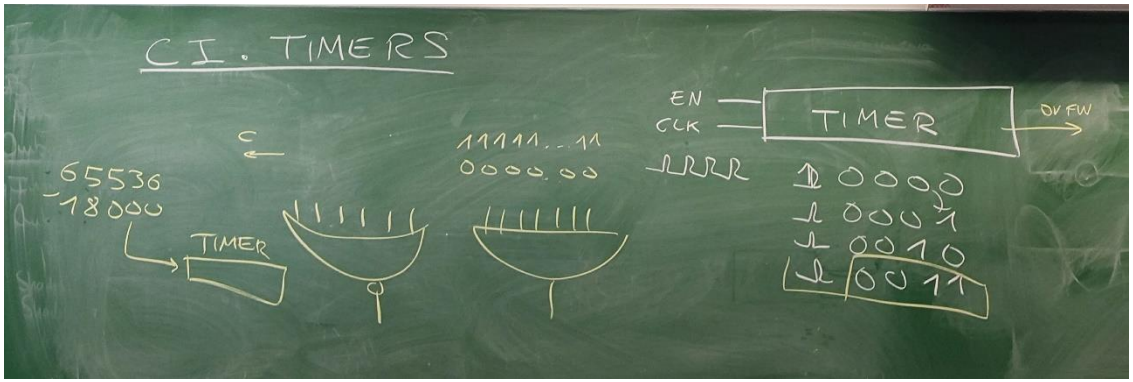
Introduction

- **Time** is represented by the count in a timer.

- There are many applications that cannot be implemented without a timer:

1. Event arrival time recording and comparison
2. Periodic interrupt generation
3. Pulse width and period measurement
4. Frequency and duty cycle measurement
5. Generation of waveforms with certain frequency and duty cycle
6. Time references
7. Event counting
8. Others



The PIC18 Timer0

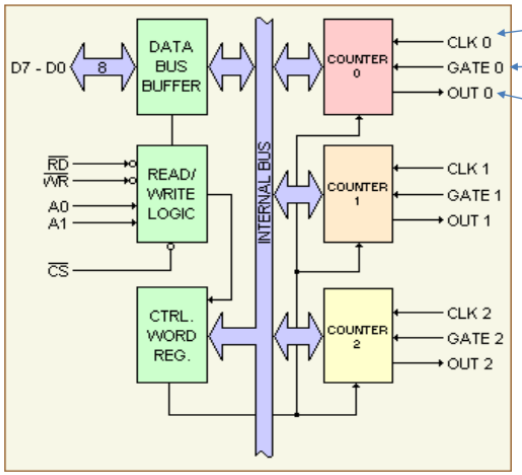
- Can be configured as 8-bit or 16-bit

- Is a timer/counter depending upon the clock source.

- An interrupt may be requested when Timer0 rolls over from 0xFFFF to 0x0000.

- Operation is controlled by the T0CON register.

Generic Timers



Clock source (IN)

Pause (IN)

Rollover/Overflow (OUT)
Count limit reached

e.g. Configurable counter:

One count:
0, 1, 2, … , N-1, N, N, N, N

Cyclic count:
0, 1, 2, … , N-1, N, 0, 1, 2, … , N-1, N

The PIC18 Timer System

- A PIC18F45K22 microcontroller has up to 7 timers: Timer 0…Timer 6.

- Timer 0, Timer 1, Timer 3 and Timer 5 are 16-bit timers whereas Timer 2, Timer 4 and Timer 6 are 8-bit.

- When a timer rolls over, an interrupt may be generated if it is enabled.

- Timer 2, Timer 4 and Timer 6 use instruction cycle clock as the clock source whereas the other timers may also use external clock input as the clock source.

- Timer 0 is designed to act as a time base (core interrupt) while the other timers are in the peripheral group (alternate time base and CCP operation).

The PIC18 Timer0

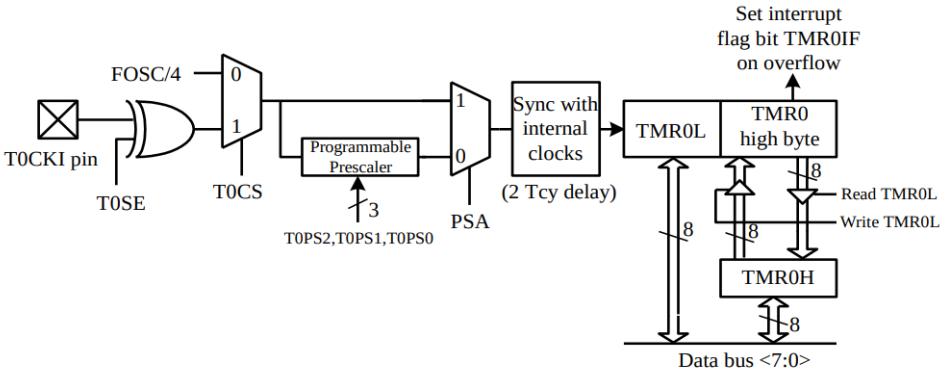


Figure 8.1b Timer0 block diagram in 16-bit mode (redraw with permission of Microchip)

T0CON Register

values after Reset →

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	TOPS<2:0>		
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

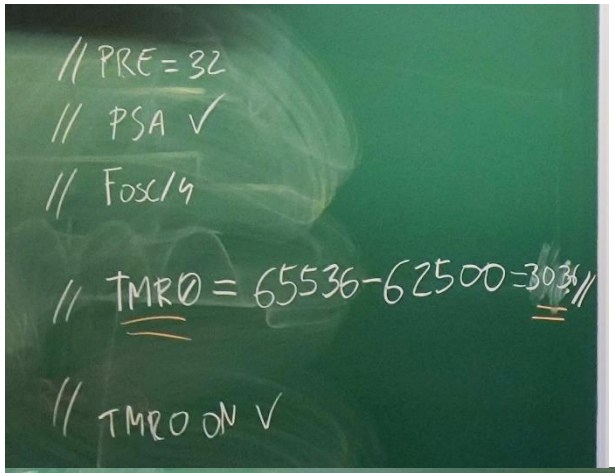
bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

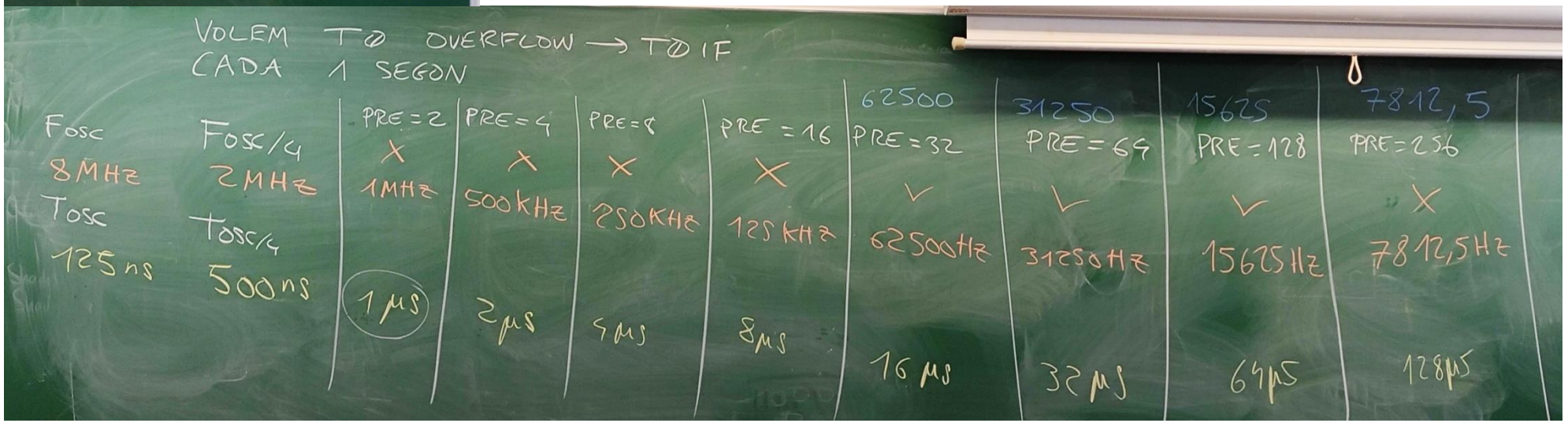
bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

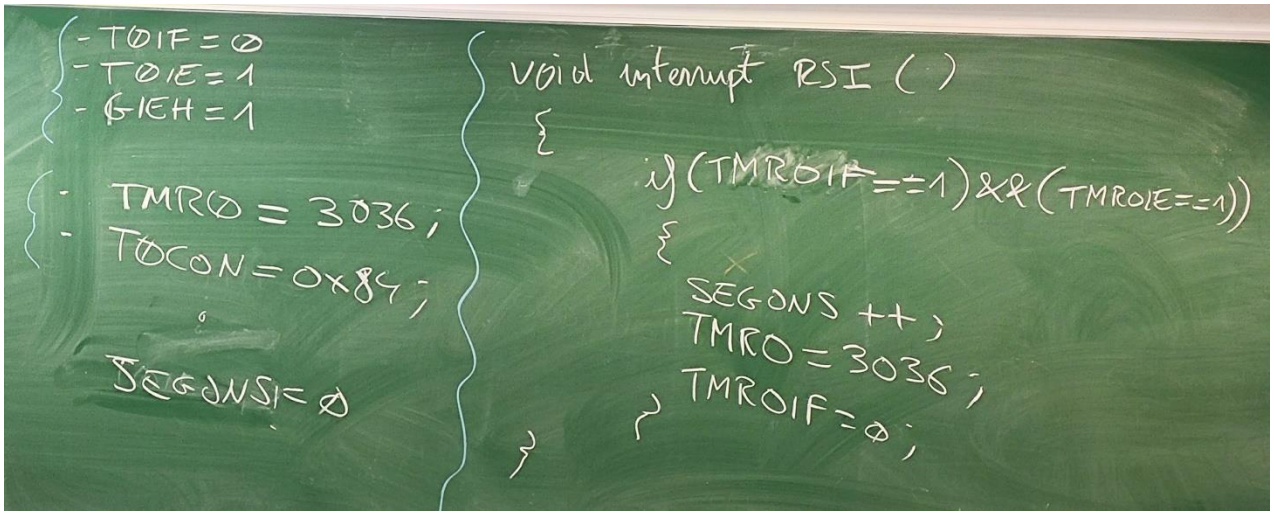
bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS<2:0>**: Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

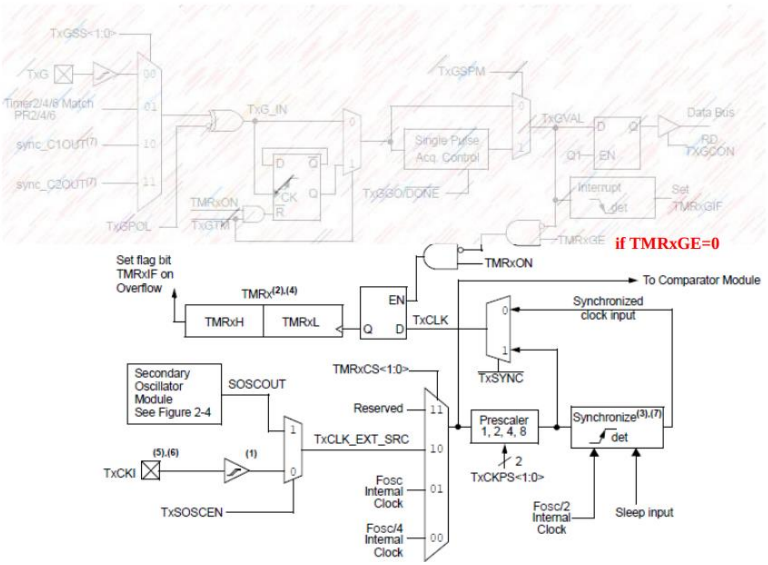






The PIC18 Timer1/3/5

The PIC18 Timer1/3/5



- Is a 16-bit timer/counter depending upon the clock source.

- An interrupt may be requested when TimerX rolls over from 0xFFFF to 0x0000.

- TimerX operation is controlled by the TxCON and TxGCON register.

- These timers have a number of frequency input sources and a complex gate enable circuitry (if TMRxGE=1)

- These timers can be used to create time delays and measure the frequency of an unknown signal (using the CCP modules).

TxCON Register (x=1/3/5)

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/0	R/W-0/u
TMRxCS<1:0>		TxCKPS<1:0>		TxSOSCEN	$\overline{\text{TxSYNC}}$	TxRD16	TMRxON
bit 7							bit 0

bit 7-6 **TMRxCS<1:0>:** Timer1/3/5 Clock Source Select bits
11 = Reserved. Do not use.
10 = Timer1/3/5 clock source is pin or oscillator:
If TxSOSCEN = 0:
External clock from TxCKI pin (on the rising edge)
If TxSOSCEN = 1:
Crystal oscillator on SOSCI/SOSCO pins
01 = Timer1/3/5 clock source is system clock (FOSC)
00 = Timer1/3/5 clock source is instruction clock (FOSC/4)

bit 5-4 **TxCKPS<1:0>:** Timer1/3/5 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **TxSOSCEN:** Secondary Oscillator Enable Control bit
1 = Dedicated Secondary oscillator circuit enabled
0 = Dedicated Secondary oscillator circuit disabled

bit 2 **$\overline{\text{TxSYNC}}$:** Timer1/3/5 External Clock Input Synchronization Control bit
TMRxCS<1:0> = 1X
1 = Do not synchronize external clock input
0 = Synchronize external clock input with system clock (FOSC)

TMRxCS<1:0> = 0X
This bit is ignored. Timer1/3/5 uses the internal clock when TMRxCS<1:0> = 1x.

bit 1 **TxRD16:** 16-Bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer1/3/5 in one 16-bit operation
0 = Enables register read/write of Timer1/3/5 in two 8-bit operation

bit 0 **TMRxON:** Timer1/3/5 On bit
1 = Enables Timer1/3/5
0 = Stops Timer1/3/5
Clears Timer1/3/5 Gate flip-flop

TxGCON Register (x=1/3/5)

REGISTER 12-2: TXGCON: TIMER1/3/5 GATE CONTROL REGISTER

R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W/HC-0/u	R-x/x	R/W-0/u	R/W-0/u
TMRxGE	TxGPOL	TxGTM	TxGSPM	TxGGO/$\overline{\text{DONE}}$	TxGVAL	TxGSS<1:0>	
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	HC = Bit is cleared by hardware

bit 7 **TMRxGE:** Timer1/3/5 Gate Enable bit

TMRxGE=0 !!!

If TMRxON = 0:
This bit is ignored
If TMRxON = 1:
1 = Timer1/3/5 counting is controlled by the Timer1/3/5 gate function
0 = Timer1/3/5 counts regardless of Timer1/3/5 gate function

bit 6 **TxGPOL:** Timer1/3/5 Gate Polarity bit
1 = Timer1/3/5 gate is active-high (Timer1/3/5 counts when gate is high)
0 = Timer1/3/5 gate is active-low (Timer1/3/5 counts when gate is low)

bit 5 **TxGTM:** Timer1/3/5 Gate Toggle Mode bit
1 = Timer1/3/5 Gate Toggle mode is enabled
0 = Timer1/3/5 Gate Toggle mode is disabled and toggle flip-flop is cleared
Timer1/3/5 gate flip-flop toggles on every rising edge.

bit 4 **TxGSPM:** Timer1/3/5 Gate Single-Pulse Mode bit
1 = Timer1/3/5 gate Single-Pulse mode is enabled and is controlling Timer1/3/5 gate
0 = Timer1/3/5 gate Single-Pulse mode is disabled

bit 3 **TxGGO/$\overline{\text{DONE}}$:** Timer1/3/5 Gate Single-Pulse Acquisition Status bit
1 = Timer1/3/5 gate single-pulse acquisition is ready, waiting for an edge
0 = Timer1/3/5 gate single-pulse acquisition has completed or has not been started
This bit is automatically cleared when TxGSPM is cleared.

bit 2 **TxGVAL:** Timer1/3/5 Gate Current State bit
Indicates the current state of the Timer1/3/5 gate that could be provided to TMRxH:TMRxL.
Unaffected by Timer1/3/5 Gate Enable (TMRxGE).

bit 1-0 **TxGSS<1:0>:** Timer1/3/5 Gate Source Select bits
00 = Timer1/3/5 Gate pin
01 = Timer2/4/6 Match PR2/4/6 output (See Table 12-5 for proper timer match selection)
10 = Comparator 1 optionally synchronized output (sync_C1OUT)
11 = Comparator 2 optionally synchronized output (sync_C2OUT)

TxCON Register (x=2/4/6)

REGISTER 13-1: TxCON: TIMER2/TIMER4/TIMER6 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TxOUTPS<3:0>				TMRxON	TxCKPS<1:0>	
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TxOUTPS<3:0>:** TimerX Output Postscaler Select bits
0000 = 1:1 Postscaler
0001 = 1:2 Postscaler
0010 = 1:3 Postscaler
0011 = 1:4 Postscaler
0100 = 1:5 Postscaler
0101 = 1:6 Postscaler
0110 = 1:7 Postscaler
0111 = 1:8 Postscaler
1000 = 1:9 Postscaler
1001 = 1:10 Postscaler
1010 = 1:11 Postscaler
1011 = 1:12 Postscaler
1100 = 1:13 Postscaler
1101 = 1:14 Postscaler
1110 = 1:15 Postscaler
1111 = 1:16 Postscaler

bit 2 **TMRxON:** TimerX On bit
1 = TimerX is on
0 = TimerX is off

bit 1-0 **TxCKPS<1:0>:** Timer2-type Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

