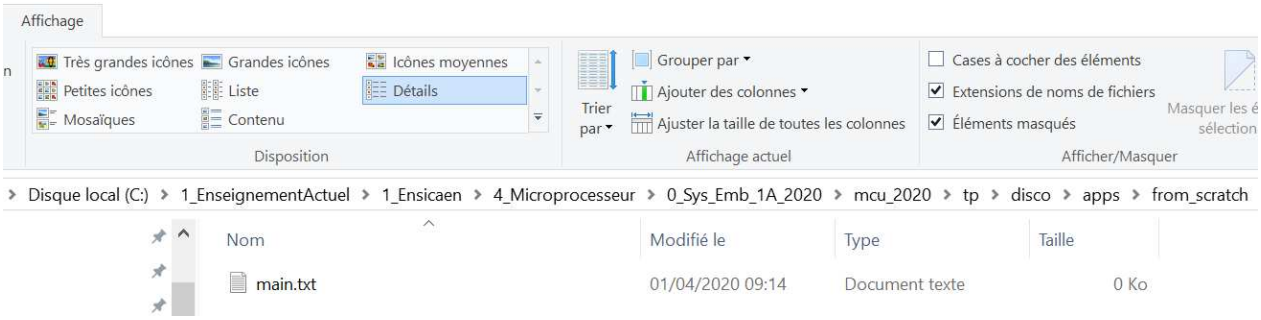
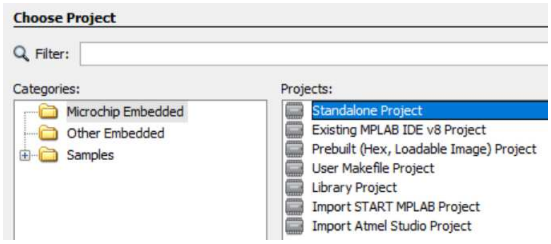
From Scratch

- Créer un nouveau fichier texte dans le dossier **disco/apps/fromscratch** nommé **main.txt**

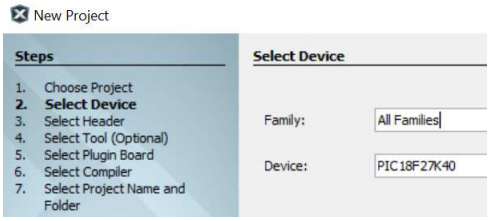


- Modifier l'extension en **.c** 

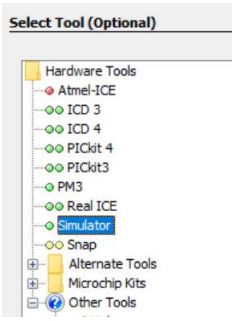
- Ouvrir MPLABx et créer un nouveau projet à partir de zéro (Standalone Project)



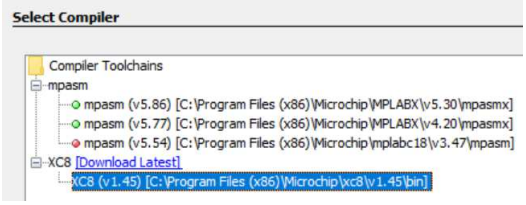
- Choisir le bon composant cible



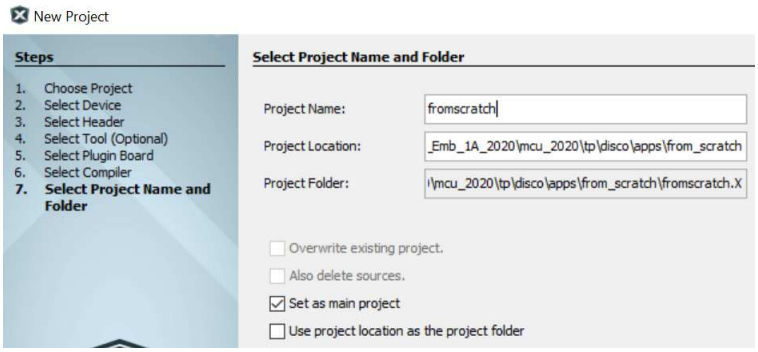
- Sélectionner le simulateur comme outil de **debug** (pour travailler sans maquette)



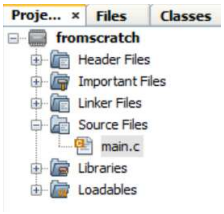
- Choisir la chaine de compilation XC8 (compilateur pour MCU 8 bits)



- Nommer le projet **fromscratch** et le placer dans le dossier **disco/apps/fromscatch**



