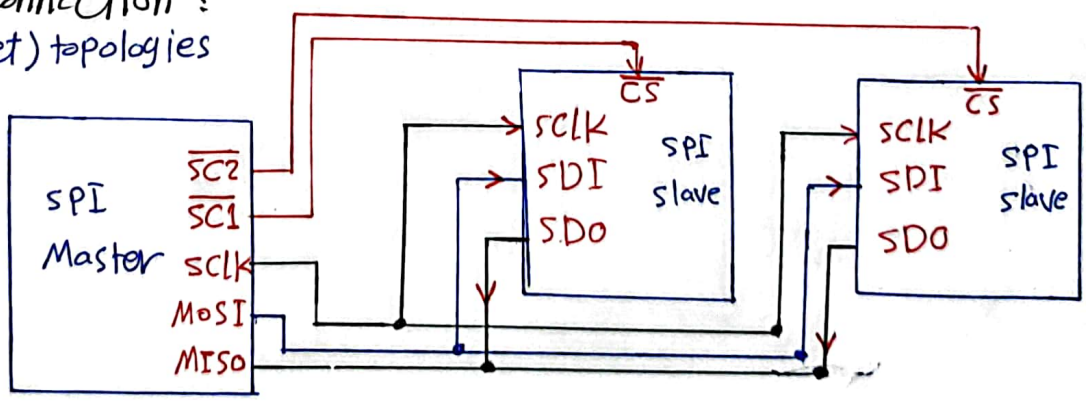
→ if we connect two open drain outputs together, if one is driving the output low and the other is driving it high

→ then the pull-up resistor ensures that the current doesn't flow freely from VDD to ground, thus there is no short circuit in this case

---

*types of SPI interfacing :-

1- Direct_connection : (star/net) topologies



* full-duplex
* 4 wires

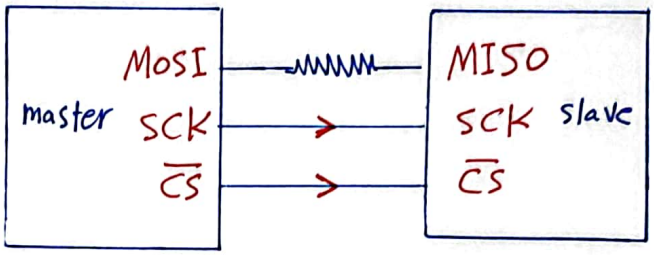
→ MOSI ⟷ MOSI / SDI

→ MISO ⟷ MISO / SDO

→ $\overline{CS}$ → chip select active low

2- Quasi Bidirectional : half- duplex mode.

→ Both master and slave use the same data line for transmission and reception
→ the transmission and reception done synchronously
→ the master and slave needs to alternate their roles synchronously.
→ it is common to add a serial resistor to prevent temporary short circuit connection.



3- simplex mode : unidirectional mode → only one data line is used.

→ one node acts as a transmitter and the other node acts as a receiver

→ MosI → MosI OR MISO ← MISO
→ SCK → SCK
→ CS → CS

4- Daisy chain (circular topology) :

→ Master MosI connected to slave-1 MISO
→ slave-1 MosI connected to slave-2 MISO
→ slave-2 MosI connected to Master MISO
→ master clock line connected to all slave SCKs
→ master CS (only one CS from the master) connected to all slave CSs

* Choosing the suitable slave in daisy chain :-

1- determine the number of slave devices connected in the daisy chain and their respective addresses. each slave device should have a unique address assigned to it.

2- define a data package structure that includes the address information and the actual data to be transmitted to the slave device.

3- after transmitting data packet, each slave device will continuously receive the data stream. when a slave device receives the address byte in the packet, it compares the received address with its own pre-defined address

4- If the received address matches, the slave device processes the payload data

→ in daisy chain, the received data is passed sequentially from one slave device to the next until it reaches the desired slave device based on the matching address.

5- multi master topology (MCU specific ex: STM32F4)

→ for example, there are two mcu nodes capable of acting as master in the system

→ all nodes, including the two mcu nodes, are set to slave mode, waiting for communication

→ when a node wants to communicate, it switches itself to activate the master function, taking control of the SPI bus.

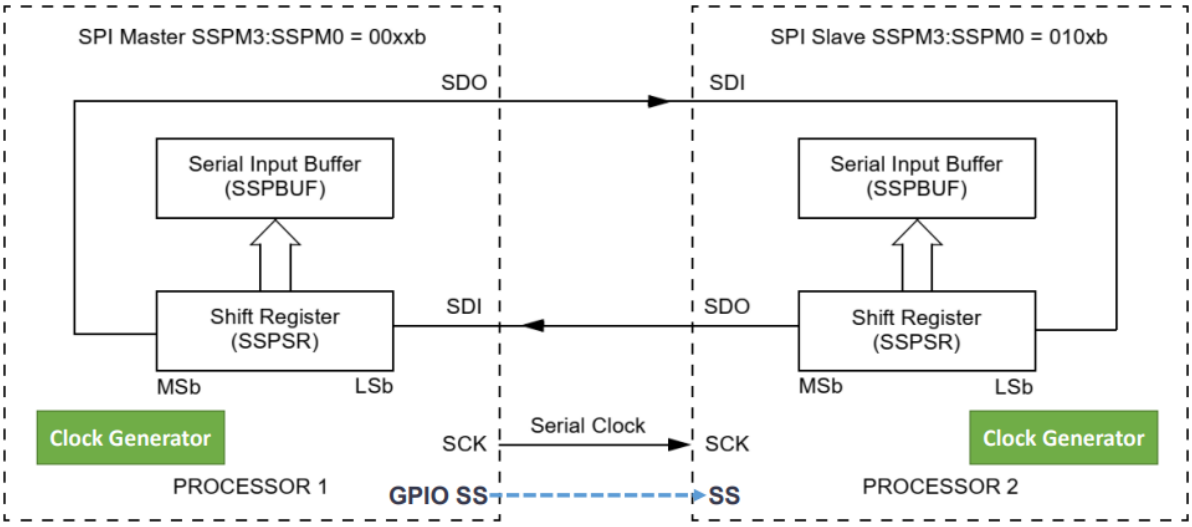
- responsible of clock generation and slave selection
- start communication session

→ CS pins detect if there are potential bus collisions, indicating the start of communication

→ after completing the communication, the master node returns to the slave mode, allowing other nodes to take control of the bus or to end the communication

MISO ⟷ MISO
MoSI ⟷ MoSI
SCK ⟷ SCK
GPIO ⟶ CS
CS ⟵ GPIO

➔ SPI Block Diagram (How SPI works?)



→ SPI consists of two shift registers, one in the master and the other in the slave side.

→ Also, there is a clock generator in the master side that generates the clock for the shift registers.

→ Shift registers are 8 bits long. So after 8 clock pulses, the contents of the two shift registers are interchanged.

→ When the master wants to send a byte of data, it places the byte in its shift register and generates 8 clock pulses

→ After 8 clock pulses the byte is transmitted to the other shift register.

→ When the master wants to receive a byte of data, the slave side should place the byte in its shift register, and after 8 clock pulses the data will be received by the master shift register.

→ It must be noted that SPI is full duplex, meaning that it sends and receives data at the same time.

→ SSPSR is the shift register for the SPI module. It shifts data in and out of the device. The data travels in a loop to the next shift register. The data is shifted out the SDO pin of one device and into the SDI pin of the other.

→ SPI creates a data loop between two devices. Data leaving the master exits on the SDO (serial data output) line. Data entering the master enters on the serial data input, SDI line.

→ A clock (SCK) is generated by the master device. It controls when and how quickly data is exchanged between the two devices.

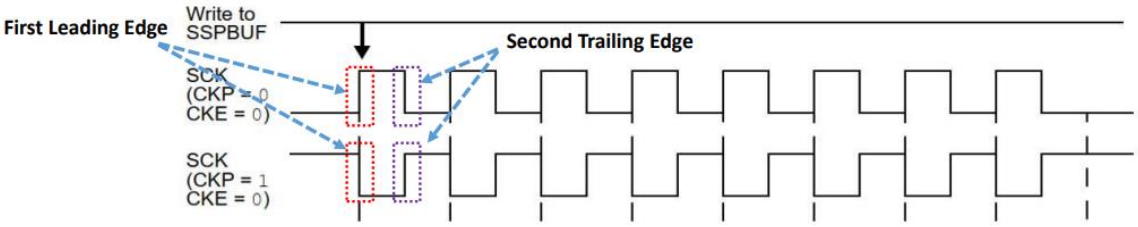
→ SS, allows a master device to control when a particular slave is being addressed. This allows the possibility of having more than one slave and simplifies the communications. When the SS signal goes low at a slave device, only that slave is accessed by SPI.

→ Once a byte of data has been exchanged between the two devices, it is copied to the SSPBUF register. The SSPBUF is then read by the user software. If any data is to be exchanged, this register is written to by your program. Writing to the SSPBUF will transfer the contents of SSPBUF to the SSPSR.

SCK	8	Master	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	Slave
SCK	7	Master	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	Slave
SCK	6	Master	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	Slave
SCK	5	Master	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	Slave
SCK	4	Master	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	Slave
SCK	3	Master	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	Slave
SCK	2	Master	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	Slave
SCK	1	Master	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	Slave

➔ **The Clock Polarity:** Define the polarity of the clock signal during the idle state.

➔ Define the first (leading) edge and the second (trailing) edge



➔ Assume the IDLE state is "Low" :
- The first leading edge is (Rising Edge)
- The second trailing edge is (Falling Edge)

➔ Assume the IDLE state is "High"
- The first leading edge is (Falling Edge)
- The second trailing edge is (Rising Edge)

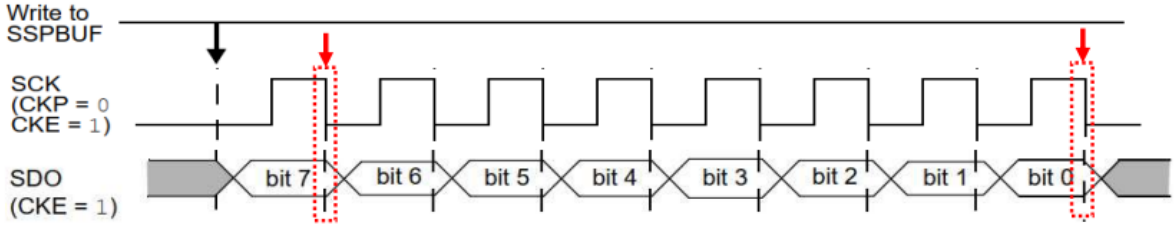
➔ **The clock polarity & phase together determine when will the data be latched on the data line**
- When to "Transmit" the data? ➔ Taking into consideration the clock edge "Clock Polarity"
- When to "Read or Sample" the data ➔(Taking into consideration the clock edge "Clock Phase"

➔ There are around **"4 Modes" on transmitting data:**
- Clock Idle is "Low" and transmitting occurs on transition from active to Idle clock state.
- Clock Idle is "Low" and transmitting occurs on transition from idle to active clock state.
- Clock Idle is "High" and transmitting occurs on transition from active to Idle clock state.
- Clock Idle is "High" and transmitting occurs on transition from idle to active clock state.

SSPCON1	bit 4	**CKP:** Clock Polarity Select bit 1 = Idle state for clock is a high level 0 = Idle state for clock is a low level
SSPSTAT:	bit 6	**CKE:** SPI Clock Select bit(1) 1 = Transmit occurs on transition from active to Idle clock state 0 = Transmit occurs on transition from Idle to active clock state

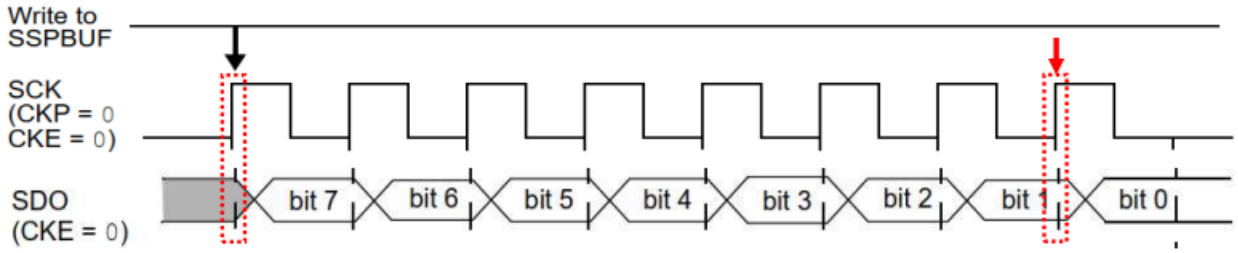
➔➔ **The clock polarity & phase together determine when the data will be latched on the data line**

1. Clock Idle is "**Low**" and transmitting occurs on transition from active **"H" to Idle "L"** clock state.
   - **SSPCON1** < **CKP :Bit-4** > = 0 & **SSPSTAT** < **CKE : Bit-6** > = 1

