Programming Embedded Systems with C/C++

INSA Lyon 3GE – IF2

Thomas Grenier

Prérequis : IF1, IF2 architecture des microcontrôleurs

Objectifs de formation

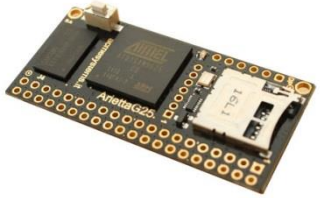
- Avoir les bases pour programmer en C des plateformes embarquées:
  - *Syntaxe du langage C : IF1*
  - **Adaptations/compilations**

- Savoir mettre en œuvre des IRQ/ISR en C sur des systèmes embarqués (à base de micro-controleur) sans OS

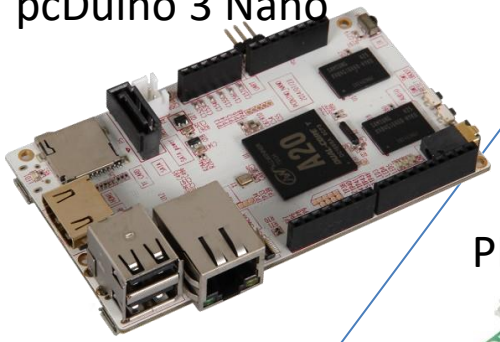
Systèmes embarqués ciblés

Raspberry Pi 3

Arietta G25

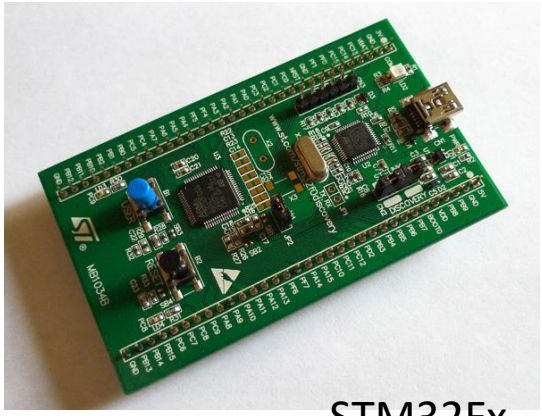


pcDuino 3 Nano



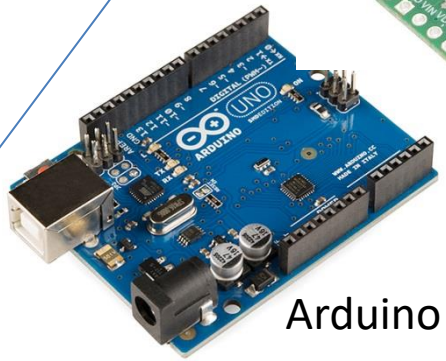
MSP430

PIC16 – PIC18



STM32Fx

Beagle Bone black



Arduino

Plan

I. Introduction

C vs Asm, « compilation », vocabulaires

II. Compilation croisée pour cibles avec OS

III. Compilation croisée pour cibles sans OS

Bibliographie

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

- Saines lectures : ~~magasine Open Silicium~~ *GNU/Linux Magazine*

Part I

INTRODUCTION

C/C++ pour les systèmes embarqués?

- Alternative – haut niveau - à l'assembleur ☺
  … mais il faut maitriser l'architecture quand même ☹
  … et pas d'opération par « bit » en C ☹ → *rappels*
- Syntaxe et rigueur ☹
  – Java, python, Julia sont plus simples !
  … Mais pas intégrables sur des petits systèmes ☹
- Compilation, bibliothèques, … ☹
  … mais on dispose d'un grand nombres de bibliothèques ☺

Systèmes d'exploitation (OS)