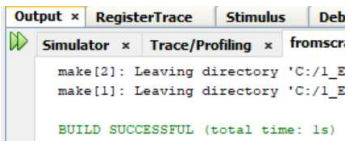
- Ajouter le fichier **main.c** au projet



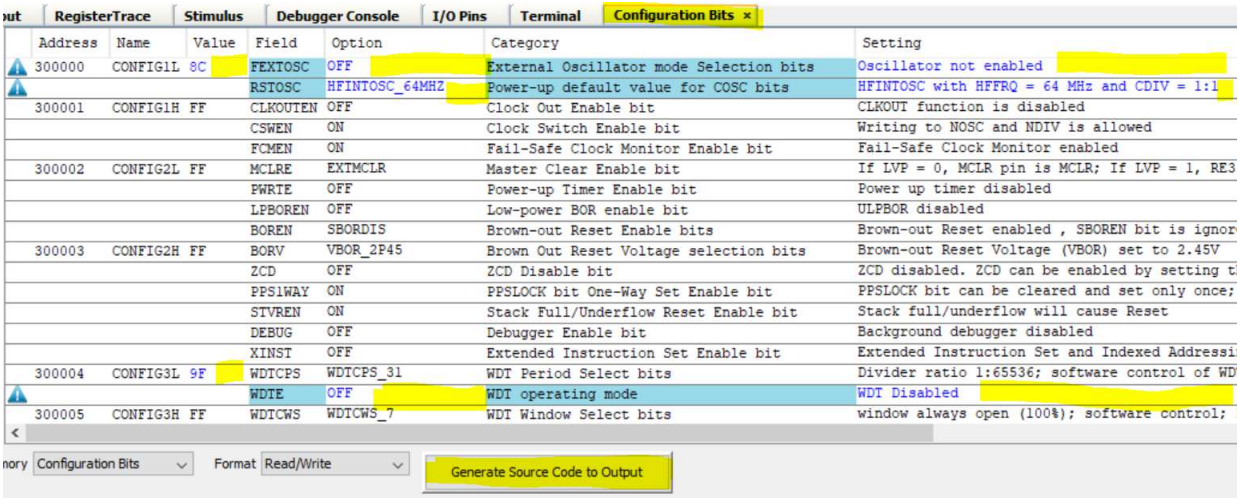
- Compléter le fichier

```c
#include <xc.h>

main()
{
    while(1)
    {
    }
}
```

- Compiler  et vérifier que c'est un succès dans la fenêtre **output**



- Menu déroulant **window > target memory views > configuration bits** pour choisir l'horloge et 2 ou 3 autres configurations

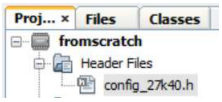


- Appuyer sur **Generate Source Code to Output**

imulator × | Trace/Profiling × | fromscratch (Build, Load) × | Config Bits Source ×

```
// PIC18F27K40 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1L
#pragma config FEXTOSC = OFF    // External Oscillator mode Selection bits (Oscillator not enabled)
#pragma config RSTOSC = HFINTOSC_64MHZ// Power-up default value for COSC bits (HFINTOSC with HFFRQ = 64 MHz and CDIV = 1:1)

// CONFIG1H
#pragma config CLKOUTEN = OFF   // Clock Out Enable bit (CLKOUT function is disabled)
#pragma config CSWEN = ON       // Clock Switch Enable bit (Writing to NOSC and NDIV is allowed)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor enabled)

// CONFIG2L
#pragma config MCLRE = EXTMCLR  // Master Clear Enable bit (If LVP = 0, MCLR pin is MCLR; If LVP = 1, RE3 pin function is MCLR )
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (Power up timer disabled)
#pragma config LPBOREN = OFF    // Low-power BOR enable bit (ULPBOR disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled , SBOREN bit is ignored)
```

- Copier tout le code généré (ctrl A, ctrl C)
- Créer un nouveau fichier de type **xc8_header.h** et le nommer **config_27k40.h** (ce fichier doit être dans le même dossier que **main.c**)



- Coller dedans ce qui a été copié juste avant

Début

```
#ifndef XC_HEADER_config_H
#define XC_HEADER_config_H

// CONFIG1L
#pragma config FEXTOSC = OFF    // Externa
#pragma config RSTOSC = HFINTOSC_64MHZ// I

// CONFIG1H
#pragma config CLKOUTEN = OFF   // Clock (
#pragma config CSWEN = ON       // Clock S
#pragma config FCMEN = ON       // Fail-Sa

// CONFIG2L
#pragma config MCLRE = EXTMCLR  // Master
#pragma config PWRTE = OFF      // Power-u
#pragma config LPBOREN = OFF    // Low-pow
#pragma config BOREN = SBORDIS  // Brown-o

// CONFIG2H
#pragma config BORV = VBOR_2P45 // Brown (
#pragma config ZCD = OFF        // ZCD Dis
#pragma config PPS1WAY = ON     // PPSLOCI
#pragma config STVREN = ON      // Stack I
#pragma config DEBUG = OFF      // Debugge
#pragma config XINST = OFF      // Extende
```

