INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Programming Embedded Systems with C/C++

INSA Lyon
3GE – IF2
3GEA – IF2a

Thomas Grenier

Prérequis : IF1 (programmation en C),
IF2 partie « architecture des microcontrôleurs »

Objectifs de formation

- Avoir les bases pour développer en C sur des plateformes embarquées:
  - *Syntaxe du langage C : IF1*
  - **Adaptations du code et de la structure d'un programme**
  - **Compilation *croisée***

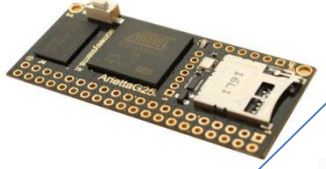
- Savoir mettre en œuvre des **IRQ/ISR en C** sur des systèmes embarqués sans OS

Systèmes embarqués ciblés

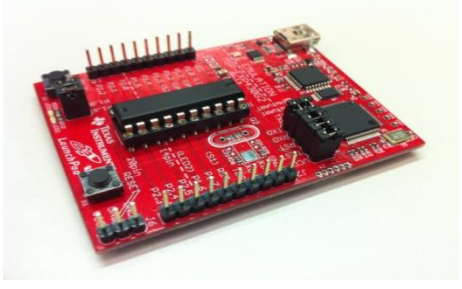
Raspberry Pi 3/4

Odroid XU4

Arietta G25

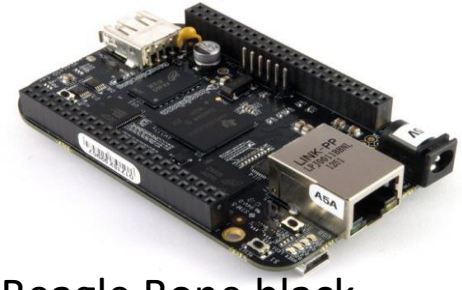


MSP430/432



pcDuino 3 Nano

PIC16 – PIC18



Beagle Bone black

Arduino UNO

STM32Fx

Bibliographie

Notecpp.pdf

- **Programming Embedded Systems** Second Edition, Barr & Massa, Ed. O'Reilly
- **Embedded Software Development: The Open-Source Approach**, Ivan Cibrario Bertolotti & Tingting Hu, CRC Press
- **Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications**, Robert Oshana, Ed. Newnes
- **Real-Time C++: Efficient Object-Oriented and Template Micro-Controller Programming**, Christopher Kormanyos, Springer

Plan

I. Introduction
Interruption, « compilation », vocabulaires

II. Compilation croisée pour cibles avec OS

III. Compilation croisée pour cibles sans OS

IV. C pour cibles embarquées sans OS

V. Interruption en C pour cibles embarquées sans OS

VI. Eléments de modélisation pour la programmation d'événements

Part I

INTRODUCTION

1- Interruptions
2- C/C++ pour les SE

Interruption, en exemple

- Premier exemple: je révise, et soudain…

  … le téléphone sonne ! Que fais-je ?

Je réponds ! … ah non ca dépend de qui appelle…

Je marque ma page pour me souvenir où j'en étais

Je note les éléments dont je dois me souvenir suite à l'appel

Je ne veux pas être dérangé

- Le rôle du développeur: définir des **évènements** « pertinents »* pour planifier des actions
  - Je fais cuire des œufs à la coque et je garde un enfant

Surveiller **la fin d'un compte à rebours**

Vérifier si une **bêtise est en vue**…

* : et des priorités (4GE)

Interruptions

- Objectif : 1- décharger l'UAL de la surveillance d'une valeur ou état (scrutation) → ex. bouton RA0 appuyé en PIC ?

```
….      ; instructions
while(PORTAbits.RA0 == 0);
….      ; suite du programme
```

- Objectif : 2- scruter plusieurs valeurs ou états en 'même temps' liés à des événements aléatoires et de temporalité possiblement différente

➔ **Systématiquement utilisées pour les applications µC/µP!**

➔ **Changement de paradigme de programmation**

Programmation séquentielle → programmation événementielle

Interruption, à retenir!

➔ **Systématiquement utilisées pour les applications µC/µP!**

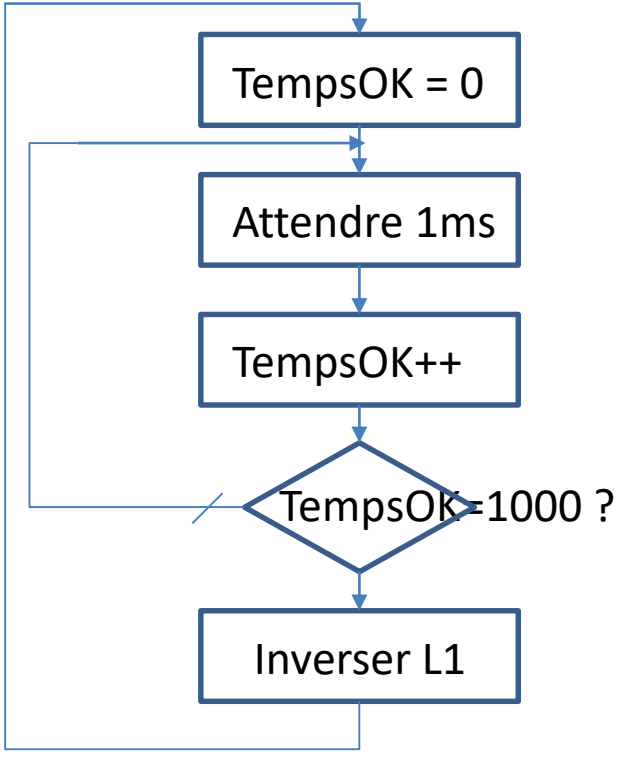
- Une interruption peut survenir n'importe quand.
- Si elle est autorisée, une interruption (IRQ) :
  - attend la fin de l'exécution de l'instruction en cours mais bloque le passage à l'instruction suivante
  - effectue ensuite une séries d'instructions afin d'exécuter une fonction spécifique à la source d'interruption → **ISR**
- S'il existe plusieurs sources d'interruption, il faudra définir des priorités: « hardware » et/ou logiciel (dans le cas d'un OS)

➔**Changement de paradigme de programmation**

Programmation séquentielle → programmation événementielle

Exemple de programmations

1- Toute les secondes, changer l'état de la sortie L1 (une LED)



2- A chaque front montant sur BP1, changer l'état de la sortie L2

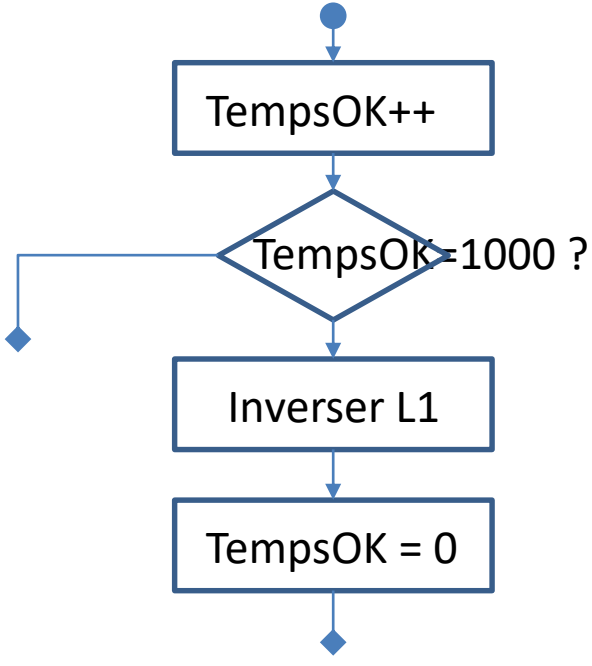
Exemple de programmations

1- Toute les secondes, changer l'état de la sortie L1 (une LED)
et à chaque front montant sur BP1, changer l'état de la sortie L2

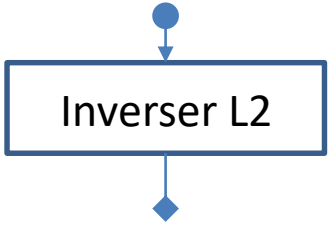
Approche événementielle

1- Toute les secondes, changer l'état de la sortie L1 (une LED)
et à chaque front montant sur BP1, changer l'état de la sortie L2

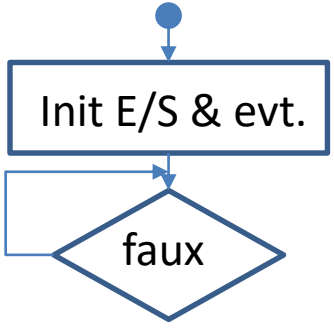
Evt. (1 ms)



Evt. (front sur BP1)



Programme principal



Pourquoi le C/C++ pour les Systèmes Embarqués?