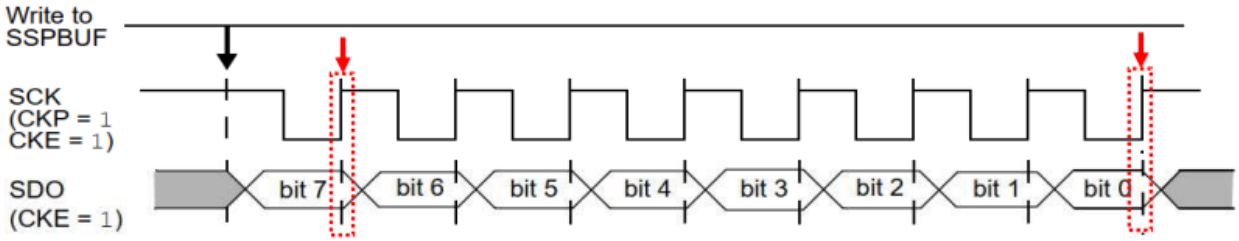


2. Clock Idle is "**Low**" and transmitting occurs on transition from idle **"L" to active "H"** clock state.

- SSPCON1 < **CKP :Bit-4** > = 0 & **SSPSTAT** < **CKE : Bit-6** > = 0

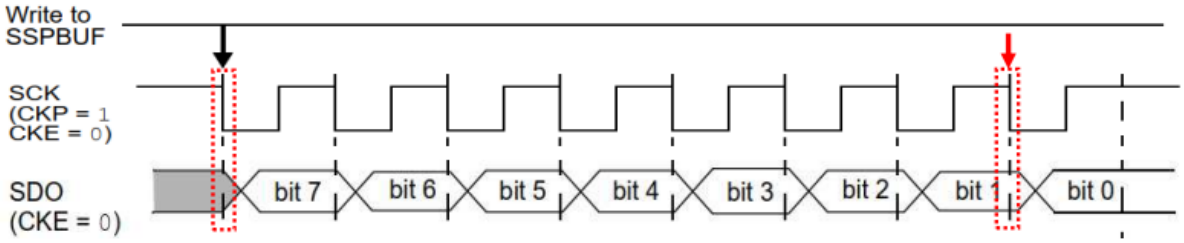


3. Clock Idle is "**High**" and transmitting occurs on transition from active **"L" to Idle "H"** clock state.
   - SSPCON1 < **CKP :Bit-4** > = 1 & **SSPSTAT** < **CKE : Bit-6** > = 1

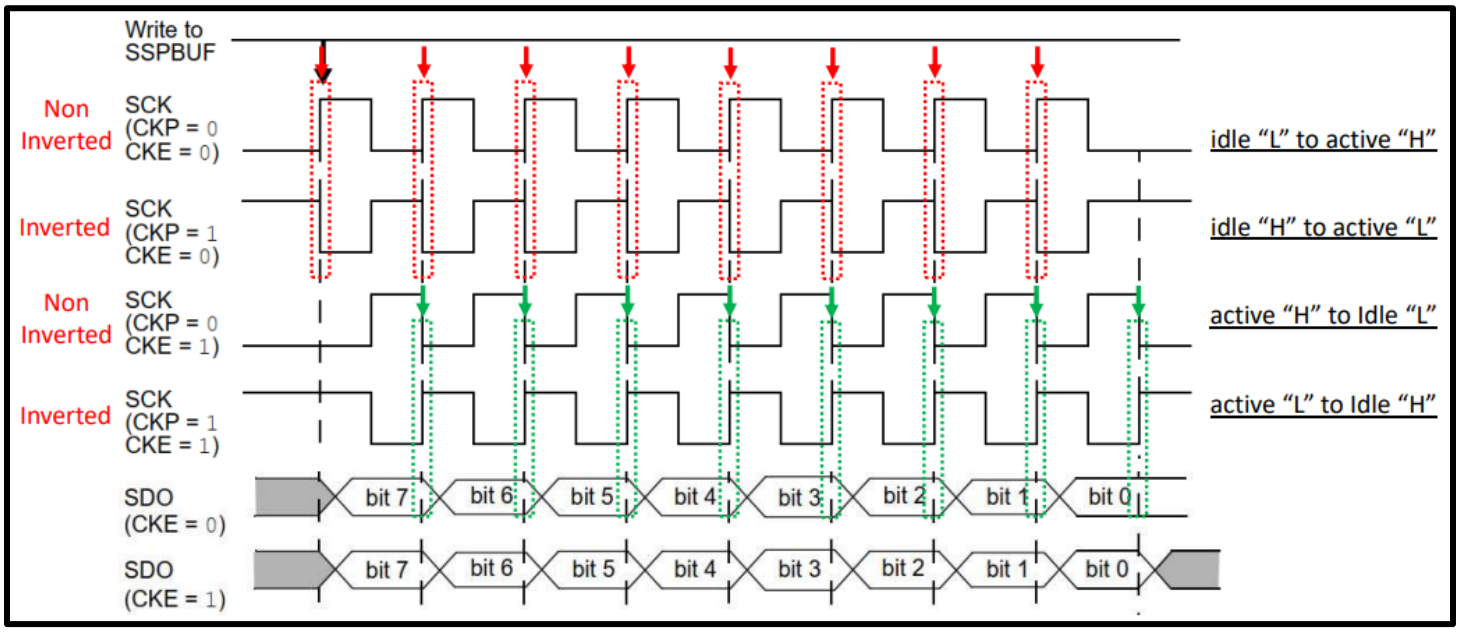


4. Clock Idle is "**High**" and transmitting occurs on transition from idle **"H" to active "L"** clock state.

- SSPCON1 < **CKP :Bit-4** > = 1 & **SSPSTAT** < **CKE : Bit-6** > = 0



➔ **The clock polarity & phase → SPI Master Mode (Transmitting)**



➔ The PIC18F4620 use another "Third Bit" to control the sampling configurations.

- When the device "**Sample**" ➔ Read the received data?

SSPSTAT:

bit 7 **SMP:** Sample bit

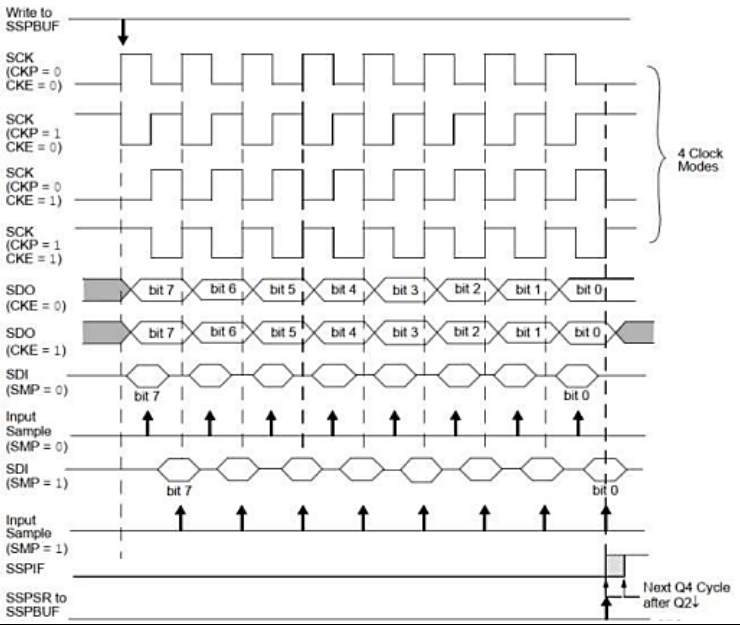
SPI Master mode:

1 = Input data sampled at end of data output time

0 = Input data sampled at middle of data output time

SPI Slave mode:

SMP must be cleared when SPI is used in Slave mode.



Summary:

- **Clock Polarity:**
  - ✓ The clock polarity determines the **idle state** of the clock signal when no data transfer is taking place. It can be configured as either high or low.
- **Idle State:**
  - ✓ The idle state refers to the default state of the clock and data lines when they are not actively transmitting data.
- **Leading Edge:**
  - ✓ The leading edge is the moment when the clock signal transitions from its idle state to an active state, either rising or falling.
  - ✓ When the leading edge occurs, the data on the data line is either sampled or shifted out/in, depending on the clock phase.
- **Trailing Edge:**
  - ✓ The trailing edge refers to the transition of the clock signal from the active state back to the idle state.
  - ✓ It is often used to shift data out or in. For example, if data is shifted out on the leading edge, it is typically shifted in on the trailing edge, and vice versa.
- **Clock Phase:**
  - ✓ The clock phase determines the timing relationship between the data and clock signals. It can be either "sample on leading edge" or "sample on trailing edge".
  - ✓ The specific clock phase is typically defined by the master device and must be set consistently across all devices on the SPI bus.
- **Sample Data:**
  - ✓ Sampling data is the process of reading or capturing the data on the data line at a specific point in time.
  - ✓ the data is typically sampled on the leading or trailing edge of the clock signal, depending on the clock phase setting.
  - ✓ The sampled data is then used for further processing or transfer.
- **Setup Data:**
  - ✓ Setting up data refers to preparing the data on the data line before it is sampled
  - ✓ The data is typically set up before the leading or trailing edge of the clock signal, depending on the clock phase setting.

➔ Master SSP Module Overview (MSSP DataSheet):

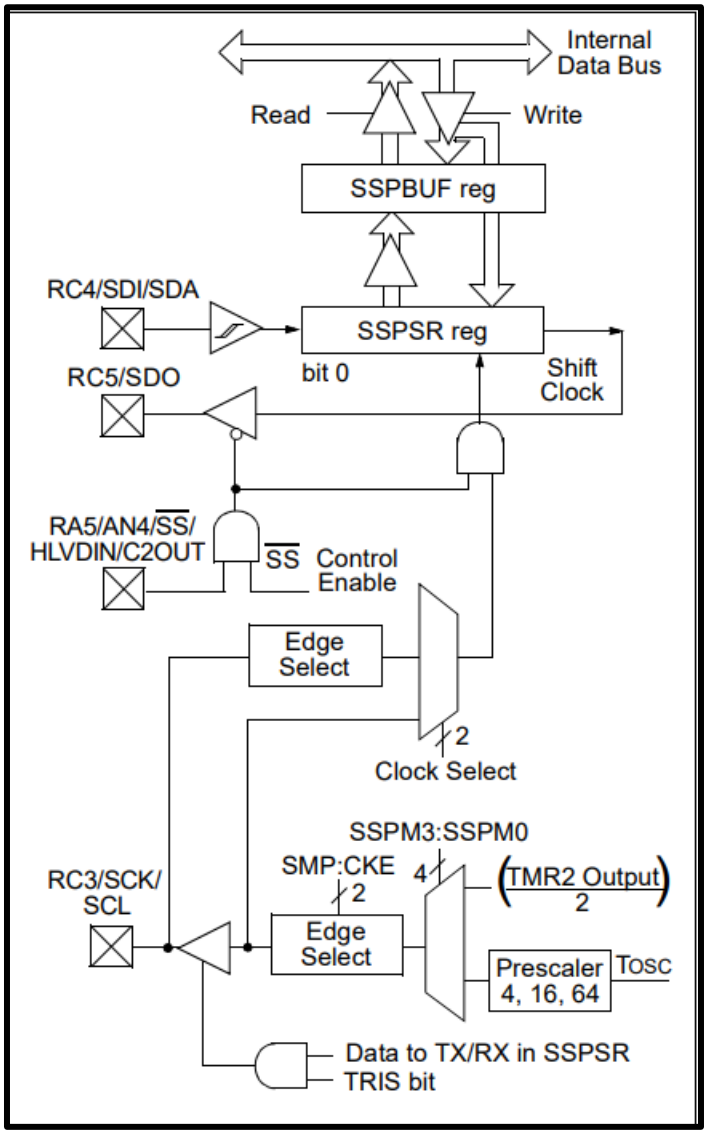
- The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communicating with other peripheral or microcontroller devices.
- These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc.
- The MSSP module can operate in one of two modes:
    1. Serial Peripheral Interface (SPI) 2. Inter-Integrated Circuit (I2C)
        - Full Master mode
        - Slave mode (with general address call)
        - The I2C interface supports the following modes in hardware:
            → Master mode
            → Multi-Master mode
            → Slave mode
- Control Registers
    - The MSSP module has three associated registers.
    - These include a status register (SSPSTAT) and two control registers (SSPCON1 and SSPCON2).
    - The use of these registers and their individual Configuration bits differ significantly depending on whether the MSSP module is operated in SPI or I2C mode.
- The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously
- To accomplish communication, typically four pins are used:
    1. Serial Data Out (SDO) – RC5/SDO
    2. Serial Data In (SDI) – RC4/SDI/SDA
    3. Serial Clock (SCK) – RC3/SCK/SCL
    4. Slave Select (SS) – RA5/AN4/SS/HLVDIN/C2OUT

MSSP BLOCK DIAGRAM (SPI MODE):

Initializing the SPI:

This is done by programming the appropriate control bits (SSPCON1<5:0>) and (SSPSTAT<7:6>). These control bits allow the following to be specified:

- Master Mode (SCK is the clock output)
- Slave Mode (SCK is the clock input)
- Clock Polarity (Idle state of SCK)
- Data input sample phase (middle or end of data output time)
- Clock edge (output data on rising/falling edge of SCK)
- Clock Rate (Master Mode only)
- Slave Select Mode (Slave Mode only)



→ The MSSP consists of a transmit/receive Shift Register (SSPSR) and a Buffer Register (SSPBUF)

→ The SSPSR shifts the data in and out of the device, MSb first.

→ The SSPBUF holds the data that was written to the SSPSR, until the received data is ready

→ Once the 8 bits of data have been received, that byte is moved to the SSPBUF Register. Then the buffer full detect bit, BF (SSPSTAT register), and the interrupt flag bit, SSPIF, are set

→ This double buffering of the received data (SSPBUF) allows the next byte to start reception before reading the data that was just received.

→ Any write to the SSPBUF Register during transmission/reception of data will be ignored, and the write collision detect bit, WCOL (SSPCON1 register), will be set.

→ User software must clear the WCOL bit so that it can be determined if the following write(s) to the SSPBUF Register completed successfully.

→ When the application software is expecting to receive valid data, the SSPBUF should be read before the next byte of data to transfer is written to the SSPBUF

→ Buffer full bit, BF (SSPSTAT register), indicates when SSPBUF has been loaded with the received data (transmission is complete).

→ When the SSPBUF is read, the BF bit is cleared. This data may be irrelevant if the SPI is only a transmitter.

→ Generally the MSSP Interrupt is used to determine when the transmission/reception has completed.

