0. Intro

Type	Size in Bytes	Range
`(unsigned) char`	1	0 .. 255
`signed char`	1	- 128 .. 127
`(signed) short (int)`	1	- 128 .. 127
`unsigned short (int)`	1	0 .. 255
`(signed) int`	2	-32768 .. 32767
`unsigned (int)`	2	0 .. 65535
`(signed) long (int)`	4	-2147483648 .. 2147483647
`unsigned long (int)`	4	0 .. 4294967295

1. GPIO

a. CHECK that all the pin are pulled-down
b. Chechk that the J24 is not being shorted

ANSELx:

Accendi il buffer d'ingresso

1 ---> Digital input buffer DISABLED
0 ---> Digital input buffer ENABLED

Default: 1

TRISx:

Accendi il buffer d'uscita

1 ---> PORTx configured as an INPUT
0 ---> PORTx configured as an OUTPUT

Default: 1

2. INTERRUPT

ALWAYS read the datasheet before using interrupts!!!

INTCON (*Pagina 109*):

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

Metti la flag *IF* a 0 quando finisce l'interrupt

Interrupt on change

```
// Abilito all'IOC i pin 7 e 6

ANSELB = 0b00111111;
IOCB = 0b11000000;
INTCON = 0b10001000;

void interrupt (){
    if(INTCON.RBIF){
        if(PORTB.RB7) speed = speed + 100;
        if(PORTB.RB6) speed = speed - 100;
    }
    INTCON.RBIF = 0; //RICORDATI
}
```

3.1 TIMER0

Timer0 è configurato solo con il registro T0CON.
Il fatto ogni tanto faccia partire un interrupt è scollegato dal timer in se

Formula del timer:

$$T_{TMR0IF} = \left(\frac{Fosc}{4}\right)^{-1} \cdot PR \cdot (256 - TMR0L_{\mathrm{INIT}})$$

T0CON (*Pagina 154*):

REGISTER 11-1: T0CON: TIMER0 CONTROL REGISTER

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	T0PS<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 **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

Timer enhanced precision:

```
INTCON = 0b10100000;
T0CON = 0b11000111;

void interrupt (){
    if(INTCON.RBIF){
        if(PORTB.RB7) speed = speed + 100;
        if(PORTB.RB6) speed = speed - 100;
    }
    INTCON.RBIF = 0; //RICORDATI

    //Gestione precisa del tempo
    if(INTCON.TMR0IF) {
    INTCON.TMR0IF = 0;

    time_ms += 32;
    time_us += 768;

  }

    if (time_us >= 1000) {
    time_us -= 1000;
```

```
    time_ms++;
  }
}
```

3.2 LCD

Inizializzazione:

VARIABLES MUST BE INITIALIZED **BEFORE** INITIALIZING THE LCD

```
// Fuori dal main

// Lcd module connections
sbit LCD_RS at LATB4_bit;
sbit LCD_EN at LATB5_bit;
sbit LCD_D4 at LATB0_bit;
sbit LCD_D5 at LATB1_bit;
sbit LCD_D6 at LATB2_bit;
sbit LCD_D7 at LATB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End Lcd module connections

void main(){

    // ---------Initialization----------
    Lcd_Init();                     // Init the display library (This will set
also the PORTB Configuration)

    Lcd_Cmd(_LCD_CLEAR);            // Clear display
    Lcd_Cmd(_LCD_CURSOR_OFF);       // Cursor OFF


}
```

The string you use `IntToString` with MUST BE 7 characters long

Timer setup

```
char txt[17];
char txtNum[7];

main(){
// ----------------Print stopwatch section----------------------
```

```
    IntToStr(stopwatch_hours, txtNum);  // Convert the number in a string
    for(i = 0; i < 7 - 4; i++)
        txtNum[i] = txtNum[i + 4];      // Shift string characters for reducing
the size
    if(txtNum[0] == ' ')                // If there is a blank character
        txtNum[0] = '0';                // Replace with 0
    strcpy(txt, txtNum);                // Copy in txt (only the first, the
others will be concat)
    strcat(txt, ":");                   // Concat the :

}
```

4. EXERCISE

Button management :

```
unsigned short int button_pressed = 0;

void main (){
    if( (PORTA & ~button_pressed) & 0b00000001 ){

        //CODE

        button_pressed = button_pressed | 0b00000001;
    }

    button_pressed = button_pressed & PORTA;

}
```

LCD timer setup :

```
unsigned short int button_pressed = 0;

void main (){
    Lcd_Out(1, 5, "00:00:00");

    //create clock string
    IntToStrWithZeros(clock_h, clock_temp);
    for(i = 0; i < 3; i++){
      clock_temp[i] = clock_temp[i+4];
    }
    strcpy(clock, clock_temp);
    strcat(clock, ':');

}
```

5. ATOMICITA'

Disattivare gli interrupt prima di accedere una variabile globale

Esempio:

```
if( PORTA & 0b00100000 ){

            INTCON.GIE = 0;
            print_distance_milli = print_distance_delay;
            INTCON.GIE = 1;
}
```

6. ADC

L'ADC effettua la conversione e restituisce un risultato a 10 bit e lo mette in `ADRESH` e `ADRESL`

Campiona a 1024 livelli tra 0 e 5 volt.