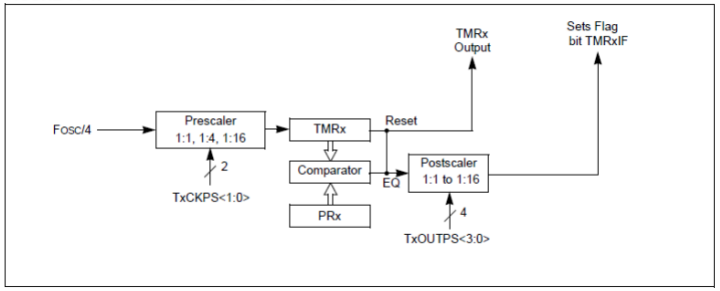
```
// RSI High Priority for handling Timer0
void interrupt RSI() {
   if (INTCONbits.TMR0IF==1 && INTCONbits.TMR0IE==1) {
      tic();
      INTCONbits.TMR0IF=0;
      TMR0=15536;
   }
}
```

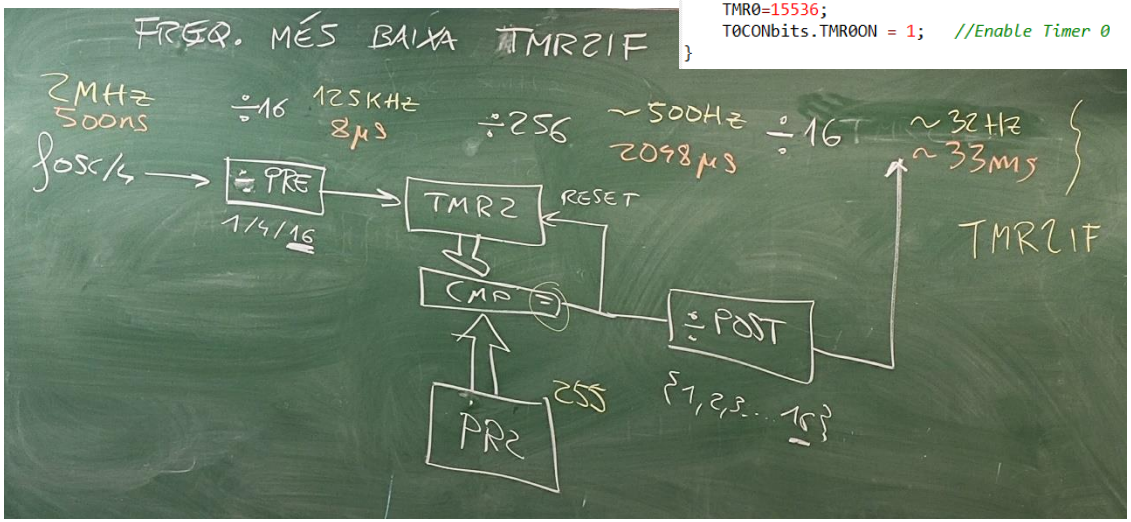
```
// Initialize PORTs, timer0 and basic PIC resources
void configPIC() {
   ANSELA=0x00;
   ANSELB=0x00;
   ANSELC=0x00;
   ANSELD=0x00;

   TRISD=0x00;
   TRISB=0x00;
   TRISA=0x00;
   TRISC=0x01;

   PORTD=0x00;
   PORTB=0x00;

    RCONbits.IPEN = 1;      //Enable priority levels
    INTCONbits.GIEH = 1;    //Enabe high priority interrupts
    INTCONbits.TMR0IF = 0;  //clear IF flag on startup
    INTCONbits.TMR0IE = 1;  //Enable interrupt on timer 0 overflow
    INTCON2bits.TMR0IP = 1;
    T0CONbits.T08BIT = 0;   //Timer 0 as 16 bit counter
    T0CONbits.T0CS = 0;     //Use internal ins. cycle clock (Fosc/4)
    T0CONbits.PSA = 0;      //Prescaler assignet for timer 0
    T0CONbits.T0PS0 = 1;
    T0CONbits.T0PS1 = 0;
    T0CONbits.T0PS2 = 0;
    TMR0=15536;
    T0CONbits.TMR0ON = 1;   //Enable Timer 0
}
```

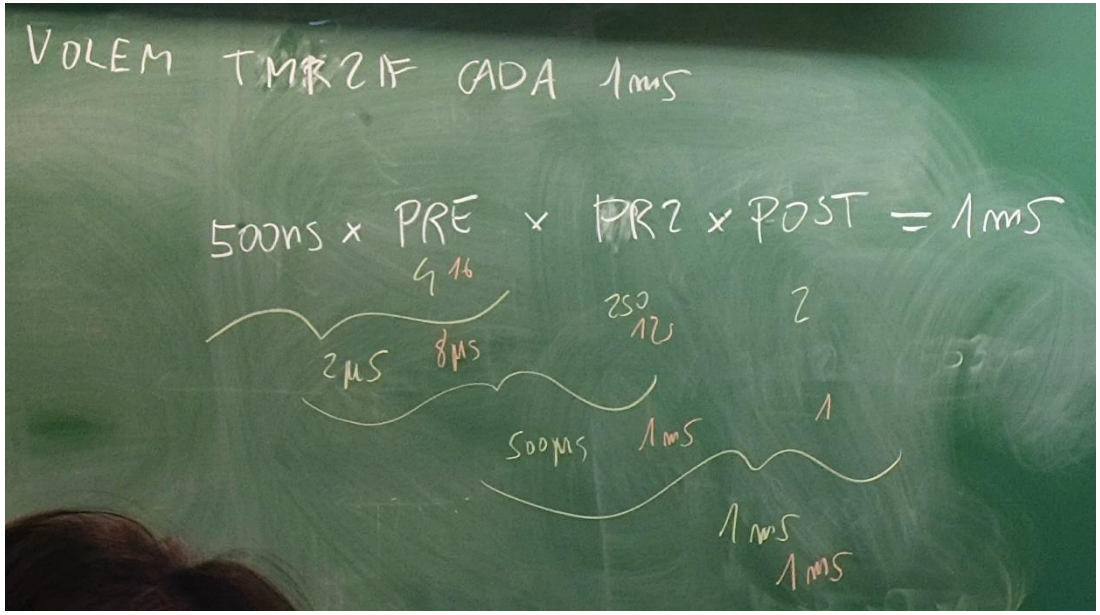
The PIC18 Timer2/4/6

- There are three 8-bit timers with instruction clock source Fcy (Fosc/4) and prescaler and postscaler block logic.
- Controlled by TxCON and related to PRx registers.
- An interrupt may be requested when Timer value matches PRx
- They can be a source for PWM signals (CCP modules).

FIGURE 13-1: TIMER2/4/6 BLOCK DIAGRAM







Capture/Compare/PWM (CCP) Modules

- PIC18F45K22 has 5 CCP Modules.
- Each CCP module requires the use of timer resource.
- In capture or compare mode, the CCP module may use either Timer1, Timer3 or Timer5 to operate.
- In PWM mode, either Timer2, Timer4 or Timer6 may be used.
- The operation of a CCP module is controlled by the CCPxCON register (to select the mode), the CCPTMRS0 and CCPTMRS1 registers (to select the operating timer) and the 16-bits data register CCPRx (CCPRxH and CCPRxL).
- A device pin (CCPx pin) can be associated to CCP module operation (with appropriate TRIS configuration).

The PIC18 CCP units

- A PIC18 device may have one, two, or **five (K22)** CCP modules.
- CCP stands for **Capture**, **Compare**, and **Pulse Width Modulation**.
- Each CCP module can be configured to perform capture, compare, or PWM function.
- In **capture** operation, the CCP module copy the contents of a 16 bit timer into a capture register on a signal edge.
- In **compare** operation, the CCP module compares the contents of a CCPR register with that of a Timer (1, 3 or 5) in every clock cycle. When these two registers are equal, the associated pin may be pulled to high, or low, or toggled.
- In **PWM** mode, the CCP module can be configured to generate a waveform with certain frequency and duty cycle. Timers 2/4/6 are related with the PWM hardware.

CCPxCON Register

14.5 Register Definitions: ECCP Control

REGISTER 14-1: CCPxCON: STANDARD CCPx CONTROL REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB<1:0>		CCPxM<3:0>			
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Reset
'1' = Bit is set	'0' = Bit is cleared	

bit 7-6 **Unused**

bit 5-4 **DCxB<1:0>:** PWM Duty Cycle Least Significant bits

Capture mode:
Unused

Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM<3:0>:** ECCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets the module)
0001 = Reserved
0010 = Compare mode: toggle output on match
0011 = Reserved

0100 = Capture mode: every falling edge
0101 = Capture mode: every rising edge
0110 = Capture mode: every 4th rising edge
0111 = Capture mode: every 16th rising edge

1000 = Compare mode: set output on compare match (CCPx pin is set, CCPxIF is set)
1001 = Compare mode: clear output on compare match (CCPx pin is cleared, CCPxIF is set)
1010 = Compare mode: generate software interrupt on compare match (CCPx pin is unaffected, CCPxIF is set)
1011 = Compare mode: Special Event Trigger (CCPx pin is unaffected, CCPxIF is set)
TimerX (selected by CxTSEL bits) is reset
ADON is set, starting A/D conversion if A/D module is enabled(1)
11xx =: PWM mode

Note 1: This feature is available on CCP5 only.

CCPTMRS0/1 (Timer select) Register

REGISTER 14-3: CCPTMRS0: PWM TIMER SELECTION CONTROL REGISTER 0

R/W-0	R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
C3TSEL<1:0>		—	C2TSEL<1:0>		—	C1TSEL<1:0>	
bit 7							bit 0

bit 7-6 **C3TSEL<1:0>:** CCP3 Timer Selection bits
00 = CCP3 – Capture/Compare modes use Timer1, PWM modes use Timer2
01 = CCP3 – Capture/Compare modes use Timer3, PWM modes use Timer4
10 = CCP3 – Capture/Compare modes use Timer5, PWM modes use Timer6
11 = Reserved

bit 5 **Unused**

bit 4-3 **C2TSEL<1:0>:** CCP2 Timer Selection bits
00 = CCP2 – Capture/Compare modes use Timer1, PWM modes use Timer2
01 = CCP2 – Capture/Compare modes use Timer3, PWM modes use Timer4
10 = CCP2 – Capture/Compare modes use Timer5, PWM modes use Timer6
11 = Reserved

bit 2 **Unused**

bit 1-0 **C1TSEL<1:0>:** CCP1 Timer Selection bits
00 = CCP1 – Capture/Compare modes use Timer1, PWM modes use Timer2
01 = CCP1 – Capture/Compare modes use Timer3, PWM modes use Timer4
10 = CCP1 – Capture/Compare modes use Timer5, PWM modes use Timer6
11 = Reserved

REGISTER 14-4: CCPTMRS1: PWM TIMER SELECTION CONTROL REGISTER 1

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	C5TSEL<1:0>		C4TSEL<1:0>	
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	

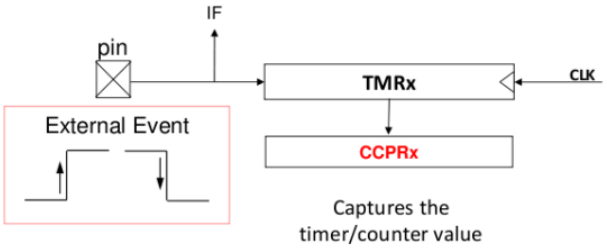
bit 7-4 **Unimplemented:** Read as '0'

bit 3-2 **C5TSEL<1:0>:** CCP5 Timer Selection bits
00 = CCP5 – Capture/Compare modes use Timer1, PWM modes use Timer2
01 = CCP5 – Capture/Compare modes use Timer3, PWM modes use Timer4
10 = CCP5 – Capture/Compare modes use Timer5, PWM modes use Timer6
11 = Reserved

bit 1-0 **C4TSEL<1:0>:** CCP4 Timer Selection bits
00 = CCP4 – Capture/Compare modes use Timer1, PWM modes use Timer2
01 = CCP4 – Capture/Compare modes use Timer3, PWM modes use Timer4
10 = CCP4 – Capture/Compare modes use Timer5, PWM modes use Timer6

Capture Function

Capture Function



Gives a reference on the instant in which the event takes place
(Application: measure the period of a given signal)

Capture Mode

- Main use of CCP is to capture **event** arrival time

- An event is represented by a signal edge.

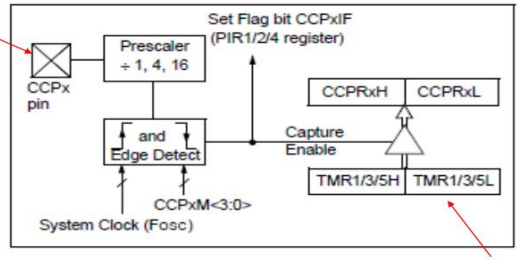
The PIC18 event can be one of the following:

1. every falling edge
2. every rising edge
3. every 4th rising edge
4. every 16th rising edge

Capture Unit

FIGURE 14-1: CAPTURE MODE OPERATION BLOCK DIAGRAM

CCPx Pin must be configured as an Input

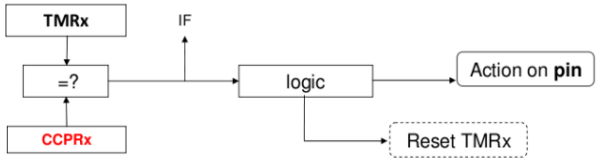


TimerX must be active and configured (prescaler, CLK source) properly.

The five capture units operate independently

Compare Function

Compare Function



Generates equally spaced time intervals
(Application: Delays, Pulse Trains)

Compare Mode

- 16-bit CCPRx register is compared against one of the Timers(1/3/5).
- When they match, one of the following actions may occur on the associated CCPx pin:

1. driven high
2. driven low
3. toggle output
4. remains unchanged

Software Interrupt Mode

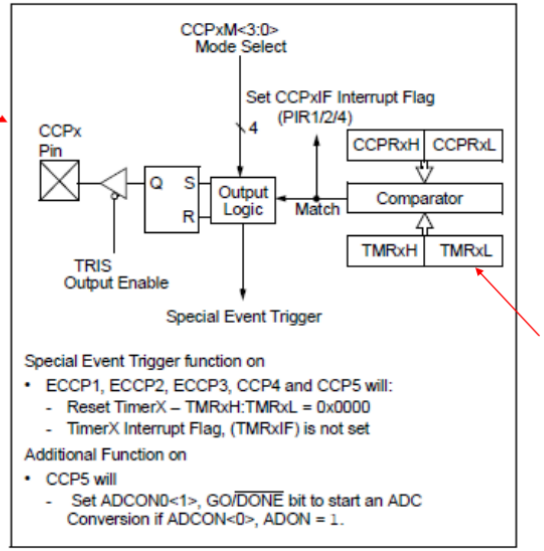
When Generate Software Interrupt mode is chosen (1010), the CCPx module does not assert control of the CCPx pin.

Special Event Trigger

The CCPx modules can also generate this event (mode 1011) to reset TMR1/3/5 depending on which timer is the base timer. When this mode is selected CCP5 will start an ADC conversion, if the ADC is enabled.

Compare Unit

FIGURE 14-2: COMPARE MODE OPERATION BLOCK DIAGRAM



CCPx Pin must be configured as an Output

TimerX must be active and configured (prescaler, CLK source) properly.

Example code:

For the circuit in Figure 8.22, write a program to generate a simple song assuming that f_{OSC} = 4MHz.

Solution:

Two tables are used by the program. 1) Table of numbers to be added to CCPR1 register to generate the waveform with the desired frequency. 2) Table of numbers that select the duration of each note. CCP1 module is used to generate te signal -compare mode toggle CCP1 pin on match-. TIMER0 is used for note duration.