→ The SSPBUF must be read and/or written. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur.

→ The SSPSR is not directly readable or writable, and can only be accessed by addressing the SSPBUF Register. Additionally, the MSSP Status Register (SSPSTAT) indicates the various status conditions

→ The MSSP module has four registers for **SPI** mode operation. These are:
- MSSP Control Register 1 (SSPCON1)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR) – Not directly accessible

→ SSPCON1 and SSPSTAT are the control and status registers in SPI mode operation. The SSPCON1 register is readable and writable. The lower 6 bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

→ SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

→ In receive operations, SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

→ <u>During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.</u>

Enabling SPI I/O:

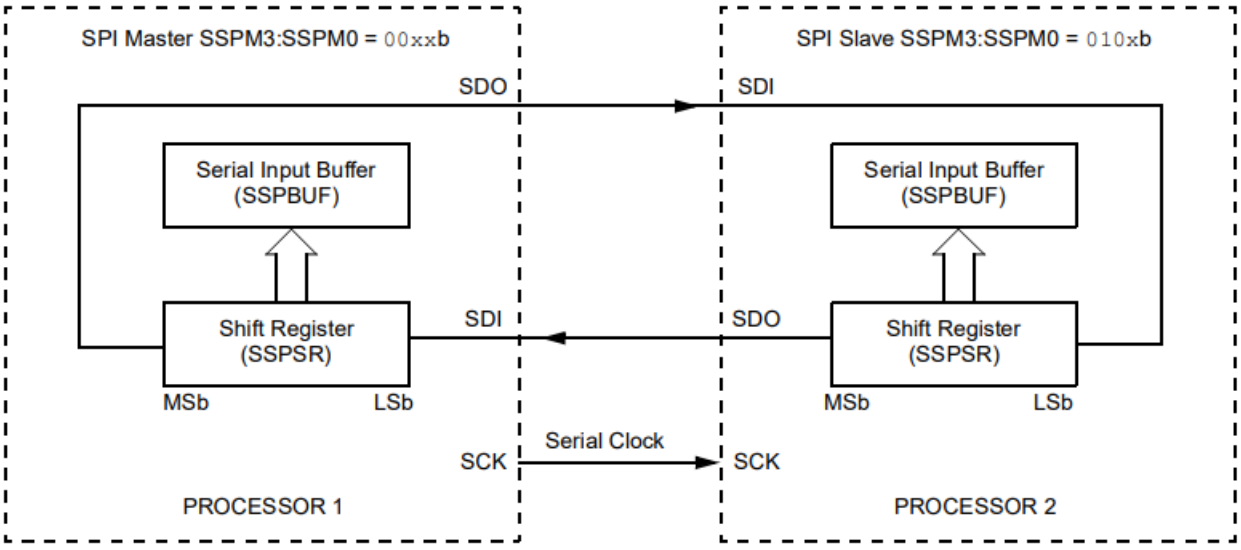
→ To enable the serial port, SSP Enable bit, SSPEN, must be set.

→ To reset or reconfigure SPI mode, clear the SSPEN bit, re-initialize the SSPCON Registers, and then set the SSPEN bit.

→ This configures the SDI, SDO, SCK, and SS pins as serial port pins.

→ For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS Register) appropriately programmed. That is:
- SDI is automatically controlled by the SPI module
- SDO must have the TRIS bit cleared
- SCK (Master Mode) must have the TRIS bit cleared
- SCK (Slave Mode) must have the TRIS bit set
- SS must have the TRIS bit set

SPI Master Mode:



→ The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave is to broadcast data by the software protocol

→ In Master Mode, the data is transmitted/received as soon as the SSPBUF Register is written to.

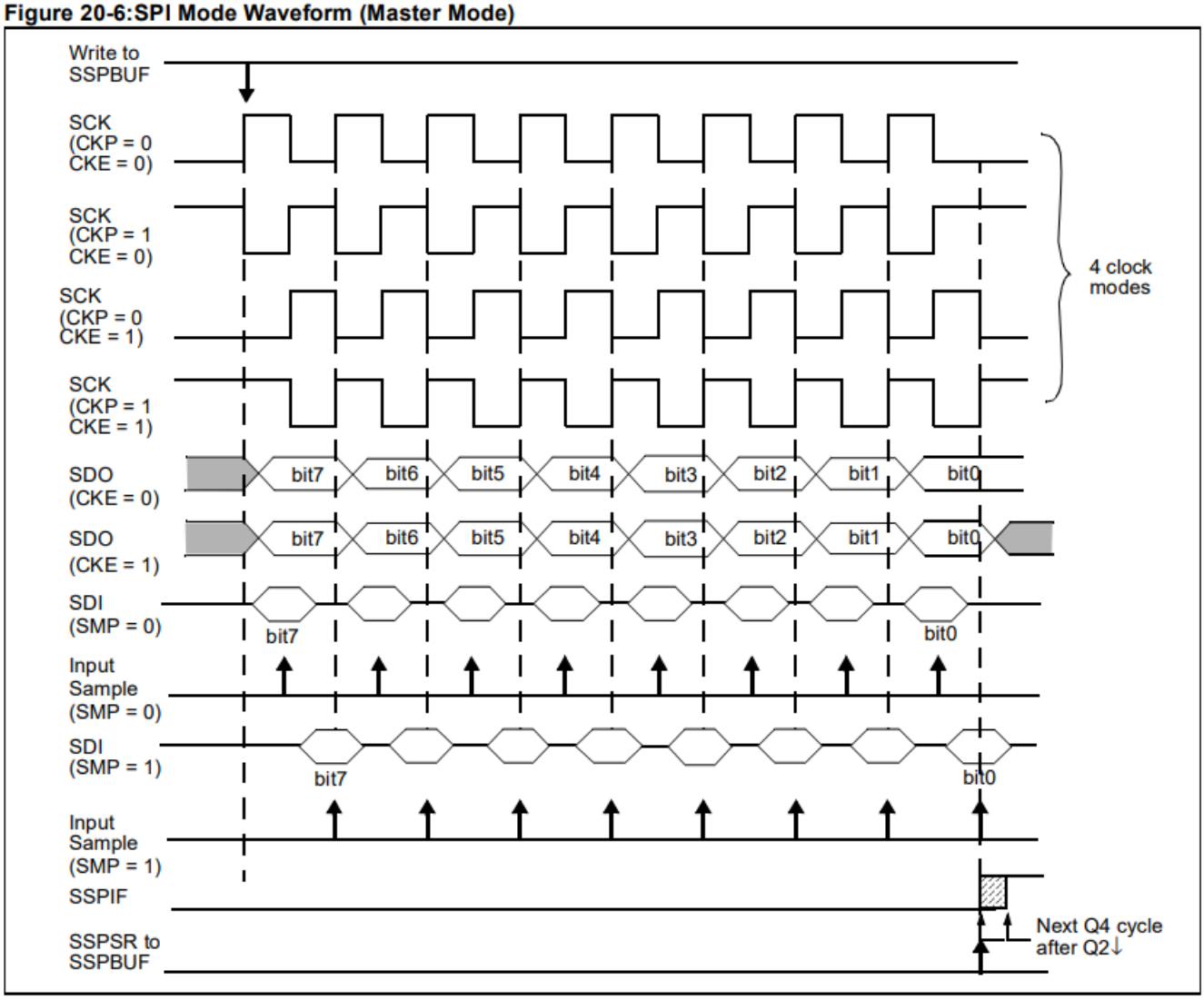
→ If the SPI is only going to receive, the SDO output could be disabled (programmed as an input).

→ The SSPSR Register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF Register (interrupts and status bits appropriately set).

→ The clock polarity is selected by appropriately programming the CKP bit. This gives waveforms for SPI communication where the Msb is transmitted first. In Master Mode, the SPI clock rate (bit rate) is user programmable to be one of the following:
- FOSC/4 (or TCY)
- FOSC/16 (or 4 • TCY)
- FOSC/64 (or 16 • TCY)
- Timer2 output/2

Following figure shows the waveforms for Master Mode. When the CKE bit is set, the SDO data is valid before there is a clock edge on SCK. The change of the input sample is shown based on the state of the SMP bit. The time when the SSPBUF is loaded with the received data is shown.

Figure 20-6:SPI Mode Waveform (Master Mode)



SPI Slave Mode:

→ In Slave Mode, the data is transmitted and received as the external clock pulses appear on SCK. When the last bit is latched, the SSPIF interrupt flag bit is set.

Slave Select Synchronization:

→ The SS pin is a Slave Select pin, and functions similar to a chip select pin.

→ The SPI must be in Slave Mode with SS pin control enabled (SSPCON1<3:0> = 04h).

→ The pin must be configured as an input by setting the corresponding TRIS bit.

→ When the SS pin is low, transmission and reception are enabled and the SDO pin is driven.

→ When the SS pin goes high, the SDO pin is no longer driven, even if in the middle of a transmitted byte, and becomes a floating output. External pull-up/ pull-down resistors may be desirable, depending on the application.

→ If the TRIS bit is cleared, making the pin an output, and the pin outputs a high , the SPI receive logic (slave mode) will be in reset. It will remain in reset until either the pin outputs a low, or the pin's TRIS bit is set and external circuits pull the pin low.

→ **Note 1**: When the SPI is in Slave Mode with SS pin control enabled, (SSPCON<3:0> = 0100) the SPI module will reset if the SS pin is set to VDD.

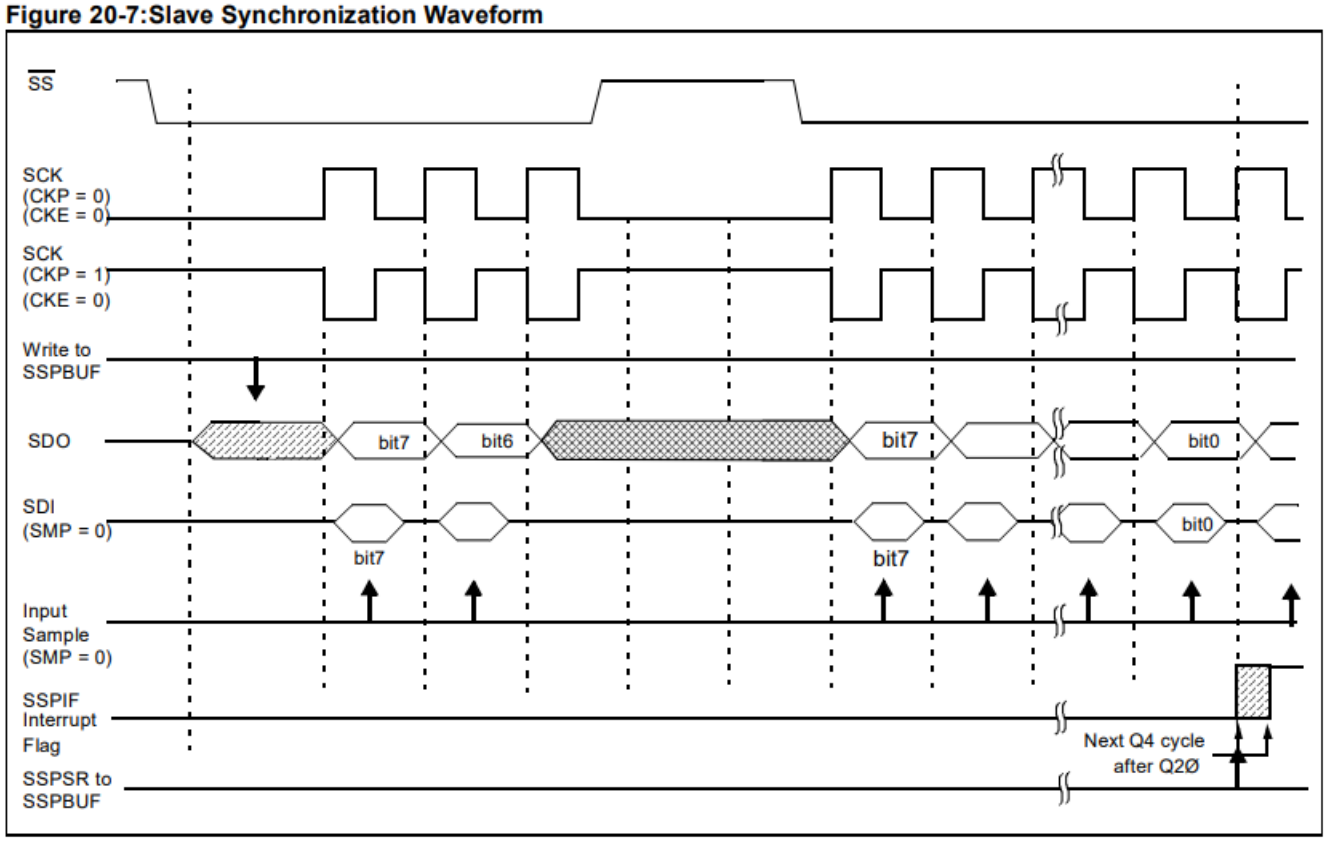
→ **Note 2**: If the SPI is used in Slave Mode with CKE set, then the SS pin control must be enabled

→ When the SPI module resets, the bit counter is forced to 0. This can be done by either by forcing the SS pin to a high level or clearing the SSPEN bit.

→ To emulate two-wire communication, the SDO pin can be connected to the SDI pin. When the SPI needs to operate as a receiver, the SDO pin can be configured as an input. This disables transmissions from the SDO. The SDI can always be left as an input (SDI function) since it cannot create a bus conflict

→ A reset disables the MSSP module and terminates the current transfer.

Figure 20-7:Slave Synchronization Waveform



THE END

Inter integrated circuit (I2C)

* I2C characteristics :-

→ serial protocol : where data is sent bit by bit over a single wire, known as the serial data (SDA) line.

