

102685 Arquitectura de Computadors i Perifèrics

PRÀCTICA N° 1- CURS 2020/2021

L'ENTORN CODEWARRIOR

1 Objectius de la pràctica

- Familiarització amb l'entorn CodeWarrior.
- Debugging del codi al microcontrolador i anàlisi de l'execució.
- Programació en Assembler.

2 Placa FRDM-KL25Z

La placa sobre la qual desenvoluparem les pràctiques serà la FRDM-KL25Z (Fig. 1) de Freescale. Aquesta placa conté un microcontrolador interconnectat a un conjunt de perifèrics. Treballarem amb aquesta placa en aquesta i les següents sessions. A continuació es llisten les especificacions:

32-bit ARM Cortex-M0+ core
Fins a 48 MHz de clock
Protocol OpenSDA per a debugging

Memoria

128 KB flash
16 KB SRAM

Perifèrics

- Capacitive touch slider
- LED
- Acceleròmetre MMA8451Q
- Botó reset

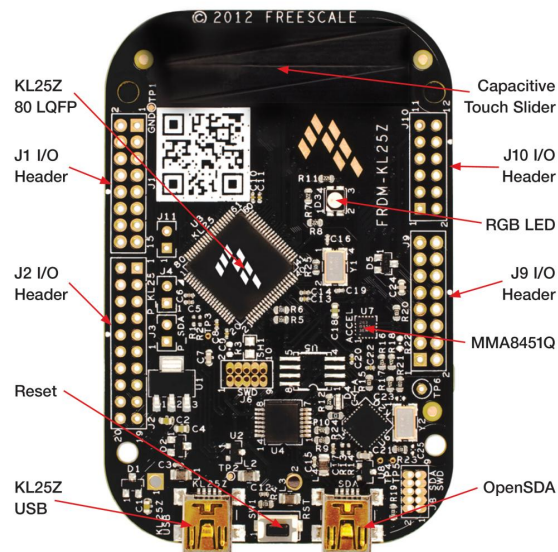


Figura 1. Descripció parts de la placa KL25Z

3 Entorn de desenvolupament

L'entorn de desenvolupament que emprarem durant les pràctiques serà la IDE CodeWarrior versió 10.5 (Fig. 2) desenvolupada per Freescale. Ens permetrà el desenvolupament, compilació i depuració de codi, així com anàlisi dels registres o memòria del microcontrolador. Aquesta IDE està basada en l'entorn Eclipse. Eclipse és usat extensament per a programar en llenguatges com Java, C, C++ o Python. CodeWarrior té diverses perspectives de treball, a les sessions de pràctiques treballarem amb la perspectiva C/C++ (perspectiva per defecte) i Debug, de la qual parlarem a la secció 3.3. En el cas de que l'alumne decideixi adquirir un microcontrolador FRDM-KL25Z, existeixen alternatives gratuïtes com la IDE KEIL o la IDE online de mbed.org.

