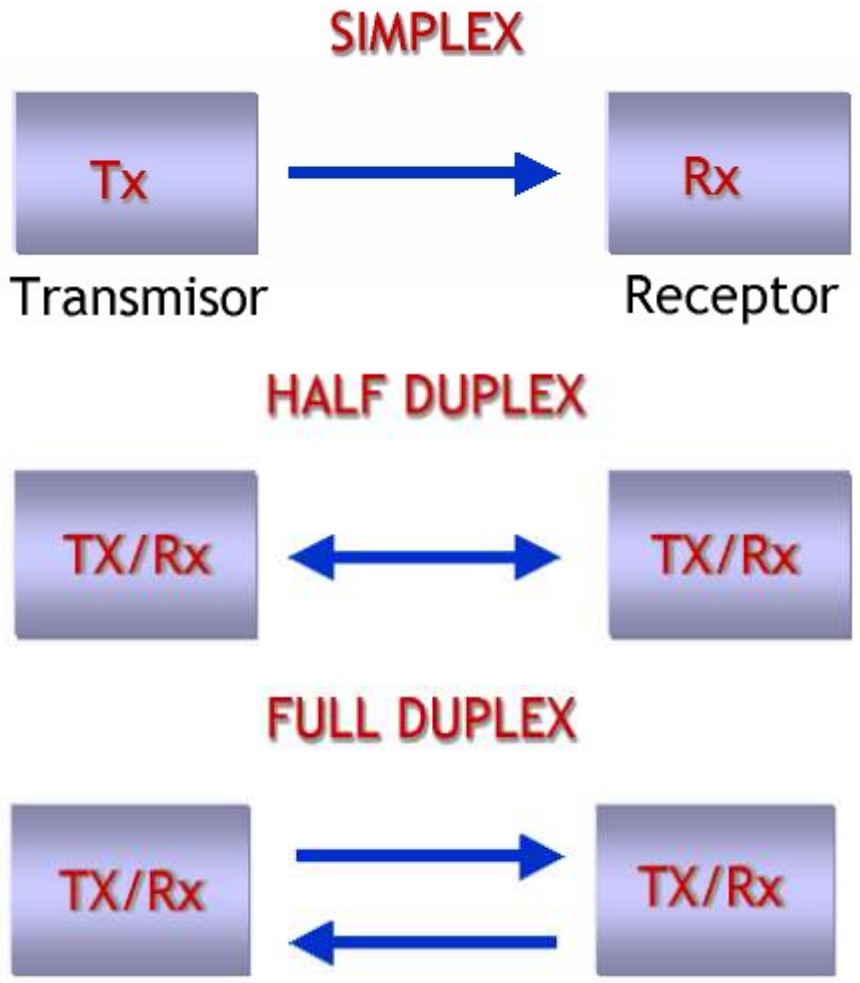
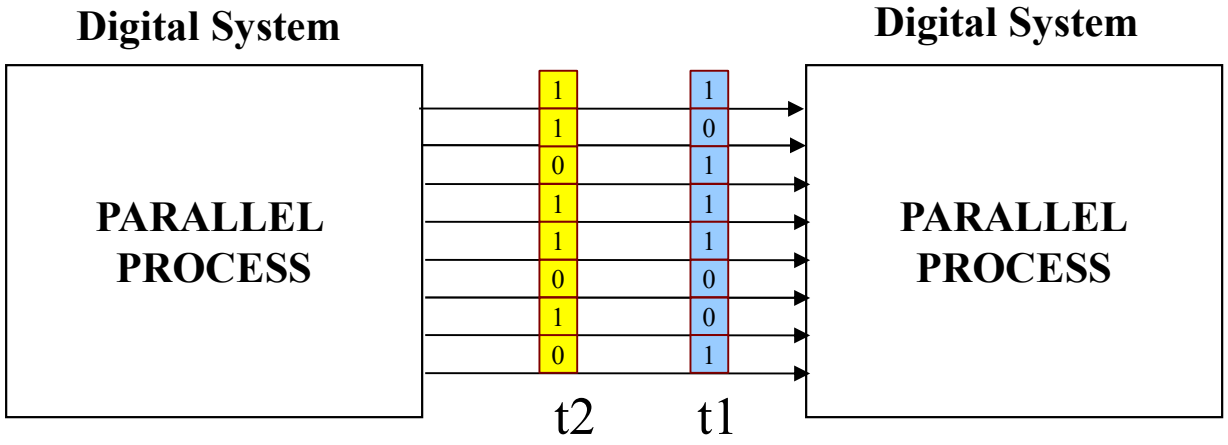
TECNOLÓGICO DE MONTERREY®

Subject 12
THE SERIAL PORT (USART)

Communication types

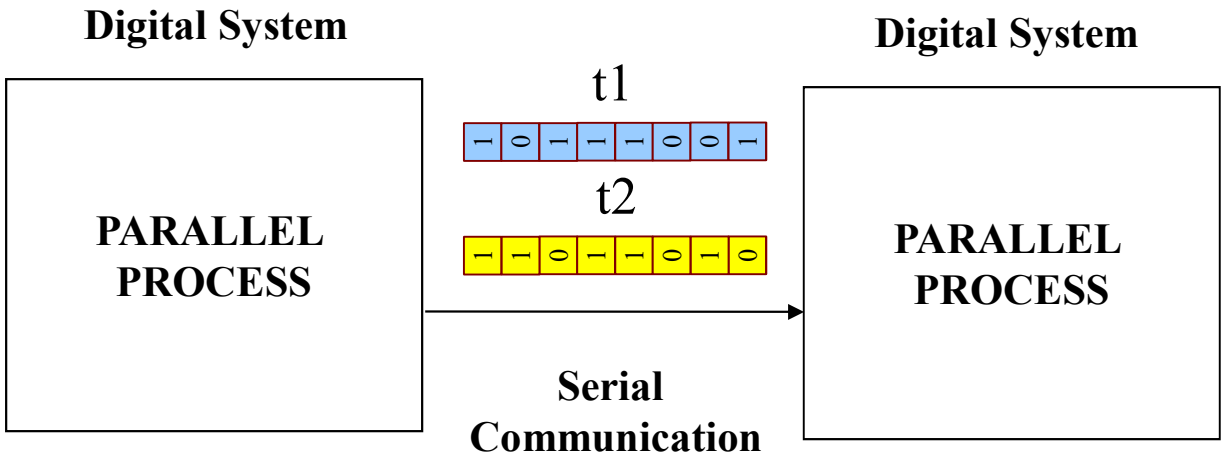


Parallel communication



- Pros:
  - The internal busses of the system be used directly with minimum or null additional processing. .
  - I short distances very high speeds can be obtained
- Cons:
  - Cables and connectors are more expensive, less distance and it gets very complex for multi-pint communication

Serial communication

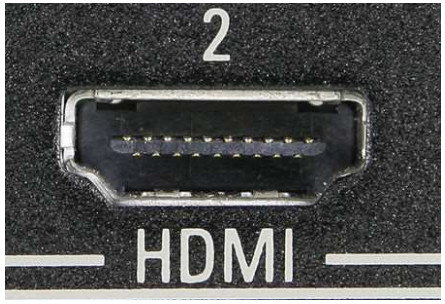


- Pros:
  - It requires less electrical connections and we can transfer the information using conventional channels like radio, light, phones etc..
- Cons:
  - Lower speed and increase of hardware complexity .

Serial Communication

Serial ports RS232
RS485
RS422

Ethernet

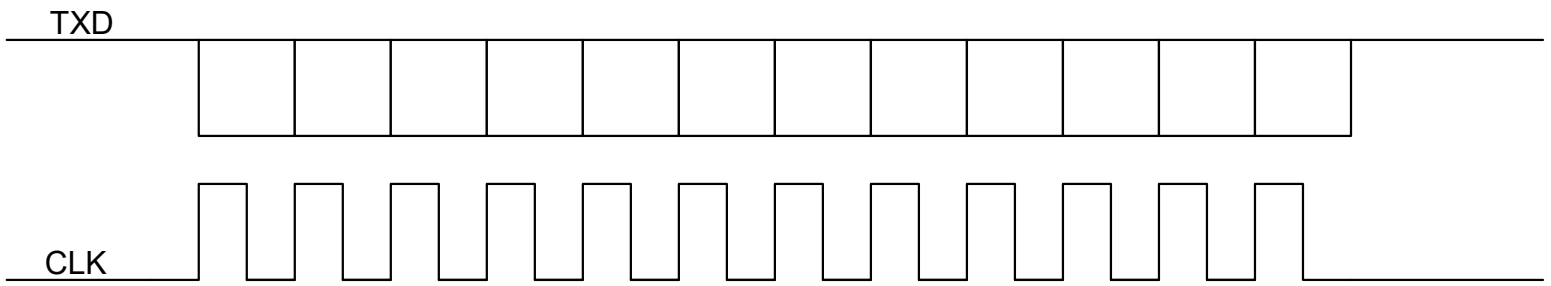


Firewire

USB

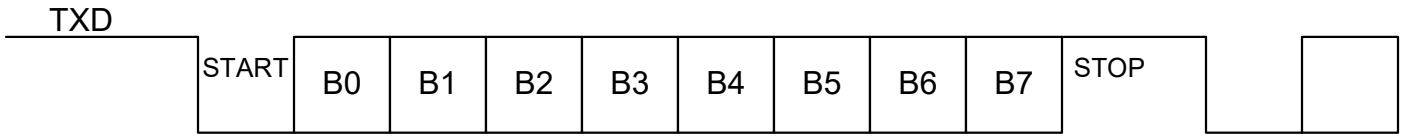
Types of serial communication

- Synchronous:
  - A clock signal is sent, this is independent or embedded in the data signal
  - Is generally used to transfer big blocks of data.