- Alternative – haut niveau - à l'assembleur ☺

  il faut maitriser l'architecture µC/µP et la « *carte* » *(hardware)*

  et pas d'opérations par « bit » en C (bsf, btfsc…) ☹

  → *rappels en C : les bases et les opérations logiques*
- Syntaxe précise et gestion mémoire manuelle ☹
  - Java, python, Julia sont plus souples mais pas intégrables, ni optimum, sur de nombreux systèmes embarqués (OS!)
  - **Nouveaux mots clés** : volatile, extern, …
- Compilation, bibliothèques, liens… ☹
  - on dispose d'un grand nombres de bibliothèques ☺
  - quand ça (tombe en) marche, ça marche ! (périmètre technologique appréhendable)

Alternative : RUST

Bases et opérations binaires en C

Notecpp.pdf

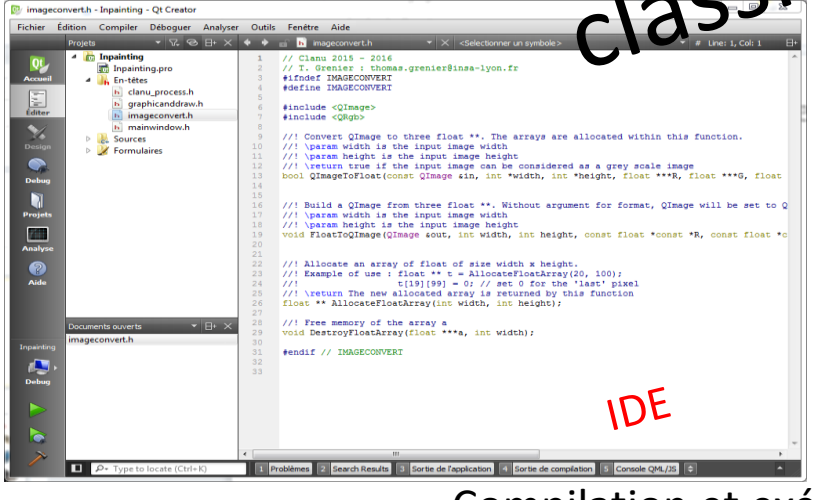
- Les bases C/C++, représentation de 15 (en décimal) en
  - Hexadécimal : **0x**0F
  - Binaire : **0b**00001111
  - Octal : **0**17 (15 en décimal !)
- Opérations sur les bits
  - &(et) |(ou) ^(ou exclusif) ~(inversion)
  - Décalages à droite et à gauche : >> << (ex. 4<<2 == 16)
- Exemple : masques en OU et ET
  - **OU**: mise à 1 du bit « 0 » sans toucher aux autres bits

    a |= 0x01; // <-> a = a | 0x01;
  - **ET**: mise à 0 du bit « 0 » sans toucher aux autres bits

    b &= ~0x01;

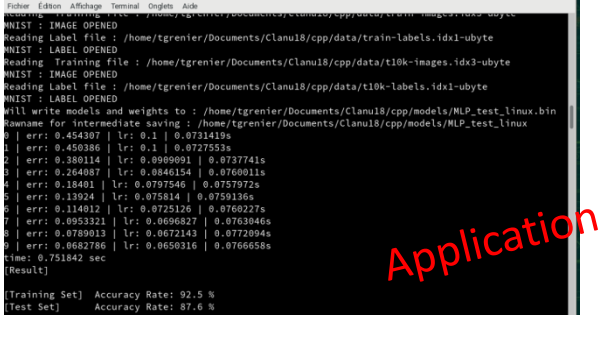
Comment développer en C/C++ ?

classique



Compilation et exécution
(débogage, …)

Ordinateur de développement
et d'exécution
« *General purpose computer* »
(dispose d'une chaine de compilation)



Application Console

Chaine de compilation

Une *Toolchain* est composée de :

- ***Compiler***: compilateur(s) (et son préprocesseur) pour traduire un langage en code exécutable pour un processeur donné
- ***Linker***: éditeur de liens, pour résoudre les dépendances
- ***Locator***: localisation des fonctions et variables

- ***Debugger*** : pour tester et mettre au point

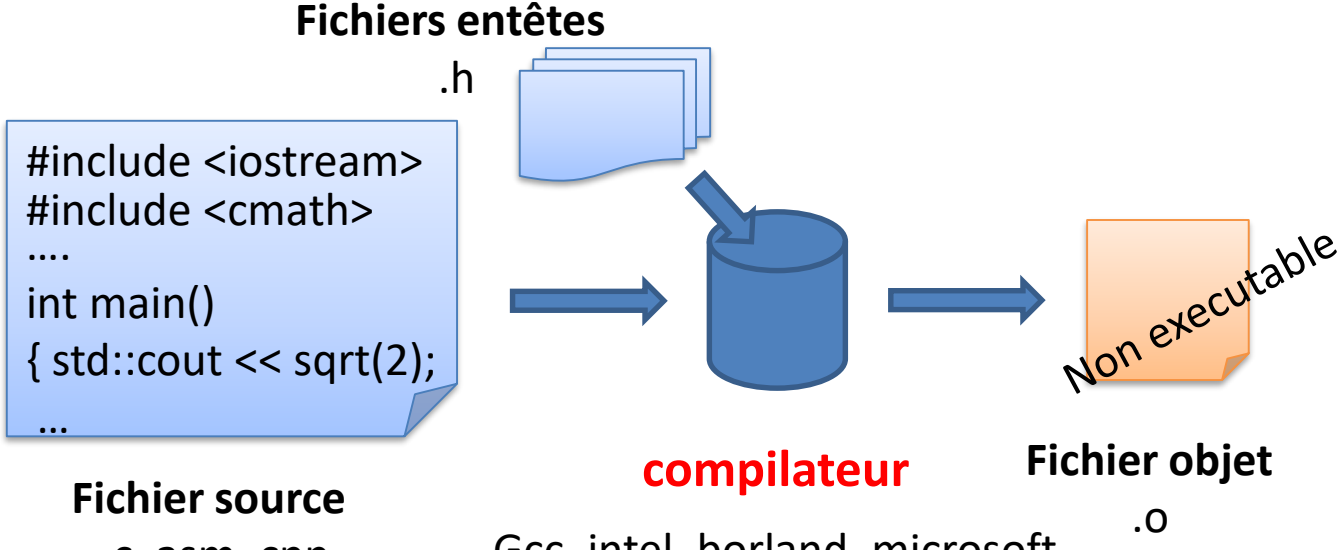
- ***Libraries*** : bibliothèques
  - pour interfacer l'environnement (stdlib, newlib): printf, malloc, …
  - Pour les fonctions usuelles (math.h, …)

Exemples de *toolchain* :

Gnu-gcc, intel, microsoft, borland, texas instrument, microchip, Clang/LLVM, IAR, …

« Compilation» ? rappels

- Compiler **n'est pas** « construire un fichier exécutable »
- Produire un fichier exécutable = compilation + édition des liens (*link*) + ….



- **Fichier source**: fichier texte contenant le code
- **Fichier objet**: fichier binaire contenant les instructions (code machine) et données provenant du processus de traduction du langage

  → Contient le code exécutable par un processeur mais fichier non exécutable

  →**Organisé en sections** (.text, .data, .bss, .debug, …)

  →Différents formats : **ELF**, COFF, **PE**, …

  Pour être exécutable, il manque:

  - Les liens vers les fonctions et variables externes (symboles)
  - Le positionnement des fonctions et variables dans les mémoires
  - Les étapes de lancement et de fin d'exécution

  →Pour obtenir un fichier exécutable, il manque :

  l'édition des liens et la localisation

Compilation, exemple simple

- … oui mais … en IF1 on a vu que:
  g++ main.cpp –o main → produit un exécutable !

- Compilation uniquement (–c)
  g++ -g -fverbose-asm simple.cpp **-c** –o simple.o
  g++ -g -fverbose-asm simple_main.cpp **-c** –o simple_main.o

- Lire les fichiers objets
  objdump -h simple.o → voir les entêtes de section
  objdump –S simple.o → voir le code machine (et assembleur)