→ synchronous protocol : each bit is synchronized with a clock signal provided by the serial clock (SCL) line.

→ Half duplex - Bidirectional : which means that data can be transmitted in both directions but not simultaneously. the same I2C bus is used for both transmitting and receiving data.

→ Byte-oriented : the I2C protocol operates on a byte level, meaning that data is sent and received in chunks of 8-bits (1 byte) at a time.

→ Two-wire bus :

- the I2C bus consists of two lines : SDA (serial data) and SCL (serial clock). these two wires are shared among multiple devices connected to the bus

→ Multi-master_malti-slave :-

- there is only a master device that controls the communication at a time, and one or more slave devices that respond to the master's commands
- each device on the bus is assigned a unique software address, allowing the master to communicate with individual slaves.
- the master device generates the clock signal used for synchronizing data transfer on the bus
- in case two or more masters try to access the bus simultaneously, collision detection and bus arbitration mechanisms are used to resolve conflicts and prevent data corruption.

→ support multiple speeds :.

1 - Bidirectional bus specification

a) standard mode (SM) up to : 100 Kbit/sec
b) Fast mode (FM) up to : 400 Kbit/sec
c) Fast mode plus (FM+) up to : 1M bit/sec
d) High speed mode (HSM) up to : 3.4 M bit/sec

2- unidirectional bus specification:-

a) ultra fast mode (U-FM) up to: 5M bit/sec

- master to slave or slave to master, only one direction
- Doesn't support multi master option

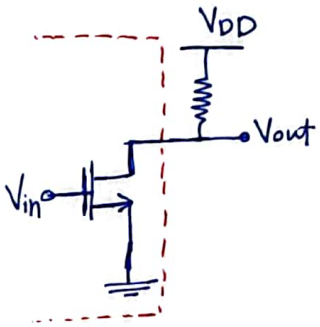
---

* a defining characteristic of I2C is that every device on the bus must connect to both the clock signal (SCL) and the data signal (SDA) via open-drain or (open-collector) output drivers

→ here are three important implications of open-drain configuration:-

a) the signal always default to logic high

- the devices can determine that the bus is available for new transmission by observing that both SCL and SDA have been logic high for a certain amount of time

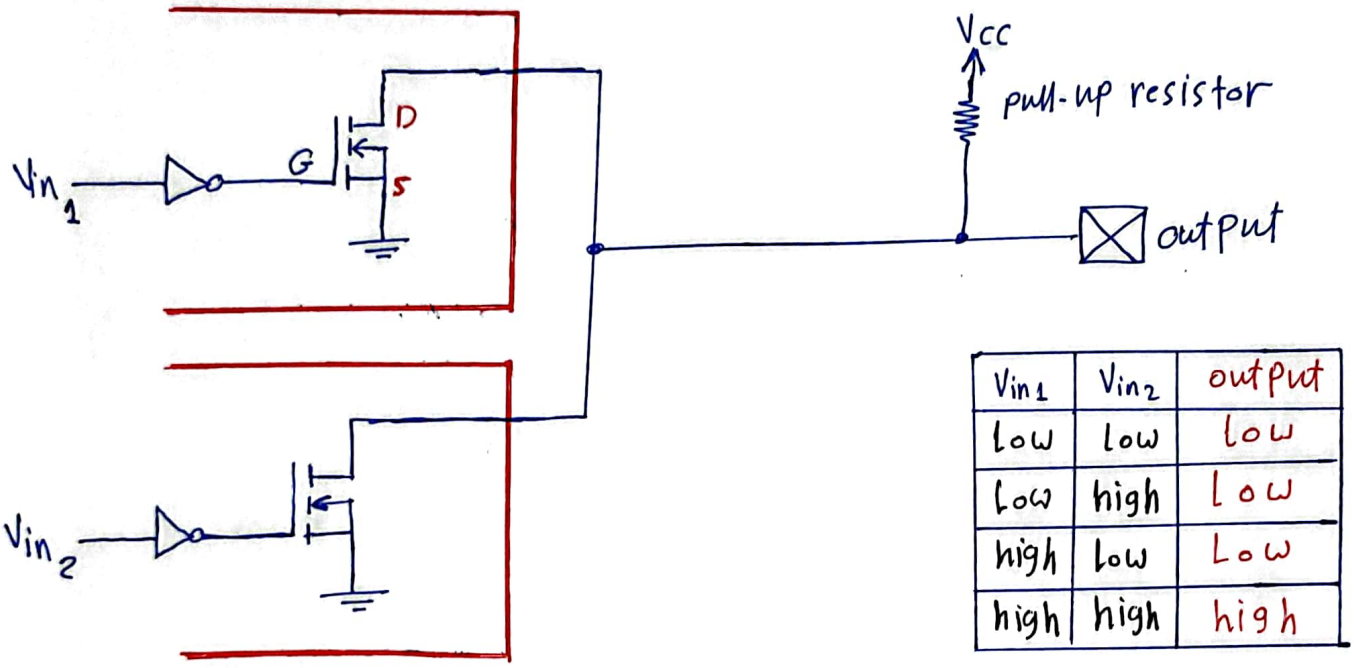


b) any device on the bus can safely drive the signals to logic "0", even if another device is trying to drive them high "And wiring"

- the "and wiring" concept emerges when multiple devices are connected using "open drain" outputs. When any device actively pulls the line low, it effectively brings the line low for all other devices connected to it. However, when no device is actively pulling the line low, the pull-up resistors ensure that the line is pulled high
- this configuration enables multi-master communication, where any device can ~~become~~ take control of the bus by actively driving the line low, while other devices passively monitor the bus state.
- to handle simultaneous transmission attempts, the I2C protocol incorporates (يتضمن) additional mechanisms such as "bus arbitration and collision detection", to ensure that only one device is actively transmitting on the bus at any given time, and preventing data corruption

- During the arbitration, masters monitor the bus and compare their intended logic level with the observed level. if a master detects a mismatch, it realizes that a higher-priority master is currently transmitting and gives up control of the bus

ex:



Vin_1	Vin_2	output
low	low	low
low	high	low
high	low	Low
high	high	high

for example:

Vin_1: 0b0101|0100 → 0: is the dominant bit

Vin_2: 0b0101|1100 → 1: is the ~~recessive~~ bit

→ the master with the dominant bit (actively pulling the line low) has a higher priority and continues its transmission, while others with the ~~recessive~~ bits give up control of the bus.

C) Devices with different supply voltages can coexist on the same bus, as long as the Lower voltage devices will not be damaged by the higher voltage.

- we can do this by pulling up the SCL and SDA to the lowest Voltage using pull-up resistor. For example (3.3 V)
- or we can this by using a Bi-directional logic level converter.

* I2C Frame format :.

[I] start and stop conditions:-

→ all transactions begin with a start and are terminated by a stop condition

→ a high to low transition on the SDA Line, while SCL is High, defines a start condition

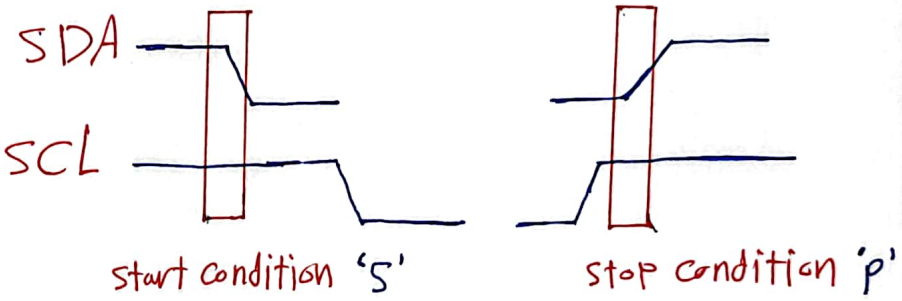
→ a Low to high transition on the SDA Line, while SCL is High, defines a stop condition

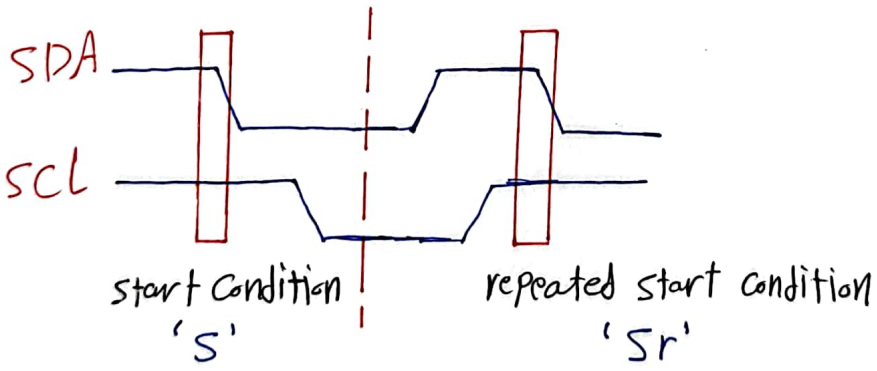
→ start and stop conditions are always generated by the master.

→ the bus is considered to be busy after the start condition.

→ the bus stays busy if a repeated start condition is generated instead of a stop condition

→ the bus is considered to be free again at a certain time after the stop condition.





→ detection of start and stop conditions by devices connected to the bus is easy if they incorporate the necessary interfacing hardware.

→ however, microcontrollers with no such interface have to sample the SDA Line at least twice per clock period to sense the transition

* byte format :

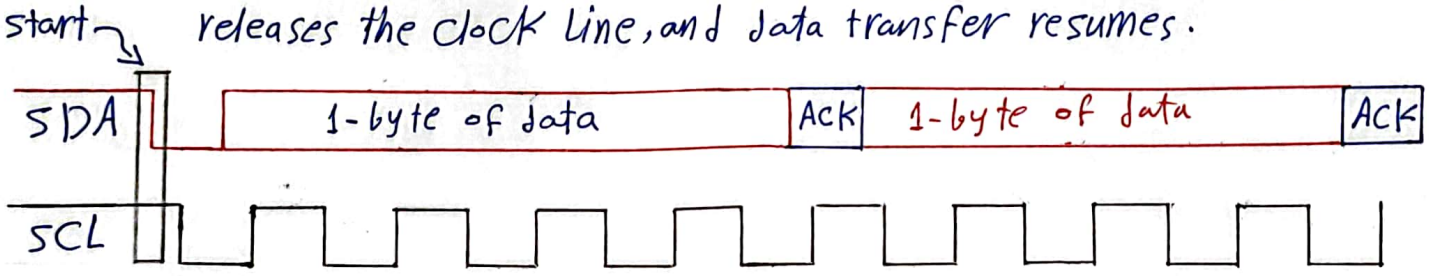
→ every word transmitted on the SDA Line must be 8-bits long → (byte-oriented)

→ the number of bytes that can be transmitted per transfer is unrestricted.

→ Data is transferred with the Most significant bit (MSB) first.

→ each byte must be followed by an acknowledge (ACK) bit → (successful receive)

→ if a slave ~~cannot~~ needs to perform other tasks before receiving or transmitting another byte of data, it can hold the clock line (SCL) Low. this action pauses the data transfer and forces the master device to wait. once the slave is ready for another byte, it releases the clock Line, and data transfer resumes.

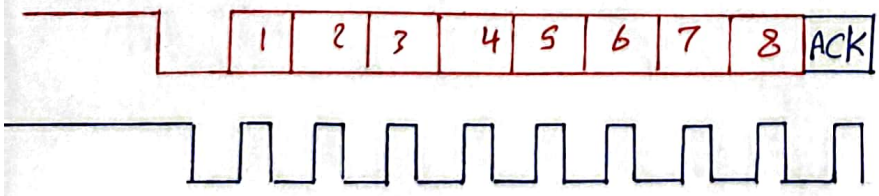


* Data Validity :-

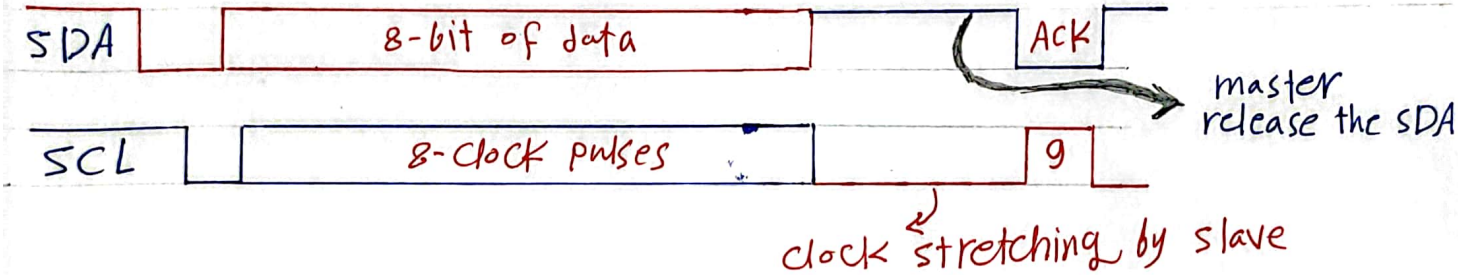
→ the data on the SDA Line must be stable during the High period of the clock. any changes to the data line during this period can lead to data corruption.

→ the state of the data Line (High or low) can only be changed when the clock signal on the SCL Line is Low.

→ for each bit of data transferred, one clock pulse is generated. this means that in order to transfer one byte of data (8-bits), the master device generates 9 clock pulses. the extra clock pulses is used for the acknowledgment bit that is sent by the receiving device.



