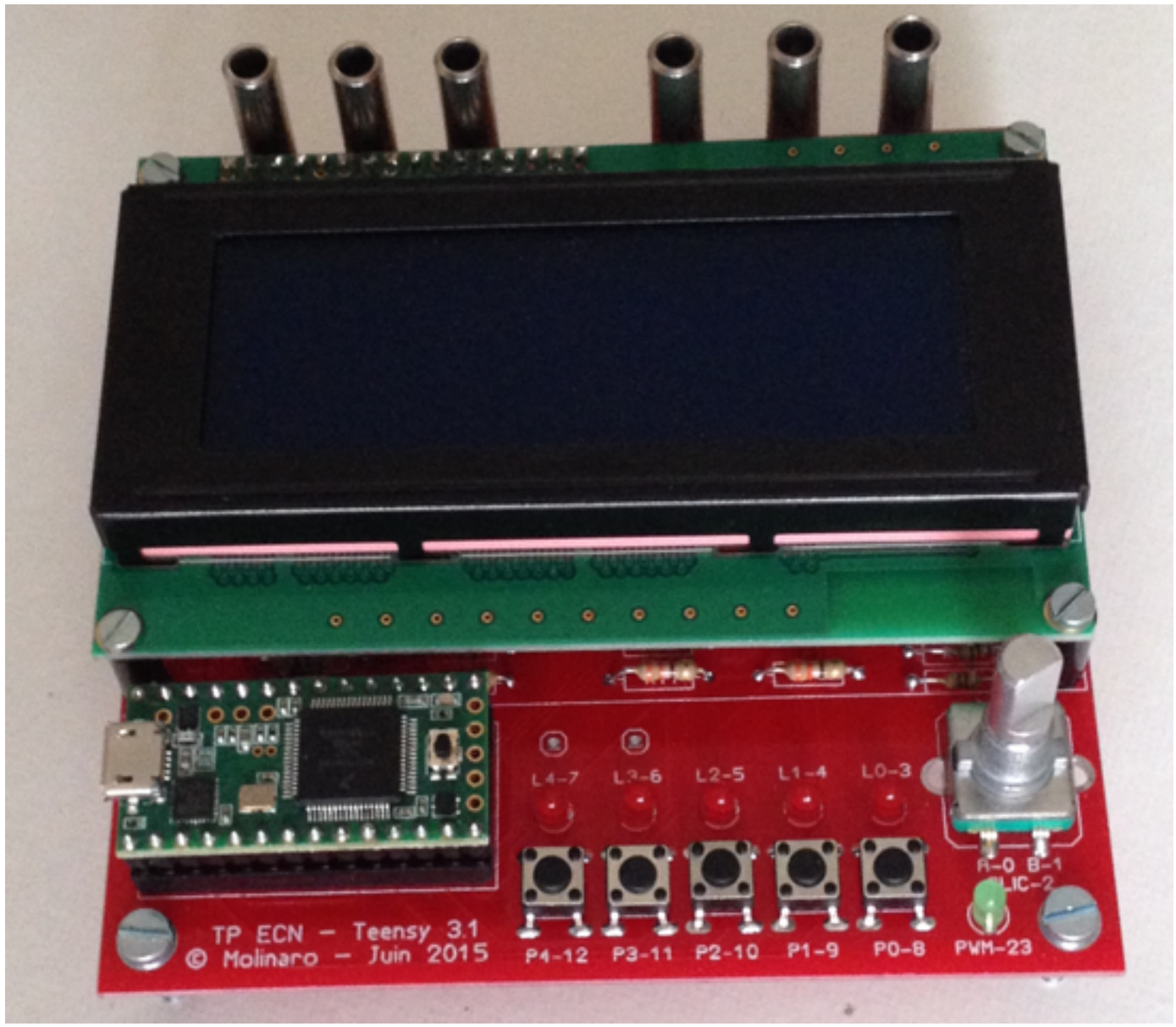
Temps Réel


L4-7
L3-6
L2-5
L1-4
L0-3
R-0 B-1
LIC-2
TP ECN - Teensy 3.1
© Molinaro - Juin 2015
P4-12
P3-11
P2-10
P1-9
P0-8
PWM-23

But de cette partie

Objectif :

- *établir une base logicielle solide pour écrire des programmes séquentiels.*

Problèmes à résoudre :

- définition des registres de contrôle du micro-contrôleur ;
- configuration du micro-contrôleur ;
- initialisation des variables globales implicitement initialisées ;
- initialisation des variables globales explicitement initialisées ;
- séparation entre initialisation et code applicatif.

Travail à faire :

- écrire un programme *blinkled* utilisant cette base.

I — Variables globales implicitement initialisées

Déclaration en C

En C, une variable globale est une variable déclarée en dehors des fonctions.

Elle est implicitement initialisée si sa déclaration ne cite pas de valeur initiale.

```
int gVariable1 ;
char gVariable2 ;
```

En C, une telle variable est initialisée au démarrage à une valeur correspondant à des zéros binaires.