Example XC8

```
#include   <xc.h>
#define    base      3125  // counter count to create 0.1 s delay
#define    NOTES     38    // total notes in the song to be played
#define    C4        0x777 // semioperiod
#define    F4        0x598 // for each
#define    G4        0x4FC // note freq
#define    A4        0x470
#define    B4        0x3F4
#define    C5        0x3BC
#define    D5        0x353
#define    F5        0x2CC

unsigned int half_cycle;

 void  delay (unsigned char xc);

 void interrupt high_ISR(void)
 {
       if (PIE1bits.CCP1IE && PIR1bits.CCP1IF) {
             PIR1bits.CCP1IF = 0;
             CCPR1 += half_cycle;
       }
 }


 void interrupt low_priority low_ISR (void)
 {
 }
```

```
const unsigned int per_arr[38]={ C4,A4,G4,A4,F4,C4,C4,C5,
                                 B4,C5,A4,A4,F4,D5,D5,D5,
                                 C5,A4,C5,C5,B4,A4,B4,C5,
                                 A4,F4,D5,F5,D5,C5,A4,C5,
                                 C5,B4,A4,B4,C5,A4};

const unsigned char wait[38] = { 3,5,3,3,5,3,3,5,
                                 3,3,5,3,3,5,3,3,
                                 5,3,3,3,2,2,3,3,
                                 6,3,5,3,3,5,3,3,
                                 3,2,2,3,3,6};
```

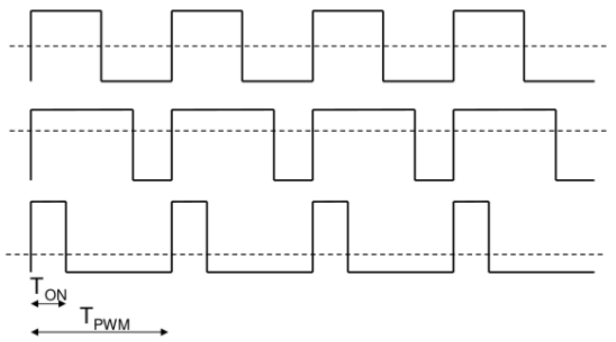
```
void main (void)
{    int i, j;

     TRISCbits.TRISC2 = 0;   /* configure CCP1 pin for output */
     T3CON = 0x81;   /* enables Timer3 in 16-bit mode, Timer1  for CCP1 time base */
     T1CON = 0x81;   /* enable Timer1 in 16-bit mode */
     CCP1CON = 0x02;   /* CCP1 compare mode, pin toggle on match */
     IPR1bits.CCP1IP = 1;   /* set CCP1 interrupt to high priority */
     PIR1bits.CCP1IF = 0;   /* clear CCP1IF flag */
     PIE1bits.CCP1IE = 1;   /* enable CCP1 interrupt */
     INTCON |= 0xC0;       /* enable high priority interrupt */

     for (j = 0; j < 3; j++) {     /* Play song several times */
          i = 0;
          half_cycle = per_arr[0];
          CCPR1 = TMR1 + half_cycle;
          while (i < NOTES) {
               half_cycle = per_arr[i];   /* get the cycle count for half period of the note */
               delay (wait[i]);           /* stay for the duration of the note */
               i++;
          }
          INTCON &= 0x3F;   /* disable interrupt */
          delay (11);               /* Pause before replay song */
          INTCON |= 0xC0;    /* re-enable interrupt */
     }
}
```

```
/* The following function runs on a PIC18 demo board running with a 4 MHz crystal  */
/* oscillator. The parameter xc specifies the amount of delay to be created        */
/* ------------------------------------------------------------------------------*/
void delay (unsigned char xc)
{
      switch (xc){
            case 1:        /* create 0.1 second delay (sixteenth note) */
                  T0CON = 0x84; /* enable TMR0 with prescaler set to 32 */
                  TMR0 = 0xFFFF - base; /* set TMR0 to this value so it overflows in 0.1 second */
                  INTCONbits.TMR0IF = 0;
                  while (!INTCONbits.TMR0IF);
                  break;
            case 2:        /* create 0.2 second delay (eighth note) */
                  T0CON = 0x84; /* set prescaler to Timer0 to 32 */
                  TMR0 = 0xFFFF - 2*base; /* set TMR0 to this value so it overflows in 0.2 second */
                  INTCONbits.TMR0IF = 0;
                  while (!INTCONbits.TMR0IF);
                  break;
            case 3:        /* create 0.4 seconds delay (quarter note) */
                  T0CON = 0x84; /* set prescaler to Timer0 to 32 */
                  TMR0 = 0xFFFF - 4*base; /* set TMR0 to this value so it overflows in 0.4 second */
                  INTCONbits.TMR0IF = 0;
                  while (!INTCONbits.TMR0IF);
                  break;
            case 4:        /* create 0.6 s delay (3 eighths note) */
                  T0CON = 0x84; /* set prescaler to Timer0 to 32 */
                  TMR0 = 0xFFFF - 6*base; /* set TMR0 to this value so it overflows in 0.6 second */
                  INTCONbits.TMR0IF = 0;
                  while (!INTCONbits.TMR0IF);
                  break;
            case 5:        /* create 1.2 second delay (3 quarter note) */
                  T0CON = 0x84; /* set prescaler to Timer0 to 32 */
                  TMR0 = 0xFFFF - 12*base; /* set TMR0 to this value so it overflows in 1.2 second */
                  INTCONbits.TMR0IF = 0;
                  while (!INTCONbits.TMR0IF);
                  break;
            case 6:        /* create 1.6 second delay (full note) */
                  T0CON = 0x84; /* set prescaler to Timer0 to 32 */
                  TMR0 = 0xFFFF - 16*base; /* set TMR0 to this value so it overflows in 1.6 second */
                  INTCONbits.TMR0IF = 0;
                  while (!INTCONbits.TMR0IF);
                  break;

            default:
            break;
      }
}
```

PWM function

Pulse Width Modulation (PWM) Function

Generates (automatically) a PWM signal



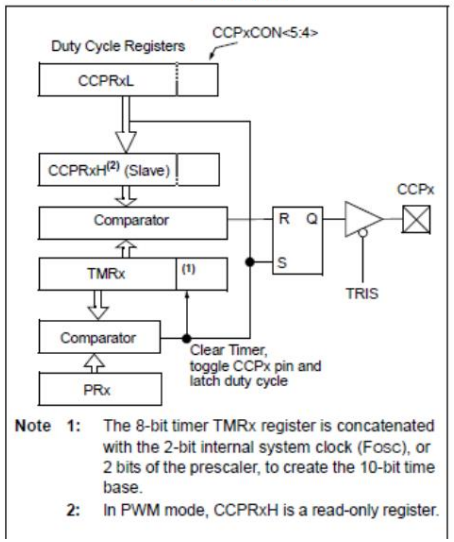
Period = T_{PWM}

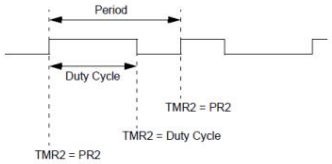
Frecuency, $F_{PWM} = 1/T_{PWM}$

Duty Cycle = $(T_{ON} / T_{PWM}) \times 100$

PWM Mode

FIGURE 14-4: SIMPLIFIED PWM BLOCK DIAGRAM





Note 1: The 8-bit timer TMRx register is concatenated with the 2-bit internal system clock (Fosc), or 2 bits of the prescaler, to create the 10-bit time base.
2: In PWM mode, CCPRxH is a read-only register.

Period & Duty cycle

Pwm formulas:

EQUATION 14-1: PWM PERIOD

$$PWM\ Period = [(PRx) + 1] \cdot 4 \cdot Tosc \cdot (TMRx\ Prescale\ Value)$$

Note 1: Tosc = 1/Fosc

EQUATION 14-2: PULSE WIDTH

$$Pulse\ Width = (CCPRxL{:}CCPxCON{<}5{:}4{>}) \cdot Tosc \cdot (TMRx\ Prescale\ Value)$$

EQUATION 14-4: PWM RESOLUTION

$$Resolution = \frac{\log[4(PRx + 1)]}{\log(2)}\ bits$$

EQUATION 14-3: DUTY CYCLE RATIO

$$Duty\ Cycle\ Ratio = \frac{(CCPRxL{:}CCPxCON{<}5{:}4{>})}{4(PRx + 1)}$$

TABLE 14-7: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (FOSC = 32 MHz)

PWM Frequency	1.95 kHz	7.81 kHz	31.25 kHz	125 kHz	250 kHz	333.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PRx Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 14-8: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (FOSC = 20 MHz)

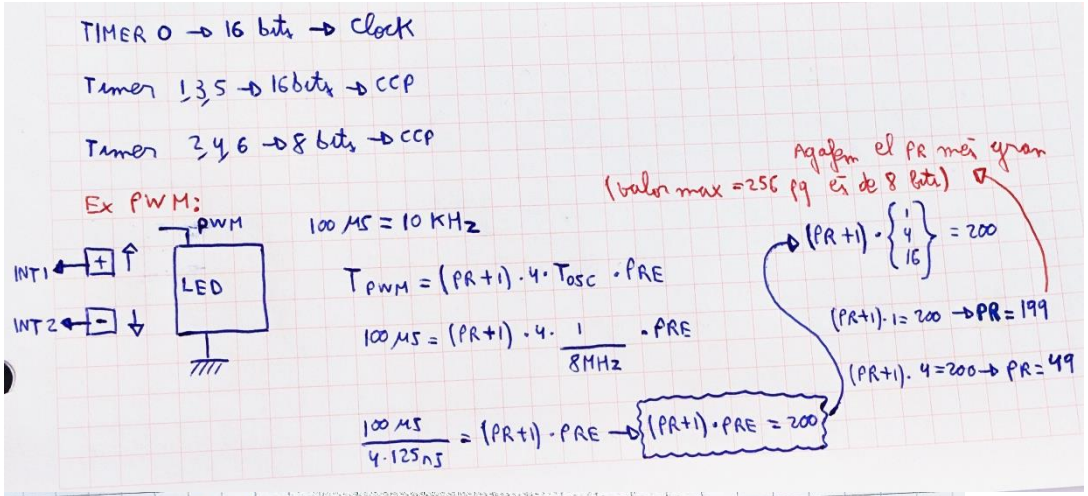
PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PRx Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

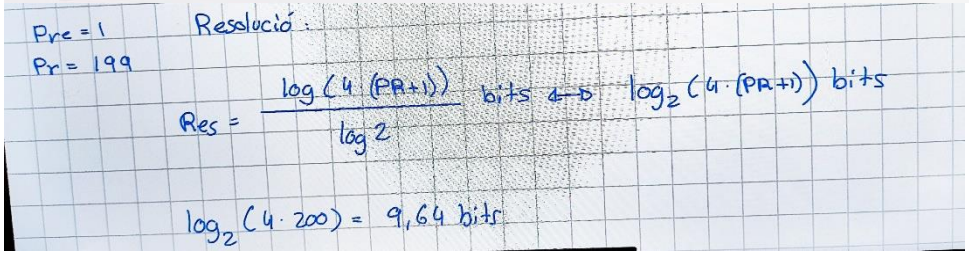
TABLE 14-9: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (FOSC = 8 MHz)

PWM Frequency	1.22 kHz	4.90 kHz	19.61 kHz	76.92 kHz	153.85 kHz	200.0 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PRx Value	0x65	0x65	0x65	0x19	0x0C	0x09
Maximum Resolution (bits)	8	8	8	6	5	5

Programming recipe.

1. Disable the CCPx pin output driver by setting the associated TRIS bit.
2. Select the 8-bit TimerX resource, (Timer2, Timer4 or Timer6) to be used for PWM generation by setting the CxTSEL<1:0> bits in the CCPTMRSx register.(1)
3. Load the PRx register for the selected TimerX with the PWM period value.
4. Configure the CCP module for the PWM mode by loading the CCPxCON register with appropriate values.
5. Load the CCPRxL register and the DCxB<1:0> bits of the CCPxCON register, with the PWM duty cycle value.
6. Configure and start the 8-bit TimerX resource:
   - Clear the TMRxIF interrupt flag bit of the PIR2 or PIR4 register.
   - Configure the TxCKPS bits of the TxCON register with the Timer prescale value.
   - Enable the Timer by setting the TMRxON bit of the TxCON register.
7. Enable PWM output pin:
   - Wait until the Timer overflows and the TMRxIF bit of the PIR2 or PIR4 register is set.
   - Enable the CCPx pin output driver by clearing the associated TRIS bit.

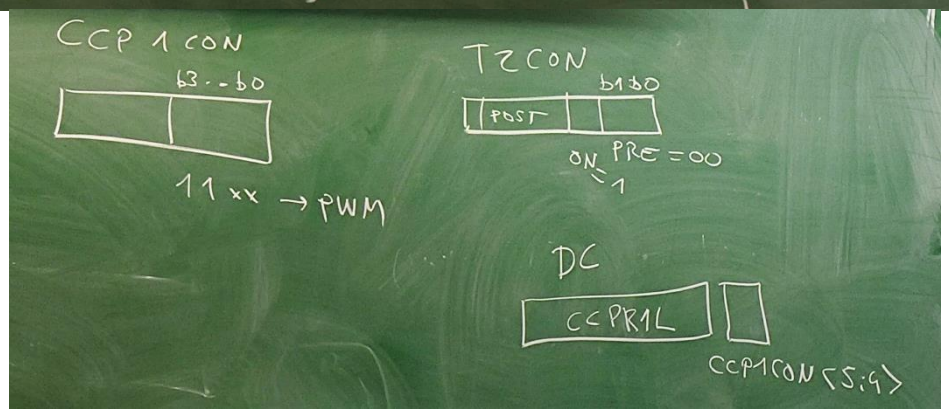




CCP1, TMR2

```
Config_PWM()
{
    CCPTMRS0 &= 0xFC;  // 1111 1100
    CCP1CON |= 0x0C;   // 0000 1100
    T2CON = 0x05;      // ON, PRE = 1
    PR = 199;
    Escrim_DC(400);
}
```

```
Void interrupt RSI()
{
    if ((INT1IF==1) && (INT1IE==1))
    {
        DC = DC+1;
        if (DC>800) DC = 800;
        EscrimDC(DC);
        INT1IF = 0;
    }
    if ((INT2IF==1) &&
        (INT2IE==1))
    {
        DC = DC-1;
        if (DC<0) DC = 0;
        EscrimDC(DC);
        INT2IF = 0
    }
}
```




```
main()
{
    ConfigBotons();
    ConfigPWM();
    ...
    while(1)
    {
    }
}
```

```
Void ConfigBotons()
{
    TRISB |= 0x06;
    INT1IE = 1;
    INT2IE = 1;
    INT1IP = 0;
    INT2IP = 0;
    IPEN = 1;
    GIEL = 1;
    GIEH = 1;
}
```

```
Void EscrimDC(int x)
{
    baix = x%4;
    alt = x/4;
    CCPR1L = alt;
    CCP1CON &= 0xCF;  // 11 00 1111
    CCP1CON += baix<<4;
}
```

DC Motor Control

- DC motor speed is regulated by controlling its average driving voltage. The higher the voltage, the faster the motor rotates.
- Changing motor direction can be achieved by reversing the driving voltage.
- Motor braking can be performed by reversing the driving voltage for certain length of time.
- Most PIC18 devices have PWM functions that can be used to drive DC motors.
- Many DC motors operate with 5 V supply.
- DC motors require large amount of current to operate. Special driver circuits are needed for this purpose.
- A simplified DC motor control circuit is shown in Figure 8.26.

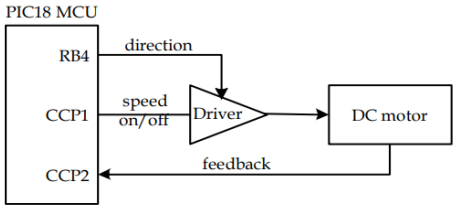
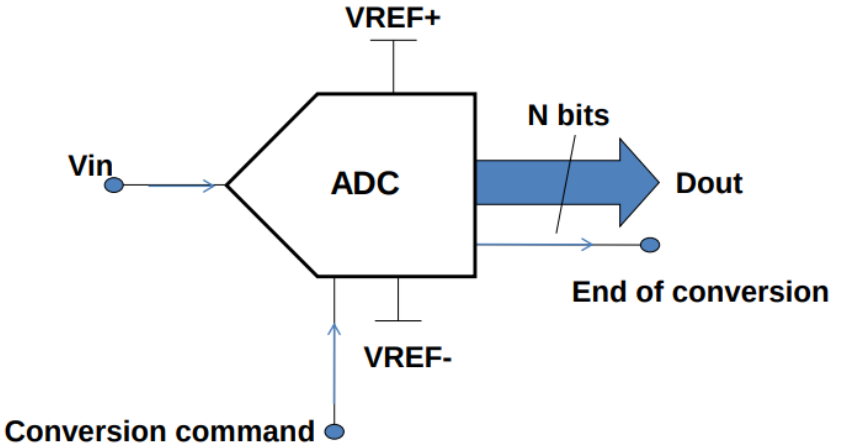


Figure 8.26A simplified circuit for DC motor control

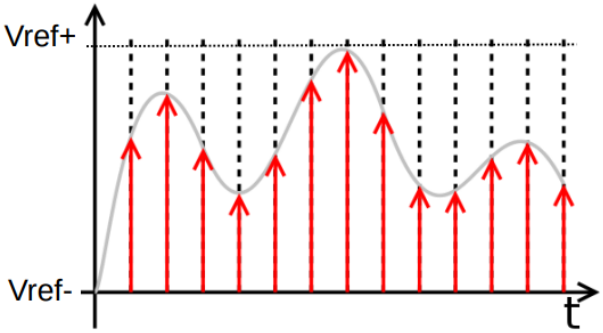
```
void interrupt RSI() {
   if (PIR1bits.TMR2IF && PIE1bits.TMR2IE)
   {
      PIR1bits.TMR2IF = 0;

   }
}

void Escriu_DC(int x)
{
   int baix = x%4;
   int alt = x/4;
   CCPR3L = alt;
   CCP3CON &= 0xCF;
   CCP3CON += baix << 4;
}

void setupCCP3Module(void)
{
   CCPTMRS0 = 0x3F;      //PWD working with Timer 2
   CCP3CON |= 0x0C;      //CCP1 on PWM mode
   PR2 = 249;
   T2CON = 0x07;
   Escriu_DC(250);
}
```

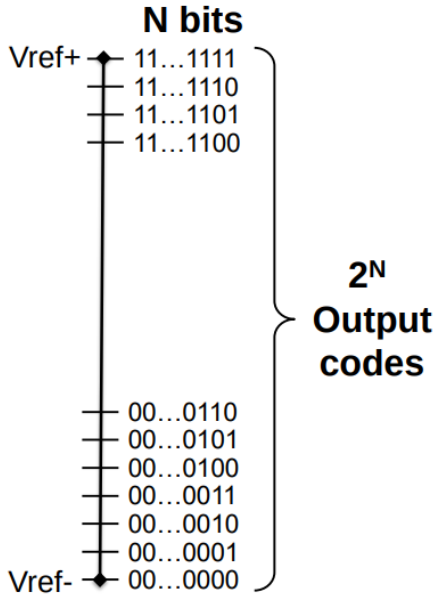
Analog to digital converter concept (ADC)

ADC Scheme (with a synchrony signal *End of conversion*)



Analog versus Digital



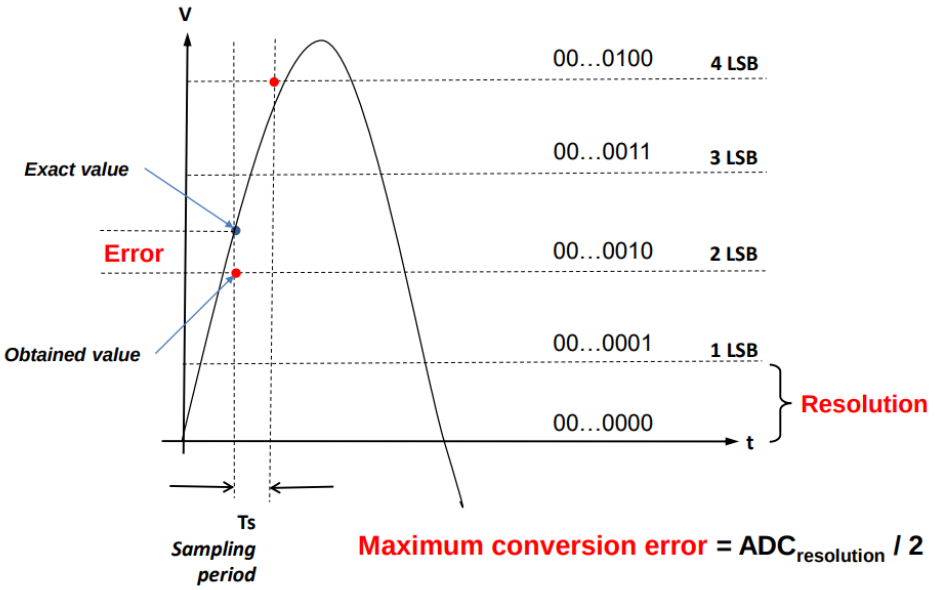


Resolution of ADC = $(V_{ref+} - V_{ref-})/(2^N - 1)$

¿Digital codes obtained represent EXACTLY the original analog value?
… NO! There's an ERROR

$$Dout = round\left[(2^N - 1) \cdot \frac{(V_{IN} - V_{REF-})}{(V_{REF+} - V_{REF-})}\right]$$

