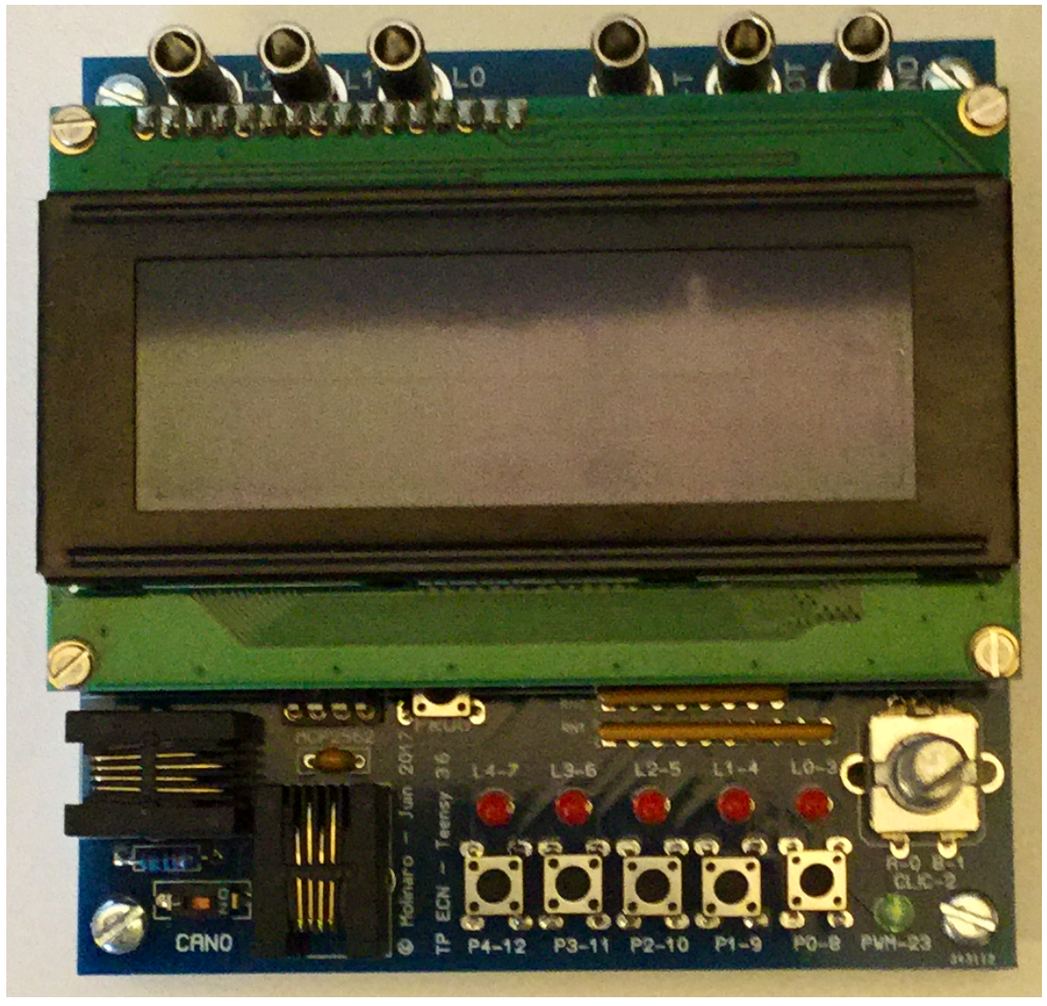
Temps Réel



Étape 02-systick

Description de cette étape

Objectif :

- compter précisément le temps, grâce à un compteur intégré.

Problèmes à résoudre :

- configuration du compteur intégré ;
- écrire une routine d'attente active **`busyWaitDuring`**.

Travail à faire :

- écrire un programme *blinkled* qui appelle **`busyWaitDuring`**. Pour cela, commencer par dupliquer le programme précédent.

CENTRALE NANTES

SysTick

SysTick est un compteur implémenté dans le processeur Cortex-M4.

Ses possibilités sont limitées mais il est très simple à mettre en œuvre.

Par défaut, il est inactivé, c'est-à-dire arrêté.

C'est en fait un décompteur cyclique : `0`, `N`, `N−1`..., `1`, `0`, `N`, ... `N` est une valeur programmable. La période est `N+1`.

Lorsqu'il est rechargé à `N`, il peut être configuré pour déclencher l'interruption n°15 (ce qui n'est pas fait dans cette étape).

CENTRALE NANTES

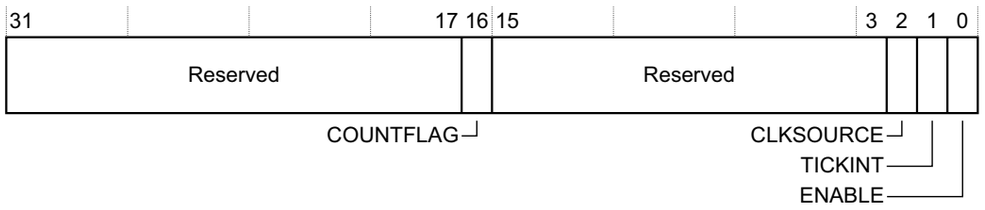
Registres du timer `SysTick`

Source : *ARM®v7-M Architecture Reference Manual*, référence *ARM DDI 0403E.b*.