* Acknowledge (ACK) and Not acknowlege (NACK) :-
  - → the ACK takes place after every byte.
  - → the Ack bit allows the receiver to signal the transmitter that the byte was successfully received
  - → the master generates all clock pulses, including the Ack ninth clock pulse.
  - → the transmitter releases the SDA line during the Ack clock pulse so the receiver can pull the SDA line low and it remains stable low during the High period of this clock pulse.
  - → if a receive device needs to perform other tasks before receiving or transmitting another byte of data, it can hold the SCL line Low to force the master into a wait state (clock stretching)



* Not Acknowledge :.
  - → when SDA remains High during the ninth clock pulse, this is defined as the Not Acknowledge signal (NACK).
  - → when this happens, the master can then generate either a stop condition to abort the transfer or send a repeated start condition to start a new transfer.
  - → there are five conditions that lead to the generation of a (NACK) :-
    1. No receiver is present on the bus with the transmitted address so there is no device to respond with an acknowledge.
    2. the receiver is unable to receive or transmit because it is performing some tasks and is not ready to start communication with the master.
    3. During the transfer, the receiver gets data or commands that it does not understand.
    4. During the transfer, the receiver cannot receive any more data.
    5. A master-receiver must signal the end of the transfer to the slave-transmitter

* Clock synchronization :

→ in an I2C bus system with multiple masters, clock synchronization and arbitration mechanisms are necessary to determine the bus master and complete its transmission.

→ single master systems do not require these mechanisms as there is only one master device.

→ clock synchronization is achieved through the wired-AND connection of I2C interfaces to the SCL line.

→ when the SCL line transitions from High to low, it triggers the masters to start counting their low period.

→ each master has its own internal clock for timing data transmission.

→ once a master's clock reaches a low state, it holds the SCL line until the clock transitions back to High

→ this keeps the SCL Line low for the master's low period, allowing data transmission.

→ if another master's clock is still within its low period when one master tries to transition the SCL Line from Low to High, the SCL Line may not immediately change state.

→ in this case, the SCL Line is held Low by the master with the longest Low period

→ masters with shorter Low periods enter a High wait-state and pause transmission until the SCL Line transitions to the High state

→ all masters count off their Low period, when they have finished counting, the clock line is released and transitions to the High state.

→ at this point, there is no difference between the master clocks and the state of the SCL Line.

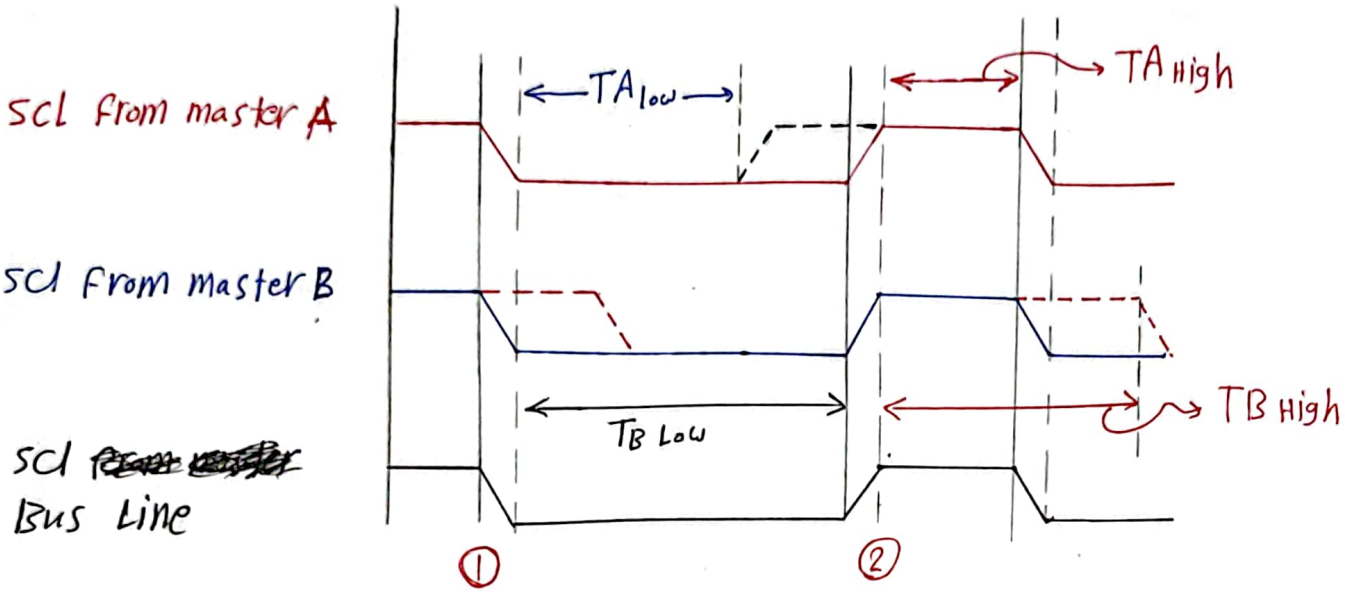
→ all masters start counting their High periods simultaneously

→ the first master to complete its High period pulls the SCL Line back Low.

→ this process generates a synchronized SCL clock

→ the Low period of the SCL clock is determined by the master with the longest clock Low period

→ the High period of the scl clock is determined by the master with the shortest clock High period
→ starting from the second clock, the scl clock will be synchronized.



Point ① : masters start counting Low period
Point ② : masters start counting High period

---

* The slave address and R/$\bar{W}$ bit:

① 7-Bit slave address mode:-

→ Data transfers follow the format shown :-

1- Start Condition (S) : initiates the data transmission.
2- Sending a 7-Bit slave address : the master sends the 7-Bit address of the target slave device
3- 8th bit (R/$\bar{W}$) : Following the 7-bit address, an additional bit, the R/$\bar{W}$ bit, is transmitted.
   → '0' : master needs to send data to the slave (write)
   → '1' : master needs to receive data from the slave (read)
4- Data transfer termination : every data transfer concludes with a stop Condition (P)
5- optional repeated start condition (Sr) : if the master wishes to continue communicating on the bus with another slave without ending the current session, it can generate a repeated start Condition (Sr) and address another slave without preceding it with a stop Gnditlon.

* the first byte transmitted after the start condition can result in one of the following data transfer formates:

1- Master-transmitter to slave-receiver :· (unchanged direction)
→ in this format, the master-transmitter sends data to the slave-receiver.
→ the slave-receiver acknowledges each byte sent by the master

■ → From slave to master
A → acknowledge (SDA low)
Ā → not acknowledge (SDA High)
S → start condition
P → stop condition

2- Master-receiver to slave transmitter :· (unchanged direction)
→ Master reads from the slave immediately after the first byte
→ at the moment of the first acknowledge, the master-transmitter becomes a master-receiver and the slave-receiver becomes a slave transmitter.
→ the first acknowledge is still generated by the slave.
→ the master generates subsequent acknowledges.
→ the stop condition is generated by the master, which sends a not-ack (Ā) just before the stop condition (donot need to read data anymore)

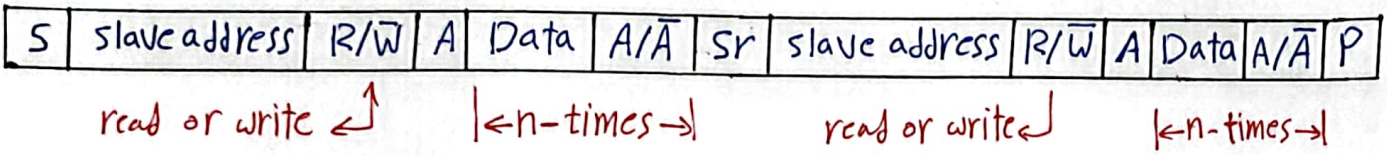
■ → From slave to master

3- Combined format :·
→ it allows for a change of direction within a data transfer.
→ in this format, both the start condition and the slave address are repeated.
→ the (R/W̄) bit is reserved to indicate the change in direction.
→ the combined format allow the master to change the direction of data transfer in the middle of a transfer.
→ to change the direction, the master sends a repeated start condition, indicating the change

→ for example: if the initial transfer was a write operation, the master can later send a repeated start condition with the same slave address but the (R/$\bar{W}$) bit reserved to perform a read operation.

→ when a master-receiver sends a repeated start condition, it sends a not-acknowledge ($\bar{A}$) before the repeated start condition.

→ the not-acknowledge indicates the end of the previous transfer and prepares the slave for a new transfer with a different direction.



* 10-bit addressing characteristics :-

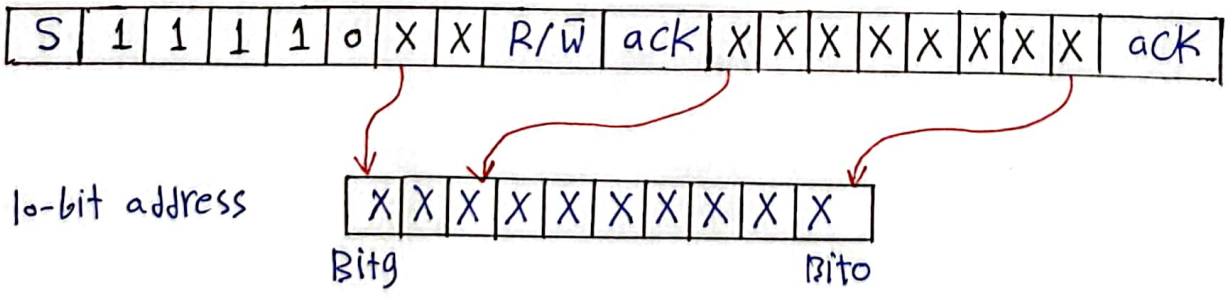
→ 10-bit addressing expands the number of possible addresses.

→ devices with 7-bit and 10-bit addresses can be connected to the same I2C bus, and both can be used in all bus speed mode.

→ the 10-bit slave address is formed the first two bytes following a start condition or a repeated start condition.

→ the first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two MSB of the 10-bit address.

→ the 8th bit of the first byte is the R/$\bar{W}$ bit that determines the direction of the message.

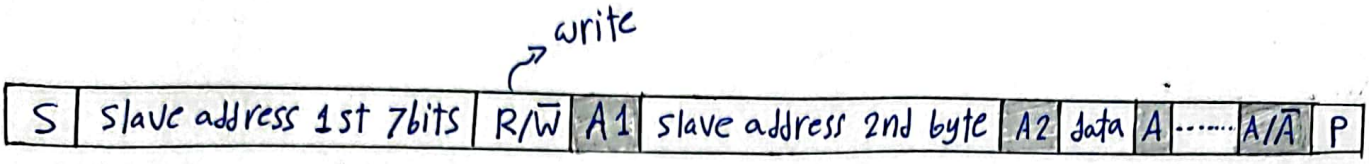


* Possible data transfer format :

1- A master-transmitter transmits to slave-receiver with a 10-bit add

2- A master-receiver reads slave-transmitter with a 10-bit address

1- A master- transmitter :-



▭ : from slave to master.

2- A master-receiver :-

→ the transfer direction is changed after the second R/W bit

→ the procedure remains the same as that described for a master-transmitter addressing a slave-receiver until the ack bit A2.

→ after repeated start condition (Sr), a matching slave remembers that it was addressed before.

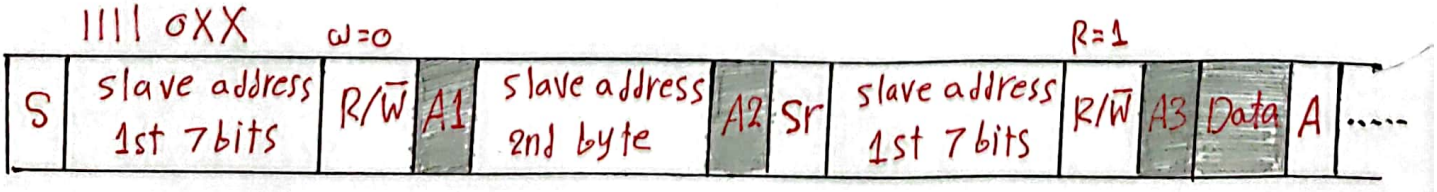
→ then, this slave checks if the first seven bits of the first byte of the slave address following Sr are the same as they were after the start condition S, and tests if the 8th bit (R/W) is 1 → read

→ if there is a match, the slave considers that it has been addressed as a transmitter and generates acknowledge A3.

→ the slave-transmitter remains addressed until it receives a stop condition (P) or until it receives another repeated start condition (Sr) followed by a different slave address.

→ after a repeated start condition, all the other slave devices will also compare the first seven bits of the first byte of the slave address (1111 0XX) with their own addresses and test the 8th (R/W̄) bit.

→ However, none of them will be addressed because R/W=1 → for 10-bit address devices " to address a new device as a slave for the first time, the first (R/W) bit must be = 0 "write" " or the (1111 0XX) slave address (for 7-bit devices) does not match.



▭ → from slave to master.

➔ I2C Reserved Addresses:

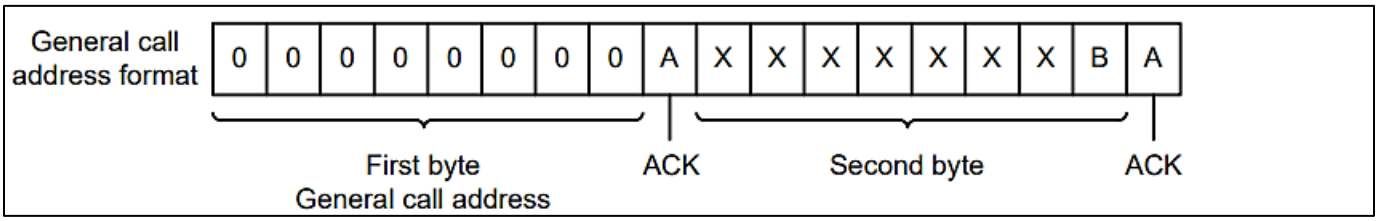
- Two groups of eight addresses (0000 XXX and 1111 XXX) are reserved for the purposes shown in table:

Target Address	R/W Bit	Description
000 0000	**0**	General call address
000 0000	**1**	START byte
000 0001	**X**	CBUS address
000 0010	**X**	Reserved for different bus format
000 0011	**X**	Reserved for future purposes
000 01XX	**X**	Hs-mode controller code
111 11XX	**1**	Device ID
111 10XX	**x**	10-bit target addressing

✓ The general call address is used for several functions including software reset.
✓ No device is allowed to acknowledge at the reception of the START byte.