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

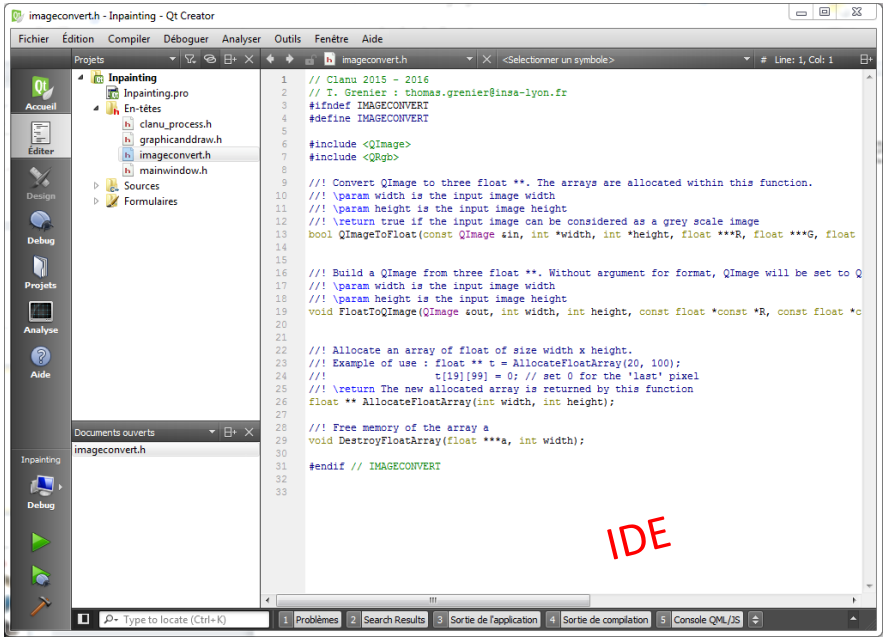
Objectif : interfacer et diriger l'utilisation de la ressource *informatique* exploitée par des applications

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

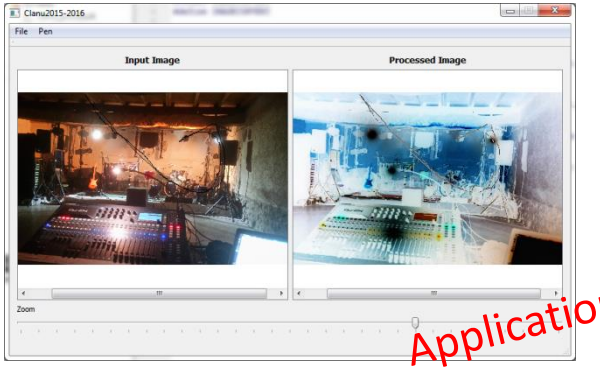
À savoir : les OS se basent sur un noyau (*kernel*)

Développement en C

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



Compilation et exécution
(débogage, …)



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

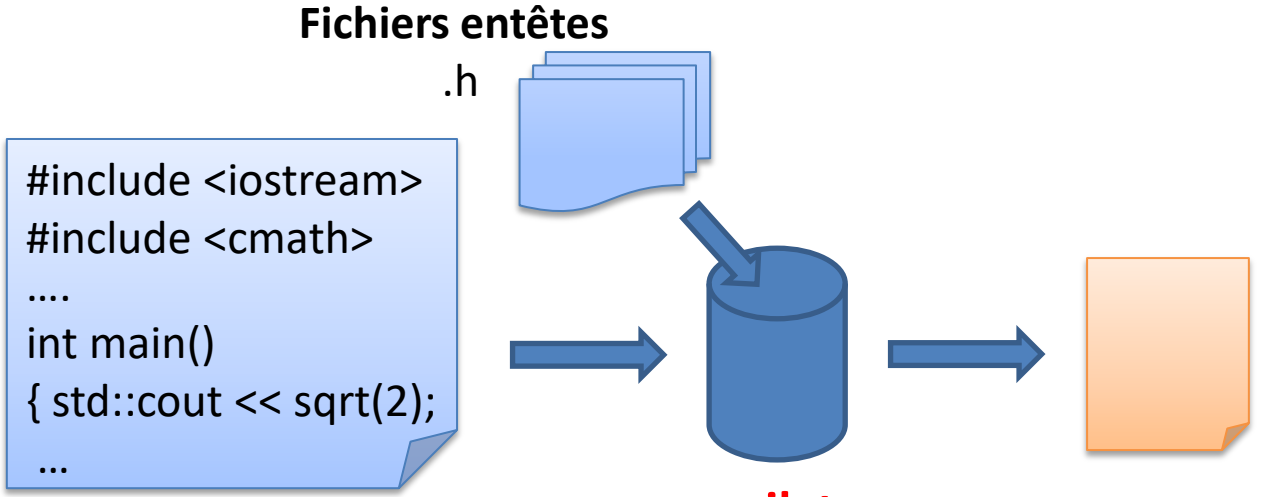
- ***Debugger*** : pour tester et mettre au point

