Chapter 7
Serial Communication

Parallel I/O vs Serial I/O

- Parallel I/O: 8 or more lines are used to transfer data simultaneously.
- Serial I/O: 1 line is used. Data is sent one bit at a time.

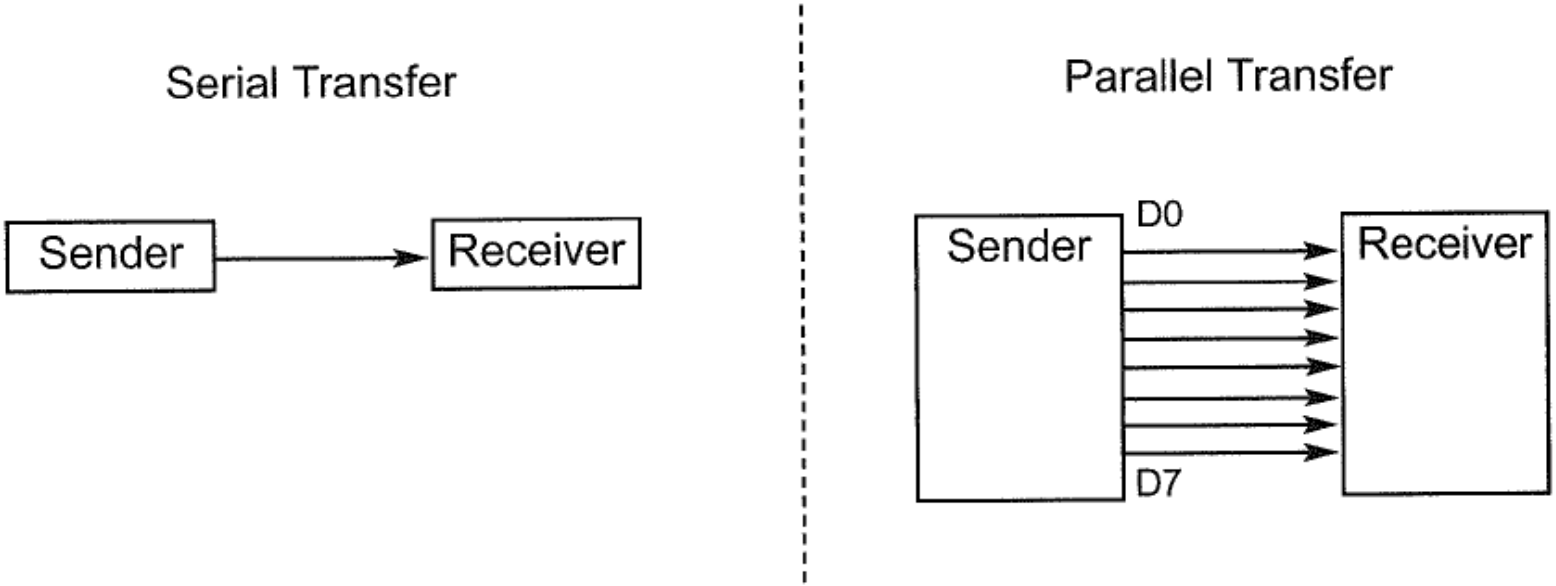


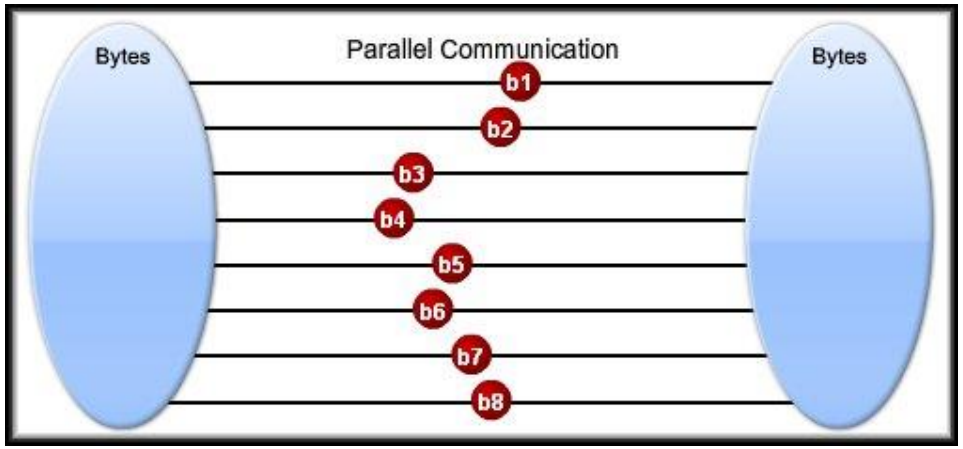
Image courtesy Mazidi, MA et al., “PIC Microcontroller and Embedded Systems Using Assembly and C for PIC18,” Pearson Education, New Jersey, 2008.

Features of Parallel Communication

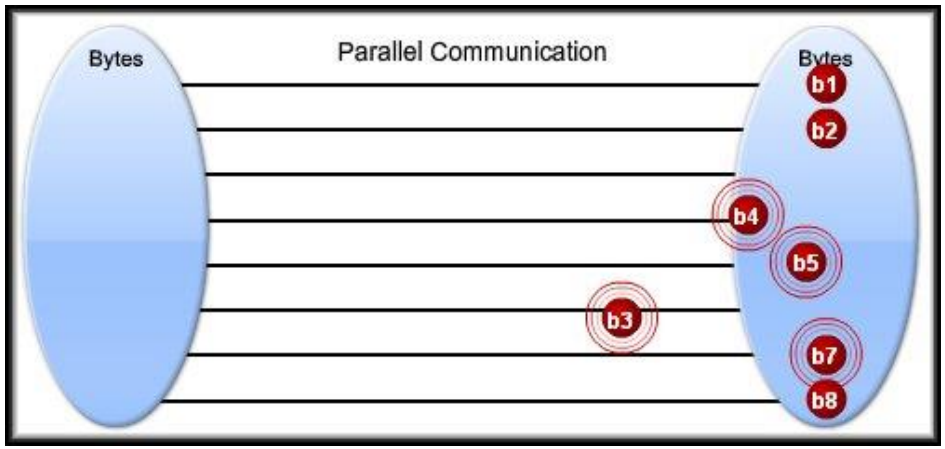
1. Parallel I/O uses many signal pins
   - PORTx – 8 bits
   - TRISx – 8 bits for data transfer direction
   - Microcontroller has limited number of pins.
2. Usually used for very short distances, due to
   - Timing skew: The signals arrive at different times.
   - Crosstalk: Signal transmitted in one channel creates an undesired effect in another channel.
3. Cable for parallel transfer is more expensive

2 drawbacks of Parallel I/O

Timing Skew



Crosstalk



Serial Communication

- The transfer of digital data over a single data transmission line one bit at a time.
- Why?
  - Reduce the cost of IC package by reducing the number of pins used for communication.
  - Timing skew and crosstalk between different channels is not an issue.
  - A serial connect requires fewer interconnecting cables and hence occupies less space.

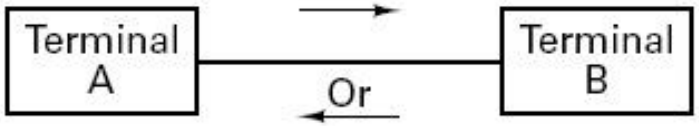
Duplex Mode

Simplex:

Transmission in only one direction

(a)

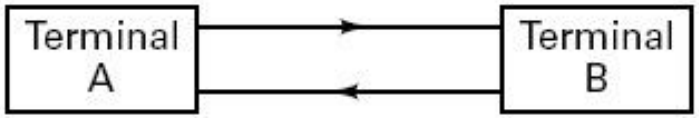
Half duplex:



Transmission in either direction, but not simultaneously

(b)

Full duplex:

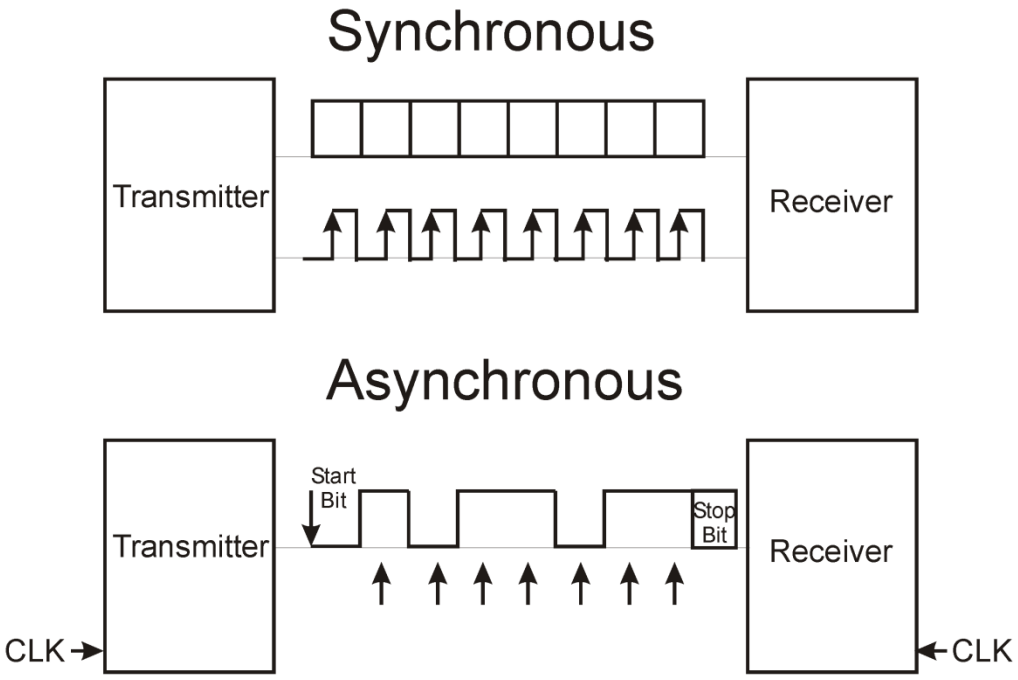


Transmission in both directions simultaneously

(c)

Synchronous vs. Asynchronous

1. *Synchronous*
   - A synchronizing clock signal is transmitted for synchronization.
2. *Asynchronous*
   - No clock signal is transmitted from the transmitter to receiver.
   - On each end, the internal clock must be the same.
   - Rely on extra bits that are wrapped around the byte to be sent (start bit, stop bit).



PIC18 Serial Module

PIC18 family has several built-in modules for serial communication:

- Universal Asynchronous Receiver Transmitter (UART) for *asynchronous* communication (Sec. 7.1).
- Inter-integrated Circuit Interface (I^2C) for *synchronous* communication with other I^2C peripherals (Sec. 7.2).

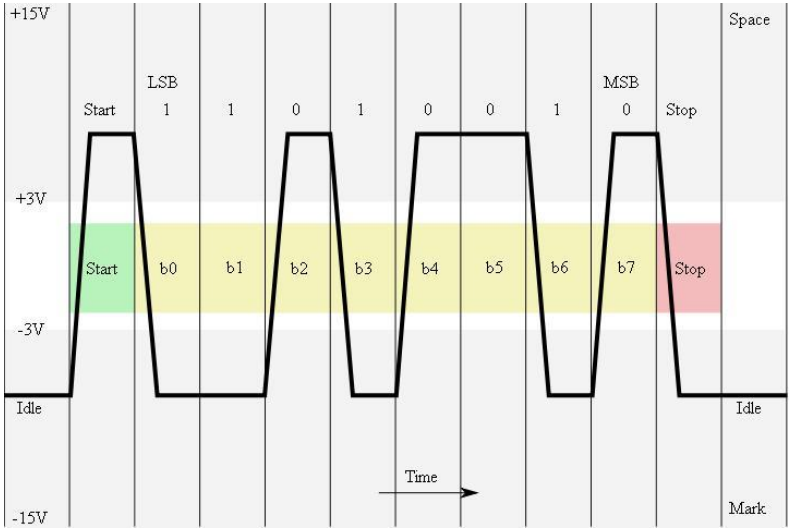
7.1 Universal Asynchronous Receiver Transmitter (UART)

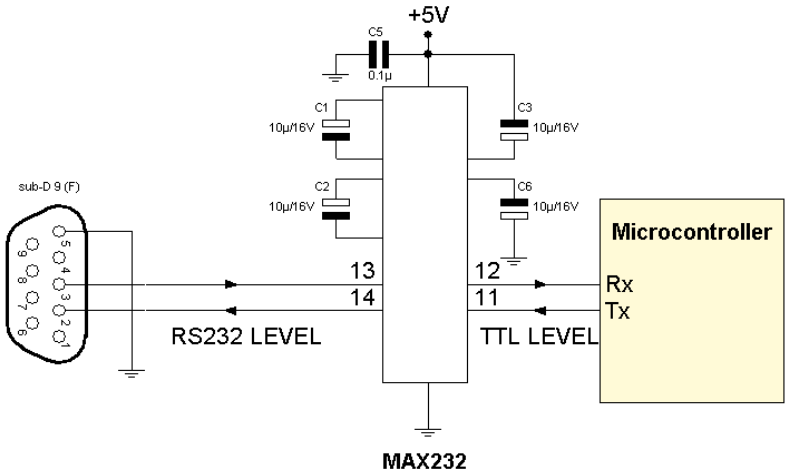
UART

- Both the transmitting and receiving ends need to be running clocks at the *same rate*.
- The data line has a default high level.
- Transmissions start with a start bit with logic level 0.
- Then the 7,8, or 9 bits of data follow
- Then an optional parity bit is added (1 if the total number of 1's in the data are odd).
- Finally, one, or two stop bits follow (stop bits have a value of 1).
- Both sides must agree on the baud rate, the number of bits, the number of stop bits and whether to use a parity bit.

