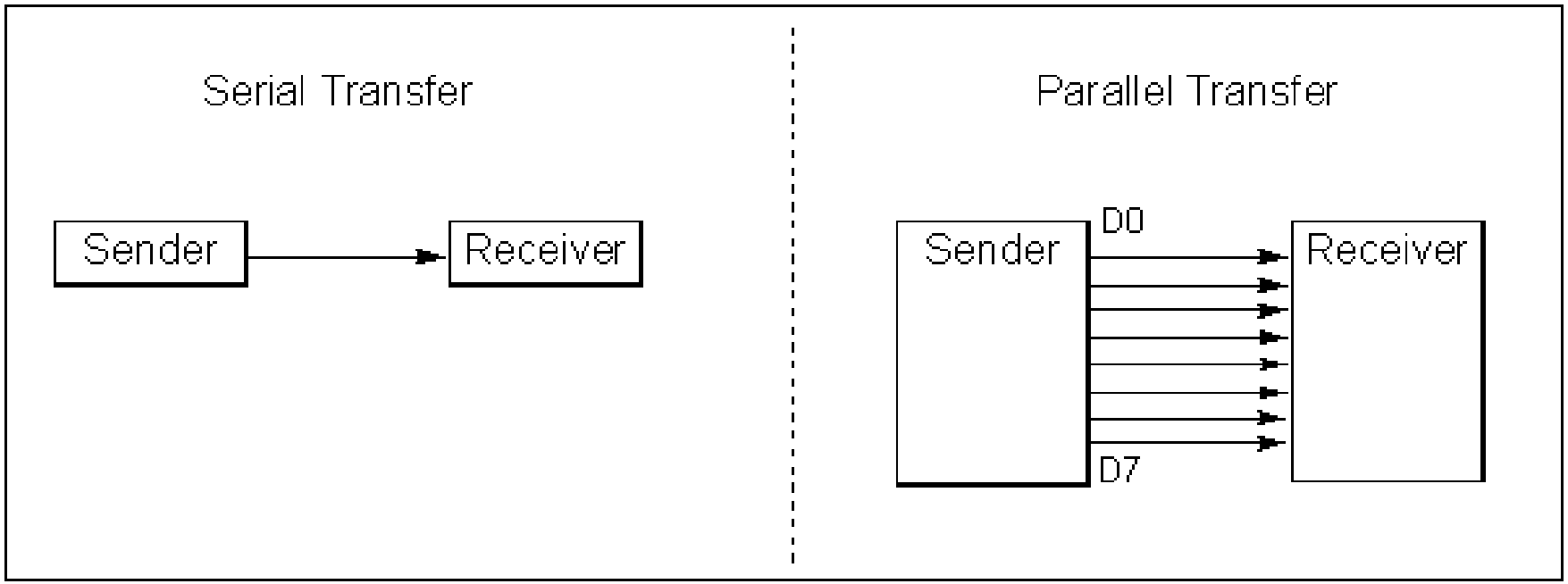
Chapter 5 Serial communication

* basics of serial communication
* USART

5.1 basics of serial communication

- computers transfer data in 2 ways – parallel, serial
- parallel:
  8 or more data lines – expensive
  short distance (few feet) – diminishing signal, interference
- serial:
  one data line – cheaper
  long distance – require modulation/demodulation
- PIC18 has serial communication capability built into it

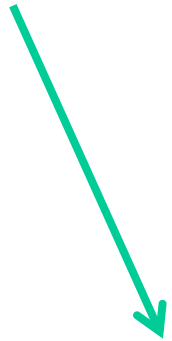


Serial versus Parallel Data Transfer

- the byte of data must be converted to serial bits using a parallel-in-serial-out shift register
- data is sent one bit at a time
- at the receiving end, there must be a serial-in-parallel-out shift register – pack bits into a byte

