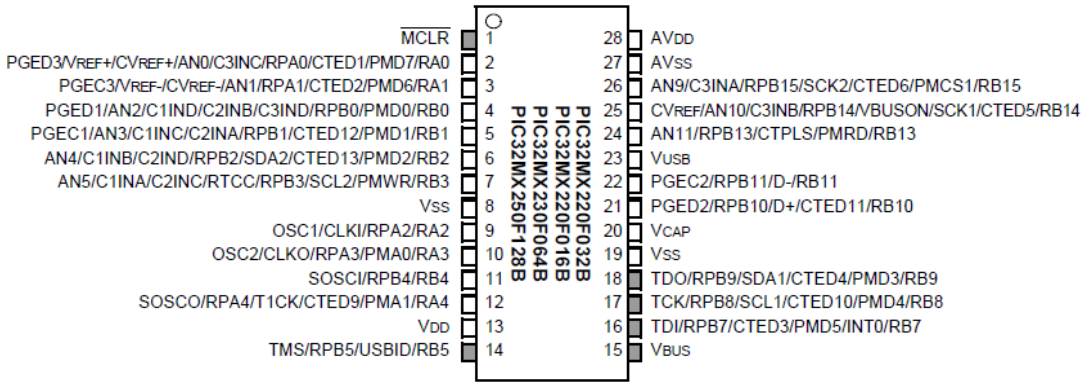
Guida **PIC32MX220F032B**.

La scelta è ricaduta su questo modello per il fatto che viene prodotto su package **DIP** e dunque facilmente utilizzabile.

Pinout O Pinning

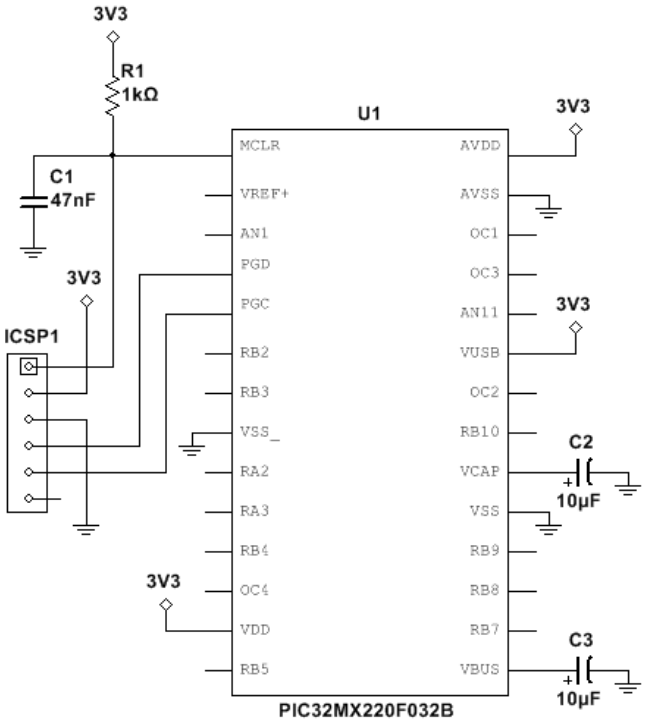
Osserviamo ora un po' più da vicino questo integrato. Ci sono alcune differenze rispetto, ad esempio, ai PIC16.
In particolare sono presenti alcuni piedini che nelle altre versioni più piccole non ci sono:

- AVDD (28)
- AVSS (27)
- VCAP (20)
- VBUS (15)



Primo Utilizzo

Su breadboard >> questo schema.



Da notare che è importantissimo rispettare tutte le connessioni indicate sullo schema.
Questo circuito è testato e funzionante.

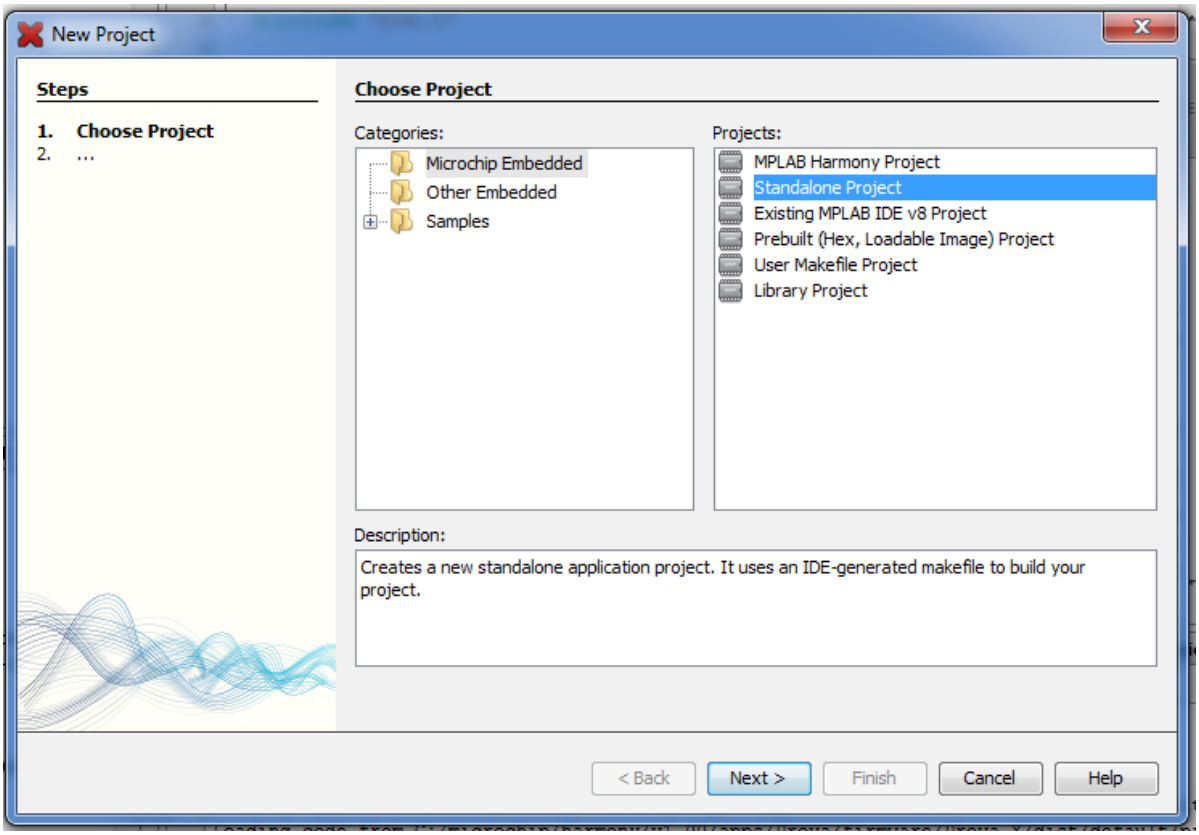
Collegamento Con Il Programmatore **PicKit 3.**