Quantization error: ± 0.5 LSB



Maximum conversion error = $ADC_{resolution}$ / 2

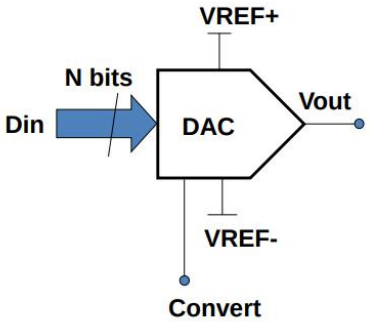
Average conversion error = 0

Devices

1. **ADC:** A/D Flash Converter
2. **DAC:** Digital to Analog Converter *
3. **ADC:** Slope A/D Converter
4. **ADC:** Successive approximations A/D Converter *

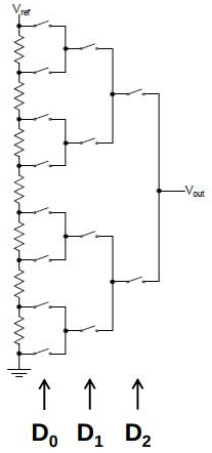
2. Digital to Analog Converter (DAC)

Symbol



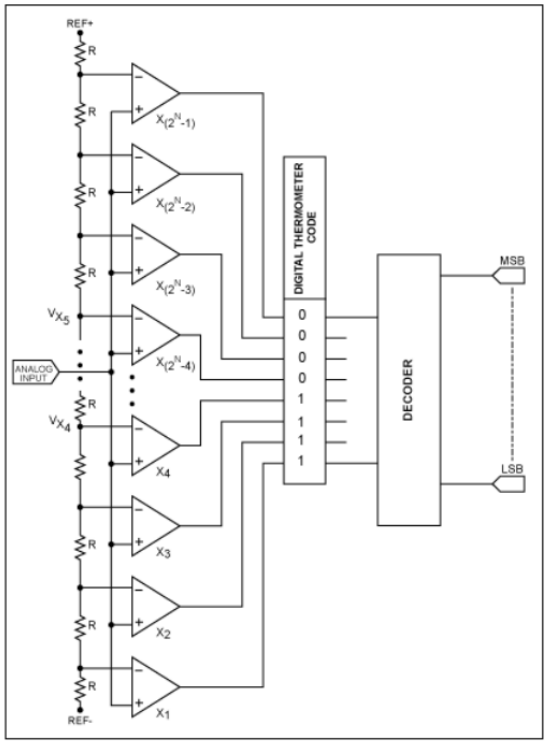
Possible implementation:

String Resistor ladder

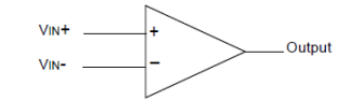


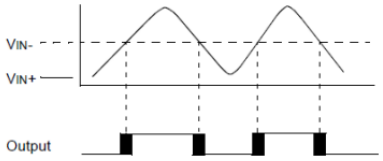
$$Vout = V_{REF-} + (V_{REF+} - V_{REF-}) \cdot \frac{D_{in}}{(2^N - 1)}$$

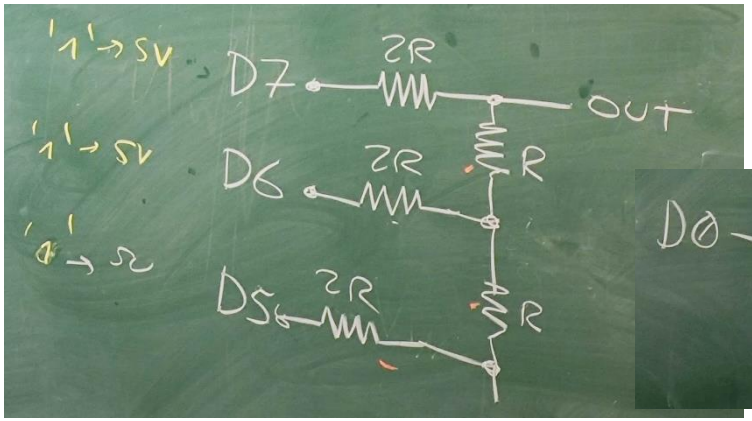
1. A/D Flash

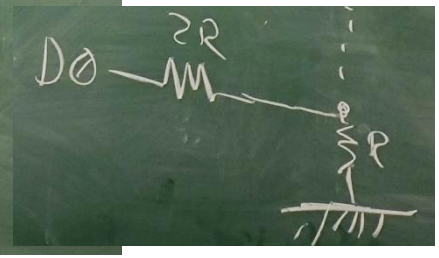


Comparator

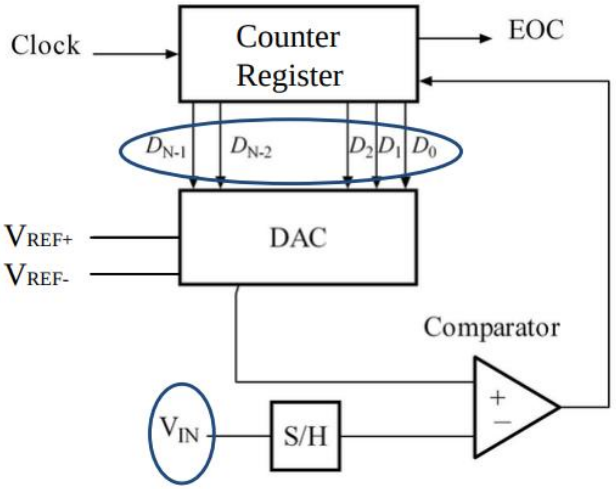


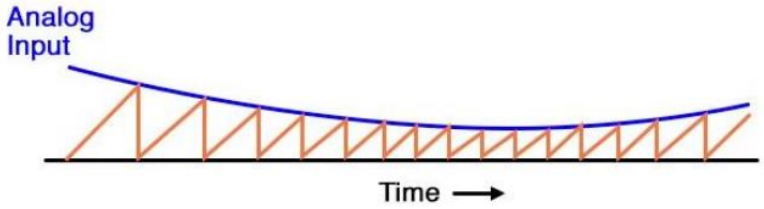




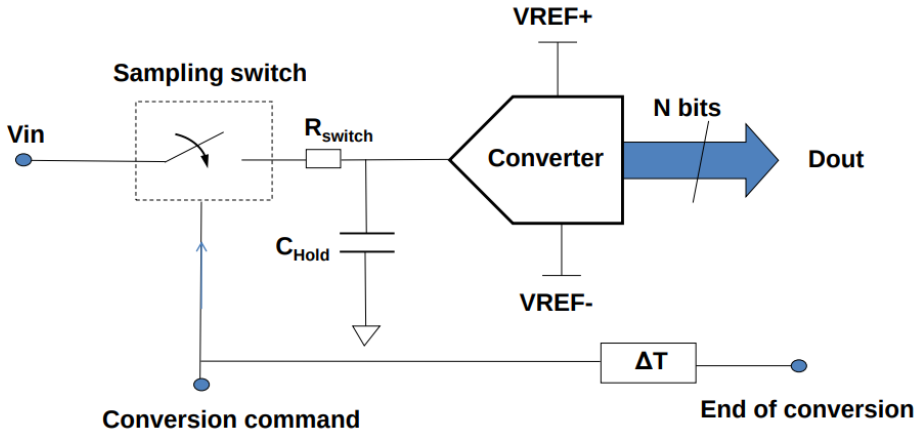


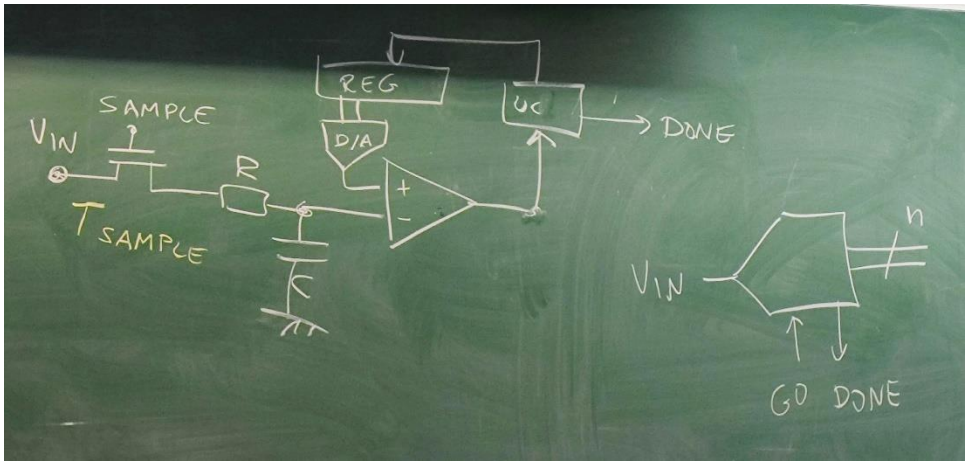
3. Slope A/D converter



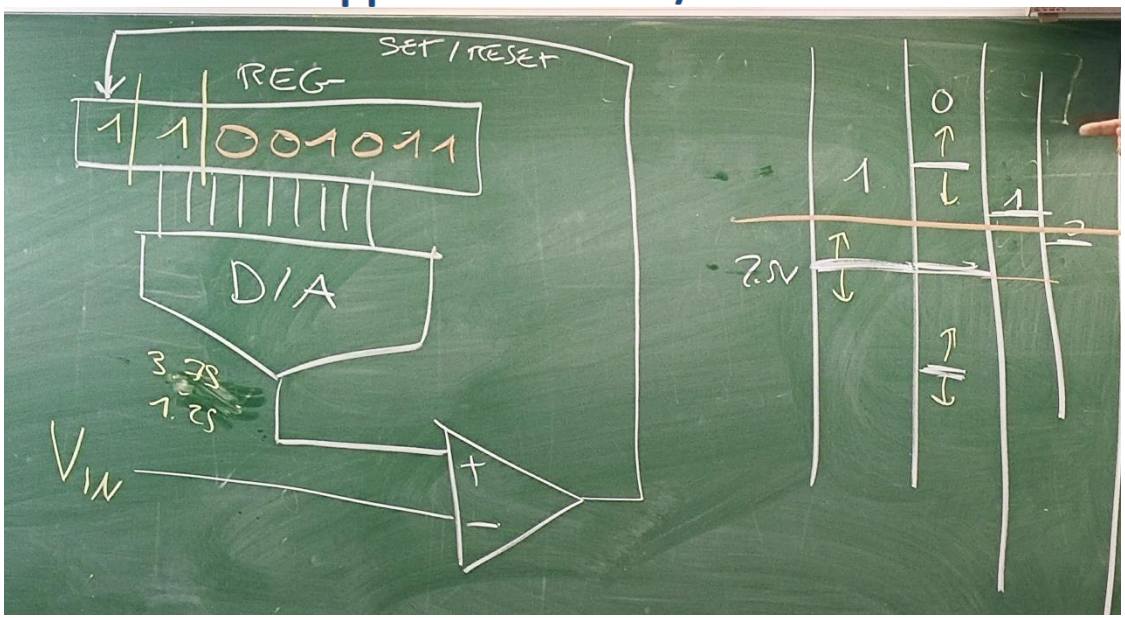


ADC: sample and hold + converter

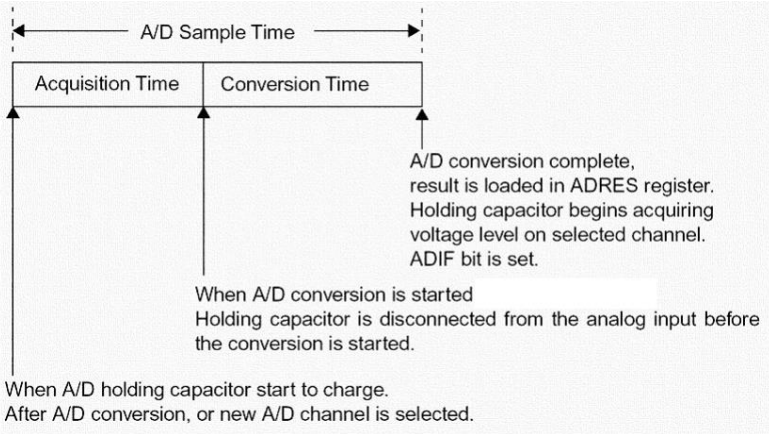




4. Successive approximations A/D Converter

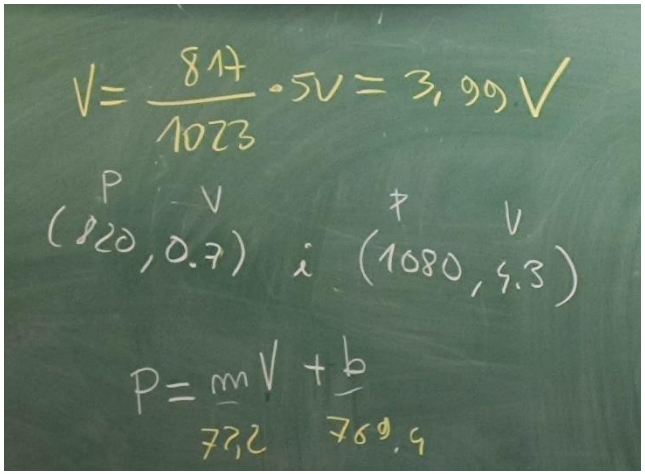


Acquisition & Conversion times



1080 = 72,2*4,3 + b

820 = 72,2*0,7 +b

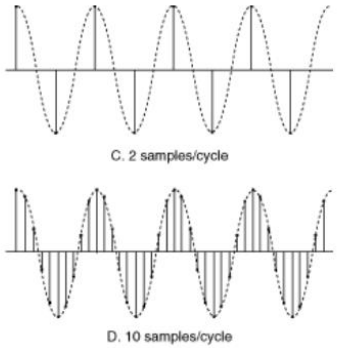


Nyquist criterion

To avoid aliasing

$f_{sampling} > 2 \cdot f_{signal\ max\ frequency}$

Nevertheless, higher sampling frequencies, higher quality



FM = 10*FS

Data Collection – Sampling Rate

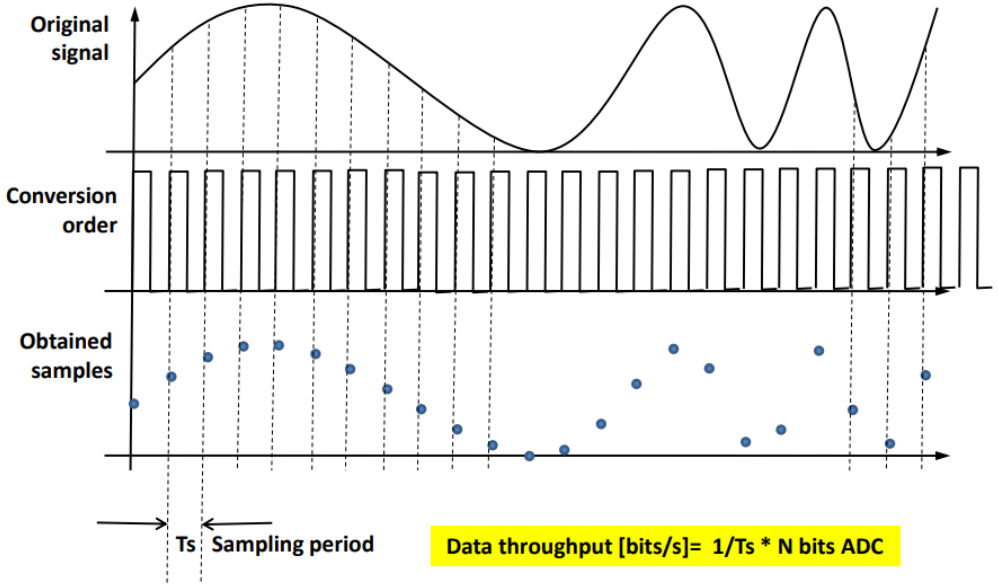
The Nyquist Rate

A signal must be sampled at a rate at least twice that of the highest frequency component that must be reproduced.

Example – Hi-Fi sound (20-20,000 Hz) is generally sampled at about 44 kHz.

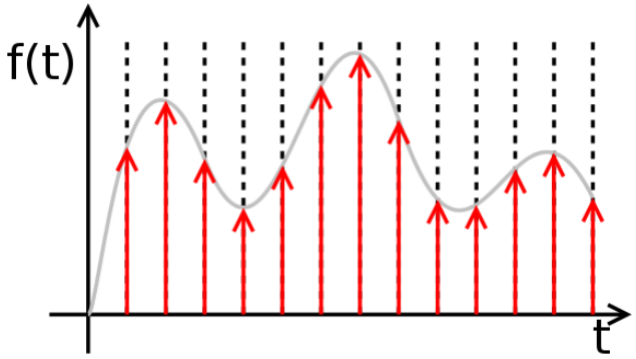
External temperature during flight need only be sampled every few seconds at most.

