Structure du compilateur C

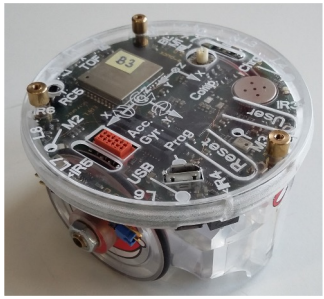
Francesco Mondada
IEM - STI - EPFL

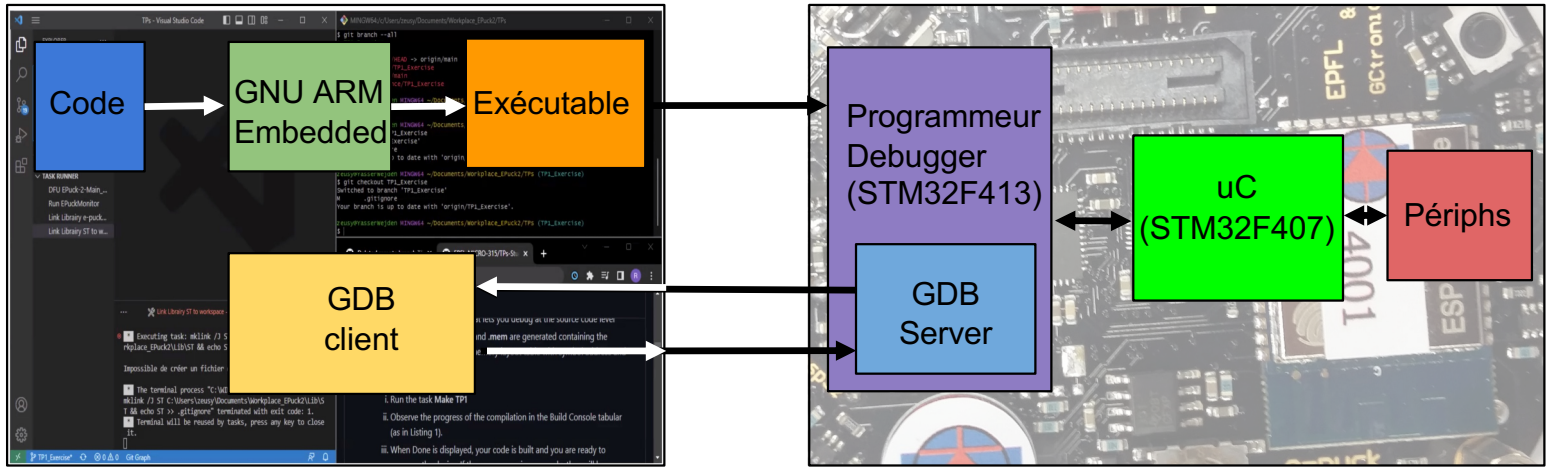
Plateforme pour ce cours

Vous avez pu noter que lors du premier TP, vous avez compilé un code sur un PC, qui s'est ensuite exécuté sur le robot e-puck2.

Dans ce cours, nous allons aborder la structure de ce procédé, soit la compilation du code C pour le transformer en un exécutable qui peut être exécuté sur le robot.

Plateforme pour ce cours





vscode

Introduction au compilateur C

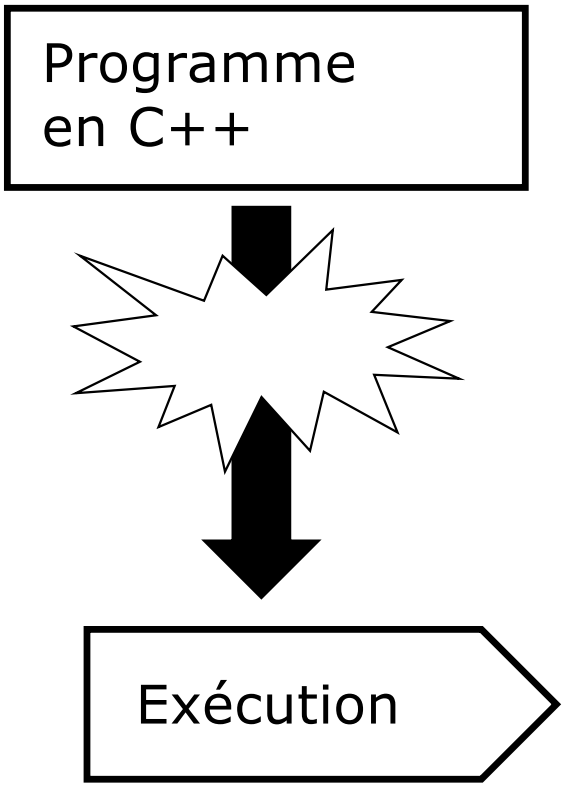
Ce cours se base sur le compilateur GCC en mode croisé.

GCC est le compilateur C du projet GNU, démarré en 1984 pour réaliser un OS similaire à UNIX mais libre. Le développement de GCC a débuté en 1987 et on est actuellement à la version 14.2 , le projet fait ~15M lignes de code. GCC est soumis à la license GPL (General Public License).

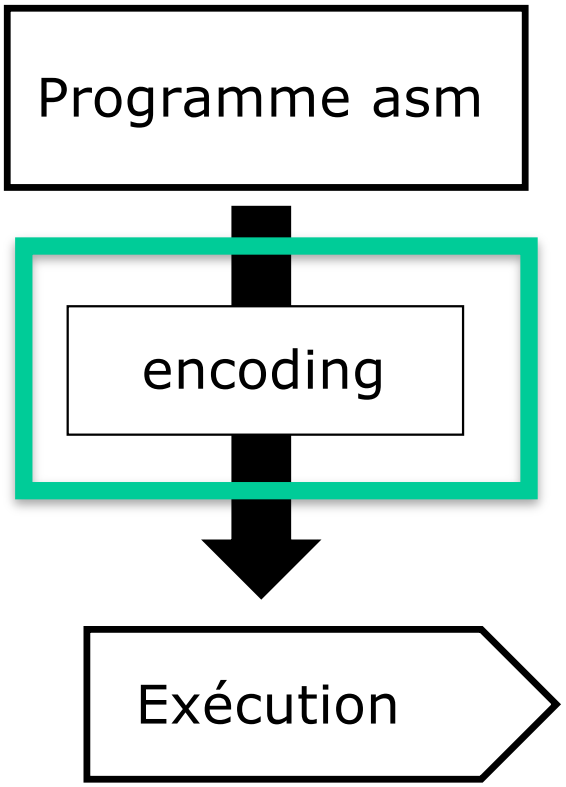
GCC supporte la grande majorité des processeurs existants à ce jour, est modulaire et permet de la compilation croisée (cross-compilation) pour un grand nombre de cibles.

Introduction

Informatique (1ère année)



Microcontroleur (2e année)



Exemple: MOV R1, #0x1

"The MOV instruction copies the value of Operand2 into Rd."

3.5.6 MOV and MVN

Move and Move NOT.

Syntax

```
MOV{S}{cond} Rd, Operand2
MOV{cond} Rd, #imm16
MVN{S}{cond} Rd, Operand2
```

where:

S	Is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see *Conditional execution* on page 3-18.
cond	Is an optional condition code. See *Conditional execution* on page 3-18.
Rd	Specifies the destination register.
Operand2	Is a flexible second operand, see *Flexible second operand* on page 3-12 for details of the options.
imm16	Is any value in the range 0-65535.

Cortex M4, Generic User Guide

Exemple

Instruction MOV{S}

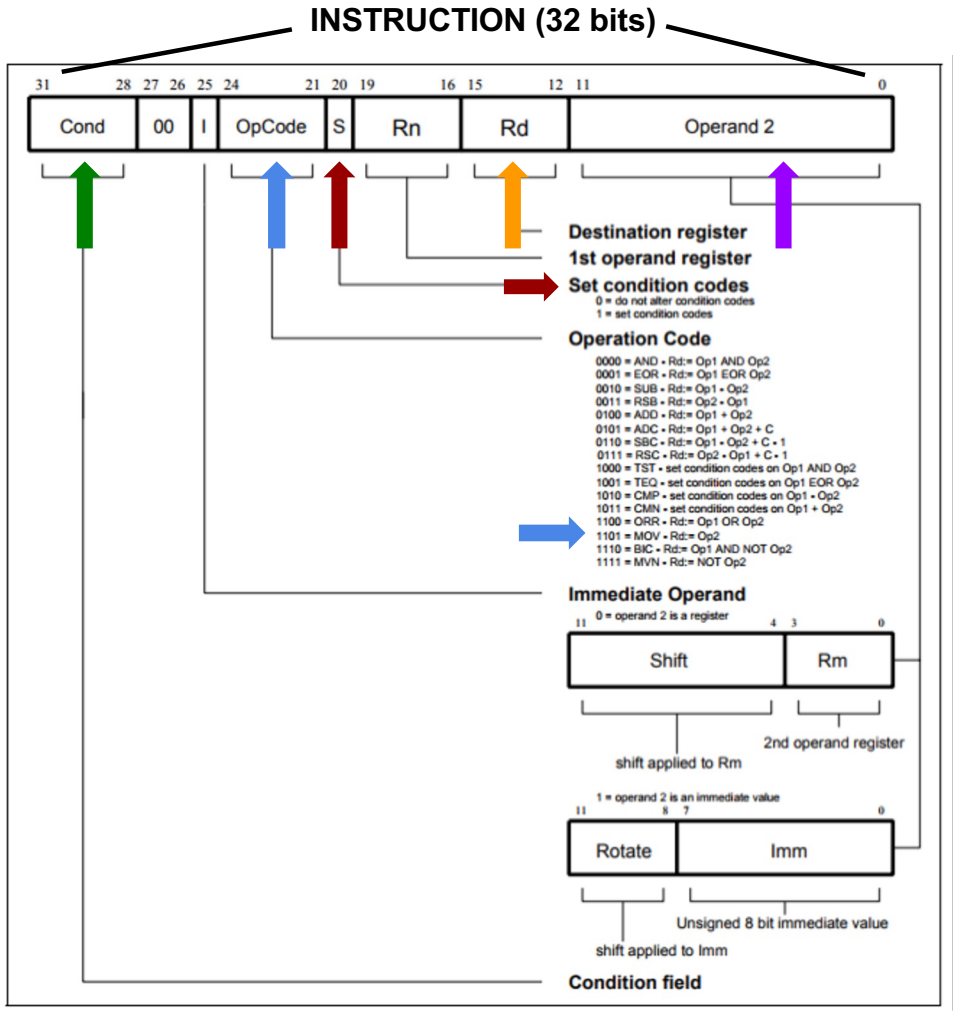
3.5.6 MOV and MVN

Move and Move NOT.

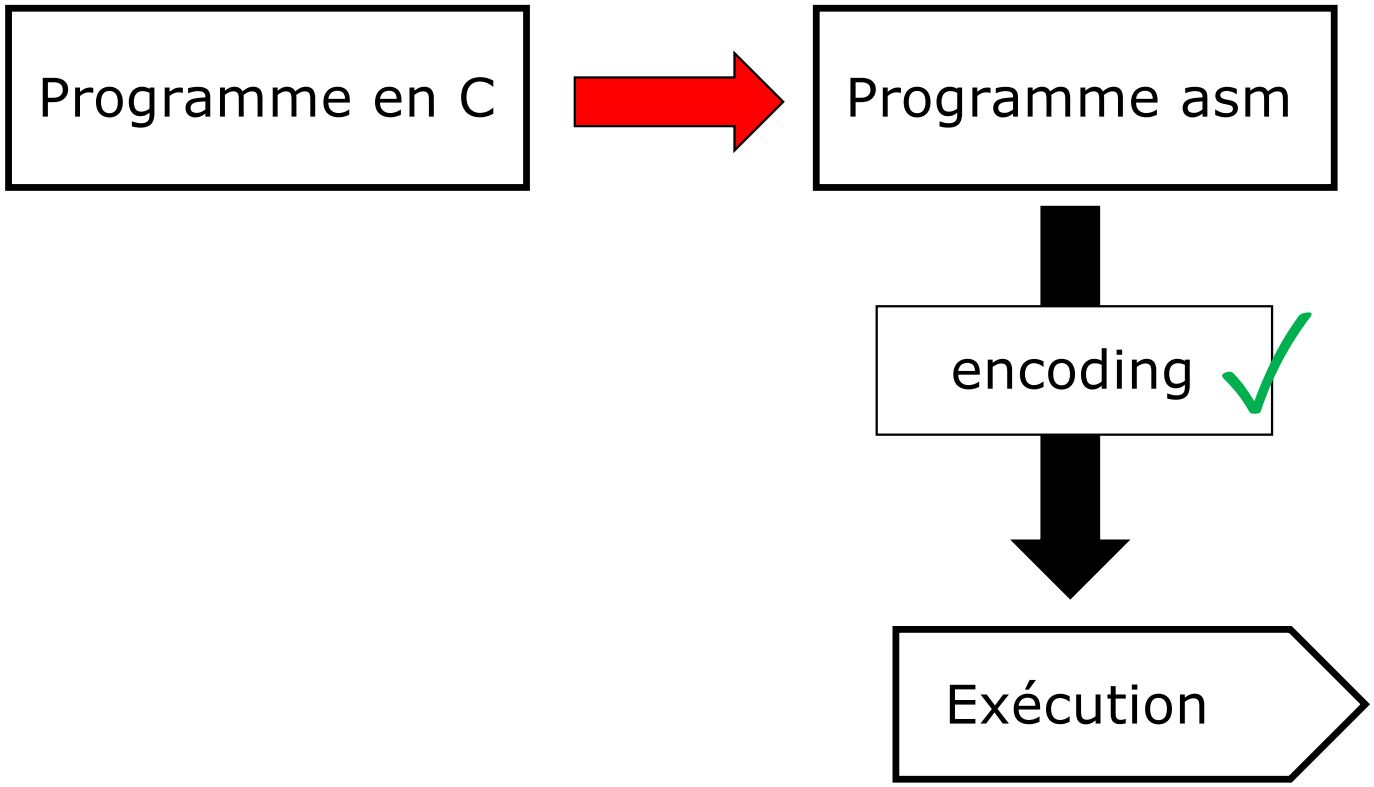
Syntax

```
MOV{S}{cond} Rd, Operand2
MOV{cond} Rd, #imm16
MVN{S}{cond} Rd, Operand2
```