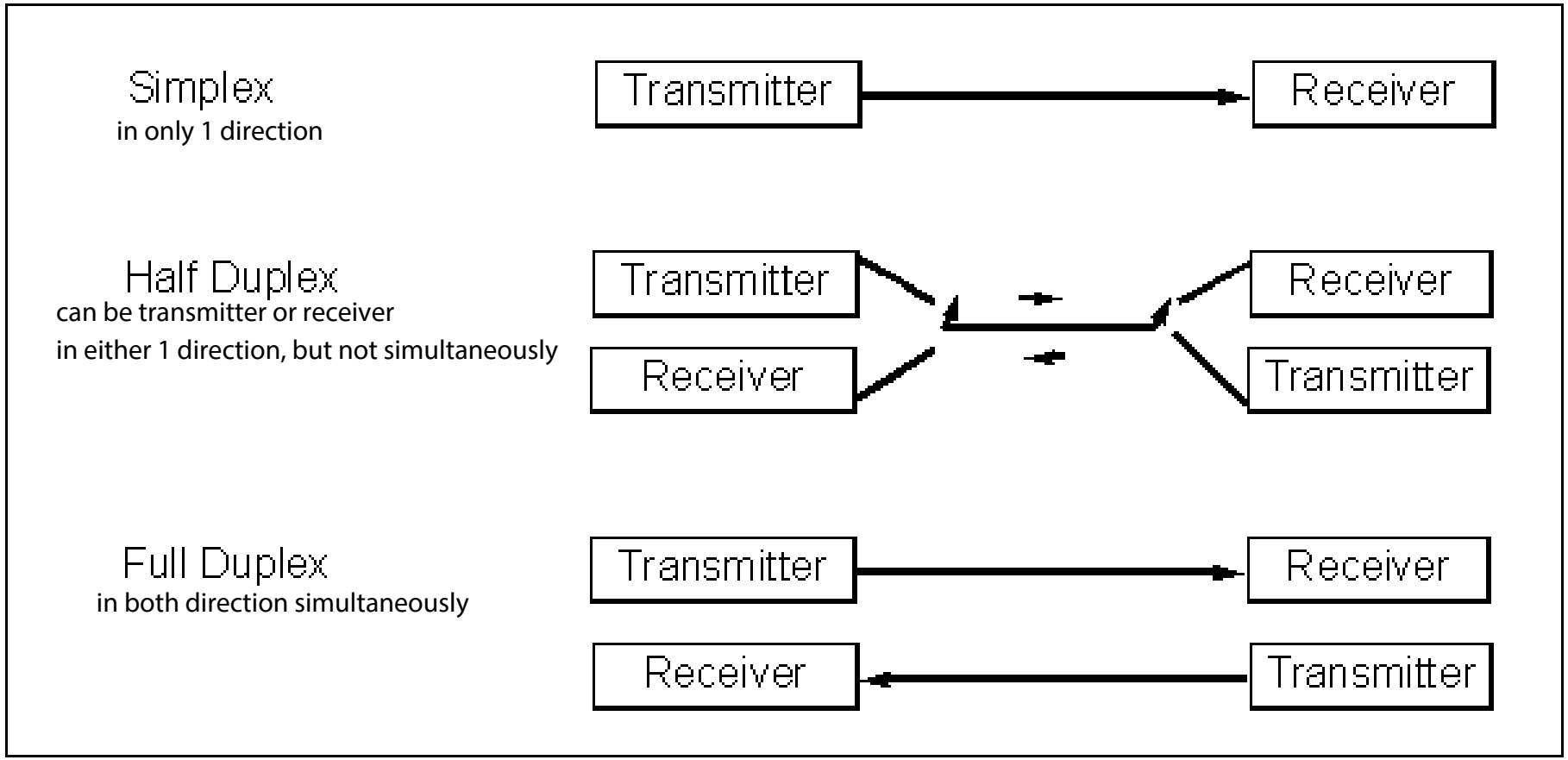
- two methods of serial data communication: Synchronous and Asynchronous