La fine della conversione può triggerare un interrupt.

Ho 3 registri per gestirlo:
ADCON0
ADCON1
ADCON2

ADCON0 (Pagina 295):

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/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

CHS is used to select the right analog input

bit 1 **GO/$\overline{\text{DONE}}$:** 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

ADCON1 non lo uso. Serve per settare i livelli di tensione

ADCON2:

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

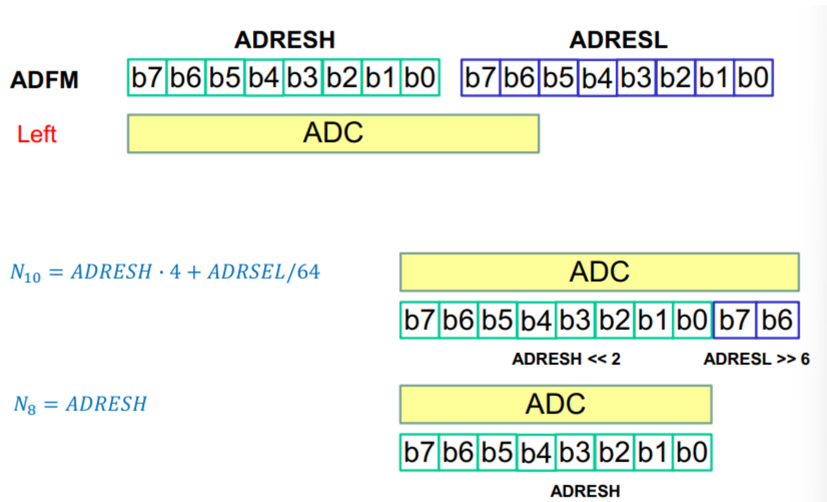
bit 6 **Unimplemented:** Read as '`0`'

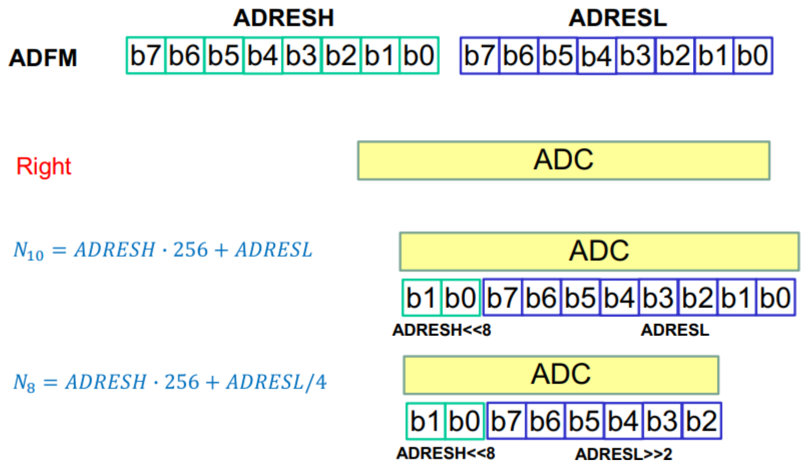
bit 5-3 **ACQT<2:0>:** A/D Acquisition time select bits. Acquisition time is the duration that the A/D charge holding capacitor remains connected to A/D channel from the instant the GO/$\overline{DONE}$ bit is set until conversions begins.
`000` = 0(1)
`001` = 2 TAD
`010` = 4 TAD
`011` = 6 TAD
`100`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

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)

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/$\overline{DONE}$ bit is set to allow the `SLEEP` instruction to be executed.