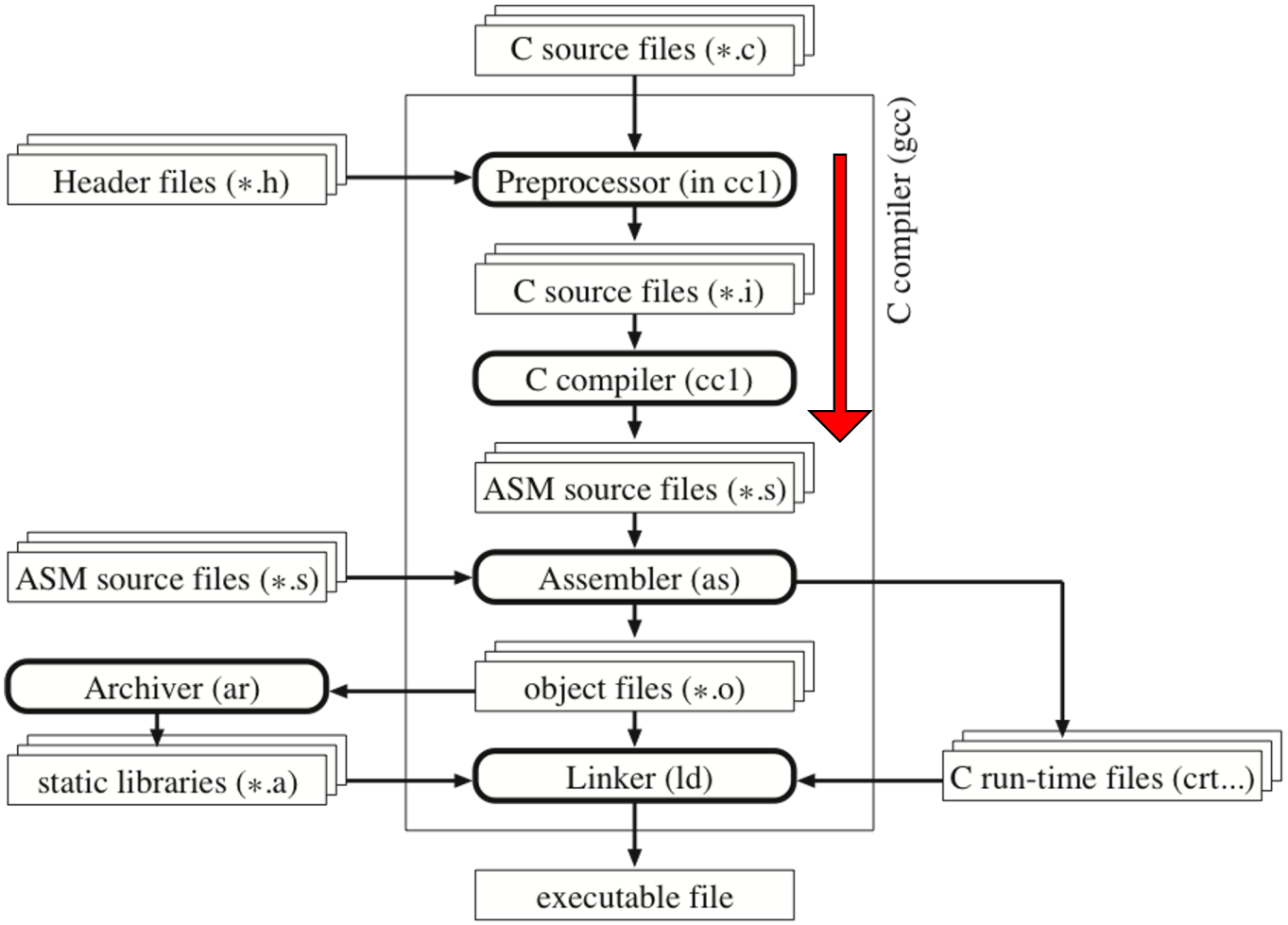
destination



Le compilateur C



Structure du compilateur C



Compiler *(from wikipedia)*

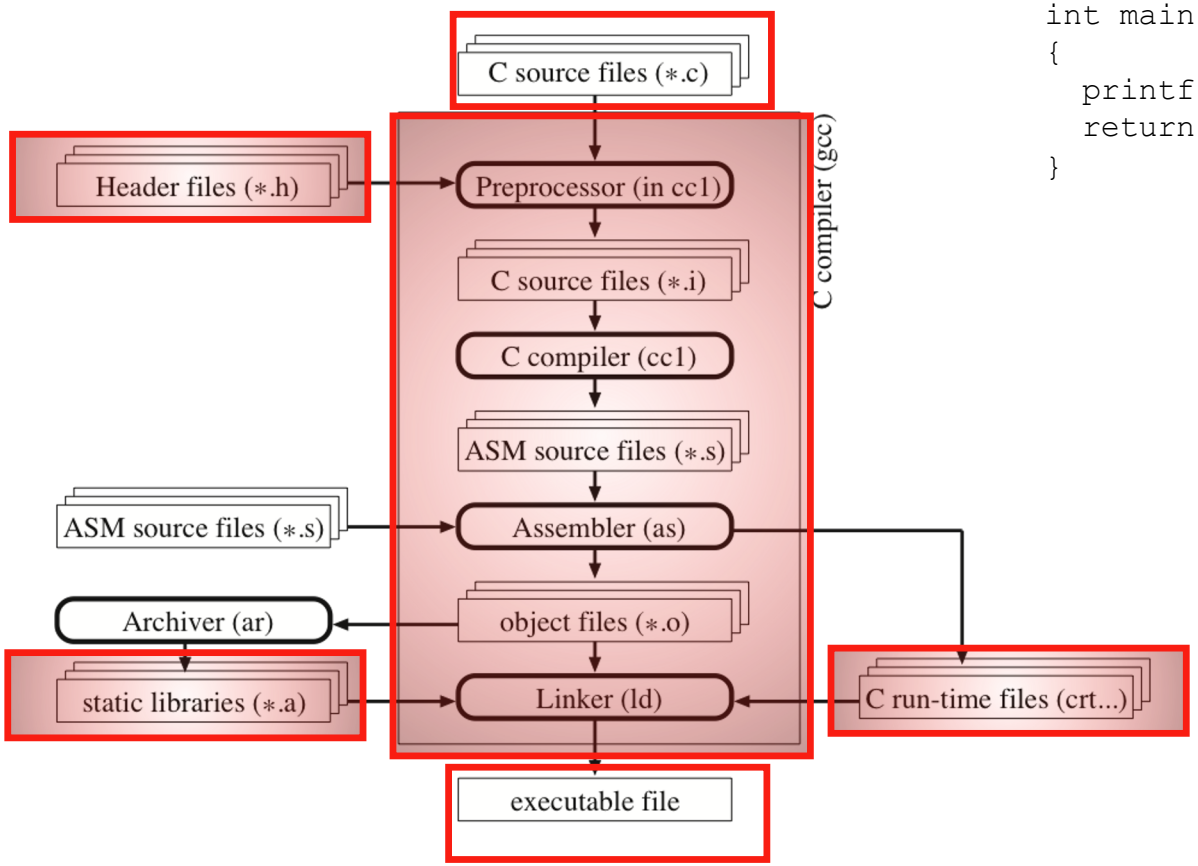
A compiler is a computer program (or set of programs) that transforms source code written in a computer language (the source language) into another computer language (the target language). The name "compiler" is primarily used for programs that translate source code from a high-level programming language **to a lower level language** (e.g., assembly language, object code, or machine code) to create an executable program.

Linker (éditeur de liens) *(from wikipedia)*

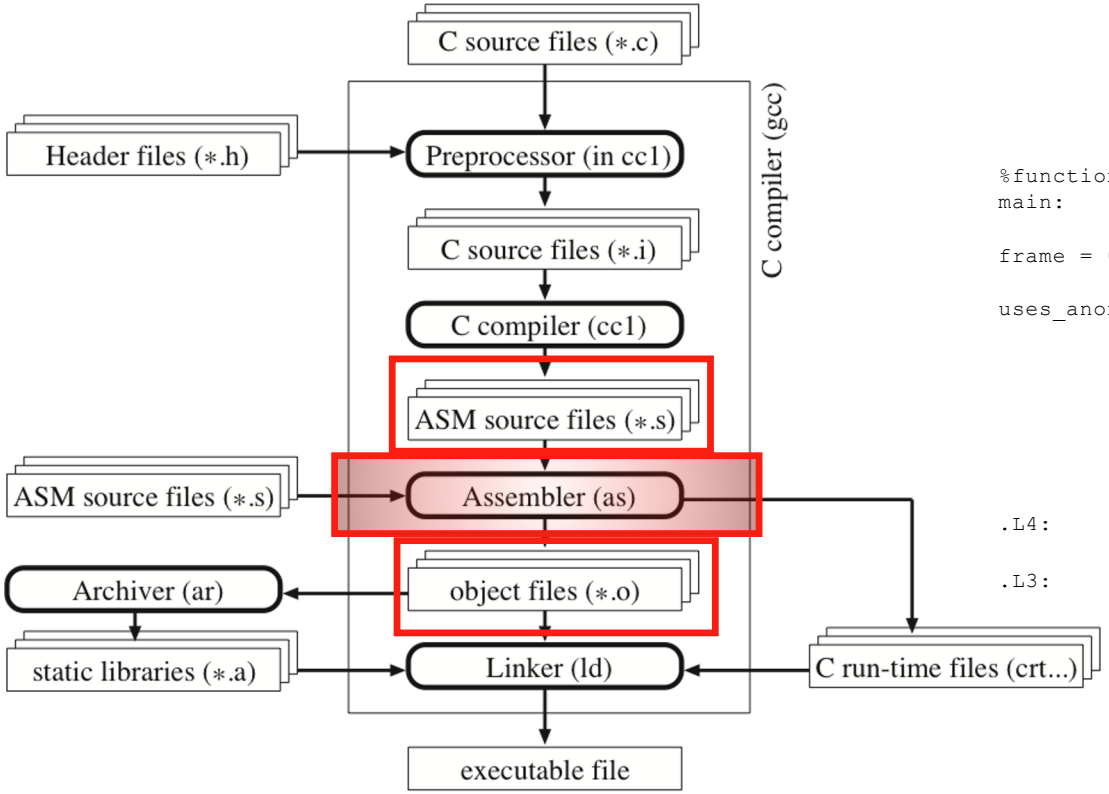
In computer science, a linker or link editor is a program that takes one or more objects generated by a compiler and combines them into a single executable program.