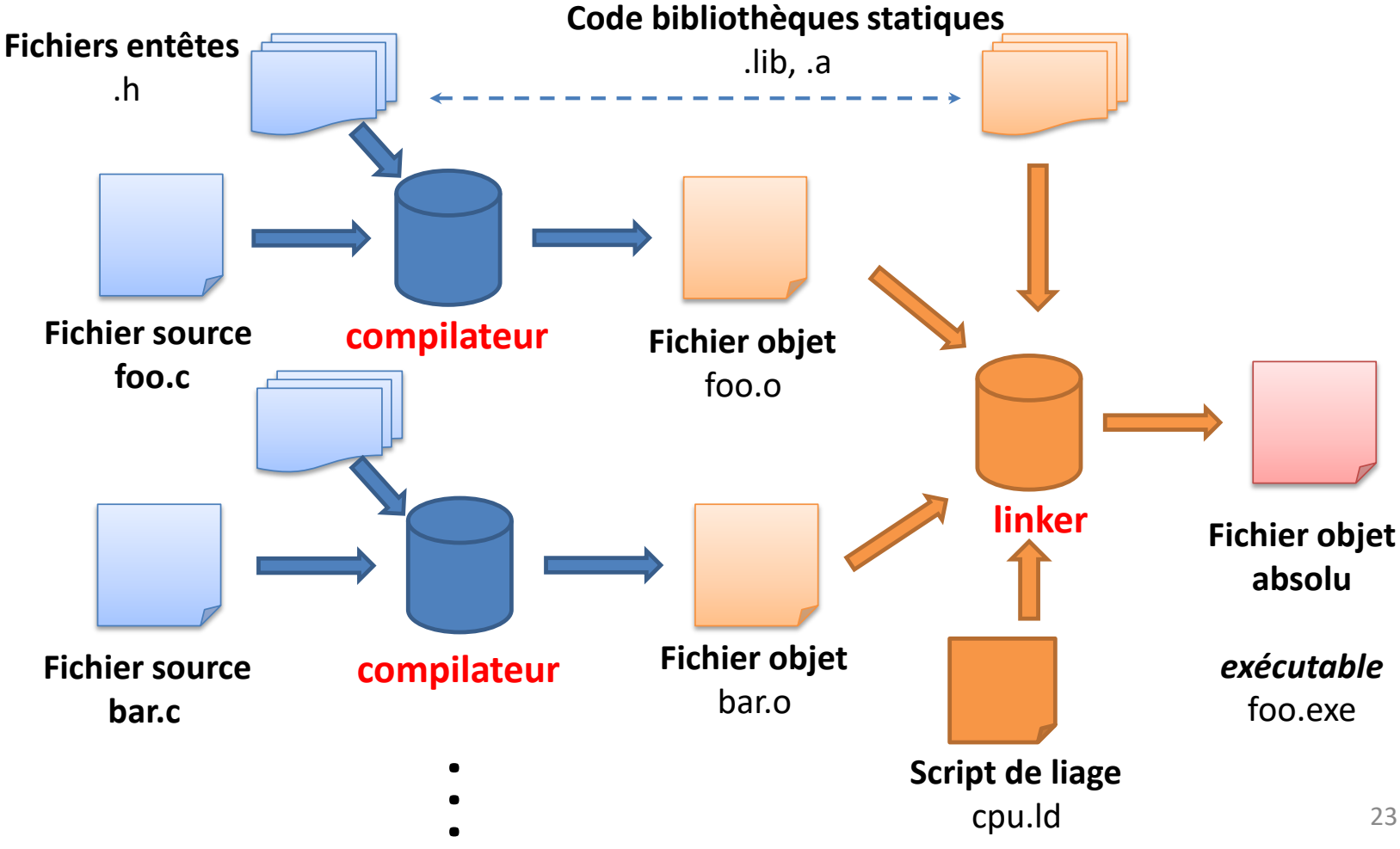
simple_main.cpp

```
int main(void)
{
 int i=10;
int j;
 for(j=0;j<10;j++)
     i=j+i;
return 0;
}
```

Objdump de simple_main.o

```
simple_main.o:     format de fichier pe-x86-64

Déassemblage de la section .text :

0000000000000000 <main>:

int main(void)
{
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 30             sub    $0x30,%rsp
   8:   e8 00 00 00 00          callq  d <main+0xd>
int i=10;
   d:   c7 45 fc 0a 00 00 00    movl   $0xa,-0x4(%rbp)
int j;
for(j=0;j<10;j++)
  14:   c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
  1b:   83 7d f8 09             cmpl   $0x9,-0x8(%rbp)
  1f:   7f 0c                   jg     2d <main+0x2d>
        i=j+i;
  21:   8b 45 f8                mov    -0x8(%rbp),%eax
  24:   01 45 fc                add    %eax,-0x4(%rbp)
for(j=0;j<10;j++)
  27:   83 45 f8 01             addl   $0x1,-0x8(%rbp)
  2b:   eb ee                   jmp    1b <main+0x1b>

return 0;
  2d:   b8 00 00 00 00          mov    $0x0,%eax
}
  32:   48 83 c4 30             add    $0x30,%rsp
  36:   5d                      pop    %rbp
  37:   c3                      retq
```

Et pour obtenir un executable ?



Et pour compiler pour les SE ?

→ *Hôte et Cibles*

Host…

Ordinateur de développement
~~et d'exécution~~
dispose d'une chaine de compilation
adaptée à l'architecture de la cible

programme

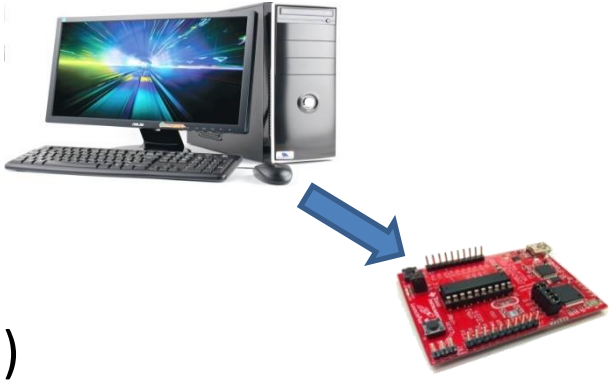
… Targets

hôte ≠ cible : compilation croisée

- Différences au niveau des processeurs

  Exemple : hôte x86_64 → cible PIC16
- Différences au niveau ressources (RAM, …)
- Différences au niveau des OS

  Exemples
  - Hôte Windows → cible linux
  - Hôte Windows → Android (ou iOS)
  - Hôte Linux A → cible Linux B
  - Hôte Linux → cible sans OS (baremetal)



Outils de développement adaptés à la compilation croisée

- Toolchain

  Gnu, IAR, intel, … , OpenEmbedded, buildroot

- IDE
  - Eclipse, Qtcreator, Visual Studio, IAR …
  - CodeComposerStudio (TI), MPLabX (Microchip), STM32CubeIDE (ST) …
  - Arduino desktop IDE, Energia (TI), …

Part II

COMPILATION CROISÉE POUR CIBLE AVEC OS

Systèmes d'exploitation (OS)

- PC : Windows, GNU/Linux, Mac OS, …
- Smartphone : Android, iOS, BlackBerry OS, Windows Phone, …
- Systèmes embarqués : QNX, uClinux, RTOS, RTAI, … (linux/windows/android)

Objectif d'un OS: interfacer et diriger l'utilisation de la ressource *informatique* exploitée par des applications

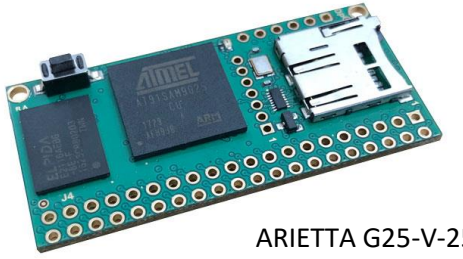
(mémoires, système de fichiers, ordonnanceurs, pilotes, …)

→ À savoir : tous les OS se basent sur un noyau (*kernel*)

Principes pour le développement

- Obtenir la toolchain du processeur cible
  - Attention aux librairies et à la configuration de la cible (architecture et carte):
- Paramétrer son système pour utiliser cette toolchain
  - IDE et parfois l'OS
- Construire l'exécutable
- Transférer l'exécutable sur la cible
- Au besoin déboguer (à distance ou JTAG)

Exemple



ARIETTA G25-V-256
~30€

- Arietta g25 (ACME SYSTEMS)
  - Arm9 , 400MHz, 256Mo DDR2
  - Toolchain utilisée: arm gnu gcc
  - Linux dédié … (adaptation/compilation/installation)
- Toolchain gnu: A-B-C-D
  - A : architecture de la cible
  - B : vendeur (parfois absent)
  - C : OS de la cible (none => pas d'OS… partie III)
  - D : Interface binaire de l'application (ABI)

  →Exemple : toolchain arm-linux-gnueabi

  →Compilateur c++ : arm-linux-gnueabi-g++

```
$ arm-linux-gnueabi-g++   first.c –o first
```

Avec OS, Interruptions: 2 types

- Au niveau application (IF4 – 5GE)
  - Interruptions logicielles qui passent par le noyau (signaux et/ou handler).
  - Plusieurs normes (dépend de OS, modèles, …).
    - Pour Unix et C : POSIX
- Au niveau matériel
  - En lien avec le noyau (interrupt handler)
    - Exemple linux: #include <linux/interrupt.h>
    - cat /proc/interrupts