RS232 Voltage Levels

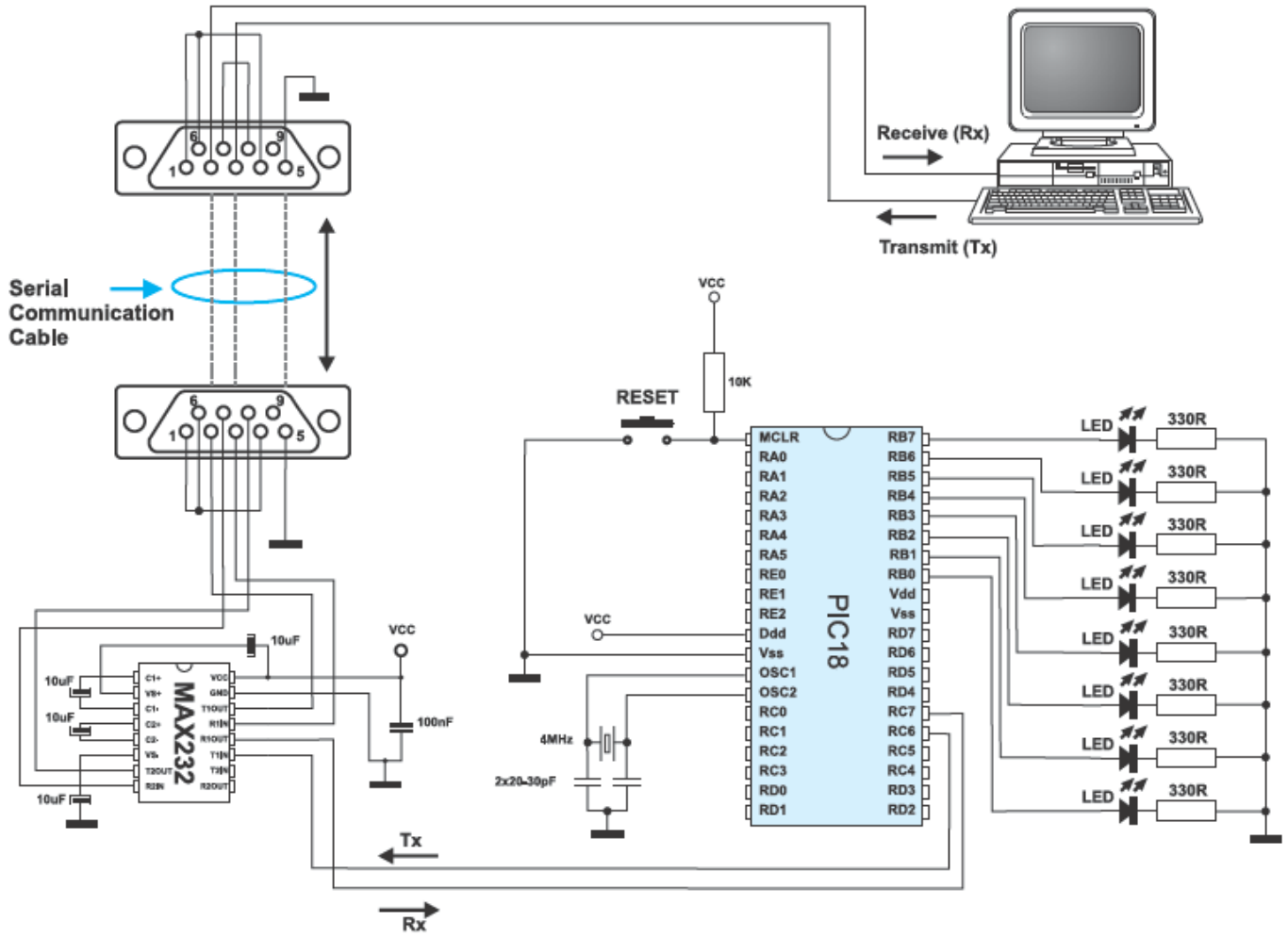
- What voltage level represents logic 1 and 0?
  - 0V and 5V (TTL levels) were not well established at the creation of this standard, plus they would result in DC level being present on the line
  - Logical 1 is represented by a negative voltage between -3V and -25V (-15V being common)
  - Logical 0 is represented by a positive voltage between 3V and 25V (15V being common)
- As microcontrollers usually output 0V for “0” and 3.3-5V for “1”, external voltage level converters are needed. MAX232 is the most common conversion circuit.





PIC18 UART-PC Communication

This example illustrates the use of the PIC18's EUSART module.



UART Asynchronous Mode

- Full-Duplex
- The operation of the UART module is controlled by five registers:
  - Transmit Status and Control (TXSTA)
  - Receive Status and Control (RCSTA)
  - Baud Rate Control (BAUDCON)
  - SPBRGH:SPBRG to specify the Baud Rate

Baud Rate

- Baud Rate: the rate at which serial symbol is transferred over a channel.
- Common Baud rates include: 1200 Baud, 2400 Baud, 4800 Baud, 9600 Baud, 19200 Baud, 38400 Baud & 115200 Baud.
- For binary TX/RX, baud rate is equivalent to the bit rate.

Baud Rate Generator (BRG)

- BRG determines the baud rate of the TX/RX
- BRG can operate in 8-bit or 16-bit modes
- The baud rate can be calculated using the following equations:

BRG16	BRGH	Baud Rate
0	0	$f_{osc}/[64(X+1)]$
0	1	$f_{osc}/[16(X+1)]$
1	0	$f_{osc}/[16(X+1)]$
1	1	$f_{osc}/[4(X+1)]$

- X = SPBRG (if BRG16 = 0) (8-bit)
- X = SPBRGH:SPBRG (if BRG16 = 1) (16-bit)

Baud Rate Generator (BRG)

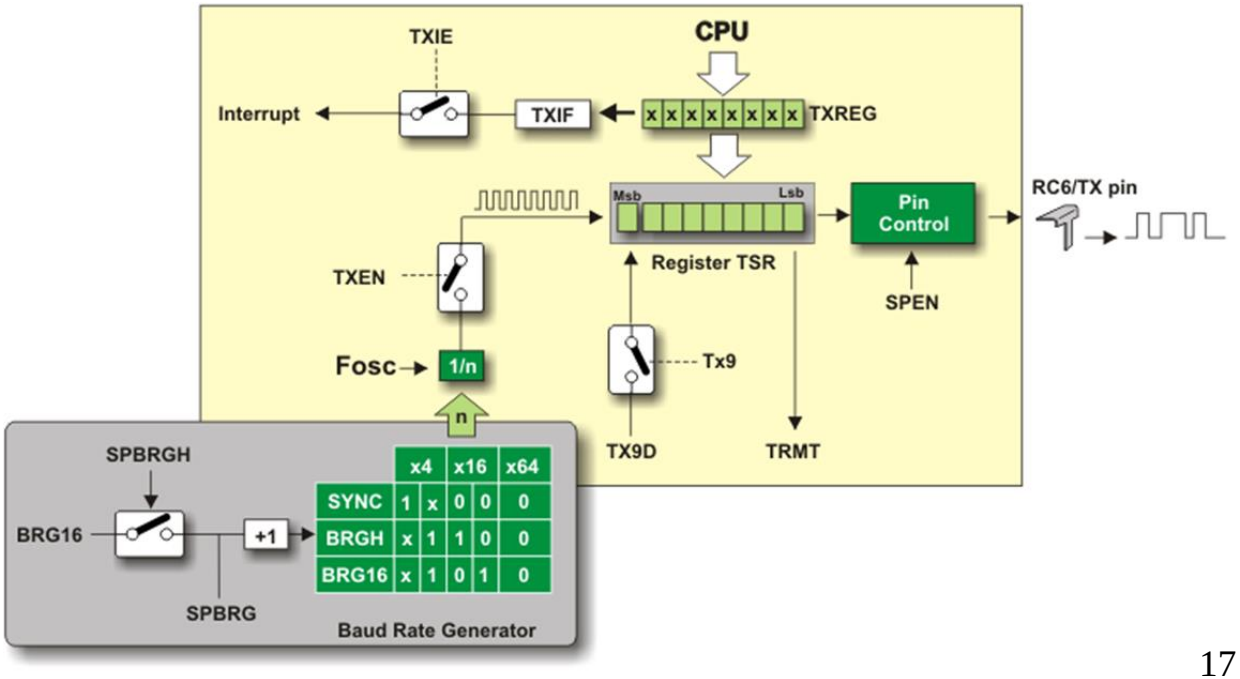
- Suppose BRG16 = 0, BRGH = 1, f_{osc} = 4MHz and want to set baud rate as 9600 bps.

$$\begin{aligned} \frac{4\times10^6}{16(X+1)} &= 9600 \text{ bps} \\ X &= 25.04 \end{aligned}$$

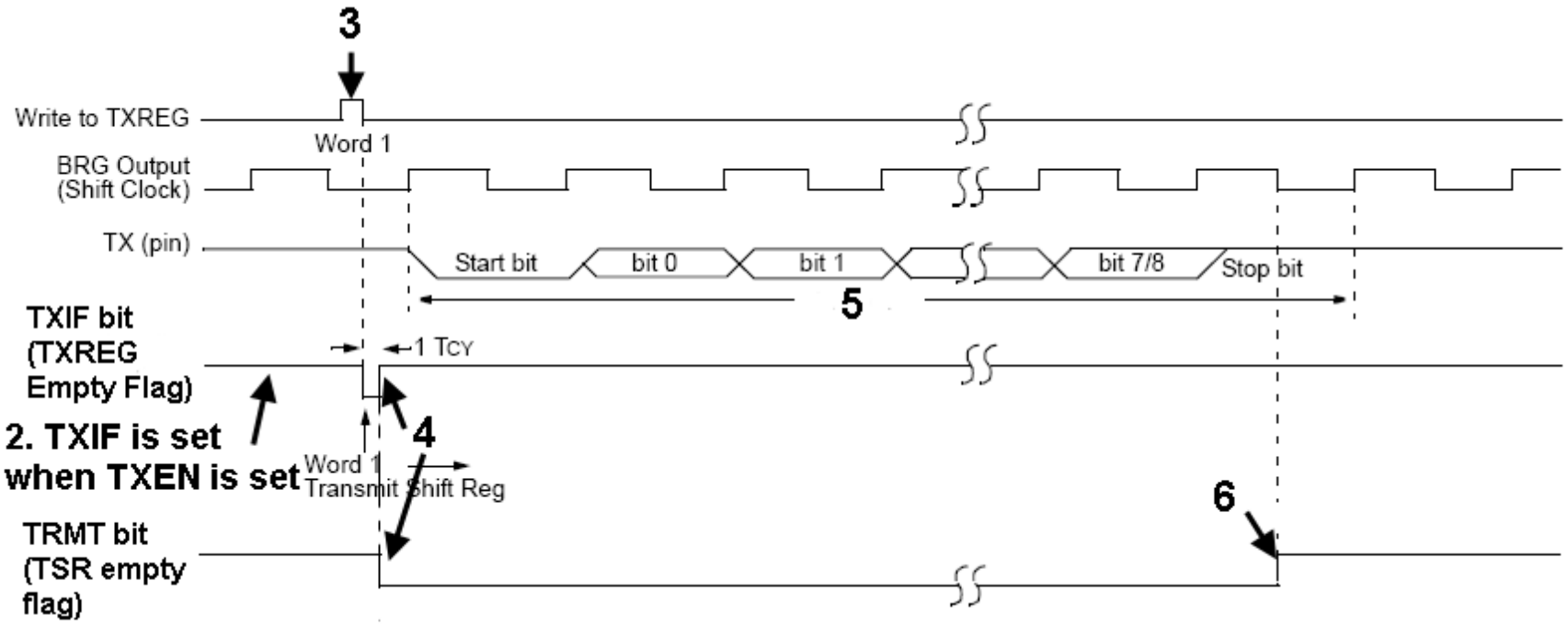
- Round to 25 and put it to SPBRG.