Transfers a single byte at a time



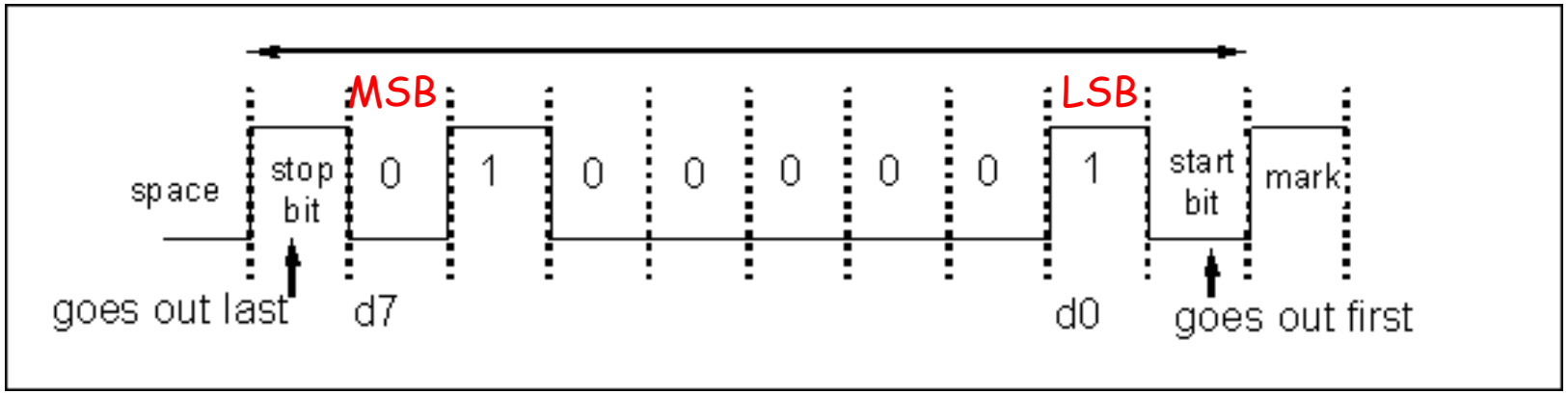
Transfers a block of data at a time

Half- and Full-Duplex transmission



Start and Stop Bits

- in the asynchronous method, each character is placed between start and stop bits (framing)



Framing ASCII 'A' (41H)

An asynchronous serial data link is said to be character oriented (7 or 8 information bits and plus several control bits). When no information is being transmitted, the line is in an idle state. Traditionally, idle state is referred to as the mark level - a logical 1 level.

A start bit: This bit precedes the first data bit indicating the beginning of a character. The start bit is a logical zero, commonly referred to as a space.

Seven to eight data bits: The number of data bits may be either seven or eight (software set by the user). The LSB bit is transmitted first and the MSB last.

An optional even or odd parity bit: If used, a parity bit is added after the last (MSB) data bit. For even parity, the parity bit is set such that the total number of 1s, including itself, is even. For odd parity, the total number of 1s is odd.

One or more stop bits: Each character is terminated with one or more stop bits. A stop bit is a logical 1, commonly referred to as a mark.