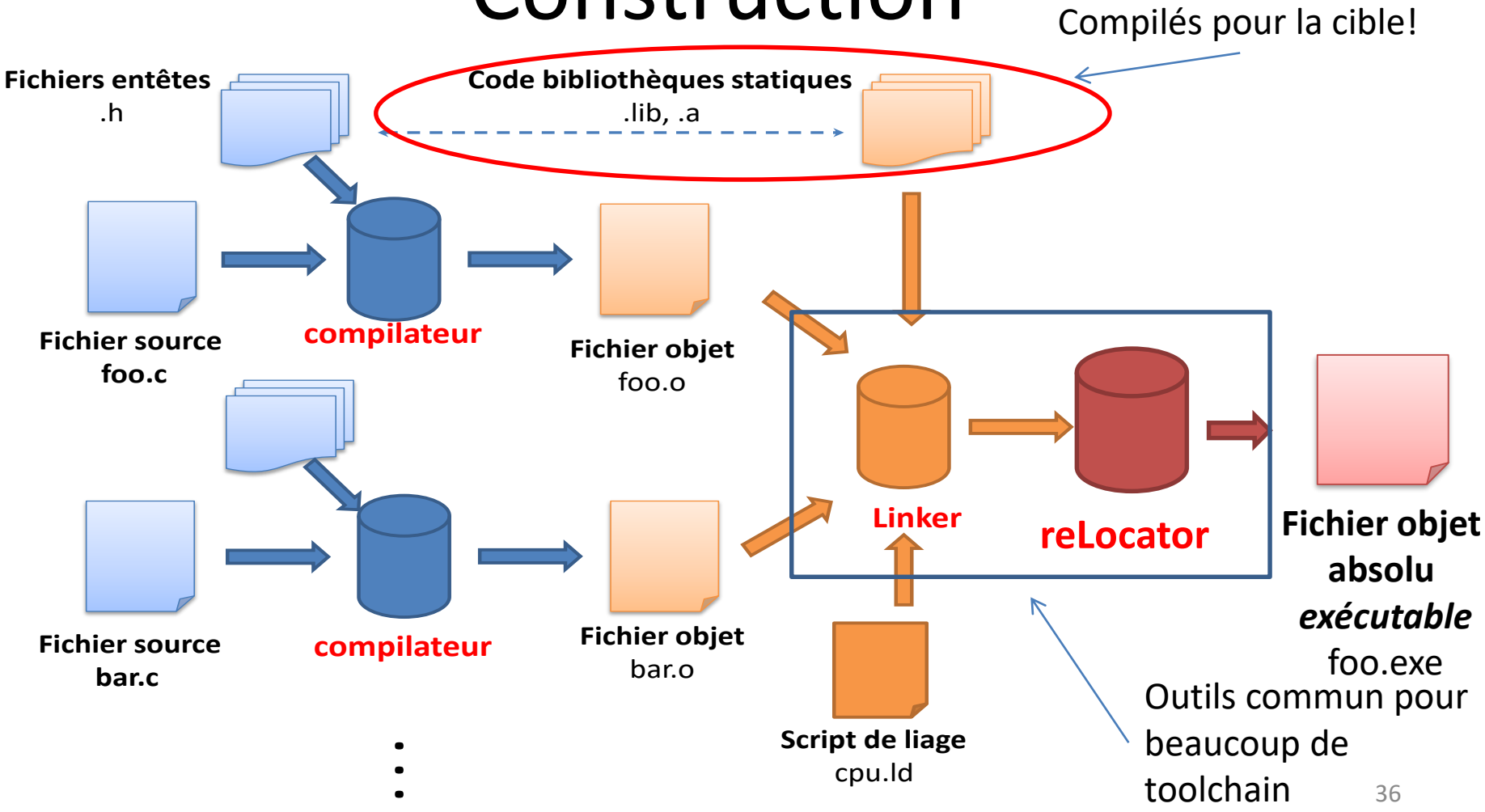
Part III

COMPILATION CROISÉE POUR CIBLE SANS OS

Principes pour le développement

1. Obtenir la *toolchain* pour le processeur cible
2. **Adapter les bibliothèques (newlib, …)**
3. **Adapter les scripts du linker à la plateforme cible (relocator)**
4. Paramétrer son système (écrire un makefile)
5. Construire l'exécutable pour la cible
6. Envoyer l'exécutable sur la cible
7. *Au besoin* déboguer (sonde de debug ou JTAG)

Construction



Hôte

Cible

Edition des sources

IDE, …

Codes sources

Construction

Toolchain
bibliothèques

exécutable

Transfert

Outils IDE
Ou ligne de commande

Visualisation capture

Protocol console

Débuguer

Interface console

JTAG, …

Interface debug

Processeur ou microcontrôleur

Protocol débug

Protocol transfert

Mémoire interne ou externe

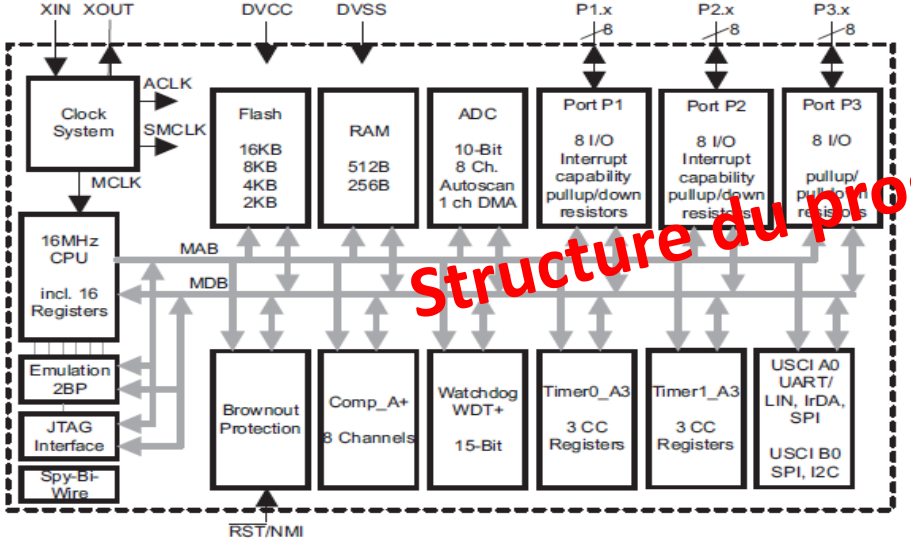
exécutable

Exemple

- ST Nucleo : STM32F0 (32 bits) (arm)
- **TI launchpad MSP430 G2553 (16 bits)**
  - 16MHz, 512o RAM, 16ko (16350o) ROM
  - 2,79 € / unit – 1290€ / 1000units (Farnell)
  - (kit launchpad msp430g2553 : ~12€)
  - *524 msp430 différents*
- Toolchain / IDE:
  - IAR, Keil, microship (XC8, XC16, …), **Gnu gcc (MSP430 : msp430-gcc)**
  - Eclipse, QtCreator, Visual Studio Code…
  - **CodeComposerStudio**, STMCubeIDE, MPLABX

Mon 1ier programme : Hello world…

- Ecrire un programme pour un MSP430G2553 qui :
  - allume la led verte (P1.6) et éteigne la led rouge (P1.0) quand on appuie sur le bouton poussoir (P1.3)
  - Puis inversement dès qu'on relâche le bouton poussoir



