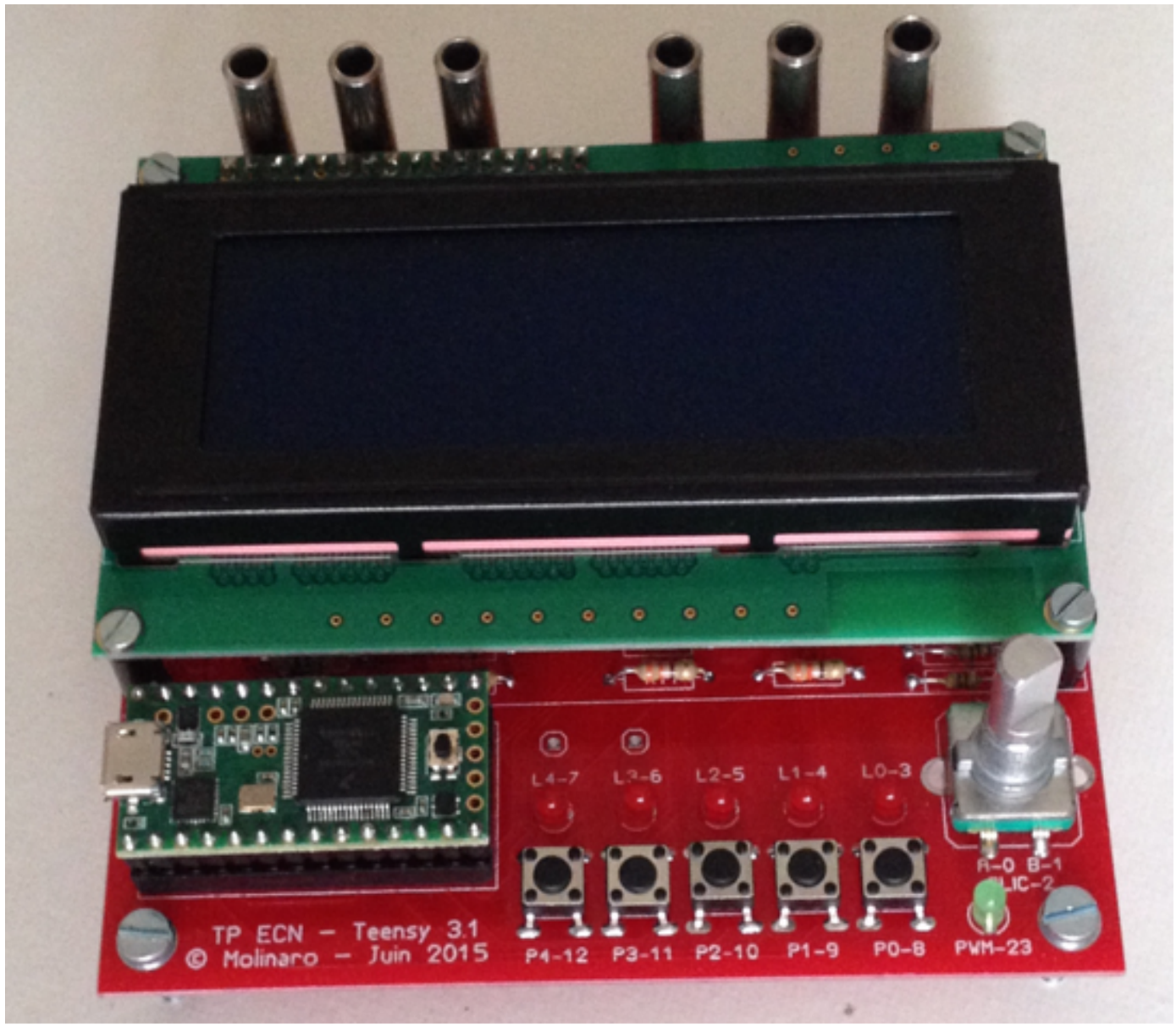
Temps Réel



Programme `04-leds`

Pierre Molinaro
Janvier 2016

But de cette partie

Objectif :

- *disposer de routines permettant d'allumer et d'éteindre les leds* `L0` *à* `L4`.