Data Transfer Rate

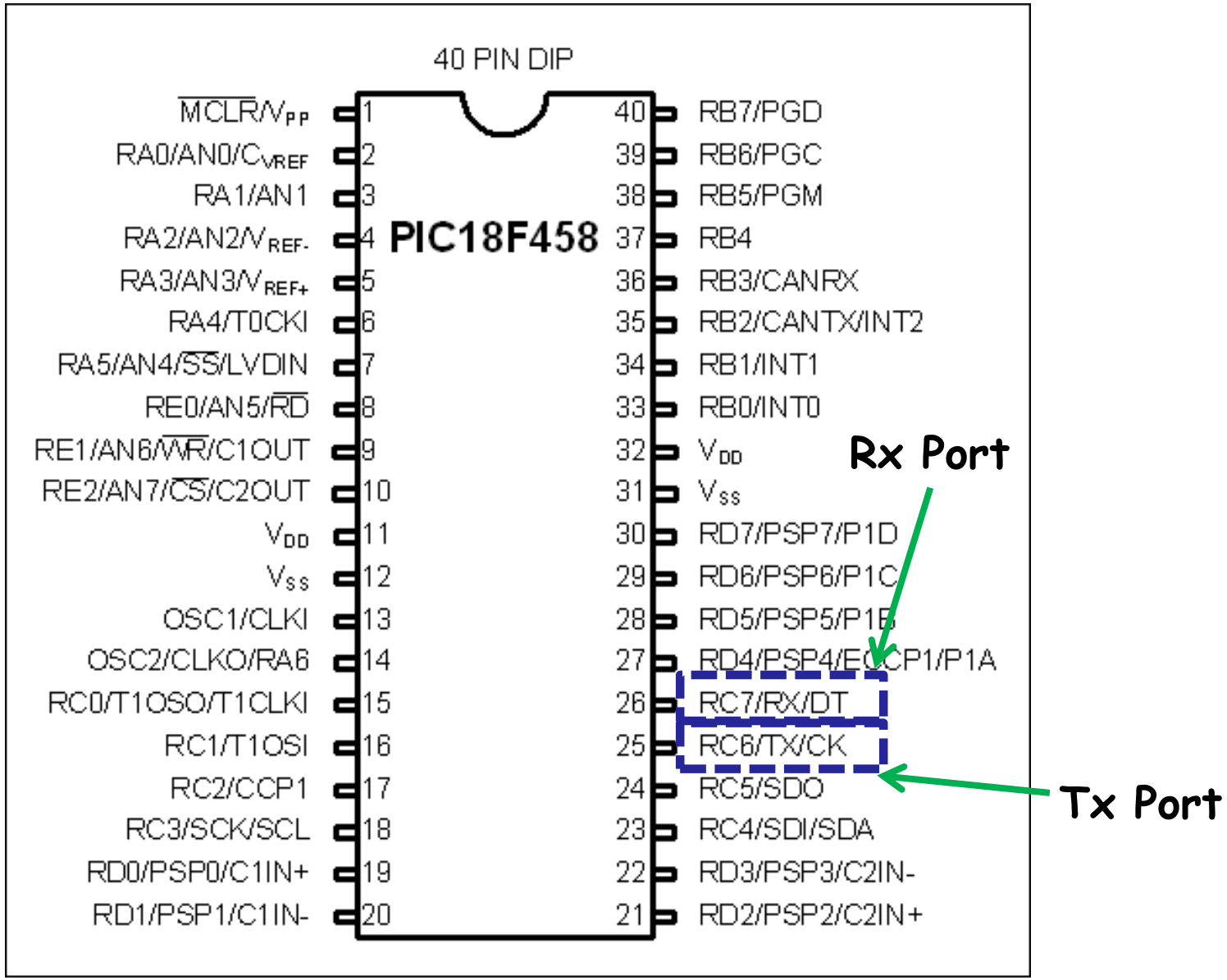
- rate of data transfer: *bps* (bits per second)
- another widely used terminology for bps is baud rate
- for asynchronous serial data communication, the baud rate is generally limited to ~~100,000~~ 200000 bps