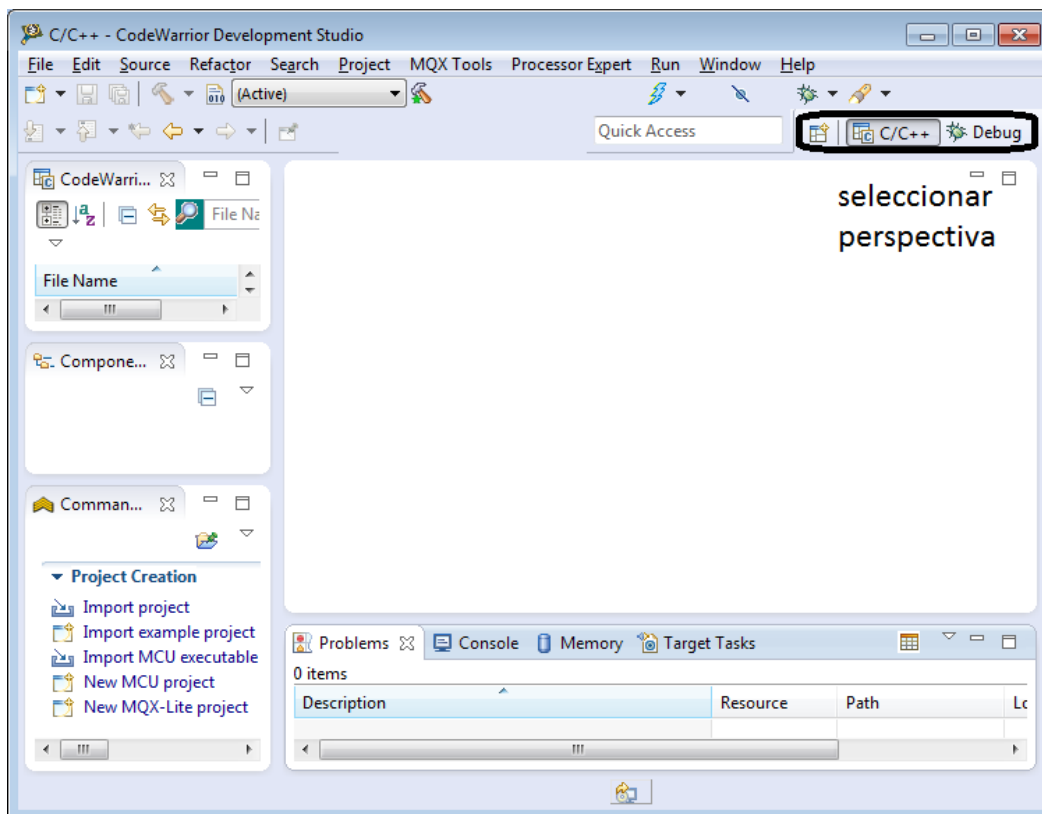


Figura 2. IDE CodeWarrior

3.1 Iniciant CodeWarrior

Trobarem l'enllaç del CodeWarrior 10.5 a l'escriptori dels laboratoris de pràctiques. Un cop obert, per a crear el projecte anem a *File -> New -> BareBoard Project*. El nom del projecte serà *practical1*. Al diàleg *Devices* escollim quin serà el dispositiu on s'executarà l'aplicació a desenvolupar i quina mena d'aplicació serà. En el nostre cas la família del processador és la *MKL25Z128* i el tipus de projecte és *Application* tal i com es mostra a la Fig. 3.

