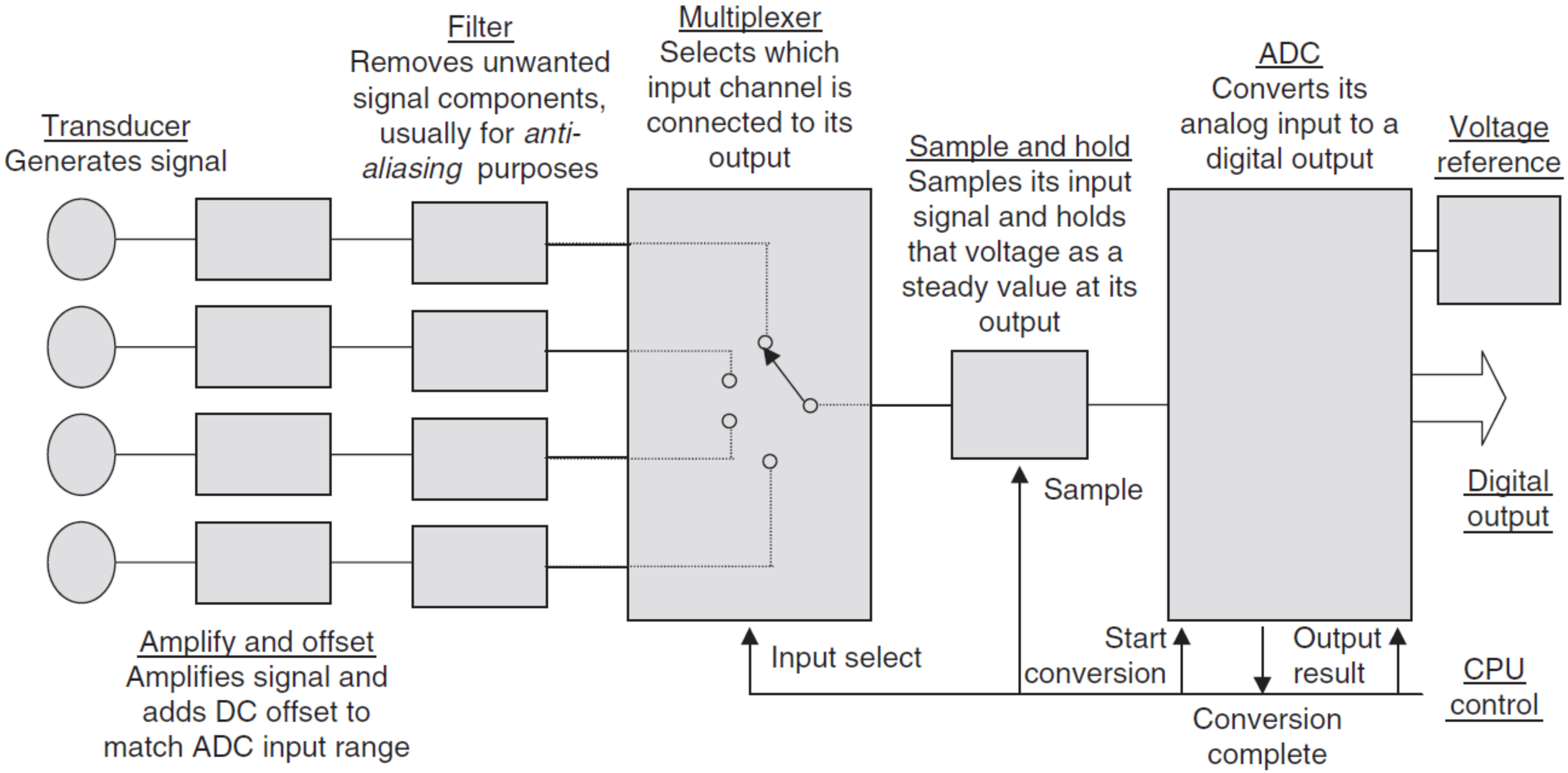
Embedded Systems
Programming and Architectures

Lecture No 10 : Data acquisition and data transfer

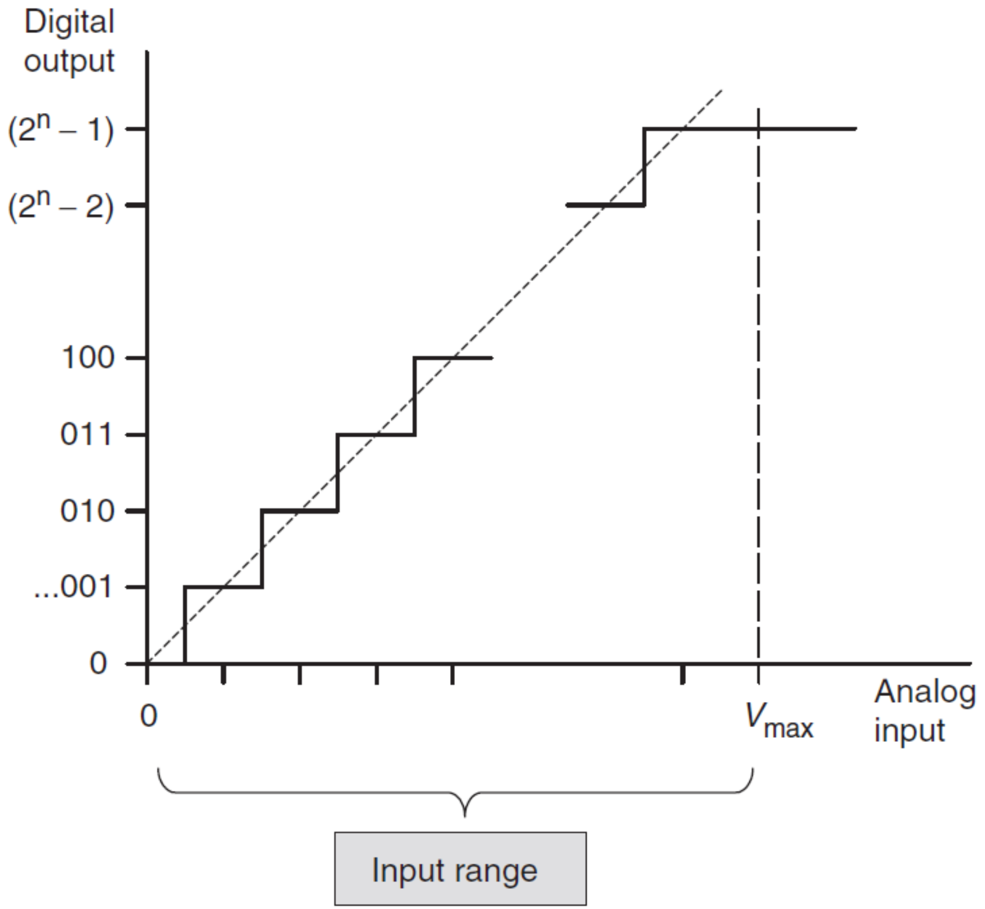
Dr John Kalomiros
Assis. Professor
Department of Post Graduate studies in Communications and Informatics

