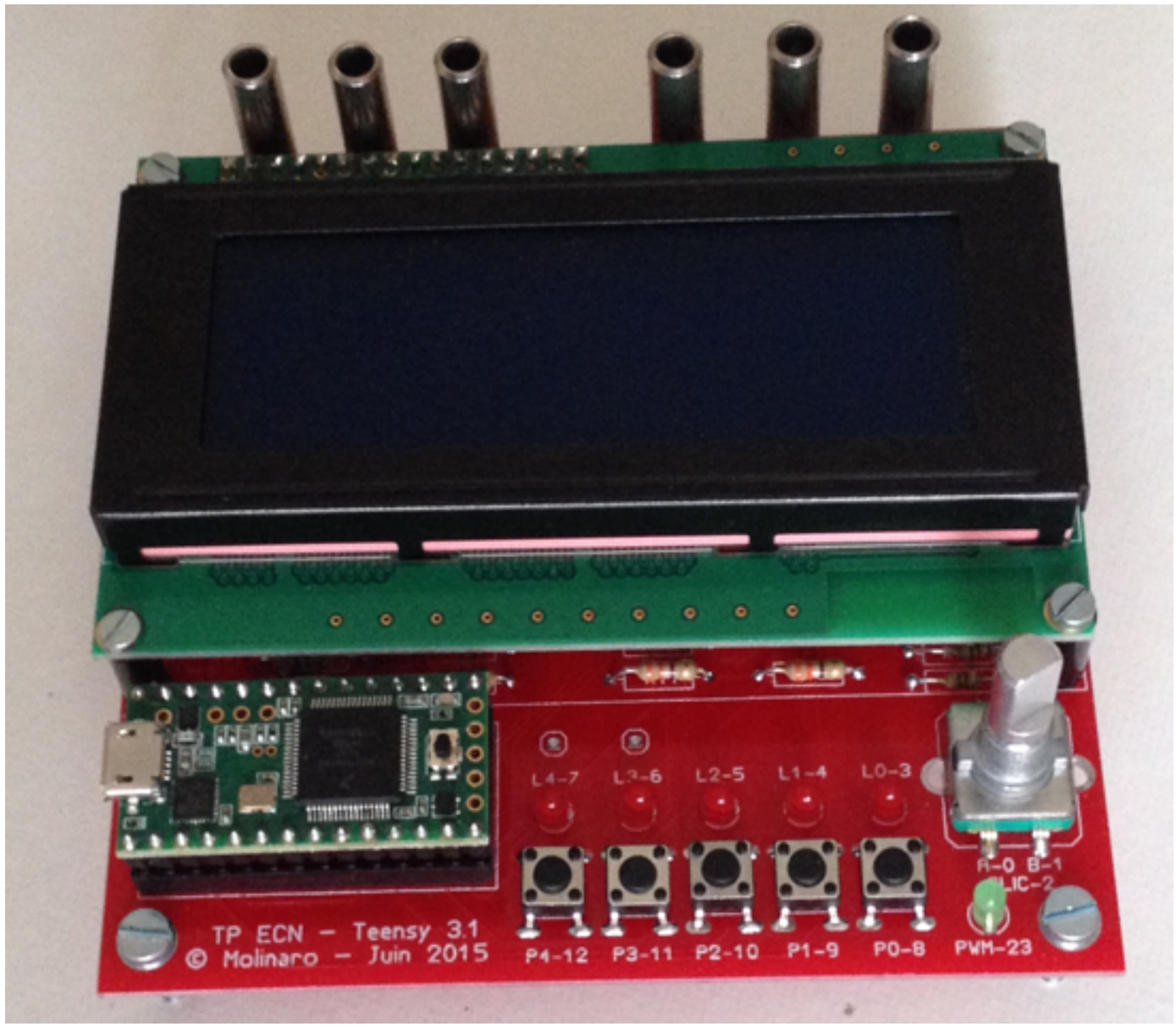
Temps Réel



Programme 03-systick

Pierre Molinaro
Janvier 2016

But de cette partie

Objectif :

- *compter précisément le temps.*

Problèmes à résoudre :

- installation de l'interruption `SysTick` ;
- écrire une routine d'attente active `busyWaitingDuringMS`.

Travail à faire :

- écrire un programme *blinkled* qui appelle `busyWaitingDuringMS`. Pour cela, dupliquer le programme précédent, et renommer `startup-sequential-no-systick.c` en `startup-sequential.c`.