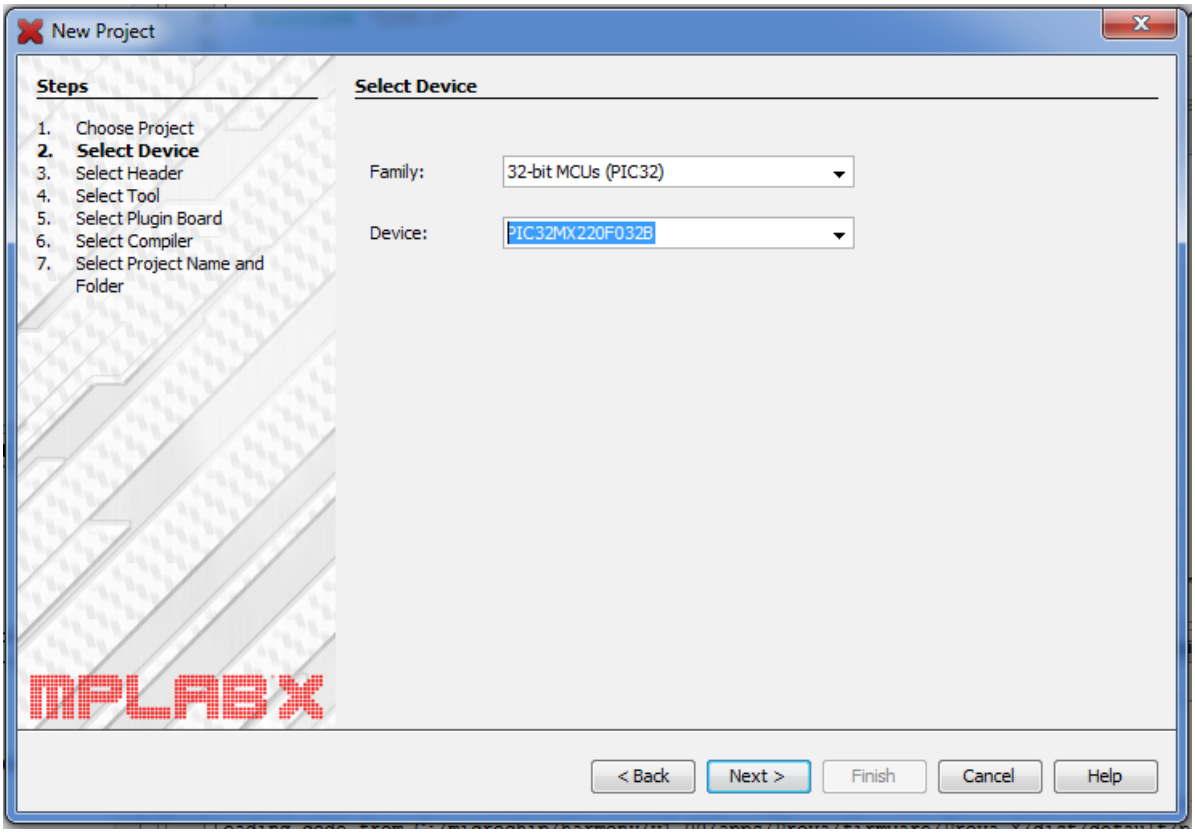
PicKit 3

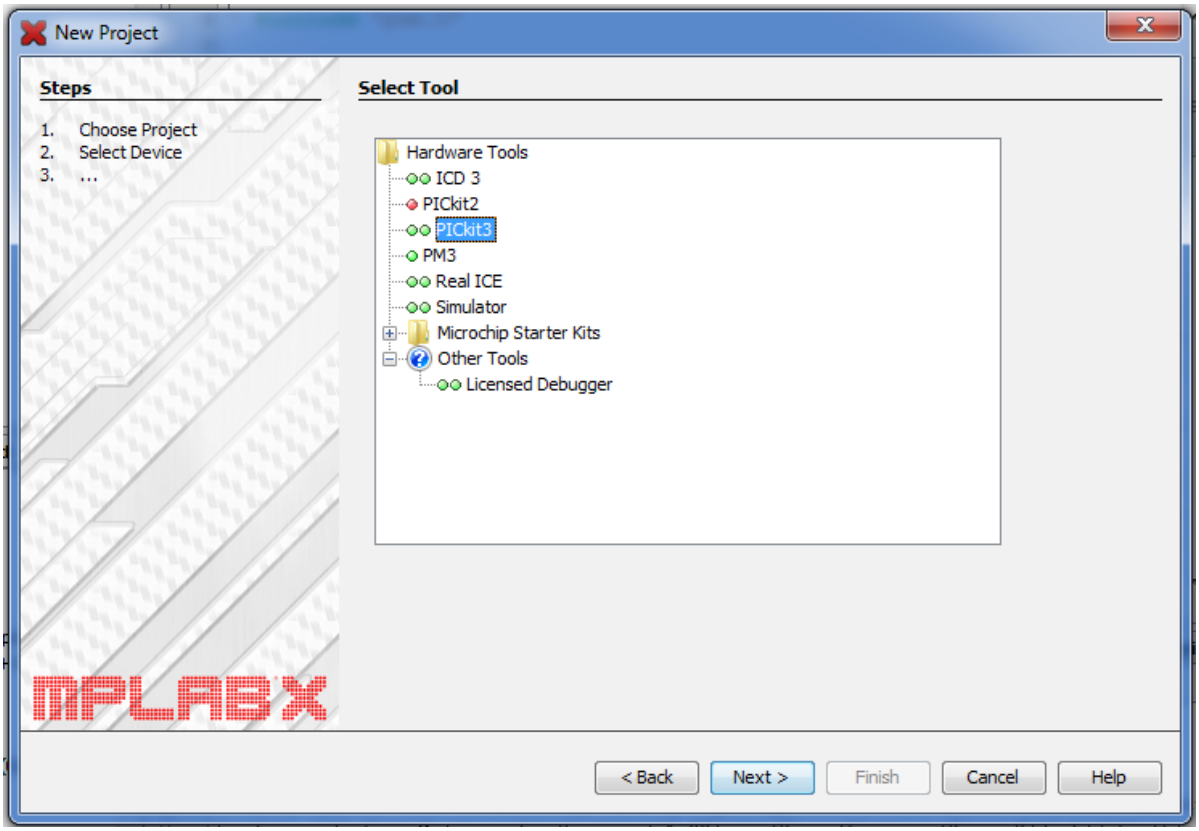
Una volta collegato apriamo MPLAB X col compilatore adatto XC32

