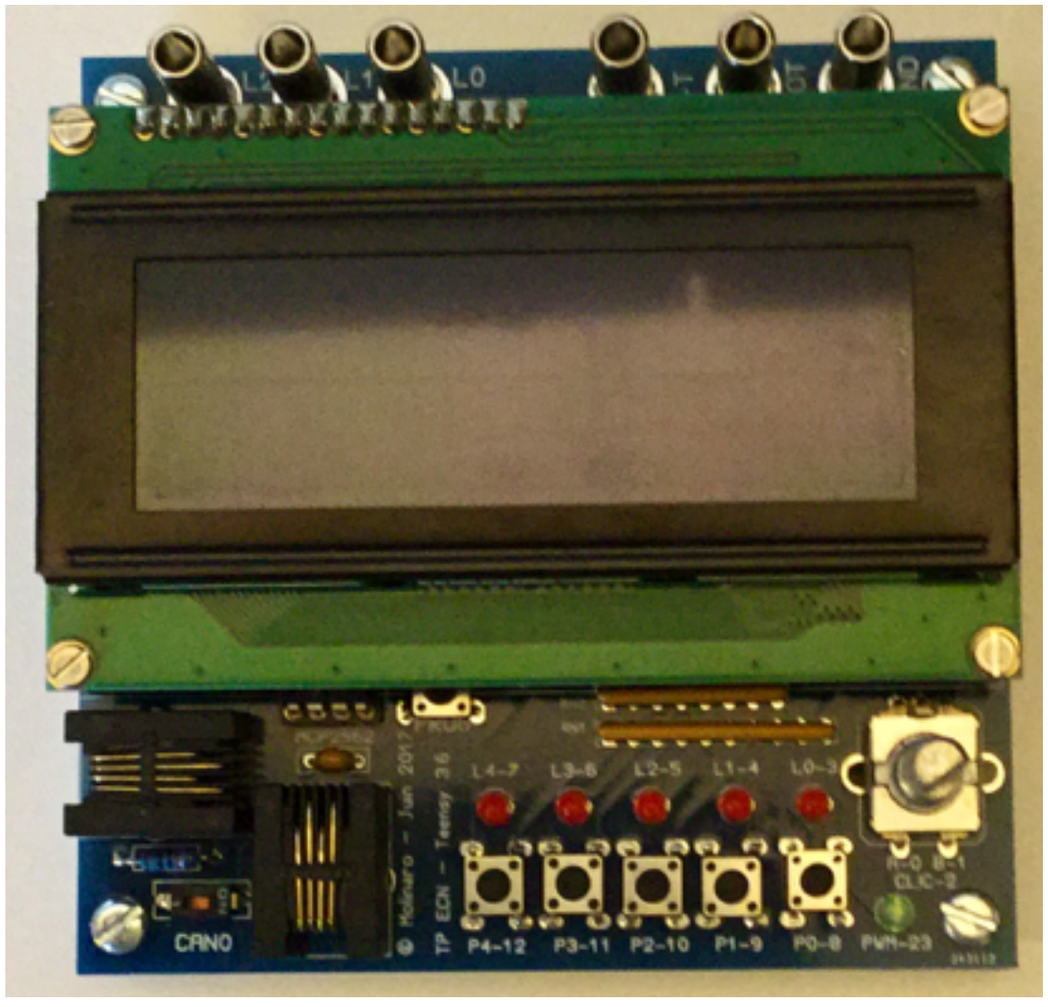
Temps Réel



Programme `03-modes-logiciels`

Pierre Molinaro

Juin 2018

Description de cette étape

Objectif : ajouter aux routines les annotations de mode logiciel.

Travail à réaliser : il est décrit à la dernière page.

CENTRALE NANTES

Les modes du processeur

Un processeur Cortex-M4 est dans l'un des deux modes suivants :

- le mode ***thread*** ;
- le mode ***handler***.