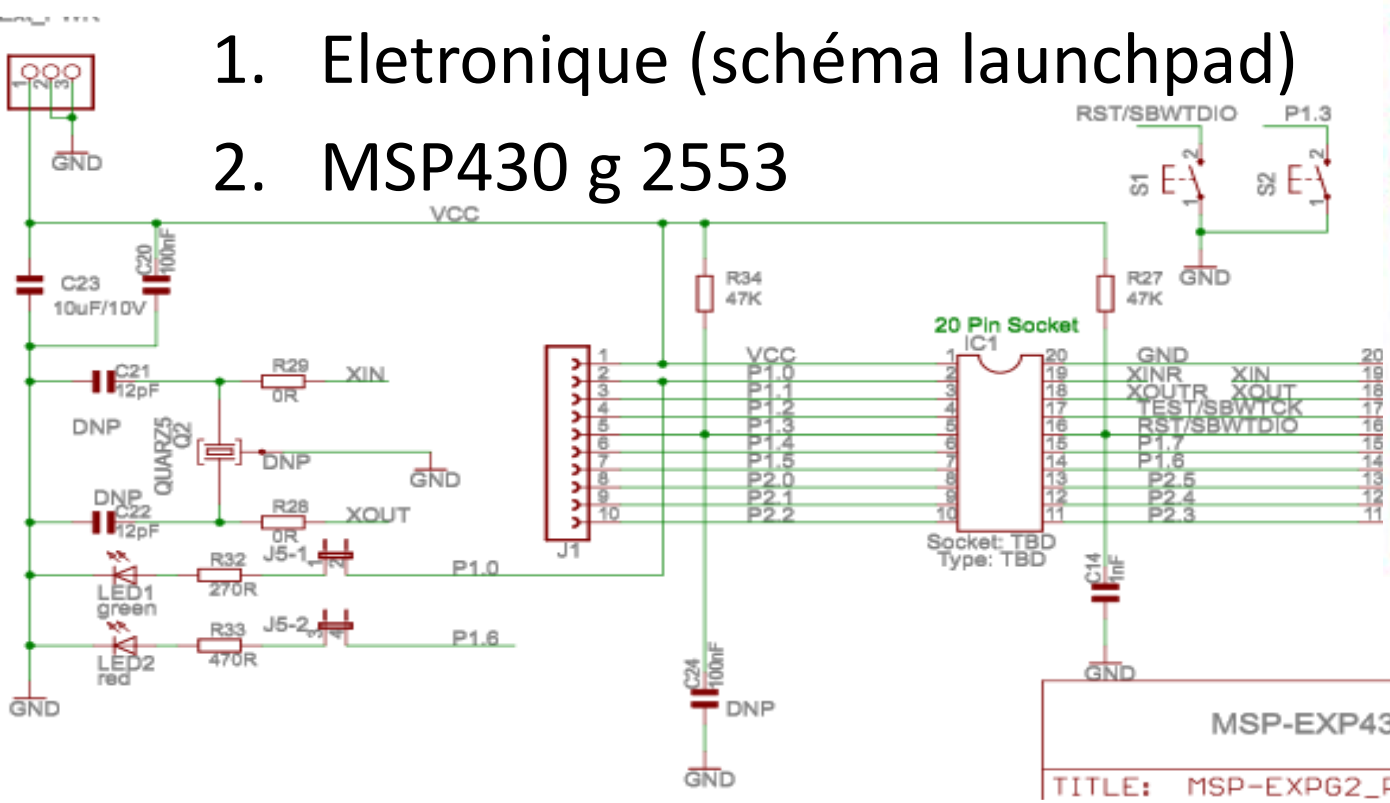
msp430g2553

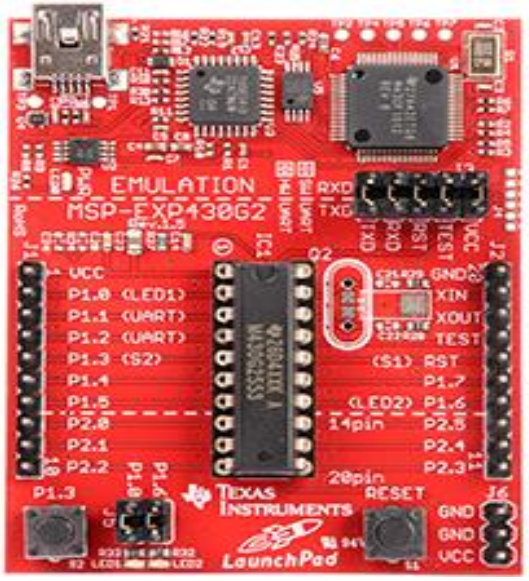
Structure du Programme C ?

Code c? … pas tout de suite!

D’abord lire la doc technique:

1. Eletronique (schéma launchpad)
2. MSP430 g 2553





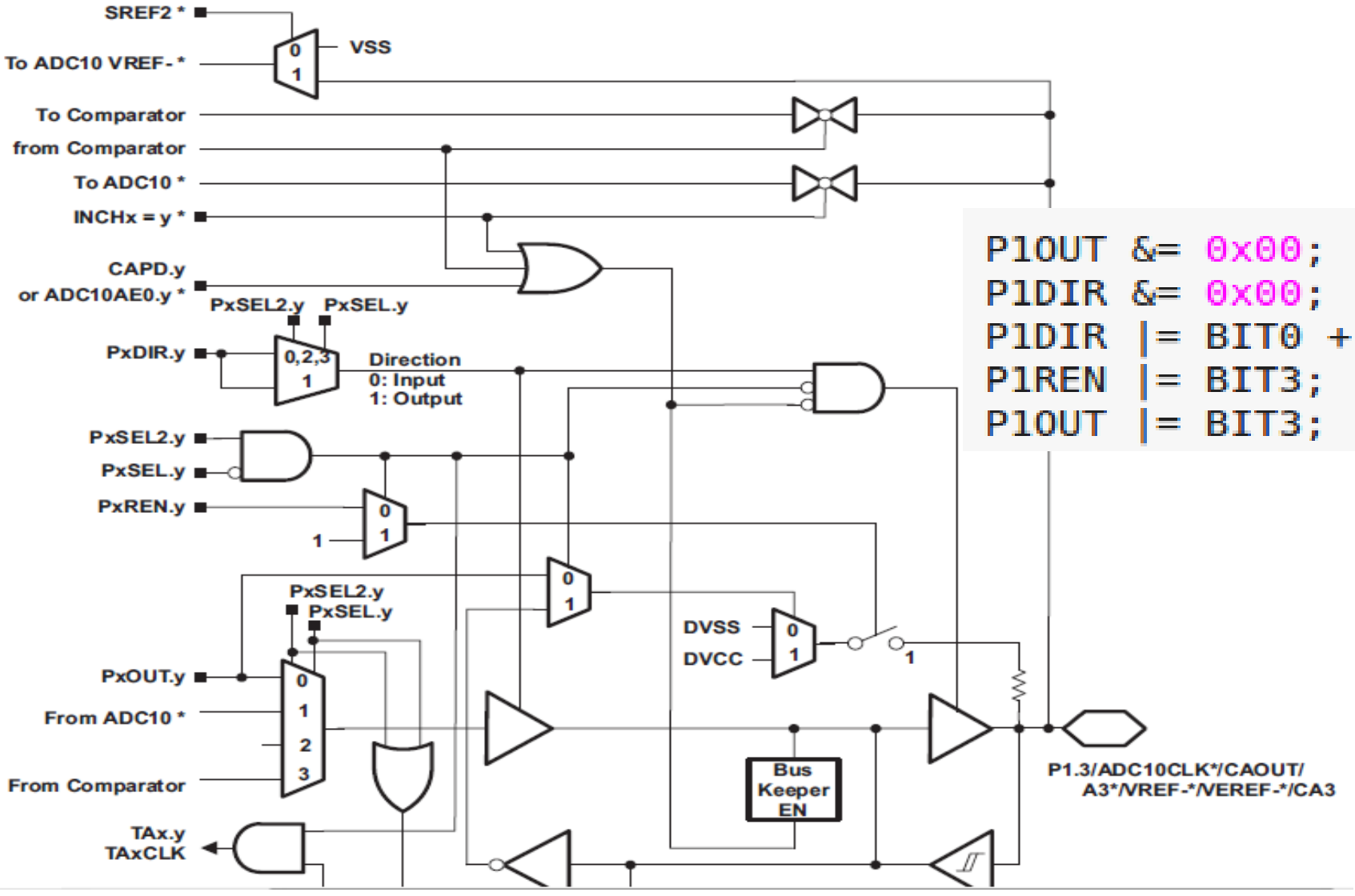
MSP430 : PORT 1

REGISTER DESCRIPTION	REGISTER NAME	OFFSET
Port P1 selection 2	P1SEL2	041h
Port P1 resistor enable	P1REN	027h
Port P1 selection	P1SEL	026h
Port P1 interrupt enable	P1IE	025h
Port P1 interrupt edge select	P1IES	024h
Port P1 interrupt flag	P1IFG	023h
Port P1 direction	P1DIR	022h
Port P1 output	P1OUT	021h
Port P1 input	P1IN	020h

1 → output
0 → Input

- LED rouge en P1.0 et verte en P1.6
- BP en P1.3

➔Configuration pour piloter les LEDs et lire la valeur du bouton?




```
P1OUT &= 0x00;
P1DIR &= 0x00;
P1DIR |= BIT0 + BIT6;
P1REN |= BIT3;
P1OUT |= BIT3;
```

Compilations + édition liens + relocation = **Construction**

- Connaitre les noms des SFR et bits du msp ?

  → #include <msp430g2553.h>

- Avoir le compilateur pour le msp ?

  →msp430-elf-gcc

- Mais de nombreux msp430 existent! Il faut compiler pour le g2553:

  → Option « -mmcu=msp430g2553 » au compilateur

```
msp430-elf-gcc -I /chemin/msp430g2553.h
               -mmcu=msp430g2553 led_bp.c -o led_bp.out
```

Ecriture du programme sur la cible

- Système de ‘flash’ des microcontrôleurs :
  - ICD (in circuit debugger), JTAG; (**ICE** : emulator)
  - Programmateur de composant
  - Parfois intégré à la carte de développement ;)
- Parfois nécessaire de faire des conversions de formats (objcopy … rendu transparent dans les IDE)
- **Cas du launchpad : mspdebug rf2500**
  - Se charge de tout: conversion binaire, programmation du processeur (via usb), exécution et debug

Tests…

CCS 12

- Led_bp.c
  - -g -c , objdump –S led_bp.o
  - -g , objdump –S led_bp.o; size led_bp.o

