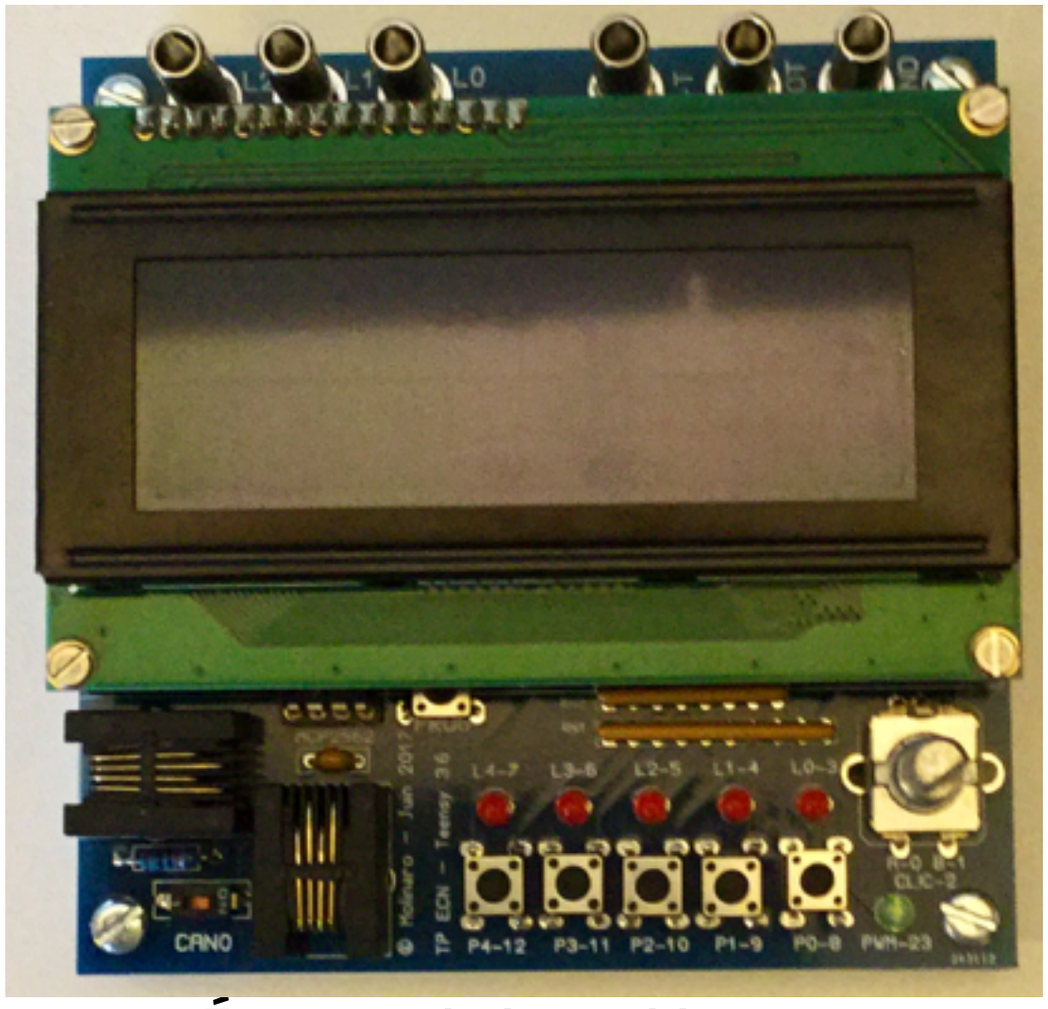
Temps Réel



Étape 05-leds-pushbuttons

Pierre Molinaro

Juin 2018

Description de cette étape

Objectif. Accéder aux leds et aux boutons poussoirs de la carte.

Description. L'archive **05-files.tar.bz2** contient deux fichiers : **teensy-3-6-digital-io.h** et **teensy-3-6-digital-io.cpp**. Ces fichiers définissent des fonctions qui permettent de programmer les ports du micro-contrôleur en entrée ou en sortie logique, d'écrire et de lire ces ports. Pour ceux qui connaissent, ces fonctions sont analogues à celles de l'Arduino, avec de légères différences.

La led Teensy. C'est la led placée sur le module Teensy 3.6, et dupliquée sur la carte de TP. Pour ce cours, on a donné à cette led un rôle particulier : elle rend compte de l'activité processeur. Elle sera donc constamment allumée, jusqu'à ce que les attentes passives de l'exécutif sont implémentées.

Travail à faire. Configurer les ports du micro-contrôleur qui sont associés aux leds et aux boutons poussoir.