PIC18 UART Transmitter

- The program loads a byte to be transmitted into TXREG.
- If TSR is empty this same byte will be loaded into TSR.
- After loading the byte to TSR, this byte will be sent out starting from the next BRG shift clock cycle.
- TXIF (TXREG empty flag) and TRMT (TSR empty flag) flags are used to indicate different phases in the transmission.



UART TX in more detail



UART TX in more detail

1. Before transmission, three bits must be specified:
   - SPEN = 1 – to enable the UART module
   - TXEN = 1 – to enable the TX module
   - SYNC = 0 – asynchronous TX
2. TXIF (TXREG empty) flag is set when TXEN is set
3. Write to TXREG.
4. Two Events:
   - Content of TXREG is transferred to TSR. TXIF is set again.
   - TSR is filled. TRMT (TSR empty) flag becomes 0.
5. A byte, framed between the start and stop bit, is transmitted out.
6. Once the stop bit has been sent out, TRMT becomes 1.

Setting up Asynchronous Transmission

```
Main:     movlw   b'00100100'     ; TXEN = 1, SYNC = 0, BRGH = 1
          movwf   TXSTA
          movlw   b'10010000'     ; SPEN = 1
          movwf   RCSTA
          movlw   25              ; Set Baud rate for 9600
          movwf   SPBRG

          movlw   ‘A’
          call    putChar
          bra     $

putChar:  btfss   TXSTA, TRMT     ; wait until the last TX finishes
          bra     putChar         ; TRMT = 1 if TX finishes
          movwf   TXREG           ; Put ‘A’ into TXREG
          return
```

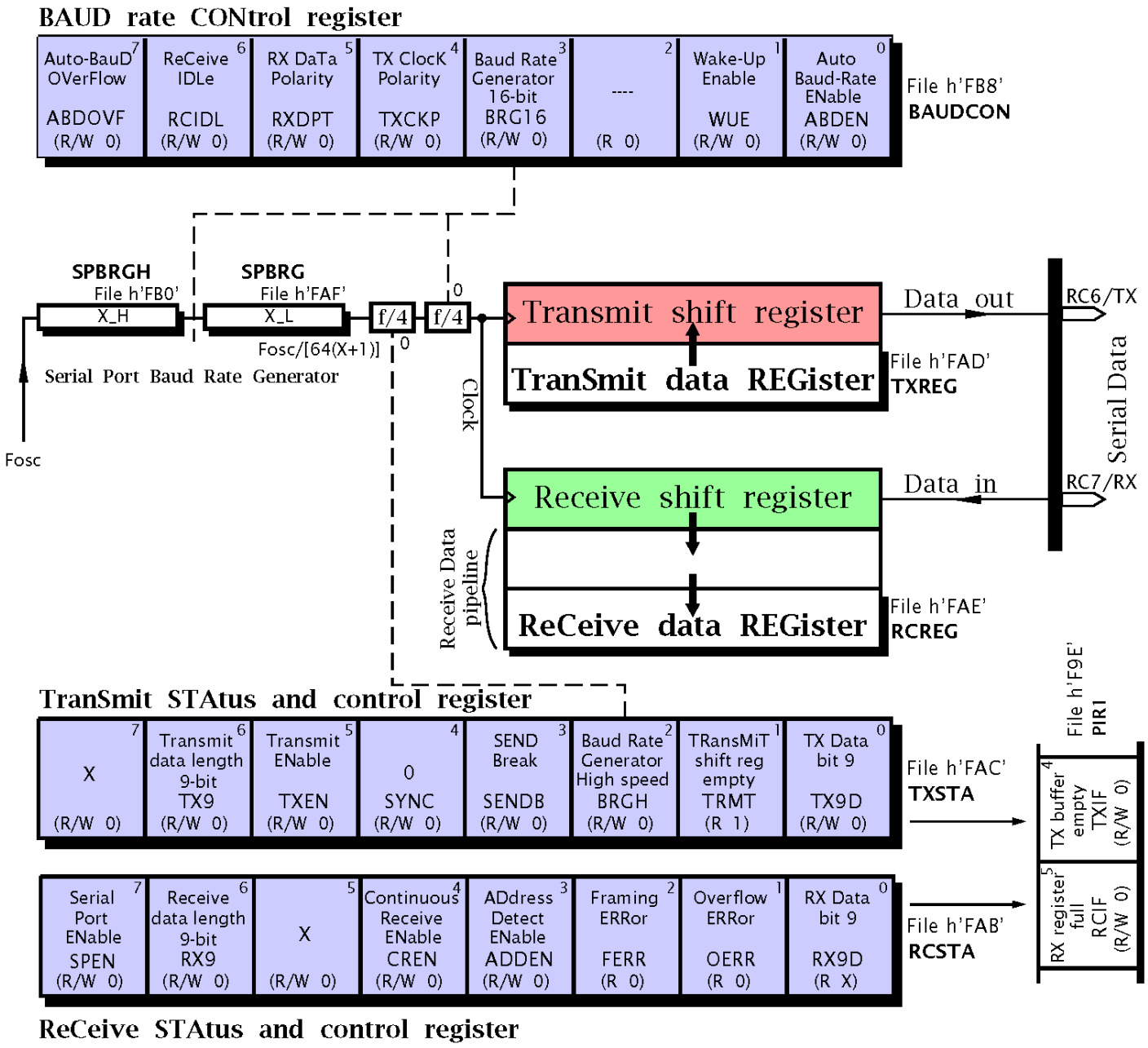
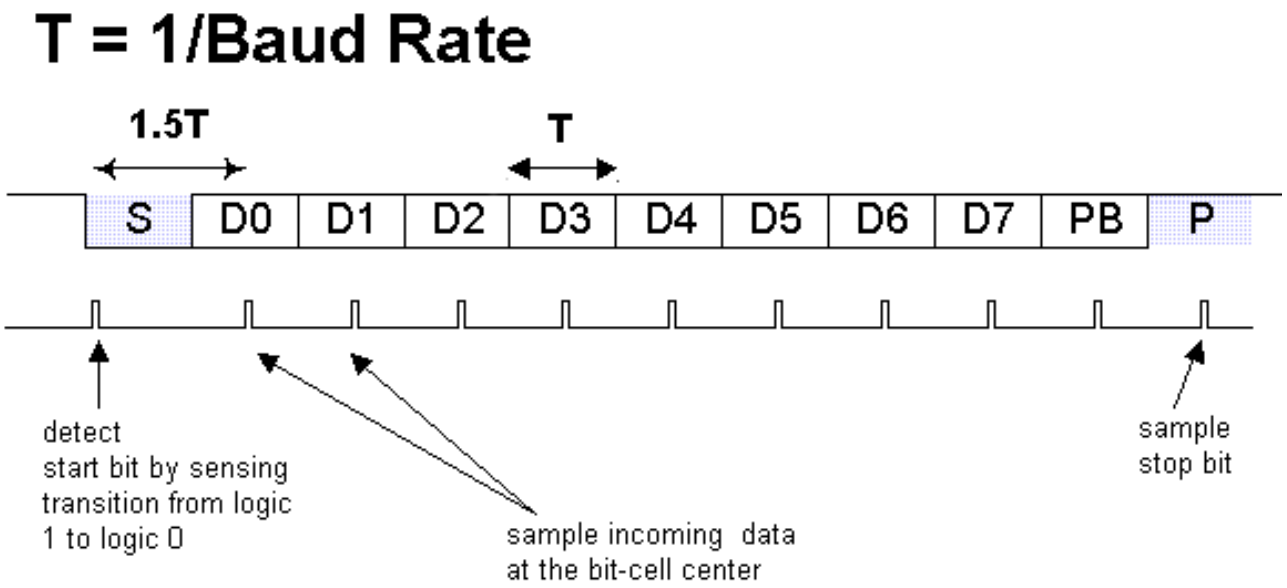


Image courtesy of Katzen, The essential PIC18 Microcontroller, Springer

PIC18 UART Reception



- Receiver must know the transmission baud rate in order to sample correctly.
- If the stop bit of 1 is not detected, *framing error* occurs.

UART Reception: Error Tolerance

- The value loaded to the SPBRG register(s) is constrained to be an integer number → Baud rate may deviate from desired frequency.
  - e.g., Suppose BRG16 = 0, BRGH = 1, f_{osc} = 4MHz and want to set baud rate as 9600 bps.

$$\frac{4\times10^6}{16(X+1)} = 9600 \text{ bps}$$

$$X = 25.04$$

  - Round to 25 and load to SPBRG
  - Actual baud rate

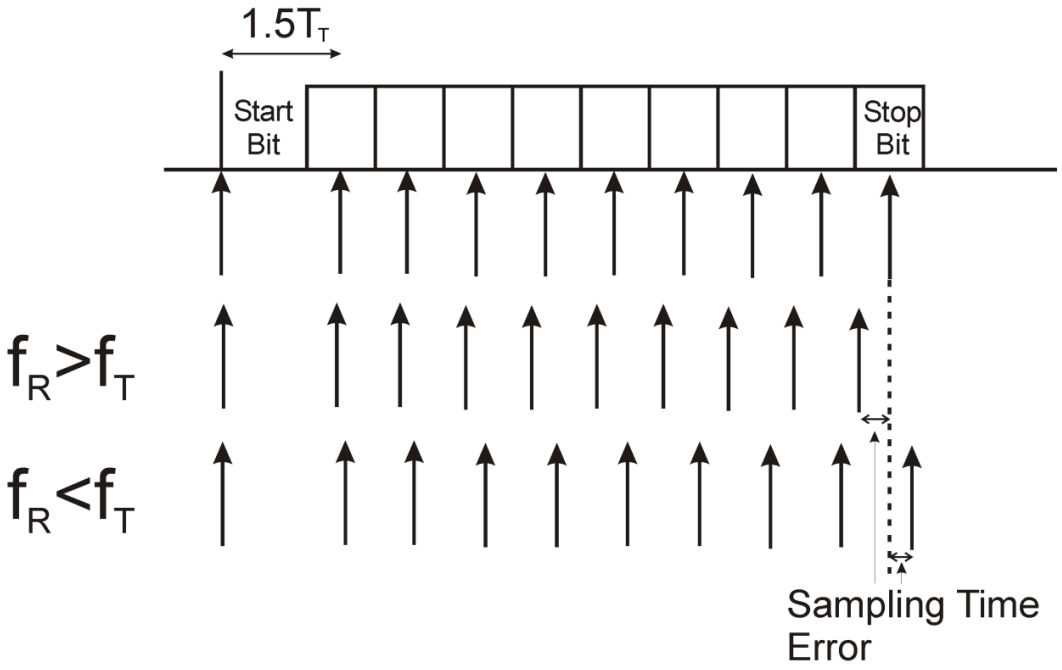
$$\frac{4 \times 10^6}{16(25+1)} = 9615 \text{ bps}$$

  - larger than 9600 bps

UART Reception: Error Tolerance

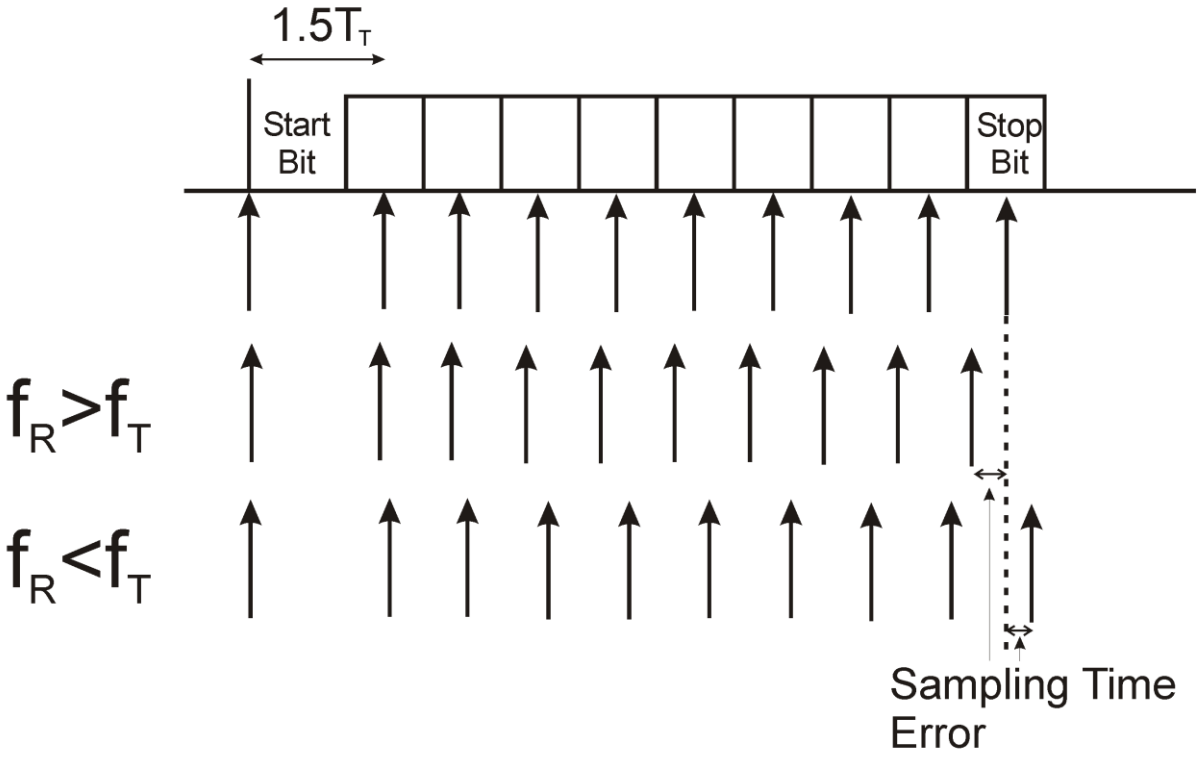
In communication with PC, MCU can serve as transmitter or receiver.