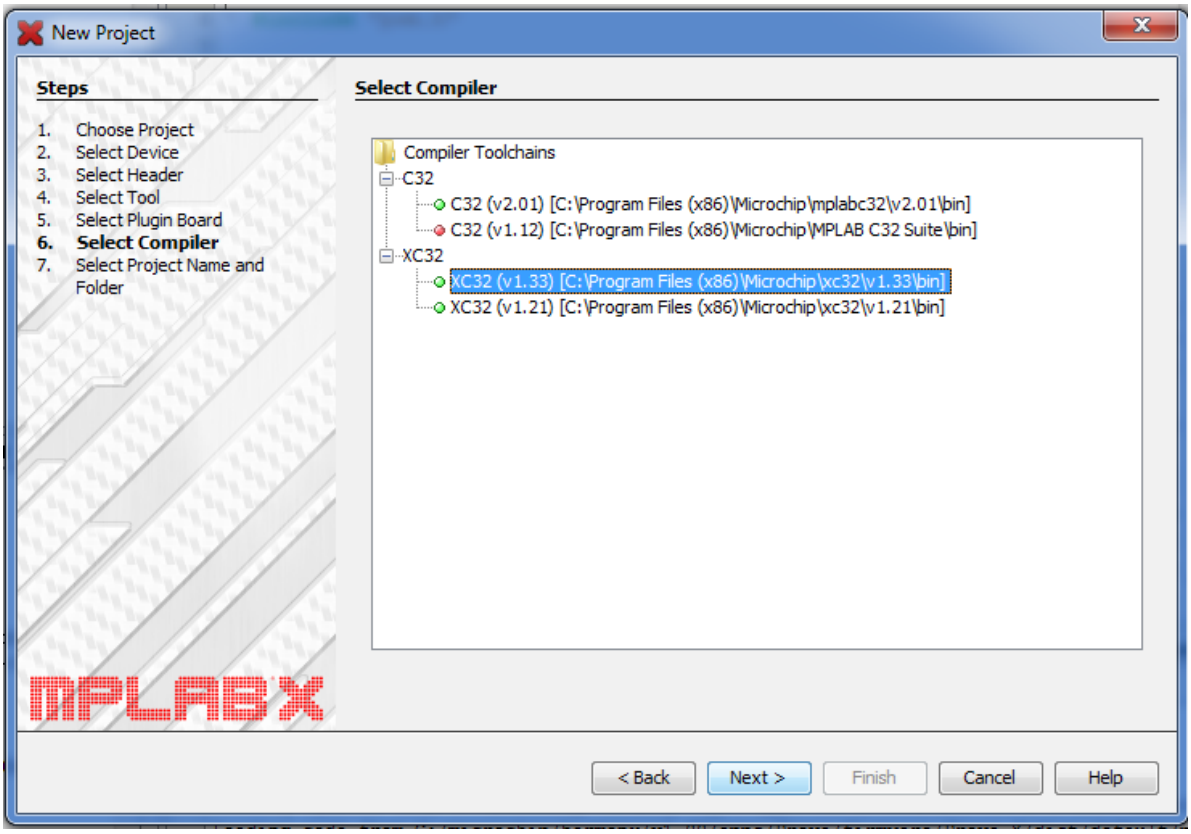
Guida Alla Programmazione

Una volta che ambiente e compilatore saranno pronti creiamo un nuovo progetto.
Seguiamo il wizard in questo modo e avremo così pronta la nostra postazione di lavoro.

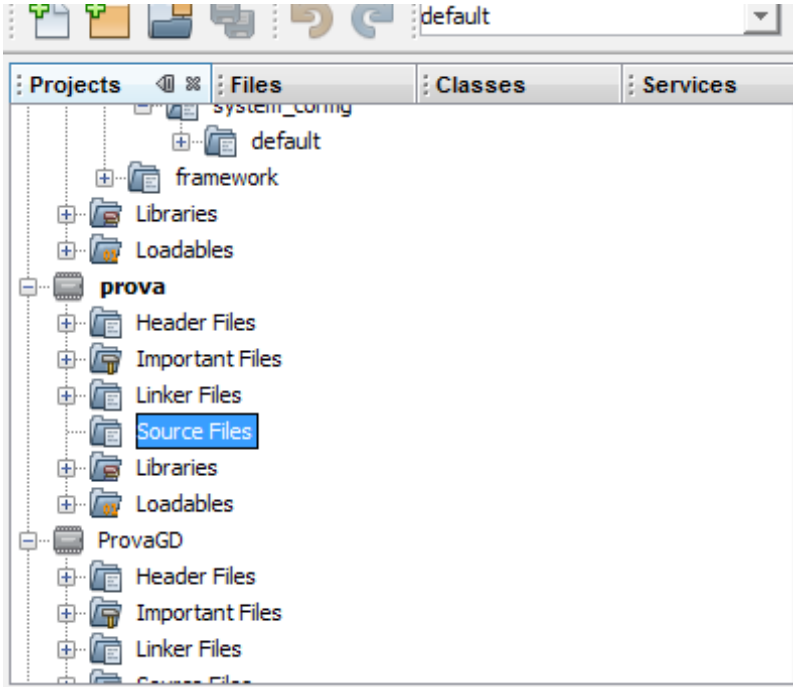



New Project
Steps
1. Choose Project
2. Select Device
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. Select Project Name and Folder
Select Device
Family:
32-bit MCUs (PIC32)
Device:
PIC32MX220F032B
MPLAB X
< Back
Next >
Finish
Cancel
Help