Types of serial communication

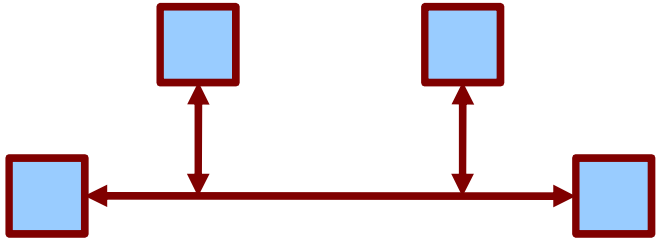
- Asynchronous:
  - Does not require a clock signal.
  - The data is subdivided in smaller packs (packets) of information, usually 8 bits.
  - In order to communicate systems, both must be configured at the same data rate, that is, the duration of the bit is know by both systems



High level application

- Point to point communication
  - RS232C
  - USB
  - HDMI

- Multipoint communication
  - RS485
  - CAN
  - LIN
  - Ethernet



EUSART in PIC18

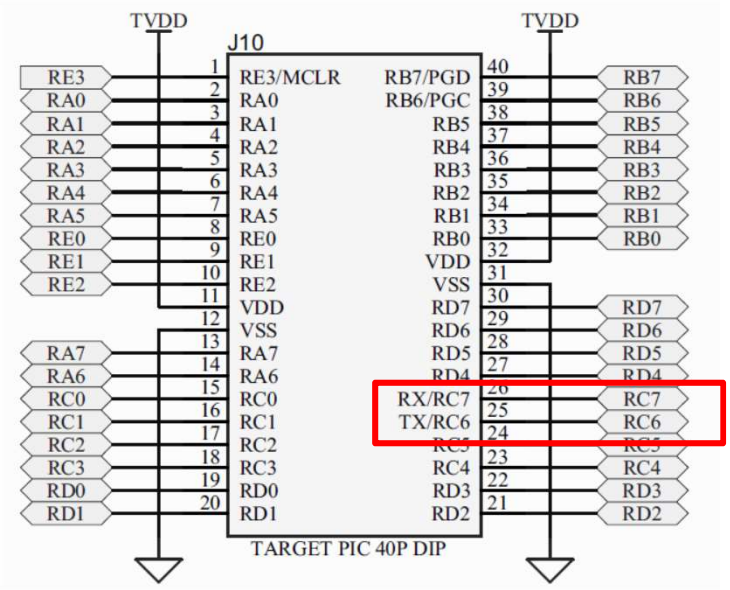
EUSART

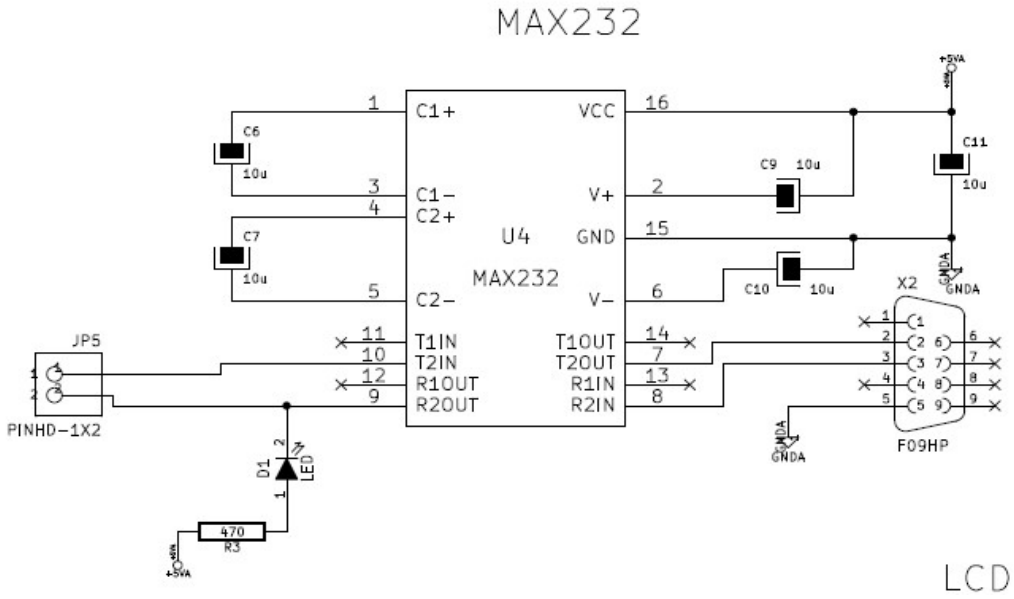
- USART (universal synchronous/asynchronous receiver/transmitter)
- The PIC18F45K50 supports 1USART'

- Full-duplex asynchronous transmit and receive
- Two-character input buffer
- One-character output buffer
- Programmable 8-bit or 9-bit character length
- Address detection in 9-bit mode
- Input buffer overrun error detection
- Received character framing error detection
- Half-duplex synchronous master
- Half-duplex synchronous slave
- Programmable clock and data polarity

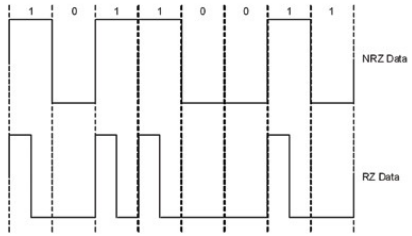
The EUSART module implements the following additional features, making it ideally suited for use in Local Interconnect Network (LIN) bus systems:

- Automatic detection and calibration of the baud rate
- Wake-up on Break reception
- 13-bit Break character transmit

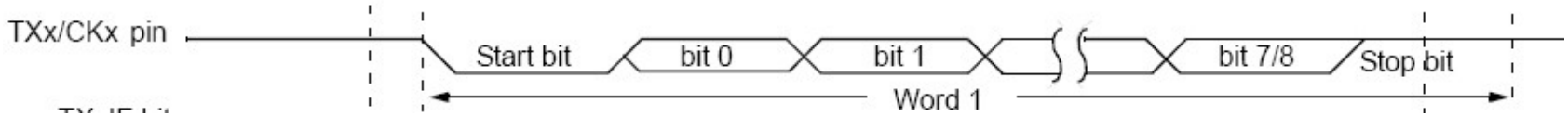

TVDD
TVDD
J10
RE3
RA0
RA1
RA2
RA3
RA4
RA5
RE0
RE1
RE2
RA7
RA6
RC0
RC1
RC2
RC3
RD0
RD1
RE3/MCLR
RA0
RA1
RA2
RA3
RA4
RA5
RE0
RE1
RE2
VDD
VSS
RA7
RA6
RC0
RC1
RC2
RC3
RD0
RD1
RB7/PGD
RB6/PGC
RB5
RB4
RB3
RB2
RB1
RB0
VDD
VSS
RD7
RD6
RD5
RD4
RX/RC7
TX/RC6
RC4
RD3
RD2
RB7
RB6
RB5
RB4
RB3
RB2
RB1
RB0
RD7
RD6
RD5
RD4
RC7
RC6
RC5
RC4
RD3
RD2
TARGET PIC 40P DIP


MAX232
C1+
C1−
C2+
C2−
VCC
V+
GND
V−
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
U4
MAX232
C6
C7
C9
C10
C11
10u
JP5
PINHD-1X2
D1
LED
470
R3
X2
F09HP
+5VA
GNDA
LCD

Asynchronous mode



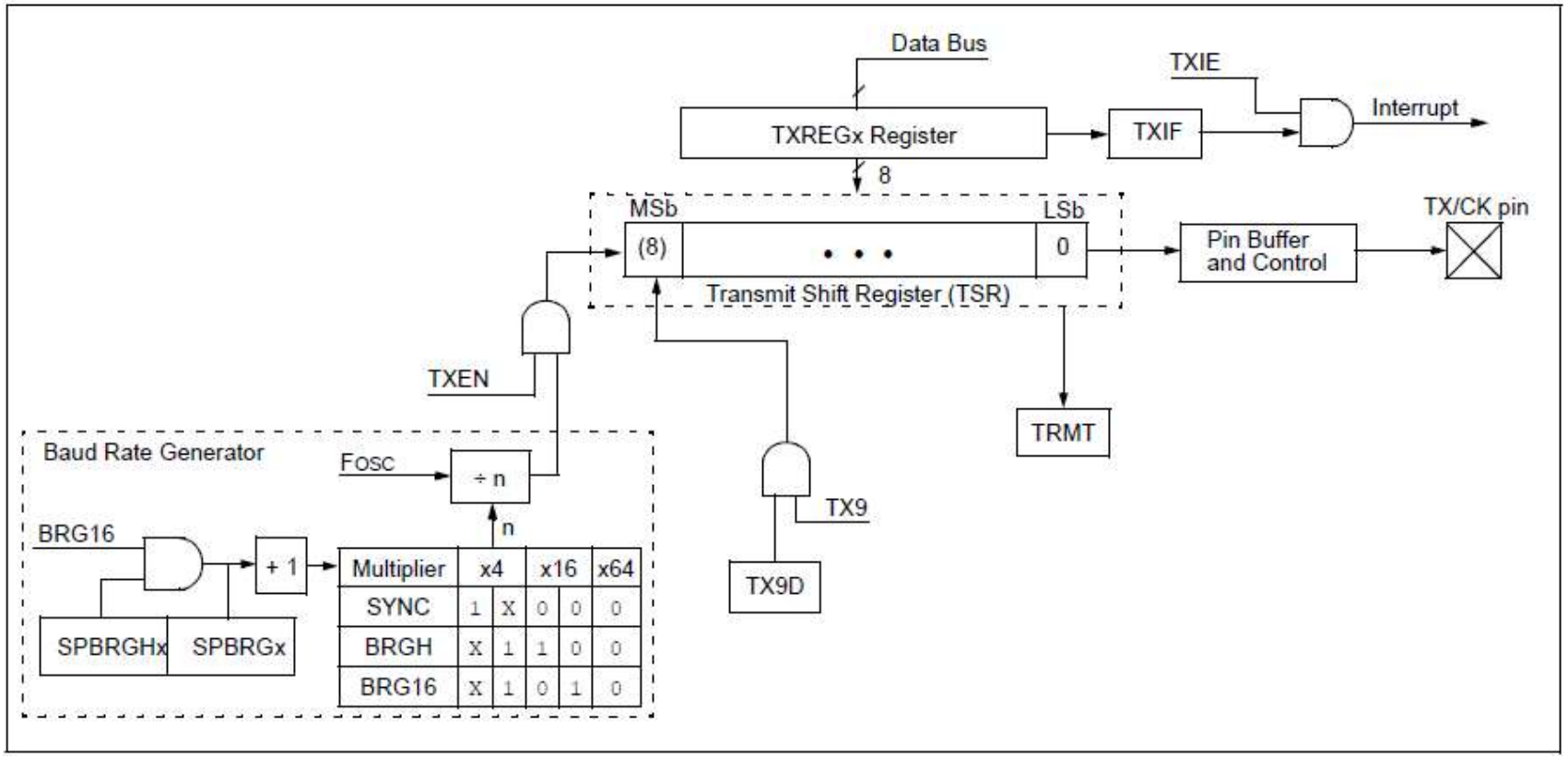
- It transmits and receives using NRZ
- 0V is a logic 0 (mark) 5V is a logic one (space)
- One start bit (0V), 8 o 9 bits of data y one or more stop bits (stop)



Asynchronous mode

- Each bit lasts 1/Baudrate
- The Baudrate is the data speed (ej. 1200Bps, 9600bps etc..)
- The least significant bit is transmitted first
- The most common scheme is to transmit 8 bits of data

USART transmission (TX)



Transmission

REGISTER 17-1: TXSTAx: TRANSMIT STATUS AND CONTROL 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