Elements of a data acquisition system



Source: Designing Embedded systems with PIC microcontrollers by Tim Wilmhurst

The ideal input-output ADC characteristic



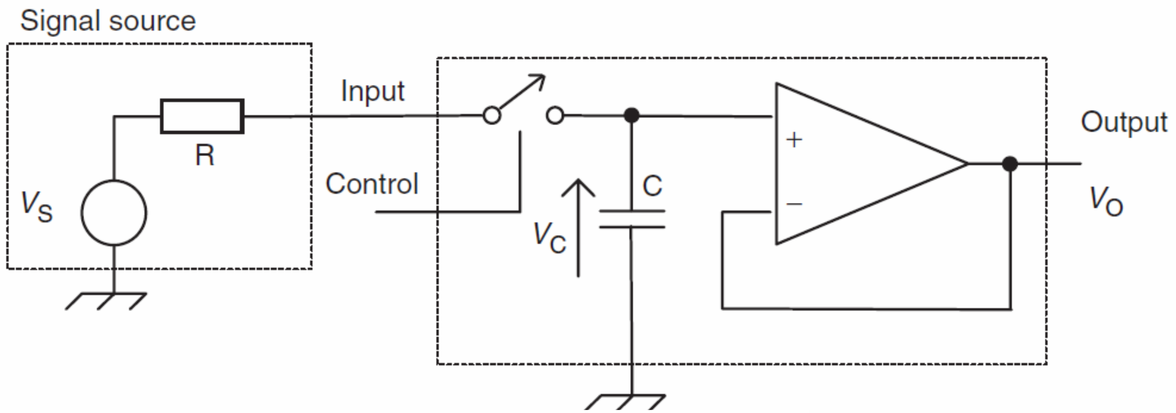
Source: Designing Embedded systems with PIC microcontrollers by Tim Wilmhurst

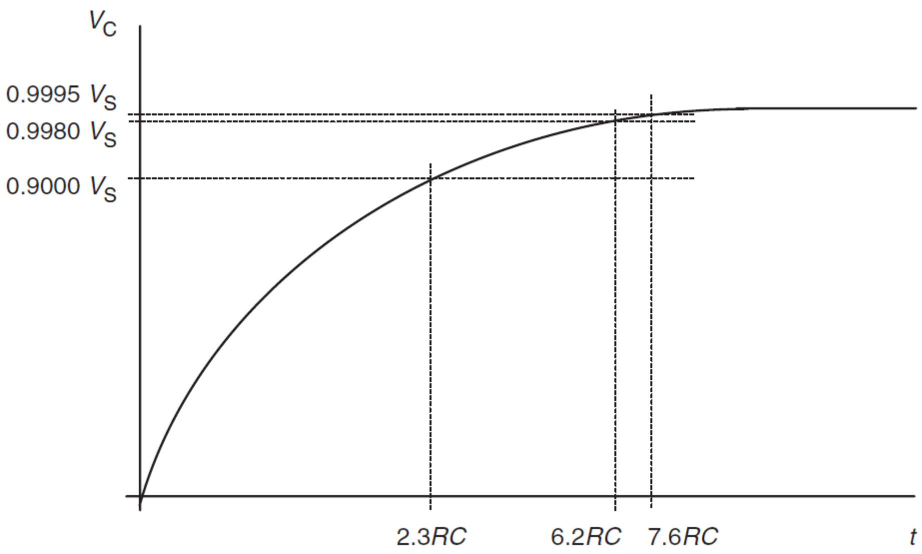
Important characteristics

- Conversion speed: the time needed for a conversion to complete
- Resolution: how much the input changes for each quantum step (this is dependent on the number of output bits)
- For periodic signals the Nyquist sampling criterion should be met.

  According to the criterion the conversion rate should be at least twice the maximum frequency
- A reference voltage determines the dynamic range of the input voltage

Sample and hold circuit.
The meaning of acquisition time





Sample and Hold acquisition time:

V_C should become close to V_S by some error value. To ensure good accuracy, the error should be less than ½ LSB. At 10 bit this means rise time t=7,6RC

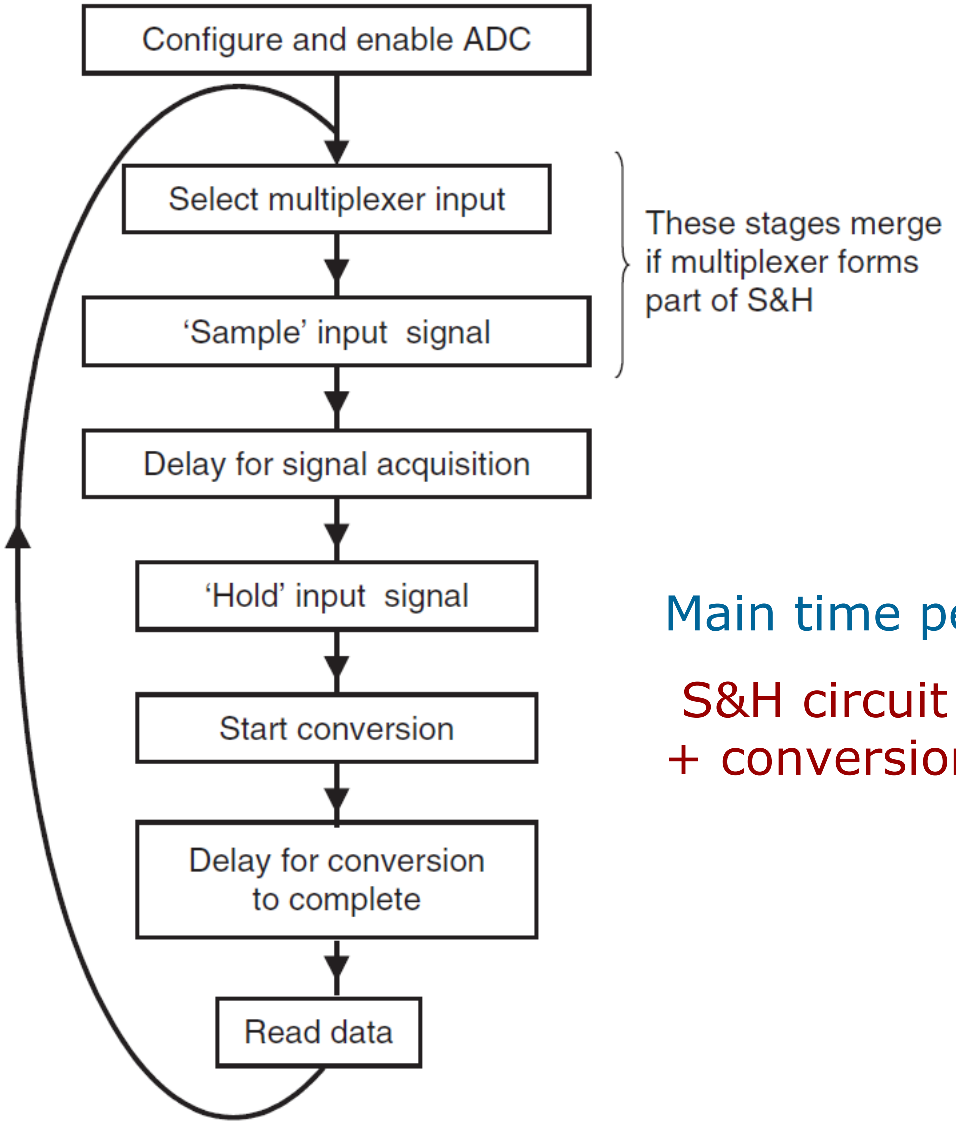
Conversion time and total conversion rate