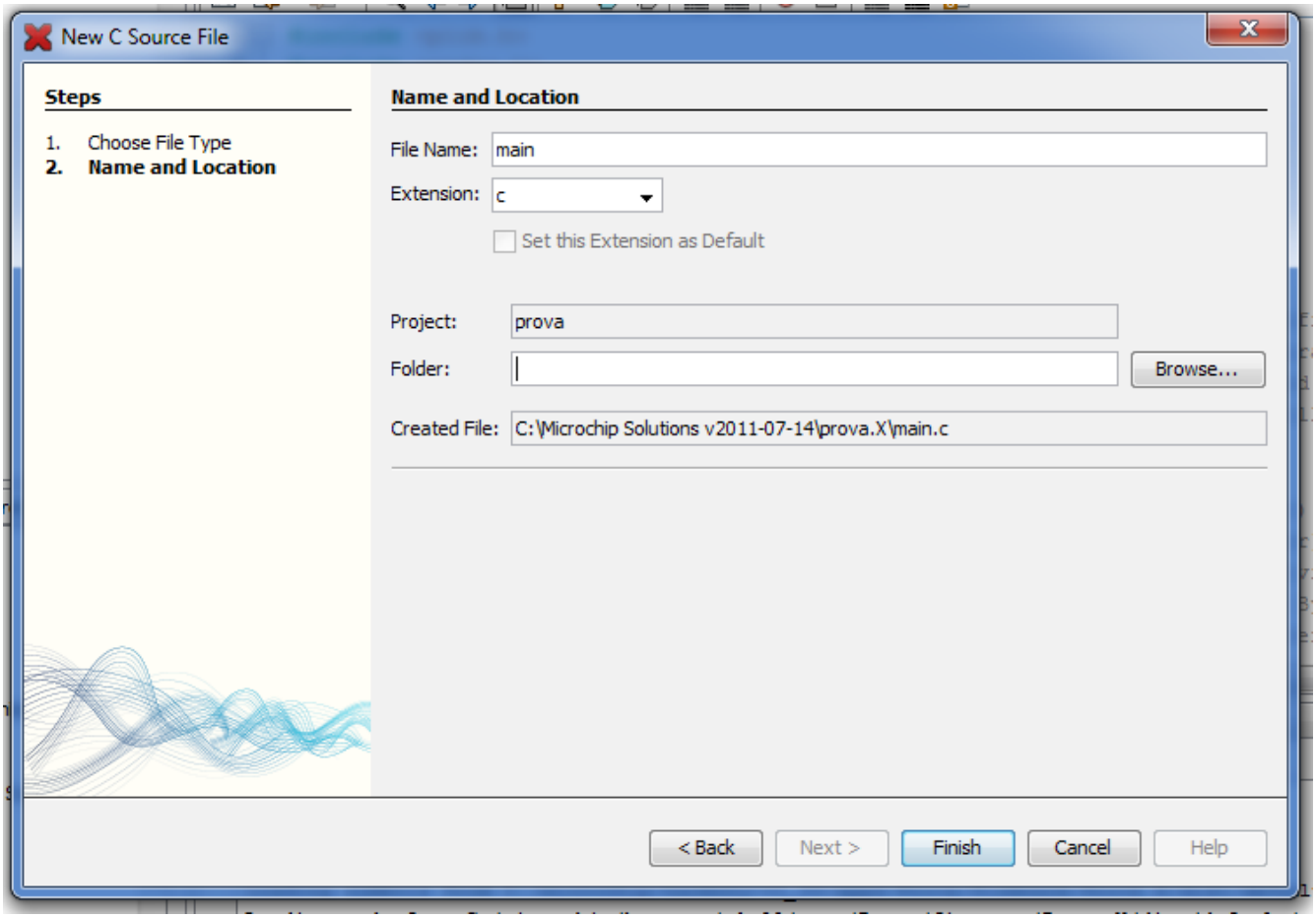

New Project
Steps
1. Choose Project
2. Select Device
3. ...
Select Tool
Hardware Tools
ICD 3
PICkit2
PICkit3
PM3
Real ICE
Simulator
Microchip Starter Kits
Other Tools
Licensed Debugger
MPLAB X
< Back
Next >
Finish
Cancel
Help

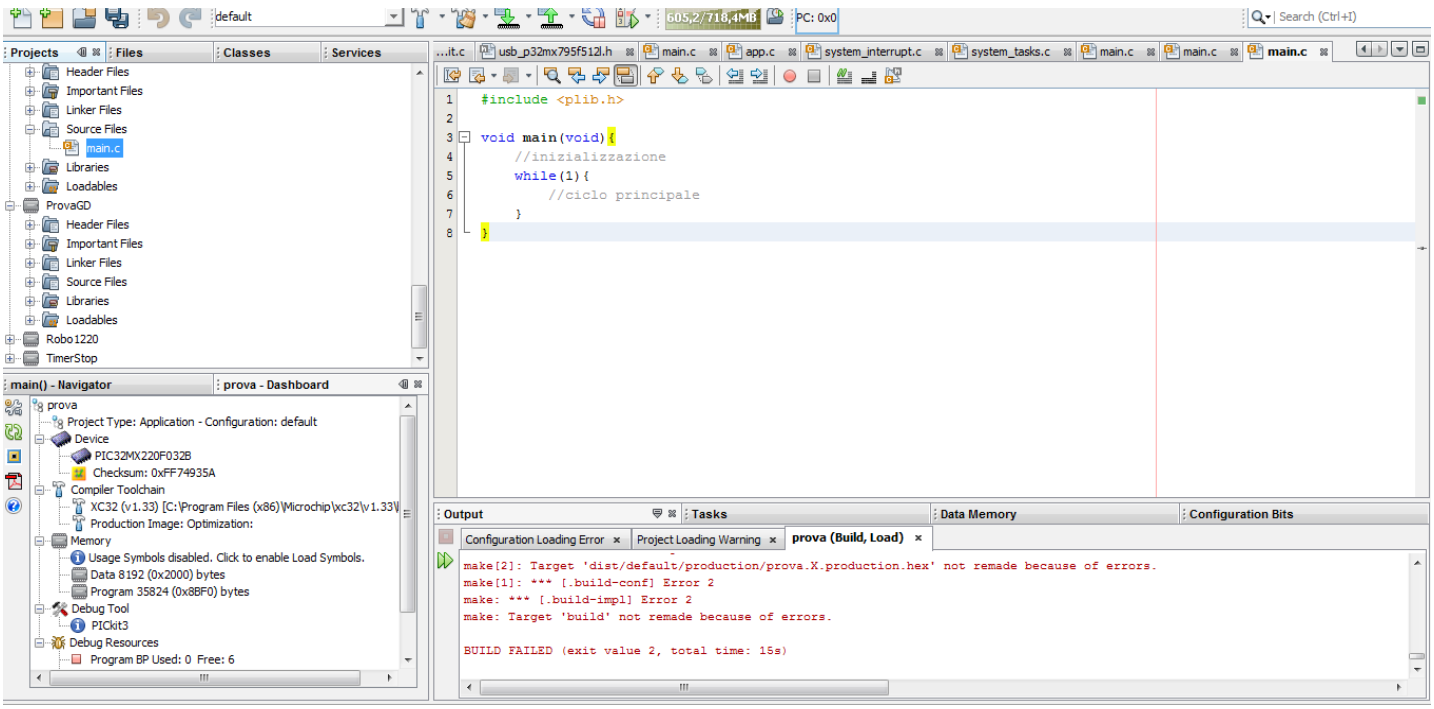


Il nostro progetto è pronto. Ora basta solo creare il file principale **main.c**.