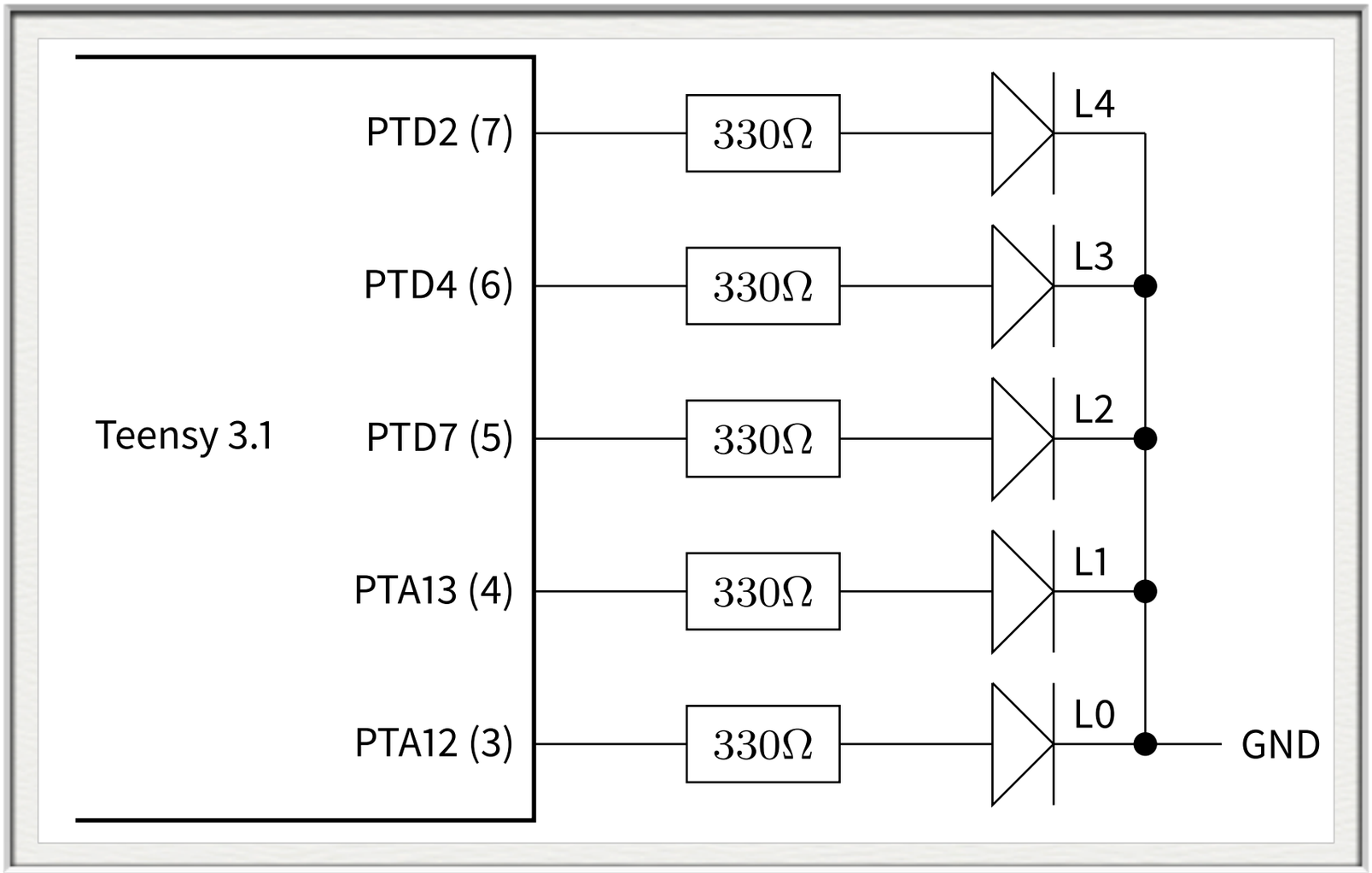
Problèmes à résoudre :