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

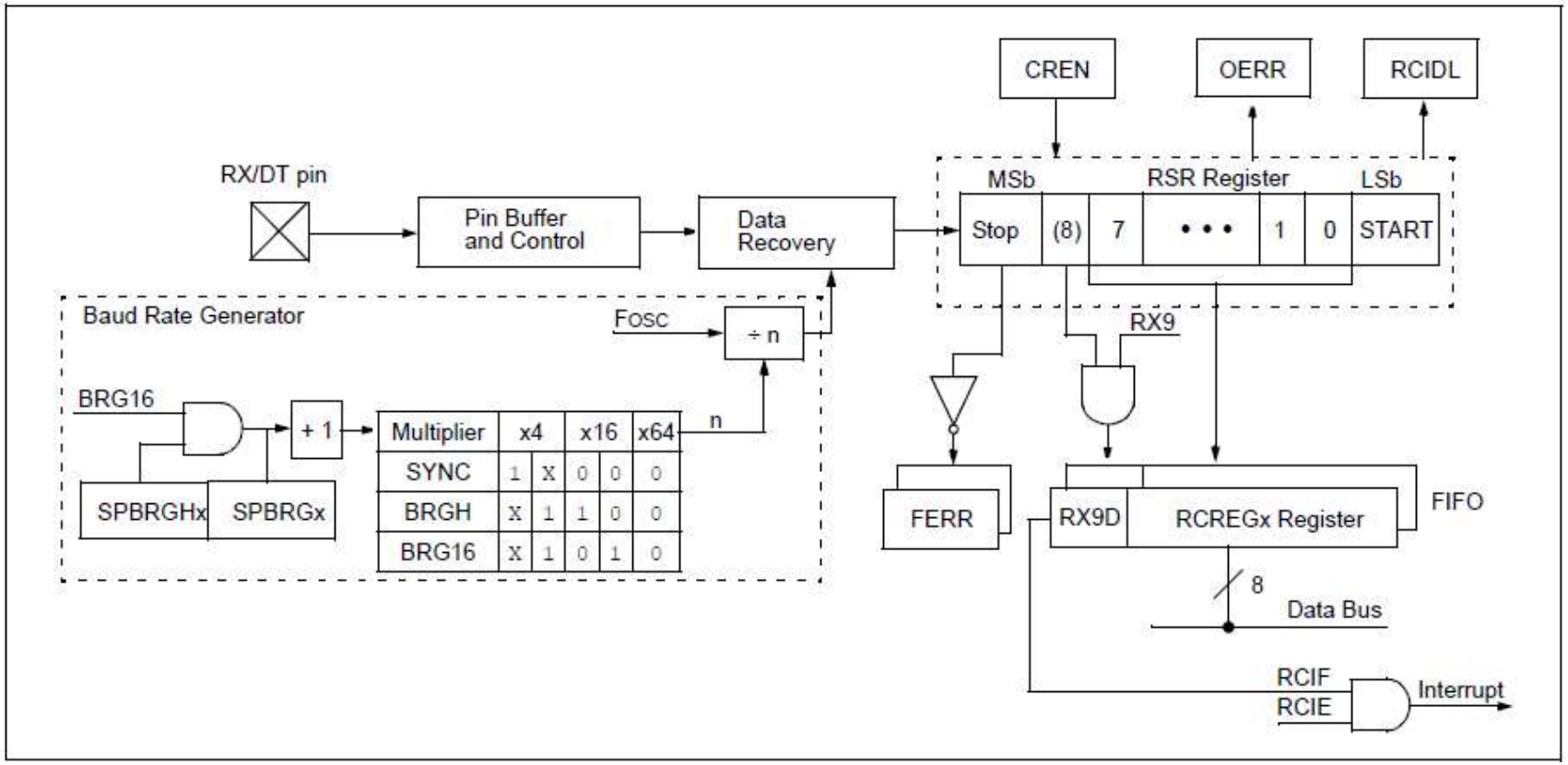
bit 3 **SENDB:** Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care

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:** Ninth bit of Transmit Data
Can be address/data bit or a parity bit.

USART reception (RX)



Reception

REGISTER 17-2: RCSTAx: RECEIVE STATUS AND CONTROL REGISTER

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

bit 7 **SPEN:** Serial Port Enable bit
- 1 = Serial port enabled (configures RXx/DTx and TXx/CKx 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. enable interrupt and load 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 8-bit (RX9 = 0):

Don't care

bit 2 **FERR:** Framing Error bit
- 1 = Framing error (can be updated by reading RCREGx register and receive next valid byte)
- 0 = No framing error

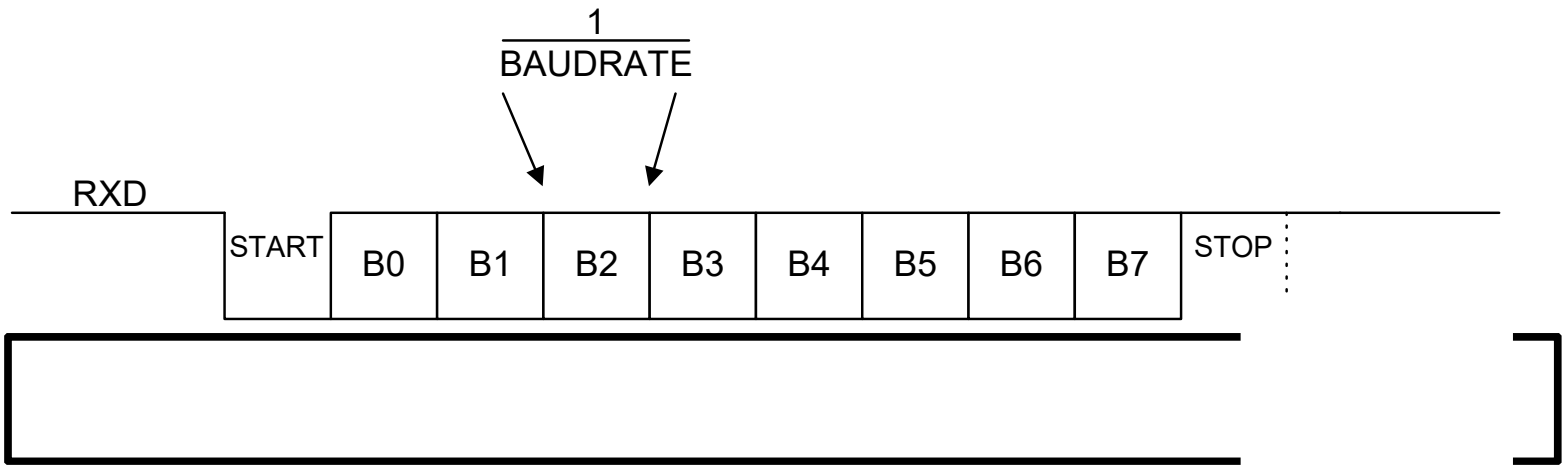
bit 1 **OERR:** Overrun Error bit
- 1 = Overrun error (can be cleared by clearing bit CREN)
- 0 = No overrun error

bit 0 **RX9D:** Ninth bit of Received Data

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

"Baudrate" generator

- It defines the speed of the data transfer (bits per second or baudrate"



The "Baudrate" generator

- Its an independent timer that can be configured of 8 or 16 bits.
- The registers SPBRGHx:SPBRGx define the transfer frequency

TABLE 17-3: BAUD RATE FORMULAS

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	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGHx, SPBRGx register pair.

Example

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

$$Desired\ Baud\ Rate = \frac{FOSC}{64([SPBRGHx:SPBRGx]+1)}$$

Solving for SPBRGHx:SPBRGx:

$$X = \frac{\frac{FOSC}{Desired\ Baud\ Rate}}{64} - 1$$

$$= \frac{\frac{16000000}{9600}}{64} - 1$$

$$= [25.042] = 25$$

$$Calculated\ Baud\ Rate = \frac{16000000}{64(25+1)}$$

$$= 9615$$

$$Error = \frac{Calc.\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate}$$

$$= \frac{(9615-9600)}{9600} = 0.16\%$$

Baudrates

BAUD RATE	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRxG value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	—	—	—	—	—	—	—	—	—
1200	—	—	—	1200	0.00	239	1202	0.16	207	1200	0.00	143
2400	—	—	—	2400	0.00	119	2404	0.16	103	2400	0.00	71
9600	9615	0.16	103	9600	0.00	29	9615	0.16	25	9600	0.00	17
10417	10417	0.00	95	10286	-1.26	27	10417	0.00	23	10165	-2.42	16
19.2k	19.23k	0.16	51	19.20k	0.00	14	19.23k	0.16	12	19.20k	0.00	8
57.6k	58.82k	2.12	16	57.60k	0.00	7	—	—	—	57.60k	0.00	2
115.2k	111.11k	-3.55	8	—	—	—	—	—	—	—	—	—

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 0											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	—	—	—	—	—	—	—	—	—
1200	—	—	—	—	—	—	—	—	—	—	—	—
2400	—	—	—	—	—	—	—	—	—	—	—	—
9600	—	—	—	9600	0.00	119	9615	0.16	103	9600	0.00	71
10417	—	—	—	10378	-0.37	110	10417	0.00	95	10473	0.53	65
19.2k	19.23k	0.16	207	19.20k	0.00	59	19.23k	0.16	51	19.20k	0.00	35
57.6k	57.97k	0.64	68	57.60k	0.00	19	58.82k	2.12	16	57.60k	0.00	11
115.2k	114.29k	-0.79	34	115.2k	0.00	9	111.1k	-3.55	8	115.2k	0.00	5

Baudrates

BAUD RATE	SYNC = 0, BRGH = 0, BRG16 = 1											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRGHx: SPBRGx (decimal)	Actual Rate	% Error	SPBRGHx: SPBRGx (decimal)	Actual Rate	% Error	SPBRGHx :SPBRGx (decimal)	Actual Rate	% Error	SPBRGHx: SPBRGx (decimal)
300	300.0	0.00	13332	300.0	0.00	3839	300.03	0.01	3332	300.0	0.00	2303
1200	1200.1	0.01	3332	1200	0.00	959	1200.5	0.04	832	1200	0.00	575
2400	2399	-0.02	1666	2400	0.00	479	2398	-0.08	416	2400	0.00	287
9600	9592	-0.08	416	9600	0.00	119	9615	0.16	103	9600	0.00	71
10417	10417	0.00	383	10378	-0.37	110	10417	0.00	95	10473	0.53	65
19.2k	19.23k	0.16	207	19.20k	0.00	59	19.23k	0.16	51	19.20k	0.00	35
57.6k	57.97k	0.64	68	57.60k	0.00	19	58.82k	2.12	16	57.60k	0.00	11
115.2k	114.29k	-0.79	34	115.2k	0.00	9	111.11k	-3.55	8	115.2k	0.00	5