Led_bp.c et Led_bp.out

```c
#include <msp430.h>

int main(void)  //Main program
{
   WDTCTL = WDTPW + WDTHOLD; // Stop

  P1OUT &= 0x00;              // Shut d
  P1DIR &= 0x00;
  P1DIR |= BIT0 + BIT6;     // P1.0 an
  P1REN |= BIT3;            // Enable
  P1OUT |= BIT3;            //Select p

   while(1) //Loop forever, we'll do
   {
    if( P1IN & BIT3 )
        {
        P1OUT |= BIT0;  // mise à 1
        P1OUT &= ~BIT6; // mise à 0
        }
    else
        {
        P1OUT &= ~BIT0; // mise à 0
        P1OUT |= BIT6;  // mise à 1
        }
   }
}
```

```
.....  ...
 6        WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
         main():
c13e:    40B2 5A80 0120      MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
 8       P1OUT &= 0x00;                // Shut down everything
c144:    425C 0021           MOV.B   &Port_1_2_P1OUT,R12
c148:    43C2 0021           CLR.B   &Port_1_2_P1OUT
 9       P1DIR &= 0x00;
c14c:    425C 0022           MOV.B   &Port_1_2_P1DIR,R12
c150:    43C2 0022           CLR.B   &Port_1_2_P1DIR
10       P1DIR |= BIT0 + BIT6;            // P1.0 and P1.6 pins outp
c154:    D0F2 0041 0022      BIS.B   #0x0041,&Port_1_2_P1DIR
11       P1REN |= BIT3;                   // Enable internal pull-up
c15a:    D2F2 0027           BIS.B   #8,&Port_1_2_P1REN
12       P1OUT |= BIT3;                   //Select pull-up mode for
c15e:    D2F2 0021           BIS.B   #8,&Port_1_2_P1OUT
16         if( P1IN & BIT3 )
c162:    B2F2 0020           BIT.B   #8,&Port_1_2_P1IN
c166:    2407                JEQ     (0xc176)
18             P1OUT |= BIT0; // mise à 1
c168:    D3D2 0021           BIS.B   #1,&Port_1_2_P1OUT
19             P1OUT &= ~BIT6; // mise à 0
c16c:    F0F2 FFBF 0021      AND.B   #0xffbf,&Port_1_2_P1OUT
c172:    4030 C162           BR      #0xc162
23             P1OUT &= ~BIT0; // mise à 0
c176:    C3D2 0021           BIC.B   #1,&Port_1_2_P1OUT
24             P1OUT |= BIT6; // mise à 1
c17a:    D0F2 0040 0021      BIS.B   #0x0040,&Port_1_2_P1OUT
c180:    4030 C162           BR      #0xc162
```

Back into systems

- relocator

Documentation technique TI

		MSP430G2553 MSP430G2513
Memory	Size	16kB
Main: interrupt vector	Flash	0xFFFF to 0xFFC0
Main: code memory	Flash	0xFFFF to 0xC000
Information memory	Size	256 Byte
	Flash	010FFh to 01000h
RAM	Size	512 Byte
		0x03FF to 0x0200
Peripherals	16-bit	01FFh to 0100h
	8-bit	0FFh to 010h
	8-bit SFR	0Fh to 00h

msp430g2553.ld

```
MEMORY {
    SFR                     : ORIGIN = 0x0000, LENGTH = 0x0010 /* END=0x0010, size 16 */
    RAM                     : ORIGIN = 0x0200, LENGTH = 0x0200 /* END=0x03FF, size 512 */
    INFOMEM                 : ORIGIN = 0x1000, LENGTH = 0x0100 /* END=0x10FF, size 256 as 4 64-
    INFOA                   : ORIGIN = 0x10C0, LENGTH = 0x0040 /* END=0x10FF, size 64 */
    INFOB                   : ORIGIN = 0x1080, LENGTH = 0x0040 /* END=0x10BF, size 64 */
    INFOC                   : ORIGIN = 0x1040, LENGTH = 0x0040 /* END=0x107F, size 64 */
    INFOD                   : ORIGIN = 0x1000, LENGTH = 0x0040 /* END=0x103F, size 64 */
    ROM (rx)                : ORIGIN = 0xC000, LENGTH = 0x3FDE /* END=0xFFDD, size 16350 */
    VECT1                   : ORIGIN = 0xFFE0, LENGTH = 0x0002
    VECT2                   : ORIGIN = 0xFFE2, LENGTH = 0x0002
    VECT3                   : ORIGIN = 0xFFE4, LENGTH = 0x0002
```

Part IV

C POUR LES MC/MP EMBARQUÉS SANS OS

Exemple 2 (scrutation, tempo)

1. Périodiquement changer l'état de P1.0

```
64
65 #include <msp430.h>
66
67 int main(void)
68 {
69    unsigned short i;
70
71   WDTCTL = WDTPW + WDTHOLD;
72   P1DIR |= 0x01;
73
74   for (;;)
75   {
76     P1OUT ^= 0x01;
77
78     i = 50000;
79     do (i--);
80     while (i != 0);
81   }
82 }
```

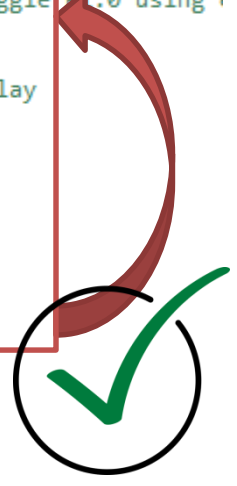
```
0000c010:   12B0 C02C           CALL    #abort
71          WDTCTL = WDTPW + WDTHOLD;                   // Stop watchdog timer
            main():
0000c014:   40B2 5A80 0120      MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
72          P1DIR |= 0x01;                              // Set P1.0 to output direction
0000c01a:   D3D2 0022           BIS.B   #1,&Port_1_2_P1DIR
76            P1OUT ^= 0x01;                            // Toggle P1.0 using exclusive-OR
            $C$L1:
0000c01e:   E3D2 0021           XOR.B   #1,&Port_1_2_P1OUT
79            do (i--);
0000c022:   3FFD                JMP     ($C$L1)
48                BIS.W     #(0x0010),SR
            $isr_trap.asm:48:59$(), __TI_ISR_TRAP():
0000c024:   D032 0010           BIS.W   #0x0010,SR
49                JMP __TI_ISR_TRAP
0000c028:   3FFD                JMP     ($isr_trap.asm:48:59$)
51                NOP                       ; CPU40 Compatibility NOP
0000c02a:   4303                NOP
100         {
            C$$EXIT(), abort():
0000c02c:   4303                NOP
108          for (;;);   /* SPINS FOREVER */
            $C$L1:
0000c02e:   3FFF                JMP     ($C$L1)
0000c030:   4303                NOP
58            return 1;
            _system_pre_init():
0000c032:   431C                MOV.W   #1,R12
```

Tempo ???

Exemple 2 (scrutation)

```
65 #include <msp430.h>
66
67 int main(void)
68 {
69   volatile unsigned short i;
70
71   WDTCTL = WDTPW + WDTHOLD;
72   P1DIR |= 0x01;
73
74   for (;;)
75   {
76     P1OUT ^= 0x01;
77
78     i = 50000;
79     do (i--);
80     while (i != 0);
81   }
82 }
```

```
68          {
            main():
0000c000:     8321                DECD.W  SP
71            WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
0000c002:     40B2 5A80 0120      MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
72            P1DIR |= 0x01;                            // Set P1.0 to output
0000c008:     D3D2 0022           BIS.B   #1,&Port_1_2_P1DIR
76              P1OUT ^= 0x01;                          // Toggle P1.0 using
            $C$L1:
0000c00c:     E3D2 0021           XOR.B   #1,&Port_1_2_P1OUT
78              i = 50000;                              // Delay
0000c010:     40B1 C350 0000      MOV.W   #0xc350,0x0000(SP)
79              do (i--);
            $C$L2:
0000c016:     8391 0000           DEC.W   0x0000(SP)
81            }
0000c01a:     9381 0000           TST.W   0x0000(SP)
0000c01e:     27F6                JEQ     ($C$L1)
0000c020:     3FFA                JMP     ($C$L2)
144         {
            _c_int00_noinit_noargs():
0000c022:     4031 0400           MOV.W   #0x0400,SP
111            if(_system_pre_init() != 0)
0000c026:     12B0 C044           CALL    #_system_pre_init
123               main(0);
0000c02a:     430C                CLR.W   R12
0000c02c:     12B0 C000           CALL    #main
125            exit(1);
0000c030:     431C                MOV.W   #1,R12
0000c032:     12B0 C03E           CALL    #abort
48                  BIS.W     #(0x0010),SR
```



Exemple 3 (scrutation) *à faire*

1. Changer l'état de la led (P1.0) sur un front du BP (P1.3)
2. Tous les 4 fronts, changer l'état de la led (P1.6)

- Led_bp_fr.c
- Led_bp_fr.c + math…

Deeper in C for Embedded Systems

- <stdint.h> : (C++11) uint8_t, int32_t, … à la place de unsigned char, int
- **static** : variable dont la durée de vie est celle du programme mais visible dans son scope de déclaration
- **extern** : idem mais déclarée dans un autre scope (et fonctionne pour des variables ou des fonctions)
- **volatile** : aucune optimisation du compilateur liée à l'utilisation de la variable
  - GPIO et SFR.
  - Exemple : volatile int a;
- **inline** : fonction dont le code sera copié au lieu d'être appelé

  optimisation en temps, mais pas en taille du programme

  Exemple : inline int add(int a, int b)

Part V

IRQ EN C POUR LES MC/MP EMBARQUÉS SANS OS

Vocabulaire interruption

- Interruption? C'est quoi?? Pourquoi faire?
- Source d'interruption: relation avec µC
  - **IRQ** : **i**nterrupt **r**e**q**uested, (*flags* activation)
  - Vecteur d'interruption
- Fonction de gestion de l'interruption : **ISR**
  - **i**nterrupt **s**ub**r**outine

Interruption, comment ca marche

- Tableau… exemple du PIC32MX

➔Comment transformer un programme pour utiliser des IRQ/ISR ?