Exemples: gcc -o hello hello.c

```
#include <stdio.h>

int main()
{
  printf("Hello World\n");
  return (0);
}
```



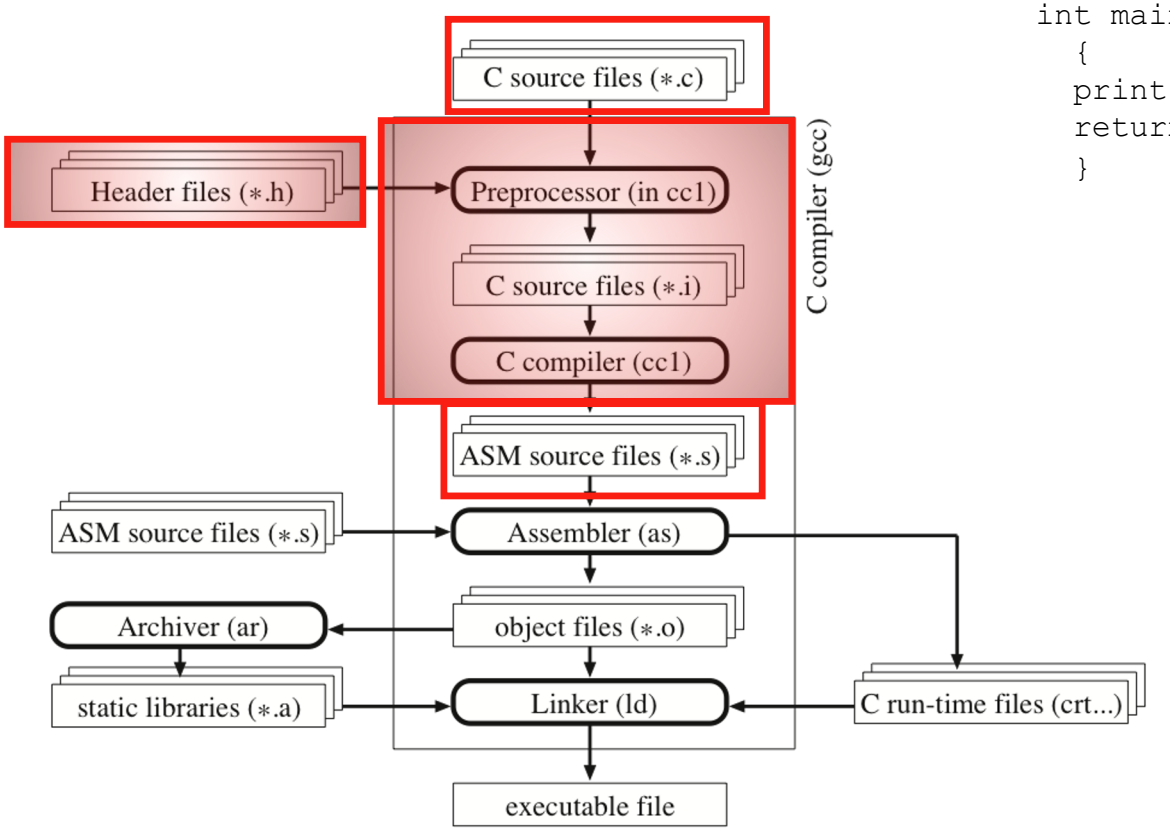
Exemples: as -o hello.o hello.s




```
.LC0:
                .ascii          "Hello
World\000"
                .text
                .align          1
                .global         main
                .syntax unified
                .thumb
                .thumb_func
                .fpu softvfp
                .type           main,
%function
main:
                @ args = 0, pretend = 0,
frame = 0
                @ frame_needed = 1,
uses_anonymous_args = 0
                push            {r7, lr}
                add             r7, sp, #0
                ldr             r0, .L3
                bl              puts
                movs            r3, #0
                mov             r0, r3
                pop             {r7, pc}
.L4:
                .align          2
.L3:
                .word           .LC0
                .size           main, .-main
```

Exemples: gcc -S hello.c

```
#include <stdio.h>

int main()
  {
  printf("Hello World\n");
  return (0);
  }
```



Cross-compilateur C

On parle de cross-compilateur quand le code est généré pour un autre processeur que celui sur lequel tourne le compilateur.

Afin de distinguer gcc (propre à l'ordinateur) des cross-gcc on ajoute un préfix au nom de gcc. Exemples:

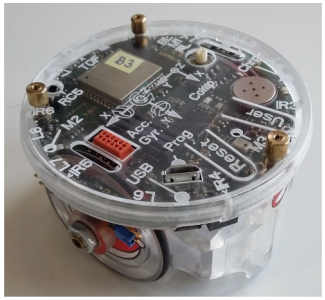
pic30-gcc
xc16-gcc
arm-none-eabi-gcc

La suite d'outils de gcc prennent la même forme (arm-none-eabi-ld, arm-none-eabi-as, etc.)

Compilation *vs* *Cross-compilation*

```
#include <stdio.h>

main()
  {
  printf("Hello World\n");
  return (0);
  }
```

```
#include <stdio.h>

main()
  {
  printf("Hello World\n");
  return (0);
  }
```



«Hello world»

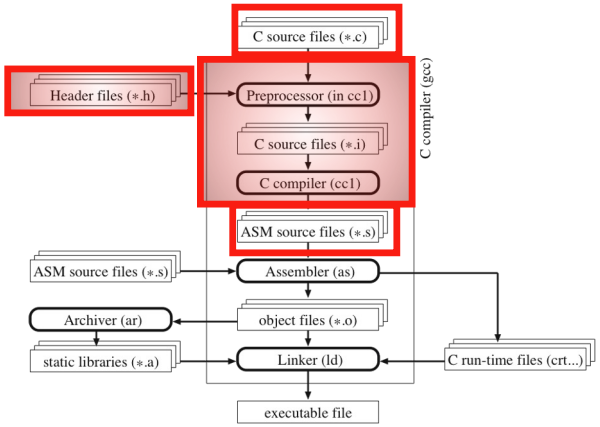
Cross-compilateur C de ARM

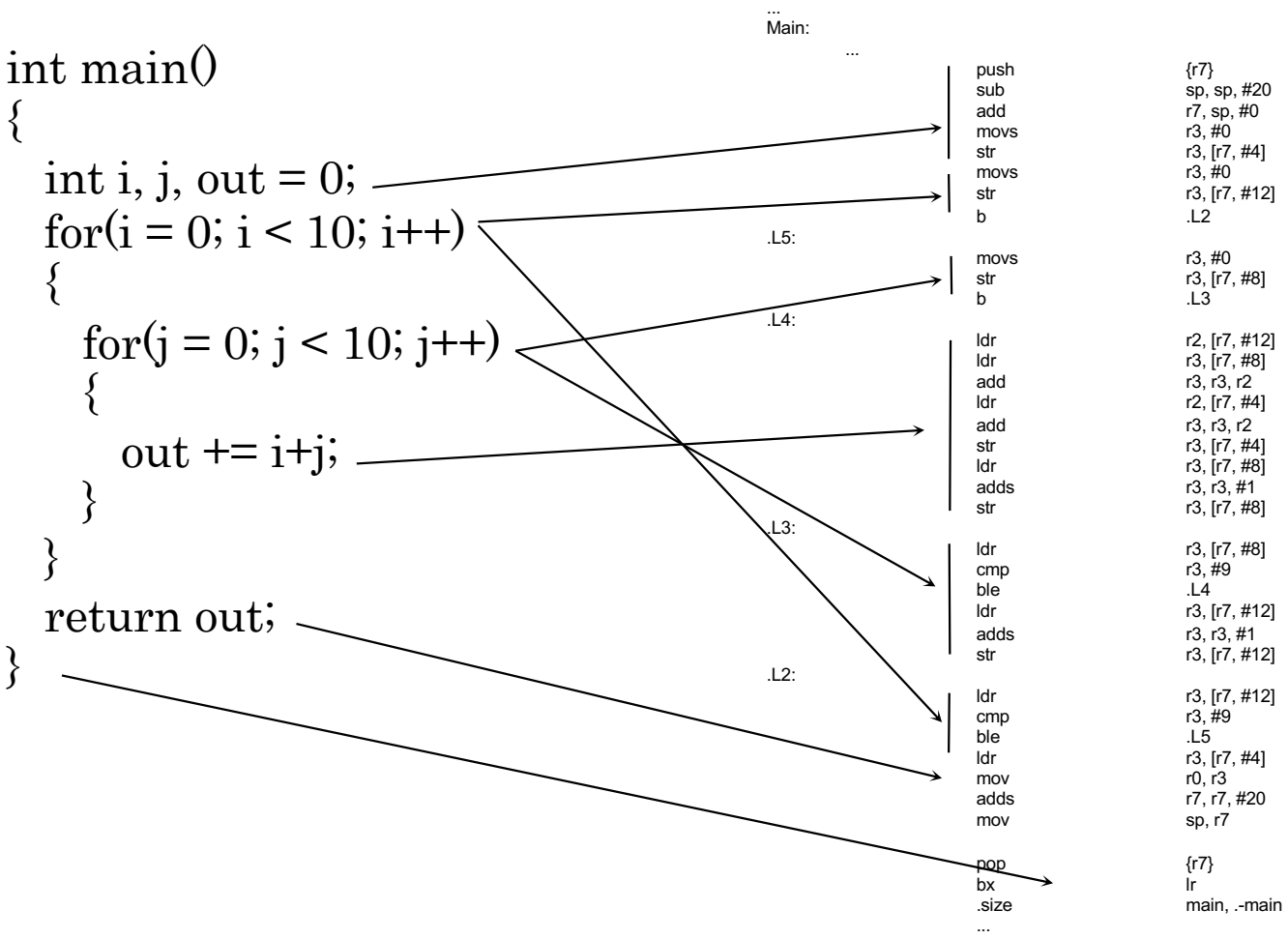
Il existe une version spécifique de GCC (*arm-none-eabi-gcc*) pour les microcontrôleurs avec une architecture ARM. C'est la chaîne d'outil (Toolchain) *GNU ARM Embedded*. Ce sont les fabricants des processeurs ARM qui maintiennent ce projet.

L'environnement **vscode** utilisé lors des TPs fait appel à ce compilateur lorsque vous faites un "build" de votre projet. Le système de construction est un simple Makefile faisant appel à tous les outil nécessaire. Mais il est aussi possible de lancer le compilateur à l'aide des lignes de commandes.

Exemples: arm-none-eabi-gcc -S test.c

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




int main()
{
int i, j, out = 0;
for(i = 0; i < 10; i++)
{
for(j = 0; j < 10; j++)
{
out += i+j;
}
}
return out;
}
...
Main:
...
push {r7}
sub sp, sp, #20
add r7, sp, #0
movs r3, #0
str r3, [r7, #4]
movs r3, #0
str r3, [r7, #12]
b .L2
.L5:
movs r3, #0
str r3, [r7, #8]
b .L3
.L4:
ldr r2, [r7, #12]
ldr r3, [r7, #8]
add r3, r3, r2
ldr r2, [r7, #4]
add r3, r3, r2
str r3, [r7, #4]
ldr r3, [r7, #8]
adds r3, r3, #1
str r3, [r7, #8]
.L3:
ldr r3, [r7, #8]
cmp r3, #9
ble .L4
ldr r3, [r7, #12]
adds r3, r3, #1
str r3, [r7, #12]
.L2:
ldr r3, [r7, #12]
cmp r3, #9
ble .L5
ldr r3, [r7, #4]
mov r0, r3
adds r7, r7, #20
mov sp, r7
pop {r7}
bx lr
.size main, .-main
...