JUSTIFICATION:





Use bitwise oparations instead of multiplication and division

POLLING:

```
//Set the ADC up
// ----- Output ------
TRISC = 0;
// -------------------

// ---- AN0 = RA0 ----
// Default is aalog input; i.e., digital input buffer off
// ANSELA.RA0 = 1;
// -------------------

// -- Set CHS (AN0) --
ADCON0.CHS0 = 0;
ADCON0.CHS1 = 0;
ADCON0.CHS2 = 0;
ADCON0.CHS3 = 0;
ADCON0.CHS4 = 0;
// -------------------

// ----- Set TAD -----
//Fosc = 8MHz TAD
// Fosc/8
// ADCS = 001
ADCON2.ADCS0 = 1;
ADCON2.ADCS1 = 0;
ADCON2.ADCS2 = 0;
// -------------------

// --- Set ACQT -----
// TACQ = 7.45 us
// TACQTmin = 8 TAD
ADCON2.ACQT0 = 0;
ADCON2.ACQT1 = 0;
ADCON2.ACQT2 = 1;
// -------------------

// --- Just. Left ---
ADCON2.ADFM = 0;
```

```
// -------------------

// ----- ADC ON -------
ADCON0.ADON = 1;
// -------------------

// Start Conv.
ADCON0.GO_NOT_DONE = 1;


//poll the results
    while (1)
    {

        // Polling
        if (ADCON0.GO_NOT_DONE == 0)
        {
            // Print 8 MSBs on PORTC
            LATC = ADRESH;

            ADCON0.GO_NOT_DONE = 1;
        }


    }
```

INTERRUPT:

CHECKLIST:

1. Fill `ADCON0` and `ADCON2` registers
2. Enable `PIE1.ADIE`
3. Lower `PIR1.ADIF` flag
4. Turn on the ADC (`ADCON0.ADON = 1`)
5. Start the conversion (`ADCON0.GO_NOT_DONE = 1`)
6. Activate and manage interrupts (peripheral interrupts) on `INTCON`
7. Retrieve and elaborate the results from `PIR1.ADIF` and `ADRESH / ADRESL`

```
//............Setup ADC

PIE1.ADIE = 1; //A/D converter interrupt enable bit '1' is on, '0' is off
PIR1.ADIF = 0; //A/D converter interrupt FLAG bit (default is already 0)

INTCON.PEIE = 1;
INTCON.GIE = 1;

ADCON0.GO_NOT_DONE = 1;


while(1){

    /*  CODE  */
```

```
}

void interrupt (){
    if(PIR1.ADIF){
        adc_10bit = (ADRESH << 2) + (ADRESL >> 6);
        PIR1.ADIF = 0;
        ADCON0.GO_NOT_DONE = 1;
    }

}
```

Valore potenziometro conversione in mv:

```
adc_10bit * 5;
adc_8bit * 20;
```

Campionare insieme da due porte diverse:

```
int adc_flag = 0;  // 0 NA; 1 adc_10bit NEW, 2 adc_8bit NEW

void main() {
    // ......Inizializzazioni

    while (1)
    {
        // .... altre parti di codice
        // Print AN0
        if (adc_flag == 1)
        {
            adc_flag = 0;
            //Code

            //Set AN1
            ADCON0.CHS0 = 1;
            // Start Acq AN1
            ADCON0.GO_NOT_DONE = 1;

        }


        //...... altre righe di codice. Potrei non voler leggere dal secondo
analog

        // Print AN1
        if (adc_flag == 2)
        {
            adc_flag = 0;

            //Code
            //Set AN0
```

```
            ADCON0.CHS0 = 0;
            // Start Acq AN0
            ADCON0.GO_NOT_DONE = 1;
        }
        //.....
    }
}

void interrupt(){

    if (PIR1.ADIF)
    {
        //AN0
        if(ADCON0.CHS0 == 0 ){
            //Code
            adc_flag = 1;
        }
        //AN1
        if( ADCON0.CHS0 == 1 ) {
            //Code
            adc_flag = 2;
        }
        PIR1.ADIF = 0;
    }
}
```

7. PWM

The board has 5 CCP modules. (Capture, Compare, PWM)

Chapter 12: Timer 1/3/5 (odd timers are for Capture or Compare)

Chapter 13: Timer 2/4/6 (even timers are for PWM)

Chapter 14: CCP modules

Every CCP module has an output pin, see Table 3

Se cambio `CCPRxL` a metà ciclo non devo preoccuparmi della stabilità, perché il passaggio a `CCPRxH` avverrà solo alla fine del ciclo.

`TMRx` has 8 bits and stores ???