Con un click destro aggiungiamo un **nuovo file C**





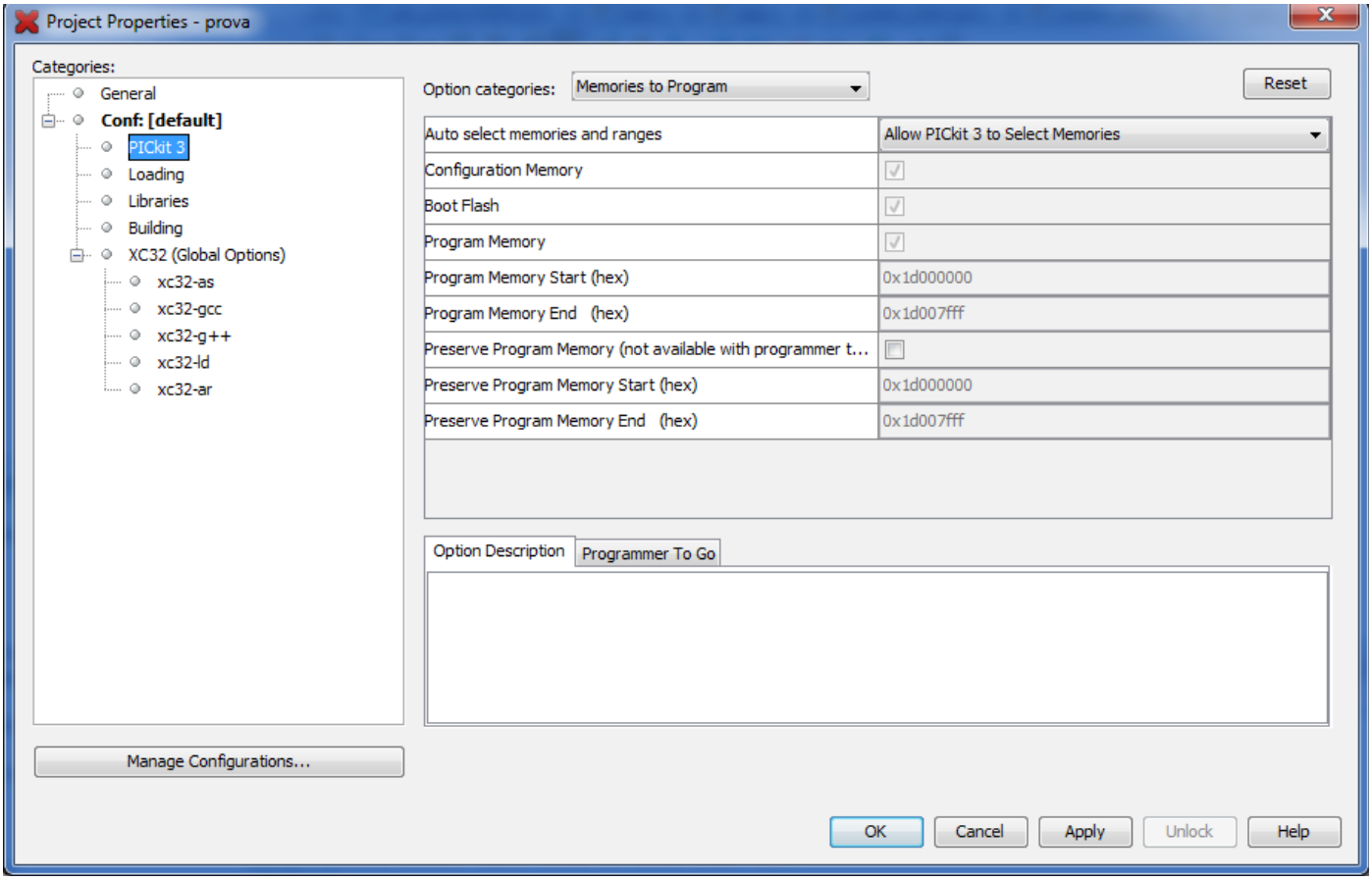
Per trovarci quindi con il nostro file sorgente che andremo così a riempire:

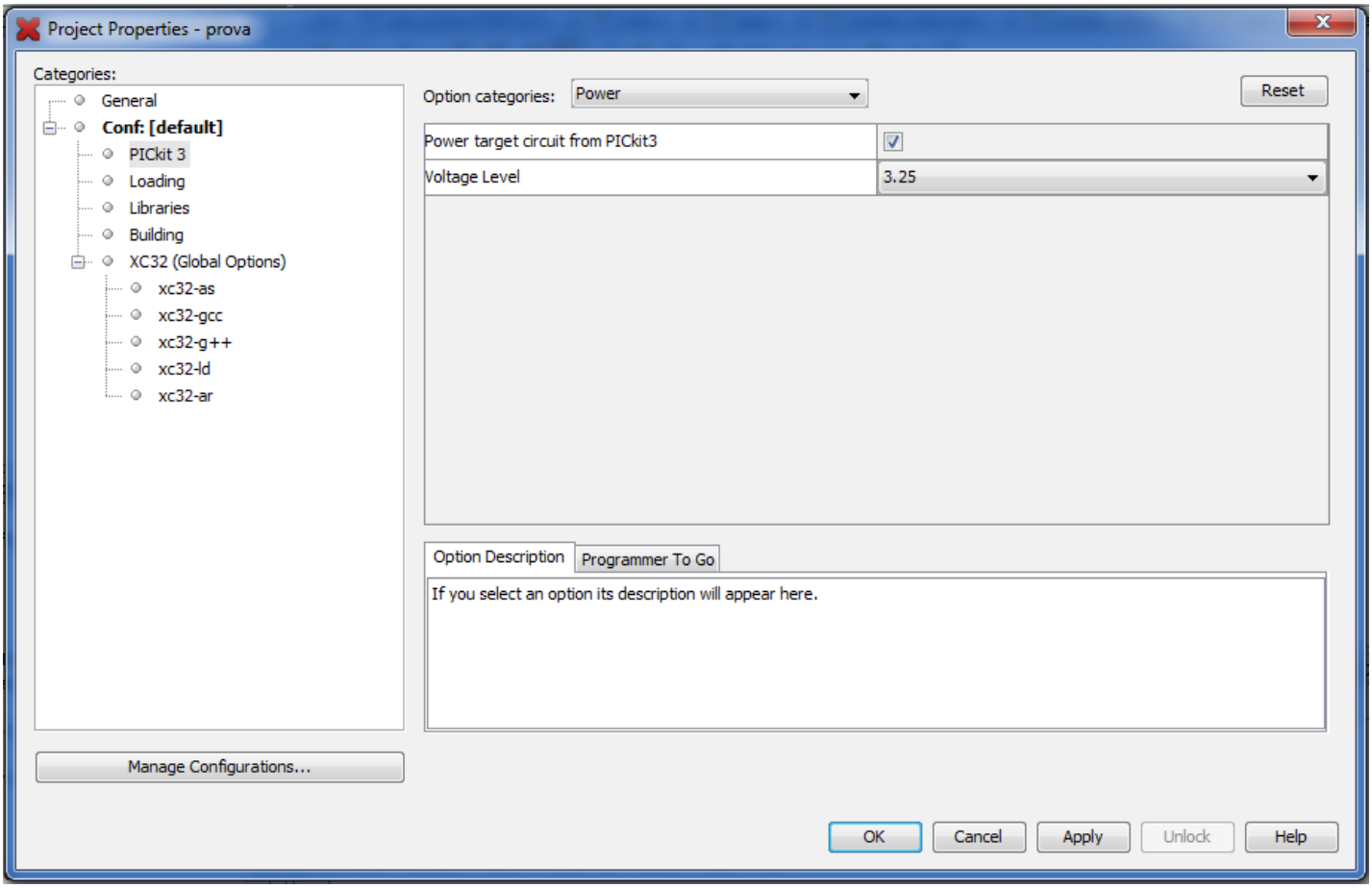
- Includiamo le librerie necessarie
- Impostiamo i **configuration bits** del microcontrollore
- Creiamo il metodo **main**