Sampling (time)

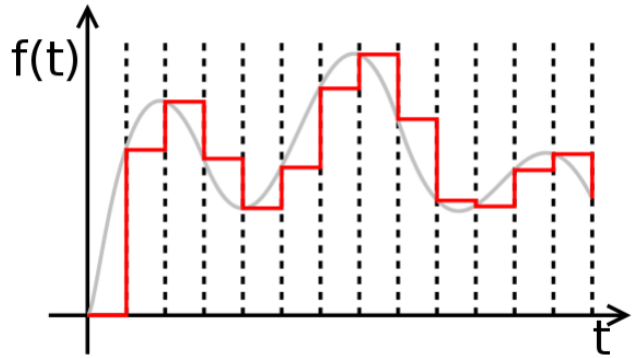


Data throughput [bits/s]= 1/Ts * N bits ADC

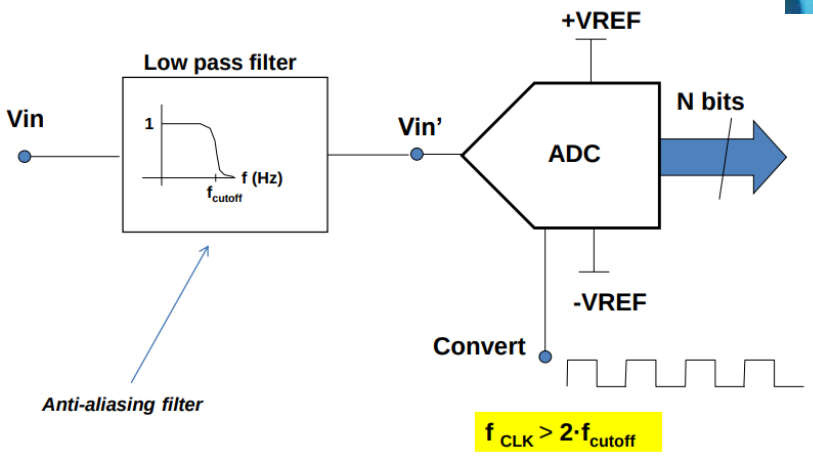
Original signal & Sampling

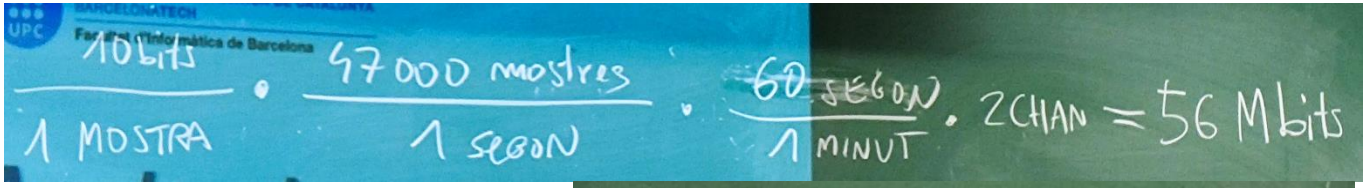


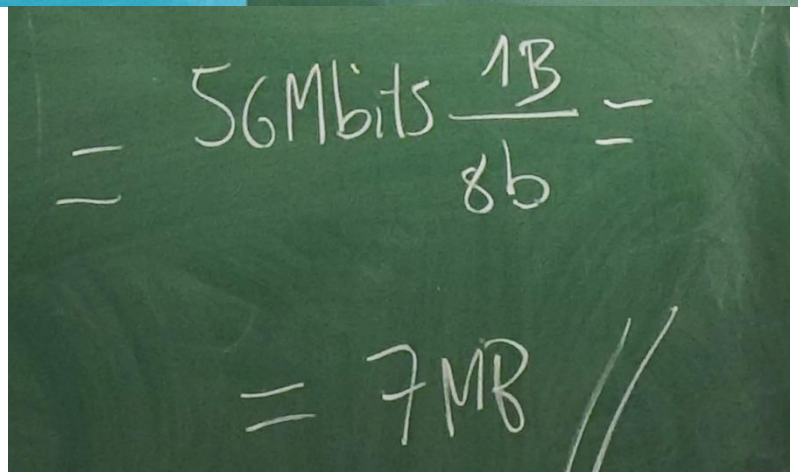
Original signal vs Sampled version



Anti-aliasing filter



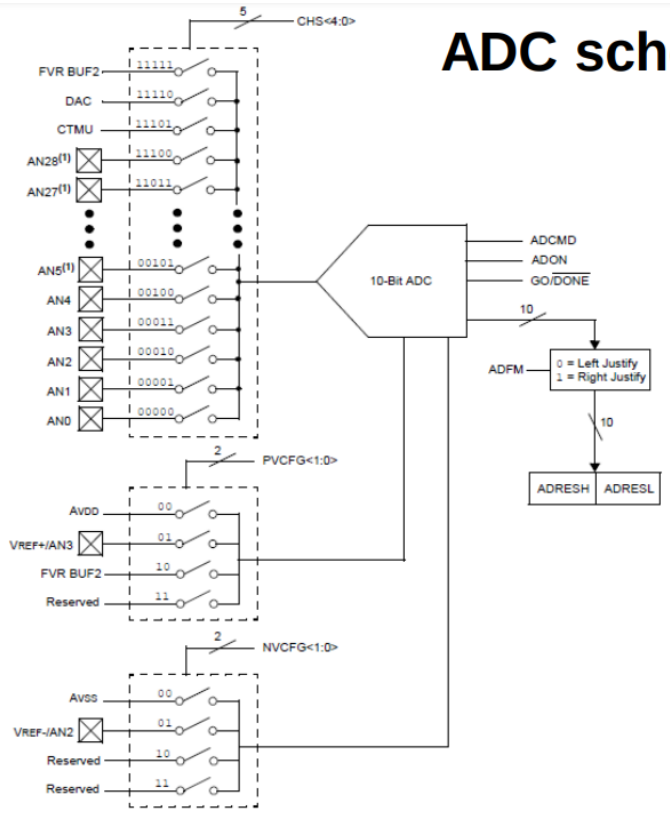




The PIC18 A/D Converter

- The PIC18 has a 10-bit A/D Successive Approximations converter.
- The number of analog inputs varies among different PIC18 devices.
- The A/D converter has the following registers:
  - A/D Result High Register (ADRESH)
  - A/D Result Low Register (ADRESL)
  - A/D Control Register 0 (ADCON0) (source selection)
  - A/D Control Register 1 (ADCON1) (reference selection)
  - A/D Control Register 2 (ADCON2) (timing selections)
- The contents of these registers vary with the PIC18 members.
- Other registers must be considered:
  ANSELx (pin configurations),
  ADIF, ADIE, ADIP (for AD interrupt)

ADC schematic



REGISTER 17-1: ADCON0: A/D CONTROL REGISTER 0

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CHS<4:0>					GO/DONE	ADON
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-2 **CHS<4:0>: Analog Channel Select bits**
00000 = AN0
00001 = AN1
00010 = AN2
00011 = AN3
00100 = AN4
00101 = AN5(1)
00110 = AN6(1)
00111 = AN7(1)
01000 = AN8
01001 = AN9
01010 = AN10
01011 = AN11
01100 = AN12
01101 = AN13
01110 = AN14
01111 = AN15
10000 = AN16
10001 = AN17
10010 = AN18
10011 = AN19
10100 = AN20(1)
10101 = AN21(1)
10110 = AN22(1)
10111 = AN23(1)
11000 = AN24(1)
11001 = AN25(1)
11010 = AN26(1)
11011 = AN27(1)
11100 = Reserved
11101 = CTMU
11110 = DAC
11111 = FVR BUF2 (1.024V/2.048V/2.096V Volt Fixed Voltage Reference)(2)

bit 1 GO/DONE: A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed/not in progress

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

Note 1: Available on PIC18(L)F4XK22 devices only.
2: Allow greater than 15 µs acquisition time when measuring the Fixed Voltage Reference.

ADCON0 Register

ADCON2 Register

REGISTER 17-3: ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT<2:0>			ADCS<2:0>		
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 **ADFM:** A/D Conversion Result Format Select bit
1 = Right justified
0 = Left justified

10 bit data result format!

bit 6 **Unimplemented:** Read as '0'

bit 5-3 **ACQT<2:0>:** A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/DONE bit is set until conversions begins.
000 = 0(1)
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

Acquisition time for AD

bit 2-0 **ADCS<2:0>:** A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC(1) (clock derived from a dedicated internal oscillator = 600 kHz nominal)

Base clock for AD

Note 1: When the A/D clock source is selected as FRC then the start of conversion is delayed by one instruction cycle after the GO/DONE bit is set to allow the SLEEP instruction to be executed.

ADCON1 Register

REGISTER 17-2: ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
TRIGSEL	—	—	—	PVCFG<1:0>		NVCFG<1:0>	
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

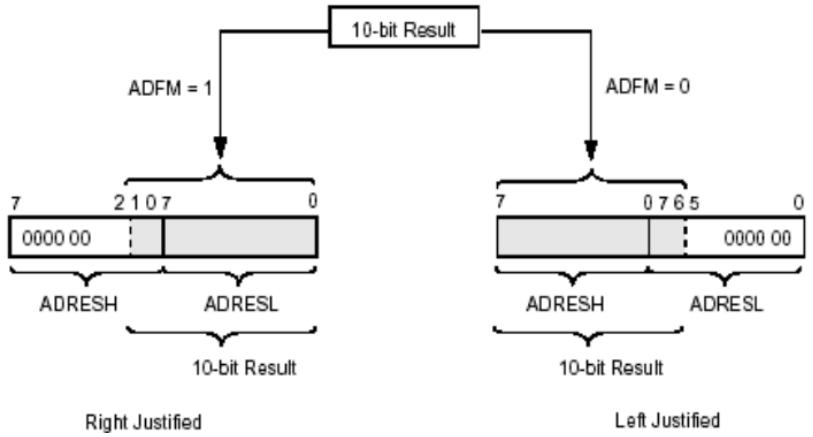
bit 7 **TRIGSEL**: Special Trigger Select bit
1 = Selects the special trigger from CTMU
0 = Selects the special trigger from CCP5

bit 6-4 **Unimplemented:** Read as '0'

bit 3-2 **PVCFG<1:0>:** Positive Voltage Reference Configuration bits
00 = A/D VREF+ connected to internal signal, AVDD
01 = A/D VREF+ connected to external pin, VREF+
10 = A/D VREF+ connected to internal signal, FVR BUF2
11 = Reserved (by default, A/D VREF+ connected to internal signal, AVDD)

bit 1-0 **NVCFG<1:0>:** Negative Voltage Reference Configuration bits
00 = A/D VREF- connected to internal signal, AVSS
01 = A/D VREF- connected to external pin, VREF-
10 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)
11 = Reserved (by default, A/D VREF- connected to internal signal, AVSS)

Result format



- The capacitor C_{HOLD} holds the voltage to be converted. It must be charged to a stable value in order to get the maximum precision.
- The required minimum acquisition time T_{ACQ} is computed as follows:

EQUATION 17-1: ACQUISITION TIME EXAMPLE

Assumptions: *Temperature = 50°C and external impedance of 10kΩ 3.0V VDD*

$$TACQ = Amplifier\ Settling\ Time + Hold\ Capacitor\ Charging\ Time + Temperature\ Coefficient$$
$$= TAMP + TC + TCOFF$$
$$= 5\mu s + TC + [(Temperature - 25°C)(0.05\mu s/°C)]$$

The value for TC can be approximated with the following equations:

$$V_{APPLIED}\left(1 - \frac{1}{2047}\right) = V_{CHOLD} \quad ;[1]\ V_{CHOLD}\ charged\ to\ within\ 1/2\ lsb$$

$$V_{APPLIED}\left(1 - e^{\frac{-Tc}{RC}}\right) = V_{CHOLD} \quad ;[2]\ V_{CHOLD}\ charge\ response\ to\ V_{APPLIED}$$

$$V_{APPLIED}\left(1 - e^{\frac{-Tc}{RC}}\right) = V_{APPLIED}\left(1 - \frac{1}{2047}\right) \quad ;combining\ [1]\ and\ [2]$$

Solving for TC:

$$TC = -C_{HOLD}(R_{IC} + R_{SS} + R_S)\ ln(1/2047)$$
$$= -13.5pF(1k\Omega + 700\Omega + 10k\Omega)\ ln(0.0004885)$$
$$= 1.20\mu s$$

Therefore:

$$TACQ = 5\mu s + 1.20\mu s + [(50°C - 25°C)(0.05\mu s/°C)]$$
$$= 7.45\mu s$$

Other Time Requirements

TABLE 27-22: A/D CONVERSION REQUIREMENTS PIC18(L)F2X/4XK22

Standard Operating Conditions (unless otherwise stated) Operating temperature Tested at +25°C							
Param. No.	**Symbol**	**Characteristic**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
130	TAD	A/D Clock Period	1	—	25	µs	-40°C to +85°C
			1	—	4	µs	+85°C to +125°C
131	TCNV	Conversion Time (not including acquisition time) **(Note 1)**	11	—	11	TAD	
132	TACQ	Acquisition Time **(Note 2)**	1.4	—	—	µs	VDD = 3V, Rs = 50Ω
135	TSWC	Switching Time from Convert → Sample	—	—	**(Note 3)**		
136	TDIS	Discharge Time	1	—	1	TCY	

Note 1: ADRES register may be read on the following TCY cycle.

2: The time for the holding capacitor to acquire the "New" input voltage when the voltage changes full scale after the conversion (VDD to VSS or VSS to VDD). The source impedance (*Rs*) on the input channels is 50 Ω.

3: On the following cycle of the device clock.

T_{AD} ≥ 1 µs

T_{ACQ} ≥ 1.4 µs (but T_{ACQ} ≥ 7.45 µs in the example)

T_{CNV} = 11, we need 11 T_{AD} for conversion.

T_{DIS} = 1 T_{CY} cycle

Procedure for Performing A/D Conversion

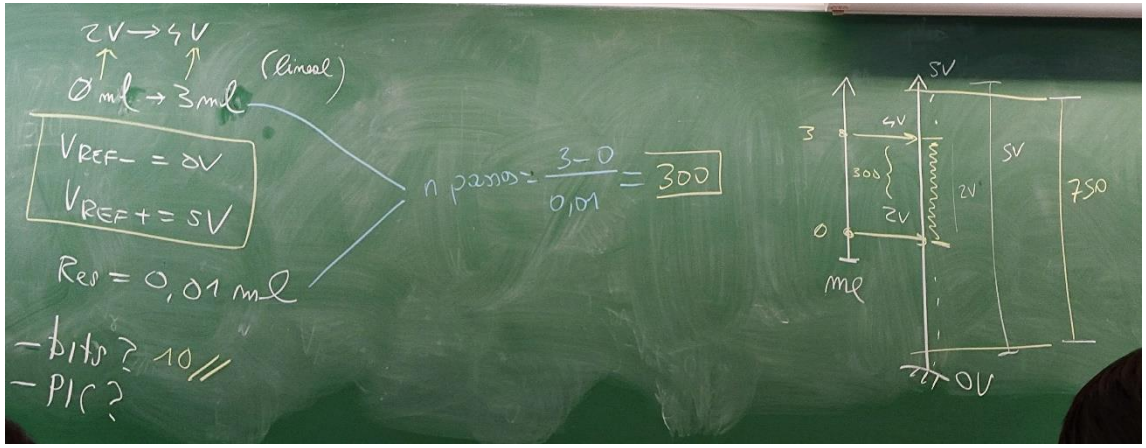
17.2.10 A/D CONVERSION PROCEDURE

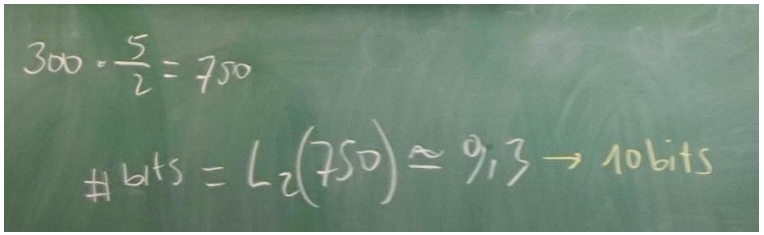
This is an example procedure for using the ADC to perform an Analog-to-Digital conversion:

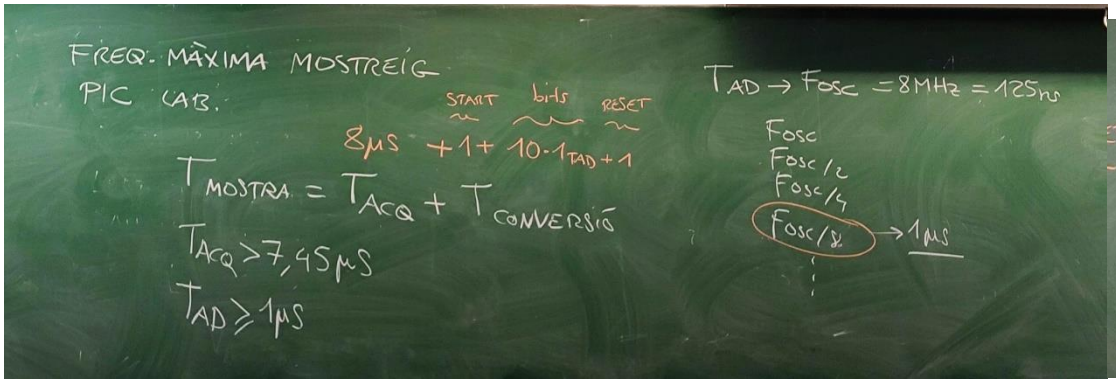
1. Configure Port:
   - Disable pin output driver (See TRIS register)
   - Configure pin as analog
2. Configure the ADC module:
   - Select ADC conversion clock
   - Configure voltage reference
   - Select ADC input channel
   - Select result format
   - Select acquisition delay
   - Turn on ADC module
3. Configure ADC interrupt (optional):
   - Clear ADC interrupt flag
   - Enable ADC interrupt
   - Enable peripheral interrupt
   - Enable global interrupt(1)
4. Wait the required acquisition time(2).
5. Start conversion by setting the GO/$\overline{DONE}$ bit.
6. Wait for ADC conversion to complete by one of the following:
   - Polling the GO/$\overline{DONE}$ bit
   - Waiting for the ADC interrupt (interrupts enabled)
7. Read ADC Result
8. Clear the ADC interrupt flag (required if interrupt is enabled).