➔ General call address:

- Addressed as 0x00 write
- used to address all devices on the I2C-bus simultaneously
- Second byte contains command
- Not all devices are designed to respond to the general call address.
- Devices can acknowledge or ignore the general call address based on their need for the data.
- Can be used for several functions including RESET
- The meaning of the general call address is specified in the second byte.



➔ Cases to Consider:

- **Case 1: When the least significant bit B (LSB of the second byte) is 'zero',** the second byte may be used to send commands to those devices receiving the general call. For example,
  - 06h: **Reset** and write programmable part of slave address by hardware.
  - 04h: Write programmable part of slave address by hardware. Behaves as above, but the device does not reset.
  - 00h: Not allowed as the second byte.
- **Case 2: When the least significant bit B is 'one'.**
  - The 2-byte sequence is a 'hardware general call'.
    - This means that the sequence is transmitted by a hardware master device, such as a keyboard scanner, which can be programmed to transmit a desired slave address.

- The second byte contains the address of the hardware master.

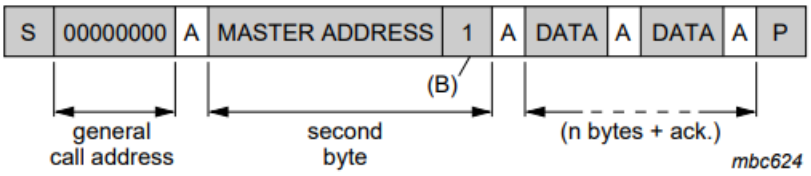


Fig 17. Data transfer from a hardware master-transmitter

➔ Software reset:

- Following a General Call, (0000 0000), sending 0000 0110 (06h) as the second byte causes software reset. This feature is optional and not all devices respond to this command.
- On receiving this 2-byte sequence, all devices designed to respond to the general call address reset and take in the programmable part of their address.
- Precautions must be taken to avoid blocking the bus with low levels on SDA or SCL lines after applying the supply voltage.

➔ START byte:

- **Connection Options for Microcontrollers:**
  - **Microcontrollers can be connected to the I2C-bus in two ways:**
    - Microcontroller **with on-chip hardware I2C-bus interface**.
    - Microcontroller without on-chip interface, **relying on software polling.**
  - Hardware devices with on-chip interfaces are faster than microcontrollers relying on software polling.
  - Software polling requires frequent monitoring of the bus, reducing the microcontroller's available processing time.
- **Start Procedure for Data Transfer:**
  - In the case of software polling, data transfer can involve a longer start procedure.
  - The start procedure includes the following steps:
    - START condition (**S**)
    - START byte (**0000 0001)**
    - Acknowledge clock pulse (**ACK**) → **SDA** : **NACK**
    - Repeated START condition (**Sr**)

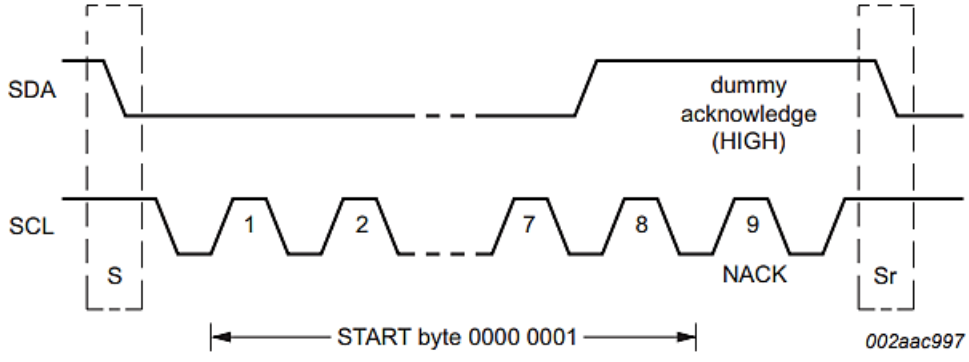


Fig 19. START byte procedure

✓ **START Byte**

- Addressed as 0x00 read
- After the START condition S has been transmitted by a master which requires bus access, the START byte (0000 0001) is transmitted.
- Another microcontroller can therefore **sample the SDA line at a low sampling rate** until one of the seven zeros in the START byte is detected
- After detection of this LOW level on the SDA line, the microcontroller can switch to a higher sampling rate to find the repeated START condition Sr which is then used for synchronization.
- A hardware receiver resets upon receipt of the repeated START condition Sr and therefore ignores the START byte.
- An acknowledge-related clock pulse is generated after the START byte.
  - ✓ No device is allowed to acknowledge the START byte.

➔ Device ID:

- The Device ID field is an optional **3-byte read-only (24 bits)** word giving the following information:
  - Twelve bits with the manufacturer name, unique per manufacturer (for example, NXP)
  - Nine bits with the part identification, assigned by manufacturer (for example, PCA9698)
  - Three bits with the die revision, assigned by manufacturer (for example, RevX)

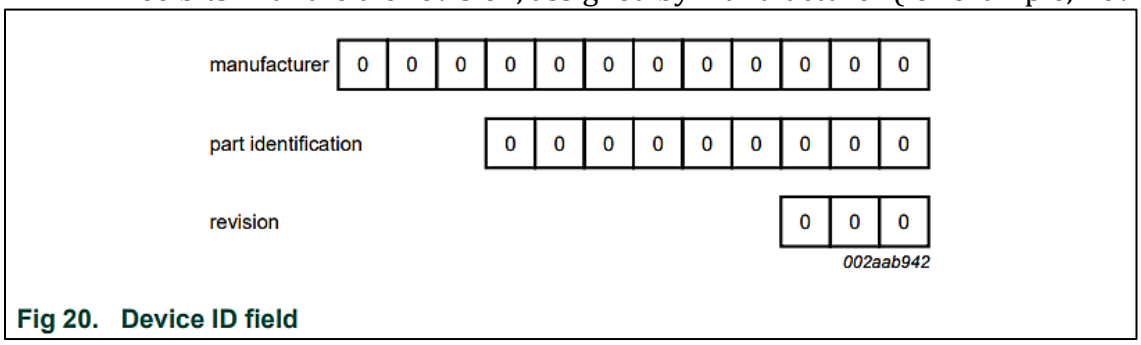


Fig 20. Device ID field

- The Device ID is **read-only**, hard-wired in the device and can be accessed as follows:
  - START condition
  - The master sends the Reserved Device ID I2C-bus address followed by the R/W bit set to '0' (write): '1111 1000'.
  - The master sends the I2C-bus slave address of the slave device it must identify. The LSB is a 'Don't care' value. Only one device must acknowledge this byte (the one that has the I2C-bus slave address).
  - The master sends a Re-START condition. Remark: A STOP condition followed by a START condition resets the slave state machine and the Device ID Read cannot be performed. Also, a STOP condition or a Re-START condition followed by an access to another slave device resets the slave state machine and the Device ID Read cannot be performed.
  - The master sends the Reserved Device ID I2C-bus address followed by the R/W bit set to '1' (read): '1111 1001'.
  - The Device ID Read can be done, starting with the 12 manufacturer bits (first byte + four MSBs of the second byte), followed by the nine part identification bits (four LSBs of the second byte + five MSBs of the third byte), and then the three die revision bits (three LSBs of the third byte).
  - The master ends the reading sequence by NACKing the last byte, thus resetting the slave device state machine and allowing the master to send the STOP condition. Remark: The reading of the Device ID can be stopped anytime by sending a NACK.

- If the master continues to ACK the bytes after the third byte, the slave rolls back to the first byte and keeps sending the Device ID sequence until a NACK has been detected.

Table 4. Assigned manufacturer IDs

Manufacturer bits												Company
11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	NXP Semiconductors
0	0	0	0	0	0	0	0	0	0	0	1	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	0	1	0	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	0	1	1	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	1	0	0	Ramtron International
0	0	0	0	0	0	0	0	0	1	0	1	Analog Devices
0	0	0	0	0	0	0	0	0	1	1	0	STMicroelectronics
0	0	0	0	0	0	0	0	0	1	1	1	ON Semiconductor
0	0	0	0	0	0	0	0	1	0	0	0	Sprintek Corporation
0	0	0	0	0	0	0	0	1	0	0	1	ESPROS Photonics AG
0	0	0	0	0	0	0	0	1	0	1	0	Fujitsu Semiconductor
0	0	0	0	0	0	0	0	1	0	1	1	Flir
0	0	0	0	0	0	0	0	1	1	0	0	O_2Micro
0	0	0	0	0	0	0	0	1	1	0	1	Atmel

I2C Datasheet and Block Diagram (How I2C works?)

➔ The MSSP module in I2C mode fully implements all master and slave functions
  → **including general call support** and **provides interrupts on Start and Stop bits** in hardware to determine a free bus (**multi-master function**)

➔ The MSSP module implements the standard mode specifications **(100 Kbit/Sec)** as well **as 7-bit and 10-bit addressing**

➔ **Data Transfer Pins:**
  → Two pins are used for data transfer in the I2C communication protocol. These pins are:
    - **SCL** Pin (Clock) – RC3/SCK/SCL
    - **SDA** Pin (Data) – RC4/SDI/SDA

➔ **Pin Configuration:** In order to enable I2C mode**, the SDA and SCL pins must be configured as inputs or outputs** in the corresponding **TRIS**.

➔ **MSSP Module Functions:**
  → The MSSP module functions are enabled by **setting the MSSP Enable bit (SSPEN) in the SSPCON register**.

➔ The **MSSP module has six registers** for I2C operation. These registers are:

1. MSSP Control Register1 (**SSPCON1**)
2. MSSP Control Register2 (**SSPCON2**)
3. MSSP Status Register (**SSPSTAT**)
4. Serial Receive/Transmit Buffer (**SSPBUF**)
5. MSSP Shift Register (**SSPSR**) - Not directly accessible
6. MSSP Address Register (**SSPADD**)

→ SSPCON1, SSPCON2 and SSPSTAT are the control and status registers in I2C mode operation.

→ The SSPCON1 and SSPCON2 registers are readable and writable.

→ The lower 6 bits of the SSPSTAT are read-only. The upper two bits of the SSPSTAT are read/write.

→ SSPSR is the shift register used for shifting data in or out.

→ SSPBUF is the buffer register to which data bytes are written to or read from.

→ SSPADD register holds the slave device address when the MSSP is configured in I2C Slave mode.

→ When the MSSP is configured in Master mode, the lower seven bits of SSPADD act as the Baud Rate Generator reload value.

➔ **I2C Operation Control:**

→ The I2C interface supports the following modes in hardware:

- Master mode
- Multi-Master mode
- Slave mode

→ The **SSPCON1** register allows control of the I2C operation. It has four mode selection bits (SSPCON1<3:0>) that allow selection of different I2C modes:

1. I2C Slave Mode (7-bit address)
2. I2C Slave Mode (10-bit address)
3. I2C Master Mode, clock = **OSC**/4 (**SSPADD** + 1)
4. I2C Slave Mode (7-bit address) with Start and Stop bit interrupts enabled
5. I2C Slave Mode (10-bit address) with Start and Stop bit interrupts enabled
6. I2C Firmware controlled master operation, slave is idle

➔ Selection of any I2C mode with the **SSPEN** bit set, forces the **SCL** and **SDA** pins to be **open-drain**, provided these pins are programmed to **inputs** by setting the appropriate **TRISC** bits.

➔ To ensure proper operation of the module**, pull-up** resistors must be provided **externally** to the **SCL** and **SDA** pins.

➔ **Data Transfer Status:**

→ The **SSPSTAT** register provides information about the status of the data transfer

- It includes detection of a Start or Stop bit, specifies if the received byte was data or address, if the next byte is the completion of a 10-bit address, and if the transfer is a read or write operation.

➔ **Data Transfer Registers:**

1. **SSPBUF**: This register is used to write or read transfer data.
2. **SSPSR**: This register shifts the data in or out of the device.

→ In receive operations, the **SSPBUF** and **SSPSR create a double buffered receiver**, allowing the reception of the next byte to begin before reading the current byte of received data.

→ When a complete byte is received, it is transferred to the **SSPBUF** register and the **SSPIF** bit is set.

→ If another complete byte is received before the **SSPBUF** register is read, a **receiver overflow occurs**, and the **SSPOV** bit in the **SSPCON1** register is **set**, causing the byte in the **SSPSR** to be lost.

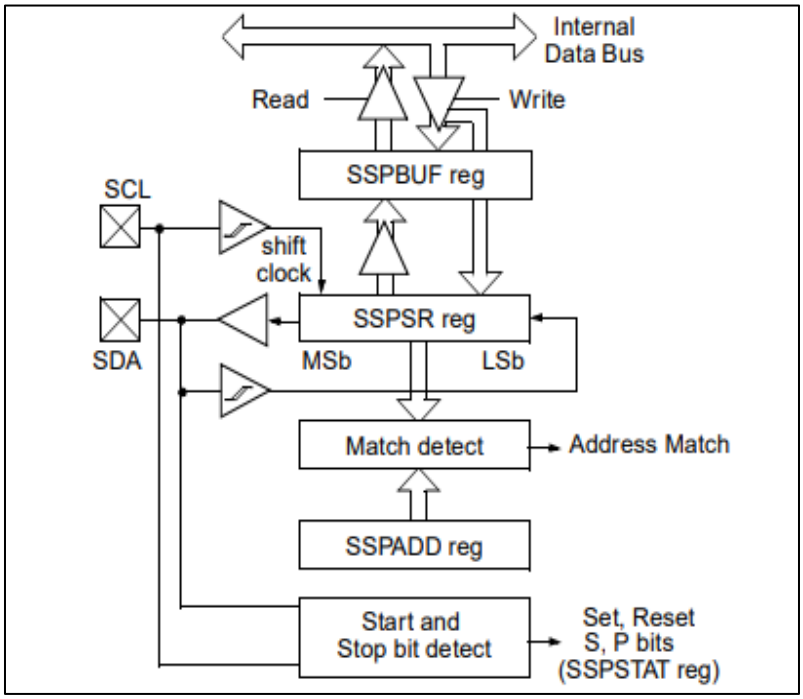
➔ **Slave Address:**

→ The **SSPADD** register holds the slave address.