Beside S & H acquisition time, the ADC needs some time for the actual signal conversion. Therefore, the total time for one conversion is given as follows:

total time for conversion = S&H acquisition time + conversion time

Later we shall discuss how these two time intervals are calculated

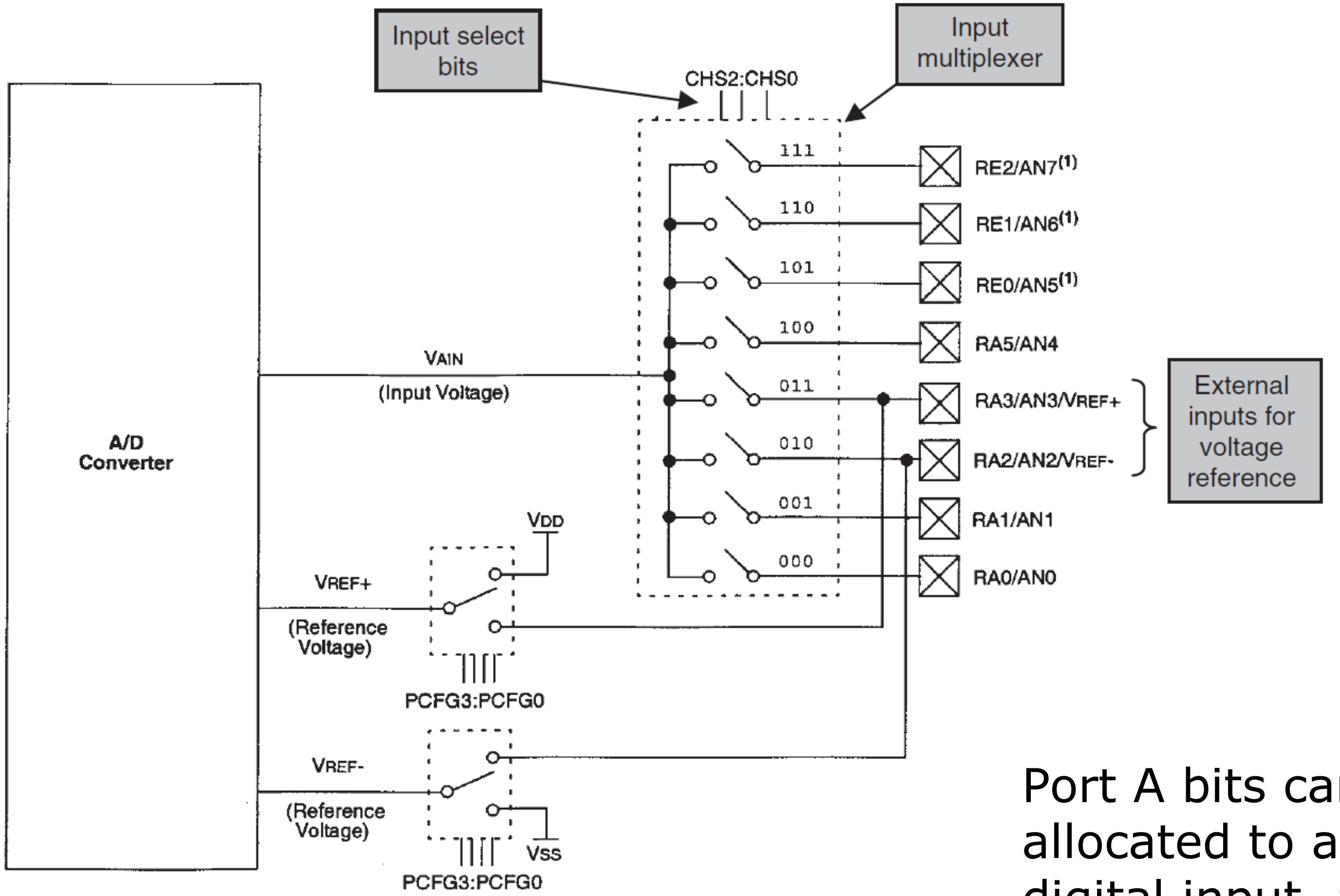
Typical ADC conversion flow



Main time periods implicated:

S&H circuit acquisition time + conversion time

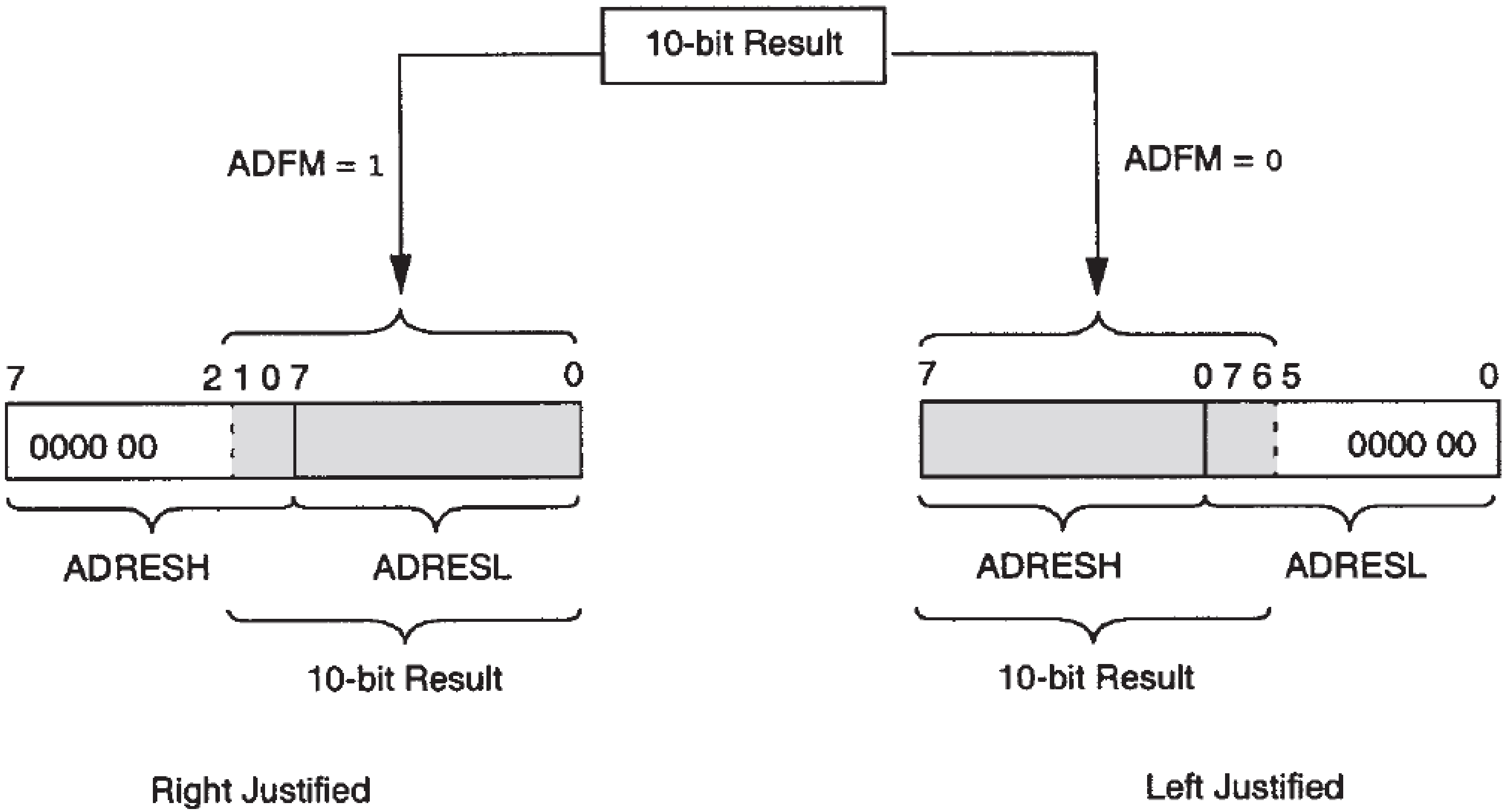
The PIC 16F87x analog to digital converter



Port A bits can be allocated to analog or digital input, according to settings in an SFR

Source: Microchip data-sheet for PIC16F8xx

Formatting the 10-bit result of the ADC



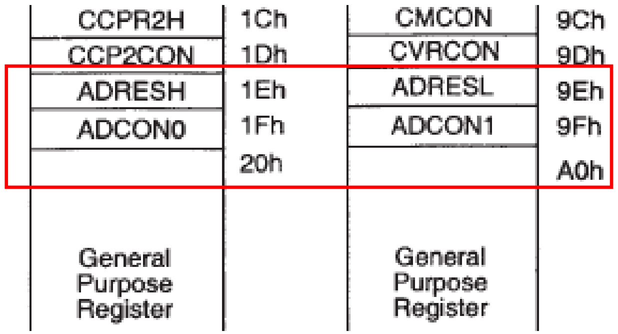
Important registers for ADC control and acquisition

ADCON0

ADCON1

ADRESH

ADRESL



Also important: TRISA, TRISB. Any bits used as analog inputs should be set as inputs

If ADC interrupts are used, then registers PIR1 and PIE1 also play a role

Controlling the ADC – the ADCON0 reg

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	FOSC/2
0	**01**	FOSC/8
0	**10**	FOSC/32
0	**11**	FRC (clock derived from the internal A/D RC oscillator)
1	**00**	FOSC/4
1	**01**	FOSC/16
1	**10**	FOSC/64
1	**11**	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit
1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Source: Microchip data-sheet for PIC16F8xx

Controlling the ADC – the ADCON1 reg

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O

Source: Microchip data-sheet for PIC16F8xx

Calculation of conversion time

The sample conversion is based on the ADC clock. This clock is derived from the external oscillator by some division. If the period of the ADC clock is T_{AD}, then the time of a single 10-bit conversion is

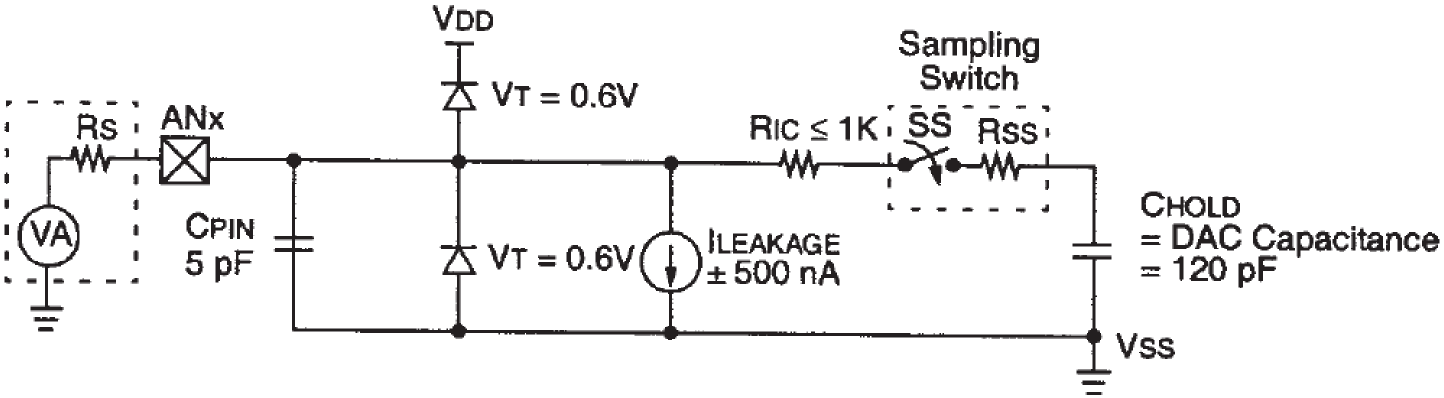
conversion time = $12 \times T_{AD}$

Clock division for ADC is controlled by bits ADCS2, ADCS1, ADCS0 of the ADCON0 and ADCON1 registers (slide 10)