Tx	Rx	f_T	f_R	Remarks
PC	MCU	9600	9615	$f_R > f_T$; $T_R < T_T$ Receiver samples earlier than desired
MCU	PC	9615	9600	$f_R < f_T$; $T_R > T_T$ Receiver samples later than desired



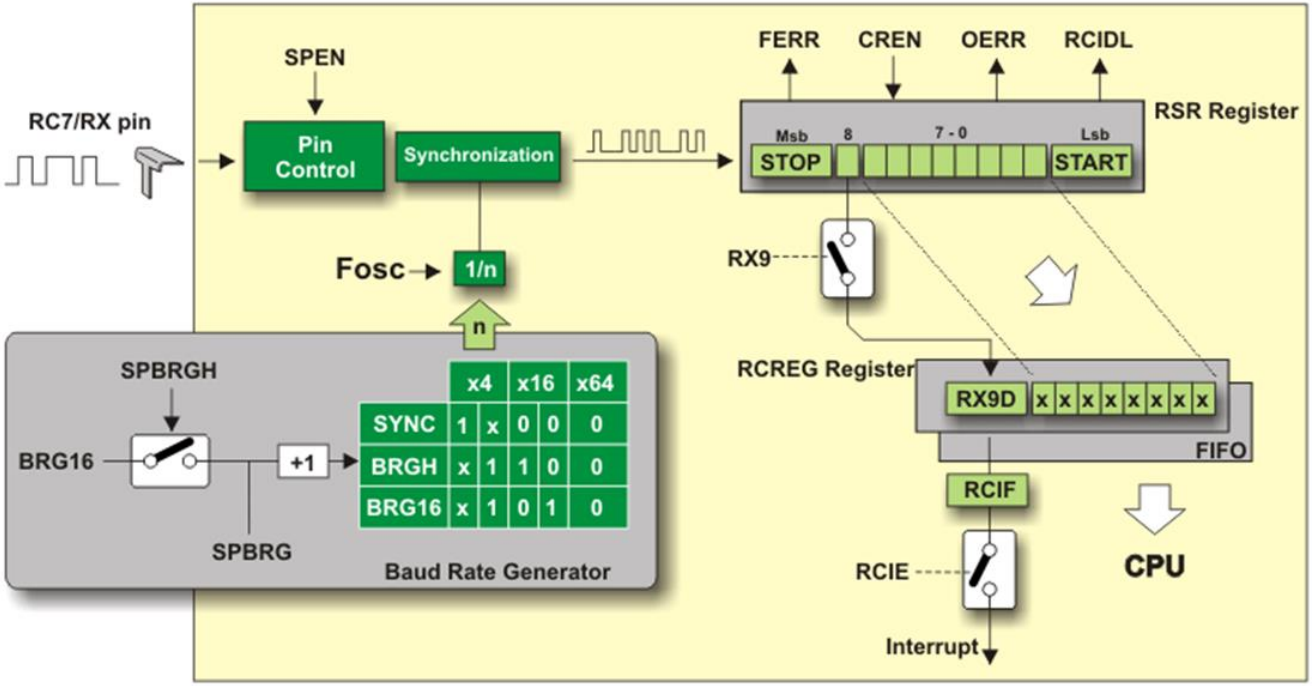
UART Reception: Error Tolerance

- Frequency drift of ±0.5 bit can be tolerated in the space of 10 bits (8 bits + start/stop bits).
- Receiver and transmitter local sample clocks must be within ±5%.
- e.g., % Error = (9615 – 9600)/9600 = 0.16% < 5%



PIC18 UART Reception

- USART peripheral needs to be enabled (RCSTA.SPEN)
- Receiving on UART needs to be enabled (RCSTA.CREN)
- The PIR1.RCIF flag will be raised by the microcontroller after it received a valid byte.
- The user needs to copy the received byte out of RCREG. This will automatically reset RCIF.



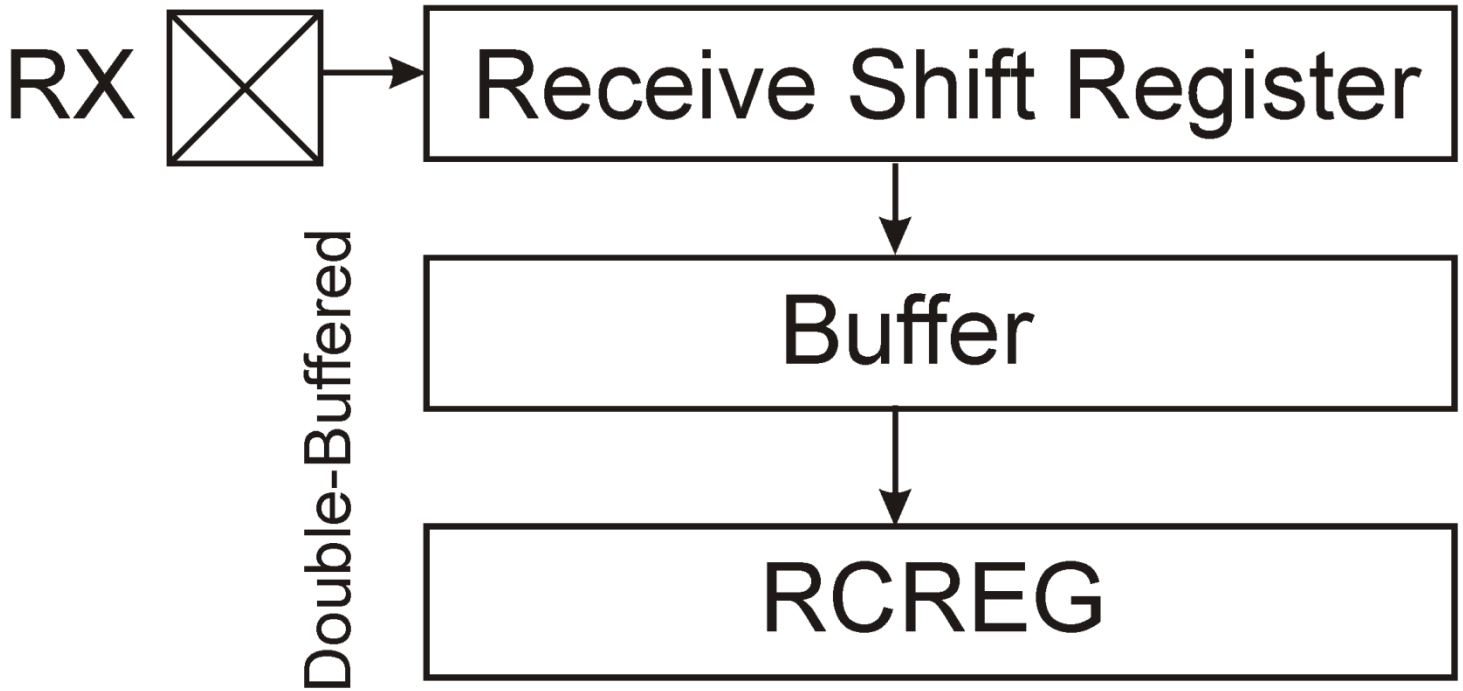
PIC18 UART Reception

This example receives bytes and output to PORTD

```
Main:       clrf    TRISD               ; set PORTD as output

            movlw   b'00100100'         ; TXEN = 1, SYNC = 0, BRGH = 1
            movwf   TXSTA
            movlw   b'10010000'         ; SPEN = 1, CREN = 1
            movwf   RCSTA
            movlw   25                  ; Set Baud rate for 9600
            movwf   SPBRG

MainLoop:   btfss   PIR1, RCIF          ; wait until receiving a complete byte
            bra     MainLoop
            movff   RCREG, PORTD        ; move the received byte to PORTD
            bra     MainLoop
```

Reception Overflow Error (OERR)

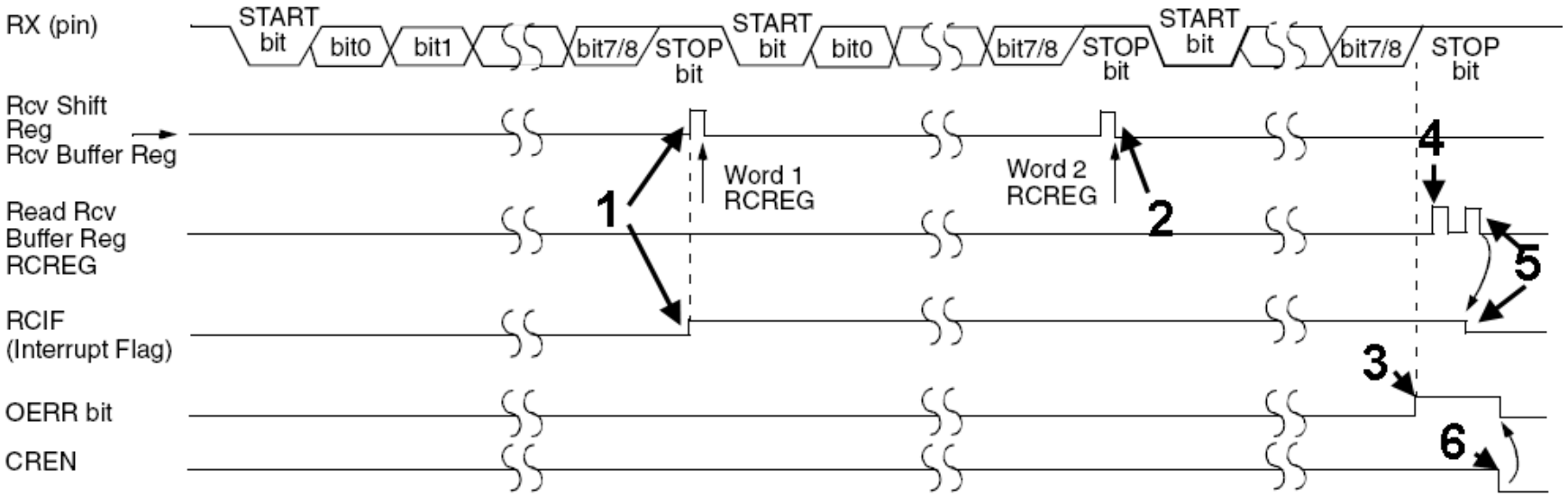
2-deep RCREG pipeline



What is Overflow Error?

- Normally, once a complete byte is received, it would be read from RCREG to another register.
- If a third byte arrives before the previous two bytes are read → no room to store the third byte → Overflow Error occurs

Overflow Error: A Detailed Look



Overflow Error: A Detailed Look

1. Received Word 1, which occupies the lower RCREG buffer. RCIF is set.
2. Received Word 2, which occupies the upper RCREG buffer.
3. 3rd byte arrives. RCREG is full. OERR is set. 3rd byte is lost.
4. Word 1 is read. Word 2 moves to the lower RCREG buffer. RCIF remains set.
5. Word 2 is read. RCIF is cleared.
6. OERR must be reset by clearing CREN and then setting it again. *No further bytes will be received until OERR is cleared.*

You should be able to ....