Organisation de code

Frank Bonnet, Francesco Mondada
IMT - STI - EPFL

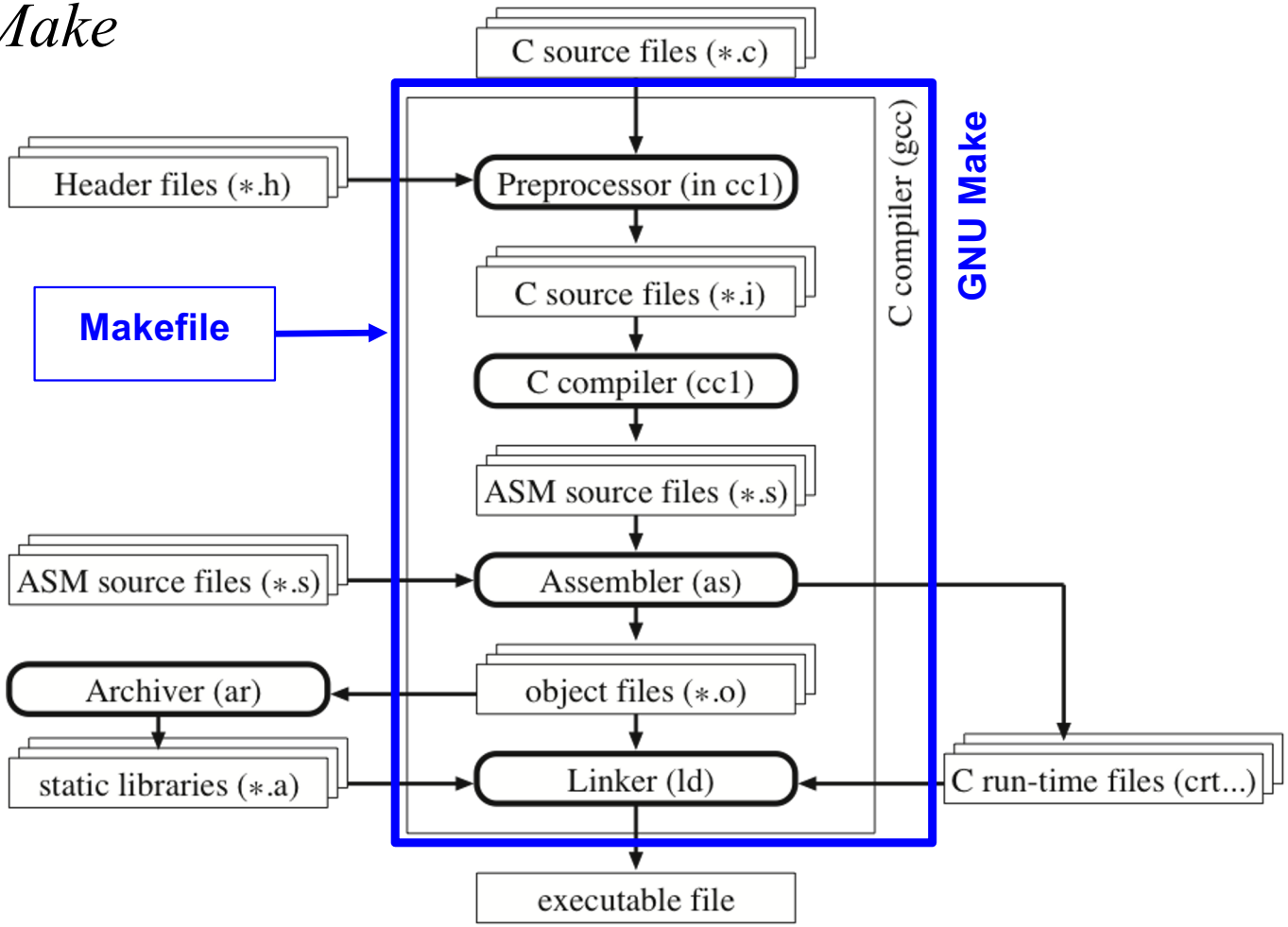
Introduction

Afin de gérer des projets de tailles différentes, une structuration du code source est nécessaire.
La structure peut prendre des formes différentes, mais quelques principes sont généraux et font appel aux possibilités offertes par le compilateur.

Dans ce cadre il est important de comprendre:

- L'utilité de structurer le code en sous-parties
- Le rôle des différents fichiers et leur utilisation:
  makefile, fichier source, définitions, librairies, de démarrage...
- Le contexte donné par un BIOS (Basic Input/Ouput System)
  ou un OS (Operating System)

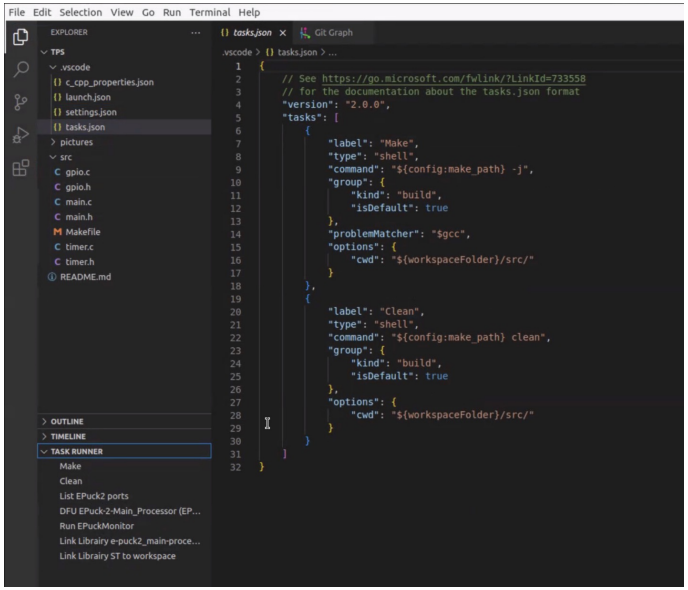
Make


C source files (∗.c)
Header files (∗.h)
Preprocessor (in cc1)
C source files (∗.i)
Makefile
C compiler (cc1)
ASM source files (∗.s)
ASM source files (∗.s)
Assembler (as)
Archiver (ar)
object files (∗.o)
static libraries (∗.a)
Linker (ld)
C run-time files (crt...)
executable file
C compiler (gcc)
GNU Make

Make

Wikipedia: **Make** is a build automation tool that automatically builds executable programs and libraries from source code by reading files called **Makefiles** which specify how to derive the target program.

Lorsque vous exécutez “build” dans vscode, vous lancez la commande **Make** qui va faire utiliser les informations du makefile pour compiler votre projet
Le makefile rend le procédé de compilation de gros projets lourds avec plusieurs contributeurs beaucoup plus simples.



Makefile (Du TP)

```
# Project, sources and paths
#

PROJECT = blinky

CSRC = main.c gpio.c timer.c
CSRC += ST/system_clock_config.c ST/stm32f4xx_ll_rcc.c ST/system_stm32f4xx.c

ASMSRC = ST/startup_stm32f407xx.s

INCDIR = . ./ST

LIBS =
LIBDIR =

LDSCRIPT = ST/STM32F407VGTx_FLASH.ld

DEFS = -DSTM32F4 -DSTM32F407xx

OBJS = $(CSRC:.c=.o) $(ASMSRC:.s=.o)
IINCDIR = $(patsubst %,-I%,$(INCDIR))
LLIBDIR = $(patsubst %,-L%,$(LIBDIR))
```

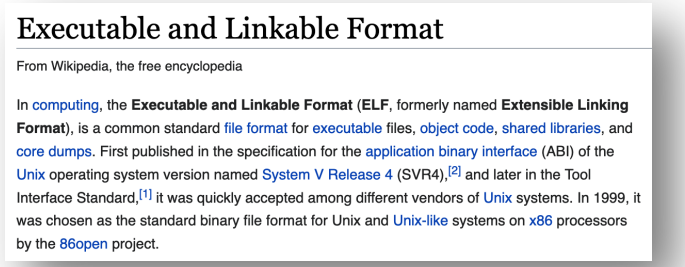
LD = linker

Makefile (Du TP)

```
# Compiler settings

MCU  = cortex-m4

TRGT = arm-none-eabi-
CC   = $(TRGT)gcc
LD   = $(TRGT)gcc
AS   = $(TRGT)gcc -x assembler-with-cpp
OD   = $(TRGT)objdump
SZ   = $(TRGT)size
NM   = $(TRGT)nm

# Compiler options
OPT = -O0 -ggdb -fomit-frame-pointer -falign-functions=16
OPT += -ffunction-sections -fdata-sections -fno-common

# THUMB-specific options
TOPT = -mthumb -mno-thumb-interwork

# Define C warning options
CWARN = -Wall -Wextra -Wundef -Wstrict-prototypes

CFLAGS = -mcpu=$(MCU) $(OPT) $(CWARN) $(DEFS) $(TOPT)

ASFLAGS = -mcpu=$(MCU) $(TOPT)

LDFLAGS = -mcpu=$(MCU) $(OPT) -nostartfiles $(LLIBDIR) $(TOPT)
LDFLAGS += -Wl,--no-warn-mismatch,--gc-sections,--script=$(LDSCRIPT)
```

Makefile (Du TP)

```
# Make targets
#

.PHONY: all
all: $(PROJECT).elf $(PROJECT).list $(PROJECT).size $(PROJECT).mem
                $(SZ) $(PROJECT).elf
                @ echo "> Done"

.PHONY: clean
clean:
                @echo "> Cleaning"
                rm -f $(OBJS)
                rm -f $(PROJECT).elf $(PROJECT).list $(PROJECT).size $(PROJECT).mem
                @echo "> Done"
…
...

$(PROJECT).mem: $(PROJECT).elf
                @echo "> Creating" $@
                $(NM) --numeric-sort --print-size $< > $@

.PHONY: flash
flash: all
                arm-none-eabi-gdb --command=../../debug.gdb --command=../../flash.gdb blinky.elf
```



Executable and Linkable Format

From Wikipedia, the free encyclopedia

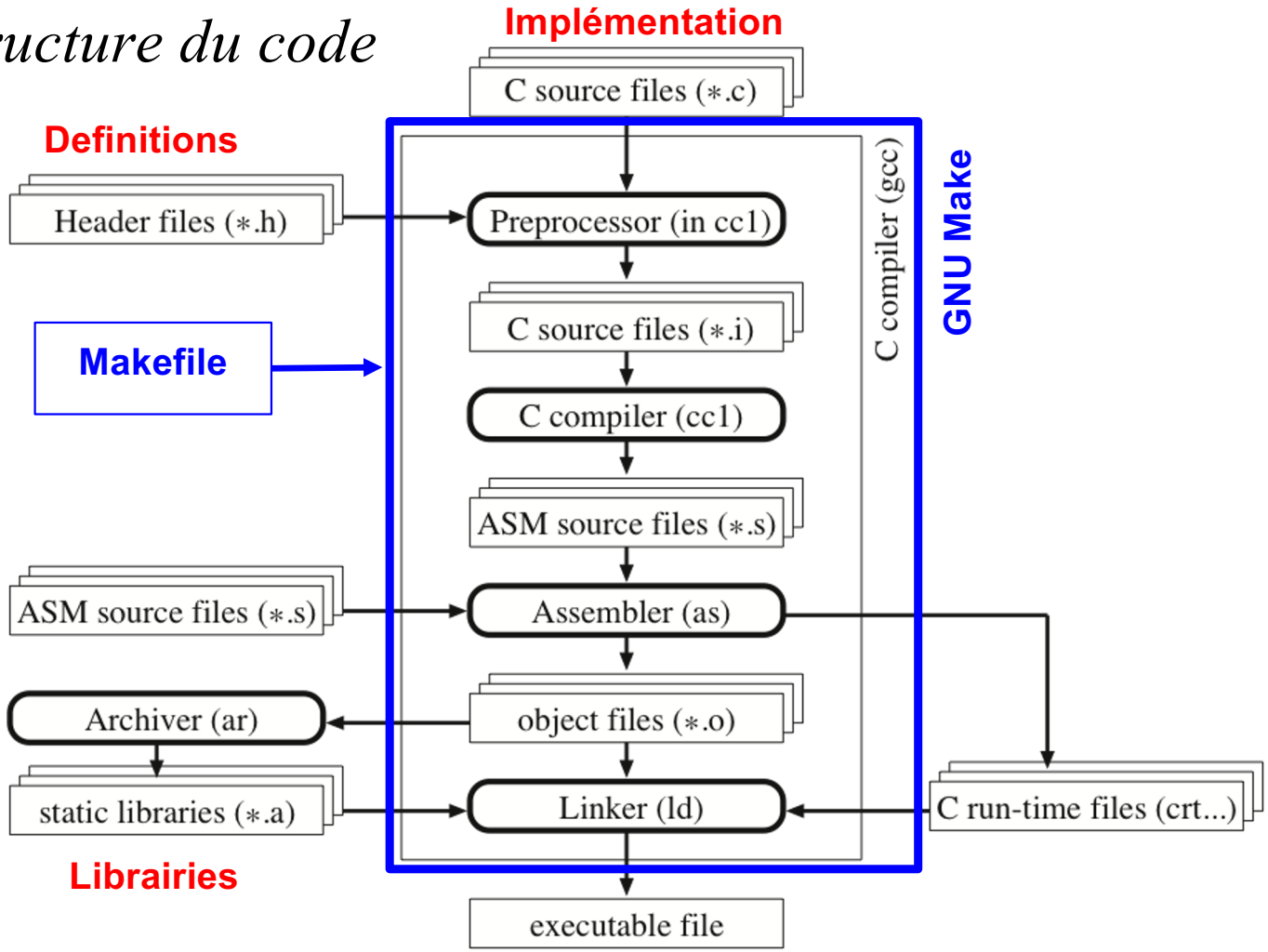
In computing, the **Executable and Linkable Format** (**ELF**, formerly named **Extensible Linking Format**), is a common standard file format for executable files, object code, shared libraries, and core dumps. First published in the specification for the application binary interface (ABI) of the Unix operating system version named System V Release 4 (SVR4),[2] and later in the Tool Interface Standard,[1] it was quickly accepted among different vendors of Unix systems. In 1999, it was chosen as the standard binary file format for Unix and Unix-like systems on x86 processors by the 86open project.