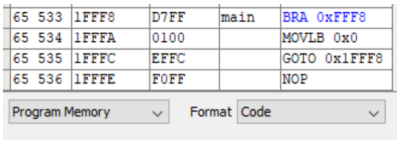
fin

```
// CONFIG5L
#pragma config CP = OFF         // Us
#pragma config CPD = OFF        // Da

// CONFIG5H

// CONFIG6L
#pragma config EBTR0 = OFF      // Ta
#pragma config EBTR1 = OFF      // Ta
#pragma config EBTR2 = OFF      // Ta
#pragma config EBTR3 = OFF      // Ta
#pragma config EBTR4 = OFF      // Ta
#pragma config EBTR5 = OFF      // Ta
#pragma config EBTR6 = OFF      // Ta
#pragma config EBTR7 = OFF      // Ta

// CONFIG6H
#pragma config EBTRB = OFF      // Bo

// #pragma config statements should p
// Use project enums instead of #defi

#include <xc.h>

#endif  /* XC_HEADER_TEMPLATE_H */
```

- L'inclure dans la fonction **main**

```
#include <xc.h>
#include "config_27k40.h"

main()
{
    while(1)
    {
    }
}
```

- **Debug Main Project** 
- **window > target memory views > Memory Program :** analyser le programme assembleur



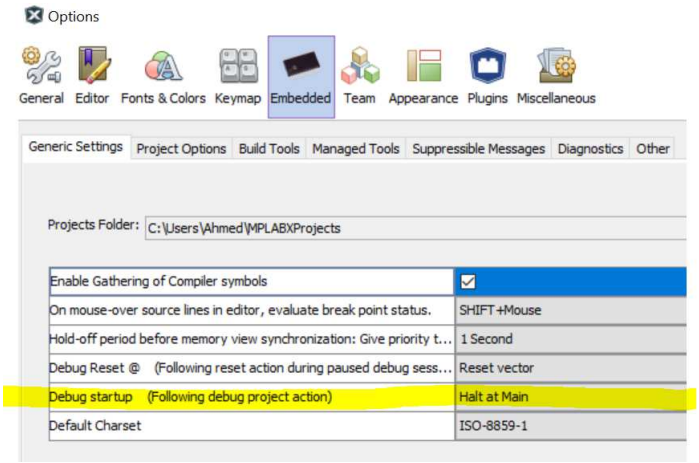
65	533	1FFF8	D7FF	main	BRA 0xFFF8
65	534	1FFFA	0100		MOVLB 0x0
65	535	1FFFC	EFFC		GOTO 0x1FFF8
65	536	1FFFE	F0FF		NOP

Program Memory | Format Code

- Compléter le programme pour allumer les LED D2, D3, D4 et D5 à tour de rôle

```
main()
{
    TRISA = 0x00; //PORTA en sortie (LED))
    LATA = 0x00;    //LED éteintes
    while(1)
    {
        LATA = 0b00010000;
        LATA = 0b00100000;
        LATA = 0b01000000;
        LATA = 0b10000000;
    }
}
```

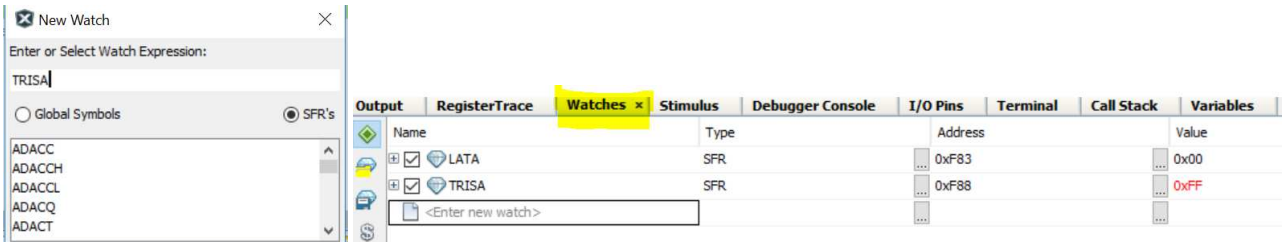
- Menu déroulant (Tools >> Options) pour arrêter le **debuger** au début du **main** au démarrage