1. Explain why voltage conversion using MAX232 is needed.
2. Explain the operation of the PIC18 UART module.
3. Explain overflow and framing errors in UART reception.
4. Program the UART module to perform data transmission and reception.

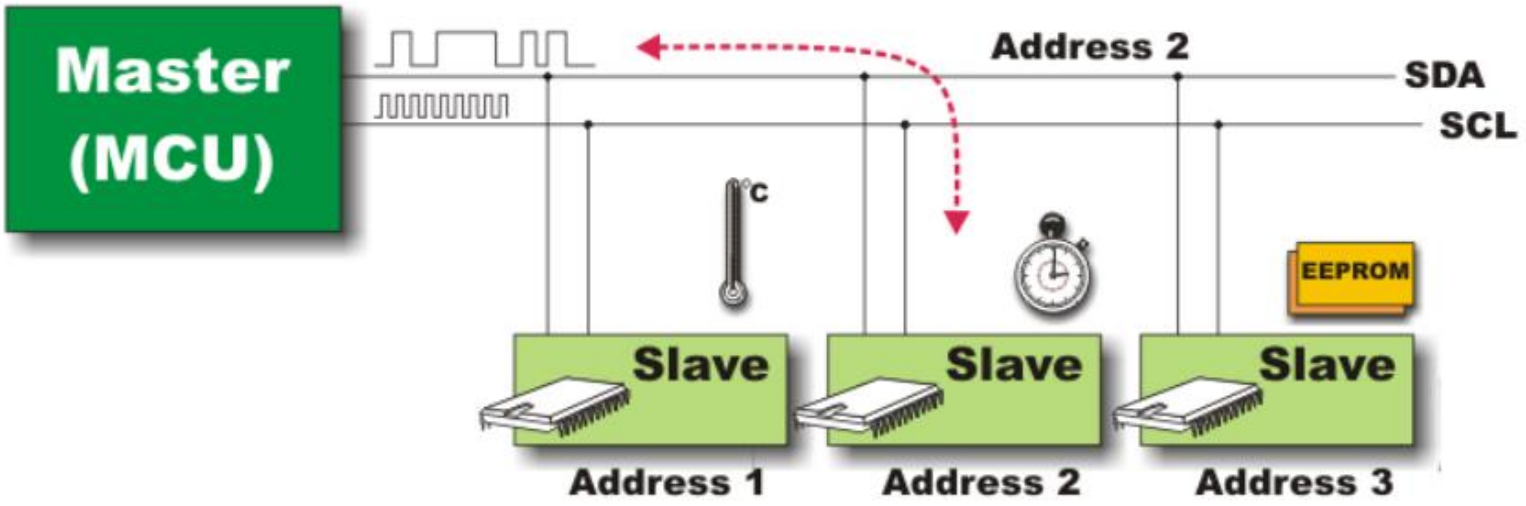
7.2: Inter-integrated Circuit Interface (I^2C)

Inter-Integrated Circuit (I^2C)

- Synchronous serial interface protocol for transmitting and receiving data from external devices.
- I^2C uses two lines: Serial CLock Signal (SCL) and Serial DAta (SDA)
- Data flow in I^2C is bidirectional (half duplex).

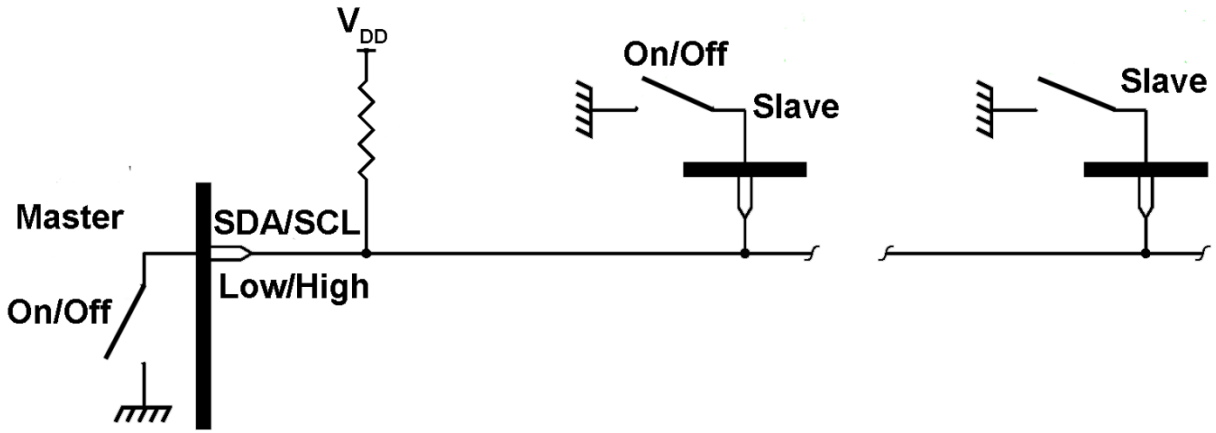
Master(s) and Slave(s)

- Master is responsible for:
  - Initializing/stopping data transfer
  - Control data transfer direction
  - Generating clock signals
- Typically, the microcontroller operates as the master and peripheral components (e.g., memory) as a slave.



Hardware Configuration

- SDA and SCL lines have two possible states: *float high* and *driven low*.
- Both the SDA and SCL lines are pulled up to V_{DD} by pull-up resistors at Idle state
- When the switch of the master or slave turns on, the signal is pulled down because there is current flowing from the I^2C bus to the switch.

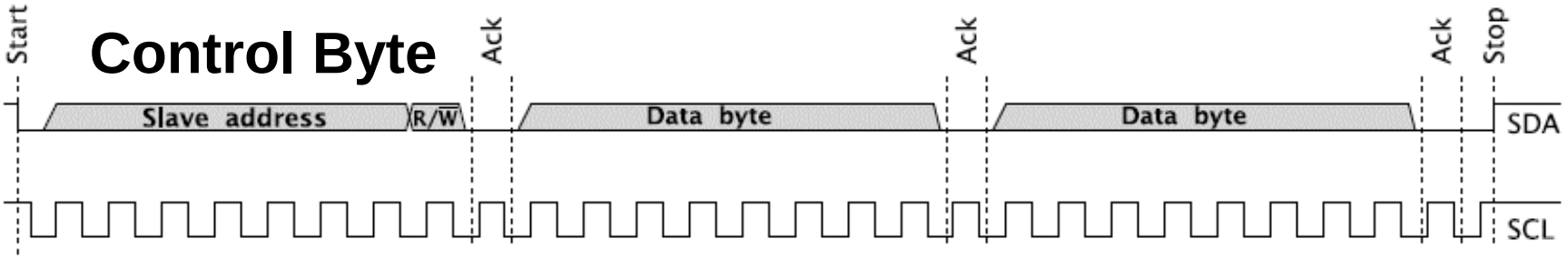


Start and Stop Conditions

- I^2C bus allows for more than one device to take over as master (although not simultaneously).
- Before a microcontroller transfers data to a slave, it needs to send a START condition.
- When transfer of data ends, the microcontroller needs to relinquish the control by sending a STOP condition.

Data Transfer between Master and Slave

1. Master sends out **START** condition
2. Master sends out a **Control Byte,** consisting of the 7-bit slave address and a bit (the last bit) indicating whether the master reads from or writes to the slave.
3. Data bytes are sent until a **STOP** condition is asserted by the master.

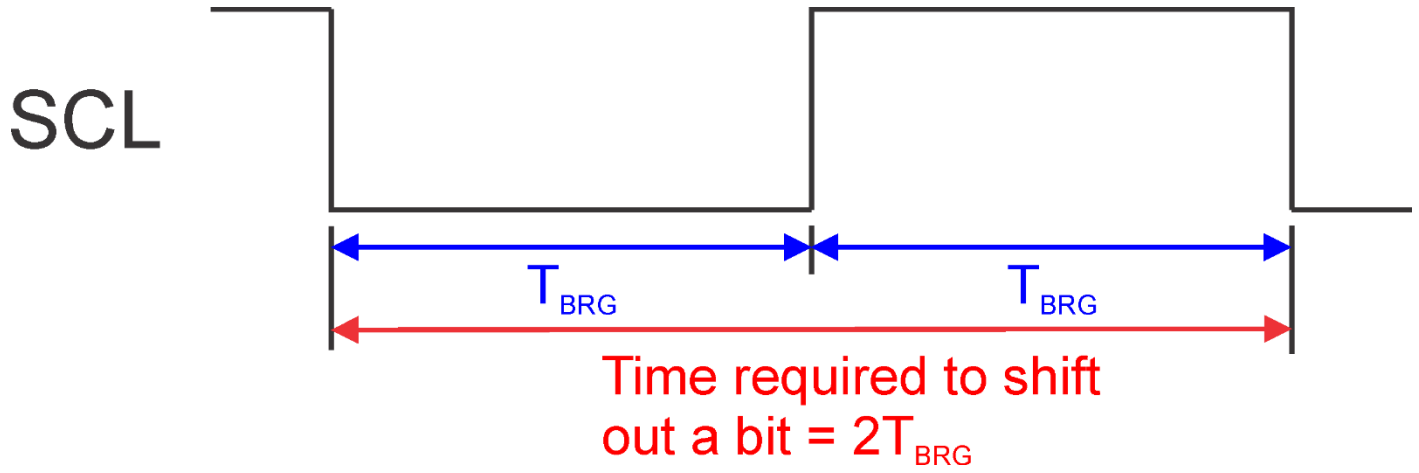


PIC18 MSSP Module in I²C Mode

- Registers supporting I²C operations:
  - MSSP Control Register 1 (SSPCON1)
  - MSSP Control Register 2 (SSPCON2)
  - MSSP Status Register (SSPSTAT)
  - Serial Receive/Transmit Buffer (SSPBUF)
  - MSSP Shift Register (SSPSR)—not directly accessible
  - MSSP Address Register (SSPADD)

Baud Rate Generator (BRG)

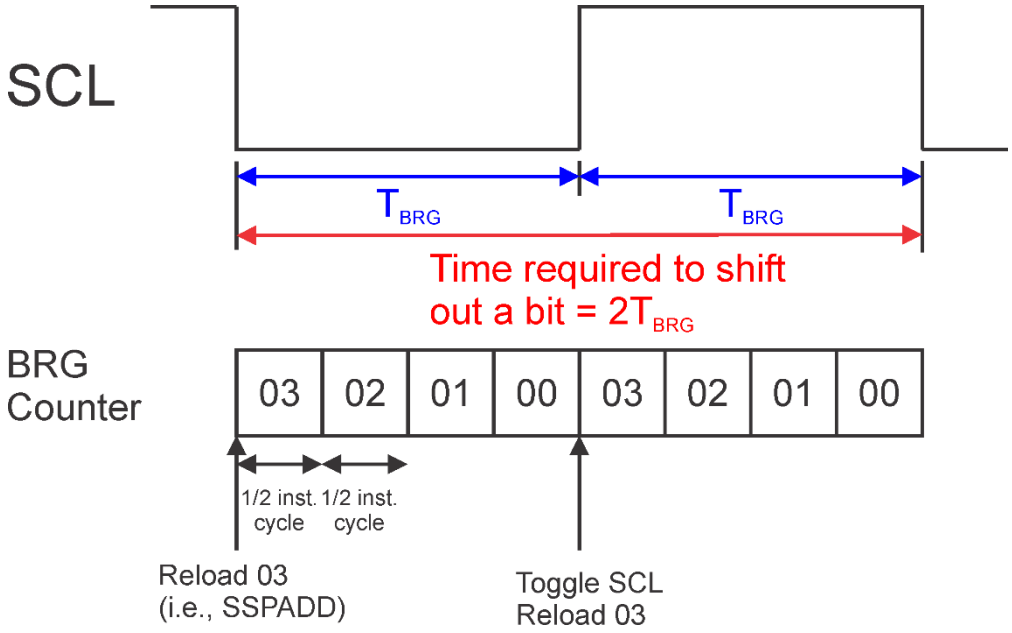
- Time required to shift out one bit of data = $2T_{BRG}$, where T_{BRG} = time required for the *BRG counter* to count down to 0 from an initial value



- The initial value of BRG counter is placed in the lower 7 bit of register SSPADD (i.e., SSPADD<0:6>), which is set by users.
- BRG counter is decremented twice per instruction cycle.