Makefile

Un makefile peut inclure plusieurs autres makefiles! (à partir du TP4), car chaque librairie ou sous-projet peut avoir son propre makefile

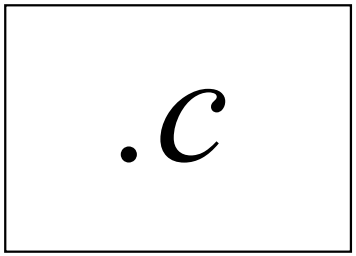
```
# Project, sources and paths
#

# Imported source files and paths
CHIBIOS = $(GLOBAL_PATH)/ChibiOS/
CHIBIOS_EXT = $(GLOBAL_PATH)/ChibiOS_ext/
# Startup files.
include $(CHIBIOS)/os/common/ports/ARMCMx/compilers/GCC/mk/startup_stm32f4xx.mk
# HAL-OSAL files.
include $(CHIBIOS)/os/hal/hal.mk
include $(CHIBIOS)/os/hal/ports/STM32/STM32F4xx/platform.mk
```

Structure du code



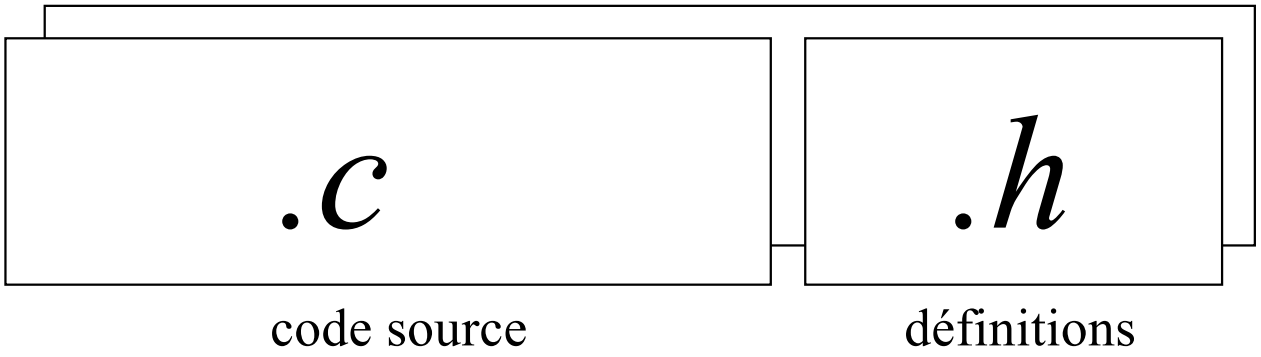
Structure de code

Petit programme C:



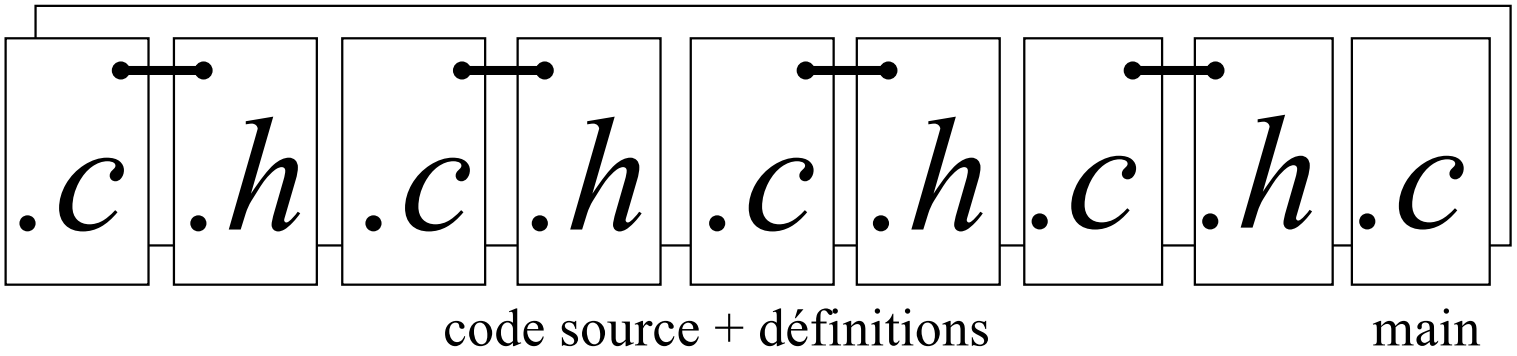
Structure de code

Programme moyen en C:



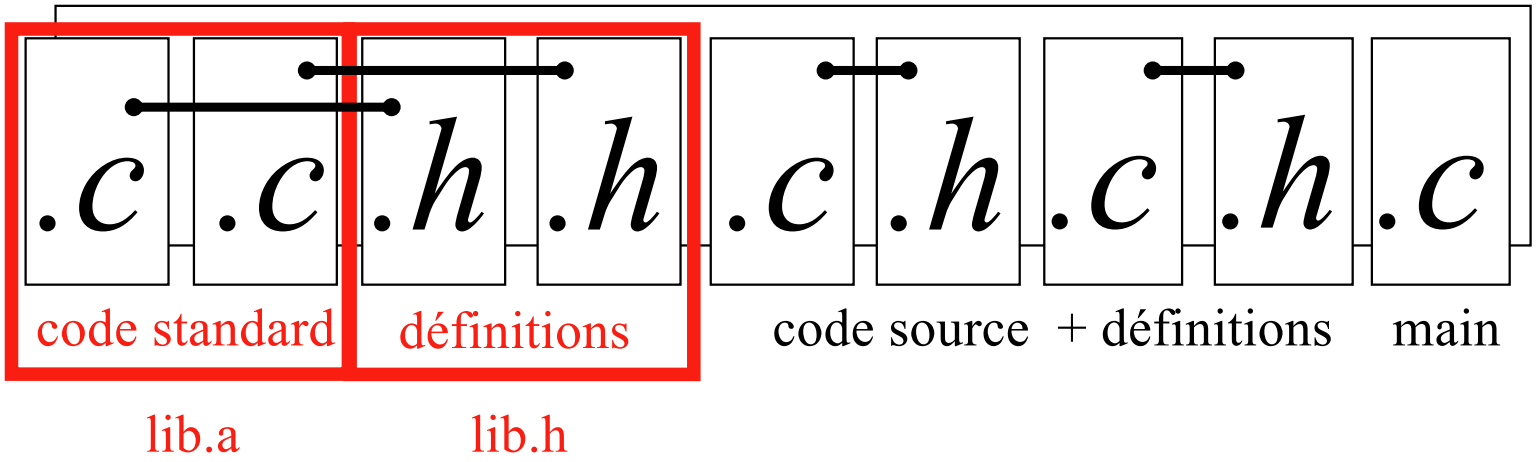
Structure de code

Programme complexe en C:

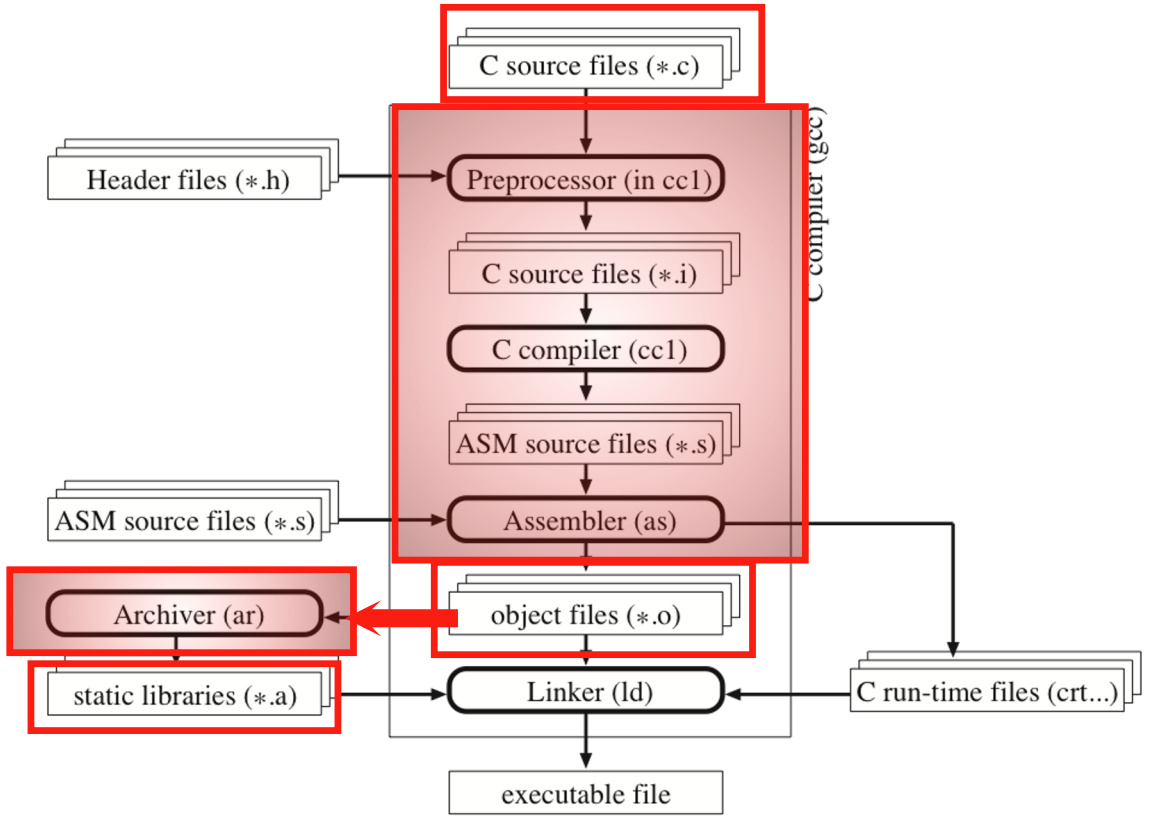


Structure de code

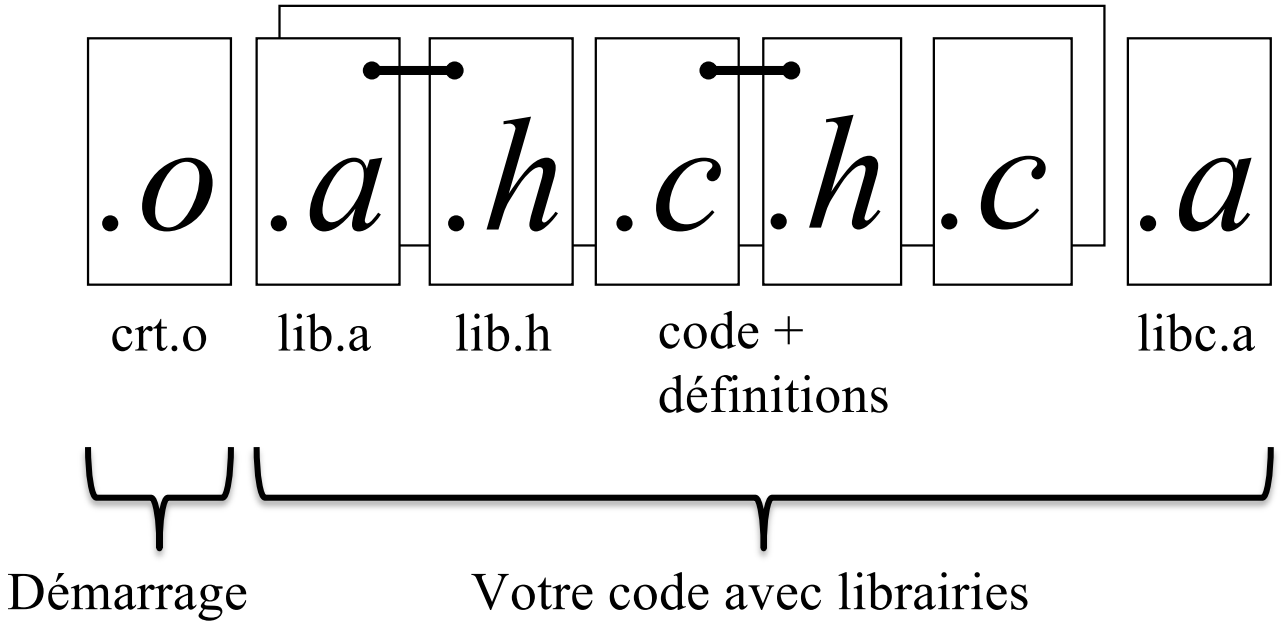
Programme complexe en C:



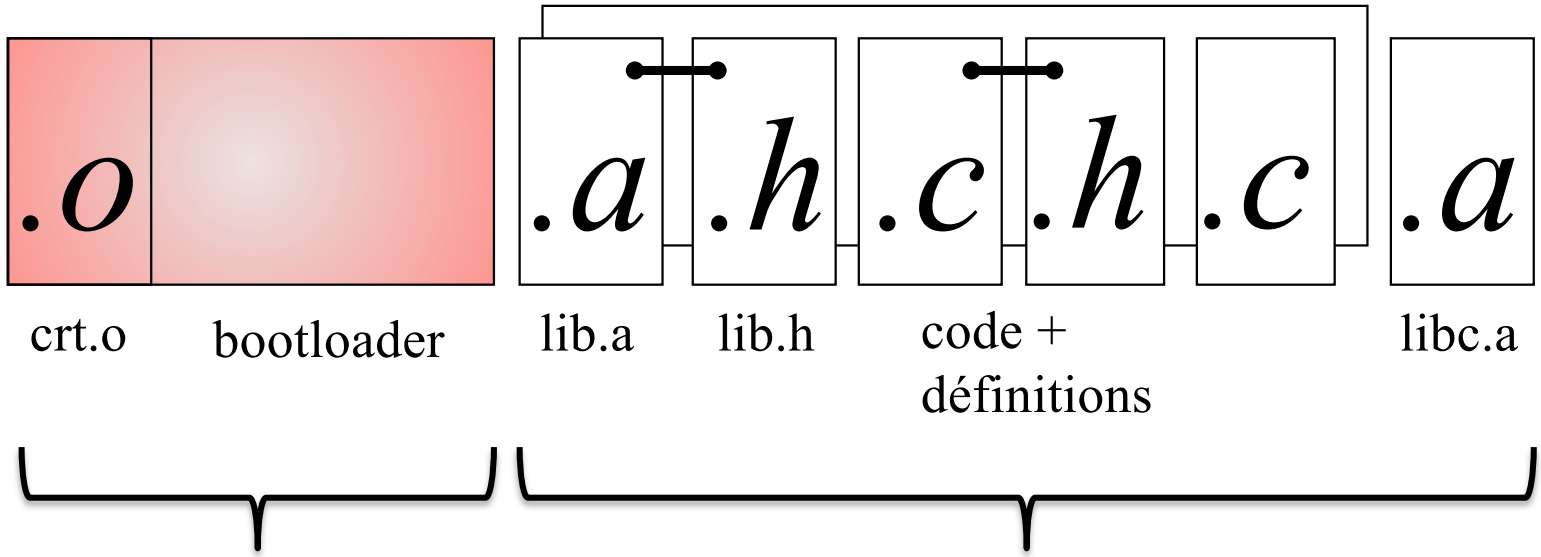
Création de librairies

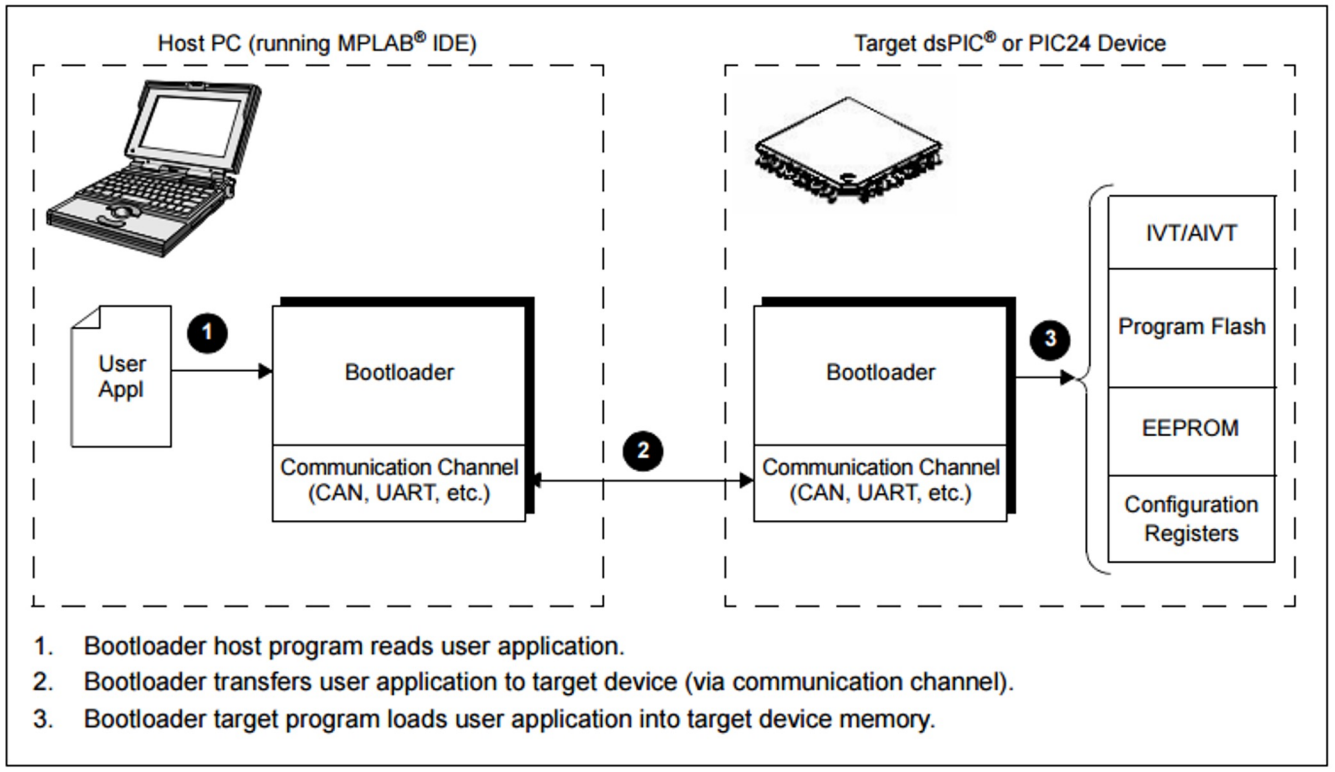


Environnement



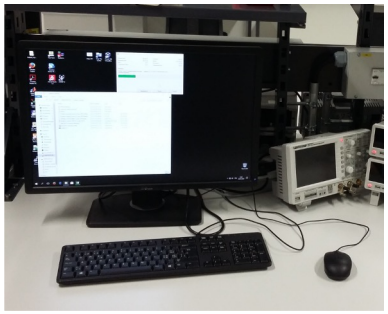
Environnement





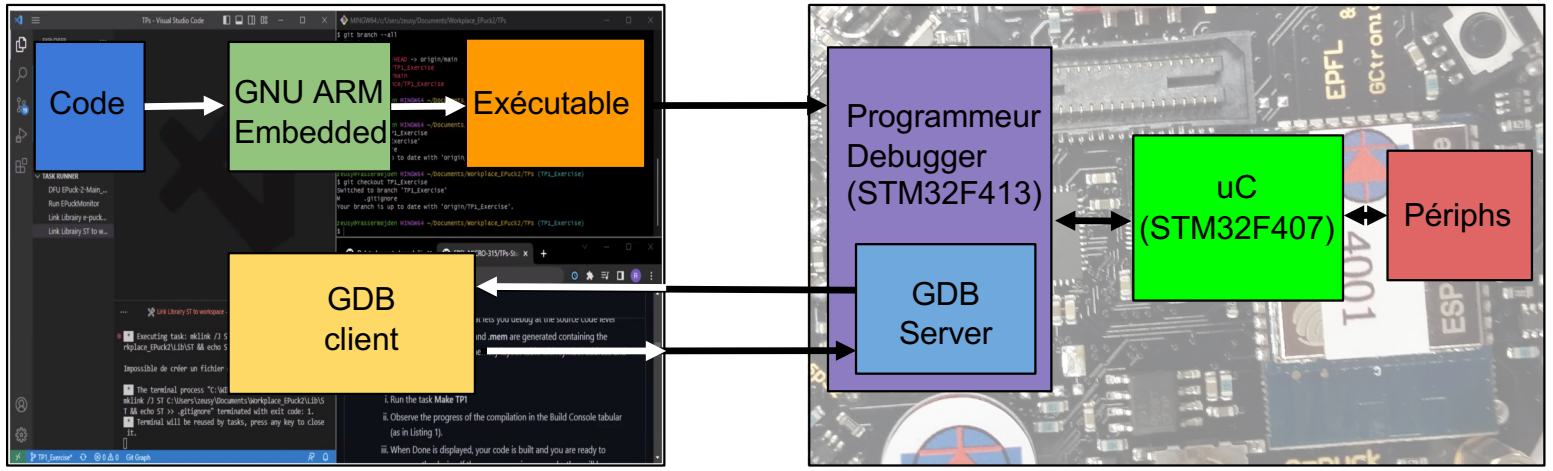
1. Bootloader host program reads user application.
2. Bootloader transfers user application to target device (via communication channel).
3. Bootloader target program loads user application into target device memory.

Sans Bootloader

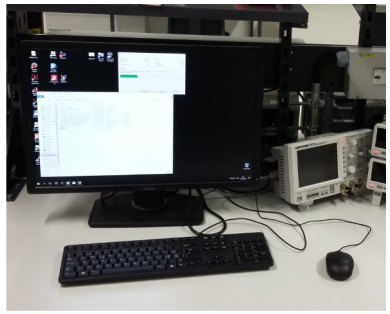


```
#include <stdio.h>

main()
   {
   printf("Hello World\n");
   return (0);
   }
```

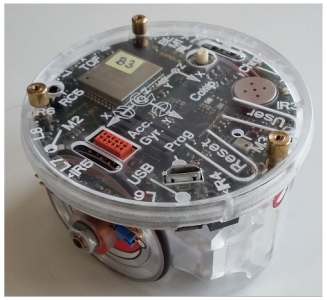
«Hello world»