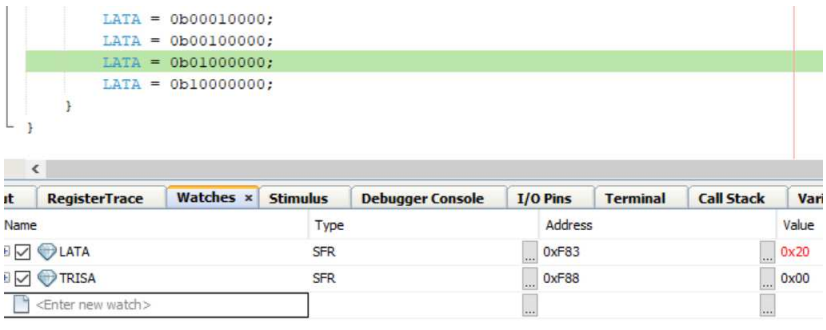
- Maintenant, faire : **Debug Main Project**, le debuger doit s'arrêter au début du main.

```
4   main()
5   {
6       TRISA = 0x00; //PORTA en sortie (LED))
7       LATA = 0x00;    //LED éteintes
8       while(1)
9       {
10          LATA = 0b00010000;
11          LATA = 0b00100000;
12          LATA = 0b01000000;
13          LATA = 0b10000000;
14      }
15  }
```

- **Debug > New Watch** pour observer l'état des registres et variables. Ajouter les registres LATA et TRISA

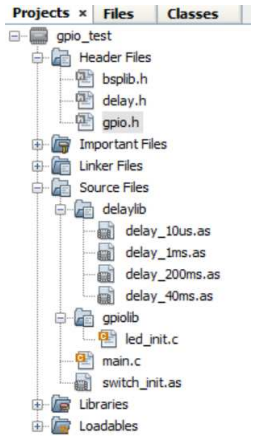


- **Step Over** (F8) pour une exécution pas à pas. Observer l'évolution de LATA à chaque pas



Programme gpio_test, partie LED

- Créer un nouveau projet nommé **gpio_test** dans le dossier **gpio/test/pjct**
- Associer les fichiers sources selon l'arborescence suivante :



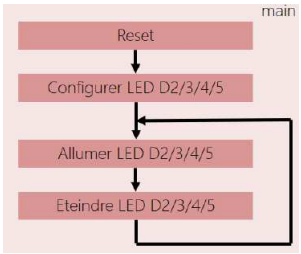
- Commencer par compléter le fichier **gpio.h** (partie LED)

```
#define led2_on() // set output PORT bit : asm("...,...")

#define led3_on() ...

#define led4_on() ...

#define led5_on() ...

#define led2_off() // clear output PORT bit

#define led3_off() ...

#define led4_off() ...

#define led5_off() ...
```

- Continuer avec **led_init.c** (même travail pour toutes les LED)

```
void led2_init(void)
{
    /* TODO */

//->    LATAbits...=...
//->    TRISAbits...=...
}
```

- Programmer l'algorithme du polycopié de TP



- Tester le fonctionnement au simulateur

Programme gpio_test, partie poussoirs

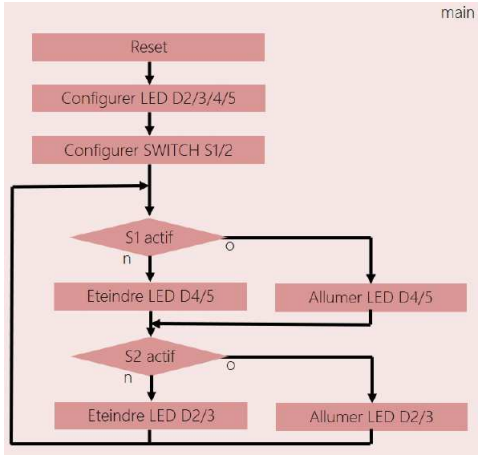
- Compléter le fichier **gpio.h** (partie switches)

```
#define switch1_read() // TRUE if PORT_bit_input = 1

#define switch2_read() ...
```