Baud Rate Generator (BRG)

e.g., If SSPADD = 03, Clock freq. = F_{OSC} = 4MHz, what is the baud rate?



$$T_{BRG} = \frac{1}{2}\frac{4}{F_{OSC}}(\text{SSPADD}+1)$$

$$\text{Baud Rate} = \frac{1}{2T_{BRG}} = \frac{F_{OSC}}{4(\text{SSPADD}+1)}$$

$$= \frac{4\times10^6}{4(3+1)} = 250 \text{ kHz}$$

Baud Rate

$T_{BRG} = (4\ T_{OSC}/2) \times (SSPADD<0:6>+1)$

$\text{Baud Rate} = 1/(2\ T_{BRG})$

$= F_{OSC} / [4 \times (SSPADD<0:6>+1)]$

where T_{OSC} = Oscillation period of clock

$F_{OSC} = 1/T_{OSC}$ = Clock frequency

Baud Rate Example

- What is the initial value we need to load to the SSPADD register if we want to set the baud rate to be 100kHz? Assume the clock has a frequency of 4MHz.
- Let x = initial value
- Baud Rate = 4MHz / [4 $\times$ (x+1)] = 100kHz
- x = 0x09

Start Condition

- Start condition is initiated by writing to SEN bit (SSPCONS2<0>)
- S bit (SSPSTAT<3>) is set during start
- When completed, SEN bit is cleared and SSPIF is set.

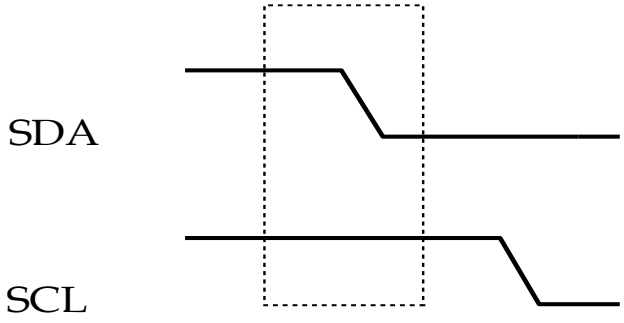
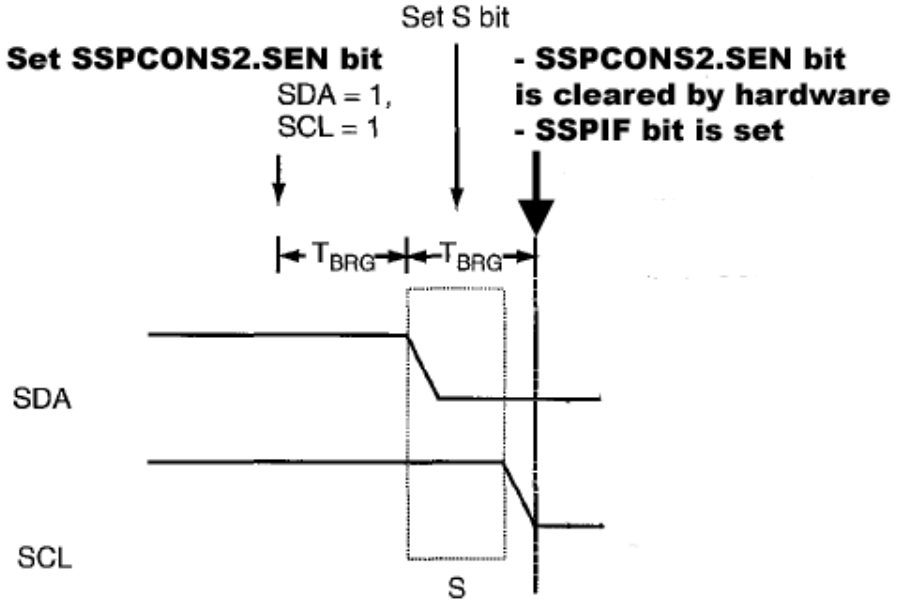


Figure 11.2 I²C Start condition



Assert a Start Condition

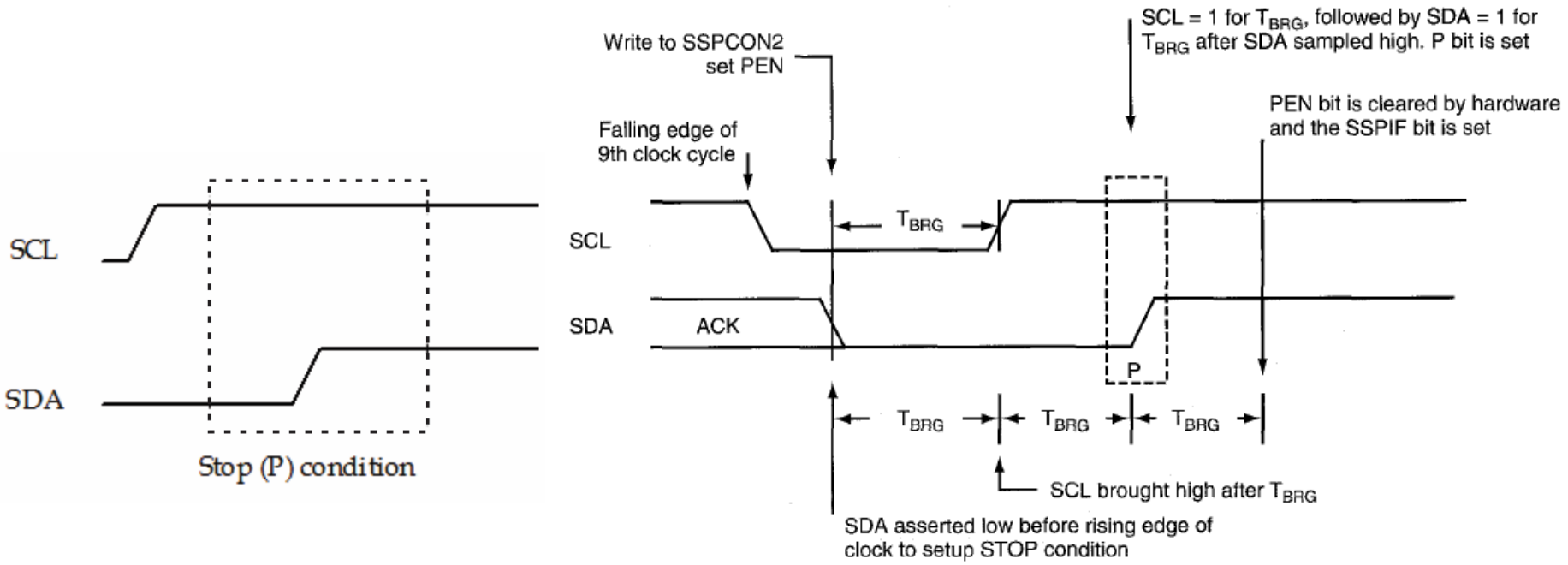
- Assert the start condition by setting the SEN bit in SSPCON2:

```
        bsf     SSPCON2, SEN
```

- Wait for the start condition to complete

```
I2C_WaitFinish: bcf PIR1, SSPIF
I2C_WaitLoop:   btfss  PIR1, SSPIF
                bra I2C_WaitLoop
                return
```

Stop Condition



Stop Condition

- Assert the stop condition by setting the PEN bit (SSPCON2<2>):

```
bsf     SSPCON2, PEN
```

- Wait for SSPIF to get asserted, indicating completion:

```
bra     I2C_WaitFinish
```

Repeated Start Condition

Imagine this situation:

- Master device wants to change data transfer direction (i.e., write to read or vice versa) but without losing control of the line
- Need to send out a new control byte (Last bit of the control byte determines data transfer direction.)
- But, control byte needs to be preceded by a Start condition.
- Start condition can only be asserted if both SDA and SCL float high →usually achieved by asserting a Stop condition.
- Master cannot assert a Stop condition because it does not want to lose control of the line.
- Master asserts a *Repeated Start* condition

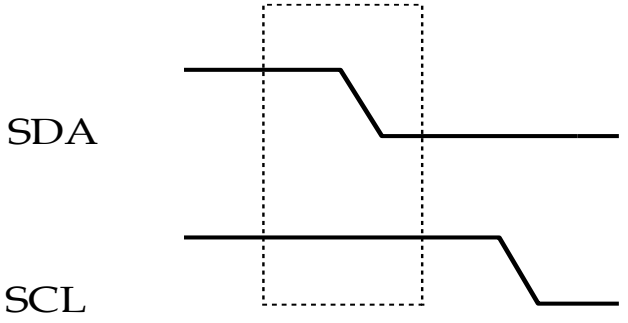


Figure 11.2 I²C Start condition

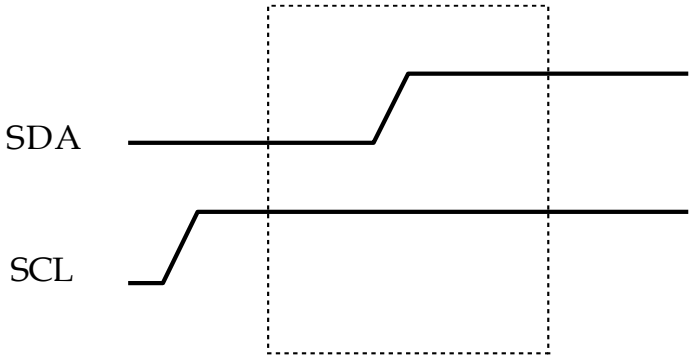
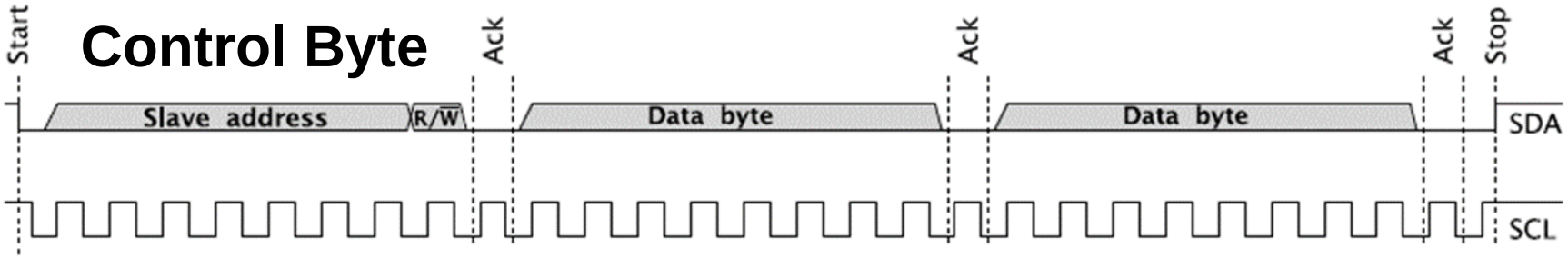


Figure 11.3 Stop (P) condition



Repeated Start Condition

- So, is it possible to bring SDA and SCL high, but without asserting a Stop condition?
- Solution: Bring SDA high first, then bring SCL high.

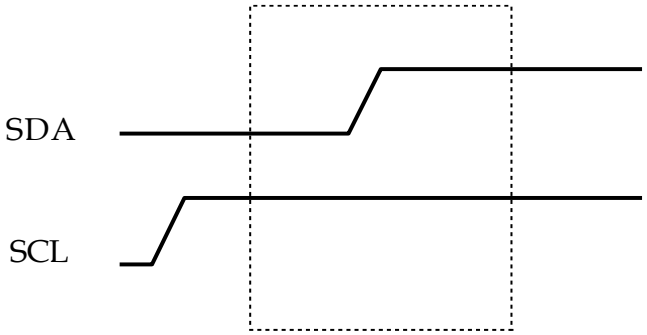
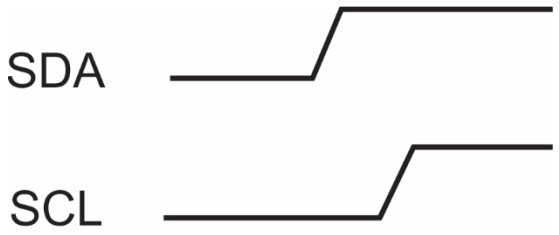