- ***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, …

« 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)
  - L'organisation des fonctions et variables dans les mémoires
  - Phases de lancement et de fin d'exécution

  → Etape(s) d'édition des liens et de localisation

Exemple C/C++ simple

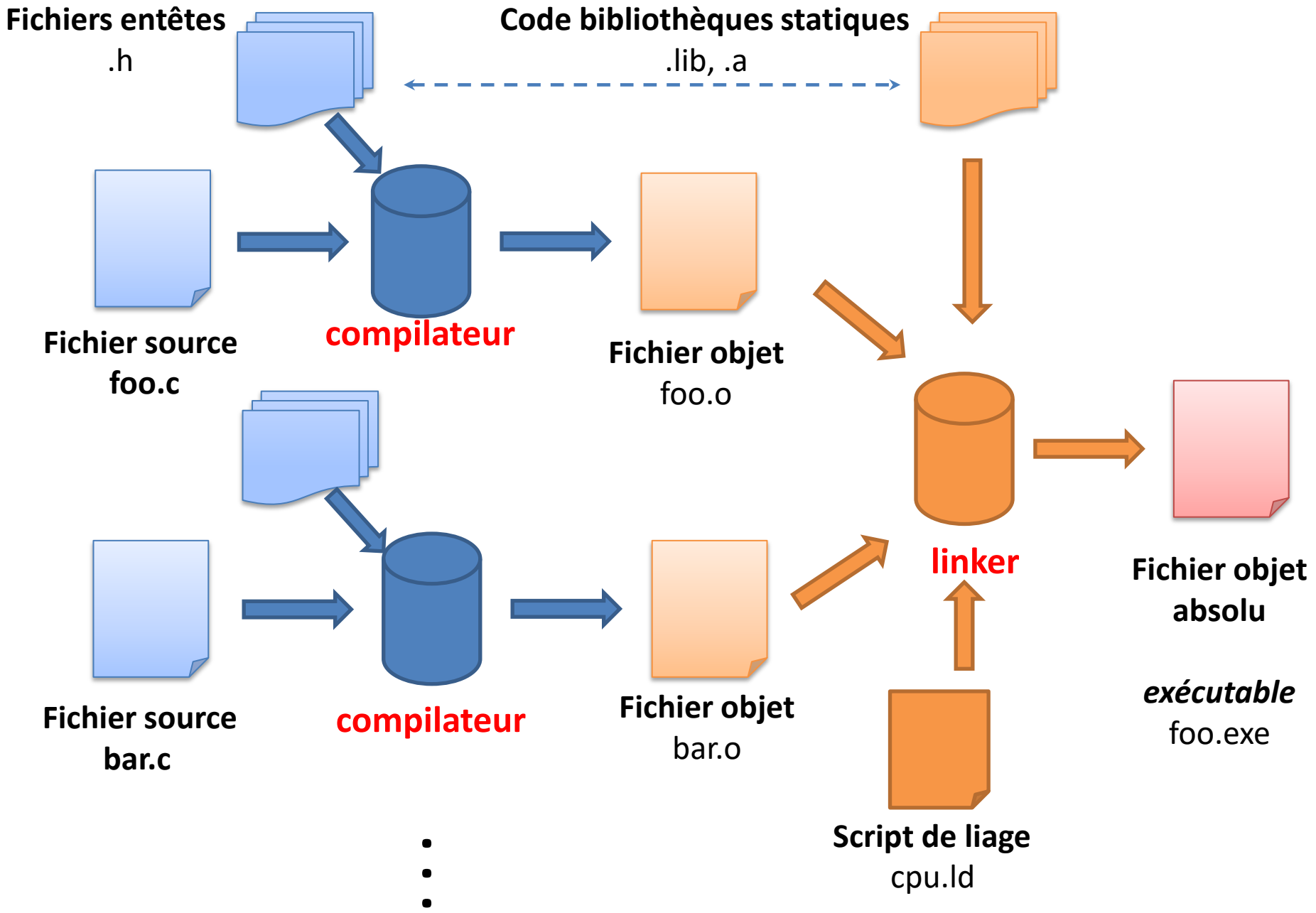
- Compilation

  g++ -g -fverbose-asm simple.cpp **-c** –o simple.o

- Lire les fichiers objets

  objdump.exe -h simple.o → voir les entêtes de section

  objdump –S simple.o → voir le code machine (et assembleur)