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRGHx: SPBRGx (decimal)	Actual Rate	% Error	SPBRGHx: SPBRGx (decimal)	Actual Rate	% Error	SPBRGHx :SPBRGx (decimal)	Actual Rate	% Error	SPBRGHx: SPBRGx (decimal)
300	300	0.00	53332	300.0	0.00	15359	300.0	0.00	13332	300.0	0.00	9215
1200	1200	0.00	13332	1200	0.00	3839	1200.1	0.01	3332	1200	0.00	2303
2400	2400	0.00	6666	2400	0.00	1919	2399.5	-0.02	1666	2400	0.00	1151
9600	9598.1	-0.02	1666	9600	0.00	479	9592	-0.08	416	9600	0.00	287
10417	10417	0.00	1535	10425	0.08	441	10417	0.00	383	10433	0.16	264
19.2k	19.21k	0.04	832	19.20k	0.00	239	19.23k	0.16	207	19.20k	0.00	143
57.6k	57.55k	-0.08	277	57.60k	0.00	79	57.97k	0.64	68	57.60k	0.00	47
115.2k	115.11k	-0.08	138	115.2k	0.00	39	114.29k	-0.79	34	115.2k	0.00	23

REGISTER 17-3: BAUDCONx: BAUD RATE CONTROL REGISTER

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

bit 7 **ABDOVF:** Auto-Baud Detect Overflow bit
Asynchronous mode:
1 = Auto-baud timer overflowed
0 = Auto-baud timer did not overflow
Synchronous mode:
Don't care

bit 6 **RCIDL**: Receive Idle Flag bit
Asynchronous mode:
1 = Receiver is Idle
0 = Start bit has been detected and the receiver is active
Synchronous mode:
Don't care

bit 5 **DTRXP**: Data/Receive Polarity Select bit
Asynchronous mode:
1 = Receive data (RXx) is inverted (active-low)
0 = Receive data (RXx) is not inverted (active-high)
Synchronous mode:
1 = Data (DTx) is inverted (active-low)
0 = Data (DTx) is not inverted (active-high)

bit 4 **CKTXP**: Clock/Transmit Polarity Select bit
Asynchronous mode:
1 = Idle state for transmit (TXx) is low
0 = Idle state for transmit (TXx) is high
Synchronous mode:
1 = Data changes on the falling edge of the clock and is sampled on the rising edge of the clock
0 = Data changes on the rising edge of the clock and is sampled on the falling edge of the clock

bit 3 **BRG16:** 16-bit Baud Rate Generator bit
1 = 16-bit Baud Rate Generator is used (SPBRGHx:SPBRGx)
0 = 8-bit Baud Rate Generator is used (SPBRGx)

bit 2 **Unimplemented:** Read as '0'

bit 1 **WUE:** Wake-up Enable bit
Asynchronous mode:
1 = Receiver is waiting for a falling edge. No character will be received but RCxIF will be set on the falling edge. WUE will automatically clear on the rising edge.
0 = Receiver is operating normally
Synchronous mode:
Don't care

bit 0 **ABDEN**: Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Auto-Baud Detect mode is enabled (clears when auto-baud is complete)
0 = Auto-Baud Detect mode is disabled
Synchronous mode:
Don't care

Example

Write a program that writes the word “Hello world !” when the program starts and when you transmit a character to the microcontroller, and returns the next character in the ASCII set. That if an “a” is received, the microcontroller must return “b”.

Use a 9600 baudrate

Example

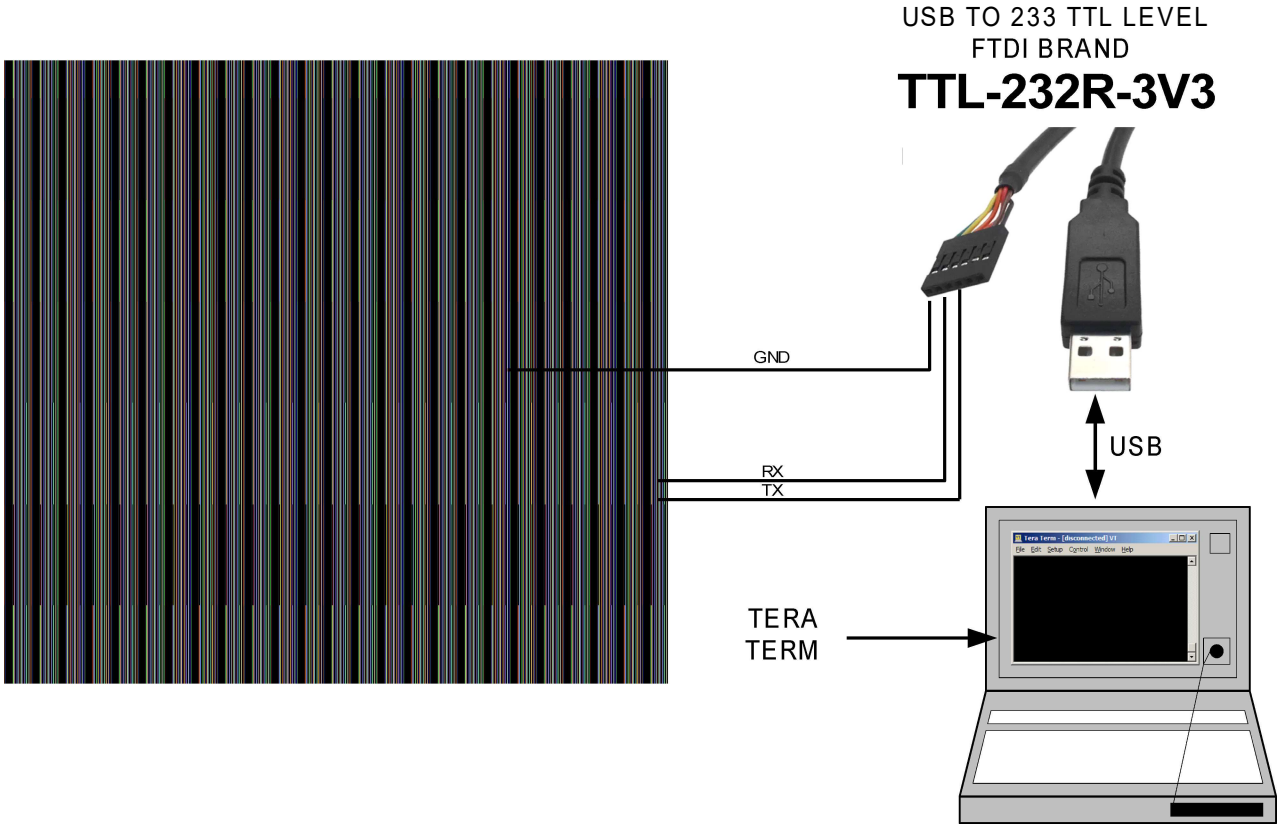
For the implementation of the example, we will use:

1) Curiosity board

2) A RS232 to USB converte (signal levels 3.3V)

3) A computer running a Terminal program like Tera Term or PuTTY

Example


USB TO 233 TTL LEVEL
FTDI BRAND
TTL-232R-3V3
GND
RX
TX
USB
TERA
TERM

Transmission setup

REGISTER 16-1: TXSTAX: TRANSMIT STATUS AND CONTROL 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 **0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	bit

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:** Ninth bit of Transmit Data
Can be address/data bit or a parity bit.

Reception setup

REGISTER 16-2: RCSTAX: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7 **1**	**0**	**0**	**1**	**0**	**0**	**0**	**0** bit 0

bit 7 **SPEN:** Serial Port Enable bit

→ 1 = Serial port enabled (configures RXx/DTx and TXx/CKx 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. enable interrupt and load 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 8-bit (RX9 = 0):

Don't care

bit 2 **FERR:** Framing Error bit

1 = Framing error (can be updated by reading RCREGx register and receive 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:** Ninth bit of Received Data

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

Baudrate setup

BAUD RATE	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRxG value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	—	—	—	—	—	—	—	—	—
1200	—	—	—	1200	0.00	239	1202	0.16	207	1200	0.00	143
2400	—	—	—	2400	0.00	119	2404	0.16	103	2400	0.00	71
9600	9615	0.16	103	9600	0.00	29	9615	0.16	25	9600	0.00	17
10417	10417	0.00	95	10286	-1.26	27	10417	0.00	23	10165	-2.42	16
19.2k	19.23k	0.16	51	19.20k	0.00	14	19.23k	0.16	12	19.20k	0.00	8
57.6k	58.82k	2.12	16	57.60k	0.00	7	—	—	—	57.60k	0.00	2
115.2k	111.11k	-3.55	8	—	—	—	—	—	—	—	—	—

REGISTER 16-3: BAUDCONX: BAUD RATE CONTROL REGISTER

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	DTRXP	CKTXP	BRG16	—	WUE	ABDEN
bit 7 0	0	0	0	0		0	0 bit 0

bit 7 **ABDOVF:** Auto-Baud Detect Overflow bit
Asynchronous mode:
1 = Auto-baud timer overflowed
0 = Auto-baud timer did not overflow
Synchronous mode:
Don't care

bit 6 **RCIDL:** Receive Idle Flag bit
Asynchronous mode:
1 = Receiver is Idle
0 = Start bit has been detected and the receiver is active
Synchronous mode:
Don't care

bit 5 **DTRXP:** Data/Receive Polarity Select bit
Asynchronous mode:
1 = Receive data (RXx) is inverted (active-low)
0 = Receive data (RXx) is not inverted (active-high)
Synchronous mode:
1 = Data (DTx) is inverted (active-low)
0 = Data (DTx) is not inverted (active-high)

bit 4 **CKTXP:** Clock/Transmit Polarity Select bit
Asynchronous mode:
1 = Idle state for transmit (TXx) is low
0 = Idle state for transmit (TXx) is high
Synchronous mode:
1 = Data changes on the falling edge of the clock and is sampled on the rising edge of the clock
0 = Data changes on the rising edge of the clock and is sampled on the falling edge of the clock