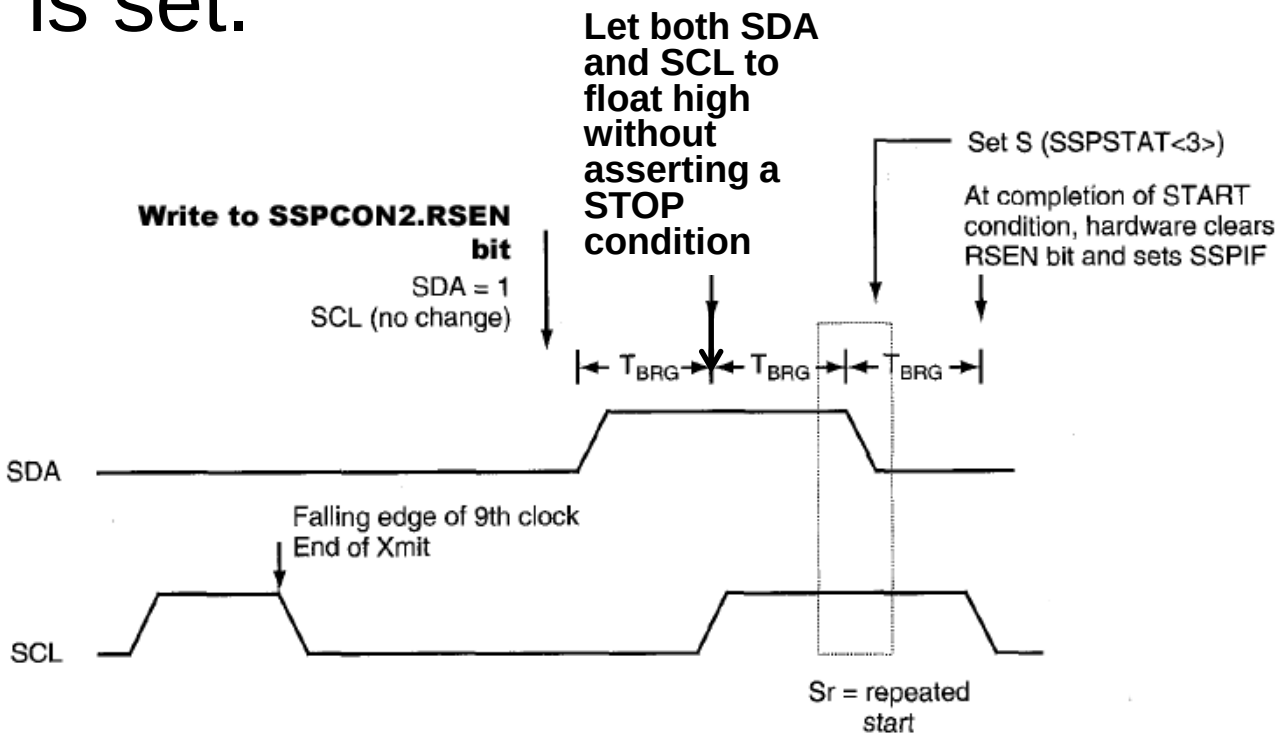
Figure 11.3 Stop (P) condition



Bring SDA, SCL to high without asserting the Stop condition

Repeated Start Condition Timing

- Activate RSEN bit (SSPCON2<1>)
- S bit (SSPSTAT<3>) is set during repeated start
- When completed, RSEN bit is cleared and SSPIF is set.



Assert a Repeated Start Condition

- Assert the repeated start condition by setting the RSEN bit in SSPCON2:

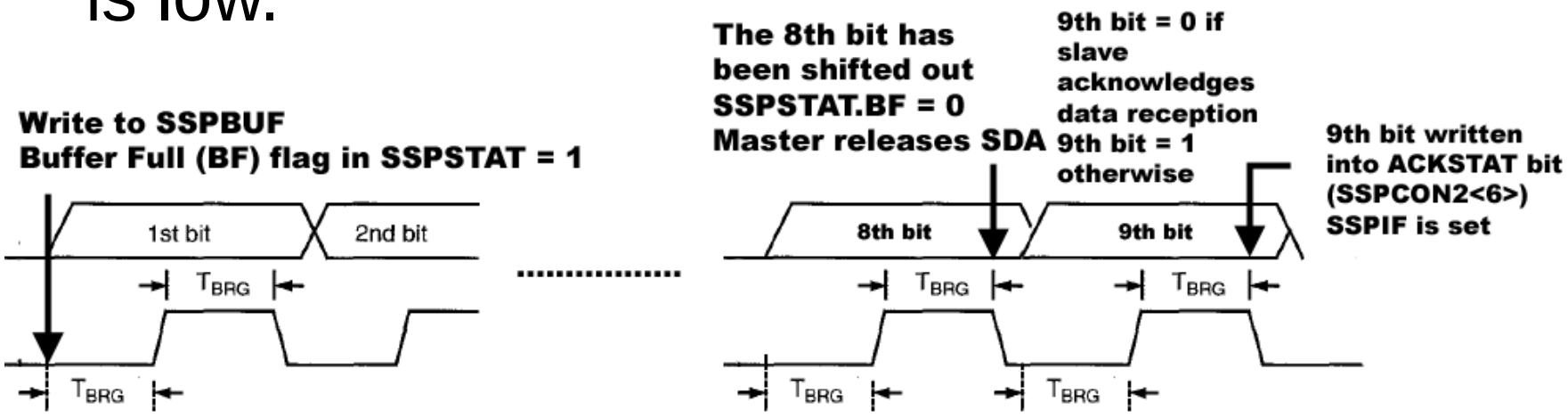
```
bsf     SSPCON2, RSEN
```

- Wait for SSPIF to get asserted:

```
bra     I2C_WaitFinish
```

Transmission

- Transmission of a data byte or an address is achieved by writing to the SSPBUF register. BF flag (SSPSTAT<0>) is set.
- Data is clocked into the receiver on SCL rising edge.
- Changes on SDA must only occur when SCL is low.



Transmission

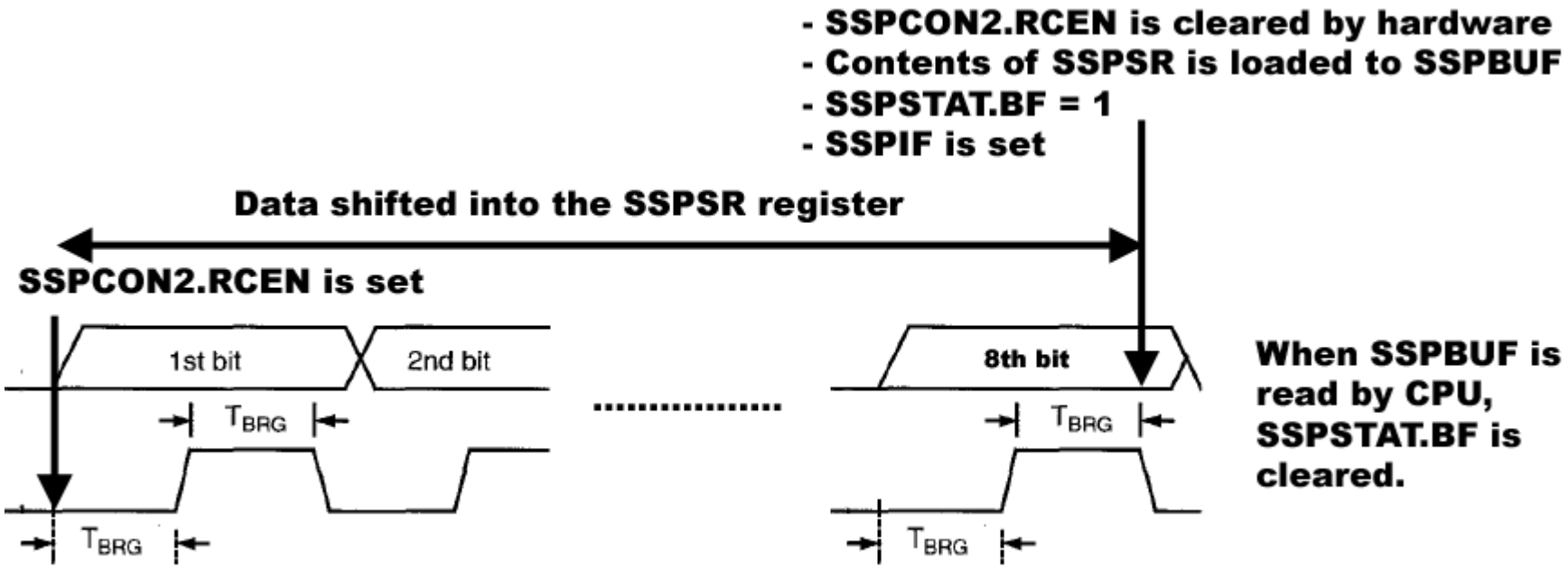
- To write data/address to I^2C bus:
  - First write to WREG register
  - Then move the value of WREG to SSPBUF

```
movwf  SSPBUF
```

  - Finally, wait for the SSPIF signal to get asserted:

```
bra  I2C_WaitFinish
```

Reception


- SSPCON2.RCEN is cleared by hardware
- Contents of SSPSR is loaded to SSPBUF
- SSPSTAT.BF = 1
- SSPIF is set
Data shifted into the SSPSR register
SSPCON2.RCEN is set
1st bit
2nd bit
8th bit
TBRG
TBRG
TBRG
TBRG
When SSPBUF is read by CPU, SSPSTAT.BF is cleared.

Reception

- To start reception:

```
bsf   SSPCON2, RCEN
```

- Wait for SSPIF to get asserted:

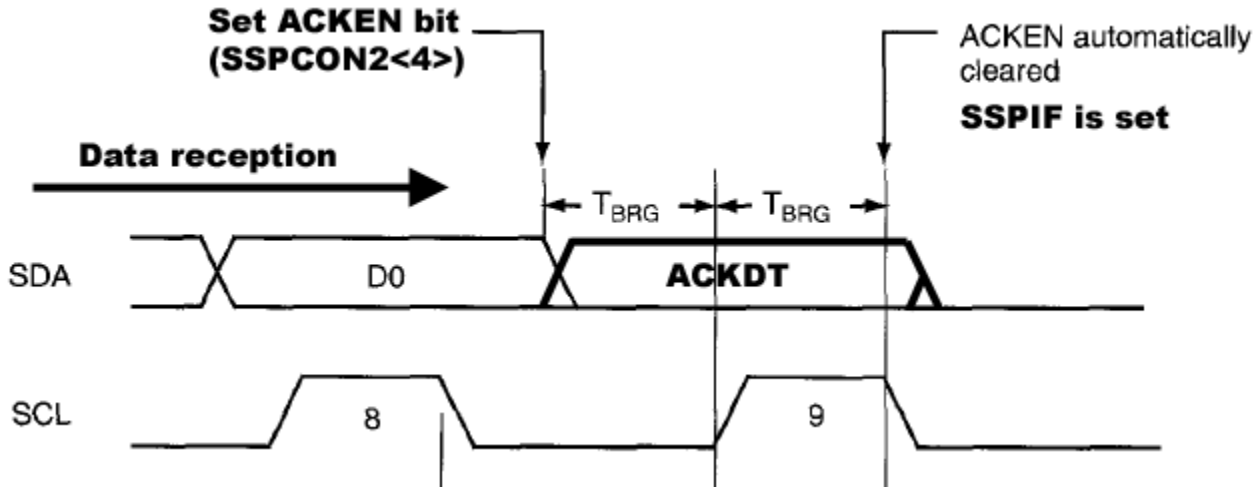
```
bra   I2C_WaitFinish
```

- Write SSPBUF into a register for further processing:

```
movff  SSPBUF, I2C_BYTE
```

Acknowledge Reception

- Clear the ACKDT bit (SSPCON2<5>) to acknowledge reception. Set ACKDT to send a negative acknowledge
- Set ACKEN to activate the acknowledgement process



Acknowledge Reception

- Set/Clear ACKDT:

```
bsf (bcf)  SSPCON2, ACKDT
```

- Set ACKEN to activate acknowledgement:

```
bsf  SSPCON2, ACKEN
```

- Wait for SSPIF to get asserted, indicating completion:

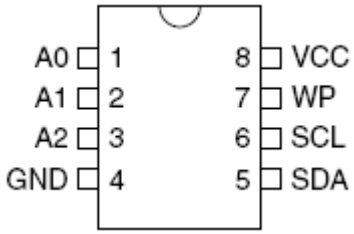
```
bra  I2C_WaitFinish
```

I²C Interface with AT24C02 EEPROM

- AT24C02 provides 2048 bits of serial EEPROM organized as 256 8-bit words.
- Control byte