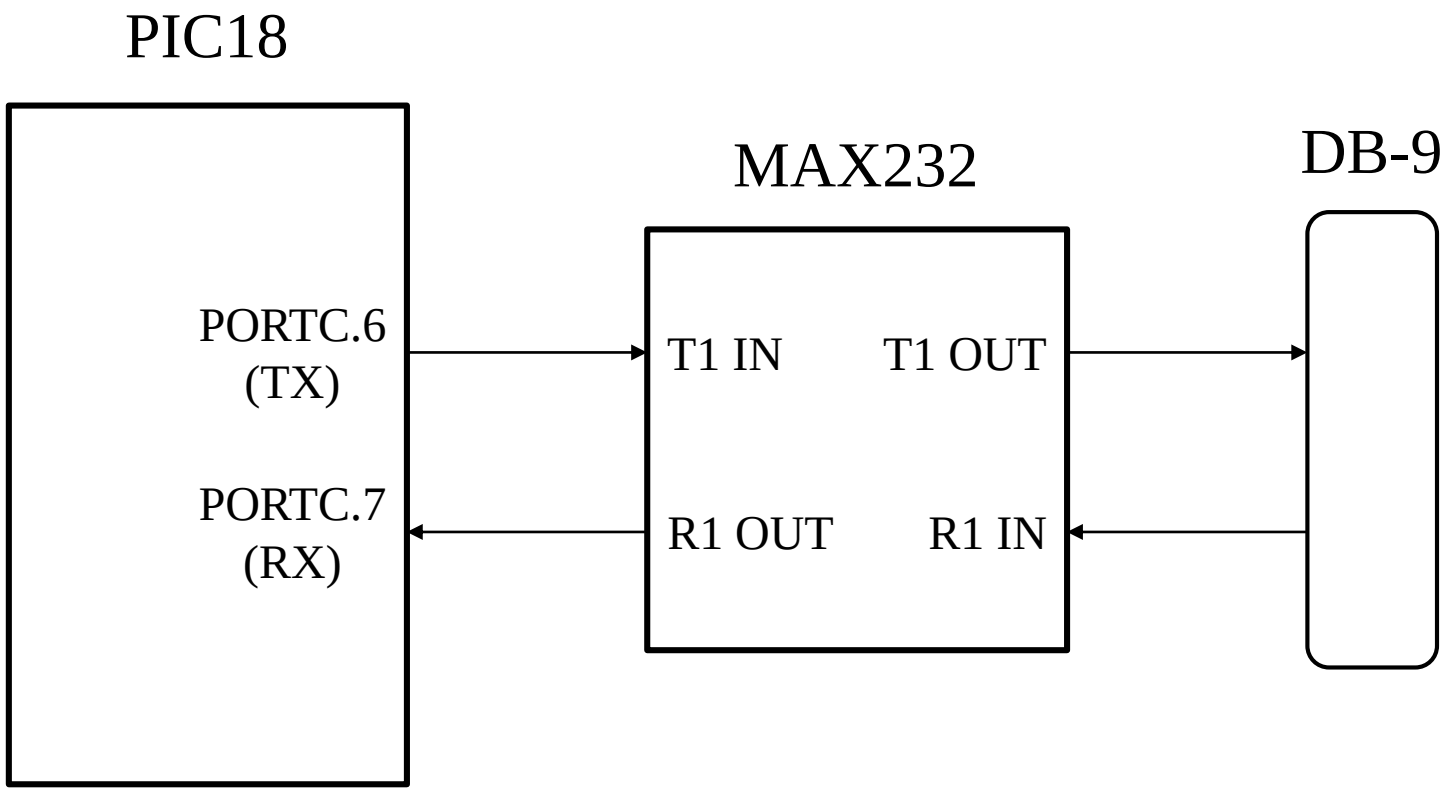
Data Transmission Errors
=Framing error - absence of the stop bits (due to synchronization problem, faulty transmission)
=Overrun error - one or more characters were received but not read from buffer
=Parity error

Interfacing standards

- in 1960, Electronic Industry Association (EIA) established the RS232 standard as a common interface for data communication equipment
- it was revised several times
- not compatible with TTL voltage level used by microprocessor – need voltage converter IC, e.g. MAX232
- 2 types of connector – DB-25, DB-9 (COM ports of computer)
- today, USB is more popular