Le compilateur C place ces variables dans des sections `.bss` ou `.bss.*` (« * » représentant toute chaîne de caractères).

Il faut donc que le programme initialise à zéro les sections `.bss` et `.bss.*`.

Définition de la section .bss dans le script d'édition de liens

internal-flash.ld

```
/*-----------------------------------------------------*/
/*       BSS (uninitialized data)                      */
/*-----------------------------------------------------*/

SECTIONS {
  .bss : {
    . = ALIGN(4);
    __bss_start = . ;
    * (.bss.*) ;
    * (.bss) ;
    * (COMMON) ;
    . = ALIGN(4);
    __bss_end = . ;
  } > sram_u
}
```

Les deux symboles `__bss_start` et `__bss_end` représentent le début et la fin des variables globales initialisées implicitement. Les instructions « `.=ALIGN(4);` » assurent que ces symboles ont une valeur multiple de 4. Enfin, « `> sram_u` » signifie que l'ensemble est logé dans la ram `sram_u`.

Comment effacer la section `.bss` au démarrage

`startup-sequential-no-systick.c`

```
extern uint32_t __bss_start ;
extern uint32_t __bss_end ;
uint32_t * p = & __bss_start ;
while (p != & __bss_end) {
  * p = 0 ;
  p ++ ;
}
```

Ce code efface la zone commençant à `__bss_start` et finissant avant `__bss_end`. Comme ces symboles ne sont pas connus au moment de la compilation, ils doivent être déclarés `extern` (c'est-à-dire non résolus). C'est l'édition de liens qui résoud ces symboles.

2 — Variables globales explicitement initialisées

Déclaration en C

En C, une variable globale est une variable déclarée en dehors des fonctions.

Elle est explicitement initialisée si sa déclaration cite une valeur.

```
int gVariable1 = 10 ;
char gVariable2 = 'A' ;
```

Le compilateur C place ces variables dans des sections `.data` ou `.data.*init*` (« * » représentant toute chaîne de caractères).

Le problème est plus compliqué :

- les valeurs initiales doivent être placées en RAM ;
- or le contenu initial de la RAM est aléatoire.

La solution :

- les valeurs initiales sont placées dans la Flash ;
- au démarrage, copie des valeurs en Flash dans la RAM.

Définition de la section .data dans le script d'édition de liens

internal-flash.ld

```
/*-----------------------------------------------*/
/*                                               */
/*         Data (initialized data)               */
/*                                               */
/*-----------------------------------------------*/

SECTIONS {
  .data : {
    FILL (0xFF)
    . = ALIGN (4) ;
    __data_start = . ;
    * (.data.*init*) ;
    * (.data*) ;
    . = ALIGN (4) ;
    __data_end = . ;
  } > sram_u AT > flash
}

/*-----------------------------------------------*/

__data_load_start = LOADADDR (.data) ;
__data_load_end   = LOADADDR (.data) + SIZEOF (.data) ;
```

Les deux symboles __data_start et __data_end représentent le début et la fin des valeurs initiales des variables globales initialisées explicitement. Ces valeurs logées dans la flash (AT > flash).

La taille correspondante est réservée dans la RAM (> sram_u), à partir du symbole __data_load_start jusqu'à __data_load_end.

Comment recopier la section .data au démarrage

startup-sequential-no-systick.c

```
extern uint32_t __data_start ;
extern uint32_t __data_end ;
extern uint32_t __data_load_start ;
uint32_t * pSrc = & __data_load_start ;
uint32_t * pDest = & __data_start ;
while (pDest != & __data_end) {
  * pDest = * pSrc ;
  pDest ++ ;
  pSrc ++ ;
}
```

Ce code recopie la zone commençant à `__data_start` et finissant avant `__data_end`. dans la zone commençant à `__data_load_start` . Comme ces symboles ne sont pas connus au moment de la compilation, ils doivent être déclarés `extern` (c'est-à-dire non résolus). C'est l'édition de liens qui résoud ces symboles.

3 — Registres de contrôle du micro-contrôleur

Problématique

Dans le premier programme, les registres de contrôle étaient définis dans le fichier `main.c` par des définitions :

```
#define PORTC_PCR5   (*((volatile uint32_t *) 0x4004B014))
#define GPIOC_PDDR   (*((volatile uint32_t *) 0x400FF094))
#define GPIOC_PSOR   (*((volatile uint32_t *) 0x400FF084))
#define GPIOC_PCOR   (*((volatile uint32_t *) 0x400FF088))
#define SIM_SCGC5    (*((volatile uint32_t *) 0x40048038))
#define WDOG_STCTRLH (*((volatile uint16_t *) 0x40052000))
#define WDOG_UNLOCK  (*((volatile uint16_t *) 0x4005200E))
```

À partir de ce programme, toutes les définitions des registres de contrôle sont regroupées dans le fichier `mk20dx256.h`.