Note: ADC clock period T_{AD} cannot be lower than 1,6 μs or the frequency F_{AD} cannot be higher than 625 KHz. Therefore:

Minimum conversion time = 12 x 1,6 μs = 19,2 μs

Calculation of acquisition time



$$t_{ac} = \text{Amplifier settling time} + \text{Hold capacitor charging time}$$

Amplifier settling time ≈ 2μs

Hold capacitor sampling time = 7,6RC=

$(R_{IC} + R_{SS} + R_S)C_H$ = (1K + 7K + R_S)120pF = 9,6 μs

(for max R_S=2,5K)

total acquisition time = 2μs + 9,6μs = 11,6 μs

Total conversion time and maximum sampling rate

total time for conversion = S&H acquisition time + conversion time

or

total time needed for one conversion = 11,6 μs + 19,2 μs = 31 μs

This means that the maximum sampling rate is 32258 samples/sec
or 32 KHz.
According to the Nyquist criterion the higher input frequency can be 16 KHz

(Yes, you may try overclocking, especially when using only 8 out of 10 bits, however be careful to test the validity of the result by measuring reference signals and comparing with lower and trusted conversion rates).

Let's write some code: initialize ADC

```
#include "p16f877.inc"
__CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_OFF & _CPD_OFF
    &_WRT_ENABLE_ON & _BODEN_ON & _LVP_OFF

        Org 00
        ;initialize ADC
        bsf STATUS, RP0
        movlw b'00011111'  ;5 first bits of PORTA inputs
        movwf TRISA
        movlw b'00000000'
        movwf TRISB
        movlw b'01000010'  ;left justified, ADCS2=1, 3 Dig 5 Analog ch
        movwf ADCON1
        bcf STATUS, RP0
        movlw b'01000001'  ;101 Fosc/16, ch0, ADON
        movwf ADCON0
        clrf PORTB
```

A/D conversion and reading ADC

```
;conversion
loop1    bcf PIR1, ADIF
         nop         ;wait for the output of S&H circuit to stabilize
         nop
         nop
         bsf ADCON0, GO
wait     btfss PIR1, ADIF
         goto wait
;read ADC
         movf ADRESH, w
         movwf PORTB
         goto loop1

         end
```

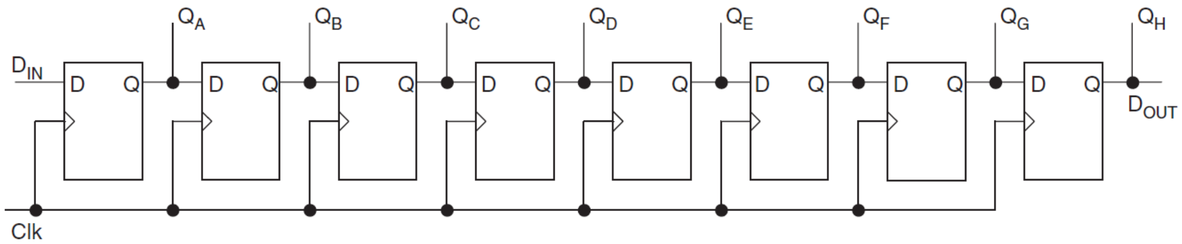
Code in C – make your own functions

```
#include <htc.h>        //or #include "pic1687x.h" for PIC16F877
#include "my_adc.h"  //header file with my ADC function definitions

/*Set CONFIGURATION BITS in code*/
__CONFIG(UNPROTECT & PWRTEN & WDTDIS & XT & LVPDIS);

void main(void)
{
    init_adc();                     // my initialize ADC

    while(1)
    {
    convert_adc();                  // my ADC conversion

    PORTB=read_adc();               //my ADC reading
    }
}
```

my_adc.h – a header file with my functions for A/D conv

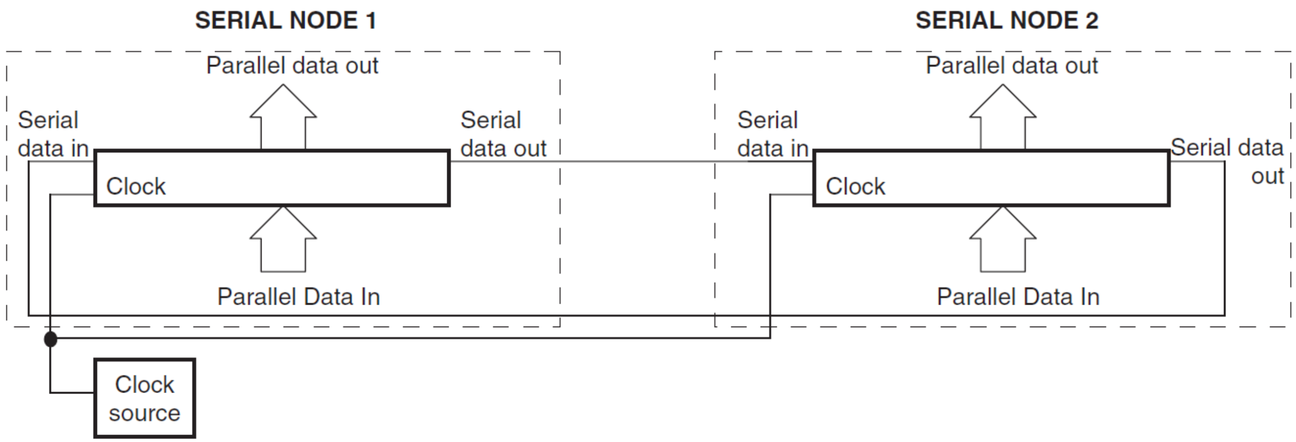
```
//user function prototypes for adc
void init_adc(void);
void convert_adc (void);
char read_adc(void);
void my_delay(void);


//user function definitions for adc
void init_adc (void)                    // initialize
{
TRISA=0b00000111;
TRISB=0x00;
ADCON1=0b01000010;                      // left justified, ADCS2=1, 3 Dig 5 Analog ch
ADCON0=0b01000001;                      // 101 Fosc/16, ch0, ADON
PORTB=0x00;
}

void convert_adc (void)                 // convert
{
ADIF=0;
my_delay();
ADCON0=ADCON0 | 0b00000100;             // GO=1
while (!ADIF);                          // wait for end of conversion (ADIF=1)
}

char read_adc (void)                    // return ADC High byte
{
return ADRESH;
}

void my_delay (void)                    // wait for S&H circuit to stabilize
{
}
```

