University of San Carlos | Department of
COMPUTER ENGINEERING

CpE 3201
Embedded Systems

Inter-Integrated Circuit Communication (I^2C)

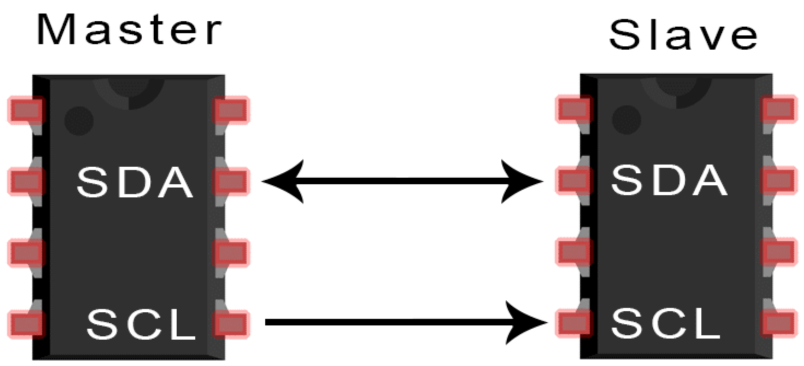
I^{2}C Communication

- **Inter-Integrated Circuit (I^{2}C or I2C)** is also a widely used communication protocol that are used in interfacing OLED displays, barometric/pressure sensors, and gyroscope/accelerometer.
- I^{2}C is a **communication protocol** that uses features ‘combining’ UART and SPI.
- It follows the **Master – Slave model** and uses 2 wires same as UART.

I^{2}C Communication

- I2C is also a **serial** communication protocol, which means that data is transferred **bit by bit**.
- Like SPI, I2C is **synchronous**, the bits of the data transferred is **synchronized with the clock shared by the master to the slave**.

I^{2}C Configuration



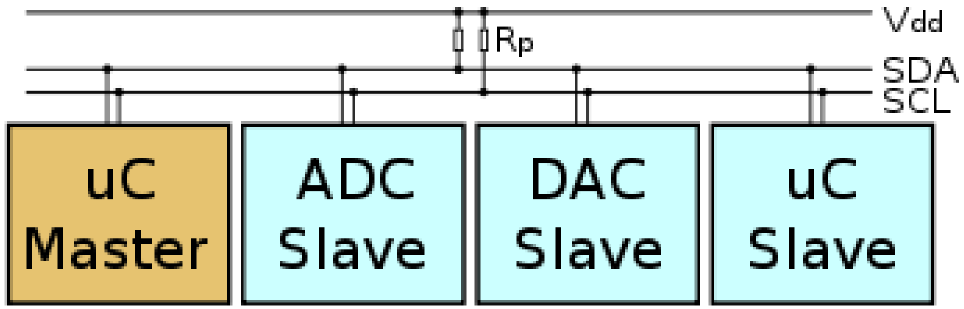
- I^{2}C has **2 pins** called SDA and SCL.
- **Serial Data (SDA)** - is the line for the master and slave data transfer.
- **Serial Clock (SCL)** – is the line that carries the clock signal

I^{2}C Terminology

- **Transmitter** - This is the device that transmits data to the bus
- **Receiver** - This is the device that receives data from the bus
- **Master** - This is the device that generates clock, starts communication, sends I^{2}C commands and stops communication
- **Slave** - This is the device that listens to the bus and is addressed by the master

I^2C Bus

- I^2C bus is popular because it is **simple to use**, there **can be more than one master**, **only upper bus speed is defined** and **only two wires with pull-up resistors are needed** to connect almost unlimited number of I^2C devices.



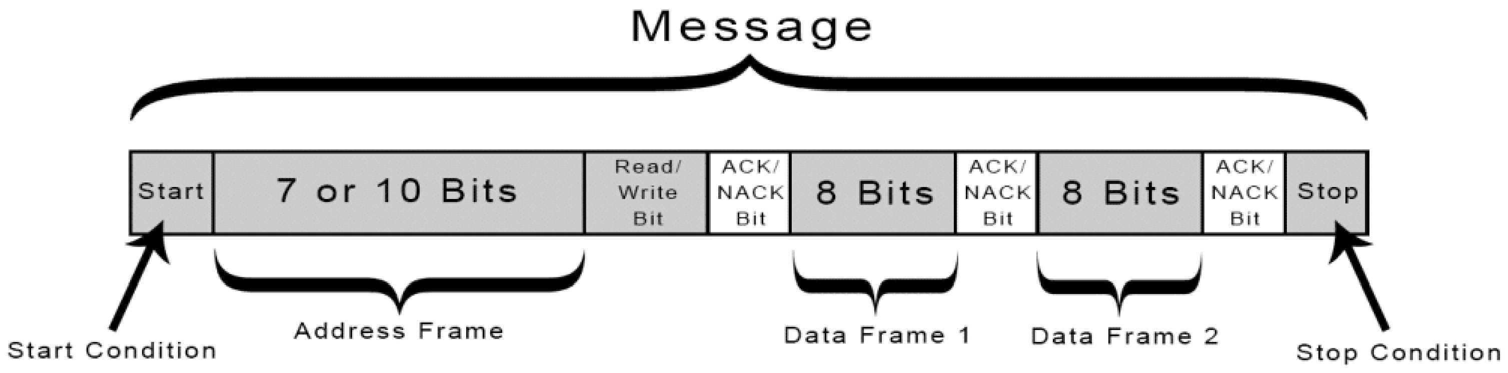
- I^2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.

I^{2}C Bus

- Each slave device has a **unique address**. Transfer from and to master device is **serial** and it is **split into 8-bit frames**.
- All these simple requirements make it very simple to implement I^{2}C interface even with cheap microcontrollers that have no special I^{2}C hardware controller.
- **Only two (2) free I/O pins needed** and **few simple I^{2}C routines** to send and receive commands.

I^{2}C Data Message

- Same as UART, data sent by the master is in **messages with start and stop bits**. A message are broken up into multiple **8-bit frames**.

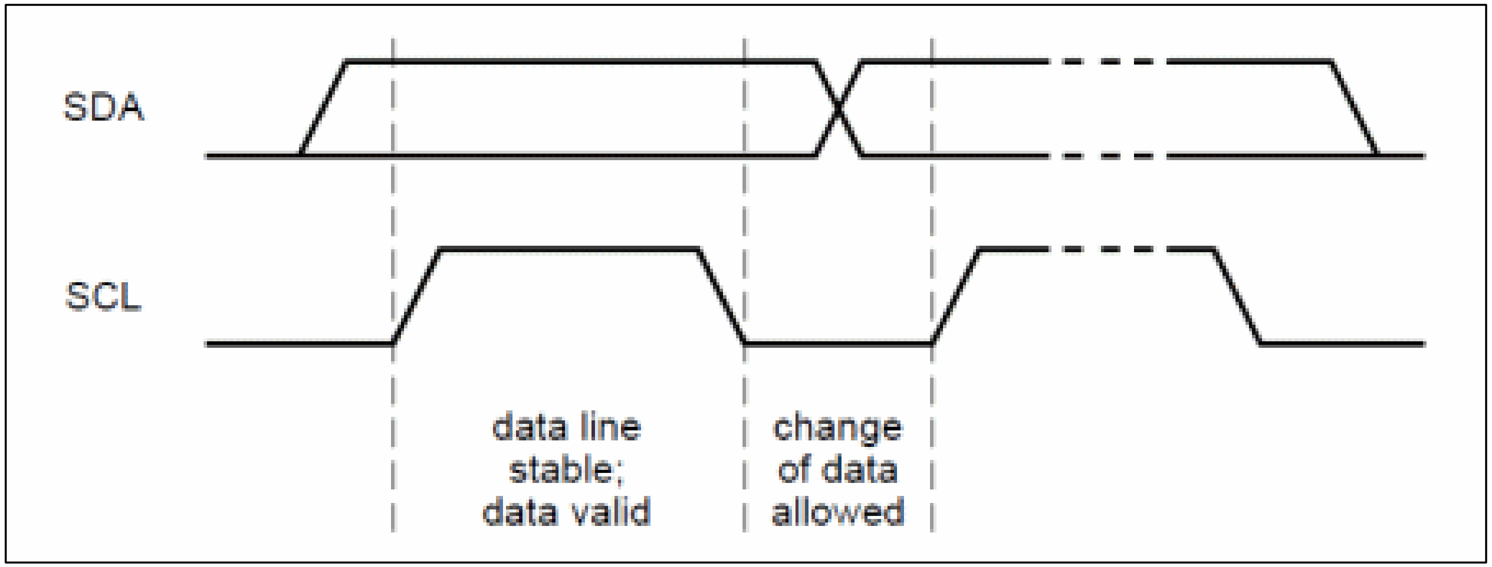


I^{2}C Data Message

- **The Address frame**
  - A 7 to 10-bit unique sequence to identify a slave.
- **Read/Write bit**
  - A bit indicating if the master is in write or read operation.
  - **Write:** sends data to slave (**low** voltage level)
  - **Read:** receives data from slave (**high** voltage level)
- **ACK/NACK bit**
  - A bit sent by the receiver to the sender indicating if a frame was successfully received (**low** voltage level).