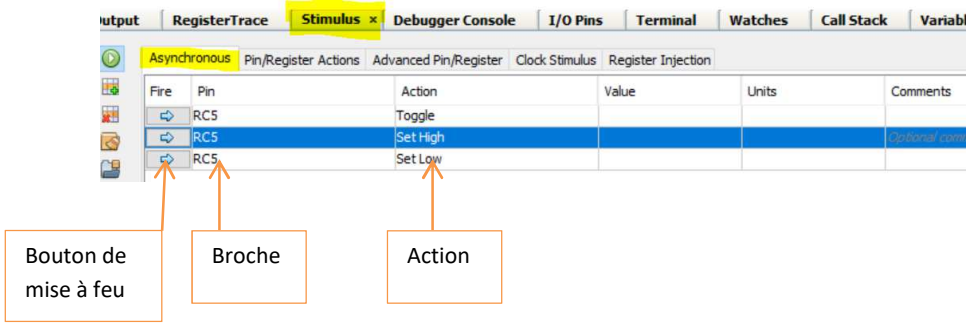
- Continuer avec **switch_init.as** (même travail pour les 2 switchs)

```
_switch1_init:
    ; TODO
//->    strore BSR in bsr_tmp
//->    configure switch1 as input (TRISB...)
//->    select ANSELB bank
//->    configure switc1 as digital (ANSELB...)
//->    restore BSR register from bsr_tmp
    return
```

- Programmer l'algorithme du TP



- Pour simuler les boutons poussoirs (entrées logiques sur les broches du PIC)



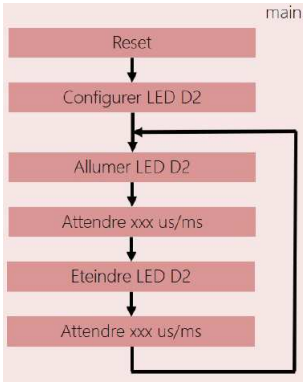
- **Step Over** (F8) pour une exécution pas à pas. Basculer entre la fenêtre précédente et la suivante pour observer l'effet des poussoirs sur les LED

Délai logiciel

- Coder la fonction **delay_10us.as**

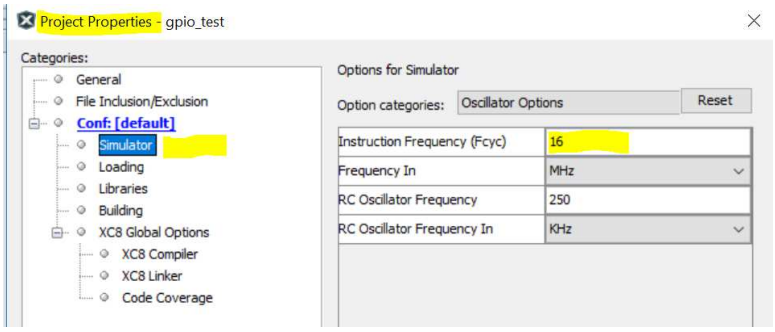
```
PSECT bss0,class=BANK0,space=1

delay_cnt1:  ds 1   ; 1 byte static memory allocation
bsr_tmp:     ds 1

; linker memory static allocation directives for code
PSECT text, class=CODE, reloc=2, space=0

global _delay_10us

_delay_10us:
    ; TODO (;->)

    ;-> save BANK nb in bsr_tmp var (BSR context)
    ;-> select BANK0 in data memory
    ;-> store value in delay_cnt1 var

delay_10us_L1:
    ;-> decremente delay_cnt1, skip if zero
    ;-> branch to delay_10us_L1

    ;-> restore BSR context
    return
END
```

- Programmer l'algorithme du TP



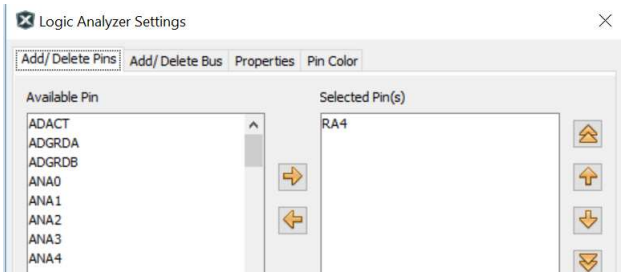

```
led2_on();
delay_10us();
led2_off();
delay_10us();
```

- Configurer le simulateur pour une fréquence de cycle machine = 64/4 = 16 MHz

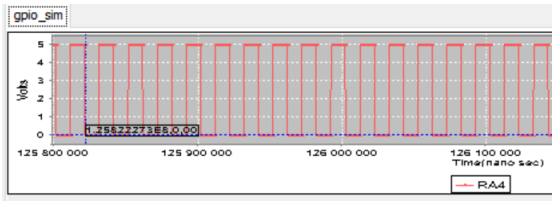


- **Debug Project** et faire **run** pour lancer l'exécution

- **Window > Simulator > Logic analyzer** : pour ouvrir l'analyseur logique. Aller dans **Settings** et sélectionner la broche RA4 (LED2)



- Faire plusieurs fois **Continue / Pause** jusqu'à voir la courbe se tracer



- Faire un zoom et mesurer la durée de la temporisation avec les curseurs