Serial communication: synchronous and asynchronous



The heart of serial communication:

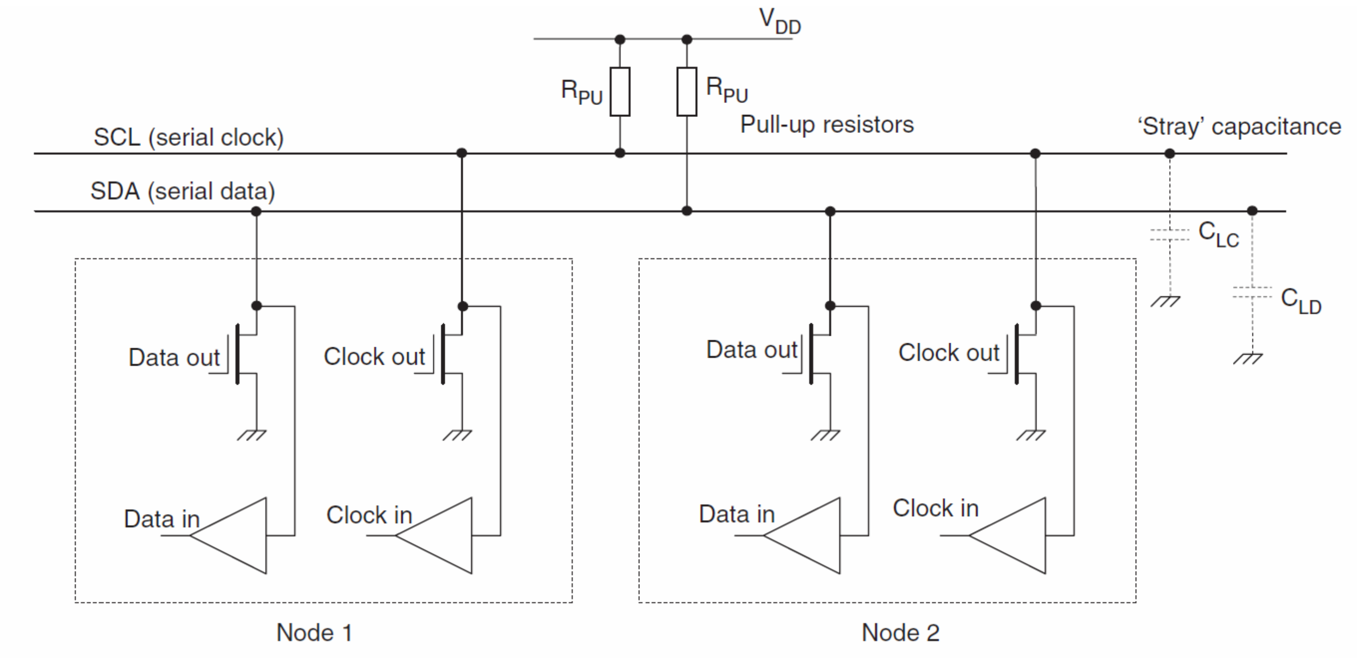
the shift register



The idea of synchronous serial com:

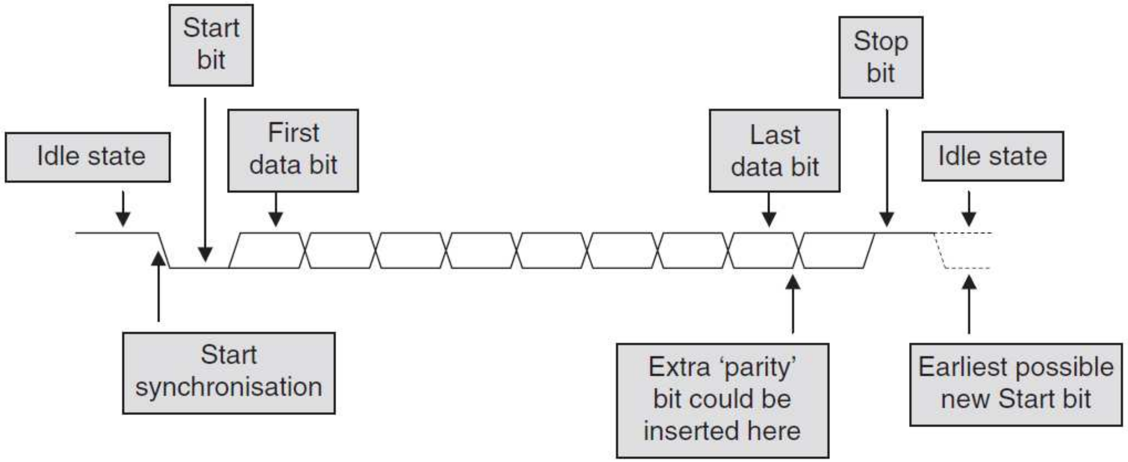
serial data and serial clock lines

Enhanced synchronous serial communication: the I^{2}C interface



A clear data-transfer protocol is established between master and slaves
Do we really need to send the clock signal wherever the data goes?
(extra line, large bandwidth, loss of synchronization over long distances)
The answer is…