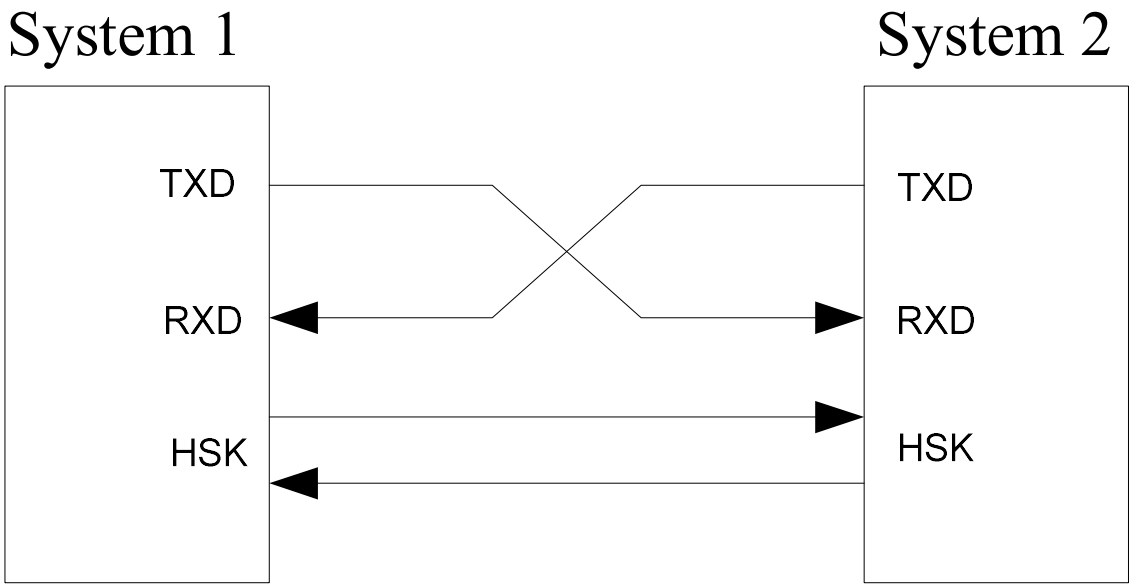
bit 3 **BRG16:** 16-bit Baud Rate Generator bit
1 = 16-bit Baud Rate Generator is used (SPBRGHx:SPBRGx)
0 = 8-bit Baud Rate Generator is used (SPBRGx)

bit 2 **Unimplemented:** Read as '0'

bit 1 **WUE:** Wake-up Enable bit
Asynchronous mode:
1 = Receiver is waiting for a falling edge. No character will be received but RCxIF will be set on the falling edge. WUE will automatically clear on the rising edge.
0 = Receiver is operating normally
Synchronous mode:
Don't care

bit 0 **ABDEN:** Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Auto-Baud Detect mode is enabled (clears when auto-baud is complete)
0 = Auto-Baud Detect mode is disabled
Synchronous mode:
Don't care

Implementación



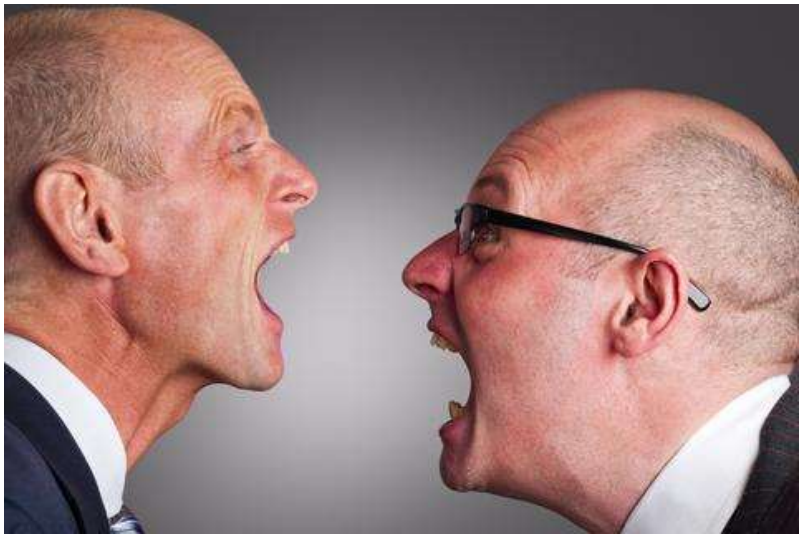
```
void int_oscillator(void);
void put_caracter(char);
void put_string(char *);
#define _XTAL_FREQ 16000000          //For the delay function
main(){
char mychar;
char message[16] = {"!Hello world!\n"};
    int_oscillator();
//Configure TX (RC6) as output y RX (RC7) as digital imput
    TRISCbits.RC7 = 1;       //RXD (RC7) input
    ANSELCbits.ANSC7 = 0;    //digital
    TRISCbits.RC6 = 0;       //TXD (RC6) output
    ANSELCbits.ANSC6 = 0;    //digital
//UART 1 configuration
   TXSTA1 = 0b00100000;     //8 bits asyncrhonous, Baud = low
   RCSTA1 = 0b10010000;     //8 bits reception eneabled
   BAUDCON1 = 0b00000000;   //Normal mode
   SPBRG1 = 25;             //9600bps Fosc = 16MHZ
   __delay_ms(200);         //Delay for baudrate generator stabilization
   //Display message "Hello world"
    put_string(&message);
    while(1){                       //Infinite loop
        //Wait to receive ca character
        while(!PIR1bits.RC1IF);          //Byte received ?
                PIR1bits.RC1IF = 0;      //Clear flag
                mychar = RCREG1;         //Read char
                mychar++;                //Next in the ASCII table
                put_caracter('_');       //Underscore as separator
                put_caracter(mychar);    //Transmit
    }
} //From main
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that outputs a caracter
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void put_caracter( char caracter)
{   while(!TXSTA1bits.TRMT);    //Termino de transmision
    TXREG1 = caracter;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that outputs a string using pointers
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void put_string(char *char_input)
{
    while(*char_input != 0x00){
        put_caracter(*char_input);
        char_input++;
    };

}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the internal oscillator at 16Mhz
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void int_oscillator(void){
    OSCCON = 0b01111110;     //Set the board oscillator to 16Mhz
}
```

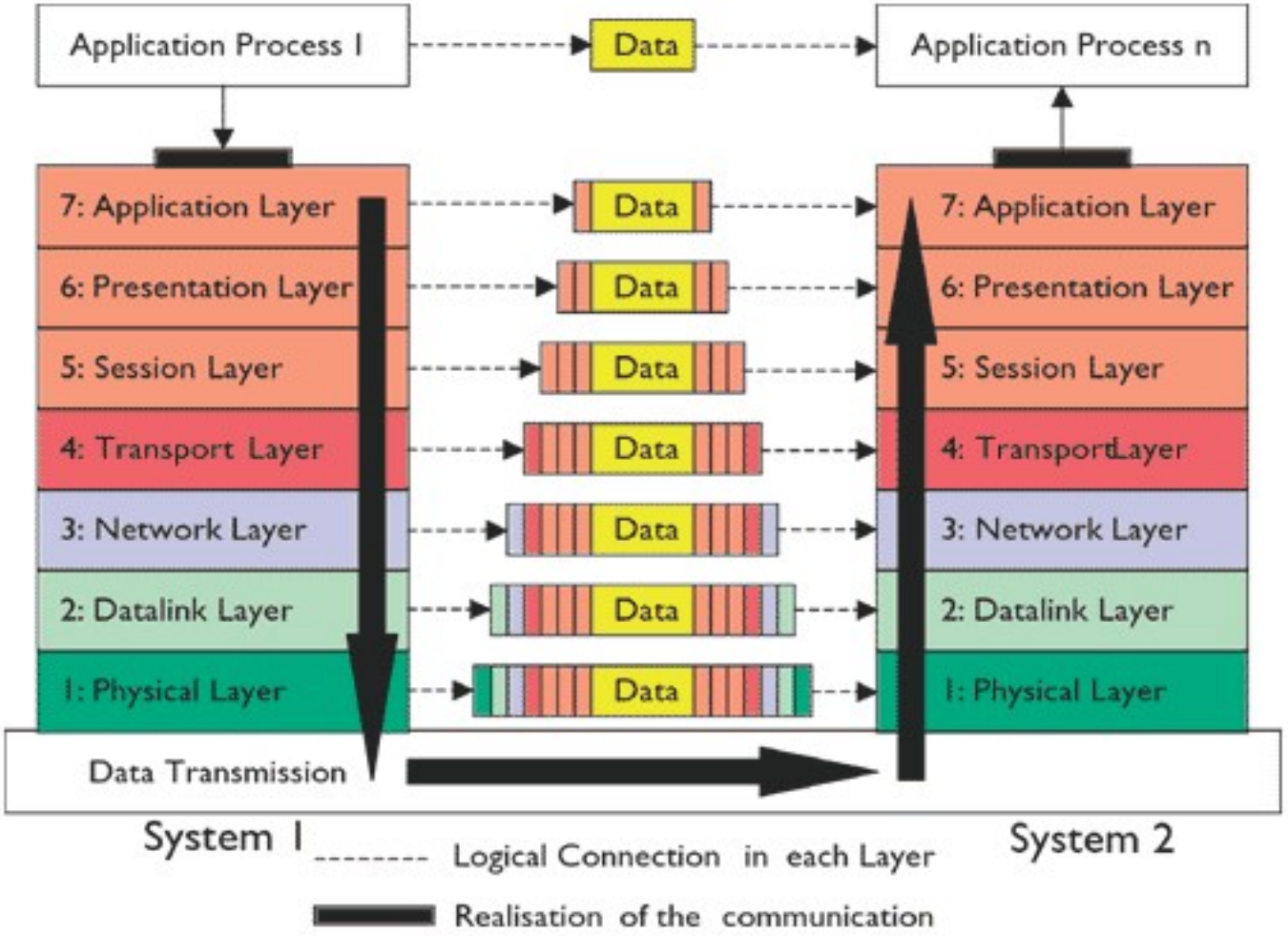
Protocols



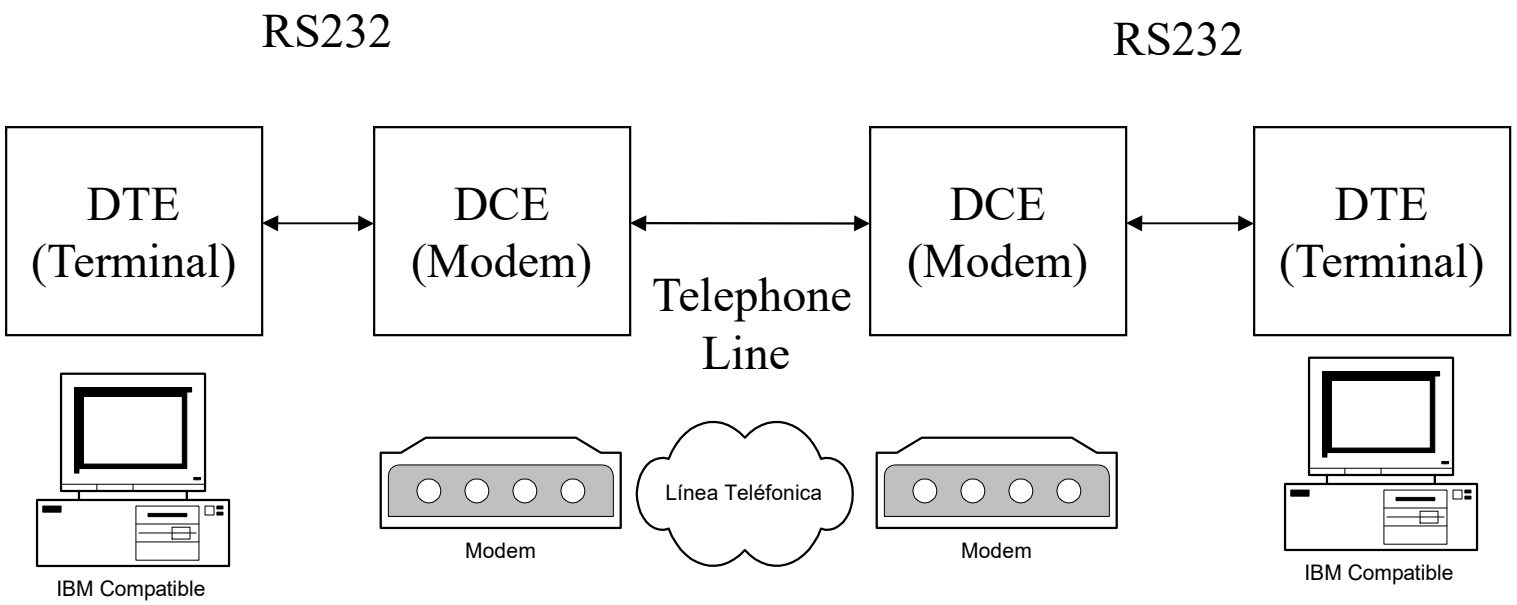
What is a protocol?