Fichiers entêtes
.h
Code bibliothèques statiques
.lib, .a
Fichier source
foo.c
compilateur
Fichier objet
foo.o
Fichier source
bar.c
compilateur
Fichier objet
bar.o
linker
Fichier objet
absolu
exécutable
foo.exe
Script de liage
cpu.ld

Hôte et Cibles

Host…

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

programme

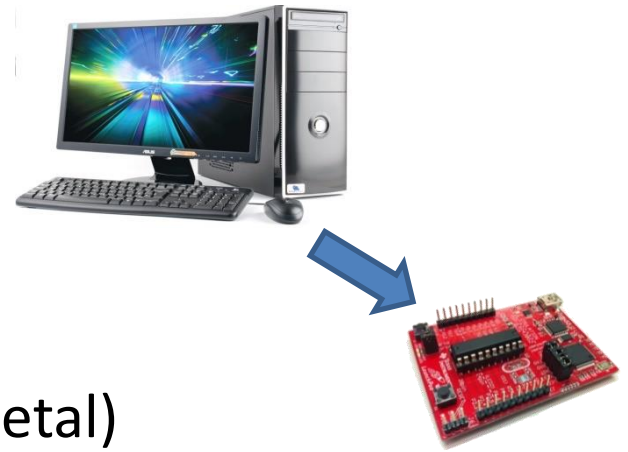
… Targets

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, OpenEmbedded, buildroot, IAR, intel, …

- IDE
  - Eclipse, Qtcreator, CodeComposerStudio, IAR …
  - MPLabX, Arduino desktop IDE, Energia, …