Description du fichier `teensy-3-6-digital-io.h` (1/5)

Le type énuméré `DigitalPort` définit les 58 ports du Teensy 3.6 :

```
enum class DigitalPort {
//--- Common with Teensy 3.1 / 3.2
    D0,  // PTB16
    D1,  // PTB17
    D2,  // PTD0
    D3,  // PTA12
    D4,  // PTA13
...........
    D23, // PTC2
//--- Only on Teensy 3.6
    D24, // PTE26
...........
} ;
```

La déclaration **`enum class`** impose de qualifier les constantes : par exemple, pour se référer au port `D0`, il faudra écrire `DigitalPort::D0`.

Description du fichier `teensy-3-6-digital-io.h` (2/5)

Le type énuméré `DigitalMode` définit le mode d'un port :

```
enum class DigitalMode {
    OUTPUT,
    OUTPUT_OPEN_COLLECTOR,
    INPUT,
    INPUT_PULLDOWN,
    INPUT_PULLUP
} ;
```

Mode	Description
`DigitalMode::OUTPUT`	Le port est en sortie : le micro-contrôleur impose une tension soit proche de 0V (sortie à zéro), soit proche de 3,3V (sortie à 1)
`DigitalMode::OUTPUT_OPEN_COLLECTOR`	Le port est en sortie : le micro-contrôleur impose une tension soit proche de 0V (sortie à zéro), soit n'impose rien (sortie à 1)
`DigitalMode::INPUT`	Le port est en entrée : le port présente un haute impédance, et c'est un circuit extérieur qui impose la tension.
`DigitalMode::INPUT_PULLDOWN`	Internement, le port est relié à 0V à travers une résistance de 50 kΩ environ.
`DigitalMode::INPUT_PULLUP`	Internement, le port est relié à 3,3V à travers une résistance de 50 kΩ environ.

CENTRALE NANTES

Description du fichier `teensy-3-6-digital-io.h` (3/5)

Fonction pinMode. Elle permet de configurer un port :

```
void pinMode (const DigitalPort inPort, const DigitalMode inMode) ;
```

Remarquer qu'il y a pas d'annotation de mode, ce qui signifie que cette fonction peut être appelée dans n'importe quel mode.

Par exemple, si l'on veut configurer le port `D0` en entrée, on écrira :

```
pinMode (DigitalPort::D0, DigitalMode::INPUT) ;
```

Description du fichier `teensy-3-6-digital-io.h` (4/5)

Fonction digitalRead. Elle effectue la lecture d'un port :

```
bool digitalRead (const DigitalPort inPort) ;
```

Cette fonction renvoie :

- **true** si la tension du port est proche de 3,3V ;
- **false** si la tension du port est proche de 0V ;
- une valeur **false** ou **true** imprévisible dans les autres cas.

Noter qu'il est valide de lire un port configuré en sortie.

Description du fichier `teensy-3-6-digital-io.h` (5/5)

Fonction digitalWrite. Écriture sur un port :

```
void digitalWrite (const DigitalPort inPort, const bool inValue) ;
```

Si le port est configuré en entrée, cette fonction n'a aucun effet.

Si le port est configuré en *sortie* (`DigitalMode::OUTPUT`) :

- si **inValue** est **false**, le micro-contrôleur impose une tension proche de 0V ;
- si **inValue** est **true**, le micro-contrôleur impose une tension proche de 3,3V.

Si le port est configuré en *sortie collecteur ouvert* (`DigitalMode::OUTPUT_OPEN_COLLECTOR`) :

- si **inValue** est **false**, le micro-contrôleur impose une tension proche de 0V ;
- si **inValue** est **true**, le micro-contrôleur n'impose aucune tension.

Fonction digitalToggle. Complémentation d'un port :

```
void digitalToggle (const DigitalPort inPort) ;
```