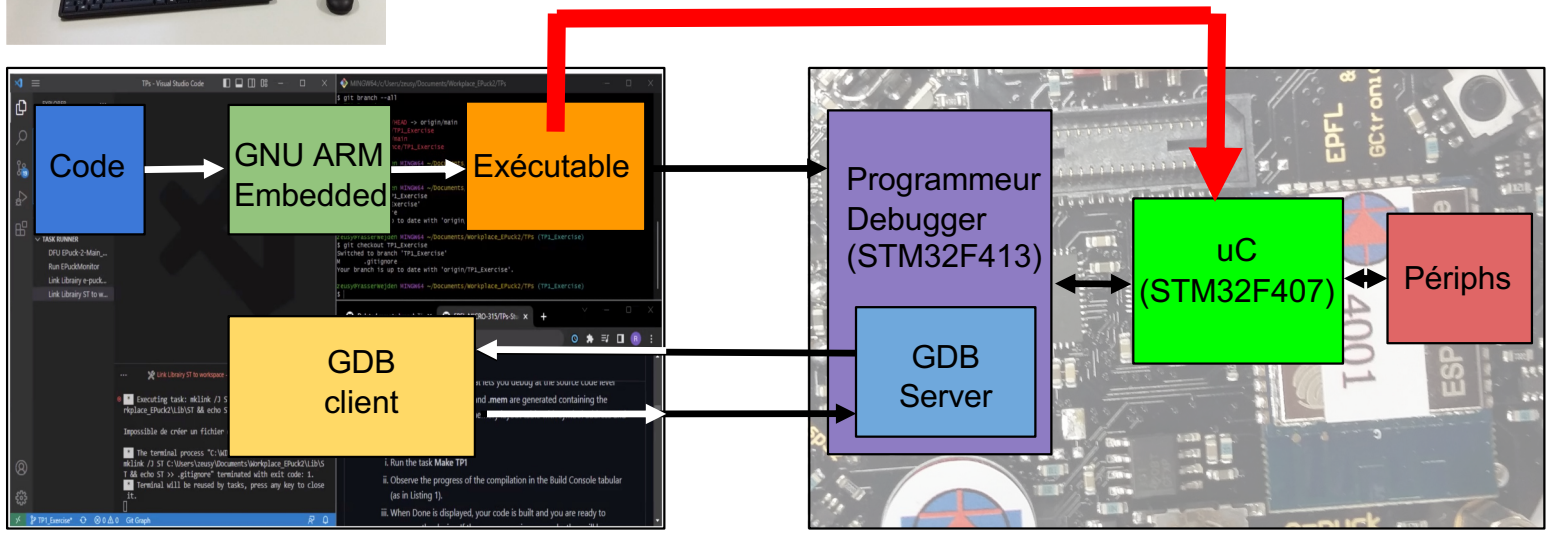
Avec Bootloader



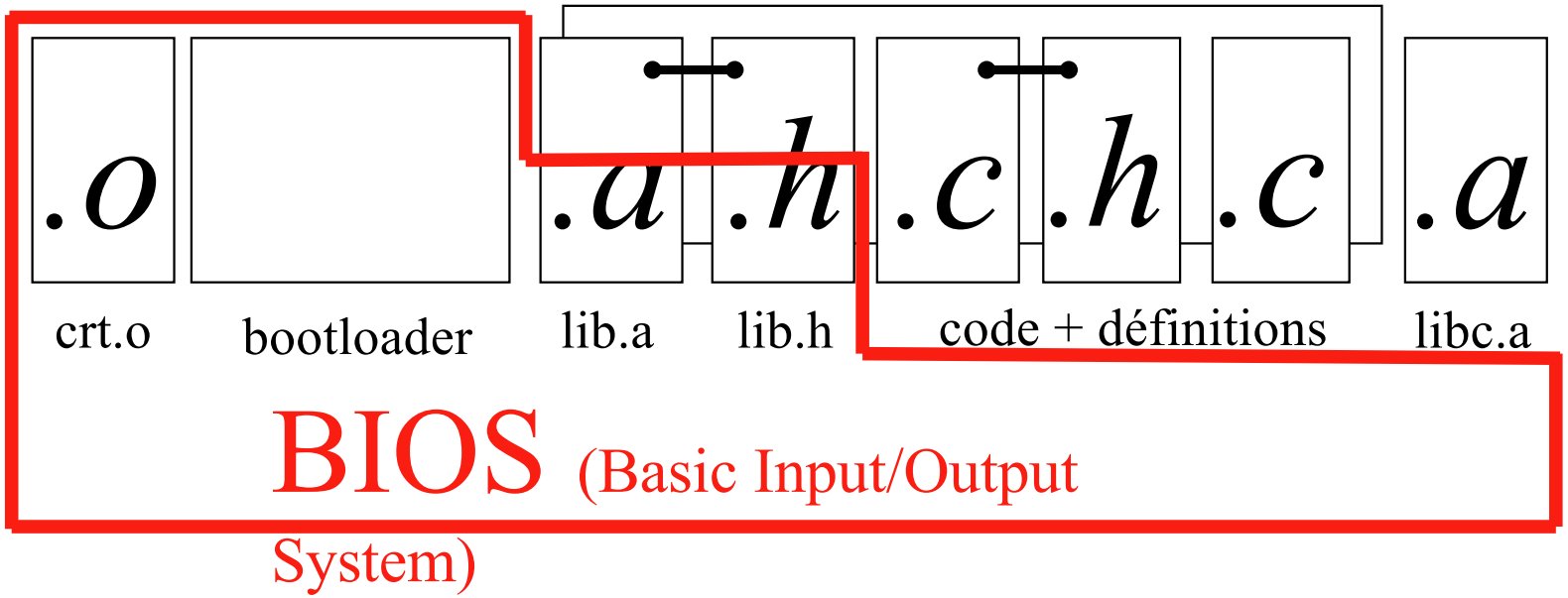
```
#include <stdio.h>

main()
   {
   printf("Hello World\n");
   return (0);
   }
```

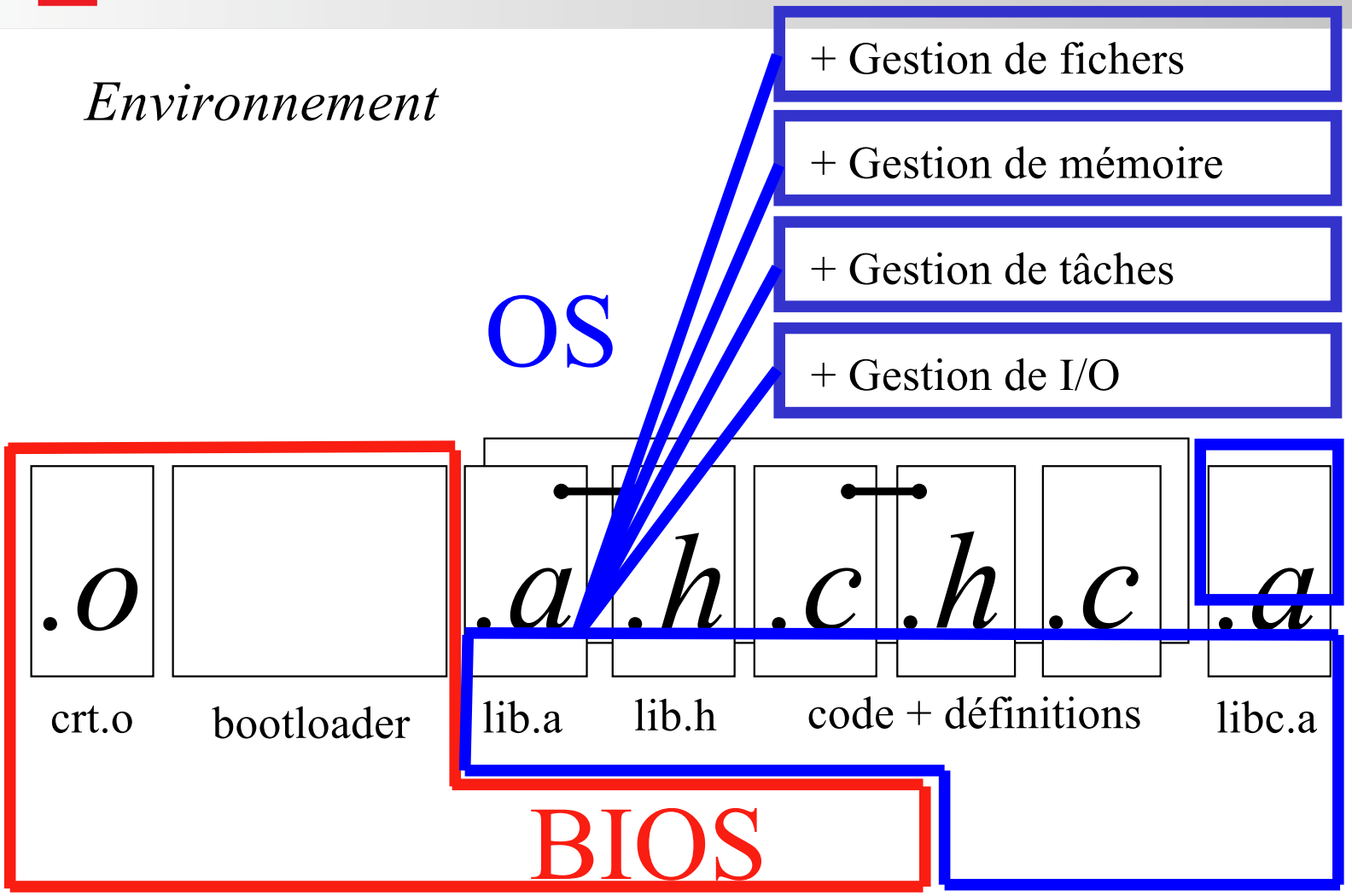


«Hello world»



Environnement




Environnement
+ Gestion de fichers
+ Gestion de mémoire
+ Gestion de tâches
+ Gestion de I/O
OS
.o
.a
.h
.c
.h
.c
.a
crt.o
bootloader
lib.a
lib.h
code + définitions
libc.a
BIOS

Abstraction matérielle: ou comment utilizer le compilateur pour le bas niveau

Francesco Mondada
IEM - STI - EPFL

Introduction

Les outils offerts par le compilateur permettent une abstraction qui facilite l'écriture, la lecture et donc le debug.

Certains de ces outils ne sont pas spécifiques du compilateur C, mais sont plus complets dans un compilateur que dans un simple assembleur.

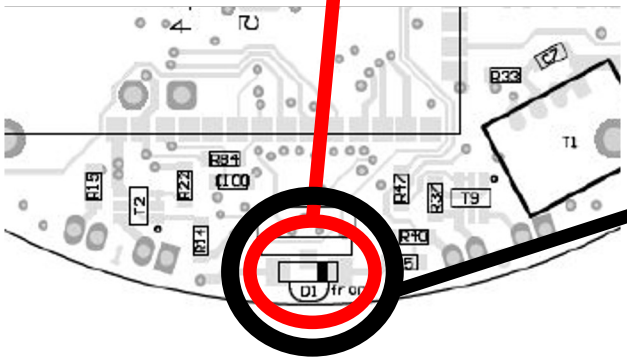
Introduction

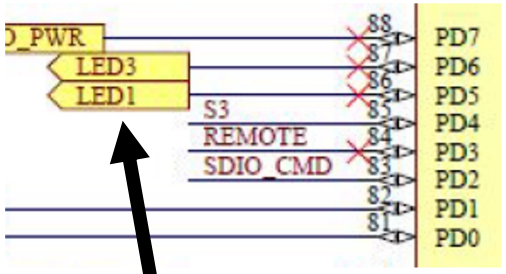
La facilité d'écriture (et de lecture) d'un code source depend énormément de la structure mise en place autour du code. Cette structure peut se faire à divers niveaux, du nom des variables jusqu'à des librairies de fonctions.

Dans ce cadre il est important de comprendre:

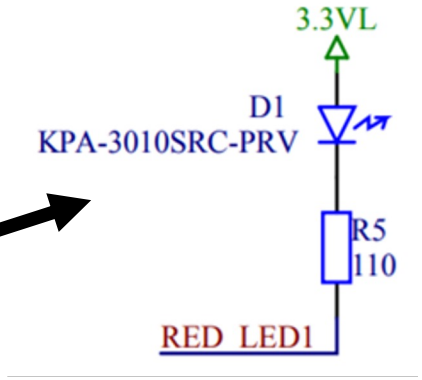
- Les mécanismes de redéfinition de noms (`#define`)
- L'utilité de structures (par exemple) pour faciliter l'écriture
- L'utilisation et la structuration de fonctions de bas niveau
- Le partage de ressources (timers etc)