Les propriétés de ces deux modes sont configurables, aussi une description complète est trop vaste : nous nous limitons dans ce cours à la description des choix qui ont été faits.

Le mode ***thread*** sera le mode d'exécution des tâches. Par défaut, les interruptions sont activées dans ce mode. Le mode ***thread*** a sa propre pile. Actuellement, tout le code s'exécute dans ce mode : cela ne sera plus le cas à partir de l'étape 06 où apparaissent les interruptions.

Le mode ***handler*** sera le mode des routines d'interruption et des services de l'exécutif. Ceux-ci ne sont pas interruptibles. Le mode ***handler*** a sa propre pile.

Les modes logiciels

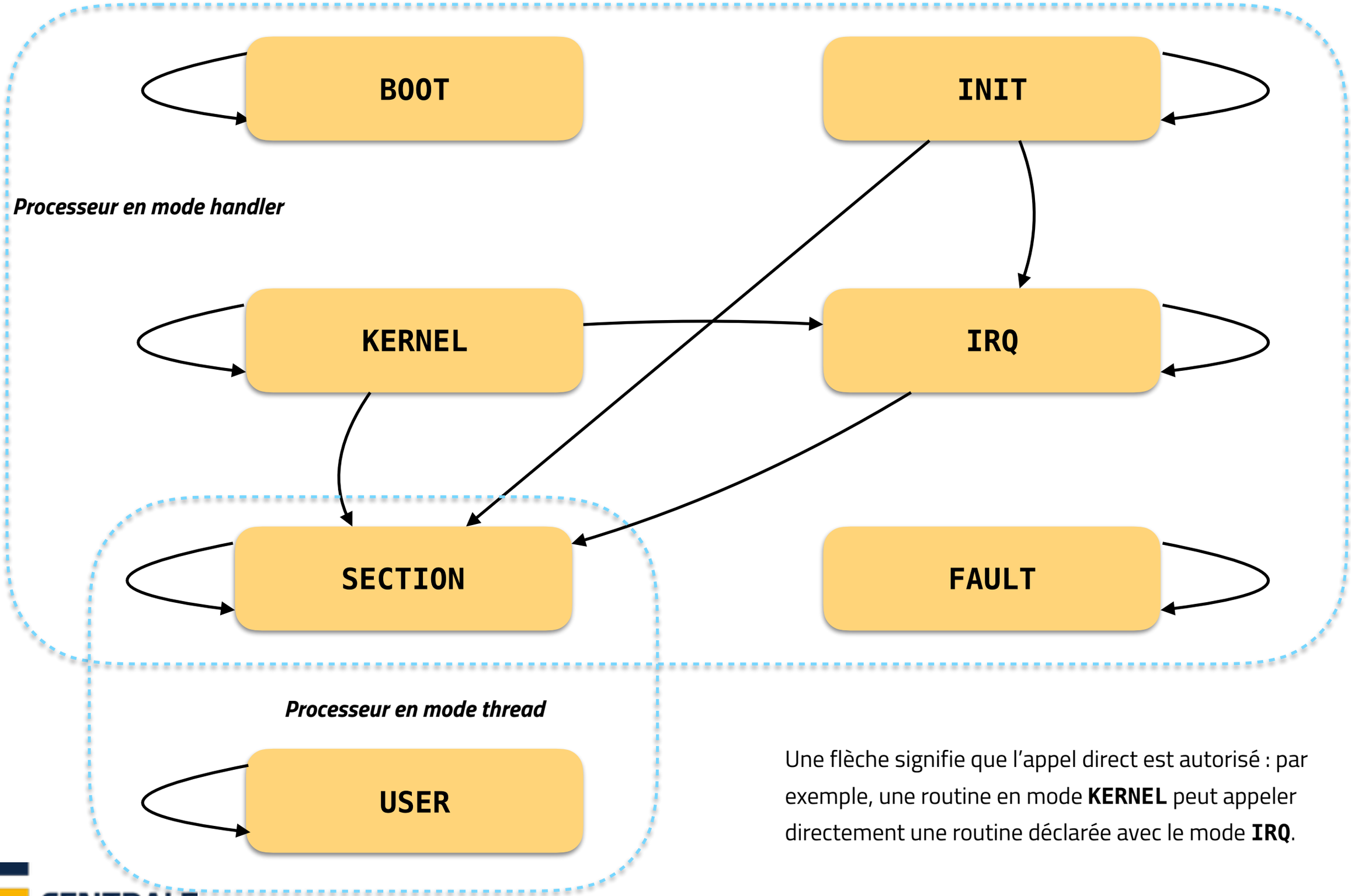
Le C et le C++ ne permettent pas de distinguer les fonctions qui doivent s'exécuter en mode *handler* et celles qui doivent s'exécuter en mode *thread*. Un appel incorrect n'est pas décelé à la compilation, et provoque le plus souvent un plantage à l'exécution.