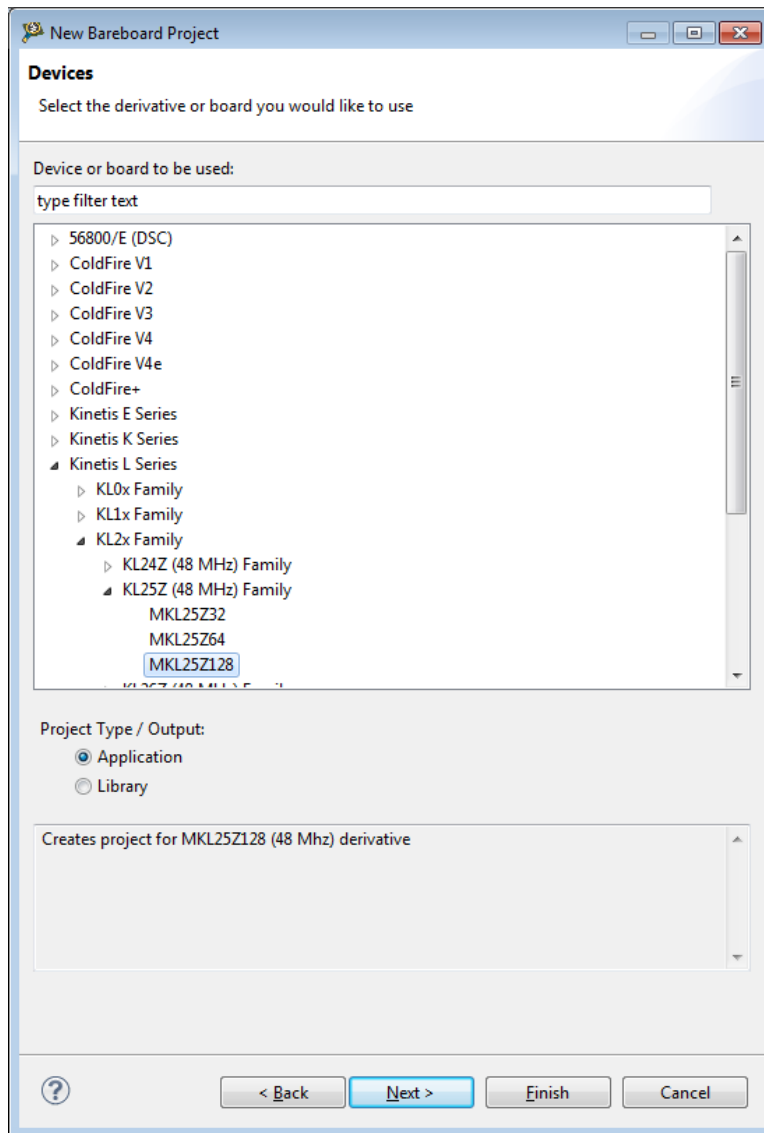


Figura 3. Selecció del dispositiu

La placa FRDM-KL25Z té un protocol per a poder descarregar el codi, i a més debugar desde l'ordinador el codi que s'executa al microcontrolador. Aquest protocol s'anomena OpenSDA. El marquem a l'apartat *Connections* i desmarquem qualsevol altre opció. Hem d'anar amb compte de no connectar el USB al port equivocat. Fixeu-vos a la Fig. 1 quin port és per a connexions USB i quin és per a depurar mitjançant el protocol OpenSDA. La pràctica la realitzarem en el llenguatge de programació assembler (ASM). Per tant escollim ASM com es mostra a la Fig. 4 i premem el botó *Finish*.