`PRx` has 8 bits and stores the PWM period

Timer frequency: $f_{TMRx} = \left(\frac{f_{osc}}{4}\right) \cdot \frac{1}{TMRxPR}$

Pwm period: $T = (PRx + 1) \cdot TMRxPS \cdot \frac{4}{f_{osc}}$

`CCPRxL` is a 8 bits register that stores the MSBs of time_up

CCPTMRS0 (Pagina 201):

Seleziona CCP da usare e timer relativo in base alla consegna dell'esercizio

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

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 7-6 **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

CCPTMRS1 (Pagina 201):

Seleziona CCP da usare e timer relativo in base alla consegna dell'esercizio

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
11 = Reserved

CCPxCON (Pagina 198):

Per PWM riempio: gli ultimi 4 bit con `11xx`

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

<u>Capture mode:</u>
Unused

<u>Compare mode:</u>
Unused

<u>PWM mode:</u>
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.

TxCON (Pagina 171):

Bit 2 is to turn the timer on, bits 1-0 set the prescaler

13.6 Register Definitions: Timer2/4/6 Control

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

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 **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

PWM usage checklist (pagina 180):

1. Disable the Pin Output
2. Select the timer by setting CCPTMRSx.CxTSEL 1:0
3. Load the PRx of the corresponding timer with the period value
4. Load CCPxCON
5. Load CCPRxL and, in case, the CCPxCON.DCxB1:0 with the duty cycle value
6. Configure the TxCON pin
7. Enable the output pin

PWM setup and usage:

```
void main() {



    // - Disaple CCP Out -
    TRISE.RE2 = 1;
    // Per evitare commutazioni spurie, potrebbero rompere qualcosa
```

```
    // --- TMR2 - CCP5 ---
    CCPTMRS1.C5TSEL1 = 0;
    CCPTMRS1.C5TSEL0 = 0;
    //Trovo info a pagina 201


    // --- Set Period ----
    PR2 = 255;
    // -------------------

    // --- PWM on CCP5 ---
    CCP5CON = 0b00001100;
    //CCP5CON.CCP5M3 = 1;
    //CCP5CON.CCP5M2 = 1;
    // Setto la modalità del PWM. E' difficile capire perché ma usiamo sempre
questa

    // ---- Set Ton ------
    // d = CCPR5L /(PR2 +1)
    CCPR5L = 0;
    // -------------------


    // ---- Set TMR2 ----- //serve sta roba?
    // TMR2 ON, Max Prescaler
    T2CON = 0b00000111;
    // Il primo è innutilizzato, gli ultimi 2 sono per il prescaler a 16



    // - Output Enable --
    TRISE.RE2 = 0;
    // -------------------

    while(1){

        Delay_ms(100);
        CCPR5L++;

    }
}
```

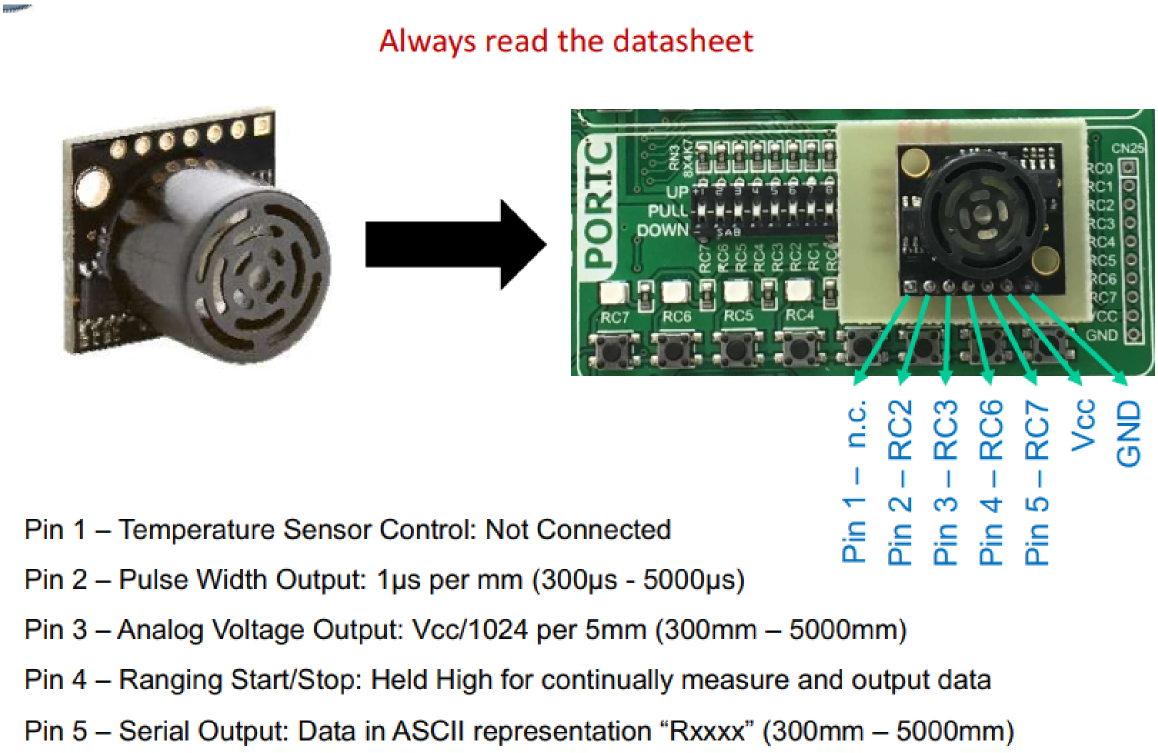
8. Sonar

The sonar has 2 main output modes: pulse width and analog output.