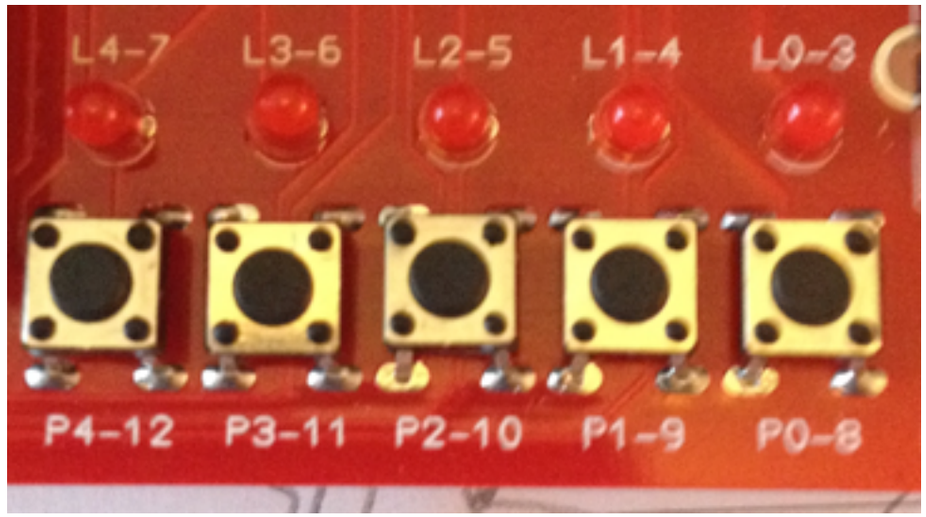
- configuration des ports du micro-contrôleur ;
- comment exécuter la routine d'initialisation ;
- écrire les routines `ledOn` et `ledOff`.

Travail à faire :

- réaliser un chenillard qui allume successivement les leds `L0` à `L4`.

Connexion des leds





Port n°7 en sortie logique au niveau bas : le micro-contrôleur impose une tension proche de zéro Volt, la led `L4` est éteinte.

Port n°7 en sortie logique au niveau haut : le micro-contrôleur impose une tension proche de 3,3V, la led `L4` est allumée.

Initialisation : routine `setupLeds`

leds.c

```
static void setupLeds (void) {
//--- Led L0 : PTA12
  PORTA_PCR12 = (1 << 8) ;
  GPIOA_PDDR |= (1 << 12) ;
//--- Led L1 : PTA13
  PORTA_PCR13 = (1 << 8) ;
  GPIOA_PDDR |= (1 << 13) ;
//--- Led L2 : PTD13
  PORTD_PCR7 = (1 << 8) ;
  GPIOD_PDDR |= (1 << 7) ;
//--- Led L3 : PTD4
  PORTD_PCR4 = (1 << 8) ;
  GPIOD_PDDR |= (1 << 4) ;
//--- Led L4 : PTD2
  PORTD_PCR2 = (1 << 8) ;
  GPIOD_PDDR |= (1 << 2) ;
//--- Port PTC5 en sortie : led Teensy
  PORTC_PCR5 = (1 << 8) ;
  GPIOC_PDDR |= (1 << 5) ;
}
```

Cette routine programme les ports correspondant aux cinq leds `L0`, `L1`, …, `L4` en sortie, ainsi que le port correspondant à led Teensy.

Désignation des leds

leds.h

```
static const uint32_t LED_L0 = (1 << 0) ;
static const uint32_t LED_L1 = (1 << 1) ;
static const uint32_t LED_L2 = (1 << 2) ;
static const uint32_t LED_L3 = (1 << 3) ;
static const uint32_t LED_L4 = (1 << 4) ;
```

Les cinq constantes `LED_L0`, `LED_L1`, … correspondent aux cinq leds.

La led *Teensy* est particulière, dans la suite elle sera utilisée pour rendre compte de l'activité processeur. Elle sera donc allumée et éteinte à partir des routines système, et l'utilisateur n'y aura plus accès.

Allumer des leds

leds.c

```
void ledOn (const uint32_t inLeds) {
//--- Led L0
  if ((inLeds & LED_L0) != 0) {
    GPIOA_PSOR = 1 << 12 ;
  }
//--- Led L1
  if ((inLeds & LED_L1) != 0) {
    GPIOA_PSOR = 1 << 13 ;
  }
//--- Led L2
  if ((inLeds & LED_L2) != 0) {
    GPIOD_PSOR = 1 << 7 ;
  }
//--- Led L3
  if ((inLeds & LED_L3) != 0) {
    GPIOD_PSOR = 1 << 4 ;
  }
//--- Led L4
  if ((inLeds & LED_L4) != 0) {
    GPIOD_PSOR = 1 << 2 ;
  }
}
```

Cette routine permet d'allumer plusieurs leds simultanément.

```
ledOn (LED_L0) ;
```

Allume la led L0.

```
ledOn (LED_L1 | LED_L4) ;
```

Allume les leds L0 et L4.

```
ledOn (0) ;
```

Aucun effet.