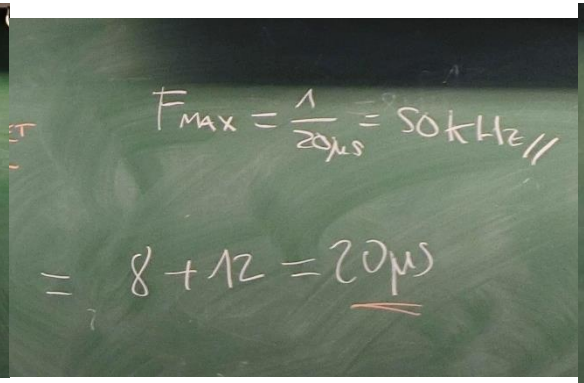
EXAMPLE 17-1: A/D CONVERSION

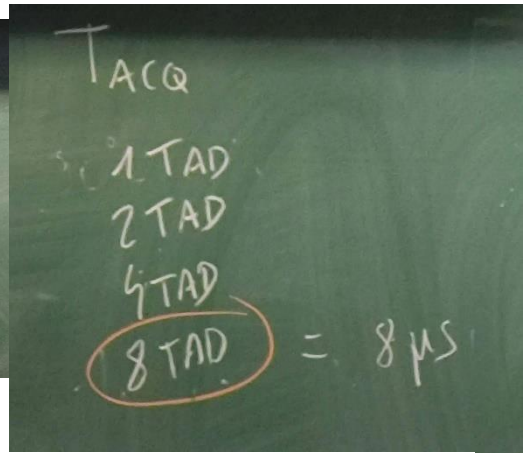
```
;This code block configures the ADC
;for polling, Vdd and Vss as reference, Frc
clock and AN0 input.
;
;Conversion start & polling for completion
; are included.
;
MOVLW     B'10101111'  ;right justify, Frc,
MOVWF     ADCON2       ; & 12 TAD ACQ time
MOVLW     B'00000000'  ;ADC ref = Vdd,Vss
MOVWF     ADCON1       ;
BSF       TRISA,0      ;Set RA0 to input
BSF       ANSEL,0      ;Set RA0 to analog
MOVLW     B'00000001'  ;AN0, ADC on
MOVWF     ADCON0       ;
BSF       ADCON0,GO    ;Start conversion
ADCPoll:
BTFSC     ADCON0,GO    ;Is conversion done?
BRA       ADCPoll      ;No, test again
; Result is complete - store 2 MSbits in
; RESULTHI and 8 LSbits in RESULTLO
MOVFF     ADRESH,RESULTHI
MOVFF     ADRESL,RESULTLO
```





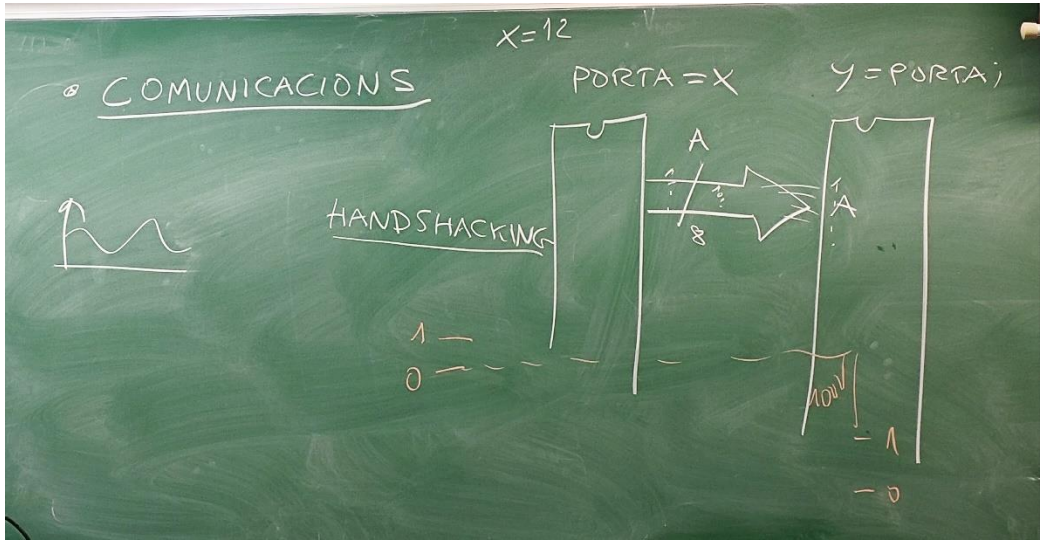


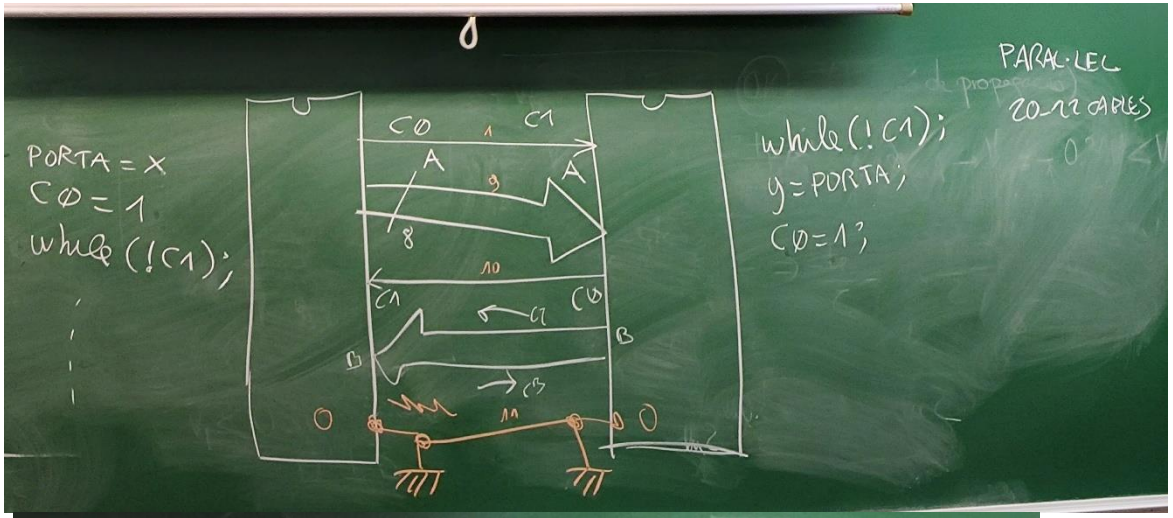


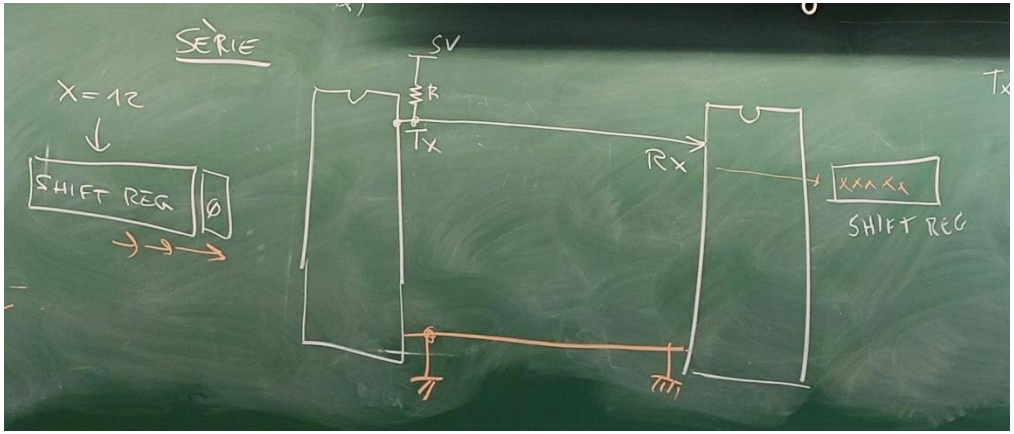


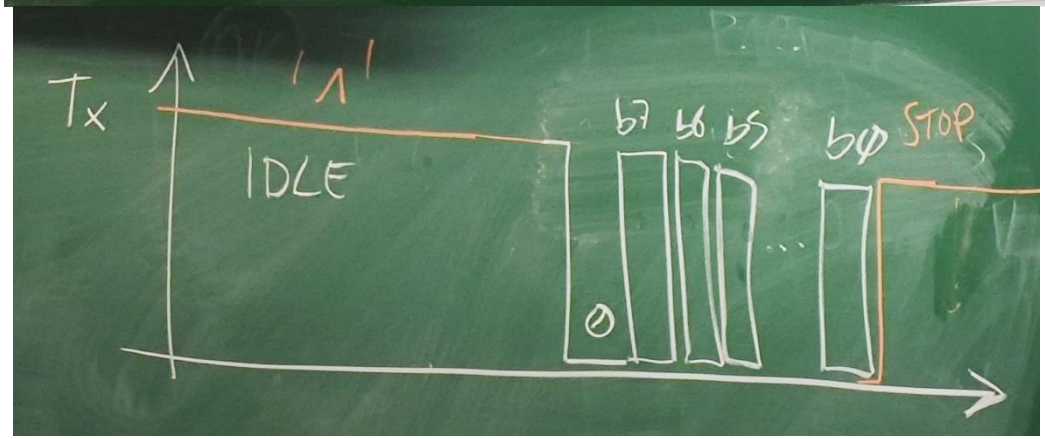

```
void configPIC() {
   ANSELA=0x00;
   ANSELB=0x00;
   ANSELC=0x00;
   ANSELD=0x00;
   ANSELE=0x02;

   TRISD=0x00;
   TRISB=0x00;
   TRISA=0x00;
   TRISC=0x00;

   PORTD=0x00;
   PORTB=0x00;
   PORTE=0x00;

   RCONbits.IPEN = 1;   //disable interrupt priorities
   INTCONbits.GIE = 1;  //enable all interrupts
   INTCONbits.PEIE = 1; //enable peripheral interrupts
   IPR1bits.TMR2IP = 1; //priority level of TMR2
   PIR1bits.TMR2IF = 0; //set 0 as the initial flag of TMR2
   PIE1bits.TMR2IE = 1; //enable TMR2 interrupt
   TRISEbits.RE0 = 0;
   TRISEbits.RE1 = 1;

   ADCON0=0x19;
   ADCON1=0x00;
   ADCON2=0xA1;

}
```

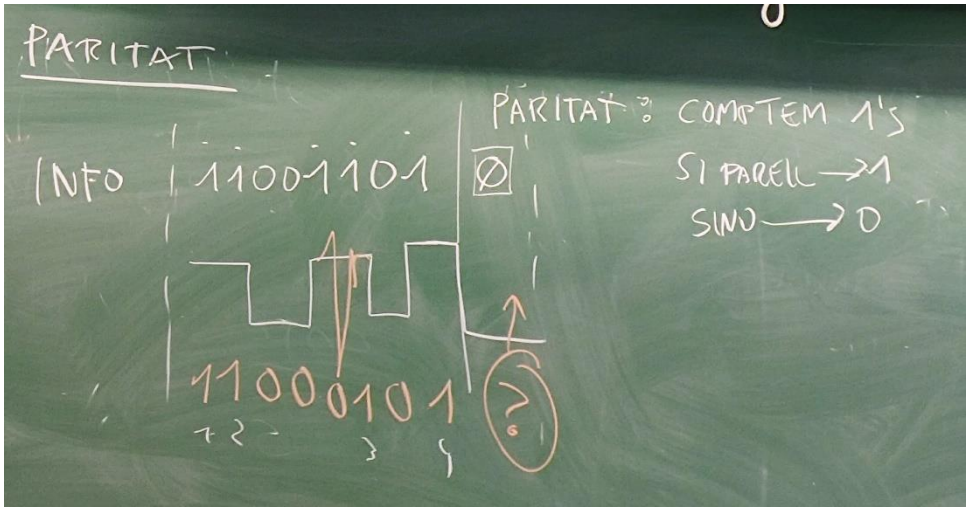
```
while (1)
{
   double v_ref = 2.54;      //Voltatge quan temperatura = 25ºC
   ADCON0bits.GO = 1;
   while (ADCON0bits.GO == 1);       //Esperem a que acabi la conversió A->D
   double v = 5.0*((double)ADRES/1023.0);
   double temp1_formula = temp_formula(v);
   double temp1_aprox = temp_aprox(v);
   if (v < v_ref) {
      percentatge = 25;
      dc = 250;
   }
   else {
      percentatge = 100;
      dc = 1000;
   }
   escriu_DC(dc);
```

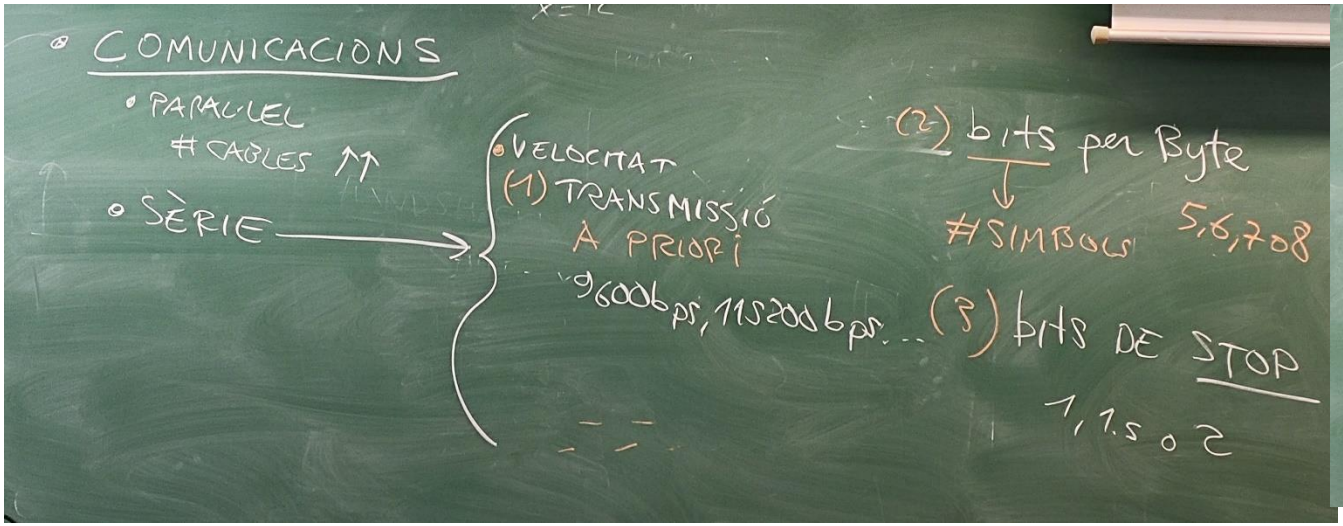


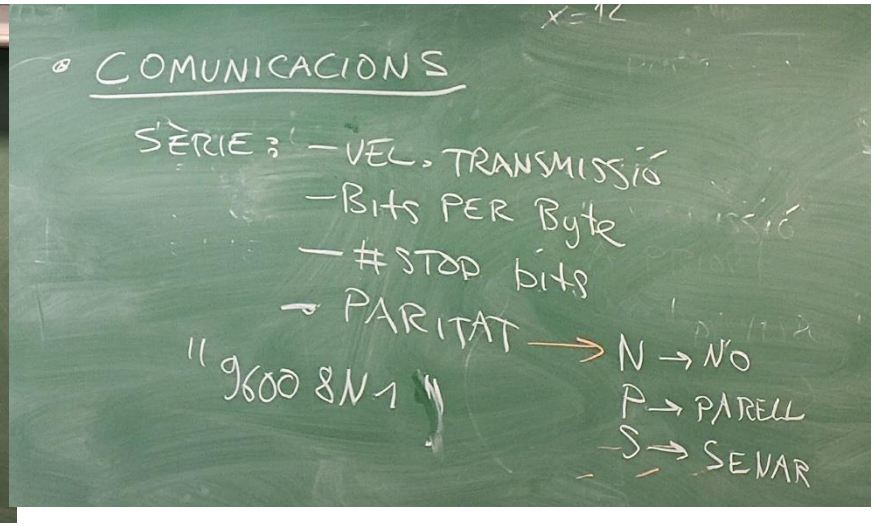

PARAL·LEL
20-22 CABLES
PORTA = x
C0 = 1
while(!C1);
while(!C1);
y = PORTA;
C0 = 1;
C0
C1
A
B

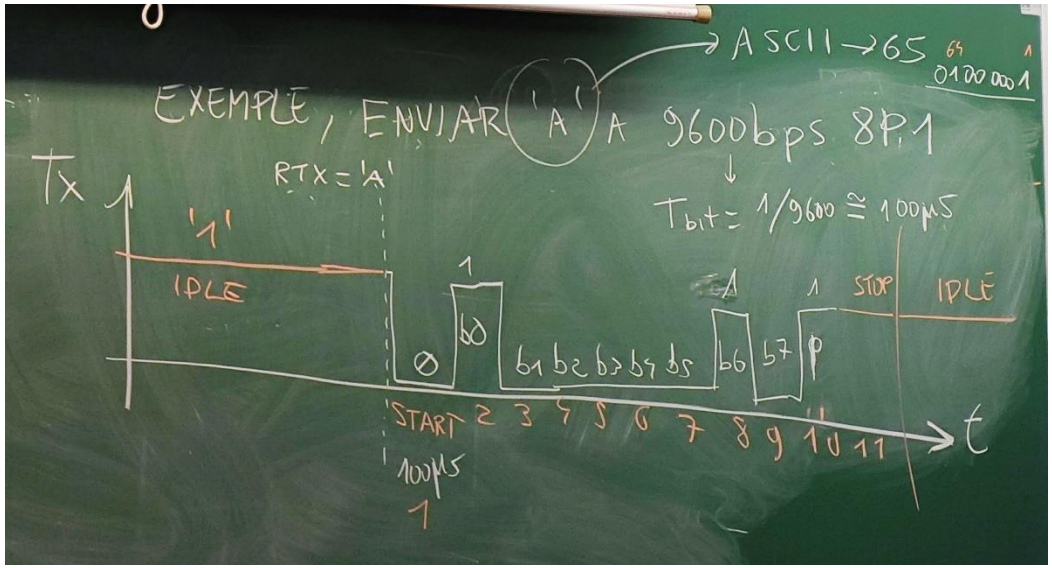

SÈRIE
SV
R
X = 12
Tx
Rx
SHIFT REG
SHIFT REG


Tx
'1'
IDLE
b7 b6 b5 b0 STOP


PARITAT
PARITAT: COMPTEM 1's
SI PARELL → 1
SINO → 0
INFO 11001101
11001101
?