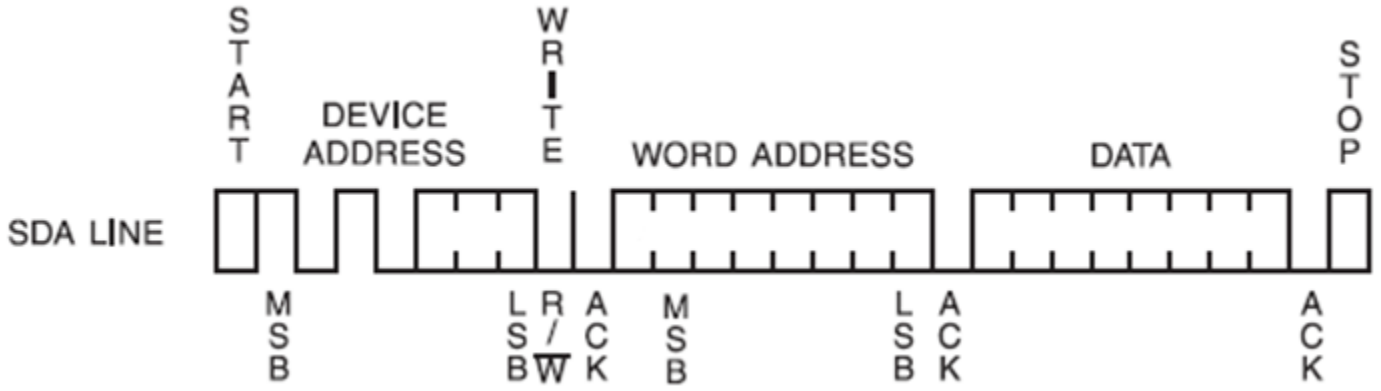
1	0	1	0	A2	A1	A0	R

  - A2, A1 and A0 are determined by inputs to the corresponding pins
  - R = 0 for write operation and R = 1 for read operation



Byte Write

1. The master asserts the Start condition
2. The master sends the control byte to AT24C02
3. AT24C02 acknowledges the reception of the control byte
4. The master sends the word address (address of the word inside AT24C02 ranges from 00 to FF) to AT24C02
5. AT24C02 acknowledges the reception of the word address
6. The master sends the data byte to AT24C02
7. The master asserts the Stop condition



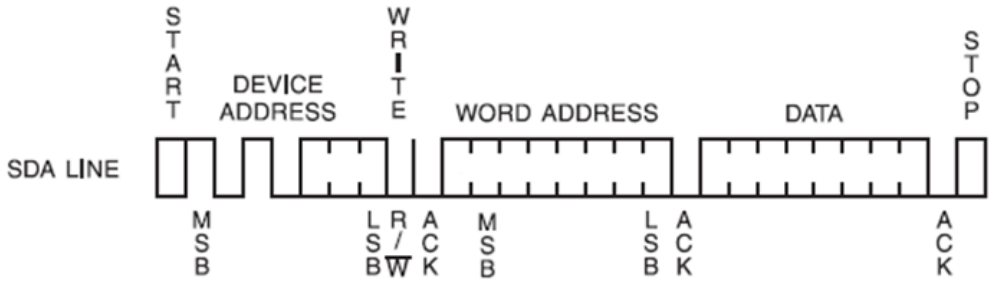
Byte Write

Recap what we defined

```
I2C_Start:      bsf     SSPCON2, SEN
                bra     I2C_WaitFinish

I2C_Stop:       bsf     SSPCON2, PEN
                bra     I2C_WaitFinish

I2C_Restart:    bsf     SSPCON2, RSEN
                bra     I2C_WaitFinish

I2C_SendAck:    bsf     SSPCON2, ACKEN
                bra     I2C_WaitFinish

I2C_SendByte:   movwf   SSPBUF
                bra     I2C_WaitFinish

I2C_RcvByte:    bsf     SSPCON2, RCEN

I2C_WaitFinish: bcf     PIR1, SSPIF
I2C_WaitLoop:   btfss   PIR1, SSPIF
                bra     I2C_WaitLoop
                return
```

Write to EEPROM

```
I2C_Write:      call    I2C_Start           ;Start
                movlw   0xA0
                call    I2C_SendByte        ;Control Byte
                movf    I2C_ADDR, W
                call    I2C_SendByte        ;Word Address
                movf    I2C_BYTE, W
                call    I2C_SendByte        ;Data
                call    I2C_Stop            ;Stop
                return
```

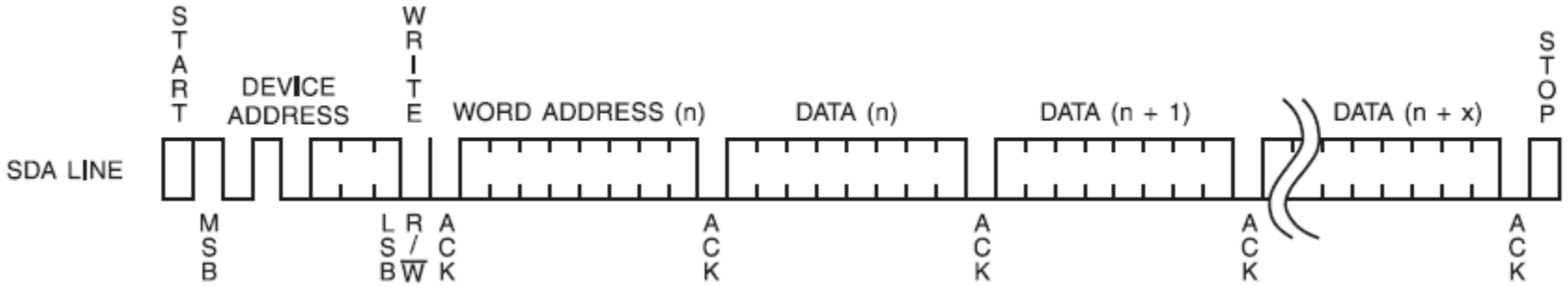


Acknowledgement Polling

- Once the Stop condition has been issued after the write operation, the EEPROM enters the internal write cycle.
- The duration of this cycle depends on the device.
- *Acknowledgement polling* involves sending a start condition followed by the control byte.
- If the internal write cycle has not been completed, the EEPROM would response with a NAK; otherwise it would response with an ACK.

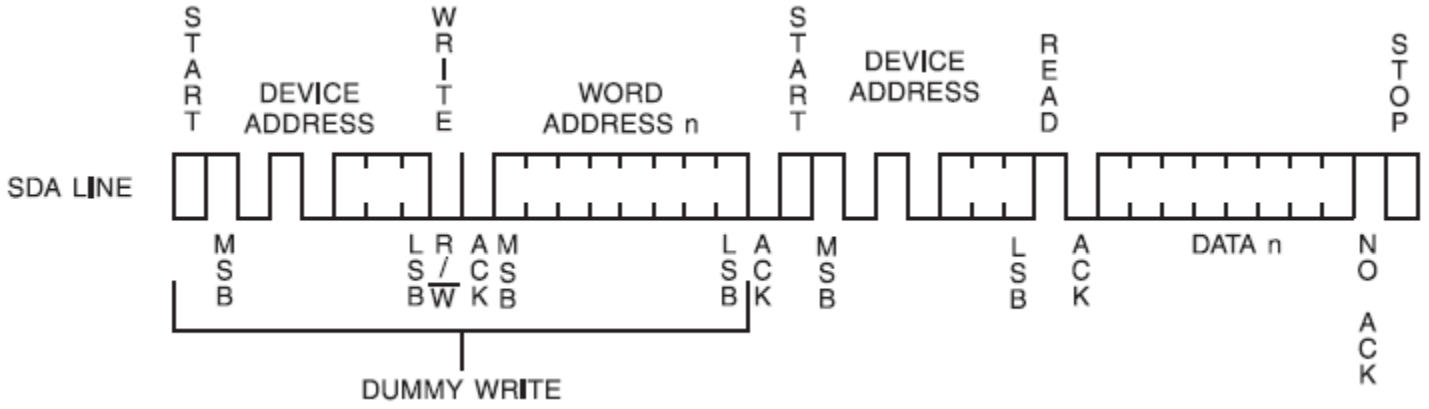
Page Write

- Instead of sending a Stop condition after the transmission of the 1st byte, send 7 more bytes.
- If more than 8 bytes are written, the data word address would “roll over”.



Byte Read

1. Perform Steps 1 to 5 of Byte Write to define the device address and word address to read from.
2. The master asserts the Repeated Start condition
3. The master sends the control byte with last bit equal to 1 indicating a read operation
4. AT24C02 acknowledges the reception of the control byte
5. The master asserts the receive condition
6. The master receives the data byte and asserts NACK to the EEPROM
7. The master asserts the stop condition



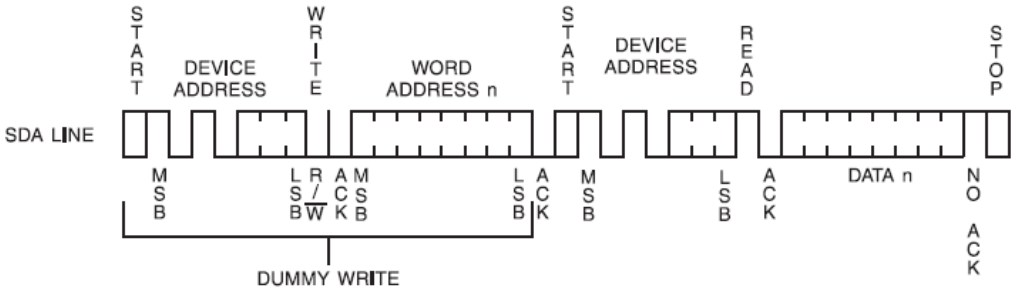
Byte Read

Conditions previously defined

```
I2C_Start:        bsf     SSPCON2, SEN
                  bra     I2C_WaitFinish

I2C_Stop:         bsf     SSPCON2, PEN
                  bra     I2C_WaitFinish

I2C_Restart:      bsf     SSPCON2, RSEN
                  bra     I2C_WaitFinish

I2C_SendAck:      bsf     SSPCON2, ACKEN
                  bra     I2C_WaitFinish

I2C_SendByte:     movwf   SSPBUF
                  bra     I2C_WaitFinish

I2C_RcvByte:      bsf     SSPCON2, RCEN

I2C_WaitFinish:   bcf     PIR1, SSPIF
I2C_WaitLoop:     btfss   PIR1, SSPIF
                  bra     I2C_WaitLoop
                  return
```

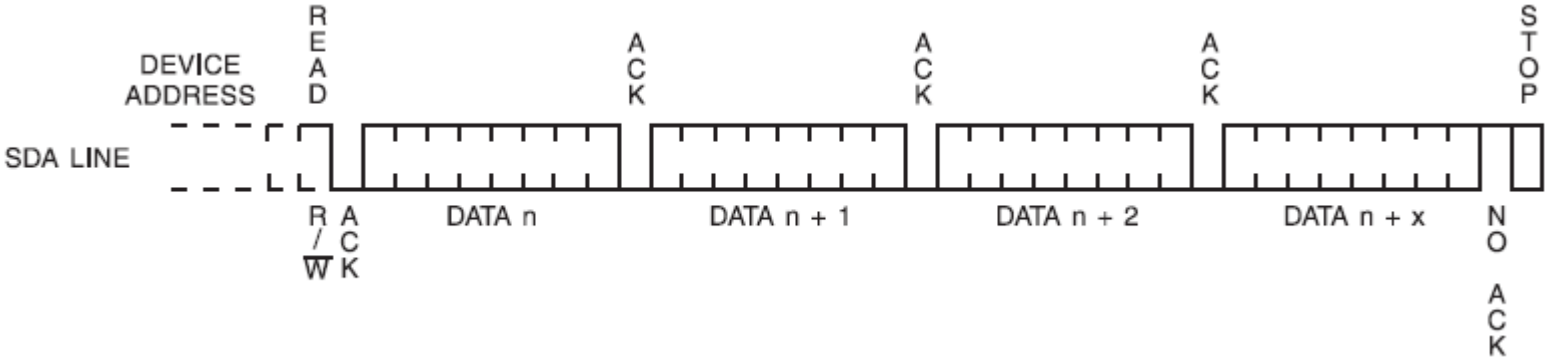
Byte Read from EEPROM

```
I2C_Read:         call    I2C_Start           ;Start
                  movlw   0xA0
                  call    I2C_SendByte        ;Control Byte
                  movf    I2C_ADDR, W
                  call    I2C_SendByte        ;Word Address
                  call    I2C_Restart         ;Restart
                  movlw   0xA1
                  call    I2C_SendByte        ;Control Byte
                  call    I2C_RcvByte         ;Data
                  movff   SSPBUF, I2C_BYTE
                  bsf     SSPCON2, ACKDT      ;Send NACK
                  call    I2C_SendAck
                  call    I2C_Stop            ;Stop
                  return
```



Sequential Read

- Instead of sending a NACK after the first byte is read, the master sends out an ACK.
- The operation terminates when the master sends out a NACK bit and assert the stop condition
- Sequential read is not limited to 8 bytes.



You should be able to ...

- Understand the characteristics of the I^2C protocol
- Describe different conditions that needs to be asserted during a I^2C transfer.
- Design the baud rate of the data transfer by setting the values in SSPADD.
- Describe and implement different methods of interfacing with the AT24C02 EEPROM.