Part II

COMPILATION CROISÉE POUR CIBLE AVEC OS

Principe

- 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
  - 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
```

Interruptions, 2 types

- Au niveau application (IF4 – 5GE)
  - Interruptions logicielles qui passent par le noyau (signaux et/ou handler).
  - Plusieurs normes (OS, modèles, …). Pour Unix et C POSIX (IF4 5GE)
- 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

- Obtenir la toolchain pour le processeur
- **Adapter les bibliothèques (newlib)**
- **Adapter les scripts du linker à la plateforme cible (relocator)**
- Paramétrer son système (écrire un makefile)
- Construire l'exécutable
- L'envoyer sur la cible
- Au besoin déboguer (sonde de debug ou JTAG)

Hôte

Cible

Edition des sources

IDE, …

Codes source

Construction

Toolchain bibliothèques

exécutable

Transfert

Outils IDE
Ou ligne de commande

Visualisation capture

Protocol console

Débuguer

Protocol débug

Protocol transfert

Interface console

JTAG, …

Interface debug

Processeur ou microcontrôleur

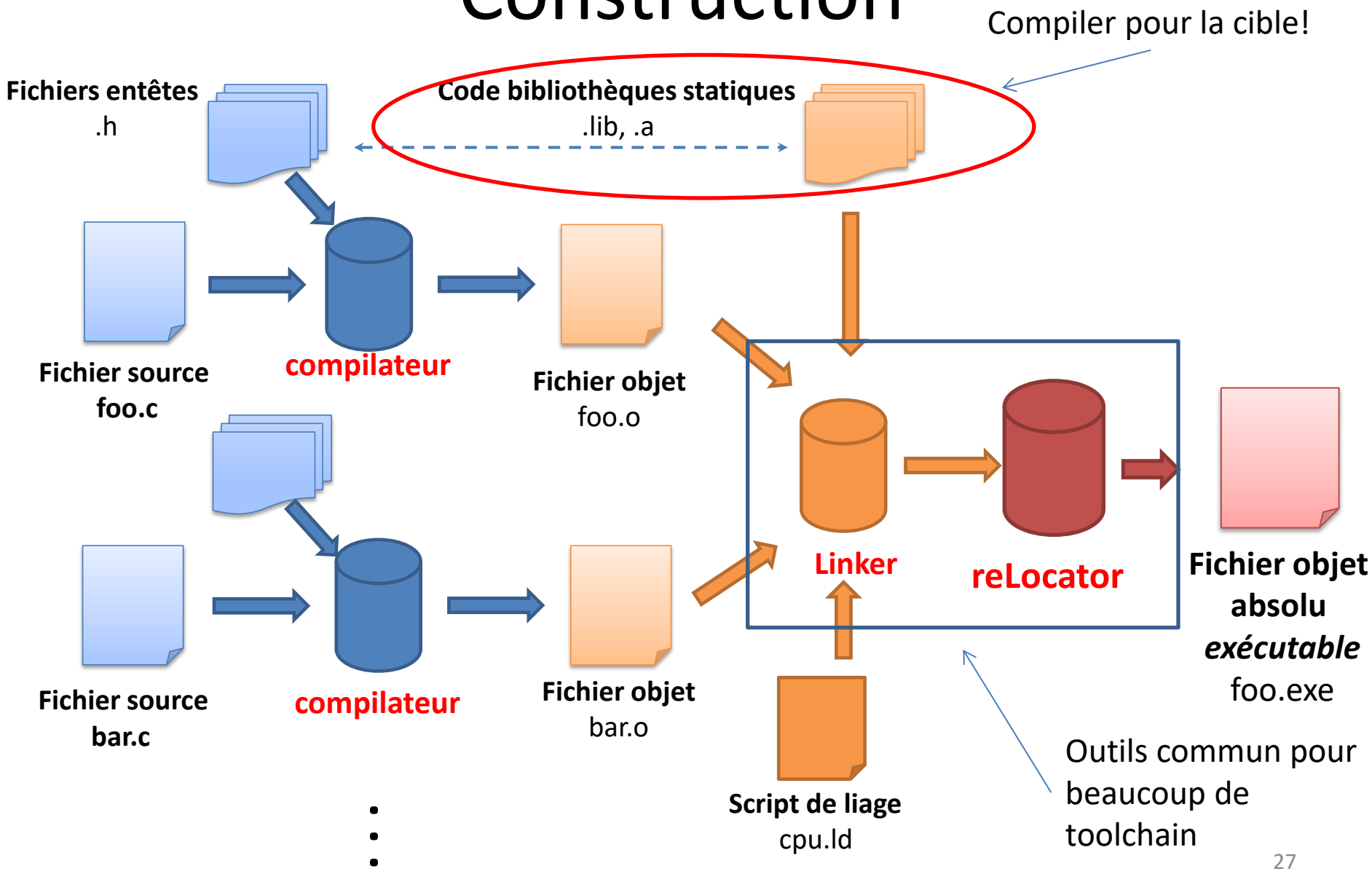
Mémoire interne ou externe

exécutable

Exemple

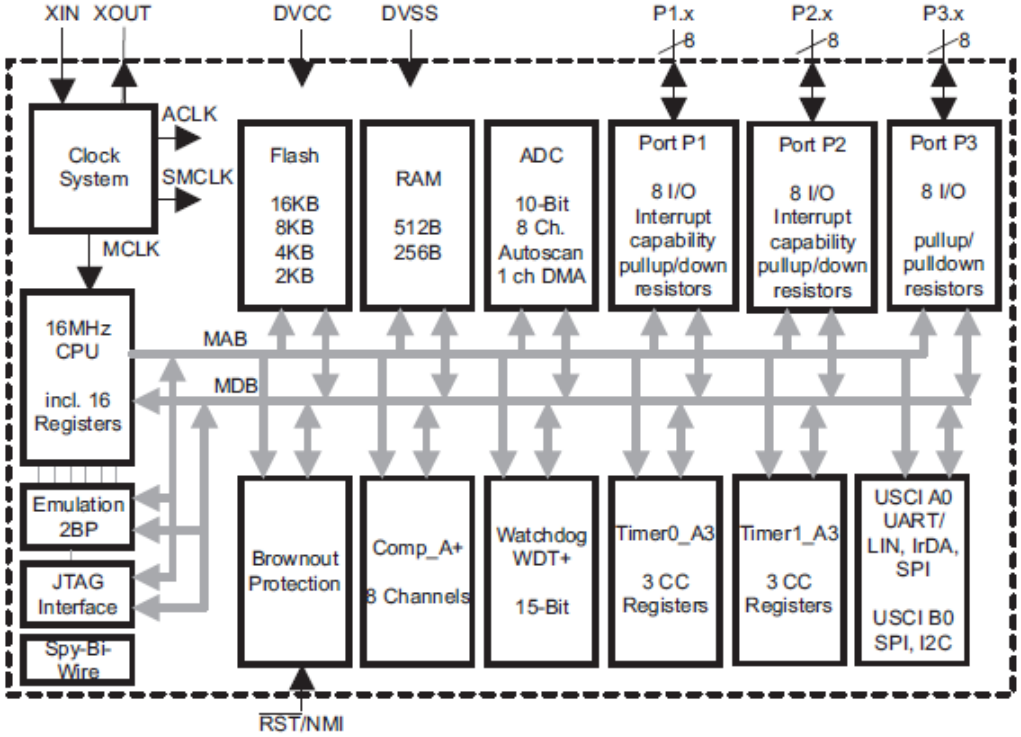
- ST Nucleo : STM32F0 (32 bits) (arm)
- **TI launchpad MSP430 G2553 (16 bits)**
  - 16MHz, 512o RAM, 16ko (16350o) ROM
  - 2,79 € / unit – 1290€ / 1000units (Farnell)
  - (kit launchpad msp430g2553 : ~12€)
  - *524 msp430 différents*
- Toolchain / IDE:
  - IAR, Keil, microship (XC8, XC16, …) …
  - CodeComposerStudio,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

Construction



Mon 1ier programme : Hello world…

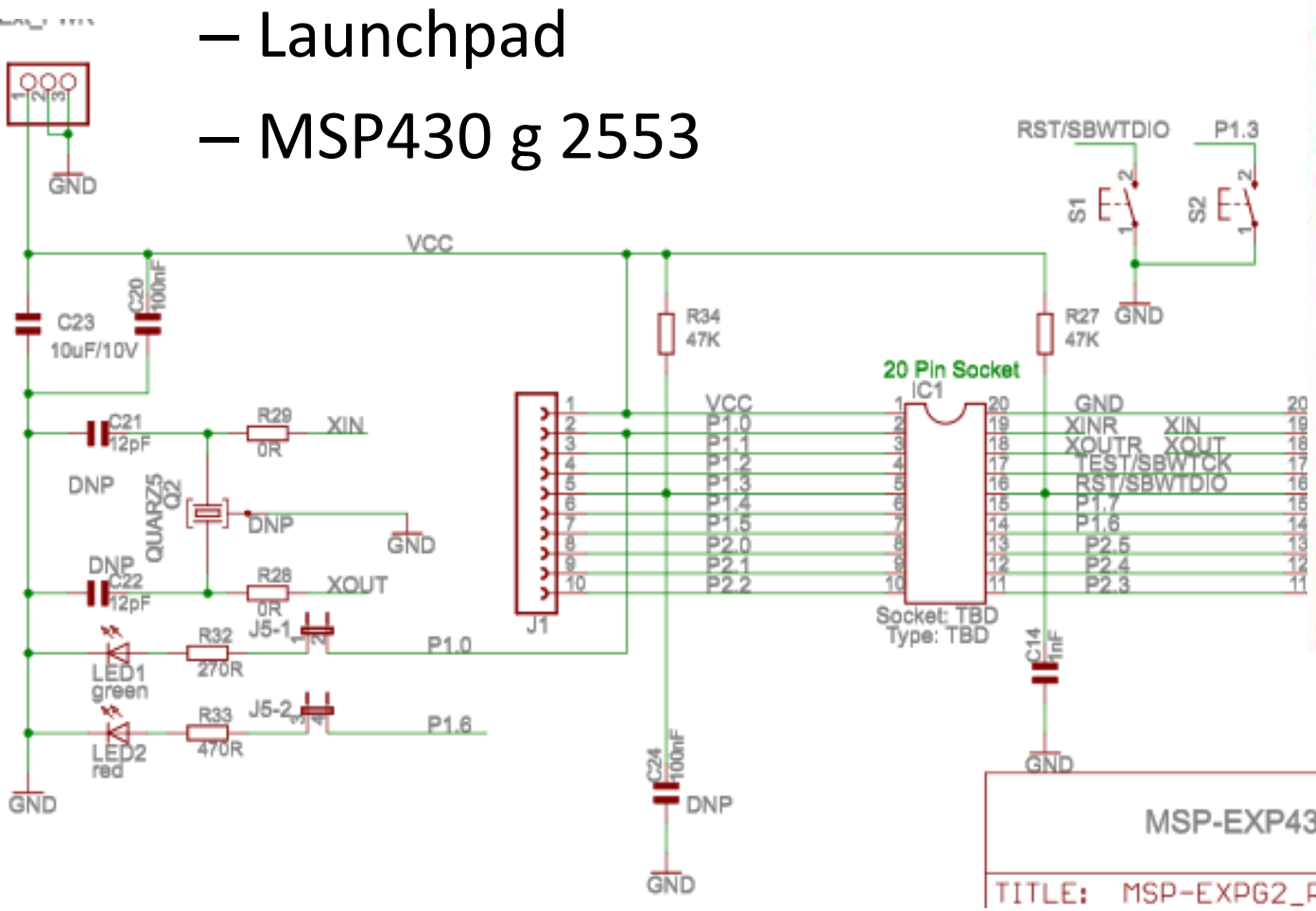
- Ecrire un programme pour un MSP430G2553 qui :
  - allume la led verte (P1.6) et éteigne la led rouge (P1.0) quand on appuie sur le bouton poussoir (P1.3)
  - Puis inversement dès qu'on relâche le bouton poussoir

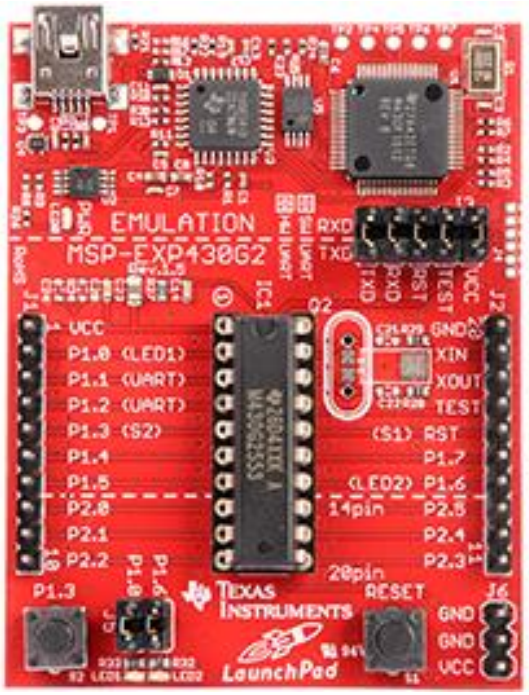


msp430g2553