Serial Data Transfer

- For **each clock pulse**, **one bit of data** is transferred. The SDA signal can only change when the SCL signal is low – **when the clock is high the data should be stable**.



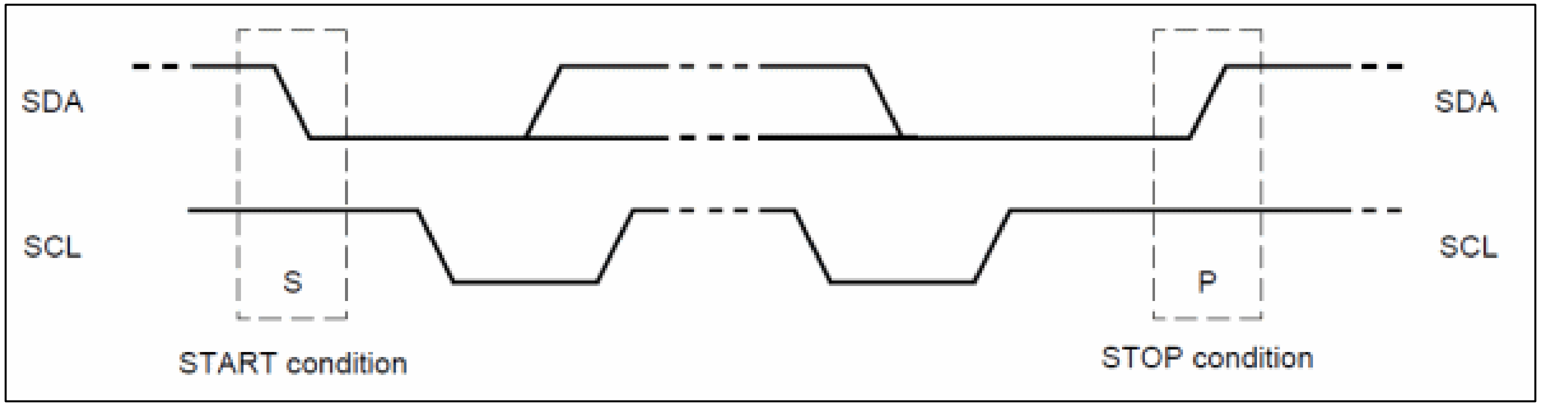
Start and Stop Conditions

- **The Start Condition**
  - SDA line switches from a **high voltage level to a low voltage level** *before* the SCL line switches from **high to low (falling edge)**.

- **The Stop Condition**
  - SDA line switches from a **low voltage level to a high voltage** *after* the SCL line switch from **low to high (rising edge)**, with the SCL line **remaining high**.

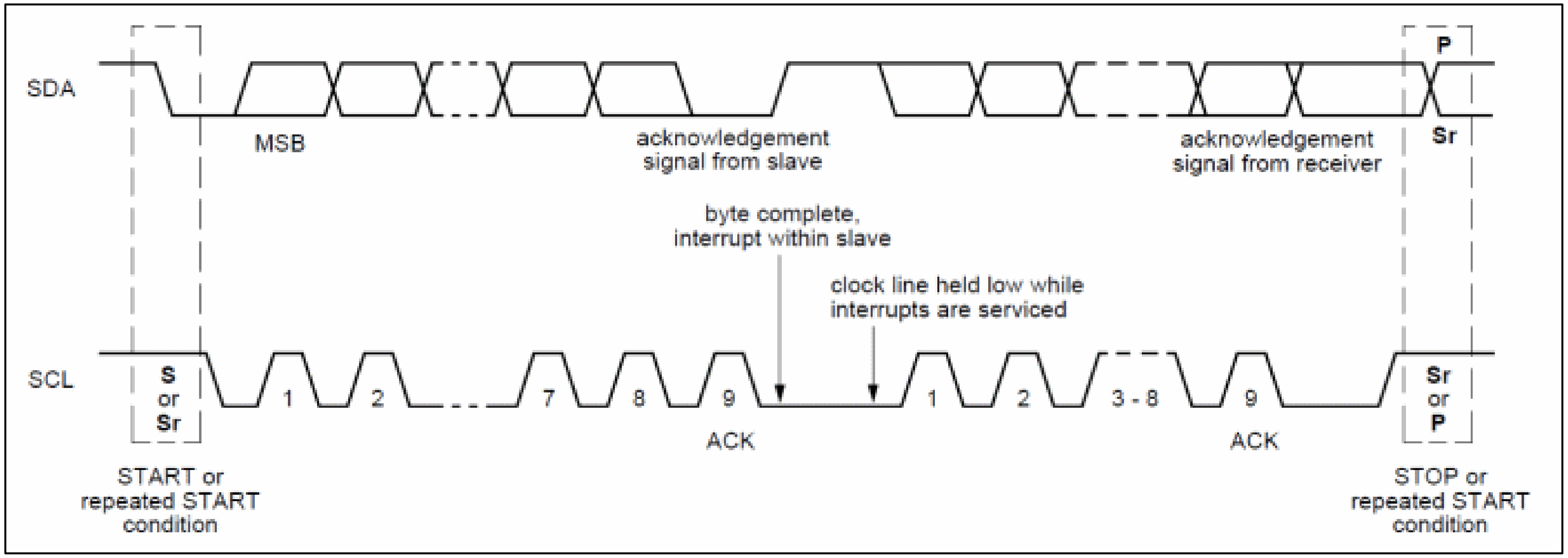
Start and Stop Conditions

- Each I^{2}C command initiated by master device starts with a **START condition** and ends with a **STOP condition**. For both conditions, **SCL has to be high**. A **high to low** transition of SDA is considered as **START** and a **low to high** transition as **STOP**.



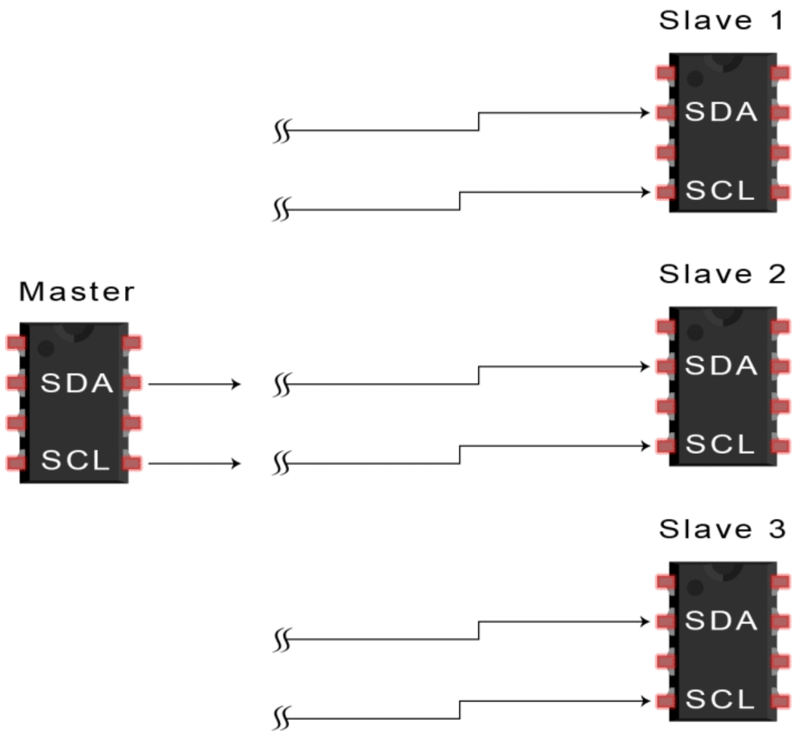
I^{2}C Data Transfer

- Data is transferred in **8-bit frames** on the I^{2}C bus, **starting with the most significant bit (MSB)**. There is no limitation on the number of bytes, however, **each byte must be followed by an Acknowledge (ACK) bit**. This bit signals whether the device is ready to proceed with the next byte.


SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
byte complete, interrupt within slave
clock line held low while interrupts are serviced
S or Sr
1
2
7
8
9
ACK
1
2
3 - 8
9
ACK
P
Sr
Sr or P
START or repeated START condition
STOP or repeated START condition

Steps of Data Transmission

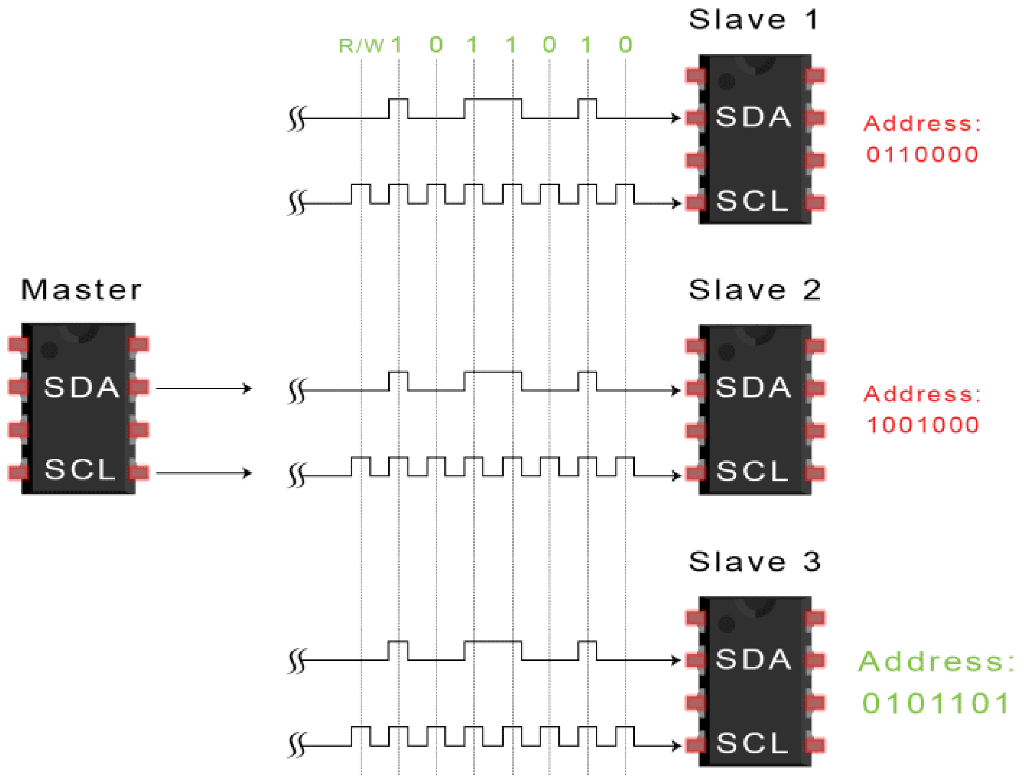
1.



The **master sends the start condition** by setting SDA line from a **high to low** *before* switching the SCL from **high to low.**

Steps of Data Transmission

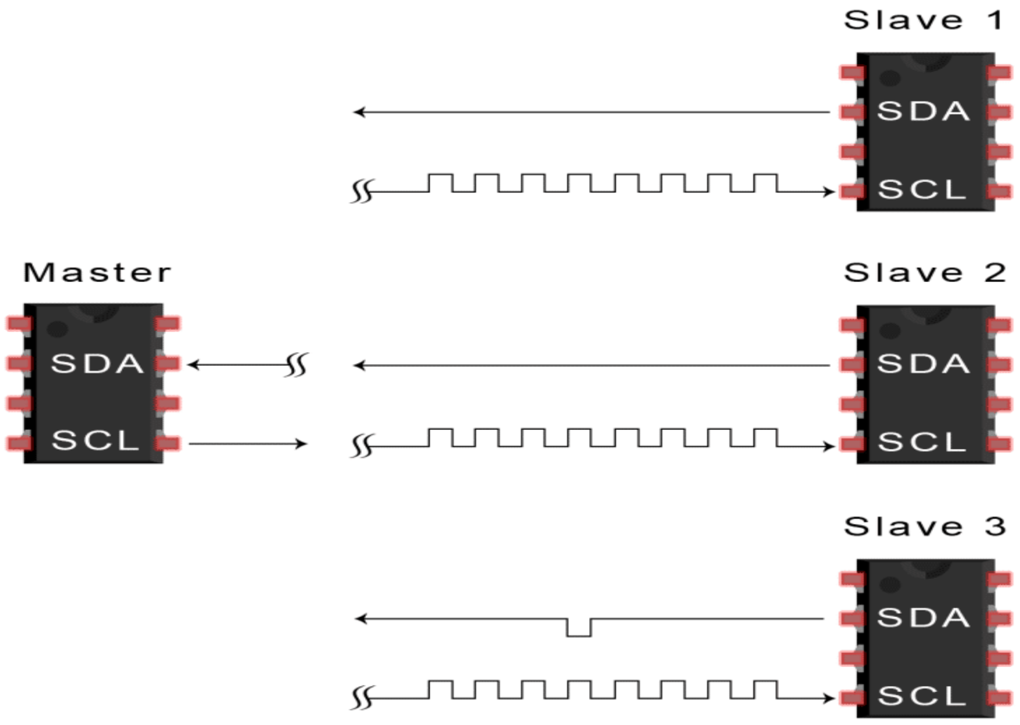
2.



The **master** then **sends the 7 or 10 bit address of the slave** it wants to communicate with, followed by an **R/W bit**.

Steps of Data Transmission

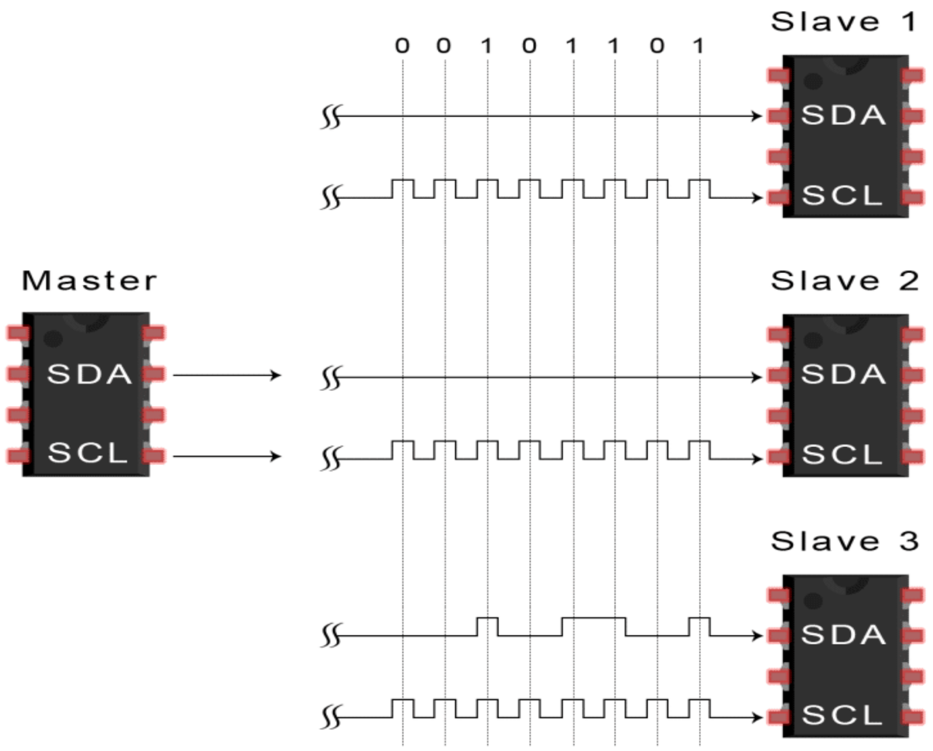
3.



Each **slave compares the sent address** to its own address. If the address matches, that **slave returns an ACK bit** by pulling the SDA line **low** for one bit.

Steps of Data Transmission

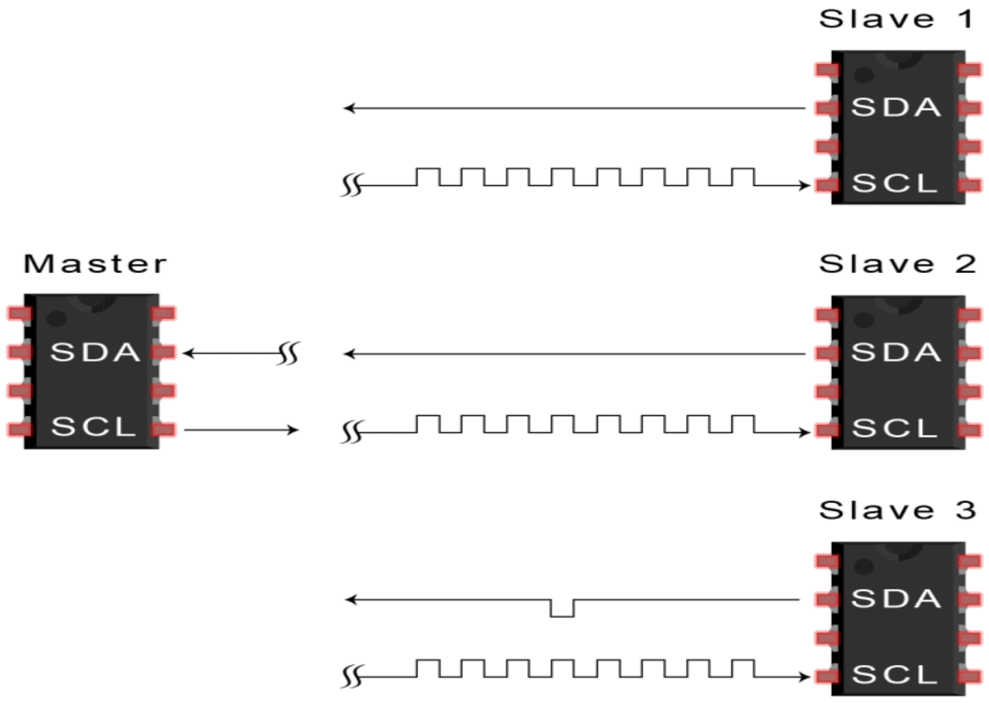
4.