Pulse width has a scale factor of 1us per mm. Range is from 300mm to 5000m.

Il pin 4 (RC6) va tenuto alto, poiché è responsabile dell'accensione del sonar.

Pin configuration:



Pulse width:

Misuro la distanza tra un rising edge e un falling edge con un CCP messo su capture.

Bisogna settare CCPxCON.CCPxM3:0 in modo che sia su falling o su rising edge.

Intercetto la flag del capture in PIRx.CCPxIF.

Il deltaT che trovo è espresso in cicli macchina, devo poi riportarlo in secondi.

Se scelgo il T_clk = 1us, per via delle semplificazioni trovo che `T_b - T_a = dist`, altrimenti ho che `dist = (T_b - T_a) * T_clk * 1mm/1us`

Useful registers:

REGISTER 9-9: PIE1: PERIPHERAL INTERRUPT ENABLE (FLAG) REGISTER 1

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
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 **Unimplemented:** Read as ‘0’.

bit 6 **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5 **RC1IE:** EUSART1 Receive Interrupt Enable bit
1 = Enables the EUSART1 receive interrupt
0 = Disables the EUSART1 receive interrupt

bit 4 **TX1IE:** EUSART1 Transmit Interrupt Enable bit
1 = Enables the EUSART1 transmit interrupt
0 = Disables the EUSART1 transmit interrupt

bit 3 **SSP1IE:** Host Synchronous Serial Port 1 Interrupt Enable bit
1 = Enables the MSSP1 interrupt
0 = Disables the MSSP1 interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE:** TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

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

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
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 **Unimplemented:** Read as ‘0’.

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared by software)
0 = The A/D conversion is not complete or has not been started

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

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 7-6 **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

Checklist:

1. Abilito l'output sul RC6 per poter poi accendere il sonar
2. Abilito l'input digitale su RC2 per ricevere il segnale dal sonar
3. Setto CCPxCON per ricevere i rising edge
4. Setto CCPTMRSx per quale CCP usare e con quale timer
5. Imposto il relativo timer in TxCON
6. In PIE1 abilito gli interrupt dal CCP1IE
7. Imposto INTCON, abilitando gli interrupt da periferica
8. Accendo il bit RC6
9. Nell'interrupt leggo la flag da PIR1, e cambio la modalità da rising a falling edge

Implementazione:

```
unsigned int amplitude =0;
unsigned int Tb, Ta = 0;
unsigned short int sensor_flag, sensor_flag_CCP = 0;

TRISC.RC6 = 0;
ANSELC.RC2 = 0;

CCP1CON = 0b00000101;
CCPTMRS0 = 0;

T1CON = 0b00010011;

PIR1 = 0; //mette la flag dell'interrupt a 0, ma già c'è
PIE1.CCP1IE = 1;
INTCON = 0b11000000;

LATC.RC6 = 1; //accendo il sonar

void main(){
        // First Line ADC
        if(sensor_flag_CCP==1){

        risultato = Tb - Ta; //forse da fare *5
            sensor_flag_CCP = 0;

        }

}

void interrupt(){
    if(PIR1.CCP1IF){
        PIR1.CCP1F = 0;
        if(CCP1CON.CCP1M0){ //Se vero => c'è stato rising edge
            Ta = (CCPR1H << 8) | CCPR1L;
            CCP1CON.CCP1M0 = 0; //setto per captare i fallin (?)
        }
        else{ // Altrimenti => Falling
            Tb = (CCPR1H << 8) | CCPR1L;
            CCP1CON.CCP1M0 = 1;
            sensor_flag_CCP = 1;
        }
    }
    }
}
```

Analog input:

Per via delle semplificazioni ho che `distanza = ADC*5mm`

Campiono RC3 con l'ADC.

Checklist:

1. Abilito l'output sul RC6 per poter poi accendere il sonar
2. Controllo che il pin sia in modalità di input analogico
3. Setuppo l'ADC
4. Accendo il bit RC6

-