Dunque andiamo a copiare questo codice nel file appena creato.

```
#include <plib.h>

// DEVCFG3
// USERID = No Setting
#pragma config PMDL1WAY = OFF           // Peripheral Module Disable Configuration (Allow multiple reconfigurations)
#pragma config IOL1WAY = OFF            // Peripheral Pin Select Configuration (Allow multiple reconfigurations)
#pragma config FUSBIDIO = OFF           // USB USID Selection (Controlled by Port Function)
#pragma config FVBUSONIO = OFF          // USB VBUS ON Selection (Controlled by Port Function)

// DEVCFG2
#pragma config FPLLIDIV = DIV_2         // PLL Input Divider (2x Divider)
#pragma config FPLLMUL = MUL_20         // PLL Multiplier (20x Multiplier)
#pragma config UPLLIDIV = DIV_12        // USB PLL Input Divider (12x Divider)
#pragma config UPLLEN = OFF             // USB PLL Enable (Disabled and Bypassed)
#pragma config FPLLODIV = DIV_2         // System PLL Output Clock Divider (PLL Divide by 2)

// DEVCFG1
#pragma config FNOSC = FRCPLL           // Oscillator Selection Bits (Fast RC Osc with PLL)
#pragma config FSOSCEN = OFF            // Secondary Oscillator Enable (Disabled)
#pragma config IESO = OFF               // Internal/External Switch Over (Disabled)
#pragma config POSCMOD = OFF            // Primary Oscillator Configuration (Primary osc disabled)
#pragma config OSCIOFNC = OFF           // CLKO Output Signal Active on the OSCO Pin (Disabled)
#pragma config FPBDIV = DIV_1           // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/1)
#pragma config FCKSM = CSDCMD           // Clock Switching and Monitor Selection (Clock Switch Disable, FSCM Disabled)
#pragma config WDTPS = PS1048576        // Watchdog Timer Postscaler (1:1048576)
#pragma config WINDIS = OFF             // Watchdog Timer Window Enable (Watchdog Timer is in Non-Window Mode)
#pragma config FWDTEN = OFF             // Watchdog Timer Enable (WDT Disabled (SWDTEN Bit Controls))
#pragma config FWDTWINSZ = WISZ_25      // Watchdog Timer Window Size (Window Size is 25%)

// DEVCFG0
#pragma config JTAGEN = OFF             // JTAG Enable (JTAG Disabled)
#pragma config ICESEL = ICS_PGx1        // ICE/ICD Comm Channel Select (Communicate on PGEC1/PGED1)
#pragma config PWP = OFF                // Program Flash Write Protect (Disable)
#pragma config BWP = OFF                // Boot Flash Write Protect bit (Protection Disabled)
#pragma config CP = OFF                 // Code Protect (Protection Disabled)

void main(void){
 //inizializzazione
 while(1){
 //ciclo principale
 }
}
```

Siamo ora pronti a programmare il microcontrollore.
Ovviamente, come abbiamo visto nello schema, l'alimentazione verrà presa direttamente dal programmatore.
Per abilitare questa funzione apriamo le **proprietà** del progetto.





Guida PIC32MX - Lezione 2

Dopo una lunga fase di preparazione siamo pronti a scrivere il nostro primo programma. Nulla di che, ovviamente, ma ci darà la possibilità di verificare se tutto funziona correttamente.
Riprendiamo proprio da dove ci eravamo fermati.
Abbiamo visto che il microcontrollore necessita di alcuni parametri di settaggio, detti **configuration bits**.
Nei paragrafo seguente cercheremo di capire come mai è stata proposta la configurazione precedente.

Configuration Bits

Chi ha già dimestichezza con in microcontrollori Microchip potrà tranquillamente tralasciare questo paragrafo perchè il discorso è simile a tutti gli altri modelli. Ci tengo però ad affrontare, anche se in breve, questo argomento perchè lo ritengo fondamentale.

```
1#pragma config PMDL1WAY = OFF          // Peripheral Module Disable Configuration (Allow multiple reconfigurations)
2#pragma config IOL1WAY = OFF           // Peripheral Pin Select Configuration (Allow multiple reconfigurations)
3#pragma config FUSBIDIO = OFF          // USB USID Selection (Controlled by Port Function)
4#pragma config FVBUSONIO = OFF         // USB VBUS ON Selection (Controlled by Port Function)
```