COMUNICACIONS
PARAL·LEL
CABLES ↑↑
SÈRIE
VELOCITAT
(1) TRANSMISSIÓ
A PRIORI
9600bps, 115200bps...
(2) bits per Byte
#SIMBOLS
5,6,7,8
(3) bits DE STOP
1, 1.5 o 2

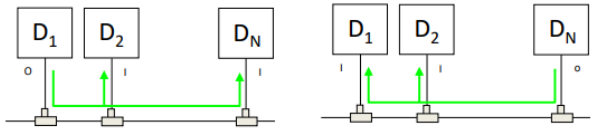

COMUNICACIONS
SÈRIE: - VEL. TRANSMISSIÓ
- Bits PER Byte
- # STOP bits
- PARITAT → N → NO
P → PARELL
S → SENAR
"9600 8N1"



Communication interfaces types

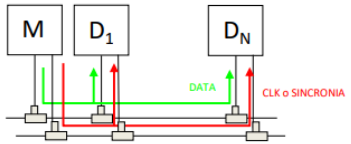
Serial

- Asynchronous
  - Half-Duplex

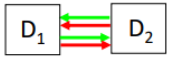


  - Full-Duplex (generally point to point (PTP) only)

- Synchronous
  - Half-Duplex

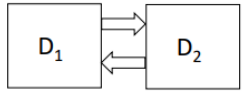


  - Full-Duplex (generally point to point (PTP) only)

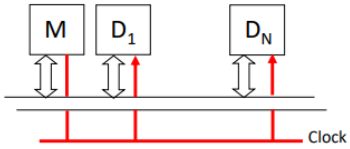


Parallel

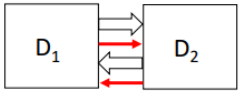
- Asynchronous (not very common) (generally point to point PTP only)



- Synchronous
  - Half-Duplex



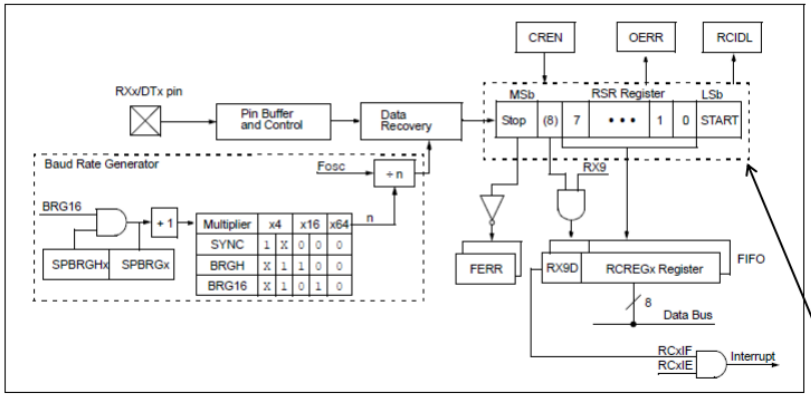
  - Full-Duplex



(generally point to point PTP only)

18F45K22 USART Rx Block

FIGURE 16-2: EUSART RECEIVE BLOCK DIAGRAM



Shift Register (Serial Input/Parallel Output)

Reception starts upon detection of a START bit
- Transmission integrity is checked testing STOP bit level
- Received data in Shift register is stored in FIFO (One single address, RCREG)
- If three characters are received in a row, FIFO overflows Overrun error
- RCIF is set HIGH until FIFO is empty

The PIC18 USART Serial Communication Interface

USART-Related Pins

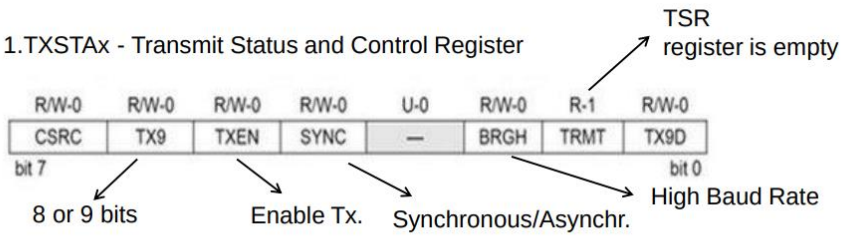
- RC6/TX1/CK1 and RC7/RX1/DT1 (USART1)
- RD6/TX2/CK2 and RD7/RX2/DT2 (USART2)

USART-Related Registers

- Transmit status register (TXSTA) - Transmit register (TXREG)
- Receive status register (RCSTA) - Receive register (RCREG)
- Baud rate Control register (BAUDCON)
- Baud rate generator register (SPBRG)

The PIC18 USART Transmit and receive control registers

1.TXSTAx - Transmit Status and Control Register



2. RCSTAx - Receive Status and Control Register

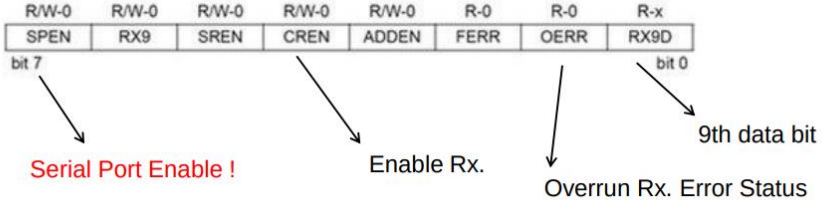


TABLE 20-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on page
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	55
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	55
BAUDCON	ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN	55
SPBRGH	EUSART Baud Rate Generator Register High Byte								55
SPBRG	EUSART Baud Rate Generator Register Low Byte								55

Legend: — = unimplemented, read as '0'. Shaded cells are not used by the BRG.

SPBRG register

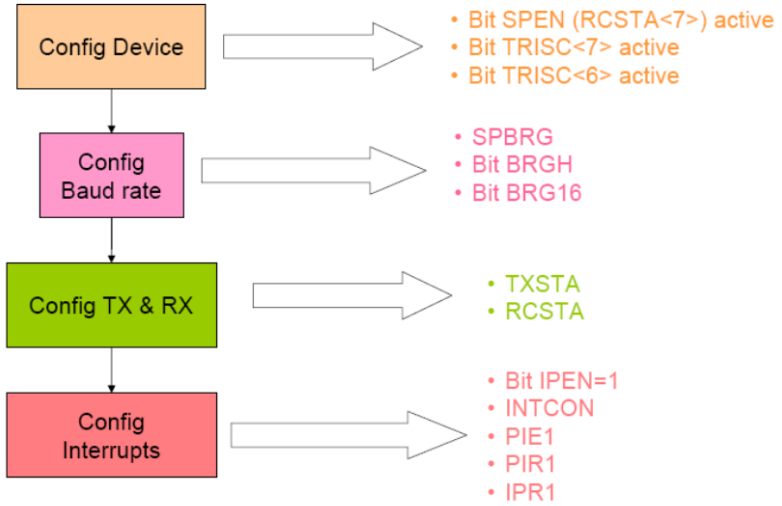
The rate selection is made by the BRGH bit in TXSTA register:
1 = High speed
0 = Low speed

TABLE 20-1: 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 SPBRGH:SPBRG register pair

Serial Comms, Recommended Initialization

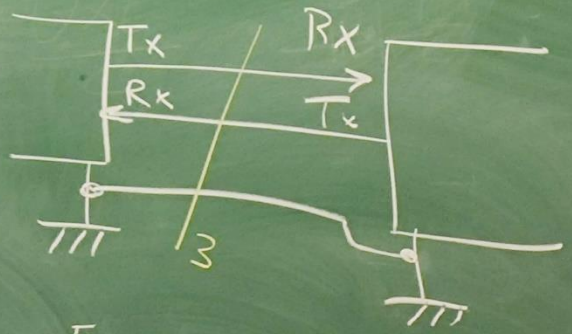


Ex. Write a subroutine to configure the USART1 transmitter to transmit data in asynchronous mode using 8-bit data format, disable interrupt, set baud rate to 9600. Assume the frequency of the crystal oscillator is 16 MHz.

```
void usart1_Init(void)
{
    TXSTA1          = 0x24; /* USART Configuration Register */
    SPBRG1          = 103;  /* Set de Baud rate */
    TRISCbits.RC7   = 1;    /* configure RX1 pin for input */
    TRISCbits.RC6   = 1;    /* configure TX1 pin for output */
    PIE1bits.TXIE   = 0;    /* disable transmit interrupt */
    RCSTA1bits.SPEN = 1;    /* enable USART port */
}
```