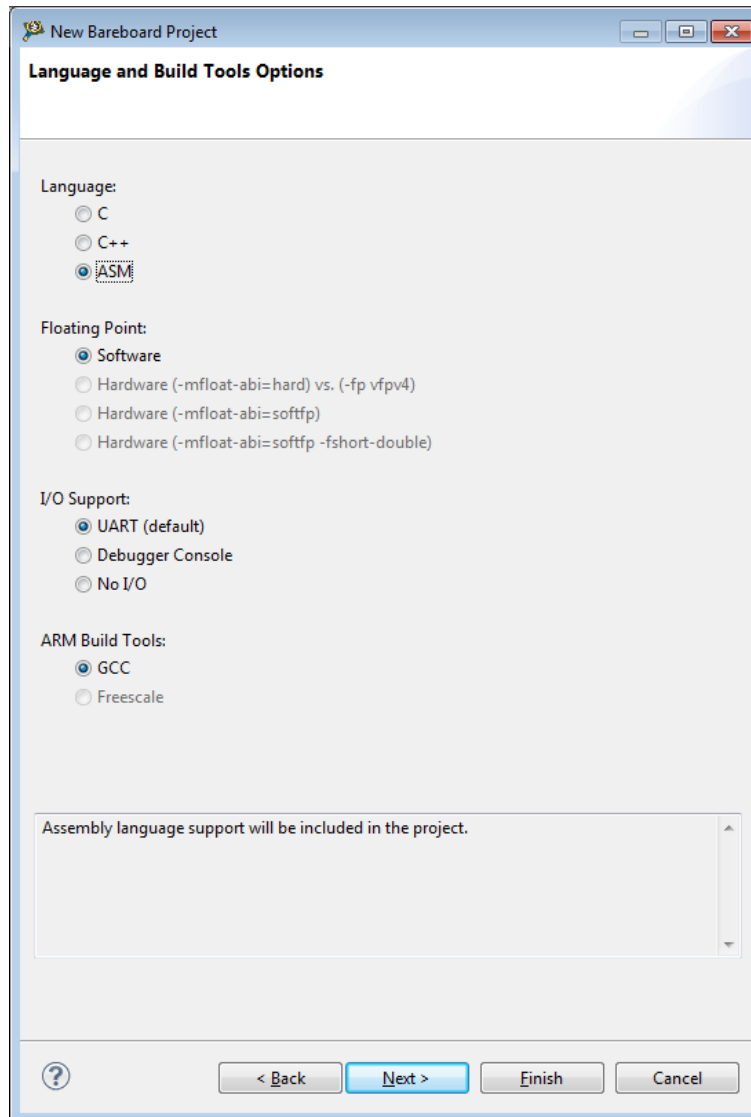


Figura 4. Selecció llenguatge

3.2 Compilar el projecte

Després de finalitzar la creació del projecte es crearà una estructura de carpetes i fitxers que podrem veure a la finestra de projectes. A la carpeta *Sources* trobarem el codi font. Utilitzarem el codi generat per defecte com a base de desenvolupament de la present sessió. El codi conté directives del assembler de GNU que comencen amb punt. La més important és *.global main* que indica quina és la funció principal del codi. De la resta com *.text*, *.type* o *.word* en podem prescindir. Per a compilar el projecte anem a *Project -> Build all* o bé premem Ctrl+B. Això compilarà tot el codi font d'exemple. El codi per defecte a l'arxiu *main.s* es mostrat a continuació:

```

/* This assembly file uses GNU syntax */
.text
.section .rodata
.align 2

.LC0:

.text
.align 2
.global main
.type main function

main:
    push {r3, lr}
    add r3, sp, #4
    mov r3, #0
    mov r0, r3
    pop {r3, pc}

    .align 2
.L3:
    .word .LC0
.end

```

3.3 Run & debug

Una vegada compilat el codi el descarregarem a la placa. Per això hem de connectar primer el USB físicament del PC al microcontrolador. Un cop connectat per a descarregar el codi a la placa s'ha executar *Run -> Run as -> CodeWarrior*. Aquesta opció l'emprarem en pràctiques següents però no en la present. En aquesta pràctica, donat que encara no interactuem amb els perifèrics i només ens comunicarem amb el processador, el que farem serà seguir l'evolució de l'execució.

El debugging (o depuració) consisteix en l'anàlisi de l'evolució del codi per a cada línia. El debugging és usat normalment per a analitzar errors, però en el nostre cas ens permetrà executar el codi fins a un punt concret, veure els valors de registres i memòria, analitzar el codi generat o veure la pila d'execució del programa. Per a depurar el codi anar a *Run->Debug*. S'obrirà la perspectiva *Debug* dividint l'interfície en més finestres, tal i com es veu a la Fig. 5.

- A la *finestra de debug* s'aniran imbricant les crides que es realitzin en programa.
- A la *finestra de codi* es mostrarà la implementació.
- A la *finestra de control* es poden consultar els valors de les variables del programa, breakpoints, registres i espai de memòria, principalment.
- A la *consola* s'hi mostraran els errors i els missatges del programa i el sistema.
- A la *finestra de Dissassembly* podem seguir l'execució del codi i veure el codi generat per a cada línia