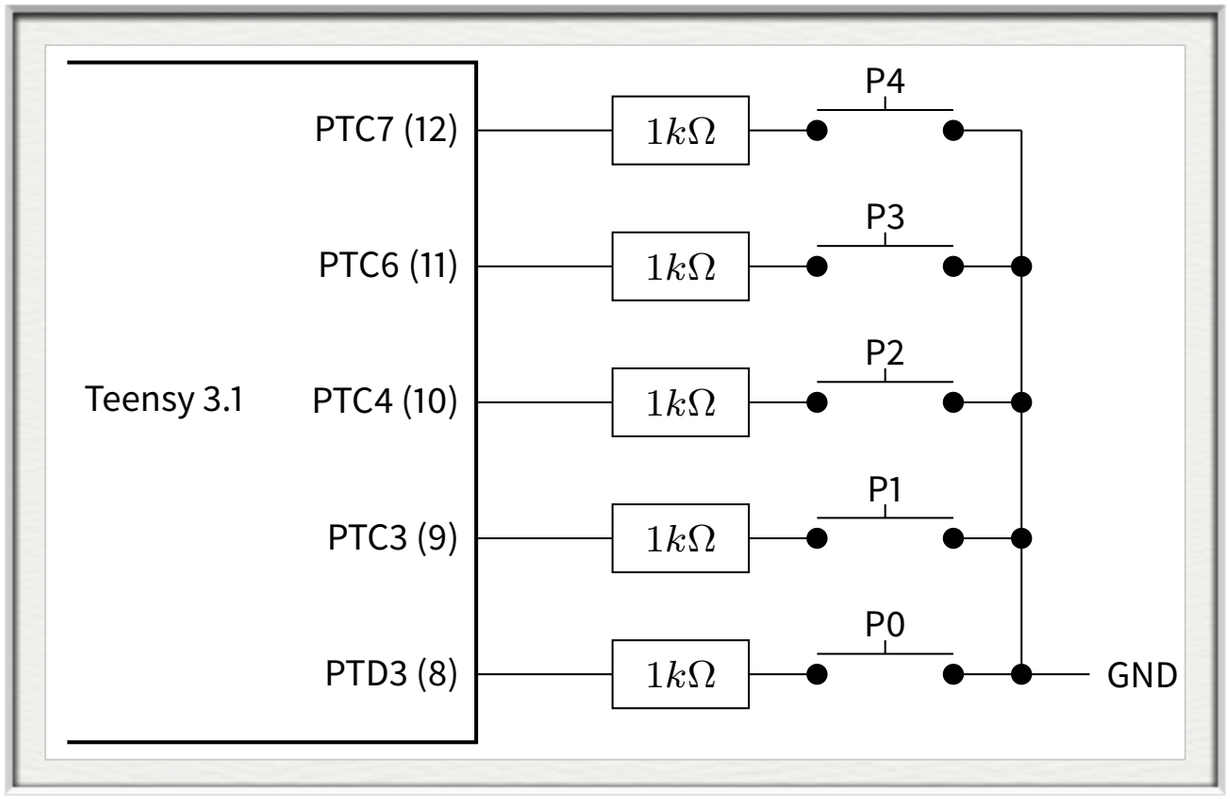
Fonctionnellement :

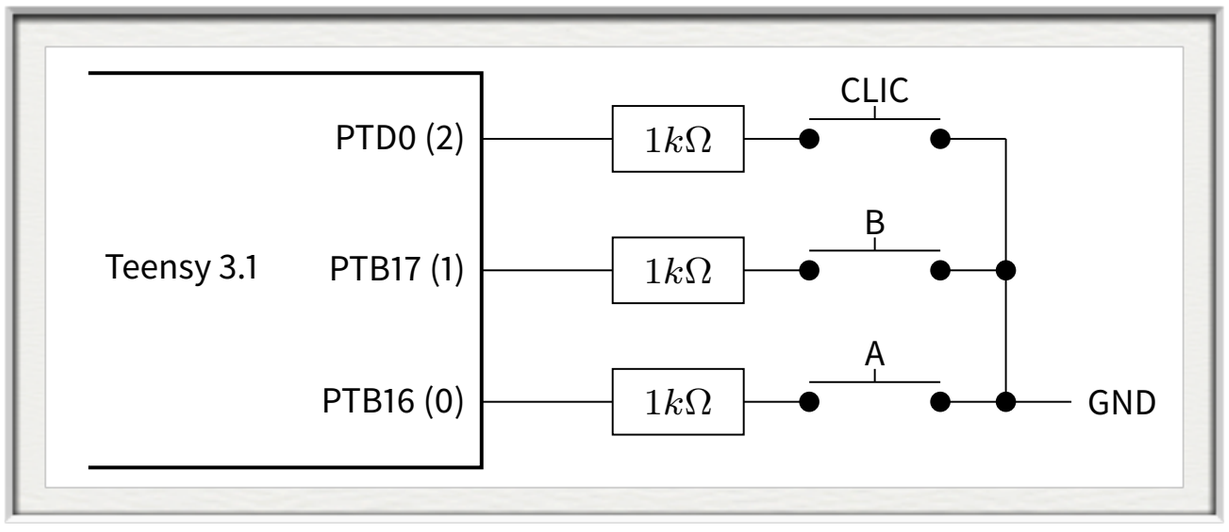
```
digitalToggle (port) ⇔ digitalWrite (port, ! digitalRead (port))
```