COMUNICACIONS SÈRIE

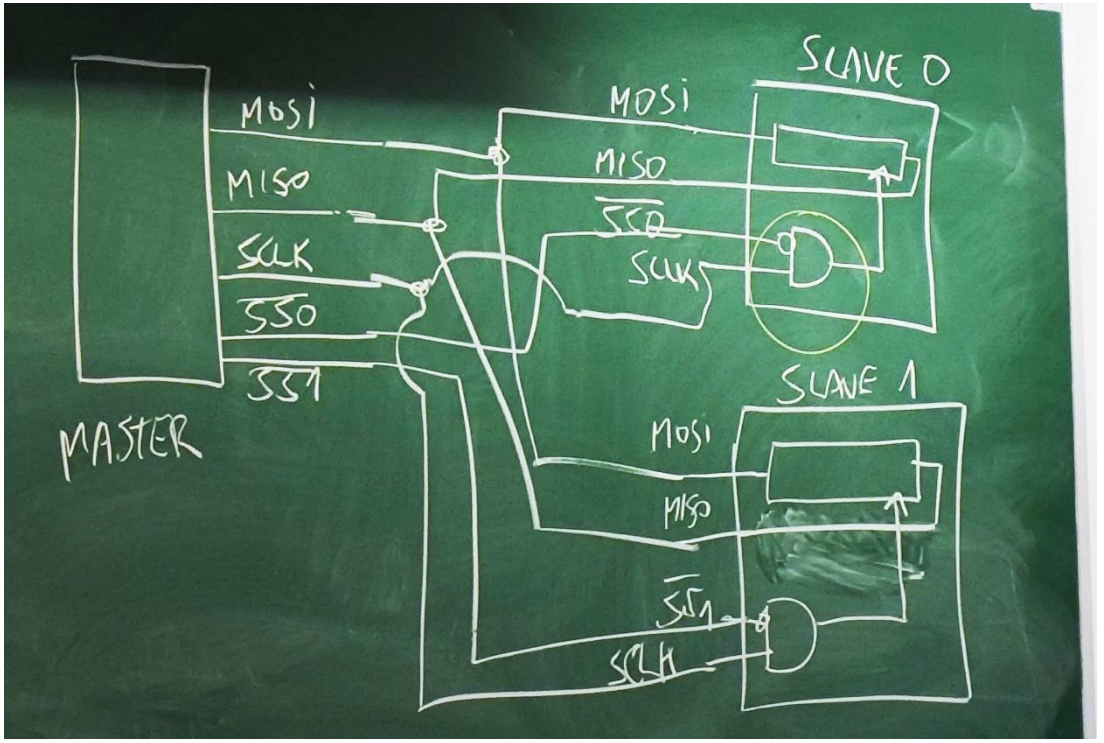
- LÍNIA SÈRIE

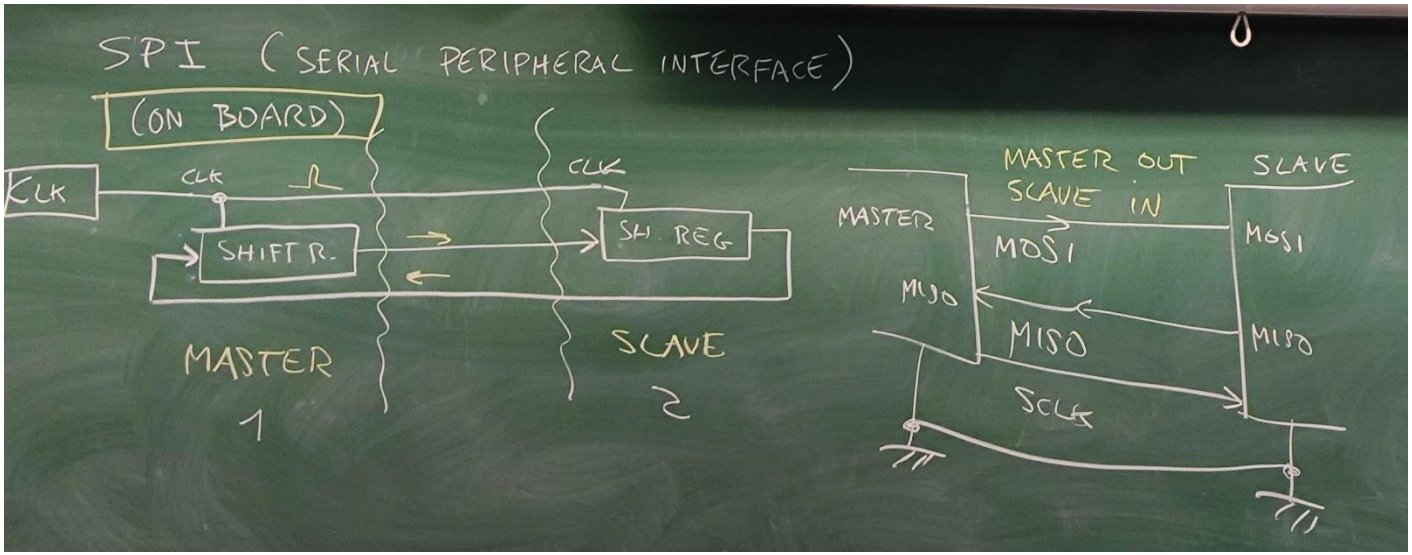


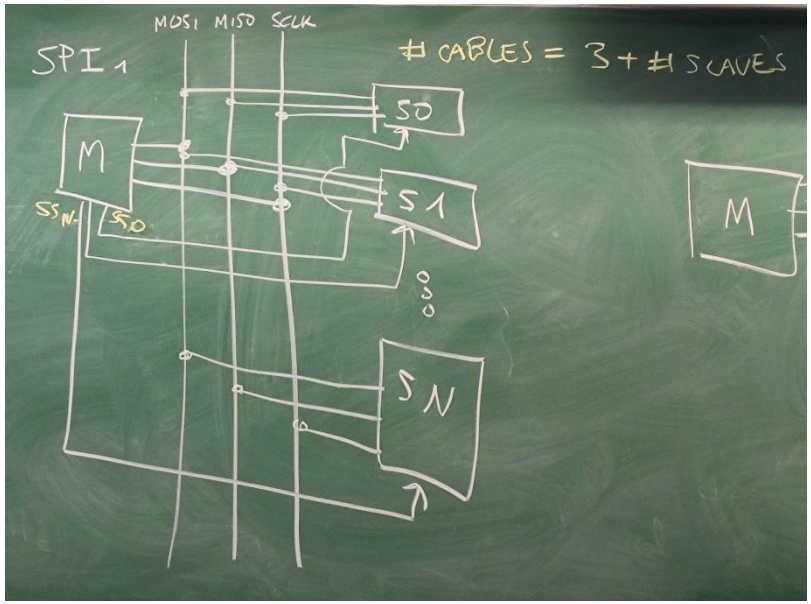
- FULL-DUPPLEX

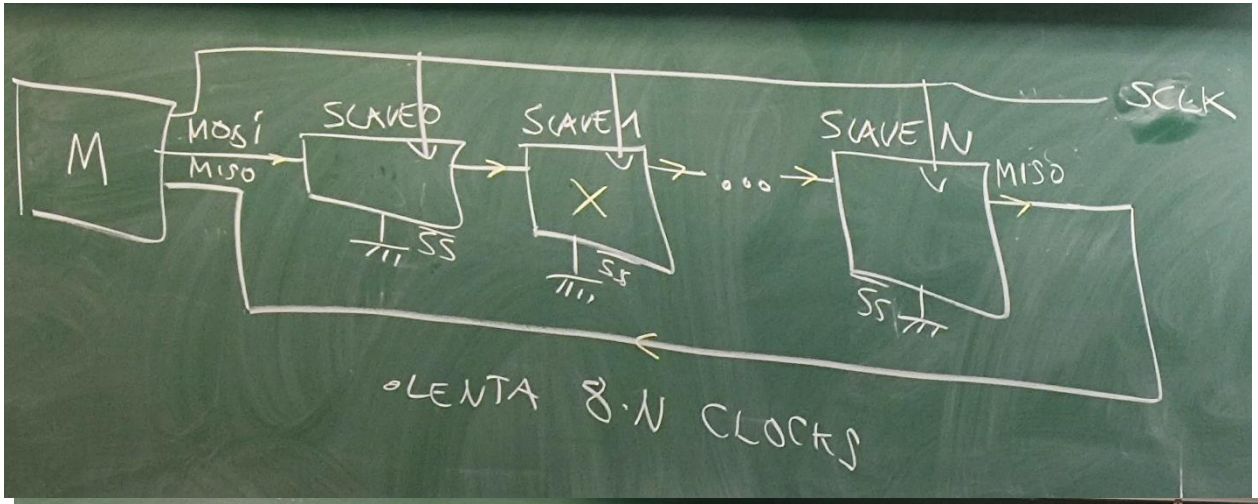
9600...115200 bps

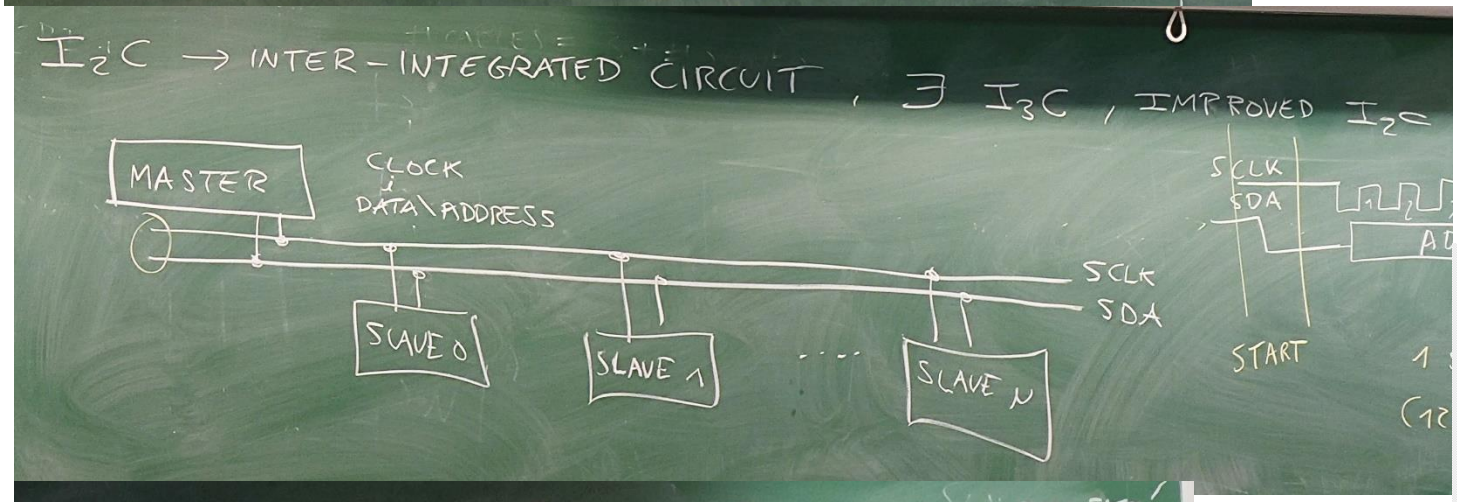
8N1 / 8P1...

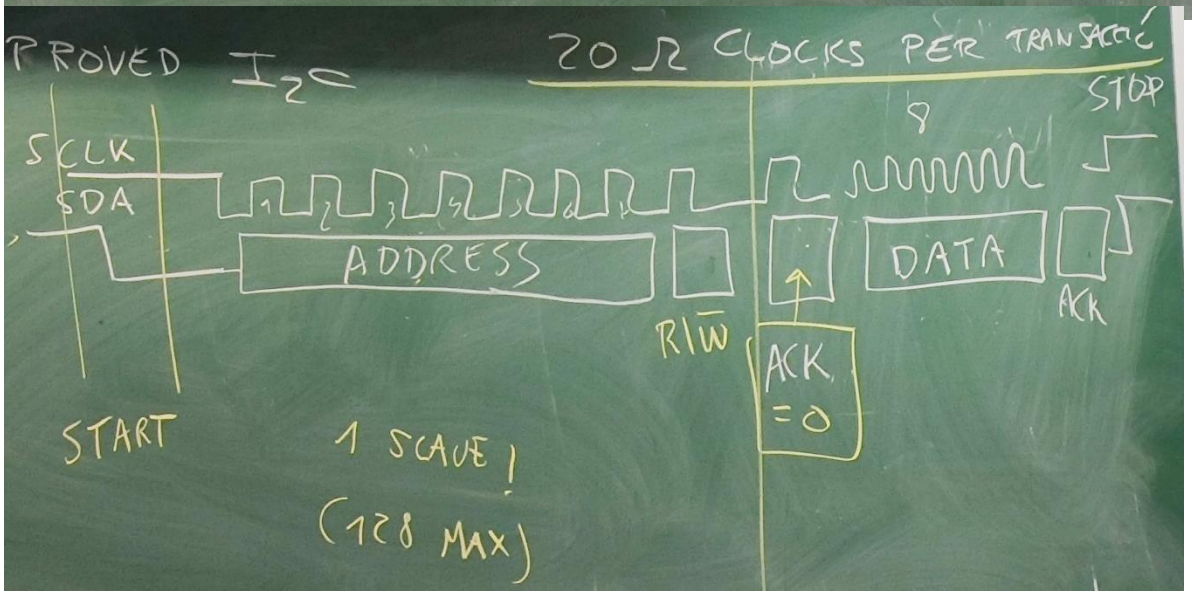






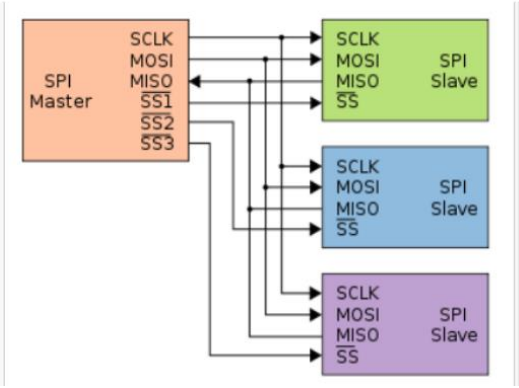






Synchronous Serial Peripheral Interface: SPI

The **Serial Peripheral Interface Bus** or **SPI** bus is a synchronous serial data link standard that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.




```
void pushTxt(char str[]) {
   TX1IE = 0;
   if (last_element == iteration) {
      int fi = strlen(str);
      for (int i = 0; i < fi; ++i) buff[i] = str[i];
      iteration = 0;
      last_element = strlen(str);
   }
   else {
      int fi = strlen(str) + last_element;
      for (int i = last_element; i < fi; ++i)  buff[i] = str[i];
      last_element = fi;
   }
   TX1IE = 1;
}

void pushChar(char str) {
   TX1IE = 0;
   if (last_element == iteration) {
      int fi = strlen(str);
      for (int i = 0; i < fi; ++i) buff[i] = str;
      iteration = 0;
      last_element = strlen(str);
   }
   else {
      int fi = strlen(str) + last_element;
      for (int i = last_element; i < fi; ++i)  buff[i] = str;
      last_element = fi;
   }
   TX1IE = 1;
}
```

```
void pushNum(int num) {
   TX1IE = 0;
   char str[16];

   sprintf(str, "%d", num);
   pushTxt(str);
}


char TX1front(void) {
   if (iteration == last_element) return '\0';
   return buff[iteration++];
}
```

```
void tic() {
   ++cont;
   if (cont == temps_molinet) zero_m = 1;
   if (cont == (temps_molinet + temps_aigua)) zero_a = 1;
}

// RSI High Priority for handling Timer0
void interrupt RSI() {
   if (TX1IE && TX1IF) {
      char c = TX1front();
      if (c != '\0') TXREG1 = c;         // Still data to write
      else TX1IE = 0;                    // No more data to transmit
   }

   if (RC1IE && RC1IF) {
      if(RCSTAbits.OERR == 1) RCSTAbits.OERR = 0;
      else if (RCSTAbits.FERR == 1) RCSTAbits.FERR = 0;
      else {
            char c = RCREG1;
            if (c == 'w') {
               PORTCbits.RC0 = 1;
               if (temps_molinet < 15) {temps_molinet += 5; act_m = 1;}
            }

            else if (c == 'x') {
            PORTCbits.RC1 = 1;
            if (temps_molinet > 5) {temps_molinet -= 5; act_m = 1;}
            }

            else if (c == 'a') {
            PORTCbits.RC3 = 1;
            if (temps_aigua > 10) {temps_aigua -= 5; act_a = 1;}
            }

            else if (c == 'd') {
            PORTCbits.RC2 = 1;
            if (temps_aigua < 20) {temps_aigua += 5; act_a = 1;}
            }

            else if (c == 's') {
            PORTCbits.RC4 = 1;
            }
      }

      RC1IF = 0;
   }

   if (PIR1bits.TMR2IF == 1 && PIE1bits.TMR2IE == 1) {
      TRISEbits.TRISE0 = 0;
      PIR1bits.TMR2IF = 0;
   }
   if (INTCONbits.TMR0IF == 1 && INTCONbits.TMR0IE == 1) {
      T0CONbits.TMR0ON = 0;
      tic();
      INTCONbits.TMR0IF=0;
      TMR0=15356;
      T0CONbits.TMR0ON = 1;
   }
}
```

```
void configPIC() {
   ANSELA=0x00;
   ANSELB=0x00;
   ANSELC=0x00;
   ANSELD=0x00;
   ANSELE=0x06;
   TRISA=0x0C;
   TRISB=0x00;
   TRISC=0xC0;
   TRISD=0x00;
   TRISEbits.TRISE0 = 1;
   TRISEbits.TRISE1 = 1;
   TRISEbits.TRISE2 = 1;
   PORTD=0x00;
   PORTB=0x00;
   PORTE=0x00;
   PORTA=0x00;
   PORTC=0x00;

   //Linia sèrie config
   SPBRG1 = 16;
   //SPBRGH = 16;                        //BaudRate = 117600 (aprox 115200)
   RCSTA1bits.SPEN = 1;                  //Serial Port Enable
   TXSTA1bits.BRGH = 1;                  //High mode
   BAUDCON1bits.BRG16 = 1;               //High baud rate
   TXSTA1bits.SYNC = 0;                  //Asincron
   TXSTA1bits.TX9 = 0;                   //8 bits
   TXSTA1bits.TXEN = 1;                  //Enable transmit
   RCSTA1bits.RX9 = 0;                   // 8 bits datsa receiving
   RCSTA1bits.CREN = 1;                  // Enable receiving
```

```
    RC1IE = 1;                          // Enable receiving interrupts (port 1 data)
    RCIE = 1;                           // Enable receiving(data)

    RCONbits.IPEN = 1;   //disable interrupt priorities
    INTCONbits.GIEH = 1;    //Enabe high priority interrupts (working with core int, no need to setup peripheral interrupts)
    INTCONbits.TMR0IF = 0;  //clear IF flag on startup
    INTCONbits.TMR0IE = 1;  //Enable interrupt on timer 0 overflow
    INTCON2bits.TMR0IP = 1;
    T0CONbits.TMR0ON = 0;   //Enable Timer 0
    T0CONbits.T08BIT = 0;   //Timer 0 as 16 bit counter
    T0CONbits.T0CS = 0;     //Use internal ins. cycle clock (Fosc/4)
    T0CONbits.PSA = 0;      //Prescaler assignet for timer 0
    T0CONbits.T0PS = 0b100; //Prescaler value
    TMR0=15356;  //Farem un timer de 0.8 s en comptes de 1s per compensar el temps perdut als delays necessaris per no detectar 1 flanc com si fossin 2 o més
    IPR1bits.TMR2IP = 1; //priority level of TMR2
    PIR1bits.TMR2IF = 0; //set 0 as the initial flag of TMR2
    PIE1bits.TMR2IE = 1; //enable TMR2 interrupt
    TRISEbits.RE0 = 0;
    TRISEbits.RE1 = 1;
    TXIE = 1;                           // Enable interrupts
    TX1IE = 1;

    ADCON0=0x19;
    ADCON1=0x00;
    ADCON2=0xA1;
    INTCONbits.GIE = 1;  //enable all interrupts
    INTCONbits.PEIE = 1; //enable peripheral interrupts
}

while (1) {
    if(RCSTAbits.OERR == 1) RCSTAbits.OERR = 0;
    else if (RCSTAbits.FERR == 1) RCSTAbits.FERR = 0;
    //Nivell d'aigua --> Si l'aigua és baixa no permetrem que es faci res i mostrarem un missatge d'error
    ADCON0bits.CHS = 7;
    ADCON0bits.GO = 1;
    while (ADCON0bits.GO == 1);       //Esperem a que acabi la conversió A->D
    double v_pot = 5.0*((double)ADRES/1023.0);
    ADCON0bits.CHS = 6;
    if (v_pot < 1.0) {
        if (bajo == 0) clearGLCD(0,7,0,127);
        writeTxt(4, 6, s5);
        bajo = 1;
        PORTAbits.RA0 = 0;
        PORTAbits.RA1 = 0;
        activat = 0;
        cont = 0;
        zero_m = 0;
        zero_a = 0;
        INTCONbits.TMR0IF = 0;
        T0CONbits.TMR0ON = 0;
    }
    if (bajo == 1 && v_pot >= 1.0) {
        bajo = 0;
        clearGLCD(0,7,0,127);
        inicialitzar_barra_t();
        inicialitzar_barra_tm();
        inicialitzar_barra_ta();
        dibuixar_RC0();
        dibuixar_RC1();
        dibuixar_RC2();
        dibuixar_RC3();
        dibuixar_RC4();
        dibuixar_RA2();
        dibuixar_RA3();
    }
    else if (bajo == 0) {
        if (PORTCbits.RC4 == 1 && apretat4 == 0) {
            __delay_ms(50);
            apretat4 = 1;
            activat = 1;
            INTCONbits.TMR0IF = 0;
            INTCONbits.TMR0IE = 1;
            TMR0 = 15356;
            T0CONbits.TMR0ON = 1;
        }
        if (activat == 1) {    //Si activat == 1 posarem en marxa la cafetera, sinó permetrem que l'usuari modifiqui els paràmetres
            if (zero_m == 0) PORTAbits.RA0 = 1;
            else PORTAbits.RA0 = 0;
            if (zero_a == 0 && zero_m == 1) PORTAbits.RA1 = 1;
            else PORTAbits.RA1 = 0;
            if (zero_m == 1 && zero_a == 1) {
                activat = 0;
                cont = 0;
                zero_m = 0;
                zero_a = 0;
                INTCONbits.TMR0IF = 0;
                T0CONbits.TMR0ON = 0;
                clearGLCD(0,7,0,127);
```

```
      const char * s_final = "CAFE LLEST!\n";
      writeTxt(4, 7, s_final);
      __delay_ms(5000);
      clearGLCD(0,7,0,127);
      inicialitzar_barra_t();
      inicialitzar_barra_tm();
      inicialitzar_barra_ta();
      dibuixar_RC0();
      dibuixar_RC1();
      dibuixar_RC2();
      dibuixar_RC3();
      dibuixar_RC4();
      dibuixar_RA2();
      dibuixar_RA3();
   }
}
else {
   if (PORTCbits.RC0 == 1 && apretat0 == 0) {
      __delay_ms(50);
      apretat0 = 1;
      if (temps_molinet < 15) {temps_molinet += 5; act_m = 1;}
   }
   else if (PORTCbits.RC1 == 1 && apretat1 == 0) {
      __delay_ms(50);
      apretat1 = 1;
      if (temps_molinet > 5) {temps_molinet -= 5; act_m = 1;}
   }
   else if (PORTCbits.RC3 == 1 && apretat3 == 0) {
      __delay_ms(50);
      apretat3 = 1;
      if (temps_aigua > 10) {temps_aigua -= 5; act_a = 1;}
   }
   else if (PORTCbits.RC2 == 1 && apretat2 == 0) {
      __delay_ms(50);
      apretat2 = 1;
      if (temps_aigua < 20) {temps_aigua += 5; act_a = 1;}
   }
   if (PORTAbits.RA2 == 1 && apretat_A[0] == 0) {
      __delay_ms(50);
      apretat_A[0] = 1;
      if (temperatura < 70) temperatura += 10;
   }
   if (PORTAbits.RA3 == 1 && apretat_A[1] == 0) {
      __delay_ms(50);
      apretat_A[1] = 1;
      if (temperatura > 50) temperatura -= 10;
   }
   double v_ref = 2.54;        //Voltatge quan temperatura = 25ºC
   ADCON0bits.GO = 1;
   while (ADCON0bits.GO == 1);         //Esperem a que acabi la conversió A->D
   double v = 5.0*((double)ADRES/1023.0);
   double temp1_formula = temp_formula(v);
   double temp1_aprox = temp_aprox(v);
   if ((temperatura - temp1_formula) <= -5.0) {
      percentatge = 100;
      dc = 1000;
      escriu_DC(dc);
   }
   else if ((temperatura - temp1_formula) == 0.0){
      percentatge = 50;
      dc = 500;
      escriu_DC(dc);
   }
   else if ((temperatura - temp1_formula) >= 5.0){
      percentatge = 0;
      dc = 11;
      escriu_DC(dc);
   }

   else {
      double qq = (temperatura - temp1_formula);
      if (qq > 0.0) {
         int r = (int)(qq*100);
         if(r >= 500) {percentatge = 0; dc = 11;}
         else {
            percentatge = (500-r)/10;
            dc = 500-r;
         }
      }

      else if (qq < 0.0) {
         int rr = (int)(-qq*100);
            percentatge = 50 + rr/10;
            dc = 500 + rr;
      }
     escriu_DC(dc);
   }
   if ((t_for_ant - temp1_formula) >= 1.0 || (t_for_ant - temp1_formula) <= -1.0) {/*actualitza_temp_GCLD(temp1_formula);*/  t_for_ant = temp1_formula;}
   dibuixar_RC0();
   dibuixar_RC1();
   dibuixar_RC2();
   dibuixar_RC3();
   dibuixar_RC4();
                dibuixar_RA2();
                dibuixar_RA3();
                dibuixar_barra_t();
                if (act_m == 1) {dibuixar_barra_tm(); act_m = 0;}
                if (act_a == 1) {dibuixar_barra_ta(); act_a = 0;}
            }
         }
      }
   }
```