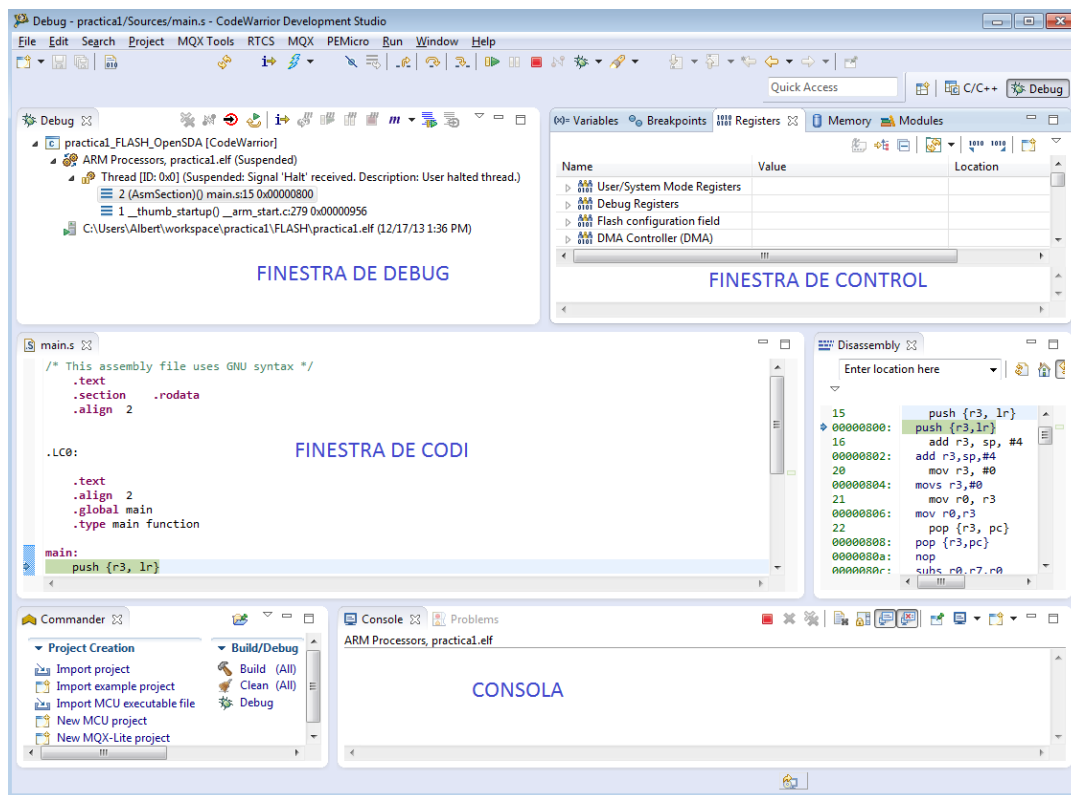


Figura 5. Perspectiva de Debug

3.3.1 Avançar i parar en la depuració del codi

Un cop CodeWarrior inicia la depuració del codi pressionarem F5 (*Step Into*) o el botó mostrat a la Fig. 6. Això permetrà evolucionar l'execució del codi on cada pas. Si posicionieu el ratolí a sobre de les altres icones de fletxes veurem que Eclipse mostra un text descriptiu. En la icona immediatament a l'esquerra del botó *Step Into*, la descripció ens diu *Step Over*. Aquesta acció ens permetrà executar fins que acabi una crida a una funció donada. A la següent icona descrita com *Step Return* ens permetrà anar fins al següent punt de retorn. El botó verd de *Play* farà l'acció *Resume* i executarà el codi indefinidament fins trobar un breakpoint. El botó vermell de *Stop* farà l'acció *Terminate* i parará l'execució.

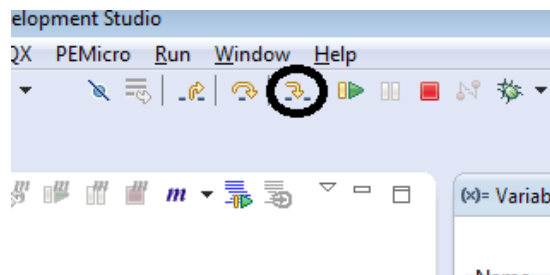


Figura 6. Botó "Step Into"

3.3.2 Breakpoints

Els breakpoints permeten forçar la depuració a parar en una línia de codi escollida. Per a introduir un breakpoint hem de fer doble click a la zona enmarcada a la Fig. 7, en la línia que vulguem. Després pressionem F8 o el botó *Resume* i el codi s'executarà fins a trobar un breakpoint. Els breakpoints són útils quan es vol comprobar el funcionament d'una zona particular del codi.