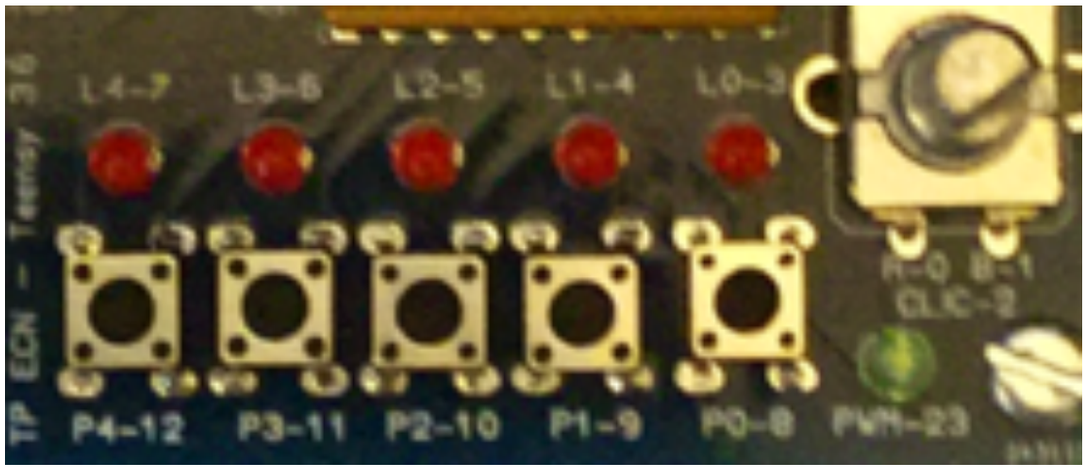
L'intérêt est que `digitalToggle` effectue la complémentation de manière atomique.

CENTRALE NANTES

Poussoirs et encodeur







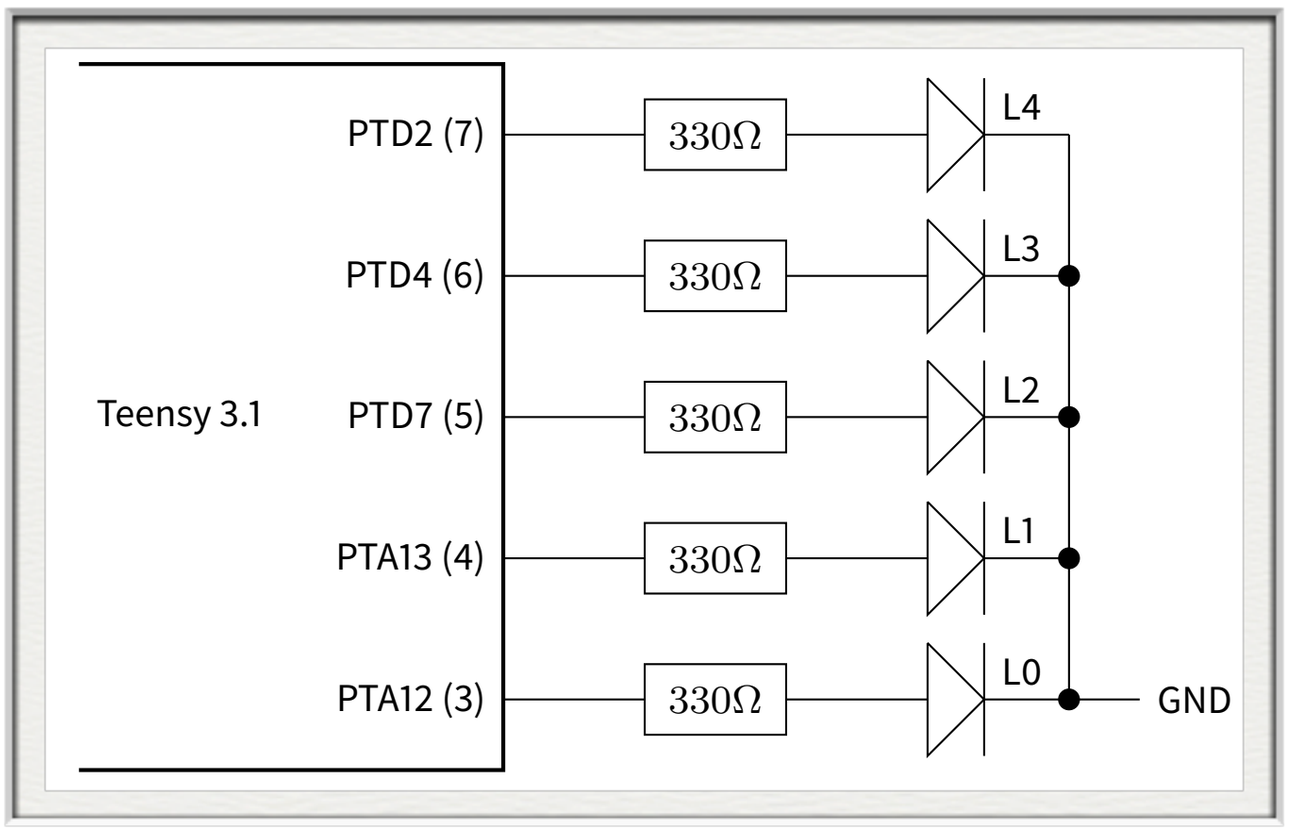
Il faut évidemment configurer les ports correspondants en entrée.