The **master** will **send/receive the data after the ACK bit**.

Steps of Data Transmission

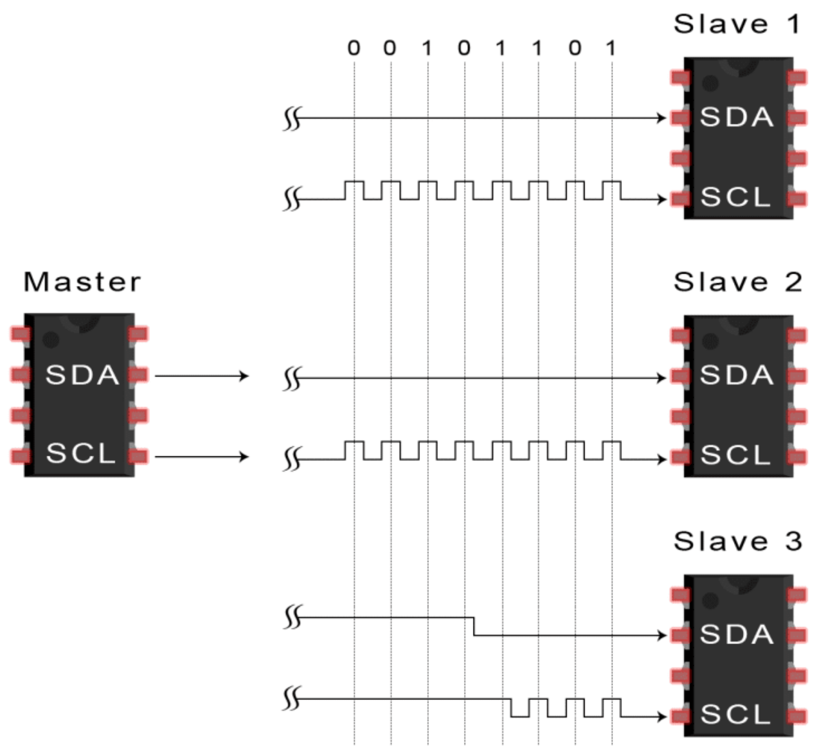
5.



After **each data frame** has been transferred, the **receiving device returns another ACK bit to the sender** to acknowledge successful receipt of the frame.

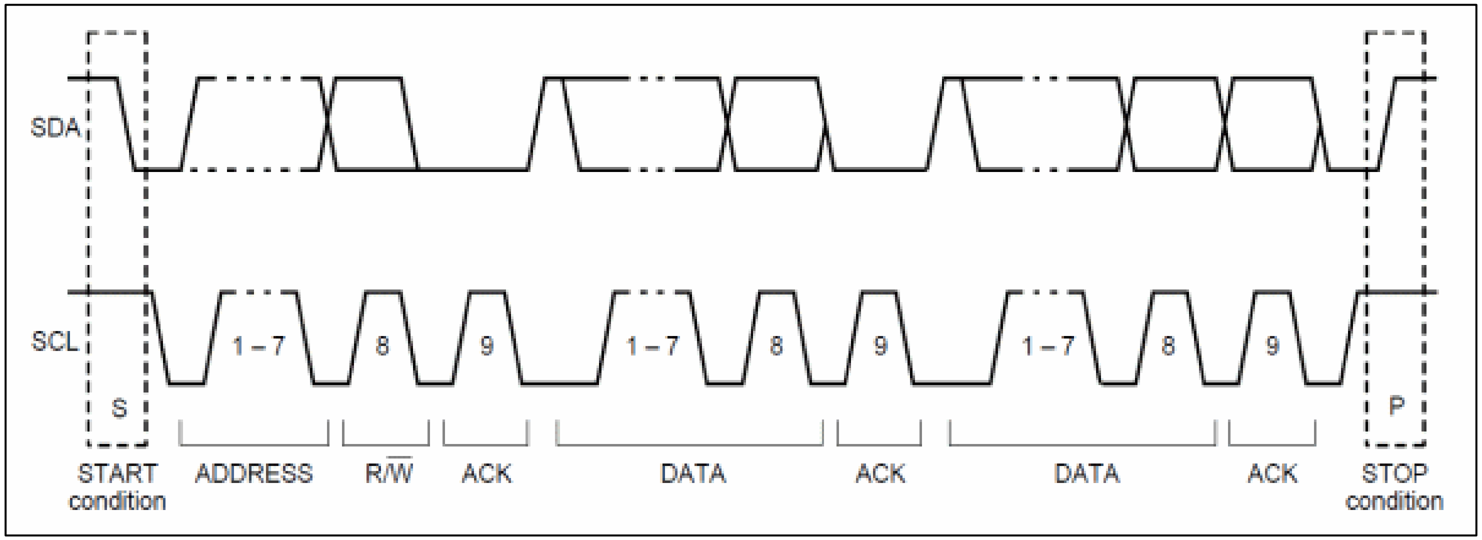
Steps of Data Transmission

6.



To stop data transmission, the **master sends a stop condition to the slave** by switching **SCL high** *before* switching **SDA high**.

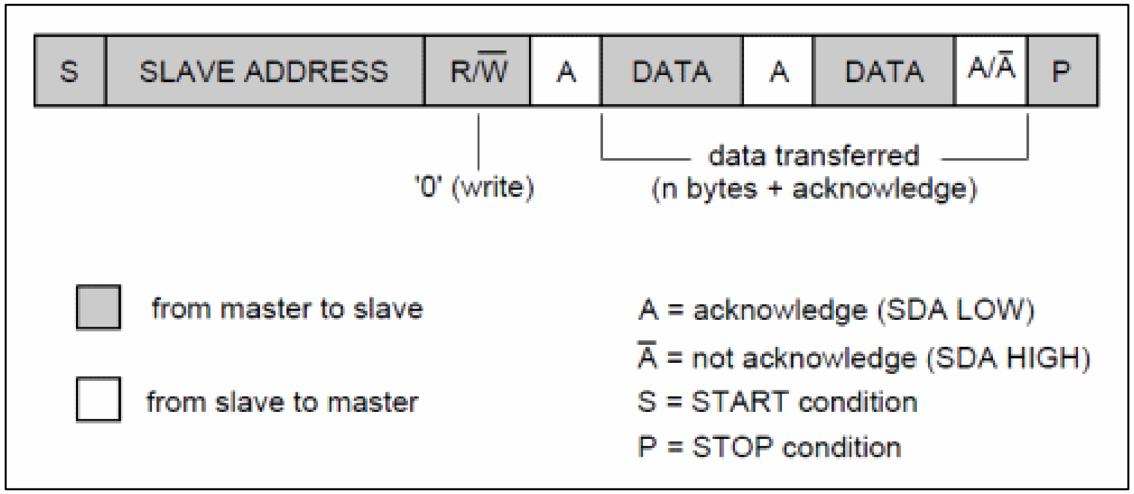
Communication with 7-bit Address



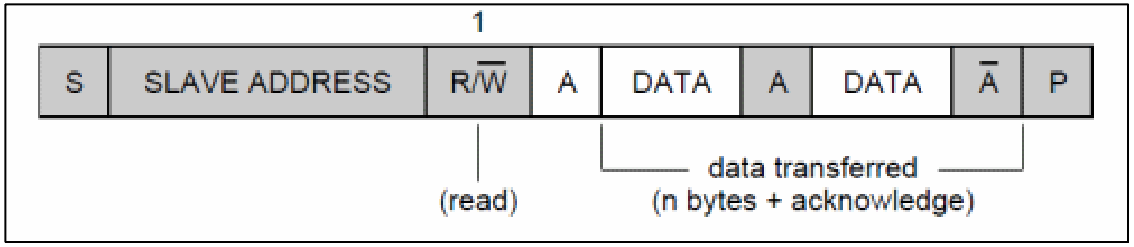
- Each slave device on the bus should have a unique **7-bit address** (up to 2^7 = 128 unique addresses).
- Communication starts with the **Start condition**, followed by the **7-bit slave address** and the **R/W bit**.
- If this bit is **0** then the **master will write to the slave** device. Otherwise (RW = **1**), the **master will read from slave** device.

Communication with 7-bit Address

If the **master only writes to the slave**, then the data transfer direction is not changed.