- A protocol is a set of rules and conventions accepted by a particular interest group to communicate between the individuals

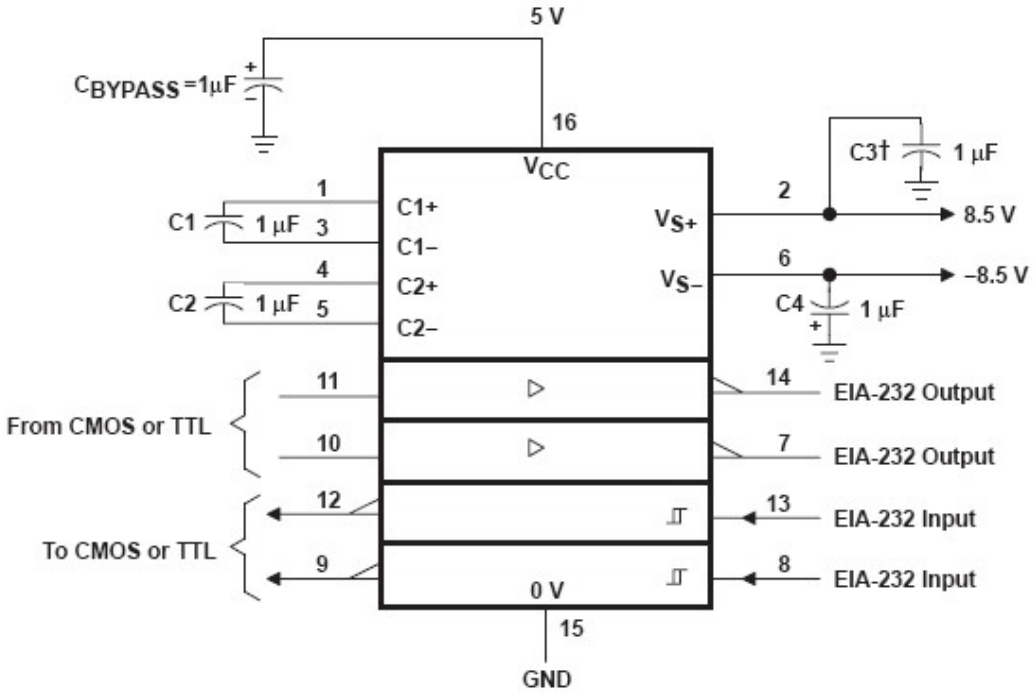
OSI/ISO Model



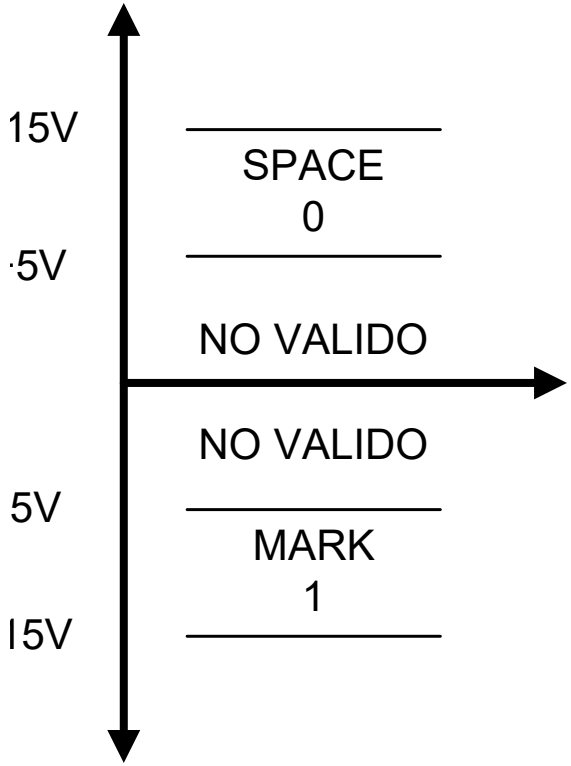
RS-232C


RS232
RS232
DTE (Terminal)
DCE (Modem)
Telephone Line
DCE (Modem)
DTE (Terminal)
IBM Compatible
Modem
Línea Teléfonica
Modem
IBM Compatible