Address	Name	Type	Reset	Description
0xE000E010	SYST_CSR	RW	0x0000000x[a]	*SysTick Control and Status Register, SYST_CSR*
0xE000E014	SYST_RVR	RW	UNKNOWN	*SysTick Reload Value Register, SYST_RVR* on page B3-678
0xE000E018	SYST_CVR	RW	UNKNOWN	*SysTick Current Value Register, SYST_CVR* on page B3-678
0xE000E01C	SYST_CALIB	RO	IMP DEF	*SysTick Calibration value Register, SYST_CALIB* on page B3-679
0xE000E020-0xE000E0FC	-	-	-	Reserved

a. See register description for information about the reset value of SYST_CSR bit[2]. All other bits reset to 0.

The SYST_CSR bit assignments are:



Bits[31:17] Reserved.

COUNTFLAG, bit[16]

Indicates whether the counter has counted to 0 since the last read of this register:

0 Timer has not counted to 0.

1 Timer has counted to 0.

COUNTFLAG is set to 1 by a count transition from 1 to 0.

COUNTFLAG is cleared to 0 by a software read of this register, and by any write to the Current Value register. Debugger reads do not clear the COUNTFLAG.

This bit is read only.

Bits[15:3] Reserved.

CLKSOURCE, bit[2] Indicates the SysTick clock source:

0 SysTick uses the IMPLEMENTATION DEFINED external reference clock.

1 SysTick uses the processor clock.

If no external clock is provided, this bit reads as 1 and ignores writes.

TICKINT, bit[1] Indicates whether counting to 0 causes the status of the SysTick exception to change to pending:

0 Count to 0 does not affect the SysTick exception status.

1 Count to 0 changes the SysTick exception status to pending.

Changing the value of the counter to 0 by writing zero to the SysTick Current Value register to 0 never changes the status of the SysTick exception.

ENABLE, bit[0] Indicates the enabled status of the SysTick counter:

0 Counter is disabled.

1 Counter is operating.

CENTRALE NANTES

Programmation du timer SysTick

```
void startSystick (void) {
 SYST_RVR = CPU_MHZ * 1000 - 1 ; // Underflow every ms
  SYST_CVR = 0 ;
  SYST_CSR = SYST_CSR_CLKSOURCE | SYST_CSR_ENABLE ;
}
```

La séquence de configuration du SysTick comporte trois instructions. Les registres de contrôle de SysTick sont déclarés dans **cortex-m4-control-registers.h**.

```
SYST_RVR = CPU_MHZ * 1000 - 1 ; // Underflow every ms
```

Cette instruction fixe la période du compteur SysTick à `CPU_MHZ * 1000`. L'horloge de SysTick est l'horloge processeur. Dans le fichier **makefile.json**, le paramètre **CPU-MHZ** permet de choisir cette fréquence, qui apparaît dans le fichier engendré **zSOURCES/base.h** sous le nom **CPU_MHZ**. Ainsi, le compteur SysTick est rechargé toutes les millisecondes, indépendamment de la fréquence du processeur.

```
SYST_CVR = 0 ;
```

Met à zéro la valeur courant du timer SysTick.

```
SYST_CSR = SYST_CSR_CLKSOURCE | SYST_CSR_ENABLE ;
```

Configure et démarre le timer :

- `SYST_CSR_CLKSOURCE`, l'horloge du timer est l'horloge processeur (certains micro-contrôleurs peuvent sélectionner une autre horloge) ;
- `SYST_CSR_ENABLE`, le compteur est activé.

CENTRALE NANTES

Fonction `busyWaitDuring`

La fonction **busyWaitDuring** attend que le nombre de milli-secondes indiquées en argument soient écoulées.

L'attente est qualifiée **active** car le processeur est immobilisé durant l'attente du délai.

L'indicateur booléen **`COUNTFLAG`** (bit n°16 du registre **`SYST_CSR`**) est mis à 1 par le matériel à chaque fois que le compteur est passe de 1 à 0 (c'est-à-dire toutes les milli-secondes). Lire le registre **`SYST_CSR`** efface automatiquement ce bit.

```
void busyWaitDuring (const uint32_t inDelayMS) {
  const uint32_t COUNTFLAG_MASK = 1 << 16 ;
  for (uint32_t i=0 ; i<inDelayMS ; i++) {
    while ((SYST_CSR & COUNTFLAG_MASK) == 0) {} // Busy wait, polling COUNTFLAG
  }
}
```

CENTRALE NANTES

Travail à faire

Écrire un programme *blinkled* utilisant la fonction **`busyWaitDuring`** pour exprimer les délais. Pour cela :

- dupliquer le programme précédent et le renommer ;
- écrire les fonctions **`startSystick`** et **`busyWaitDuring`** dans un nouveau fichier **`time.cpp`** ;
- écrire le fichier d'en-tête **`time.h`**, qui déclare les prototypes de ces deux fonctions ;
- modifier la fonction **`setup`** dans le fichier fichier **`setup-loop.cpp`** de façon à appeler **`startSystick`** ;
- modifier la fonction **`loop`** dans le fichier fichier **`setup-loop.cpp`** de façon à appeler **`busyWaitDuring`**.

Indications :

- dans l'écriture du fichier d'en-tête **`time.h`**, il ne faut faire aucune supposition sur l'ordre d'inclusion des fichiers ; aussi, il est recommandé de faire précéder la déclaration des prototypes par une ligne **`#include <stdint.h>`** (elle-même précédée de **`#pragma once`**) ;
- la première ligne du fichier **`time.cpp`** doit être **`#include "all-headers.h"`** : ainsi, on est sûr que toutes les déclarations sont disponibles.

CENTRALE NANTES