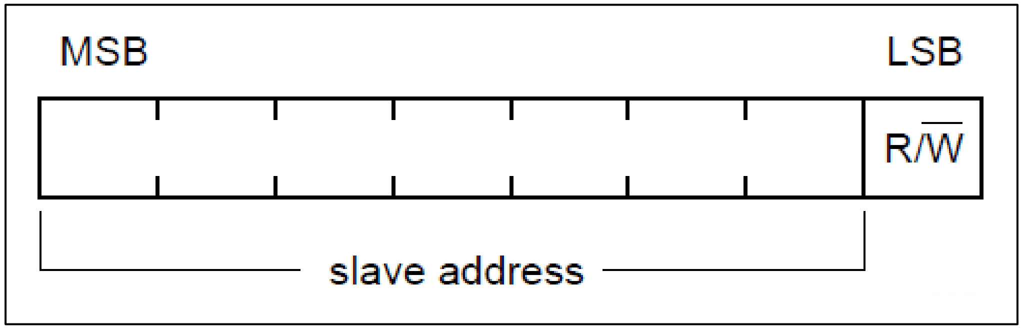


If the **master needs to read from the slave**, then it simply sends the I2C address with the **R/W bit set to read**.



7-bit Addressing

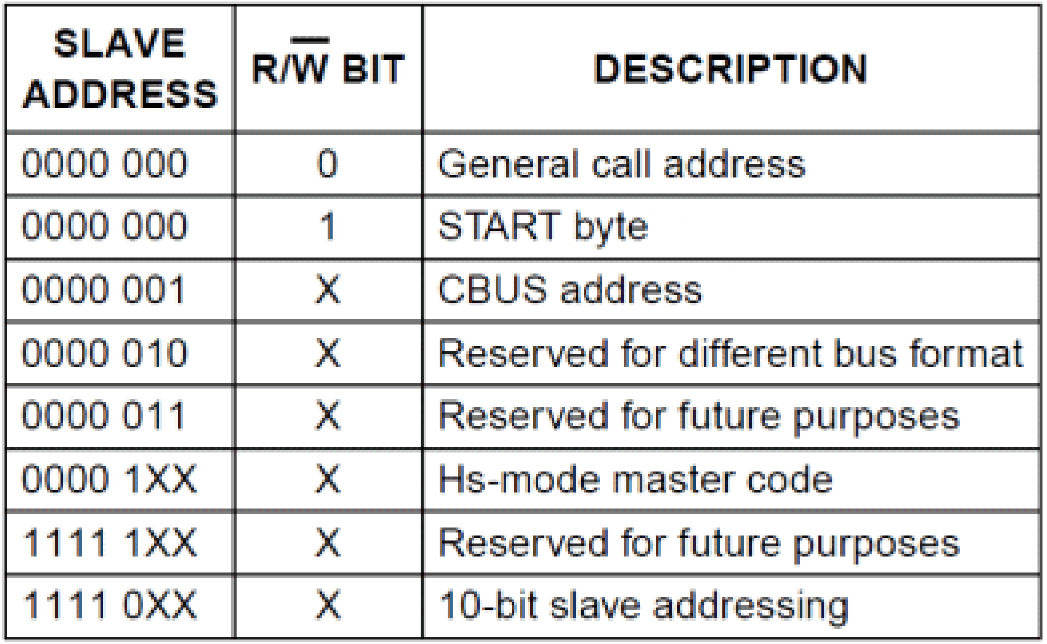
- A slave address **may contain a fixed** and **a programmable part**.
- Some slave devices have few bits of the I^{2}C address dependent on the level of address pins.
- This way, it is possible to have on the same I^{2}C bus more than one I^{2}C device with the same fixed part of I^{2}C address.



A "write" operation to a slave device with address 0x40 => **1000 0000**

A "read" operation from a slave device with address 0x40 => **1000 0001**

7-bit Addressing



SLAVE ADDRESS	R/$\overline{W}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	Reserved for different bus format
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

MSSP Module of PIC16F877A

- The **Master Synchronous Serial Port (MSSP)** module is a **serial interface**, useful for communicating with other peripheral or microcontroller devices such as serial EEPROMs, shift registers, display drivers, A/D converters, etc.
- The MSSP module can operate in **one of two modes**:
  - Serial Peripheral Interface (SPI)
  - Inter-Integrated Circuit (I^2C)

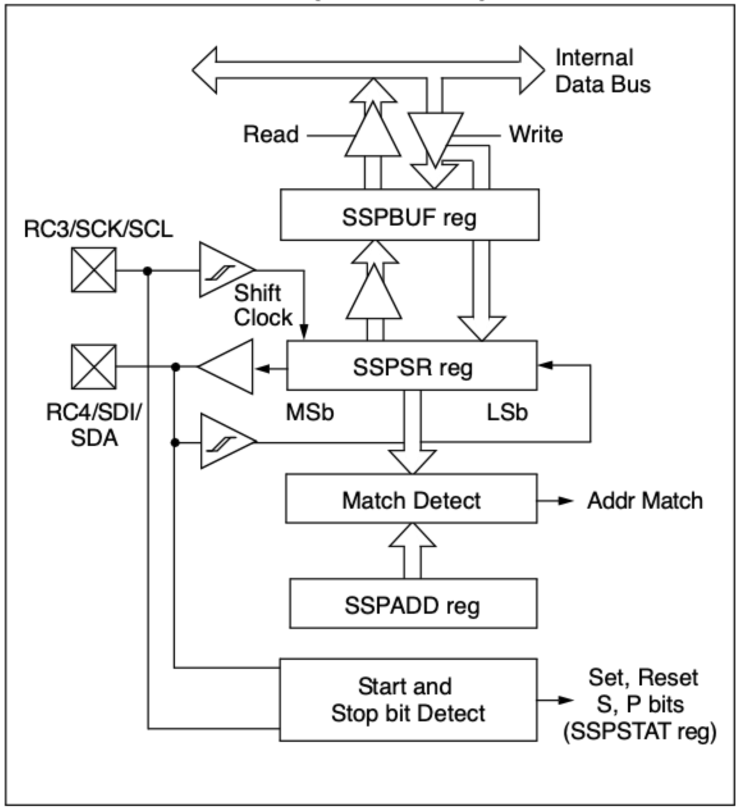
I^{2}C Mode

- The MSSP module in I^{2}C 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**, as well as 7-bit and 10-bit addressing.

I^{2}C Mode

- **Two pins** are used for data transfer:
  - **Serial clock (SCL)**– **RC3**/SCK/SCL
  - **Serial data (SDA)**– **RC4**/SDI/SDA
- The <u>user must configure these pins as **inputs** or **outputs**</u> through the **TRISC<4:3>** bits.

FIGURE 9-7: MSSP BLOCK DIAGRAM (I^{2}C MODE)



I^2C Registers

- MSSP Control Register (SSPCON)*
- MSSP Control Register 2 (SSPCON2)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer Register (SSPBUF)
- MSSP Shift Register (SSPSR)**
- MSSP Address Register (SSPADD)

* in the PIC16F87X data sheet, it is referred as SSPCON1 (Register 9-4, page 82)

** not directly accessible

REGISTER 9-4: SSPCON1: MSSP CONTROL REGISTER 1 (I^{2}C MODE) (ADDRESS 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:
1 = A write to the SSPBUF register was attempted while the I^{2}C conditions were not valid for a transmission to be started. (Must be cleared in software.)
0 = No collision

In Slave Transmit mode:
1 = The SSPBUF register is written while it is still transmitting the previous word. (Must be cleared in software.)
0 = No collision

In Receive mode (Master or Slave modes):
This is a "don't care" bit.

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:
1 = A byte is received while the SSPBUF register is still holding the previous byte. (Must be cleared in software.)
0 = No overflow

In Transmit mode:
This is a "don't care" bit in Transmit mode.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit

1 = Enables the serial port and configures the SDA and SCL pins as the serial port pins
0 = Disables the serial port and configures these pins as I/O port pins

Note: When enabled, the SDA and SCL pins must be properly configured as input or output.

bit 4 **CKP:** SCK Release Control bit

In Slave mode:
1 = Release clock
0 = Holds clock low (clock stretch). (Used to ensure data setup time.)

In Master mode:
Unused in this mode.