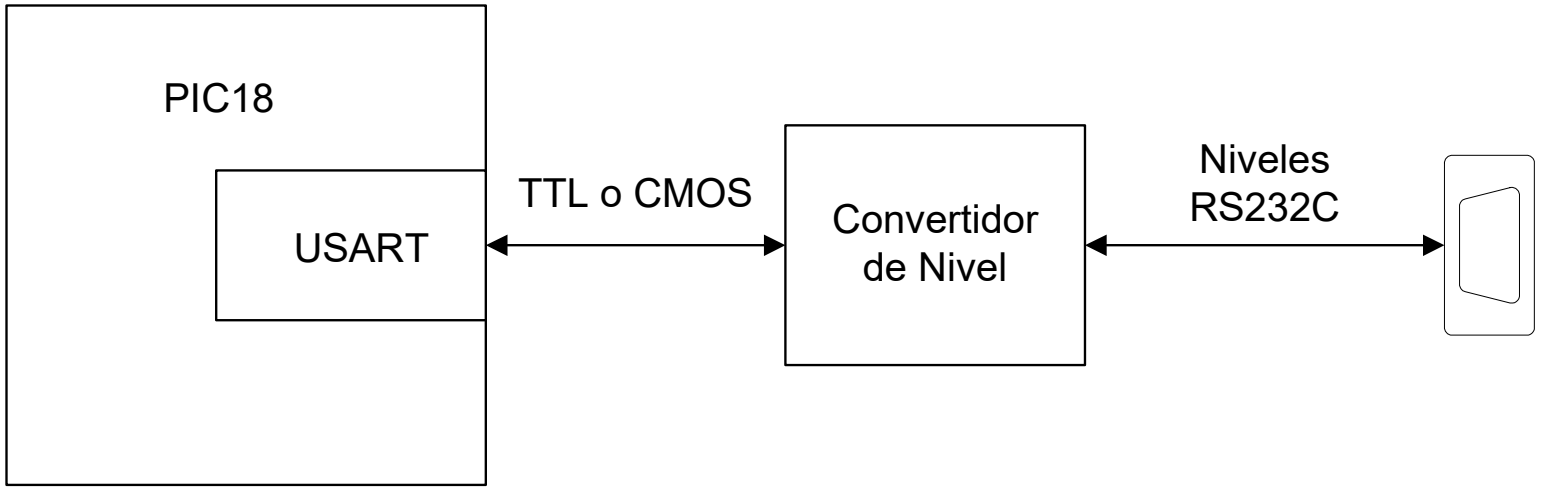
RS-232C



MAX232 level converter



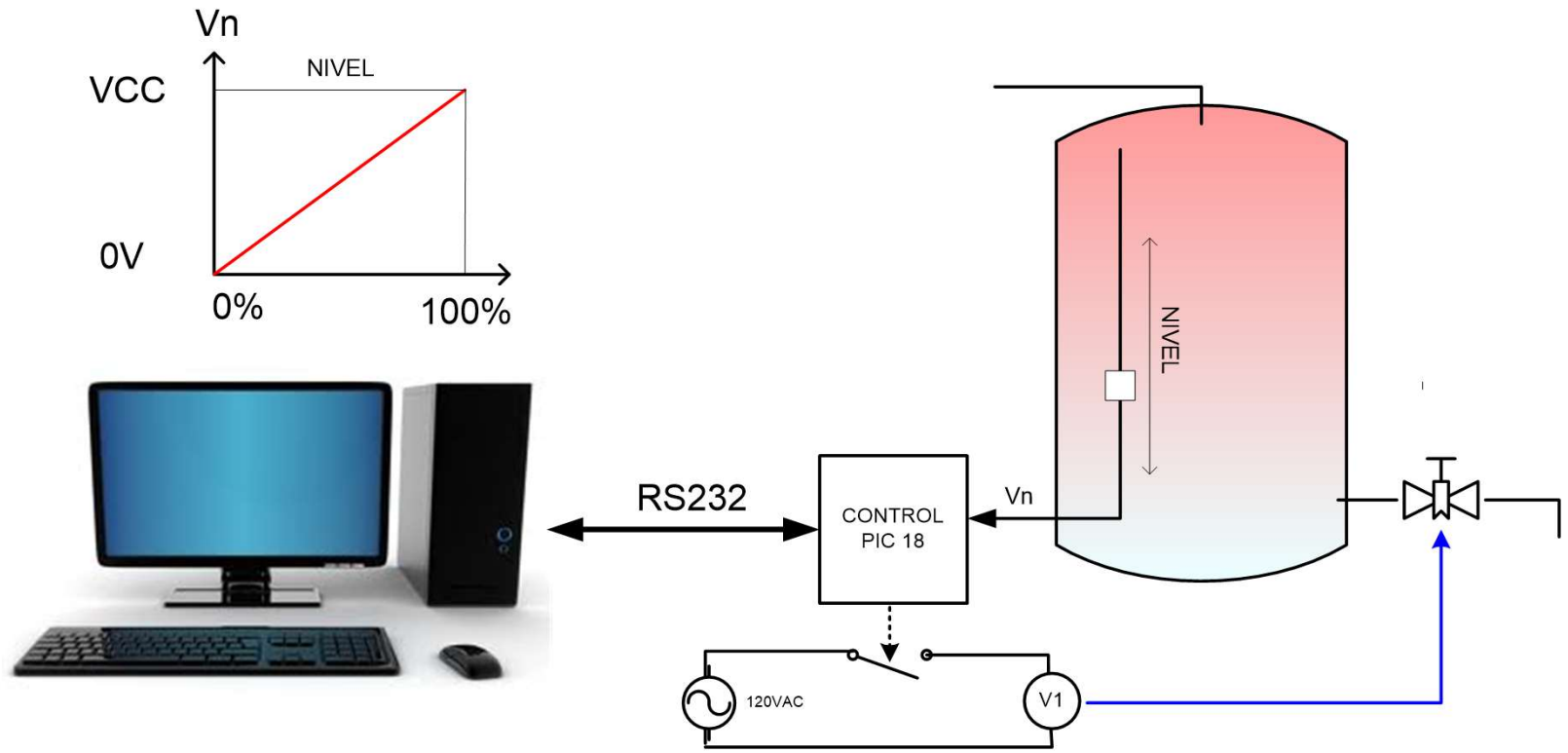
RS-232C



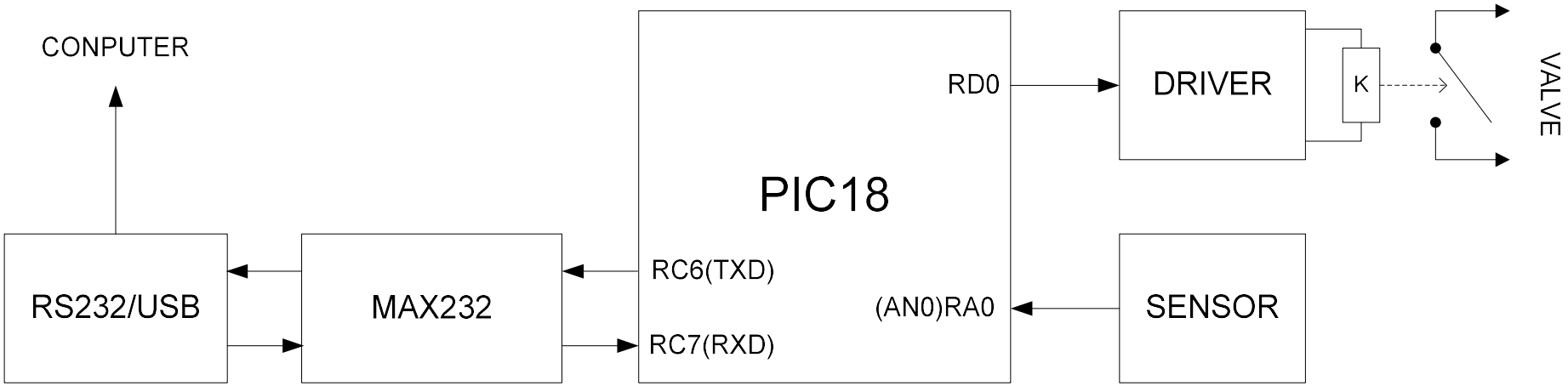
Example

- Write a program that implements a level control using serial communication
- Using a command sent by the DTE (computer), the user will be able to mody the level set point in a tank
- As a response to the command the controller must answer with the current level value

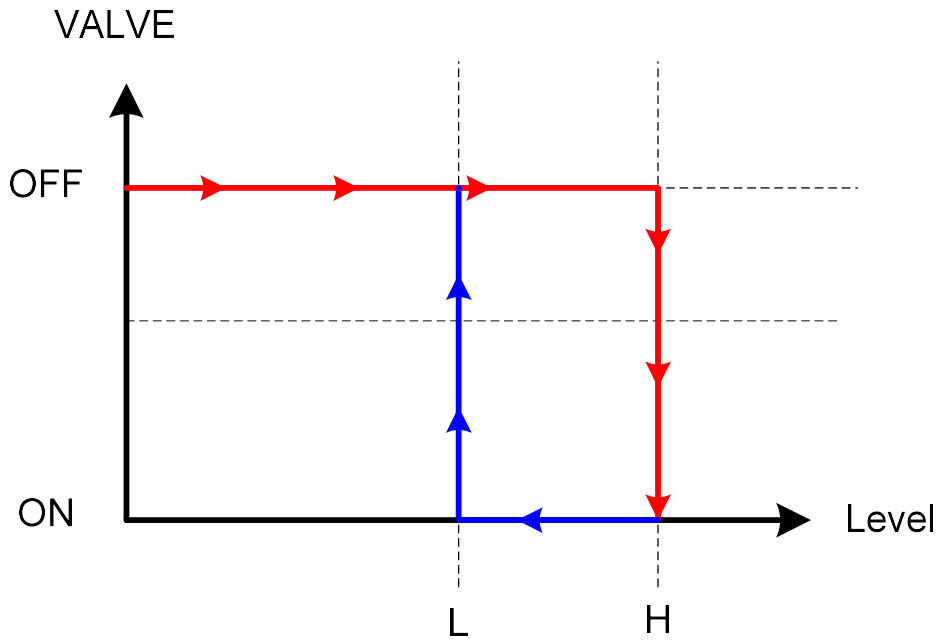
System topology



Controller design



Control algorithm



Protocol

- The communication is made by commands with the following structure sent at 9600bps

:LLLHHH<CR><LF>

- The first bytes is the start identifier and is equal to character ":" (colon)
- LLL Defines the lower level of the control band and is number between 000 and 100 (numeric characters)
- HHH Defines the higher level of the control band and is number between 000 and 100 (numeric characters)
- To each command the controller must respond with a text NNN that is equal to the current level

>NNN<CR><LF>

Protocol

- If the command is truncated from the DTE to the DCE (Master to controller) , the controller (slave) must detect this issue and abort the reception process.
- Use a state machine coding scheme to write the protocol and the control algorithm

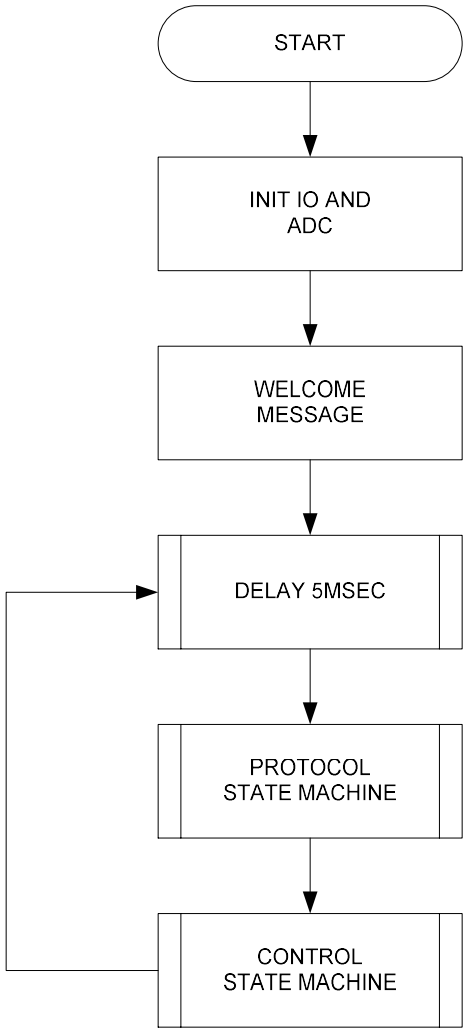
Example commands

- The computer tells the controller it wants to maintain the level between 45% y 60% of the tank

`:045060<CR><LF>`

- Supposing that at the moment the command is set, the current level is 35% the controller will answer back with the following text string.

`>035<CR><LF>`

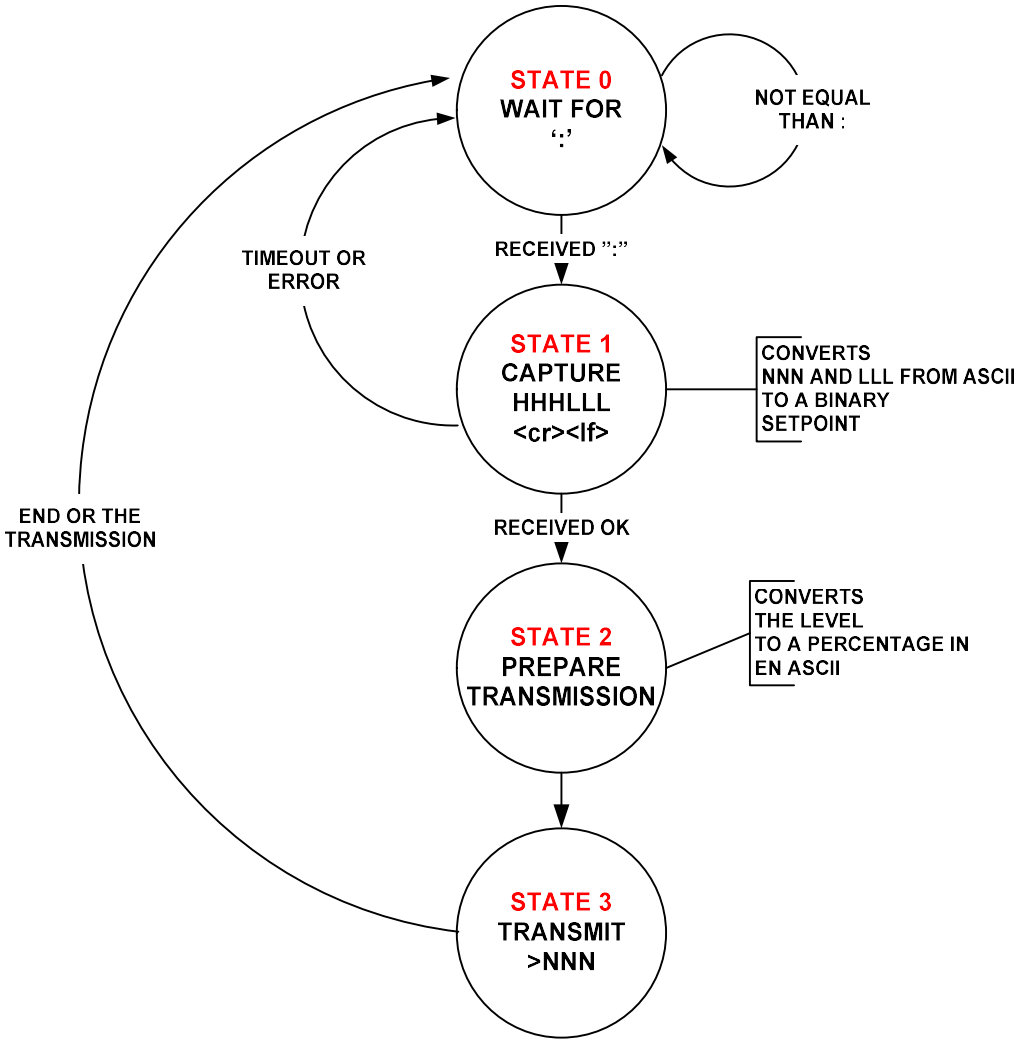

START
INIT IO AND ADC
WELCOME MESSAGE
DELAY 5MSEC
PROTOCOL STATE MACHINE
CONTROL STATE MACHINE

```
#include<xc.h>

void init_oscillator(void);      //Init oscillator
void put_caracter(char);         //Output
void init_uart(void);            //Inicializa UART
void put_string(char *);         //Outputs a string
void protocol_machine(void);     //State machine that executes prococol
void init_adc(void);             //Inits the ADC
void control_machine(void);      //State machine for control
unsigned char setpoint_low;      //Stores low value set point
unsigned char setpoint_high;     //Stores high value set point
unsigned char tank_level = 0;    //Stores the current tank level
unsigned char error_flag = 0;    //Flag that tells main if error
#define _XTAL_FREQ 16000000      //For the delay function
#define CTE_TIMEOUT    1000      //5 for intra character delay
#define VALVE  PORTAbits.RA4     //Define the control output
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the UART
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_uart(void){
        TRISCbits.RC7 = 1;      //RXD (RC7) is input
        ANSELCbits.ANSC7 = 0;   //digital
        TRISCbits.RC6 = 0;      //TXD (RC6) is output
        ANSELCbits.ANSC6 = 0;   //digital
//Uart configuration
        TXSTA1 = 0b00100000;    //8 bits asyncrhonous, Baud = low
        RCSTA1 = 0b10010000;    //8 bits reception eneabled
        BAUDCON1 = 0b00000000;  //Normal mode
        SPBRG1 = 25;            //9600bps Fosc = 16MHZ
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Function that inits the ADC
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void init_adc(void){
    TRISAbits.RA0 = 1;          // RA0 as input
    ANSELAbits.ANSA0 = 1;       // RA0 is analog
    //Config the ADC
    ADCON0 = 0b00000001;        //Select AN0 and turn on ADC
    ADCON1 = 0b00000000;        //Vref+ (VCC), Vref-(GND)
    ADCON2 = 0b00111110;        //Just left, 20TAD, FOSC/4
}
```