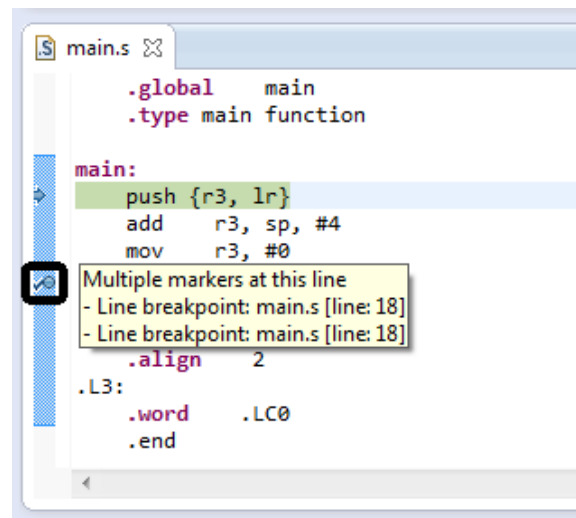


Figura 7. Breakpoint

3.3.3 SP, PC i registres

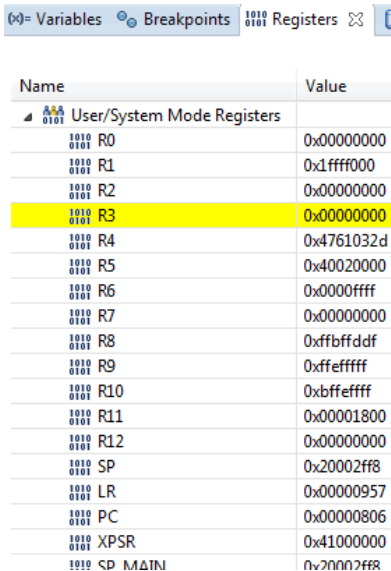
El processador *Cortex-M0+* té 12 registres de propòsit general enumerats de R0-R12 i que podem usar al programar. Altres registres específics com SP (*Stack Pointer*), PC (*Program Counter*) o xPSR (*Special-Purpose Program Status Register*) ens permet veure bits d'estat que són accessibles però que no emprarem per a guardar els valors de les variables dels nostres algoritmes. En la Fig. 8 es mostra el registre xPSR que ens permetrà veure els bits d'estat. Aquesta informació la podem consultar a la finestra de control.



Figura 8. Registre xPSR

N: Flag d'estat negatiu
Z: Flag d'estat zero
C: Bit de carry
V: Bit d'overflow

En la finestra de control despleguem la llista *User/System Module Registers*. Es mostraran tots els registres del processador. Veurem en groc els valors que han sigut modificats en l'últim pas de l'execució tal i com es veu a la Fig. 9. En aquesta finestra trobarem també informació dels ports i perifèrics.



Name	Value
1010 0101 User/System Mode Registers	
1010 0101 R0	0x00000000
1010 0101 R1	0x1ffff000
1010 0101 R2	0x00000000
1010 0101 R3	0x00000000
1010 0101 R4	0x4761032d
1010 0101 R5	0x40020000
1010 0101 R6	0x0000ffff
1010 0101 R7	0x00000000
1010 0101 R8	0xffbffdff
1010 0101 R9	0xffefffff
1010 0101 R10	0xbffeffff
1010 0101 R11	0x00001800
1010 0101 R12	0x00000000
1010 0101 SP	0x20002ff8
1010 0101 LR	0x00000957
1010 0101 PC	0x00000806
1010 0101 XPSR	0x41000000
1010 0101 CDP MΔINI	0x70007ff8

Figura 9. Registres

4 Exercici pràctic

Tradueix el següent codi de C a ASM. Guarda la variable *i* al registre r3 i la variable *c* al registre r7. Depura el codi i comproba l'evolució dels valors del registre a la finestra de control.

```
void main() {
    int i = 0;
    int c = 10;

    for(i=0; i < c; i++) {
        c--;
    }
}
```

Documentació

Al Campus Virtual podem trobar els següents arxius que seran útils per a la sessió:

- Document ASM ARM
- Datasheet ARM Cortex-M0+
- Datasheet KL25Z