→ In 10-bit mode, the user needs to write the high byte of the address **(1111 0 A9 A8 0).**

→ After the high byte address match, **the low byte of the address (A7:A0) needs to be loaded.**

➢ I2C Slave Mode Block Diagram:



→ In Slave mode, the **SCL** and **SDA** pins must be configured as **inputs** (**TRISC**<4:3> set). The MSSP module **will override the input state with the output data when required (slave-transmitter).**

→ The I2C Slave mode hardware will always generate an interrupt on an address match. Through the mode select bits, the user can also choose to interrupt on Start and Stop bits.

→ When an address is matched, or the data transfer after an address match is received, the hardware automatically will generate the **Acknowledge** (**ACK**) pulse and load the **SSPBUF** register with the received value currently in the **SSPSR** register.

→ Any combination of the following conditions will cause the **MSSP** module not to give this **ACK** pulse:

- The Buffer Full bit, **BF** (**SSPSTAT**<0>), was set before the transfer was received.
- The overflow bit, **SSPOV** (**SSPCON1**<6>), was set before the transfer was received.

→ In this case, the **SSPSR** register value is not loaded into the **SSPBUF**, but bit, **SSPIF** (**PIR1**<3>), is set.

→ The **BF** bit is cleared by reading the **SSPBUF** register, while bit, **SSPOV**, is cleared through software.

→ The SCL clock input must have a minimum high and low for proper operation.

→ The high and low times of the I2C specification, as well as the requirement of the MSSP module, are shown in timing parameter 100 and parameter 101.

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock high time	100 kHz mode	4.0	—	µs	PIC18CXXX must operate at a minimum of 1.5 MHz
			400 kHz mode	0.6	—	µs	PIC18CXXX must operate at a minimum of 10 MHz
			SSP Module	1.5TCY	—	ns	
101	TLOW	Clock low time	100 kHz mode	4.7	—	µs	PIC18CXXX must operate at a minimum of 1.5 MHz
			400 kHz mode	1.3	—	µs	PIC18CXXX must operate at a minimum of 10 MHz

➔ Addressing:

- Once the MSSP module has been enabled, it waits for a **Start condition** to occur. Following the **Start condition**, the 8 bits are shifted into the **SSPSR** register.
- All incoming bits are sampled with the **rising edge** of the clock (**SCL**) line.
- The value of register **SSPSR**<7:1> is compared to the value of the **SSPADD** register.
- The address is compared on the **falling edge** of the **eighth clock** (**SCL**) pulse.
- If the addresses match and the BF and SSPOV bits are clear, the following events occur:
  - ✓ The SSPSR register value is loaded into the SSPBUF register.
  - ✓ The Buffer Full bit, BF, is set.
  - ✓ An ACK pulse is generated.
  - ✓ MSSP Interrupt Flag bit, SSPIF (PIR1<3>), is set (interrupt is generated, if enabled) on the falling edge of the ninth SCL pulse.
- In 10-Bit Addressing mode, two address bytes need to be received by the slave.
- The five Most Significant bits (**MSbs**) of the first address byte specify if this is **a 10-bit address**
- Bit **R/W (SSPSTAT<2>)** must specify **a write** so the slave device will receive the second address byte.
- For **a 10-bit address**, **the first byte would equal '11110 A9 A8 0',** where **'A9'** and **'A8'** are the **two MSbs of the address.**
- The sequence of events for 10-bit addressing is as follows, with steps 7 through 9 for the slave-transmitter:
  1. Receive **first (high) byte** of address (**bits SSPIF, BF and UA (SSPSTAT<1>) are set**).
  2. Update the **SSPADD** register with **second (low) byte of address** (**clears bit, UA, and releases the SCL line**).
  3. Read the **SSPBUF** register (**clears bit, BF**) and **clear flag bit, SSPIF**.
  4. Receive **second (low) byte of address** (**bits, SSPIF, BF and UA, are set**).
  5. Update the **SSPADD** register with **the first (high) byte of address**. If match **releases SCL** line, this will **clear bit, UA.**
  6. Read the **SSPBUF** register (**clears bit, BF**) and **clear flag bit, SSPIF**.
  7. Receive **Repeated Start condition**.
  8. **Receive first (high) byte of address (bits, SSPIF and BF, are set).**
  9. **Read the SSPBUF register (clears bit, BF) and clear flag bit, SSPIF.**

➔ Reception:

- When **the R/W (Read/Write) bit** of the address byte is **clear** (indicating a **write** operation) and an address match occurs, the **R/W bit** of the **SSPSTAT** register is cleared.
- The received address is then loaded into the **SSPBUF** register, and the **SDA** line is held **low** to indicate an **acknowledgment** (**ACK**).
- An **MSSP interrupt** is generated for each data transfer byte.
- The software must **clear** the flag bit **SSPIF** (**PIR1**<3>) to acknowledge the interrupt.
- The status of the byte can be determined by checking the **SSPSTAT** register.
- If **SEN** (**Start Condition Enable**) is enabled (**SSPCON2<0> = 1**), the **RC3/SCK/SCL** line will be **held low (clock stretch)** following each data transfer.
- The clock stretch must be **released by setting the bit CKP (SSPCON<4>).**

➔ Transmission:

➢ Address Match and R/W Bit:

- When the **R/W (Read/Write)** bit of the incoming address byte is **set** (indicating a read operation) and an address match occurs, the **R/W bit of the SSPSTAT register** is **set**.
- The received address is loaded into the **SSPBUF register**.
- The **ACK** pulse is sent on the **ninth bit**, and **the RC3/SCK/SCL pin is held low**, **regardless of SEN (Start Condition Enable).**
- Clock stretching is used **to prevent the master from asserting** another clock pulse until the slave is done preparing the transmit data.

➢ Loading Transmit Data:

- To transmit data, the transmit data must be loaded into the **SSPBUF** register, which also loads the **SSPSR** register.
- The **RC3/SCK/SCL** pin should be enabled by **setting** the **CKP** bit (**SSPCON1**<4>).
- The **eight data** bits are shifted out on the **falling edge** of the **SCL** input, ensuring that the **SDA** signal is valid during **the SCL high time**.

➢ ACK Pulse and Data Transfer Completion:

- The **ACK** pulse from the master-receiver is latched on the **rising edge** of the **ninth SCL input pulse.**
- If the **SDA** line is **high** (**not ACK**), **the data transfer is complete.**
- In this case, when the **ACK** is latched by the slave, the slave logic is **reset**, and the slave monitors for **another occurrence of the Start bit.**
- If the **SDA** line **was low (ACK),** the next transmit data must be loaded into the **SSPBUF** register.
- Again, the **RC3/SCK/SCL** pin must be **enabled by setting the CKP bit**.

➢ MSSP Interrupt and Status:

- **An MSSP interrupt** is generated for each data transfer byte.
- The **SSPIF** bit must be **cleared in software**.
- The **SSPSTAT** register is used to determine the status of the byte.
- The **SSPIF** bit is **set** on **the falling edge** of the **ninth clock pulse**.

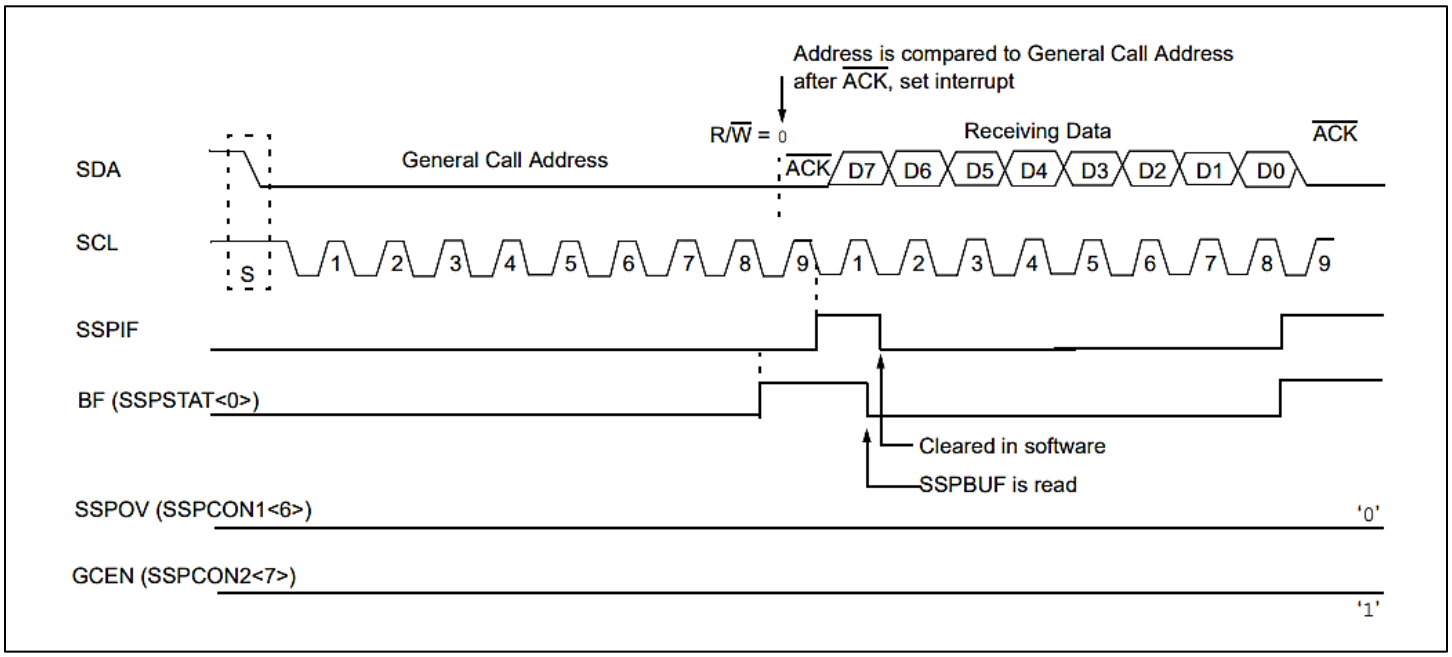
➔ GENERAL CALL ADDRESS SUPPORT:

1. **Addressing Procedure:**
   - In the I2C bus, **the first byte** after the Start condition typically determines the slave device addressed by the master.
   - An exception is the **general call address**, which can address all devices on the bus.
   - When the general call address is used, all devices should theoretically respond with an Acknowledge.
2. **General Call Address:**
   - The general call address **is one of eight addresses reserved** for specific purposes in the I2C protocol.
   - It consists of all '0's with R/W (Read/Write) bit set to 0.
3. **Recognition of General Call Address:**
   - The general call address is recognized when the General Call Enable bit, **GCEN**, is enabled (**SSPCON2<7> is set**).

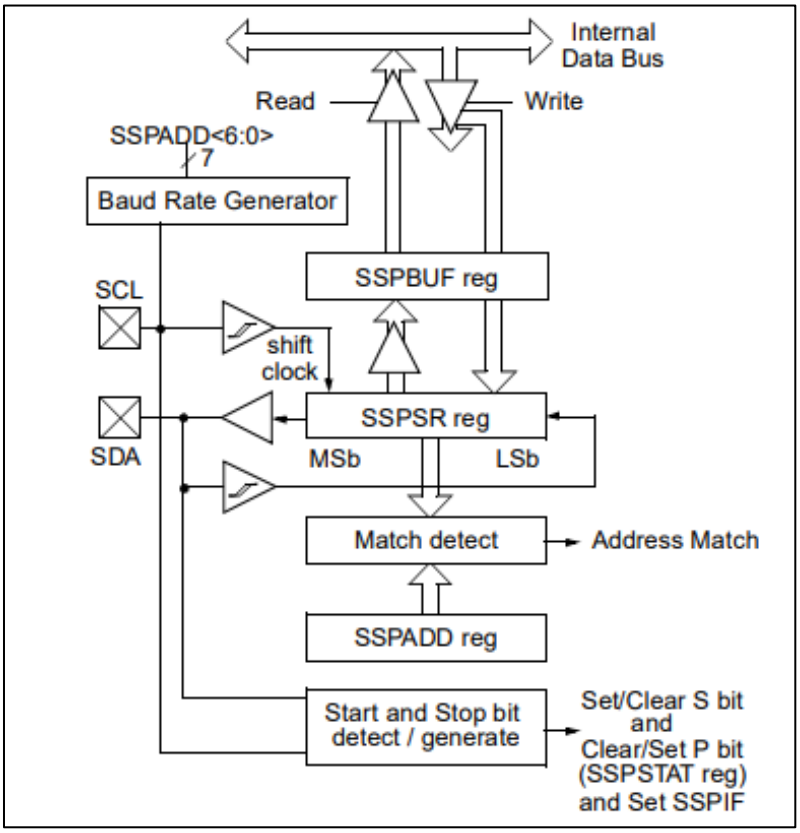
- After detecting a Start bit, 8 bits are shifted into the **SSPSR**, and the address is compared against the **SSPADD.**
- It is also compared to the general call address and fixed in hardware.