Protocol State Machine

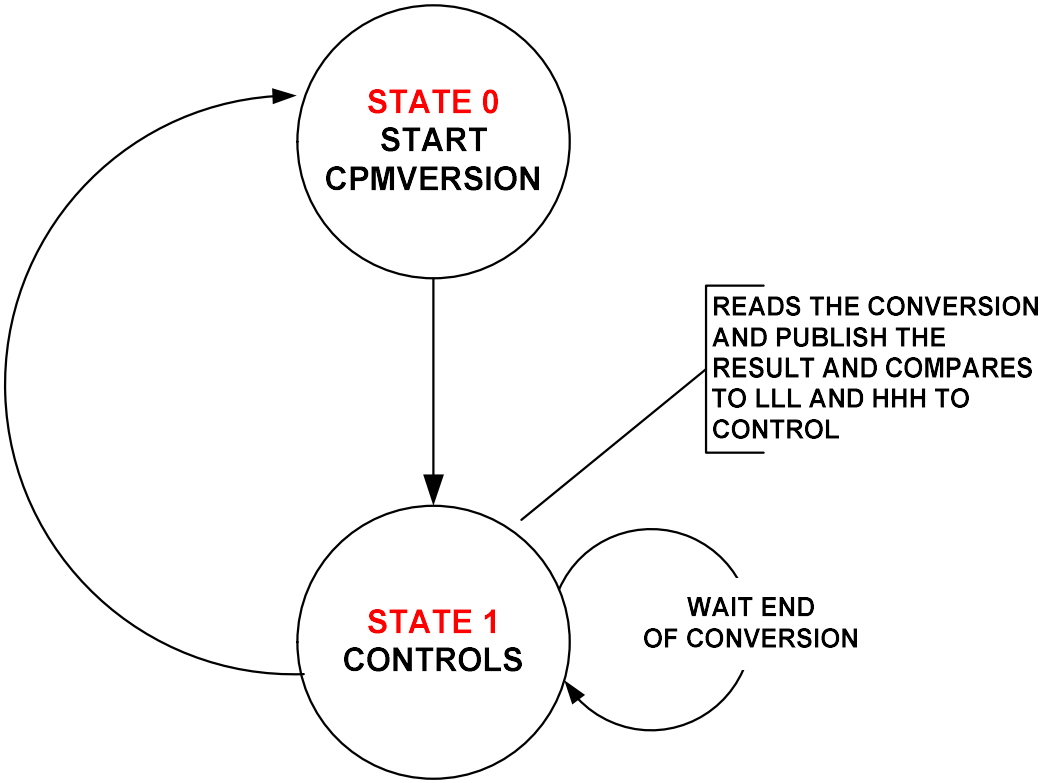


```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Executes the protocol
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void protocol_machine(void){
    static unsigned char state = 0;
    static int timeout = 0;
    static char datos[9];
    static char i;
    int result,cent,dece,unid;
    char caracter;
    switch(state){
        case 0x00:
            if(!PIR1bits.RC1IF) break;              //Byte received?)
            caracter = RCREG1;                      //Transfer to variable
            PIR1bits.RC1IF = 0;                     //Clear the flag
            if(caracter != ':') {
                error_flag = 1;                     //Tell main there was an error
                break;                              //If not a valid start abort
            };
            timeout = CTE_TIMEOUT;                  //Load the timeout constant
            state = 0x01;                          //Chance to next state
            i = 0;                                  //Starts the index = 0;
            break;
```

```
case 0x01:
    if(!PIR1bits.RC1IF){
        timeout--;                          //Check if timeout
        if(timeout == 0){
            state = 0x00;                   //If it was, then back state 0
            error_flag = 1;                 //Tell main there was an error
        }
        break;
    }                                       //Else we continue
    timeout = CTE_TIMEOUT;                  //Reload the timeout constant
    PIR1bits.RC1IF = 0;                     //Clear the flag
    datos[i] = RCREG1;                      //Store byte in array
    i++;
    if(i != 8) break;                       //Did we finish ??
    //Verify if the last characters are = 0x0d y 0x0a
    if((datos[6] != 0x0d) || (datos[7] != 0x0a) ){
        //If it was not terminated we abort and return to state 0
        error_flag = 1;                 //Tell main there was an error
        state = 0x00;
        break;
    }
    //Contert the string LLL to a number from 0 to 255
    result = (datos[0]-'0')*100 + (datos[1]-'0')*10 + (datos[2]-'0');
    result = (result * 255)/100;
    setpoint_low = (unsigned char)result;
    //Contert the string HHH to a number from 0 to 255
    result = (datos[3]-0x30)*100 + (datos[4]-0x30)*10 + (datos[5]-0x30);
    result = (result * 255)/100;
    setpoint_high = (unsigned char)result;
    state = 0x02;
    break;
```

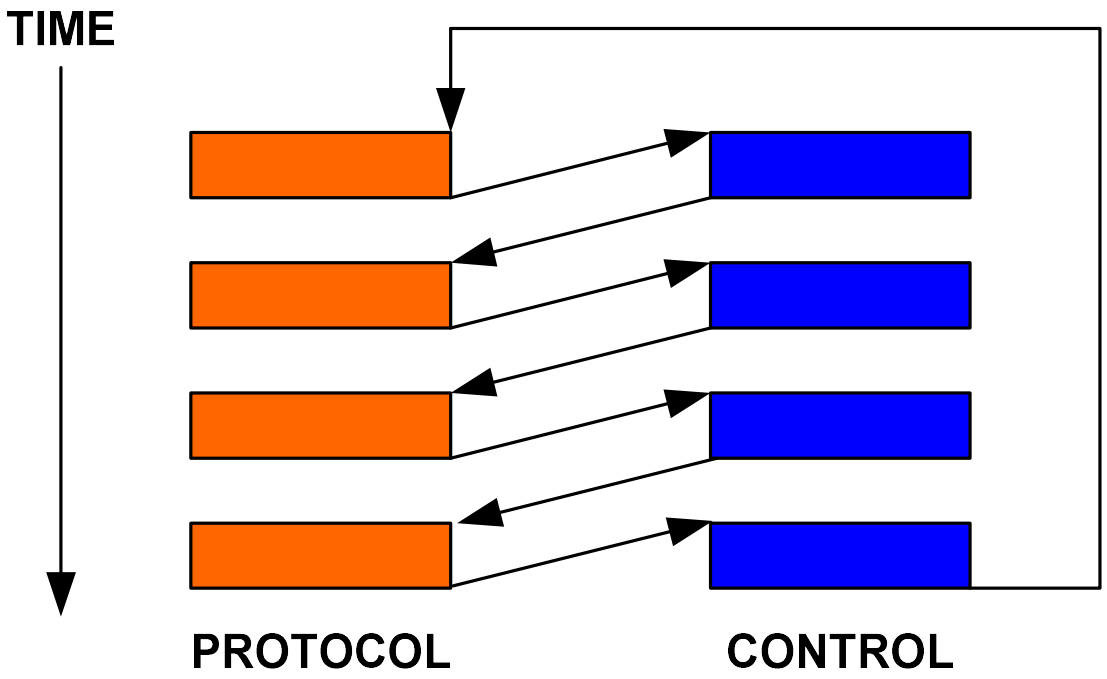
```
        case 0x02:
            //All was ok now we prepare the answer
            datos[0] = '>';
            result = (((int)tank_level*100)/255);       //Conver to a percentage
            cent = result /100;                         //The hundreds
            dece = (result - cent*100)/10;              //The tenths
            unid = (result -cent*100 - dece*10);        //THe units
            datos[1] = (unsigned char)(cent + '0');
            datos[2] = (unsigned char)(dece + '0');
            datos[3] = (unsigned char)(unid + '0');
            datos[4] = 0x0d;
            datos[5] = 0x0a;
            state = 0x03;
            i=0;
            break;
        case 0x03:
            if(!TXSTA1bits.TRMT) break;      //Finish ?
            TXSTA1bits.TRMT = 0;
            TXREG1 = datos[i];
            i++;
            if(i != 6) break;    //End of string ?
            state = 0x00;
            break;
}
```

Control state machine



```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+ Executes the control
//++++++++++++++++++++++++++++++ ++++++++++++++++++++++++++++++++++++++++++++
void control_machine(void){
    static unsigned char state = 0;
    switch(state){
        case 0x00:
           ADCON0bits.DONE = 1;      //Start the conversion
           state = 0x01;
           break;
        case 0x01:
            if(ADCON0bits.DONE) break;  //Finish ?
            tank_level = ADRESH;        //Read the AD converter
            //Algoritmo de control
            if(tank_level > setpoint_high )  VALVE = 1;     //Open valve
            if(tank_level < setpoint_low  )  VALVE = 0;     //Close valve
            state = 0x00;
            break;
    }
}
```

```
main(){
char const message[20] = {"LEVEL CONTROL\n"};        //Const stores the array
char const errormsg[20] ={"\nPROTOCOL ERROR\n"};     //In ROM
        init_oscillator();       //Init oscillator
        TRISAbits.RA4 =0;        //Output  Our control output
        PORTAbits.RA4 =0;        //Initial value
        init_adc();              //Init ADC
        init_uart();             //Config the UART
        put_string(&message);    //Output initial message
    while(1){                    //Infinite loop
        __delay_ms(5);           //Delay de 5msec
        if(error_flag){
            put_string(&errormsg);
            error_flag = 0;
        }
        protocol_machine();      //Call protocol machine
        control_machine();       //Call control machine
    }// while(1)
} //from main() TEMA_12_PIC_2.C
```


TIME
PROTOCOL
CONTROL