De toute façon, ces deux seuls modes ne suffisent pas à caractériser toutes les classes de routines qui peuvent exister dans un exécutif.

Ce cours définit donc les sept modes logiciels suivants :

Mode logiciel	Commentaire
BOOT	C'est le mode des routines référencées dans la section **boot.routine.array**. Dans ce mode, les constructeurs des variables globales ne sont pas encore appelés, il ne faut pas référencer d'instances de classe C++. Les interruptions sont masquées.
INIT	C'est le mode des routines référencées dans la section **init.routine.array**. Dans ce mode, les constructeurs des variables globales ont été appelés. Les interruptions sont masquées.
KERNEL	Ce sera le mode des services de l'exécutif qui peuvent être appelés par le code d'une tâche, à travers une construction qui utilise l'instruction **svc** (Service Call). Exemple : la primitive **P** d'un sémaphore.
IRQ	Ce sera le mode des services de l'exécutif qui peuvent être appelés par le code d'une tâche (comme pour le mode **KERNEL**), ou par une interruption matérielle. Exemple : la primitive **V** d'un sémaphore.
SECTION	Mode d'une routine ininterruptible. Selon le contexte d'appel, le processeur peut être en mode *thread* ou en mode *handler*.
FAULT	Un processeur Cortex-M4 se surveille, et une exception **Fault** se déclenche en cas d'erreur. Actuellement, dans cette situation, l'exécution plante. Dans l'étape 09, on prendra en compte cette exception pour afficher un message d'erreur.
USER	C'est actuellement le mode des routines **setup** et **loop**. Quand l'exécutif sera ajouté, ce sera le mode d'exécution des tâches.

Graphe des modes logiciels



Une flèche signifie que l'appel direct est autorisé : par exemple, une routine en mode **KERNEL** peut appeler directement une routine déclarée avec le mode **IRQ**.

CENTRALE NANTES

Comment exprimer les modes logiciels (1/4)

Chaque fonction pour laquelle le mode logiciel est important va comporter en premier argument une variable dont la classe représente un des modes.

Lors de l'appel d'une fonction, cette variable devra être mentionnée.

Par la déclaration des constructeurs dans chacune de ces classes, on définit facilement les changements de modes qui sont autorisés.

Par exemple, la classe correspondante au mode **BOOT** est :

```
  class BOOT_mode_class { // PROVISOIRE (voir dans les pages suivantes)
    private : BOOT_mode_class (void) ;
    private : BOOT_mode_class & operator = (const BOOT_mode_class &) ;
    public : BOOT_mode_class (const BOOT_mode_class &) ;
  } ;
```

Par exemple, on déclare une fonction qui doit s'exécuter dans ce mode par :

```
void maFonctionBoot (const BOOT_mode_class MODE) ; // PROVISOIRE (voir pages suivantes)
```

Et cette routine doit être appelée par :

```
maFonctionBoot (MODE) ;
```

C'est-à-dire que l'on ne peut le faire qu'à partir d'une fonction elle-même déclarée dans le mode **BOOT**.