Éteindre des leds

leds.c

```
void ledOff (const uint32_t inLeds) {
//--- Led L0
  if ((inLeds & LED_L0) != 0) {
    GPIOA_PCOR = 1 << 12 ;
  }
//--- Led L1
  if ((inLeds & LED_L1) != 0) {
    GPIOA_PCOR = 1 << 13 ;
  }
//--- Led L2
  if ((inLeds & LED_L2) != 0) {
    GPIOD_PCOR = 1 << 7 ;
  }
//--- Led L3
  if ((inLeds & LED_L3) != 0) {
    GPIOD_PCOR = 1 << 4 ;
  }
//--- Led L4
  if ((inLeds & LED_L4) != 0) {
    GPIOD_PCOR = 1 << 2 ;
  }
}
```

Cette routine permet d'éteindre plusieurs leds simultanément.

```
ledOff (LED_L0) ;
```

Éteint la led L0.

```
ledOff (LED_L1 | LED_L4) ;
```

Éteint les leds L0 et L4.

```
ledOff (0) ;
```

Aucun effet.

Comment appeler la routine `setupLeds` ?

Une solution : l'appeler au début de la routine `setup`.

Dans un véritable projet, de nombreuses routines peuvent être ainsi appelées. Le danger est d'oublier de les appeler.

Dans ce cours, on va mettre en place une solution plus générale : *il suffit que le fichier fasse partie du projet pour la routine soit automatiquement appelée*. Plus besoin de penser à écrire l'appel dans `setup`, l'exécution au démarrage est automatique.

C'est cette solution qui est présentée dans les pages qui suivent.

Retour sur le script de l'éditeur de liens

internal-flash.ld

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*                                  Code                                   */
/*                                                                         */
/*-------------------------------------------------------------------------*/

SECTIONS {
  .text : {
    FILL(0xff)
    __code_start = . ;
  /*--- Tableau des routines d'initialisation */
    . = ALIGN (4) ;
    __init_routine_array_start = . ;
    KEEP (*(init_routine_array)) ;
    . = ALIGN (4) ;
    __init_routine_array_end = . ;
 /*--- Initialisation des objets globaux C++ */
    …
```

La section `init_routine_array` est bornée par les symboles `__init_routine_array_start` et `__init_routine_array_end`.

Section `init_routine_array` et démarrage

startup-sequential.c

```
//---------6- Exécuter les routines d'initialisation de la section init_routine_array
  extern void (* __init_routine_array_start) (void) ;
  extern void (* __init_routine_array_end) (void) ;
  ptr = & __init_routine_array_start ;
  while (ptr != & __init_routine_array_end) {
    (* ptr) () ;
    ptr ++ ;
  }
```

Le code de démarrage considère que chaque mot de la section `init_routine_array` est l'adresse d'une routine qu'il exécute.

Donc, il suffit de placer l'adresse de la routine `setupLeds` dans la section `init_routine_array`.

Placer l'adresse de `setupLeds` dans la section `init_routine_array` (1/2)

Attention : c'est l'adresse de la routine qu'il faut placer dans la section `init_routine_array`, et non pas la routine elle-même.

Première solution (correcte, mais non retenue) :

```
static void (* r) (void)
  __attribute__ ((section ("init_routine_array")))
  __attribute__ ((unused))
  __attribute__ ((used)) = setupLeds ;
```

On déclare une variable `r`, initialisée à l'adresse de la routine, que l'on place dans la section `init_routine_array`.

L'attribut `unused` ordonne à GCC de ne pas produire de warning si `r` est inutilisé (ce qui est le cas). L'attribut `used` ordonne à GCC de conserver la variable `r`, même si elle est inutilisée (ce qui est le cas).

Référence :

https://gcc.gnu.org/onlinedocs/gcc-4.9.3/gcc/Variable-Attributes.html#Variable-Attributes

Placer l'adresse de `setupLeds` dans la section `init_routine_array` (2/2)

Seconde solution (retenue) : on utilise la macro `MACRO_INIT_ROUTINE`, définie dans le fichier `internal-macros.h` (dans l'archive `04-source.tbz`).

`leds.c`

```
MACRO_INIT_ROUTINE (setupLeds) ;
```

Intérêt : l'écriture est beaucoup plus simple.

Travail à faire

Écrire un programme réalisant un chenillard. Pour cela :

- dupliquer le programme précédent et le renommer ;
- récupérer les fichiers `internal-macros.h`, `leds.h`, `leds.c` sur le serveur pédagogique, archive `04-sources.tbz`.