Asynchronous serial communication

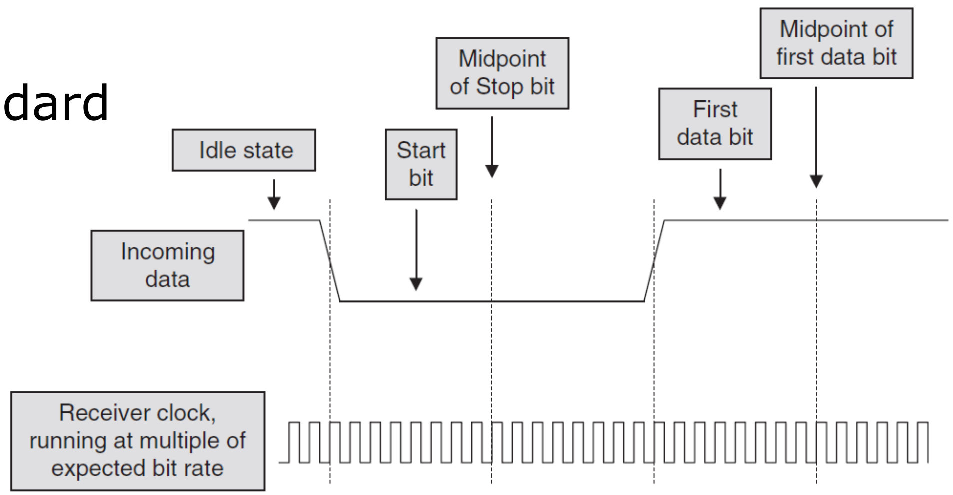


Data rate is predetermined

Each byte is framed with a start and stop bit

RS232 is a common standard

for asynchronous serial communication



Internal clock runs at a multiple of the baud rate

Main registers of the serial port

PIC USART can be used as synchronous master, synchronous slave or asynchronous port. As asynchronous port it is full duplex, that is it can transmit and receive simultaneously, using separate shift registers for Transmit and Receive.

Input and output rate is controlled by a circuit called baud rate generator. The serial port clock has a frequency which is a multiple of the baud rate.

USART operation is controlled by two registers, TXSTA and RCSTA.

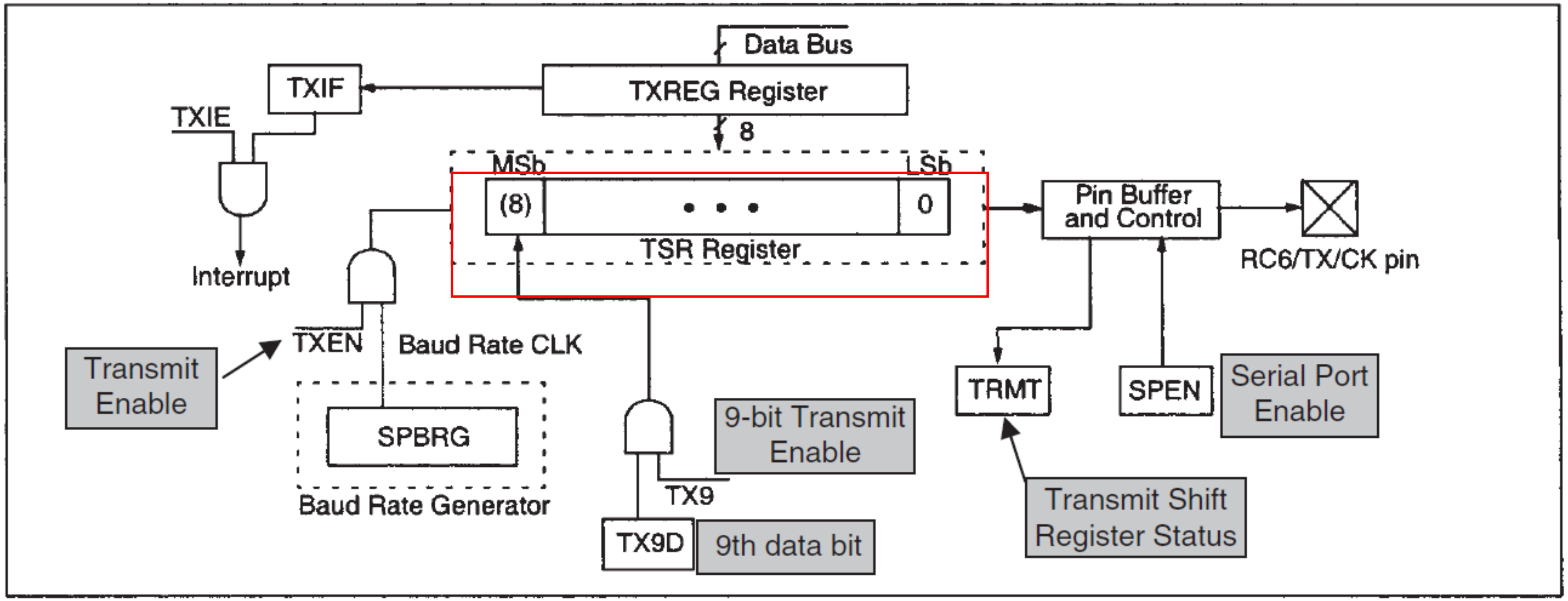
RCSTA.SPEN enables the Serial Port TXSTA.SYNC selects synchronous or asynchronous mode

Data to be transmitted are written by the program into TXREG

Received data go into RXREG

Serial Port Baud rate Generator SPBRG register controls the baud rate, depending on the value it holds.

Transmitter block diagram



TXSTA register TXEN, TRMT, TX9 control the transmit process

TXIE and TXIF belong to PIE1 and PIR1 registers

SPEN belongs to RCSTA

TXSTA, for reference

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	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 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

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 Parity bit

Control of the baud rate generator

Bit BRGH controls high and low rate

$$\text{For } \mathbf{BRGH} = 0 \qquad \text{Baud rate} = \frac{f_{\text{osc}}}{64([\mathbf{SPBRG}] + 1)}$$

$$\text{For } \mathbf{BRGH} = 1 \qquad \text{Baud rate} = \frac{f_{\text{osc}}}{16([\mathbf{SPBRG}] + 1)}$$