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits

1111 = I^{2}C Slave mode, 10-bit address with Start and Stop bit interrupts enabled
1110 = I^{2}C Slave mode, 7-bit address with Start and Stop bit interrupts enabled
1011 = I^{2}C Firmware Controlled Master mode (Slave Idle)
1000 = I^{2}C Master mode, clock = FOSC/(4 * (SSPADD + 1))
0111 = I^{2}C Slave mode, 10-bit address
0110 = I^{2}C Slave mode, 7-bit address

Note: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

REGISTER 9-5: SSPCON2: MSSP CONTROL REGISTER 2 (I^{2}C MODE) (ADDRESS 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7 **GCEN:** General Call Enable bit (Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)
1 = Not Acknowledge
0 = Acknowledge

Note: Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (Master Receive mode only)
1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence Idle

bit 3 **RCEN:** Receive Enable bit (Master mode only)
1 = Enables Receive mode for I^{2}C
0 = Receive Idle

bit 2 **PEN:** Stop Condition Enable bit (Master mode only)
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Stop condition Idle

bit 1 **RSEN:** Repeated Start Condition Enabled bit (Master mode only)
1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition Idle

bit 0 **SEN:** Start Condition Enabled/Stretch Enabled bit
In Master mode:
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Start condition Idle
In Slave mode:
1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is enabled for slave transmit only (PIC16F87X compatibility)

REGISTER 9-3: SSPSTAT: MSSP STATUS REGISTER (I^2C MODE) (ADDRESS 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	$D/\overline{A}$	P	S	$R/\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Slew Rate Control bit

In Master or Slave mode:
1 = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for high-speed mode (400 kHz)

bit 6 **CKE:** SMBus Select bit

In Master or Slave mode:
1 = Enable SMBus specific inputs
0 = Disable SMBus specific inputs

bit 5 **$D/\overline{A}$:** Data/$\overline{\text{Address}}$ bit

In Master mode:
Reserved.

In Slave mode:
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit

1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last

Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 3 **S:** Start bit

1 = Indicates that a Start bit has been detected last
0 = Start bit was not detected last

Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 2 **$R/\overline{W}$:** Read/$\overline{\text{Write}}$ bit information (I^2C mode only)

In Slave mode:
1 = Read
0 = Write

Note: This bit holds the $R/\overline{W}$ bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not $\overline{\text{ACK}}$ bit.

In Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress

Note: ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Idle mode.

bit 1 **UA:** Update Address (10-bit Slave mode only)

1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit

In Transmit mode:
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

In Receive mode:
1 = Data Transmit in progress (does not include the $\overline{\text{ACK}}$ and Stop bits), SSPBUF is full
0 = Data Transmit complete (does not include the $\overline{\text{ACK}}$ and Stop bits), SSPBUF is empty

Initialization (Master Mode)

1. Set **RC3** (**SCL**) and **RC4** (**SDA**) pins to **input**.
2. **Enable synchronous serial port** via the **SSPEN** bit in SSPCON1.
3. Configure MCU to operate in **Master Mode** (**SSPM3:SSPM0**) in SSPCON1.
4. **Set start** and **stop conditions** to **idle** (**SEN**, **PEN**) in SSPCON2.
5. **Set receive** and **acknowledge enable** to **idle** (**RCEN**, **ACKEN**) in SSPCON2.
6. **Configure clock speed** by **setting SSPADD lower 7-bit** to the **baud rate generator reload value**.

$$SSPADD = \frac{F_{OSC}}{4 * clock} - 1$$

For example, a clock speed of 100 KHz with an F_{OSC} = 4 MHz:

$$SSPADD = \frac{\text{4MHz}}{4 * \text{100KHz}} - 1 = 9 \text{ or } 0\text{x}09$$

```
/* I2C initialization sequence*/
void init_I2C_Master(void)
{
   TRISC3 = 1;       // set RC3 (SCL) to input
   TRISC4 = 1;       // set RC4 (SDA) to input
   SSPCON = 0x28;    // SSP enabled, I2C master mode
   SSPCON2 = 0x00;   // start condition idle, stop condition idle
                     // receive idle
   SSPSTAT = 0x00;   // slew rate enabled
   SSPADD = 0x09;    // clock frequency at 100 KHz (FOSC = 4MHz)
}
```

I^{2}C Wait

- Before performing a data transmission, **make sure no operation in I^{2}C bus** (*all operations are complete*).
- To do this, check the **R/$\overline{\text{W}}$** bit in **SSPSTAT** if transmission is in progress or that **ACKEN**, **RCEN**, **PEN**, **RSEN,** and **SEN** in **SSPCON2** are set to **0** (*to be absolutely sure that no I^{2}C operation is going on*).
- Once either conditions is satisfied, an operation (*send/ receive*) can now be started.

```
void I2C_Wait(void)
{
   /* wait until all I2C operations are finished*/
   while((SSPCON2 & 0x1F) || (SSPSTAT & 0x04));
}
```

Start and Stop Conditions

- Each time an operation is performed (*read or write*), a **start condition** must be issued via the **SEN** bit in SSPCON2.
- After an operation, a **stop condition** must be issued via the **PEN** bit in SSPCON2.
- *Before* setting **SEN** or **PEN** bits, make sure that **no operation is currently on going**.
- In between the start and stop condition, the **slave address** and **8-bit data frame** will be sent (***master mode transmit***).
- The **master** can issue **another address** + **R/W** byte without issuing a stop condition by enabling **repeated start** via the **RSEN** bit in SSPCON2.

```
void I2C_Start(void)
{
   /* wait until all I2C operations are finished*/
   I2C_Wait();
   /* enable start condition */
   SEN = 1;     // SSPCON2
}

void I2C_RepeatedStart(void)
{
   /* wait until all I2C operations are finished*/
   I2C_Wait();
   /* enable repeated start condition */
   RSEN = 1;     // SSPCON2
}

void I2C_Stop(void)
{
   /* wait until all I2C operations are finished*/
   I2C_Wait();
   /* enable stop condition */
   PEN = 1;     // SSPCON2
}
```

I^{2}C Transmit (Master Mode)

- The **"7-bit slave address + R/W"** or **data frame** to be transmitted to the I^{2}C bus shall be written to the **SSPBUF** register.
- The contents of **SSPBUF** is automatically written to the **SSPSR** (*not directly accessible*), which is responsible for shifting out the bits serially for transmission.
- *Before* writing to **SSPBUF**, make sure that no operation is ongoing.

```
void I2C_Send(unsigned char data)
{
    /* wait until all I2C operations are finished*/
    I2C_Wait();
    /* write data to buffer and transmit */
    SSPBUF = data;
}
```

R/W bit in **SPSSTAT** is automatically set to '0' when writing to SSPBUF. Writing to **SSPBUF** starts the transmission which is done automatically by the MCU.

I^{2}C Receive (Master Mode)

- The **data received from the slave device** via I^{2}C bus is received bit by bit in the **SSPSR** (*not directly accessible*).
- When the SSPSR receives a complete byte, the 8-bit data frame is stored in the **SSPBUF** register.
- To verify if receive is complete, **BF** status bit in SSPSTAT register or **SSPIF** interrupt flag in PIR1 register can be checked.
- During I^{2}C communication, the master should send an **acknowledge** bit *after* receiving an 8-bit data frame from the slave. The **master acknowledge bit** should be set via the **ACKDT** bit in SSPCON2 register.
- When receive is complete, the master has to **enable the "acknowledge sequence"** via the **ACKEN** bit in SSPCON2.

```
unsigned char I2C_Receive(unsigned char ack)
{
   unsigned char temp;
   I2C_Wait();    // wait until all I2C operations are finished
   RCEN = 1;      // enable receive (SSPCON2 reg)
   I2C_Wait();    // wait until all I2C operations are finished
   temp = SSPBUF; // read SSP buffer
   I2C_Wait();    // wait until all I2C operations are finished
   ACKDT = (ack)?0:1;  // set ACK or NACK
   ACKEN = 1;     // enable acknowledge sequence
   return temp;
}
```

ACKDT is the "ACK" or "NACK" bit that will be sent when the **acknowledge sequence** is started. When the master issues an "ACK", the slave will send the next byte. However, if the master issues an "NACK", then the slave will not send the next byte and releases the clock.

I^{2}C Send/Receive Example (Master Mode)

- **Process to send I^{2}C message* to slave:**
  - Initialize I^{2}C as **master** mode.
  - Initiate **start** condition.
  - Send the **7-bit address** followed by the **8-bit data frame(s)****.
  - Initiate **stop** condition.
- **Process to read data from slave:**
  - Initialize I^{2}C as **master** mode. *(not needed if I^{2}C is already initialized)*
  - Initiate **start** condition.
  - Send the **7-bit address of the slave**.
  - Read data frame(s) and send **ACK/NACK**.
  - Initiate **stop** condition.