Ressources matérielles

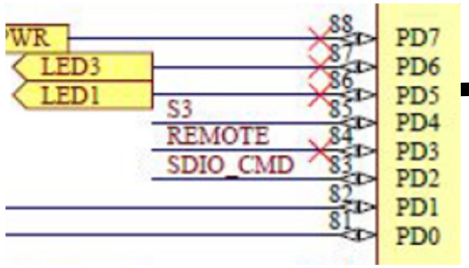




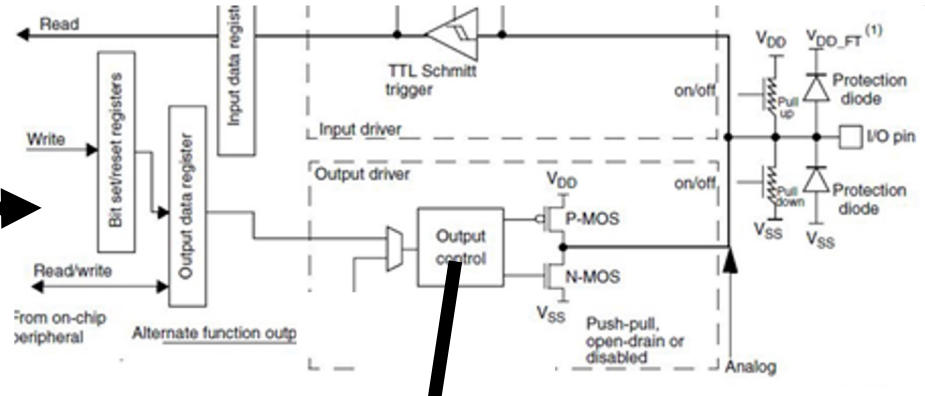
PD5



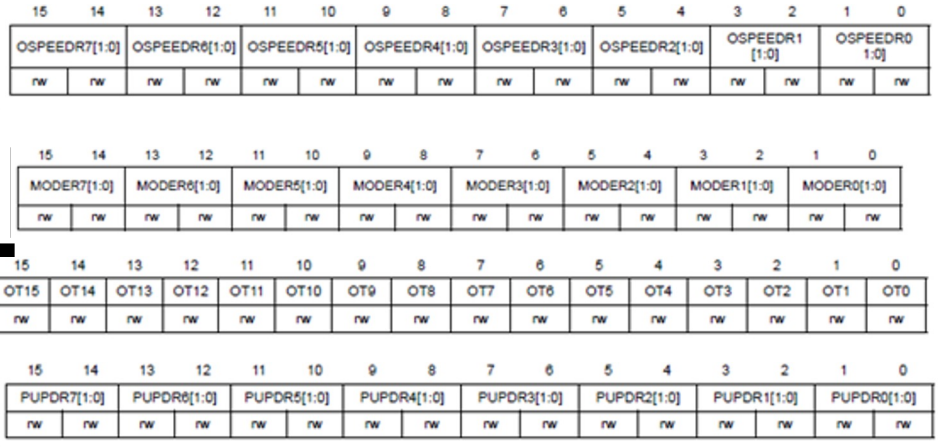
Ressources matérielles



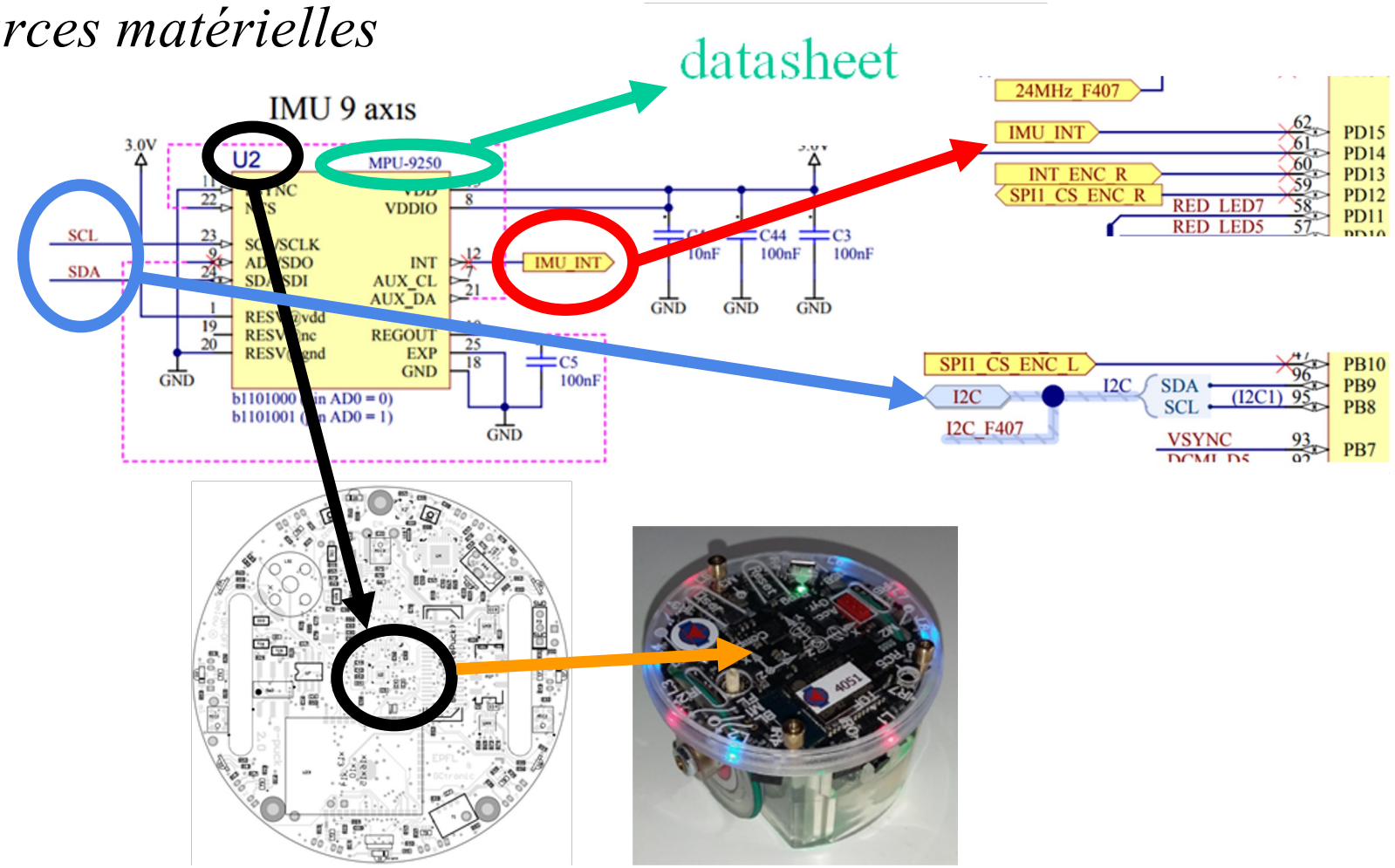
PD5




```
5  void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin)
6  {
7    // Output type open-drain : OTy = 1
8    port->OTYPER |= (1 << pin);
9
10   // Output data low : ODRy = 0
11   port->ODR &= ~(1 << pin);
12
13   // Floating, no pull-up/down : PUPDRy = 00
14   port->PUPDR &= ~(3 << (pin * 2));
15
16   // Output speed highest : OSPEEDRy = 11
17   port->OSPEEDR |= (3 << (pin * 2));
18
19   // Output mode : MODERy = 01
20   port->MODER = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
21 }
```



Ressources matérielles



Abstraction au niveau de nom de variables

Lecture de code:

- Que fait l'instruction port -> BSRR = (1 << (pin + 16))?
- Que fait l'instruction gpio_set(no_led) ?

Ecriture de code:

- Pour écrire port -> BSRR = (1 << (pin + 16)) il faut:
  - ✓ trouver sur quel port et quel bit est connecté la LED
  - ✓ voir dans quel registre on doit faire quoi
- L'écriture gpio_set (ou led_set..) est simple et intuitive

Couches d'abstraction:

8.4.7 GPIO port bit set/reset register (GPIOx_BSRR) (x = A..I/J/K)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

`stm32f407xx.h`

```
typedef struct
{
  __IO uint32_t MODER;
  __IO uint32_t OTYPER;
  __IO uint32_t OSPEEDR;
  __IO uint32_t PUPDR;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t LCKR;
  __IO uint32_t AFR[2];
} GPIO_TypeDef;
```

VOTRE CODE

```
void gpio_set(GPIO_TypeDef *port,unsigned int pin)
{
                port ->BSRR = (1 << (pin + 16));
}
```

`gpio.c`

Couches d'abstraction: fonctions

```
while(1)
{
  LED0 = 1;
  for(i=0;i<300000;i++)
    asm("nop");
  LED0 = 0;
  LED1 = 1;
  for(i=0;i<300000;i++)
    asm("nop");
  LED1 = 0;
  LED2 = 1;
  for(i=0;i<300000;i++)
    asm("nop");
  LED2 = 0;
}
```

```
void gpio_set(led_no){
…
}

void gpio_clear(led_no){
…
}

void wait_nop(number_of_nops){
…
}

while(1){
  for(led=0;led<3;led++)  {
    gpio_set(led);
    wait_nop(300000);
    gpio_clear(led);
  }
}
```

Gestion d'interruptions: abstraction

```
#include <stm32f4xx.h>
#include <gpio.h>
#include <main.h>



#define TIMER_CLOCK 84000000            /* APB1 clock */
#define PRESCALER_TIM7 (TIMER_CLOCK/10000) /* timer frequency: 10kHz */
#define COUNTER_MAX_TIM7 10000       /* timer max counter 1 sec */
```

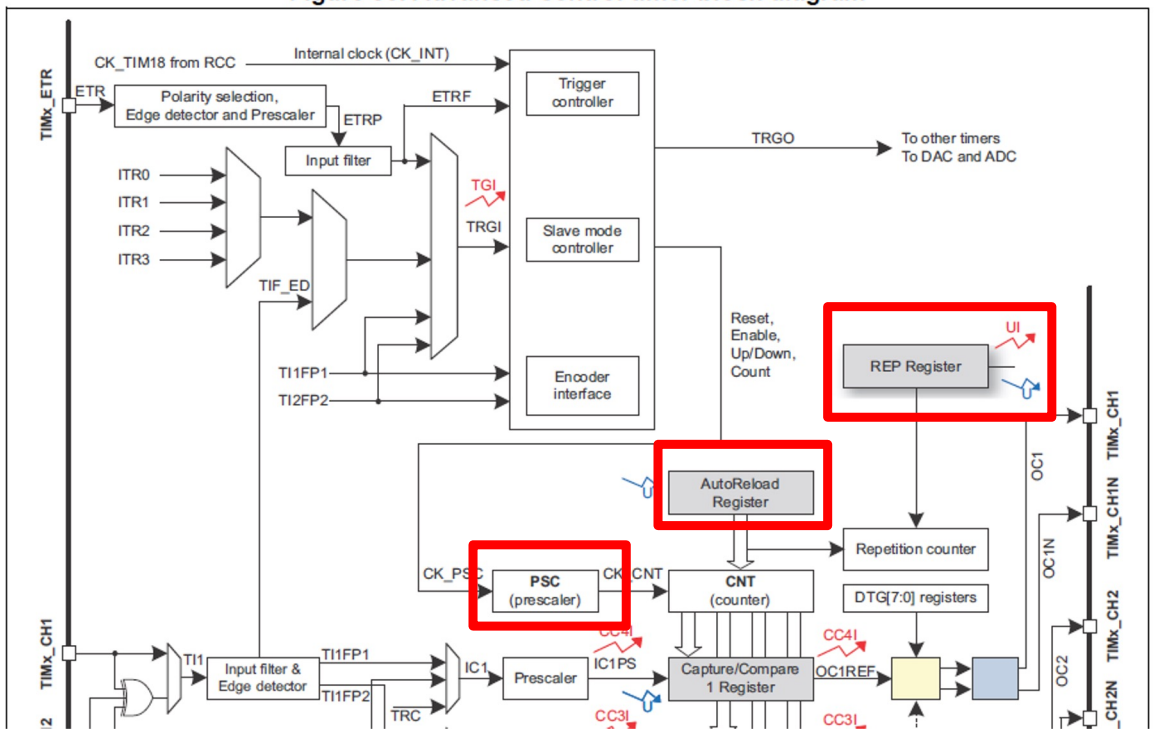
Gestion de temps par interruption

```
void timer7_start(void)
{
    // Enable TIM7 clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM7EN;

    // Enable TIM7 interrupt vector
    NVIC_EnableIRQ(TIM7_IRQn);

    // Configure TIM7
    TIM7->PSC = PRESCALER_TIM7;
    TIM7->ARR = COUNTER_MAX_TIM7 - 1;// note: timer reload takes 1 cycle, thus -1
    TIM7->DIER |= TIM_DIER_UIE;  // enable update interrupt
    TIM7->CR1 |= TIM_CR1_CEN;    // enable timer
}

void TIM7_IRQHandler(void)
{
    // Clear interrupt flag
    TIM7->SR &= ~TIM_SR_UIF;

    /* toggle LED */
    gpio_toggle(LED_USED);
}
```

Initialisation du TIMER5

```
// Configure TIM7
TIM7->PSC = PRESCALER_TIM7;
TIM7->ARR = COUNTER_MAX_TIM7 - 1;
TIM7->DIER |= TIM_DIER_UIE;
```

Figure 86. Advanced-control timer block diagram



TP de cette semaine

(suite TPIntro si besoin!!!!!)

Exercices de programmation en C
Etude de la structure du compilateur C

Utilisation de TIMER.
> Programmation d'un PWM pour régler l'intensité d'une LED

>Programmation du contrôle de moteurs pas à pas en C

1 / 1

<<< List of responses | Print this Response

Respondent: **- Anonymous -**

Evaluation du cours et du TP

Ceci est un questionnaire anonyme, donnez votre avis franchement, merci.

1 * Donnez votre évaluation du cours de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	○	◉	○
Forme	○	○	○	○	◉	○

2 * Donnez votre évaluation du TP de la semaine (1 nul, 6=excellent)

	1	2	3	4	5	6
Contenu	○	○	○	◉	○	○
Forme	○	○	◉	○	○	○

3 Vos commentaires en détail (si nécessaire):

Les installations de logiciels sont toujours longs et parfois compliquées : apparitions de bugs inconnus, etc... Cependant, les installations pourraient être simplifiées avec des consignes beaucoup plus claires, En effet, le site possédant toutes les marches à suivre est un vrai labyrinthe. On s'y perd, il donne des infos dans le désordre, ouvre plein de liens différents qui se retrouvent à différents endroits. Mettre à jour et faire un pdf avec les étapes dans l'ordre chronologique serait plus efficace à mon avis.

QUIZ

Vérification connaissances cours 1

Quiz Settings Questions Results Question bank More

Ce quiz vous permet de vérifier vos connaissances acquises lors du premier cours.

Continue the last preview

Grading method: First attempt

Attempts: 23


Just do git
Introduction and basic commands
Come discover git, its advantages and how to start
Thursday, March 6 18:15 CM 1 105
Intermediate commands
Come improve your knowledge and speed up your usage
Thursday, March 13 18:15 CM 1 105
Round table for advanced commands
Come discuss your favorite commands and learn some new ones. Being familiar with git is recommended
Thursday, March 20 18:15 CM 1 100

Résumé 1 à disposition!

Avez-vous utilisé le GPT mis à disposition? Des retours sont bienvenus. Attention à l'usage d'autres GPTs (acemate):

8. Intellectual Property Rights

8.1 Users may only upload materials they have created themselves or for which they have been explicitly authorized to upload by the rights holder.

8.2 By uploading materials, users grant acemate a transferable, non-exclusive right to use the respective content. These usage rights are neither limited in time nor space. acemate is entitled to reproduce, distribute, and offer these contents to other users. The right also includes publication. acemate is entitled to edit the uploaded materials to the extent necessary for presentation and provision on the platform. acemate is entitled to convert the content into other formats and to watermark and/or logo the uploaded materials to prevent reproduction or other misuse.

8.3 acemate may remove individual or all contents uploaded by users and individual or all links set by users without giving reasons. This cannot be interpreted as acemate reviewing the uploaded content or set links.

8.4 If users infringe the intellectual property rights of others, acemate has the right to block their account.