1. **Changements de conception**
2. Echange de données avec les autres fonctions?

➔ Connaitre les mémoires et « flags » liés aux IRQ

➔ *Utilisation de mot clé spécifique du C*

Exemple 3 (**IRQ**)

1. Changer l'état de la led (P1.0) sur un front du BP (P1.3)
2. Tous les 4 fronts, changer l'état de la led (P1.6)

- Led_bp_fr_irq_v1.c (scrutation d'un sémaphore)
- Led_bp_fr_irq_v2.c (pilotage des IOs via ISR)

Led_bp_fr_irq_v1.c

```c
#include <msp430.h>
volatile char semaphore;

void __attribute__ ((interrupt(PORT1_VECTOR)))
      Interrupt_Port_1(void)
{

    volatile unsigned short i = 5000;
    do (i--);
    while (i != 0);

    semaphore = 1;
    P1IFG &= ~BIT3; // P1.3 interrupt flag cleared
}
```

```c
int main(void)  //Main program
{
   WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

   P1OUT &= 0x00;  // Shut down everything
   P1DIR &= 0x00;
   P1DIR |= BIT0; // Set P1.0 to output and P1.3 to input direction
   P1DIR |= BIT6; // Set P1.6 to output and P1.3 to input direction

   P1OUT &= ~BIT0; // set P1.0 to Off
   P1OUT &= ~BIT6; // set P1.6 to Off

   P1REN |= BIT3;  // Enable internal pull-up/down resistors
   P1OUT |= BIT3;  //Select pull-up mode for P1.3

   P1IE |= BIT3; // P1.3 interrupt enabled
   P1IFG &= ~BIT3; // P1.3 interrupt flag cleared

   __bis_SR_register(GIE); // Enable all interrupts

   char count = 0;
   semaphore  = 0;

   while(1) //Loop forever, we'll do our job in the interrupt routine...
   {
       while( semaphore != 1 ); // poll sempahore
       P1OUT ^= BIT0;  // Toggle P1.0
       semaphore = 0;
       count ++;
       if(  count  == 4 )
         {
         P1OUT ^= BIT6; // set P1.6 to Off
         count = 0;
         }
    }
   return 0;
}
```

Led_bp_fr_irq_v2.c

```
void __attribute__ ((interrupt(PORT1_VECTOR)))
      Interrupt_Port_1(void)
{
    static unsigned char count = 0;

    volatile unsigned short i = 5000;
    do (i--);
    while (i != 0);

    P1OUT ^= BIT0;  // Toggle P1.0
    count ++;
    if( count == 4 )
        {
        P1OUT ^= BIT6; // set P1.6 to Off
        count  = 0;
        }

    P1IFG &= ~BIT3; // P1.3 interrupt flag cleared
}
```

```
int main(void)  //Main program
{
   WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

   P1OUT &= 0x00; // Shut down everything
   P1DIR &= 0x00;
   P1DIR |= BIT0; // Set P1.0 to output and P1.3 to input direction
   P1DIR |= BIT6; // Set P1.6 to output and P1.3 to input direction

   P1OUT &= ~BIT0; // set P1.0 to Off
   P1OUT &= ~BIT6; // set P1.6 to Off

   P1REN |= BIT3; // Enable internal pull-up/down resistors
   P1OUT |= BIT3; //Select pull-up mode for P1.3

   P1IE |= BIT3; // P1.3 interrupt enabled
   P1IFG &= ~BIT3; // P1.3 interrupt flag cleared

   __bis_SR_register(GIE); // Enable all interrupts

   while(1) //Loop forever, we'll do our job in the interrupt routine...
   {
   }
   return 0;
```

Back into systems

- **relocator**

Documentation technique

		MSP430G2553 MSP430G2513
Memory	Size	16kB
Main: interrupt vector	Flash	0xFFFF to 0xFFC0
Main: code memory	Flash	0xFFFF to 0xC000
Information memory	Size	256 Byte
	Flash	010FFh to 01000h
RAM	Size	512 Byte
		0x03FF to 0x0200
Peripherals	16-bit	01FFh to 0100h
	8-bit	0FFh to 010h
	8-bit SFR	0Fh to 00h

msp430g2553.ld

```
MEMORY {
    SFR                 : ORIGIN = 0x0000, LENGTH = 0x0010 /* END=0x0010, size 16 */
    RAM                 : ORIGIN = 0x0200, LENGTH = 0x0200 /* END=0x03FF, size 512 */
    INFOMEM             : ORIGIN = 0x1000, LENGTH = 0x0100 /* END=0x10FF, size 256 as 4 64-
    INFOA               : ORIGIN = 0x10C0, LENGTH = 0x0040 /* END=0x10FF, size 64 */
    INFOB               : ORIGIN = 0x1080, LENGTH = 0x0040 /* END=0x10BF, size 64 */
    INFOC               : ORIGIN = 0x1040, LENGTH = 0x0040 /* END=0x107F, size 64 */
    INFOD               : ORIGIN = 0x1000, LENGTH = 0x0040 /* END=0x103F, size 64 */
    ROM (rx)            : ORIGIN = 0xC000, LENGTH = 0x3FDE /* END=0xFFDD, size 16350 */
    VECT1               : ORIGIN = 0xFFE0, LENGTH = 0x0002
    VECT2               : ORIGIN = 0xFFE2, LENGTH = 0x0002
    VECT3               : ORIGIN = 0xFFE4, LENGTH = 0x0002
```

Fonction d'interruption en C

- **void __attribute__ ((interrupt(***VECTOR_IRQ***)))**
  ***My_Isr_for_vector*(void)**
- #pragma vector=*VECTOR_IRQ*
  __interrupt void *My_Isr_for_vector*(void)
- Exemples *Led_bp_fr* précédents :
  - *VECTOR_IRQ* → PORT1_VECTOR
  - *My_Isr_for_vector* → Port_1
- Objdump –dS blink.out

Les étapes d'une ISR

prologue

- *1) (désactiver les nouvelles IRQ?)*
- 2) sauvegarder le contexte (registres, priorité, …)
- 3) vérifier la source d'IRQ (si plusieurs sources sur un même vecteur)
- 4) traiter l'interruption (code spécifique)
- 5) valider le traitement de l'interruption (réarmer le flag de la source…)

épilogue

- 6) restaurer le contexte
- 7) retour de fonction d'interruption

Les (3) étapes d'une ISR **à écrire en C**

prologue

- *1) (désactiver les nouvelles IRQ?)*
- 2) sauvegarder le contexte (registres, priorité, …)
- **3) vérifier la source d'IRQ (si plusieurs sources sur un même vecteur)**
- **4) traiter l'interruption (code spécifique)**
- **5) valider le traitement de l'interruption (réarmer le flag de la source…)**

épilogue

- 6) restaurer le contexte
- 7) retour de fonction d'interruption

Différences entre fonction et ISR

c

```c
char count;
volatile char tmp;

    void Fonction_Port_1(void)
    {
        P1OUT ^= BIT0;  // Toggle P1.0
        count ++;
        tmp = count * 3;
        P1IFG &= ~BIT3; // P1.3 flag cleared
    }
```

```c
void __attribute__ ((interrupt(PORT1_VECTOR)))
      Interrupt_Port_1(void)
{
    P1OUT ^= BIT0;  // Toggle P1.0
    count ++;
    tmp = count * 3;
    P1IFG &= ~BIT3; // P1.3 interrupt flag cleared
}
```