logical 1 (mark): -3V to -25V
logical 0 (space) +3V to +25V


40 PIN DIP
PIC18F458
MCLR/VPP 1
RA0/AN0/CVREF 2
RA1/AN1 3
RA2/AN2/VREF- 4
RA3/AN3/VREF+ 5
RA4/T0CKI 6
RA5/AN4/SS/LVDIN 7
RE0/AN5/RD 8
RE1/AN6/WR/C1OUT 9
RE2/AN7/CS/C2OUT 10
VDD 11
VSS 12
OSC1/CLKI 13
OSC2/CLKO/RA6 14
RC0/T1OSO/T1CLKI 15
RC1/T1OSI 16
RC2/CCP1 17
RC3/SCK/SCL 18
RD0/PSP0/C1IN+ 19
RD1/PSP1/C1IN- 20
40 RB7/PGD
39 RB6/PGC
38 RB5/PGM
37 RB4
36 RB3/CANRX
35 RB2/CANTX/INT2
34 RB1/INT1
33 RB0/INT0
32 VDD
31 VSS
30 RD7/PSP7/P1D
29 RD6/PSP6/P1C
28 RD5/PSP5/P1B
27 RD4/PSP4/ECCP1/P1A
26 RC7/RX/DT
25 RC6/TX/CK
24 RC5/SDO
23 RC4/SDI/SDA
22 RD3/PSP3/C2IN-
21 RD2/PSP2/C2IN+
Rx Port
Tx Port


PIC18
MAX232
DB-9
PORTC.6
(TX)
PORTC.7
(RX)
T1 IN
T1 OUT
R1 OUT
R1 IN

Serial data transfer protocols

- USART (Universal Synchronous Asynchronous Receiver and Transmitter)
- SPI (Synchronous Peripheral Interface)
- I2C (InterIntegrated Circuit)
- CAN bus (Controller Area Network bus)

…

5.2 USART

- synchronous mode can be used to transfer data between PIC18 and external peripheral with half-duplex transmission, e.g. ADC
- asynchronous mode can be used to connect PIC18 to computer's COM port with full-duplex transmission
- 6 registers to control the operation:
  - SPBRG
  - TXREG
  - RCREG
  - TXSTA
  - RCSTA
  - PIR1

SPBRG Baud Rate Generate register

- The baud rate can be calculated using the following equations:

BRGH (in TXSTA)	Baud Rate
0 low speed	$f_{osc}/[64(X+1)]$
1 high speed	$f_{osc}/[16(X+1)]$

***X* is the 8-bit value loaded into SPBRG register**

f_{osc} = 10 MHz, BRGH = 0

x

Baud Rate	SPBRG (decimal)	SPBRG (hex)
38400	3	3
19200	7	7
9600	15	F
4800	32	20
2400	64	40
1200	129	81

f=20Mhz BRGH=1
Baud Rate=20e6/(16(1+129=9615
SPBRG=129

TXREG Transmit REGister

- the 8-bit data must be loaded into TXREG first (*TXREG can be accessed like other registers*)
- the data is then fetched into TSR (Transmit Shift Register)
- TSR adds start and stop bits
- the 10-bit data is transferred serially via the TX pin

Examples:

```
movlw     0x41
movwf     TXREG

movff     PORTB, TXREG
```

RCREG Receive REGister

- the 10-bit data is received serially via the RX pin
- eliminate start and stop bits
- pack bits into a byte
- the 8-bit data is loaded into RCREG (*RCREG can be accessed like other registers*)

Example:

```
movff          RCREG, PORTB
```

TXSTA Transmit STAtus register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care.

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9:** 9-Bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC:** EUSART Mode Select bit

1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB:** Send Break Character bit

Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed

Synchronous mode:
Don't care.

bit 2 **BRGH:** High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode.

bit 1 **TRMT:** Transmit Shift Register Status bit

1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data

Can be address/data bit or a parity bit.

Note 1: SREN/CREN overrides TXEN in Sync mode.