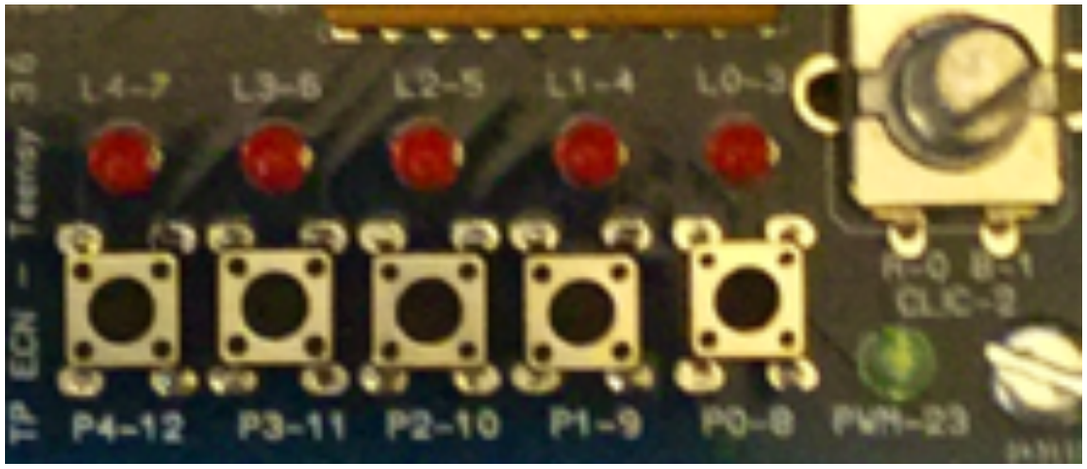
Poussoir appuyé : le port correspondant est à une tension de 0V, l'appel de `digitalRead` retourne **false**.

Poussoir relâché : si le port est configuré en `INPUT`, la valeur retournée est imprévisible ; si le port est configuré en `INPUT_PULLUP`, l'appel de `digitalRead` retourne **true**.

CENTRALE NANTES

Leds





Il faut configurer les ports correspondants en sortie (OUTPUT).

Appel de `digitalWrite` avec un argument valant `false` : le micro-contrôleur impose au port correspondant une tension proche de 0V, la led est éteinte.

Appel de `digitalWrite` avec un argument valant `true` : le micro-contrôleur impose au port correspondant une tension proche de 3,3V, la led est allumée.

CENTRALE NANTES

Travail à faire

Dupliquer le répertoire de l'étape précédente et renommez-le par exemple **05-leds-pushbuttons**.

Ajoutez aux sources les deux fichiers **teensy-3-6-digital-io.h** et **teensy-3-6-digital-io.cpp** contenus dans l'archive **05-files.tar.bz2**.

Écrire un nouveau fichier **dev-board-io.h** qui définit des constantes désignant les ports associés aux leds et boutons poussoirs. En effet, désigner la led `L0` par `DigitalPort::D3` n'est pas très lisible. On déclarera donc dans ce fichier :

```
static const DigitalPort L0_LED = DigitalPort::D3 ;
```

Et de même pour les poussoirs :

```
static const DigitalPort P0_PUSH_BUTTON = DigitalPort::D8 ;
```

Écrire un nouveau fichier **dev-board-io.cpp** qui contient une fonction ***init*** (voir étape précédente), qui configure :

- les ports correspondants aux leds en sortie ;
- les ports correspondants aux poussoirs en entrée ;
- le port de la led Teensy (`DigitalPort::D13`) en sortie et au niveau haut (de façon à allumer la led).

Modifier les fonctions **setup** et **loop** :

- retirer tous les accès directs aux registres de contrôle :
- utiliser digitalWrite ou digitalToggle pour agir sur une led ;
- utiliser digitalRead pour lire la position d'un poussoir, et selon sa position allumer / éteindre une led.