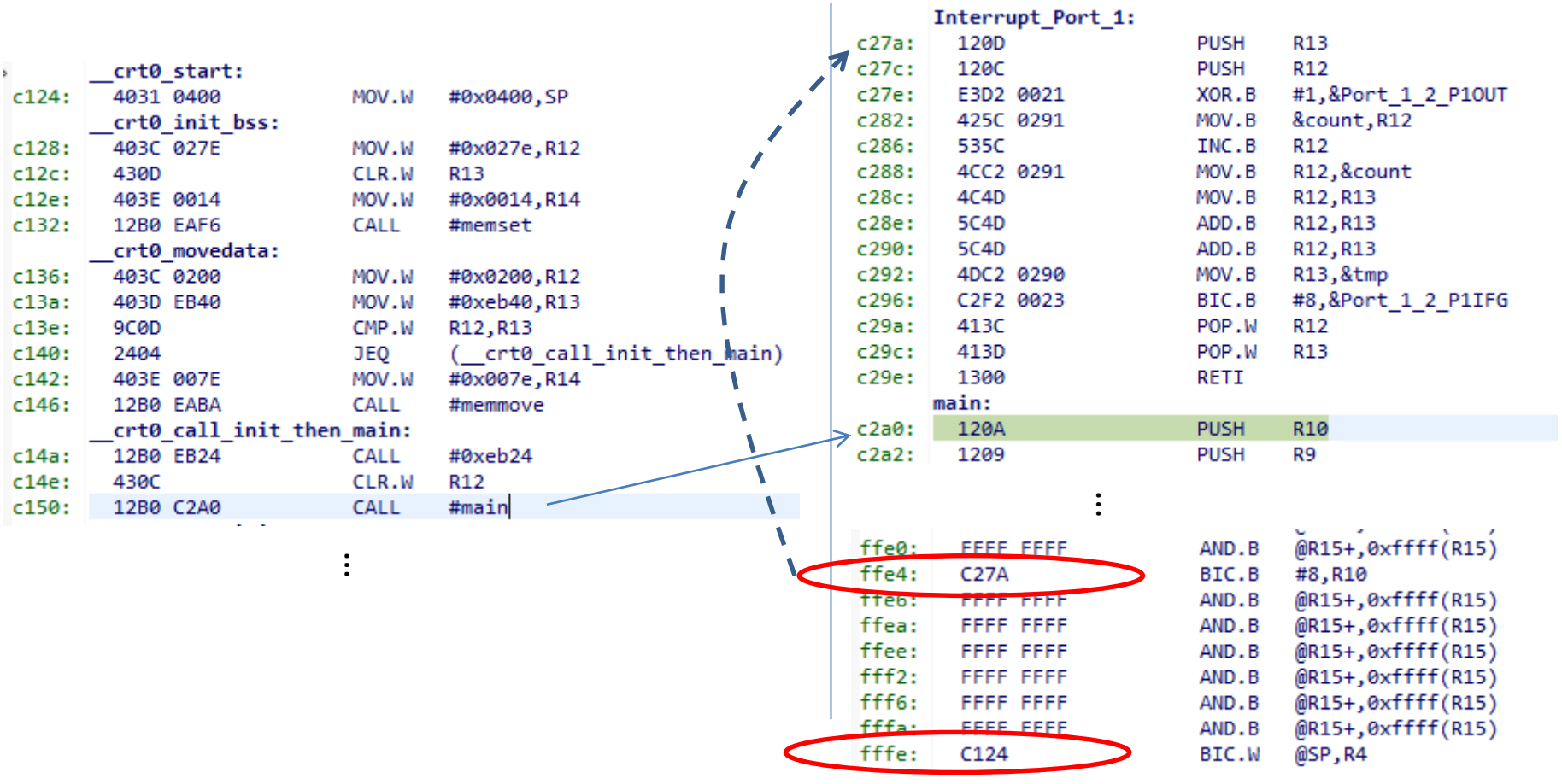
asm

```
    Fonction_Port_1:
c256:   E3D2 0021           XOR.B   #1,&Port_1_2_P1OUT
c25a:   425C 0291           MOV.B   &count,R12
c25e:   535C                INC.B   R12
c260:   4CC2 0291           MOV.B   R12,&count
c264:   4C4D                MOV.B   R12,R13
c266:   5C4D                ADD.B   R12,R13
c268:   5C4D                ADD.B   R12,R13
c26a:   4DC2 0290           MOV.B   R13,&tmp
c26e:   C2F2 0023           BIC.B   #8,&Port_1_2_P1IFG
c272:   4130                RET
```

```
    Interrupt_Port_1:
c274:   120D                PUSH    R13
c276:   120C                PUSH    R12
c278:   E3D2 0021           XOR.B   #1,&Port_1_2_P1OUT
c27c:   425C 0291           MOV.B   &count,R12
c280:   535C                INC.B   R12
c282:   4CC2 0291           MOV.B   R12,&count
c286:   4C4D                MOV.B   R12,R13
c288:   5C4D                ADD.B   R12,R13
c28a:   5C4D                ADD.B   R12,R13
c28c:   4DC2 0290           MOV.B   R13,&tmp
c290:   C2F2 0023           BIC.B   #8,&Port_1_2_P1IFG
c294:   413C                POP.W   R12
c296:   413D                POP.W   R13
c298:   1300                RETI
```

Différences entre fonction et ISR

Code assembleur et réallocation



msp430G2553.ld

```
MEMORY {
  SFR             : ORIGIN = 0x0000, LENGTH = 0x0010
  RAM             : ORIGIN = 0x0200, LENGTH = 0x0200
  INFOMEM         : ORIGIN = 0x1000, LENGTH = 0x0100
  INFOA           : ORIGIN = 0x10C0, LENGTH = 0x0040
  INFOB           : ORIGIN = 0x1080, LENGTH = 0x0040
  INFOC           : ORIGIN = 0x1040, LENGTH = 0x0040
  INFOD           : ORIGIN = 0x1000, LENGTH = 0x0040
  ROM (rx)        : ORIGIN = 0xC000, LENGTH = 0x3FDE
  BSLSIGNATURE    : ORIGIN = 0xFFDE, LENGTH = 0x0002
  VECT1           : ORIGIN = 0xFFE0, LENGTH = 0x0002
  VECT2           : ORIGIN = 0xFFE2, LENGTH = 0x0002
  VECT3           : ORIGIN = 0xFFE4, LENGTH = 0x0002
  VECT4           : ORIGIN = 0xFFE6, LENGTH = 0x0002
  VECT5           : ORIGIN = 0xFFE8, LENGTH = 0x0002
  VECT6           : ORIGIN = 0xFFEA, LENGTH = 0x0002
  VECT7           : ORIGIN = 0xFFEC, LENGTH = 0x0002
  VECT8           : ORIGIN = 0xFFEE, LENGTH = 0x0002
  VECT9           : ORIGIN = 0xFFF0, LENGTH = 0x0002
  VECT10          : ORIGIN = 0xFFF2, LENGTH = 0x0002
  VECT11          : ORIGIN = 0xFFF4, LENGTH = 0x0002
  VECT12          : ORIGIN = 0xFFF6, LENGTH = 0x0002
  VECT13          : ORIGIN = 0xFFF8, LENGTH = 0x0002
  VECT14          : ORIGIN = 0xFFFA, LENGTH = 0x0002
  VECT15          : ORIGIN = 0xFFFC, LENGTH = 0x0002
  RESETVEC        : ORIGIN = 0xFFFE, LENGTH = 0x0002
}
```

msp430G2553.h

(offset from 0xFFE0)

```
#define TRAPINT_VECTOR          ( 1)    /* 0xFFE0 TRAPINT */
#define PORT1_VECTOR            ( 3)    /* 0xFFE4 Port 1 */
#define PORT2_VECTOR            ( 4)    /* 0xFFE6 Port 2 */
#define ADC10_VECTOR            ( 6)    /* 0xFFEA ADC10 */
#define USCIAB0TX_VECTOR        ( 7)    /* 0xFFEC USCI A0/B0 Transmit */
#define USCIAB0RX_VECTOR        ( 8)    /* 0xFFEE USCI A0/B0 Receive */
#define TIMER0_A1_VECTOR        ( 9)    /* 0xFFF0 Timer0)A CC1, TA0 */
#define TIMER0_A0_VECTOR        (10)    /* 0xFFF2 Timer0_A CC0 */
#define WDT_VECTOR              (11)    /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR      (12)    /* 0xFFF6 Comparator A */
#define TIMER1_A1_VECTOR        (13)    /* 0xFFF8 Timer1_A CC1-4, TA1 */
#define TIMER1_A0_VECTOR        (14)    /* 0xFFFA Timer1_A CC0 */
#define NMI_VECTOR              (15)    /* 0xFFFC Non-maskable */
#define RESET_VECTOR            ("reset")/* 0xFFFE Reset [Highest Priority] */
```

Part VI

ELÉMENTS DE MODÉLISATION POUR LA PROGRAMMATION D'EVENEMENTS

Modélisation simple

➔Il est difficile de changer sa manière de construire un programme de la forme classique à la forme « évènements »

- Une approche progressive :

  1- S'intéresser aux transitions des grafcet ou des machines d'états → identifier les évènements

  2- Programmer chaque évènement avec un organigramme

  3- Utiliser des sémaphores sous forme de variables d'état (machine d'état) pour avoir la notion de séquence lors du traitement des évènements

Exemple du métronome

- Réaliser un programme de métronome visuel
  - Un front sur BP1 augmente le tempo et le diminue pour BP2.
  - La LEDV donne le tempo. La LED restera allumée 50ms.
  - La LEDR donne le premier temps (mesure de 4 temps). La LED restera allumée 50ms.

Modélisation réaliste

➔Il est difficile de déterminer l'ensemble des évènements et qu'il faut exécuter plusieurs taches en en concurrence

➔Approche ordonnancement et de couches (OS…):

1- Décomposer en plusieurs couches les contraintes. La plus basse étant le matériel, la plus haute les échanges software

2- Programmer la gestion de chaque couche (commencer par la plus basse) puis remonter en s'appuyant uniquement sur les fonctionnalités de la couche précédente [UML]

3- Utiliser un ordonnanceur optimisant une fonction de coût pour exécuter les différentes tâches. [UML]

4- s 'appuyer sur des messages, contrôle de flux

➔ IF3 et IF4