RCSTA Receive STAtus register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled (held in Reset)

bit 6 **RX9:** 9-Bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN:** Continuous Receive Enable bit

Asynchronous mode:

1 = Enables receiver

0 = Disables receiver

Synchronous mode:

1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)

0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit

Asynchronous mode 9-bit (RX9 = 1):

1 = Enables address detection, enables interrupt and loads the receive buffer when RSR<8> is set

0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

Asynchronous mode 9-bit (RX9 = 0):

Don't care.

bit 2 **FERR:** Framing Error bit

1 = Framing error (can be updated by reading RCREG register and receiving next valid byte)

0 = No framing error

bit 1 **OERR:** Overrun Error bit

1 = Overrun error (can be cleared by clearing bit CREN)

0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data

This can be an address/data bit or a parity bit and must be calculated by user firmware.

PIR1 Peripheral Interrupt Request register 1

---	--	RCIF	TXIF	--	---	--	

RCIF Receive interrupt flag bit
1 = The UART has received a byte of data and it is sitting in the RCREG register (receive buffer), waiting to be picked up.
Upon reading the RCREG register, the RCIF is cleared to allow the next byte to be received.
0 = The RCREG is empty.

TXIF Transmit interrupt flag bit
0 = The TXREG register is full.
1 = The TXREG (transmit buffer) register is empty.

The importance of TXIF: To transmit a byte of data, we write it into TXREG. Upon writing a byte into TXREG, the TXIF flag is cleared. When the entire byte is transmitted via the TX pin, the TXIF flag bit is raised to indicate that it is ready for the next byte. So, we must monitor this flag before we write a new byte into TXREG, otherwise, we wipe out the last byte before it is transmitted.

1. Before transmission, three bits must be specified:
   SPEN = 1 – to enable the serial port
   TXEN = 1 – to enable transmit
   SYNC = 0 – to select asynchronous mode
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 flag becomes 0.
5. A byte, framed between the start and stop bits, is transmitted out.
6. Once the stop bit has been sent out, TRMT becomes 1.

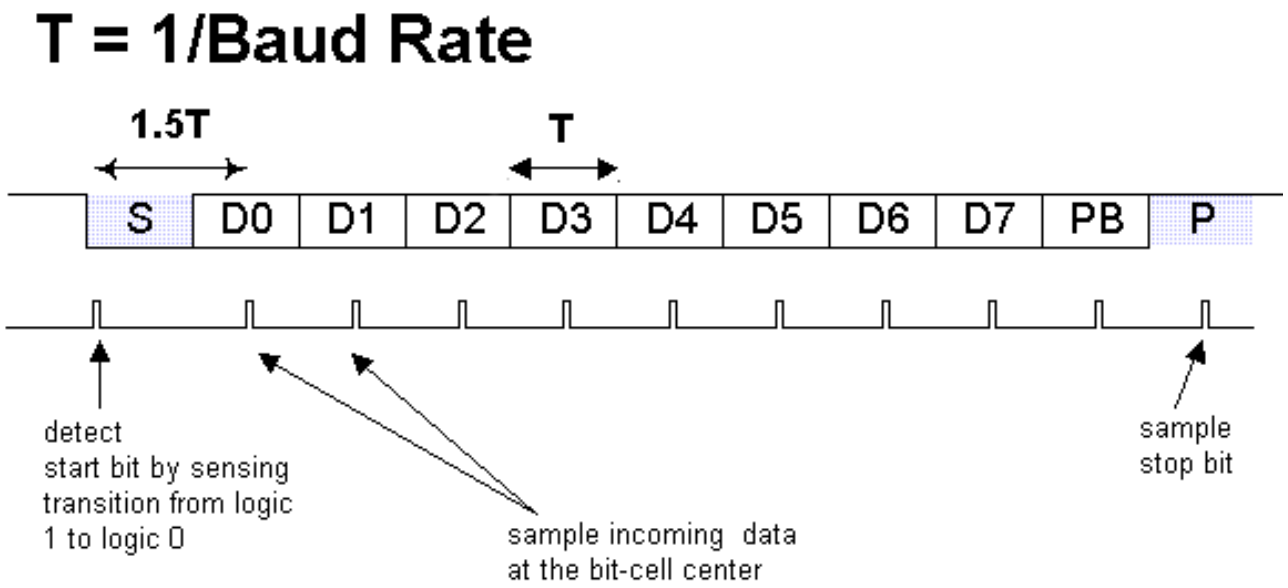
Setting up asynchronous transmit (4 MHz)