* message includes the start condition, address, data frames & stop condition (R/W bit is set automatically)

I^{2}C Send/Receive Example (Master Mode)

- In this example, the **master will send a data frame from PORTD** to **slave with address 0x10**.
- Then **read data** from the slave with address 0x10. Data from the slave will be written to **PORTB**.
- PORTB -> LEDS, PORTD <- DIP switches (8-bit)

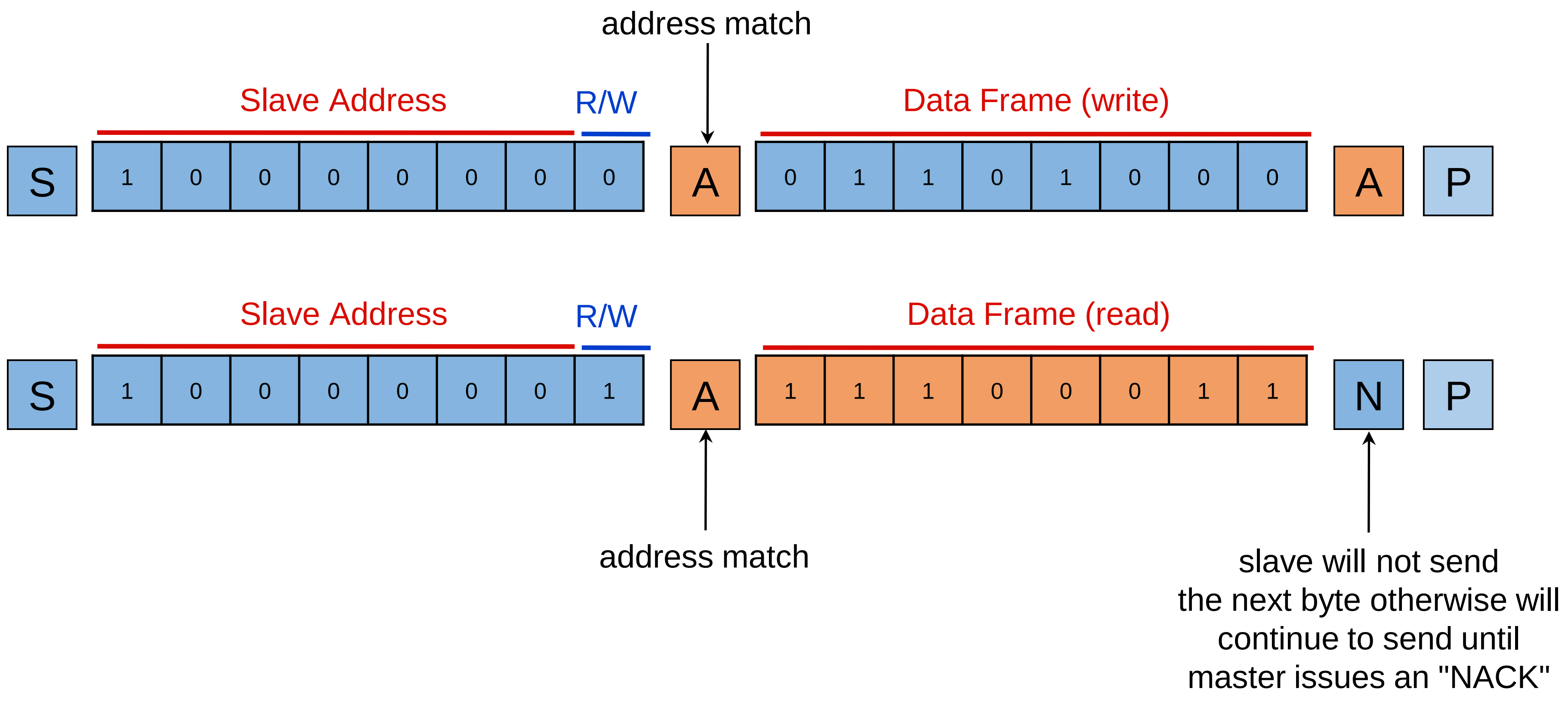
```
void main(void)
{
   TRISB = 0x00;          // set all bits in PORTB to output
   PORTB = 0x00;          // set all LEDs to off
   TRISD = 0xFF;          // set all bits in PORTD to input
   init_I2C_Master();     // initialize I2C as master
   for(;;)
   {
      I2C_Start();        // initiate start condition
      I2C_Send(0x10);     // send the slave address + write
        // for error handling, check if slave sent ACK bit via ACKSTAT bit
        // in SSPCON2 (for an address match)

      I2C_Send(PORTD);    // send 8-bit data frame
      I2C_Stop();         // initiate stop condition
      __delay_ms(200);    // delay before next operation

      I2C_Start();        // initiate start condition
      I2C_Send(0x11);     // send the slave address + read
        // for error handling, check if slave sent ACK bit via ACKSTAT bit
        // in SSPCON2 (for an address match)

      PORTB = I2C_Receive(0);    // read data and not acknowledge (NACK)
                                 // end of read operation
                                 // write received data to PORTB
      I2C_Stop();         // initiate stop condition
      __delay_ms(200);    // delay before next operation

   }
}
```



🟦	Master to slave
🟧	Slave to master

S - start Condition

P - stop Condition

A - acknowledge (ACK)

N - not acknowledge (NACK)

Initialization (Slave Mode)

1. Set **RC3** (**SCL**) and RC4 (**SDA**) pins to **input**.
2. **Disable slew rate control** via **SMP** bit (SSPSTAT).
3. **Enable synchronous serial port** via **SSPEN** (SSPCON).
4. Set **CKP** bit to **“release clock”** (SSPCON1).
5. Configure MCU to operate in **Slave Mode, 7-bit address** via **SSPM3:SSPM0** (SSPCON1).
6. **Enable clock stretching** for **slave transmit & receive** via **SEN** bit (SSPCON2).
7. Set a **valid* 7-bit slave address** via the **SSPADD** register.
8. **Enable synchronous serial port interrupt** via **SSPIE** bit (PIE1).
9. **Clear interrupt flag** via **SSPIF** bit (PIR1).
10. **Enable peripheral interrupts** via **PEIE** and **GIE** bits (INTCON).

* for valid slave addresses, see here

```
/* I2C initialization sequence*/
void init_I2C_Slave(unsigned char slave_add)
{
   TRISC3 = 1;        // set RC3 (SCL) to input
   TRISC4 = 1;        // set RC4 (SDA) to input
   SSPCON = 0x36;     // SSP enabled, CKP release clock (SCK)
                      // I2C slave mode 7-bit address
   SSPCON2 = 0x01;    // start condition idle, stop condition idle
                      // receive idle
   SSPSTAT = 0x80;    // slew rate control disabled
   SSPADD = slave_add;  // 7-bit slave address
   SSPIE = 1;         // enable SSP interrupt
   SSPIF = 0;         // clear interrupt flag
   PEIE = 1;          // enable peripheral interrupt
   GIE = 1;           // enable unmasked interrupt
}
```

Receive Interrupts (Slave Mode)

1. **Hold the clock low** via the **CKP** bit (SSPCON).
2. **Check for overflow** via **SSPOV** bit or **data collision** via **WCOL** bit (SSPCON). If overflow or data collision occurs, no master read/write operation can happen.
3. **Check if master operation is “read"** via the **R/$\overline{\text{W}}$** bit and the **last received data frame matches the address** via the **D/$\overline{\text{A}}$** bit (SSPSTAT).
4. **Check if master operation is “write”** via the **R/$\overline{\text{W}}$** bit and the **last received data frame matches the address** via the **D/$\overline{\text{A}}$** bit (SSPSTAT).
5. **Clear** the SSP interrupt flag (**SSPIF**).

```
void interrupt ISR(void)
{
   unsigned char temp;
   CKP = 0;           // hold clock low (SSPCON reg)
   if(WCOL || SSPOV) // check if overflow or data collision (SSPCON reg)
   {
      temp = SSPBUF;  // read SSPBUF to clear buffer
      WCOL = 0;       // clear data collision flag
      SSPOV = 0;      // clear overflow flag
      CKP = 1;        // release clock (SSPCON reg)
   }
   /* check operation if “write” or "read" */
   if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW)      // write to slave
   {
      temp = SSPBUF;  // read SSPBUF to clear buffer
      while(!BF);     // wait until receive is complete (SSPSTAT reg)
      /* read data from SSPBUF */
      /* data = SSPBUF; */
      CKP = 1;         // release clock (SSPCON reg)
   }
   else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW)  // read from slave
   {
      temp = SSPBUF;  // read SSPBUF to clear buffer
      BF = 0;         // clear buffer status bit (SSPSTAT reg)
      /* send data by writing to SSPBUF */
      /* SSPBUF = data; */
      CKP = 1;        // release clock (SSPCON reg)
      while(BF);      // wait until transmit is complete (SSPSTAT reg)
   }
   SSPIF = 0;         // clear interrupt flag
}
```

I^2C Send/Receive Example (Slave Mode)