Nelle prime due righe si specifica la possibilità di configurare più volte le varie funzioni (SPI, PWM, ecc) da assegnare ai vari pin. Le altre due, invece, riguardano il modulo **USB** che probabilmente descriverò in un altra lezione.
Per un'applicazione che fa uso dell' USB potrete comunque vedere un articolo che ho scritto riguardo ai PIC18F.

```
1#pragma config FPLLIDIV = DIV_2        // PLL Input Divider (2x Divider)
2#pragma config FPLLMUL = MUL_20        // PLL Multiplier (20x Multiplier)
3#pragma config UPLLIDIV = DIV_12       // USB PLL Input Divider (12x Divider)
4#pragma config UPLLEN = OFF            // USB PLL Enable (Disabled and Bypassed)
5#pragma config FPLLODIV = DIV_2        // System PLL Output Clock Divider (PLL Divide by 2)
```

Tralasciando **UPLLIDIV** e **UPLLEN** che riguardano sempre il lato USB ci concentriamo sugli altri 3 settaggi. Questi riguardano il **PLL** che fornirà la frequenza di clock al microcontrollore. Il segnale fornito dall'**oscillatore** selezionato verrà diviso 2 volte, moltiplicato per 2o ed infine diviso 2 volte. **Si prenderà quindi tale segnale e lo si moltiplicherà per 5**.

```
1#pragma config FNOSC = FRCPLL          // Oscillator Selection Bits (Fast RC Osc with PLL)
2#pragma config FSOSCEN = OFF           // Secondary Oscillator Enable (Disabled)
3#pragma config IESO = OFF              // Internal/External Switch Over (Disabled)
4#pragma config POSCMOD = OFF           // Primary Oscillator Configuration (Primary osc disabled)
5#pragma config OSCIOFNC = OFF          // CLKO Output Signal Active on the OSCO Pin (Disabled)
6#pragma config FPBDIV = DIV_1          // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/1)
7#pragma config FCKSM = CSDCMD          // Clock Switching and Monitor Selection (Clock Switch Disable, FSCM Disabled)
8#pragma config WDTPS = PS1048576       // Watchdog Timer Postscaler (1:1048576)
9#pragma config WINDIS = OFF            // Watchdog Timer Window Enable (Watchdog Timer is in Non-Window Mode)
10#pragma config FWDTEN = OFF           // Watchdog Timer Enable (WDT Disabled (SWDTEN Bit Controls))
11#pragma config FWDTWINSZ = WISZ_25    // Watchdog Timer Window Size (Window Size is 25%)
```

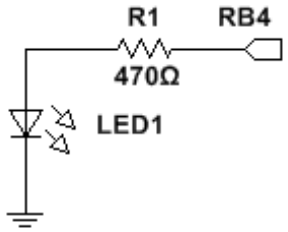
Con queste righe andiamo a specificare il tipo di **oscillatore primario FNOSC** (interno con pll), secondario **FSOSCEX** (disabilitato) e qualche altra impostazione per ora trascurabile sempre riguardante il clock (**IESO**, **POSCMOD**, **OSCIOFNC**). Inoltre sarà possibile far funzionare le periferiche quali PWM, PMP, ecc ad una frequenza inferiore a quella dell'oscillatore primario, questo tramite **FPBDIV**. Gli ultimi 5 parametri riguardano una funzione che andremo ad approfondire nelle prossime lezioni, ovvero il **WATCHDOG** (letteralmente cane da guardia). Questa funzione altro non fa che resettare il micro in caso di malfunzionamento software.

```
1#pragma config JTAGEN = OFF            // JTAG Enable (JTAG Disabled)
2#pragma config ICESEL = ICS_PGx1       // ICE/ICD Comm Channel Select (Communicate on PGEC1/PGED1)
3#pragma config PWP = OFF               // Program Flash Write Protect (Disable)
4#pragma config BWP = OFF               // Boot Flash Write Protect bit (Protection Disabled)
5#pragma config CP = OFF                // Code Protect (Protection Disabled)
```

Passiamo ora all'ultima parte della configurazione. **PWP**, **BWP**, **CP** riguardano la protezione del codice. **JTAGEN** abilita la porta di programmazione **JTAG**, ma noi non la useremo. Novità rispetto ai microcontrollori più piccolini è la possibilità di selezionare quali piedini avranno la funzione di PGD e PGC per la programmazione. In questo caso, tramite **ICESEL** abbiamo scelto **4** e **5**.

Accendere Un LED

Alla "noiosa" descrizione dei parametri di configurazione segue la preparazione hardware. In questo caso è molto semplice. Consideriamo un **pin di I/O**, ad esempio **RB4 (pin 11)**.



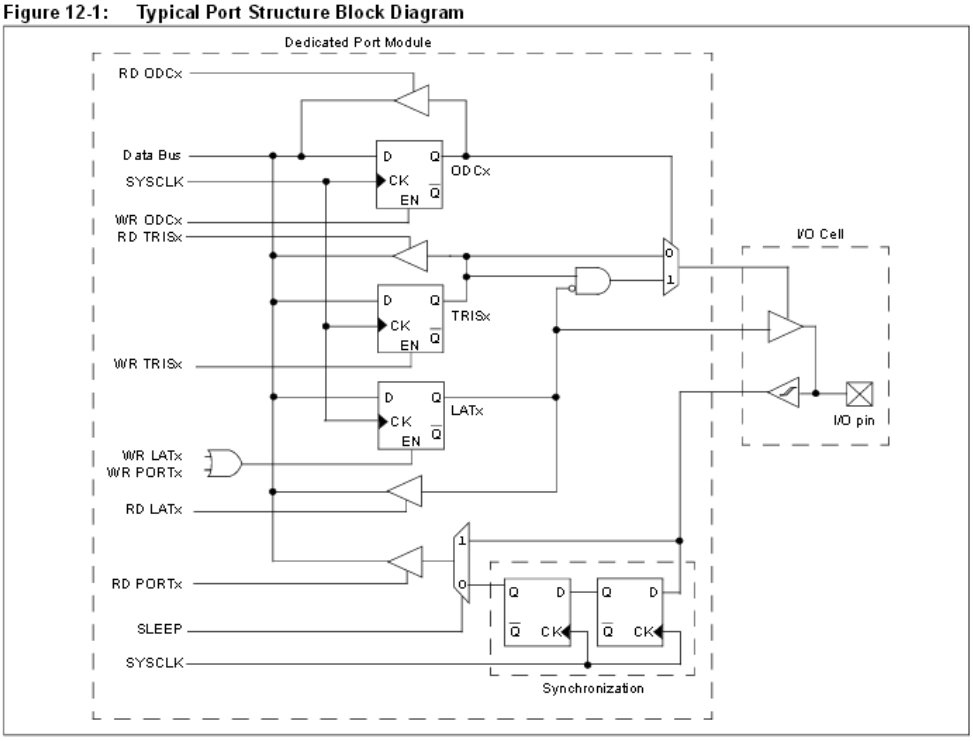
Il nostro scopo sarà quello di pilotare il **LED** facendolo **lampeggiare**.
Ci vorremmo trovare quindi un segnale ad onda quadra con periodo dell'ordine del secondo.
Per ottenere questo risultato procederò **razionalmente** (solo per questa volta).
Vediamo quindi nel dettaglio cosa deve fare il programma:

1. Configurare il microcontrollore (Configuration bits)
2. Setup dei registri che controllano le periferiche (in questo consideriamo come periferica l'I/O)
3. Accendere il LED
4. Aspettare un tempo T
5. Spegnere il LED
6. Aspettare un tempo T
7. Ritornare al punto 3

Porta I/O

Dovremo quindi scrivere su un pin di i/o. Ad esso si accede mediante alcuni registri che ci troveremo già mappati (con un nome facile da ricordare). Il costruttore fornisce lo schema logico del generico piedino.