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

Over:    movlw   a'A'
         call    putChar
         bra     Over

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

Setting up asynchronous transmit (10 MHz)

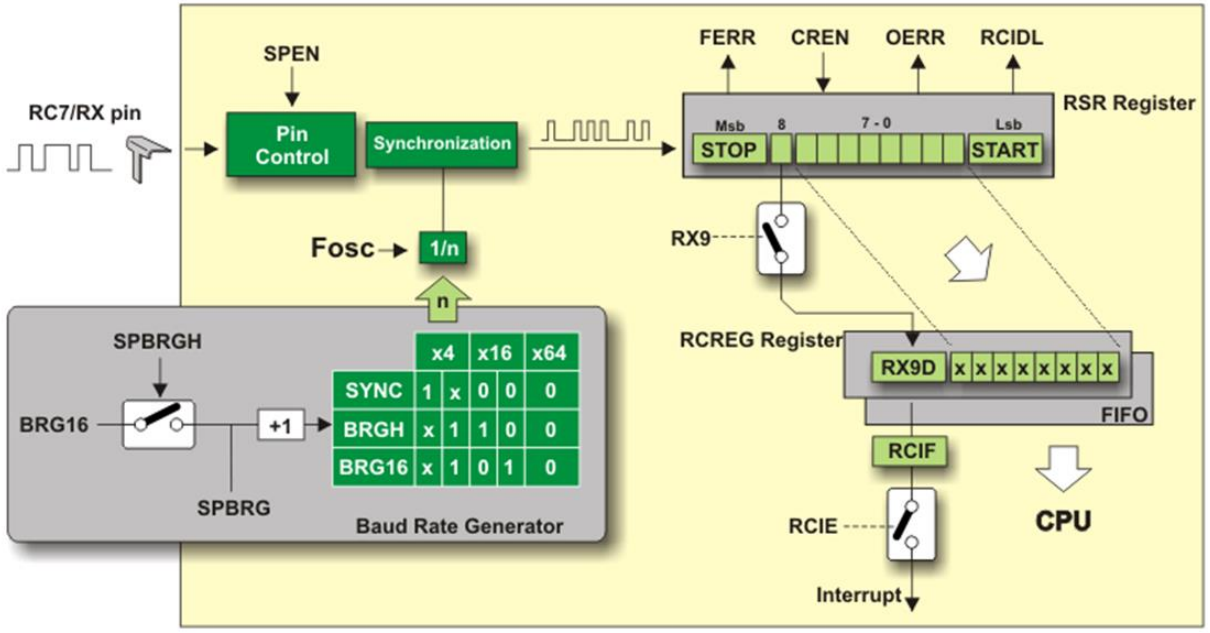
```
Main:     movlw   b'00100100'       ; TXEN = 1, SYNC = 0, BRGH = 1
          movwf   TXSTA
          movlw   b'10010000'       ; SPEN = 1
          movwf   RCSTA
          movlw   D'64'             ; set baud rate for 9600
          movwf   SPBRG

Over:     movlw   a'A'
          call    putChar
          bra     Over

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

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.

- 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.



This example receives bytes and output to PORTD (4 MHz)

```
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    d'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
```

In practice, we will not use polling to wait an input. We use the interrupt concept.

In software,

- Configure some registers to enable the UART RX interrupt in the beginning part of the program.
- Write an interrupt service routine to process the input byte.

So, in the normal condition, the system serves some other tasks. When an input byte is arrived, the hardware will generate an interrupt (for UART RX). The CPU will finish the current instruction. After identify the source of interrupts (using polling to identify, check IF bits of each possible source), the CPU can jump to the UART RX service routine.

Summary

- basics of serial communication
- USART