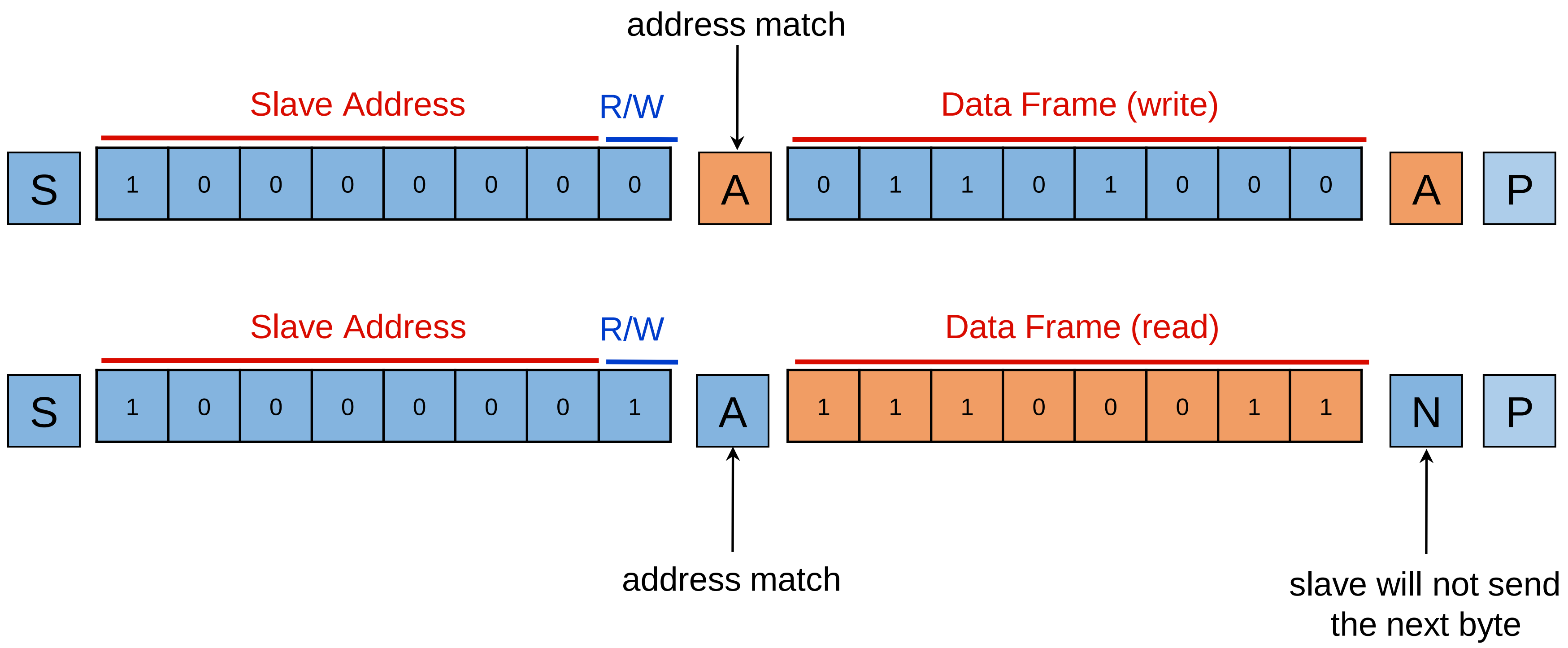
- For this example, the MCU will be configured as a **slave device with an address of 0x10**.
- When the master "writes" data to the slave, data from the master will be written to **PORTB**.
- When the master "reads" data from the slave, data from **PORTD** will be sent to the master.
- **No foreground routine** since all operations are interrupt based.
- PORTB -> LEDS, PORTD <- DIP switches (8-bit)

```
void main(void)
{
    TRISB = 0x00;    // set all bits in PORTB to output
    PORTB = 0x00;    // all LEDs in PORTB are off
    TRISD = 0xFF;    // set all bits in PORTD to input
    init_I2C_Slave(0x10);   // initialize I2C as slave with address 0x10

    for(;;)
    {
    }
}
```

```
void interrupt ISR(void)
{
   unsigned char temp;
   CKP = 0;            // hold clock low (SSPCON reg)
   if(WCOL || SSPOV) // check if overflow or data collision (SSPCON reg)
   {
      temp = SSPBUF;   // read SSPBUF to clear buffer
      WCOL = 0;        // clear data collision flag
      SSPOV = 0;       // clear overflow flag
      CKP = 1;         // release clock (SSPCON reg)
   }
   /* check operation if "write" or "read"*/
   if(!SSPSTATbits.D_nA && !SSPSTATbits.R_nW)     // write to slave
   {
      temp = SSPBUF;   // read SSPBUF to clear buffer
      while(!BF);      // wait until receive is complete (SSPSTAT reg)
      PORTB = SSPBUF; // write data from master to PORTB
      CKP = 1;         // release clock (SSPCON reg)
   }
   else if(!SSPSTATbits.D_nA && SSPSTATbits.R_nW) // read from slave
   {
      temp = SSPBUF;   // read SSPBUF to clear buffer
      BF = 0;          // clear buffer status bit (SSPSTAT reg)
      SSPBUF = PORTD; // send data from PORTD to master
      CKP = 1;         // release clock (SSPCON reg)
      while(BF);       // wait until transmit is complete (SSPSTAT reg)
   }
   SSPIF = 0;         // clear interrupt flag
}
```

Address Match (Write & Read)



🟦	Master to slave
🟧	Slave to master

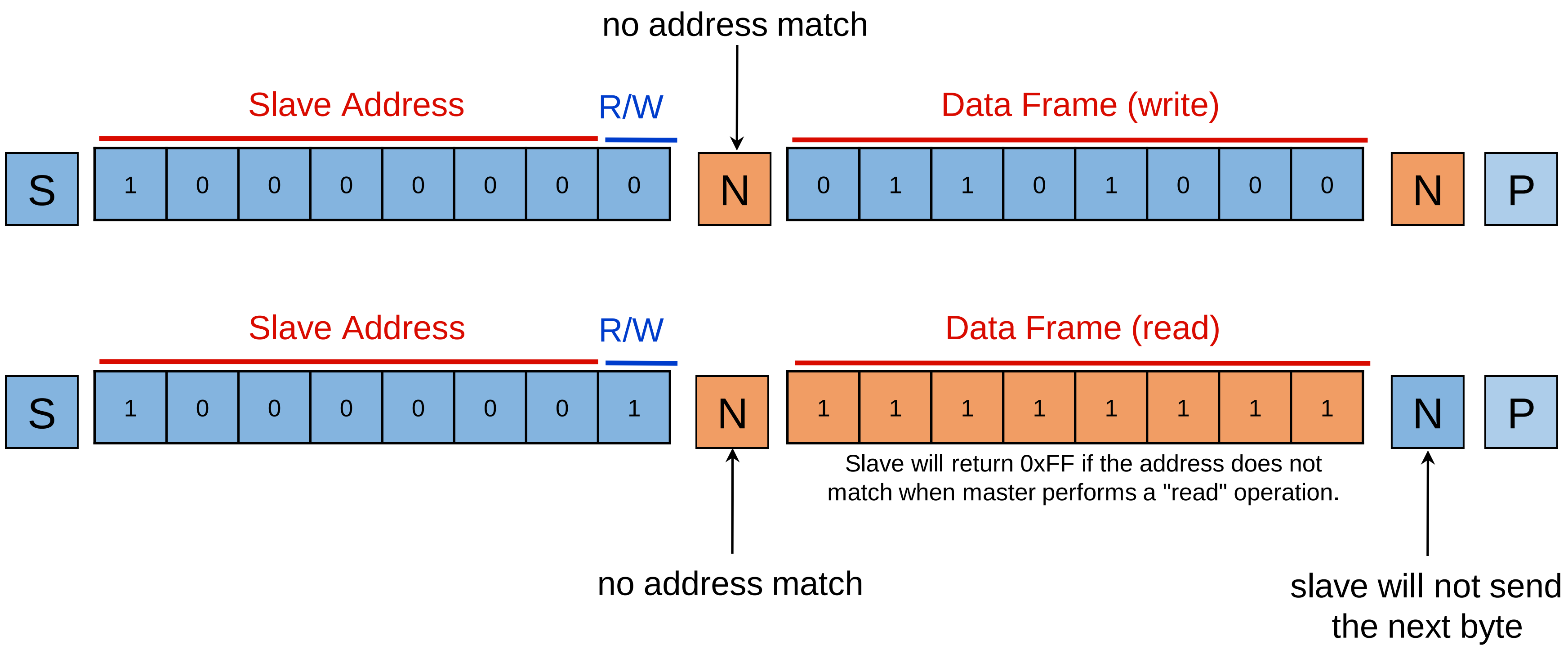
S - start Condition

P - stop Condition

A - acknowledge (ACK)

N - not acknowledge (NACK)

No Address Match (Write & Read)



🟦	Master to slave
🟧	Slave to master

S - start Condition

P - stop Condition

A - acknowledge (ACK)

N - not acknowledge (NACK)

University of San Carlos | Department of
COMPUTER ENGINEERING

CpE 3201
Embedded Systems

End of Lecture

References:

- PIC16F87X Data Sheet, Microchip Technology Inc. 2003.
- https://i2c.info/
- https://electrosome.com/i2c-pic-microcontroller-mplab-xc8/
- https://circuitdigest.com/microcontroller-projects/i2c-communication-with-pic-microcontroller-pic16f877a
- https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html#