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 un décompteur cyclique : `0`, `N`, `N-1`…, `1`, `0`, `N`, …, `N` étant une valeur programmable. La période est N+1.

Lorsqu'il est rechargé à N, il déclenche l'interruption n°15.

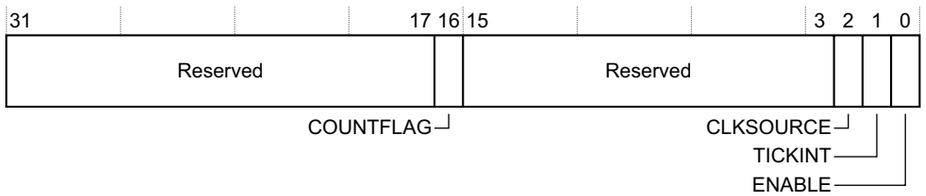
Registres du timer SysTick

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.

Programmation du timer SysTick

startup-sequential.c

```
SYST_RVR = 96000 - 1 ; // Interrupt every 96000 core clocks, i.e. every ms
SYST_CVR = 0 ;
SYST_CSR = SYST_CSR_CLKSOURCE | SYST_CSR_TICKINT | SYST_CSR_ENABLE ;
```

Ces instructions sont à placer entre la copie de la section *.data* et l'exécution des constructeurs des variables globales.

```
SYST_RVR = 96000 - 1 ;
```

Cette instruction fixe la période du compteur SysTick à 96000. Comme son horloge est à la fréquence de 96 MHz, l'interruption SysTick se déclenche toutes les milli-secondes.

```
SYST_CVR = 0 ;
```

Met à zéro la valeur courant du timer SysTick.

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

Configure et démarre le timer :

- SYST_CSR_CLKSOURCE, l'horloge du timer est l'horloge processeur à 96 MHz ;
- SYST_CSR_TICKINT, l'interruption n°15 est déclenchée à chaque remise à zéro ;
- SYST_CSR_ENABLE, le compteur est activé.

Routine d'interruption systickHandler

startup-sequential.c

```
//-------------------------------------------------------------------------*
//  systick handler                                                        *
//-------------------------------------------------------------------------*

static volatile uint32_t gUpTimeInMilliseconds ;

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

static void systickHandler (void) {
  gUpTimeInMilliseconds ++ ;
}
```

La variable `gUpTimeInMilliseconds` va compter le nombre de milli-secondes écoulées depuis le démarrage du micro-contrôleur.

Important pour la suite : déclarer la variable `gUpTimeInMilliseconds` avec le qualificatif `volatile`.

Installation de la routine d'interruption `systickHandler`

startup-sequential.c

```
//--- ARM Core System Handler Vectors
  { ResetISR, // 1
    NULL, // 2
    NULL, // 3
    NULL, // 4
    NULL, // 5
    NULL, // 6
    NULL, // 7
    NULL, // 8
    NULL, // 9
    NULL, // 10
    NULL, // 11
    NULL, // 12
    NULL, // 13
    NULL, // 14
    systickHandler // 15
  },
//--- Non-Core Vectors
  …
```

Il suffit de placer la routine d'interruption `systickHandler` à l'emplacement du vecteur d'interruption n°15.

Fonction uptimeMS

startup-sequential.c

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

uint32_t uptimeMS (void) {
  return gUpTimeInMilliseconds ;
}

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

La fonction uptimeMS retourne le nombre de milli-secondes écoulées depuis le démarrage du micro-contrôleur.

Fonction busyWaitingDuringMS

busy-waiting.c

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

void busyWaitingDuringMS (const uint32_t inDurationInMS) {
  const uint32_t deadline = uptimeMS () + inDurationInMS ;
  while (uptimeMS () < deadline) {} // Attente active
}

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

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

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

Travail à faire

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

- dupliquer le programme précédent et le renommer ;
- renommer `startup-sequential-no-systick.c` en `startup-sequential.c` ;
- écrire la routine `busyWaitingDuringMS` dans un fichier `busy-waiting.c` ;
- écrire le fichier d'en-tête `busy-waiting.h` correspondant ;
- écrire le fichier d'en-tête `uptime-ms.h` qui déclare la fonction `uptimeMS` ;
- modifier 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.