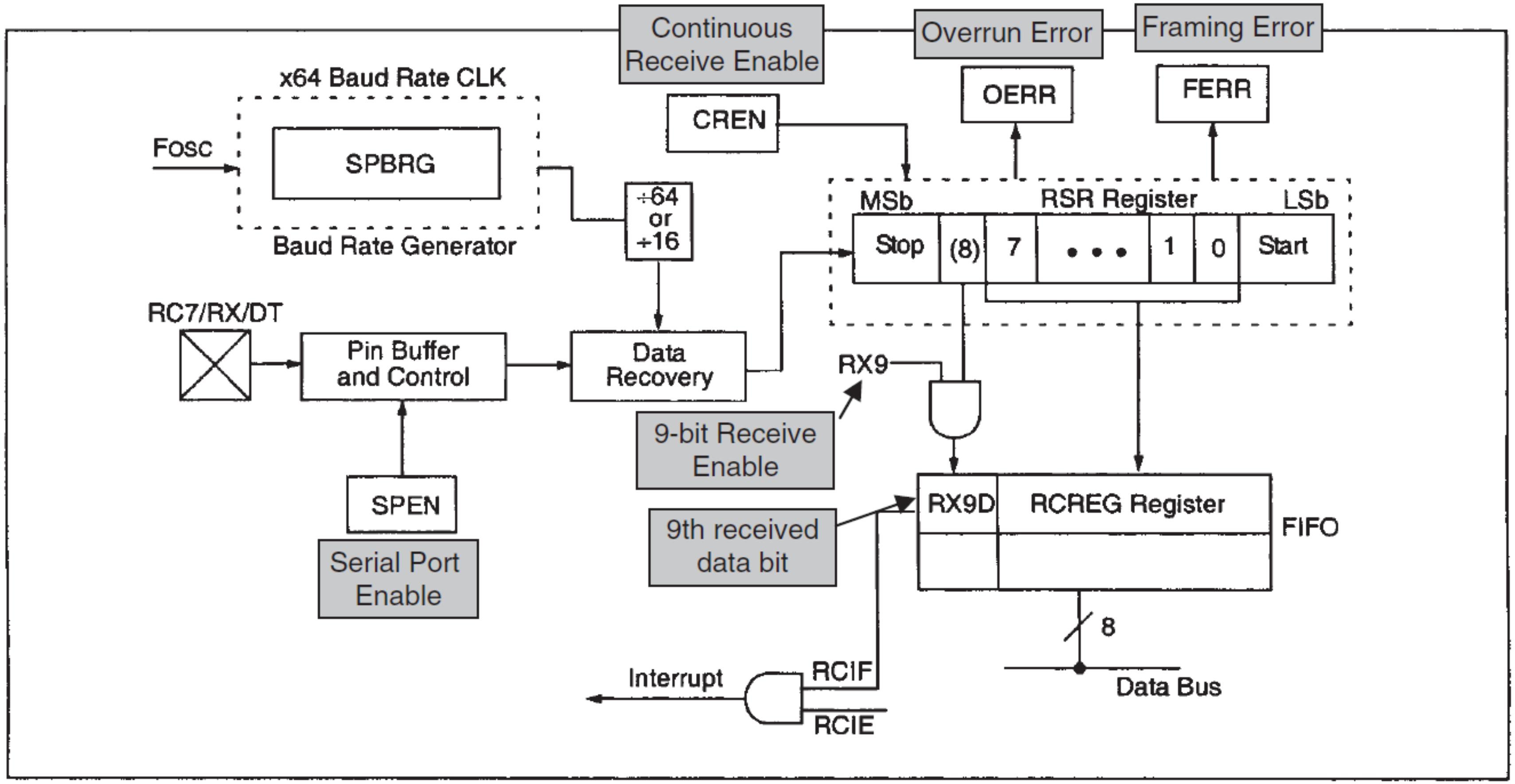
Registers related to asynchronous transmission

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Transmit procedure

- Initialize SPBRG writing the proper baud rate code value (see slide 25). If high baud rate is required then set BRGH=‘1’
- Enable asynchronous serial port by resetting ‘0’ the SYNC bit (TXSTA<4>) and setting ‘1’ the SPEN bit (RCSTA<7>).
- If interrupts are to be used then TXIE bit has to be enabled (PIE1<4>). In this case INTCON bits <6> and <7> should be set.
- Transmission is enabled by setting TXEN bit (TXSTA<5>).
- To begin serial data transfer test TXIF and if it is set then load TXREG with data. (TXIF=1 means TSR is loaded, therefore TXREG is empty). The flag is auto-reset when TXREG is reloaded.

Receiver block diagram



RCSTA, for reference

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

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

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 continuous receive
0 = Disables continuous receive
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 load of 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

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG 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:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

Registers related to asynchronous reception

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Receive procedure

- Two first steps are same as in transmit procedure
- Continuous reception is enabled by setting CREN=‘1’ in RCSTA
- When RCIF is ‘1’ a byte has been received in RCREG
- RCREG is read and used for further processing

Initialization of serial port, no interrupts

```
;initialize serial port
initialize bsf STATUS,RP0
           movlw 0x00
           movwf TRISB              ;PORTB output for display
           movlw b'10000001'        ;RC7 (RX) input, RC0 input
           movwf TRISC
           movlw b'00100100'        ;Transmit enable, high speed baud rate
           movwf TXSTA
           movlw d'103'             ;2404 bps, fosc=4MHz, see slide 25
           movwf SPBRG
           bcf STATUS,RP0
           movlw b'10010000'        ;port is on, 8-bit transfer, no address detect,
           movwf RCSTA              ;continuous receive enabled
           return
```

Send and Receive subroutines

```
send     btfss PIR1,TXIF     ; wait for TXIF to become '1'
         goto $-1            ; before TXREG is loaded
         movwf TXREG
         return

receive  btfss PIR1,RCIF     ; when new character is received RCIF=1
         goto receive        ; wait to receive
         movf RCREG,w        ; read RCREG
         return
```

Example main routine for serial send and receive

```
        Org 0
        call initialize
        movlw 0xAA      ;test pattern
        movwf PORTB
main    movf PORTD,w
        call send
        call receive
        movwf PORTB
        goto main
```

Interrupt on byte reception: activation and ISR

In subroutine initialization we add the following lines (interrupt enable)

```
bsf STATUS, RP0
bsf PIE1, RCIE          ;Enable Reception Interrupts
bcf STATUS, RP0
bcf PIR1, RCIF          ;clear Reception interrupt flag
bsf INTCON, PEIE        ;Enable Peripheral Interrupts
bsf INTCON, GIE         ;General Interrupt Enable
```

Instead of subroutine receive we now write an ISR:

```
ISR_rec     movf RCREG, W
            bcf PIR1, RCIF     ;clear Reception interrupt flag
            retfie
```

Project No 8

1. Write a C application that reads values from an analog Channel (ch3) and transmits them in digital form through asynchronous serial port.

2. Write a C application for a second MCU that receives through serial com the values transmitted above and displays them on the LEDs of PORTB

Required reading:

Designing Embedded Systems with PIC microcontrollers

by Tim Wilmshurst, chapters 10 and 11.