4. **Matching General Call Address:**
   - If the general call address matches, the contents of the **SSPSR** are transferred to the **SSPBUF**.
   - The **BF** (Buffer Full) flag bit is **set** (eighth bit), and on the falling edge of the ninth bit (**ACK** bit), the **SSPIF** (SSP Interrupt Flag) is set.
5. **Handling the Interrupt:**
   - When the interrupt is serviced, the source of the interrupt can be checked by reading the contents of the **SSPBUF**.
   - The value can be used to determine if the address was device-specific or a general call address.
6. **10-Bit Mode Consideration:**
   - In **10-bit mode**, updating the **SSPADD** is required for the second half of the address to match, and the **UA** (**Update Address**) bit (**SSPSTAT**<1>) is set.
   - If the general call address is sampled while the **GCEN** bit is **set**, and the slave is configured in **10-Bit Addressing mode**, the second half of the address is not necessary.
   - The **UA** bit **will not be set**, and the slave will begin receiving data after the **Acknowledge**.

➔ **SLAVE MODE GENERAL CALL ADDRESS SEQUENCE (7 OR 10-BIT ADDRESSING MODE):**



➢ I2C Master Mode Block Diagram:



→ **Master Mode** of operation is supported by interrupt generation on the detection of the **Start and Stop conditions**.

→ The **Stop (P) and Start (S)** bits are **cleared** when a reset occurs or when the MSSP module is disabled.

→ Control of the I2C bus may be taken when the **P bit is set**, or **the bus is idle with both the S and P bits clear**

→ In Master Mode, the **SCL** and **SDA** lines are manipulated by the MSSP hardware

→ The following events will cause SSP Interrupt Flag bit, SSPIF, to be set (SSP Interrupt if enabled):

- Start condition
- Stop condition
- Data transfer byte transmitted/received
- Acknowledge Transmit
- Repeated Start

→ **Multi-Master Mode:**

- **Interrupt Generation and Bus Free Detection:**
  - ✓ In Multi-Master Mode, the detection of Start and Stop conditions generates interrupts, allowing determination of when the bus is free.
  - ✓ The Stop (P) and Start (S) bits are cleared upon reset or when the MSSP module is disabled.
  - ✓ Control of the I2C bus can be taken when the P bit (SSPSTAT register) is set or when both the S and P bits are clear.

- ✓ Enabling the MSSP Interrupt while the bus is busy will generate an interrupt when the Stop condition occurs.

- **Monitoring SDA Line for Arbitration:**
  - ✓ In multi-master operation, the SDA line must be monitored for arbitration to check if the signal level matches the expected output level.
  - ✓ This arbitration check is performed in hardware, and the result is stored in the BCLIF bit.
- **States where Arbitration can be Lost:**
  - ✓ Arbitration can be lost in the following states:
    - Address transfer
    - Data transfer
    - Start condition
    - Repeated Start condition
    - Acknowledge condition
- **I2C Master Mode Support:**
  - ✓ Master Mode is enabled by **setting** and **clearing** the appropriate **SSPM** bits in **SSPCON1** and by **setting** the **SSPEN** bit.
  - ✓ Once Master Mode is enabled, the user has **six options for controlling the I2C bus**:
    - Assert a Start condition on **SDA** and **SCL**.
    - Assert a **Repeated Start** condition on **SDA** and **SCL**.
    - Write to the **SSPBUF** Register to initiate transmission of data/address.
    - Generate a **Stop condition on SDA and SCL**.
    - Configure the I2C port to receive data.
    - Generate an **acknowledge** condition at the end of a received byte of data.

- **Note on Event Queuing:**
  - ✓ The MSSP Module, when configured in I2C Master Mode, does not allow queuing of events.
  - ✓ For example, the user cannot initiate a Start condition and immediately write the **SSPBUF** Register to initiate transmission before the Start condition is complete.
  - ✓ In such cases, the **SSPBUF** will not be written, and the **WCOL** (Write Collision) bit will be **set**, indicating that the write to the **SSPBUF** did not occur.
- **I2C Master Mode Operation:**
  - ✓ The master device is responsible for generating all serial clock pulses as well as the **Start and Stop conditions**.
  - ✓ A transfer is ended with either a **Stop condition or a Repeated Start condition**, and the bus is not released with a Repeated Start condition.
  - ✓ In master transmitter mode, serial data is output through **SDA**, and **SCL** outputs the serial clock.
  - ✓ The first transmitted byte **contains the slave address of the receiving device (7 bits) and the Read/Write (R/W) bit, with the R/W bit set to '0'.**
  - ✓ Serial data is transmitted **8 bits at a time**, and after each byte, an **Acknowledge** bit is received. **Start and Stop conditions indicate the beginning and end of a serial transfer**.

- ✓ In **master receive mode**, the first transmitted byte contains the slave address of the transmitting device (7 bits) and the R/W bit, with the **R/W bit set to '1'.**
- ✓ Serial data is received via the **SDA** pin, and **SCL** outputs the serial clock. Serial data is received 8 bits at a time, and after each byte, an Acknowledge bit is transmitted. Start and Stop conditions indicate the beginning and end of transmission.

- **Setting SCL Clock Frequency:**
  - ✓ **The baud rate generator used for SPI mode operation is now used to set the SCL clock frequency for I2C operation.**
  - ✓ The **SSPADD** Register's lower 7 bits contain the reload value for the baud rate generator.
  - ✓ The baud rate generator starts counting automatically upon writing to the **SSPBUF**, and it stops counting and keeps the **SCL** pin in its last state after completing the operation (i.e., transmission of the last data bit followed by ACK).
- **Typical Transmit Sequence:**
  - ✓ The typical transmit sequence involves the following steps:
    - The user generates a Start condition by setting the Start enable bit, **SEN** (SSPCON2 register).
    - **SSPIF** is set, and the MSSP module waits for the required start time before any other operation takes place.
    - The user loads the **SSPBUF** with the address to transmit.
    - The address is shifted out through the **SDA** pin until all 8 bits are transmitted.
    - The MSSP module shifts in the **ACK** bit from the slave device and writes its value into the **SSPCON2** Register.
    - The MSSP module **generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.**
    - The user loads the **SSPBUF** with eight bits of data.
    - The data is shifted out through the **SDA** pin until all 8 bits are transmitted.
    - The MSSP module shifts in the **ACK** bit from the slave device and writes its value into the **SSPCON2** Register.
    - The MSSP module **generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.**
    - The user generates a Stop condition by setting the Stop enable bit, **PEN** (SSPCON2 register).
    - An interrupt is generated once the Stop condition is complete.

→ **Baud Rate Generator:**

- In I2C Master Mode, the reload value for the **BRG** is located in the lower 7 bits of the **SSPADD** Register (Figure 20-18).
- When the **BRG** is loaded with this value, the **BRG** counts down to 0 and stops until another reload has taken place.
- The **BRG** count is **decremented twice per instruction cycle (TCY)** on the Q2 and Q4 clocks
- In I2C Master Mode, the **BRG** is reloaded automatically. If clock arbitration is taking place for instance, the BRG will be reloaded when the **SCL** pin is sampled **high** (Figure 20-19)**.**

Figure 20-18: Baud Rate Generator Block Diagram

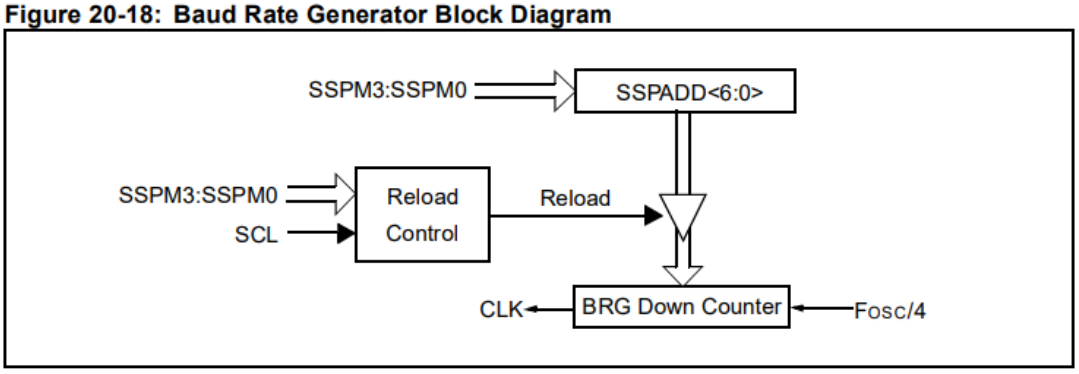
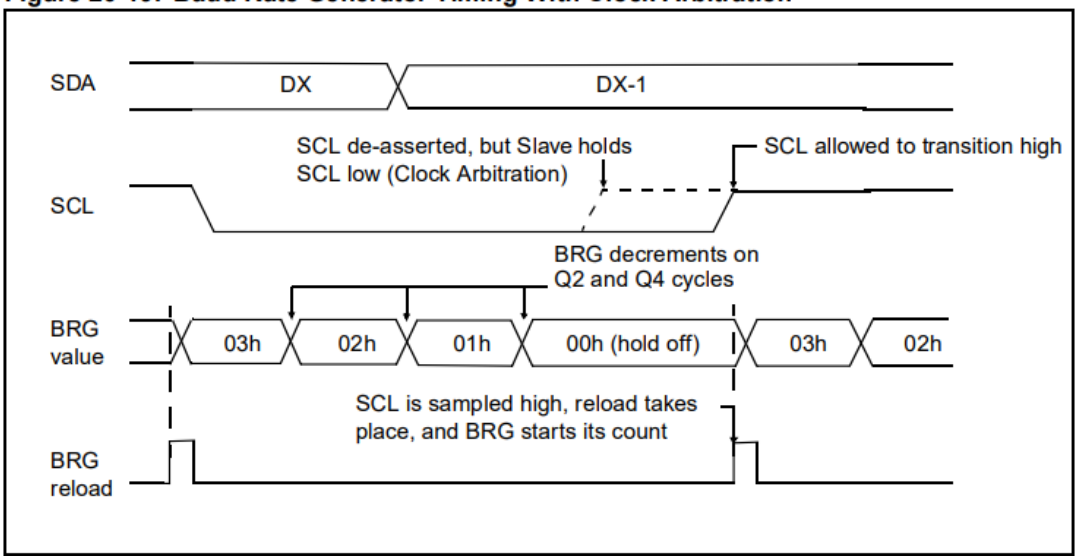


Figure 20-19: Baud Rate Generator Timing With Clock Arbitration



➔ I2C MASTER MODE TRANSMISSION

- → Transmission of a data byte, a 7-bit address, or the other half of a 10-bit address is performed by writing a value to the SSPBUF register.
- → Writing to SSPBUF sets the Buffer Full flag bit (BF) and starts the Baud Rate Generator for the next transmission.
- → Each bit of address/data is shifted out onto the SDA pin after the falling edge of SCL.
- → SCL is held low for one Baud Rate Generator rollover count (**TBRG**), and data should be valid before SCL is released high.
- → After the eighth bit is shifted out, the BF flag is cleared and the master releases SDA.
- → The slave device being addressed can respond with an ACK bit during the ninth bit time if an address match occurred or if data was received properly.
- → The status of ACK is written into the ACKDT bit on the falling edge of the ninth clock.
- → If the master receives an Acknowledge, the Acknowledge Status bit (ACKSTAT) is cleared. If not, the bit is set.
- → After the ninth clock, the SSPIF bit is set, and the master clock (Baud Rate Generator) is suspended until the next data byte is loaded into the SSPBUF.

➔ BF Status Flag:

- → In Transmit mode, the BF bit (SSPSTAT<0>) is set when the CPU writes to SSPBUF and is cleared when all 8 bits are shifted out.

➔ WCOL Status Flag:

- → If the user writes to SSPBUF while a transmit is already in progress, the WCOL flag is set, and the contents of the buffer remain unchanged.
- → If SSPBUF is rewritten within 2 TCY, the WCOL bit is set, and SSPBUF is updated.
- → The user should verify that the WCOL flag is clear after each write to SSPBUF to ensure correct transfer.

➔ ACKSTAT Status Flag:

- → In Transmit mode, the ACKSTAT bit (SSPCON2<6>) is cleared when the slave has sent an Acknowledge (ACK = 0) and is set when the slave does not Acknowledge (ACK = 1).
- → A slave sends an Acknowledge when it has recognized its address or when it has properly received its data.

➔ I2C MASTER MODE RECEPTION:

- → Master mode reception is enabled by setting the Receive Enable bit (**RCEN**) in SSPCON2.

Note: The MSSP module must be in an Idle state before setting the RCEN bit; otherwise, the RCEN bit will be disregarded.

- → The Baud Rate Generator starts counting, and on each rollover, the state of the SCL pin changes and data is shifted into the SSPSR.
- → After the falling edge of the eighth clock, the receive enable flag is automatically cleared, and the contents of the SSPSR are loaded into the SSPBUF.
- → The BF flag bit is set, the SSPIF flag bit is set, and the Baud Rate Generator is suspended from counting, holding SCL low.
- → The MSSP is now in an **idle** state waiting for the next command.
- → When the buffer is read by the CPU, the BF flag bit is automatically cleared.
- → The user can then send an Acknowledge bit at the end of reception by setting the Acknowledge Sequence Enable bit (ACKEN) in SSPCON2.

➔ BF Status Flag:

- → In receive operation, the BF bit is set when an address or data byte is loaded into SSPBUF from SSPSR. It is cleared when the SSPBUF register is read.

➔ SSPOV Status Flag:

- → In receive operation, the SSPOV bit is set when 8 bits are received into the SSPSR, and the BF flag bit is already set from a previous reception.

➔ WCOL Status Flag:

- → If the user writes to SSPBUF while a receive is already in progress, the WCOL bit is set, and the contents of the buffer remain unchanged.