Code c? … pas tout de suite

- Lecture de la doc technique…
  - Launchpad
  - MSP430 g 2553





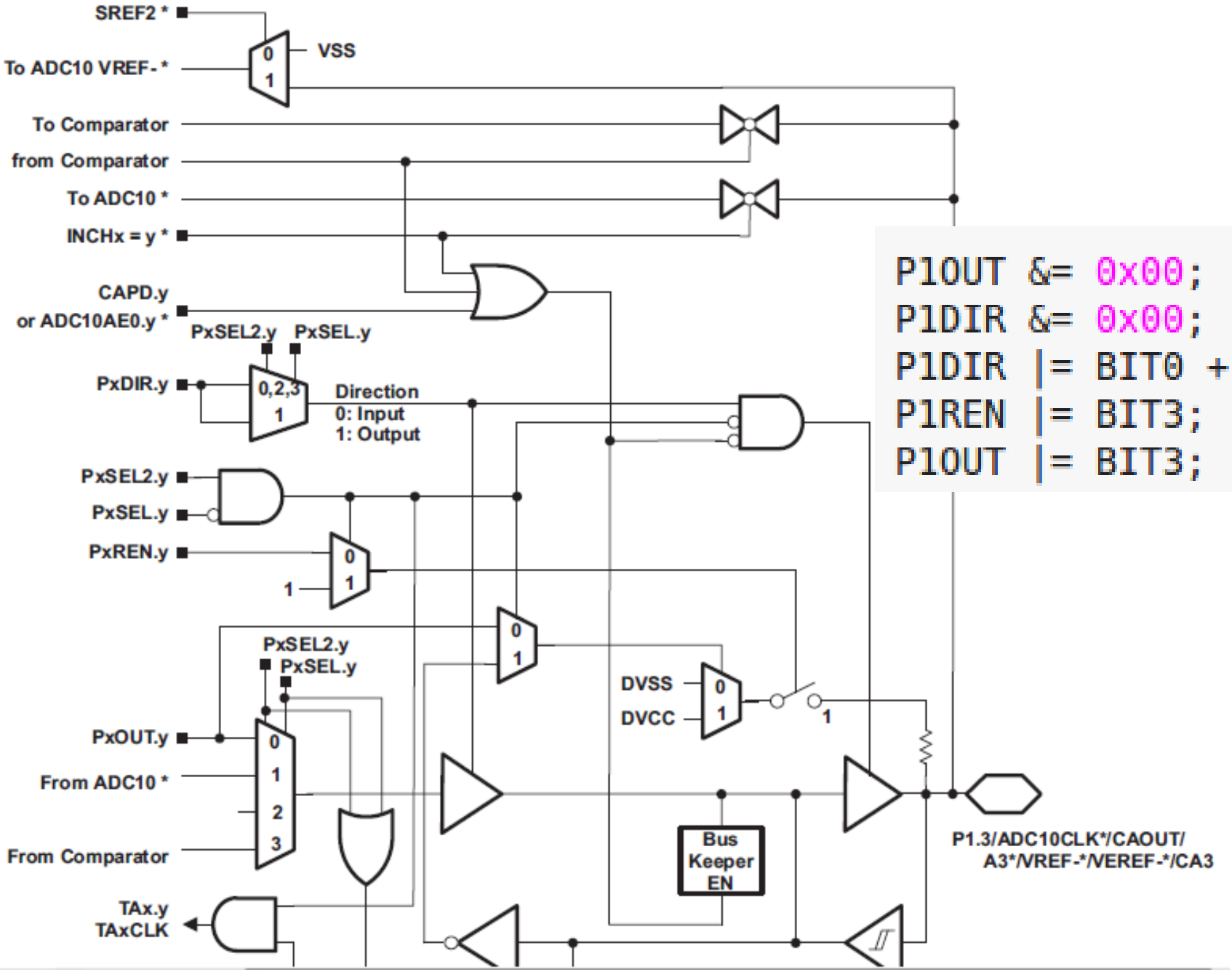
PORT 1

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 il existe plein de msp430, il faut compiler pour le g2553:
  → -mmcu=msp430g2553

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

Ecriture

- Système de ‘flash’ du microcontrôleur :
  - 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 …)
- **Cas du launchpad : mspdebug rf2500**
  - Se charge de tout: conversion binaire, programmation du processeur (via usb), exécution et debug

Tests…

- Led_bp.c
  - -g -c , objdump –S led_bp.o
  - -g , objdump –S led_bp.o; size led_bp.o
- Led_bp_fr.c
- Led_bp_fr.c + math…

Back into systems

- relocator

Documentation tech

		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
```

Deeper in C

- <stdint.h> : (C++11) uint8_t, …
- **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 fichier
- **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 IV

INTERRUPTIONS

Vocabulaire

- Interruption? C'est quoi??
- Source d'interruptions
  - **IRQ** : interrupt requested
  - Vecteur d'interuption
- Fonction de gestion de l'interruption : **ISR**
  - interrupt subroutine

Interruption, comment ca marche

- Tableau…

Interruptions en C

- **void __attribute__ ((interrupt(***VECTOR_IRQ***)))**
  ***My_Isr_for_vector*(void)**

- Exemple blink.c :
  - *VECTOR_IRQ* → PORT1_VECTOR
  - *My_Isr_for_vector* → Port_1
- Objdump –dS blink.out

Les étapes d'une ISR

- *1) (désactiver les nouvelles IRQ?)*
- 2) sauvegarder le contexte
- 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…)
- 6) restaurer le contexte
- 7) retour de fonction d'interruption

Les étapes d'une ISR à écrire en C

- *1) (désactiver les nouvelles IRQ?)*
- 2) sauvegarder le contexte
- **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…)**
- 6) restaurer le contexte
- 7) retour de fonction d'interruption