Figure 12-1: Typical Port Structure Block Diagram



Ad ogni pin sono associati 3 bit:

- **TRISx** che permette di scegliere se il piedino è di ingresso oppure di uscita
- **PORTx** per leggere tale piedino, se TRISx è posto a 1 (ingresso)
- **LATx** per scrivere, se TRISx è posto a 0 (uscita)

Delay

Siamo pronti per scrivere un po' di codice. Iniziamo con la funzione di ritardo. La mia proposta di soluzione è un po' sperimentale. In sostanza dovremo scrivere un ciclo che faccia una serie di operazioni nulle. Il problema risiede nello stimare quante di queste operazione andranno fatte affinchè passi circa mezzo secondo.

Sappiamo che il clock è a **40Mhz**, dunque scriviamo:

```
1void delayHalfSecond(){
2 int i = 10000000;
3 while(i>0) i--;
4}
```

La mia ipotesi è che la CPU lavori ad **1.5 MIPS/Mhz**, ovvero a **60MIPS**.
Dunque in un secondo eseguo 60 milioni di operazioni.
Ad ogni iterazione eseguo tre operazioni (confronto, decremento e salto), quindi mi trovo a 20 milioni di iterazioni al secondo.
Perciò affinchè trascorra mezzo secondo dovrei eseguire 10 milioni di tali iterazioni.
Questo è vero solo senza recupero operandi, ovvero se la variabile i è tenuta in **cache**.
Resta perciò da verificare se i conti che ho fatto sono validi sperimentalmente.

Il Programma
L'inizializzazione del dispositivo andrà opportunamente separata dal programma principale.

```
1//libreria in cui sono mappati i vari registri
2#include <plib.h>
3
4// DEVCFG3
5// USERID = No Setting
6#pragma config PMDL1WAY = OFF           // Peripheral Module Disable Configuration (Allow multiple reconfigurations)
7#pragma config IOL1WAY = OFF            // Peripheral Pin Select Configuration (Allow multiple reconfigurations)
8#pragma config FUSBIDIO = OFF           // USB USID Selection (Controlled by Port Function)
9#pragma config FVBUSONIO = OFF          // USB VBUS ON Selection (Controlled by Port Function)
10
11// DEVCFG2
12#pragma config FPLLIDIV = DIV_2         // PLL Input Divider (2x Divider)
13#pragma config FPLLMUL = MUL_20         // PLL Multiplier (20x Multiplier)
14#pragma config UPLLIDIV = DIV_12        // USB PLL Input Divider (12x Divider)
15#pragma config UPLLEN = OFF             // USB PLL Enable (Disabled and Bypassed)
16#pragma config FPLLODIV = DIV_2         // System PLL Output Clock Divider (PLL Divide by 2)
17
18// DEVCFG1
19#pragma config FNOSC = FRCPLL           // Oscillator Selection Bits (Fast RC Osc with PLL)
20#pragma config FSOSCEN = OFF            // Secondary Oscillator Enable (Disabled)
21#pragma config IESO = OFF               // Internal/External Switch Over (Disabled)
22#pragma config POSCMOD = OFF            // Primary Oscillator Configuration (Primary osc disabled)
23#pragma config OSCIOFNC = OFF           // CLKO Output Signal Active on the OSCO Pin (Disabled)
24#pragma config FPBDIV = DIV_1           // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/1)
25#pragma config FCKSM = CSDCMD           // Clock Switching and Monitor Selection (Clock Switch Disable, FSCM Disabled)
26#pragma config WDTPS = PS1048576        // Watchdog Timer Postscaler (1:1048576)
27#pragma config WINDIS = OFF             // Watchdog Timer Window Enable (Watchdog Timer is in Non-Window Mode)
28#pragma config FWDTEN = OFF             // Watchdog Timer Enable (WDT Disabled (SWDTEN Bit Controls))
29#pragma config FWDTWINSZ = WISZ_25      // Watchdog Timer Window Size (Window Size is 25%)
30
31// DEVCFG0
32#pragma config JTAGEN = OFF             // JTAG Enable (JTAG Disabled)
33#pragma config ICESEL = ICS_PGx1        // ICE/ICD Comm Channel Select (Communicate on PGEC1/PGED1)
34#pragma config PWP = OFF                // Program Flash Write Protect (Disable)
35#pragma config BWP = OFF                // Boot Flash Write Protect bit (Protection Disabled)
36#pragma config CP = OFF                 // Code Protect (Protection Disabled)
37
38//ritardo di mezzo secondo
39void delayHalfSecond(){
40    int i = 10000000;
41    while(i>0) i--;
42}
43
44//inizializzazione del dispositivo
45void setup(void){
46    TRISBbits.TRISB4 = 0; //RB4 è un output
47    LATBbits.LATB4 = 0; //Inizializzo RB4 a 0
48}
49
50//programma principale
51void main(void){
52    setup(); //chiamo il setup
53    while(1){
54        //ciclo infinito
55        LATBbits.LATB4 = 1; //accendo il led
56        delayHalfSecond();  //aspetto mezzo secondo
57        LATBbits.LATB4 = 0; //spengo il led
58        delayHalfSecond();  //aspetto mezzo secondo
59    }
60}
```

Siamo quindi riusciti ad eseguire il nostro primo programma.
Molto semplice, ma che al tempo stesso ci fa capire come ragionare per scrivere programmi un po' più complessi.
L'importante è tenere a mente la struttura del ciclo principale.
Contrariamente ad altri programmi, il **ciclo infinito** è indispensabile.
Infatti una volta terminato il programma avviene il reset del microcontrollore, cosa molto indesiderata!