4 — Configuration du micro-contrôleur

Problématique

Le micro-contrôleur démarre sur une horloge interne de 8 MHz environ, avec des réglages d'accès en Flash et en RAM non optimisés.

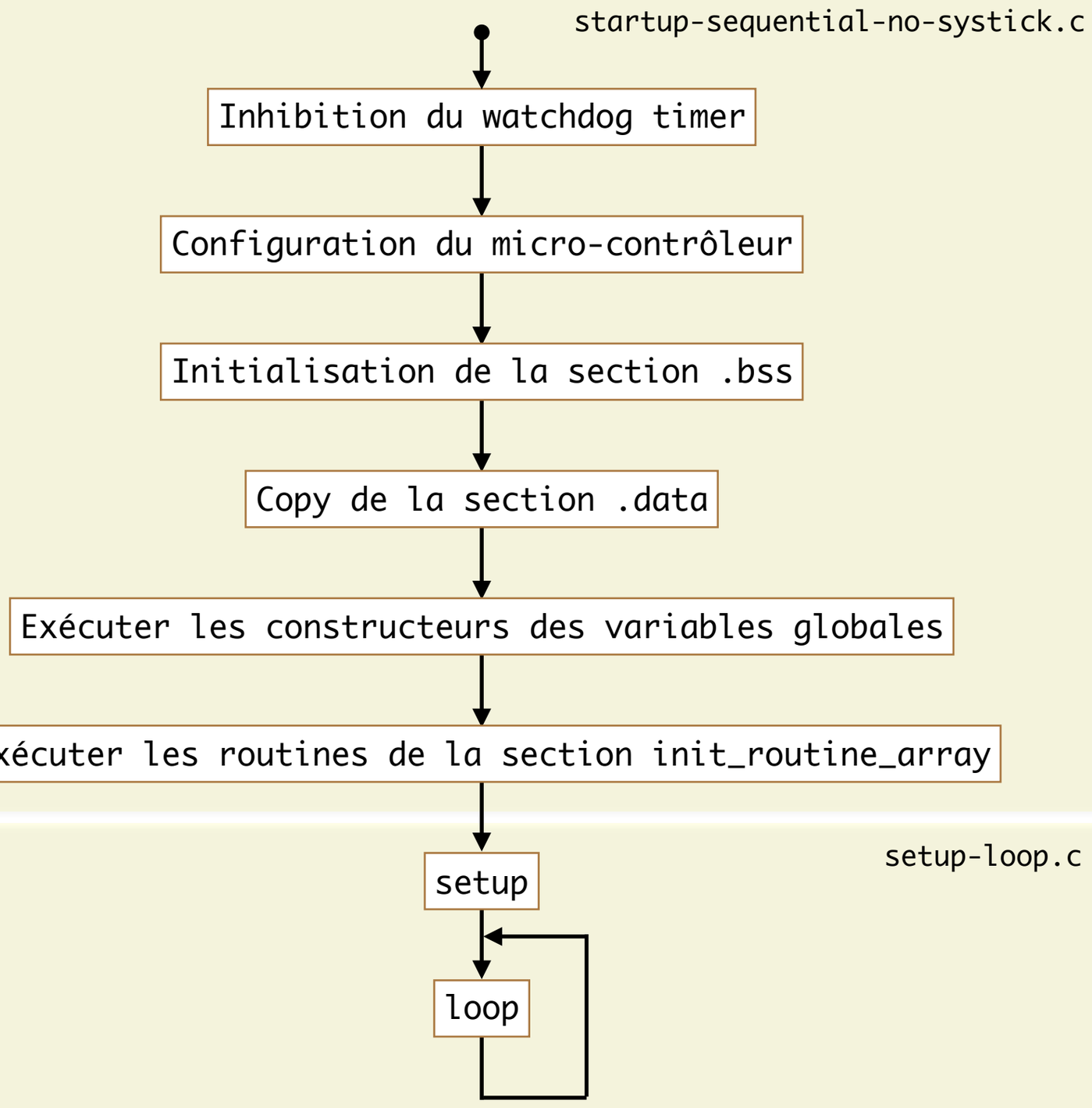
Le processeur peut fonctionner jusqu'à 96 MHz.

De plus, les périphériques sont par défaut inactivés (pour économiser l'énergie).

Le code C correspondant n'est pas décrit (au delà de l'objectif de ce cours).

5 — Séparation entre initialisation et applicatif

Structure du programme



Vous n'avez pas à modifier ce fichier.

Fichier à modifier.

6 — Travail à faire

Travail à faire

Écrire un programme *blinkled* utilisant cette base. Pour cela :

- dupliquer le premier programme et le renommer ;
- récupérer sur le serveur pédagogique les fichiers `mk20dx256.h`, `setup-loop.h`, `setup-loop.c` et `startup-sequential-no-systick.c` ;
- modifier le fichier `makefile.json` en conséquence ;
- compléter le fichier `setup-loop.c`.

La led à faire clignoter est la led *Teensy*. Pour configurer le port correspondant, pour allumer et éteindre la led, voir le programme précédent.