CENTRALE NANTES

Comment exprimer les modes logiciels (2/4)

Autre exemple : le mode **SECTION** peut être appelé à partir des modes **KERNEL**, **IRQ** et **INIT**.

```
class SECTION_mode_class { // PROVISOIRE (voir dans les pages suivantes)
  private: SECTION_mode_class (void) ;
  private: SECTION_mode_class & operator = (const SECTION_mode_class &) ;

  public: SECTION_mode_class (const IRQ_mode_class &) ;
  public: SECTION_mode_class (const KERNEL_mode_class &) ;
  public: SECTION_mode_class (const INIT_mode_class &) ;
} ;
```

Les quatre constructeurs définissent les changements de mode autorisés.

CENTRALE NANTES

Comment exprimer les modes logiciels (3/4)

Mais : en procédant comme indiqué dans les deux pages précédentes, l'argument de mode apparaît dans le code engendré, ce qui n'est pas souhaitable (augmentation de la taille du code, ralentissement) et même inutile.

Voici la solution qui a été retenue : les annotations relatives au mode **BOOT** sont définies comme suit (voir le fichier **software-modes.h** pour les autres déclarations) :

```
#ifdef CHECK_SOFTWARE_MODES // DÉFINITIF
  #define MODE  inSoftwareMode
  #define MODE_ inSoftwareMode,
#else
  #define MODE
  #define MODE_
#endif

#ifdef CHECK_SOFTWARE_MODES
  class BOOT_mode_class {
    private: BOOT_mode_class (void) ;
    private: BOOT_mode_class & operator = (const BOOT_mode_class &) ;
    public: BOOT_mode_class (const BOOT_mode_class &) ;
 } ;
#endif

#ifdef CHECK_SOFTWARE_MODES
  #define BOOT_MODE const BOOT_mode_class MODE
  #define BOOT_MODE_ const BOOT_mode_class MODE,
#else
  #define BOOT_MODE void
  #define BOOT_MODE_
#endif
```

Comment exprimer les modes logiciels (4/3)

La construction d'un projet provoque deux compilations du même source C++ :

- chaque fichier source C++ est d'abord compilé avec l'option **–DCHECK_SOFTWARE_MODES** ; cette option a pour effet de définir la variable de compilation **CHECK_SOFTWARE_MODES**, et par suite d'activer la vérification des modes logiciels ; le code engendré par cette compilation n'est pas utilisé ;
- chaque fichier source C++ est ensuite compilé sans l'option ci-dessus ; la vérification des modes n'est pas activée, et le code engendré est exactement le même que celui que l'on obtiendrait sans ces annotations : c'est ce code qui est utilisé par l'édition de liens.

En construisant un projet avec **1-verbose-build.py**, on peut voir le détail des commandes :

```
[ 41%] Checking start-teensy-3-6.cpp
[ 41%] /Users/pierremolinaro/treel-tools/gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-eabi-gcc -mthumb -
mcpu=cortex-m4 -x c++ -DCHECK_SOFTWARE_MODES -Os -Wall -Wextra -Werror -Wreturn-type -Wformat -Wshadow -Wsign-
compare -Wpointer-arith -Wparentheses -Wcast-align -Wcast-qual -Wwrite-strings -Wswitch -Wswitch-enum -
Wuninitialized -Wsign-conversion -ffunction-sections -fdata-sections -Wno-unused-parameter -Wshadow -std=c++14
-fno-rtti -fno-exceptions -Woverloaded-virtual -Weffc++ -fno-threadsafe-statics -Wmissing-declarations -
Wsuggest-override -c sources/start-teensy-3-6.cpp -o zBUILDS/start-teensy-3-6.cpp.check.o -DSTATIC= -I zSOURCES
-I /Volumes/dev-svn/GITHUB/real-time-kernel-teensy/solutions/03-software-modes -I sources -MD -MP -MF zBUILDS/
start-teensy-3-6.cpp.check.o.dep
.......
[ 58%] Compiling start-teensy-3-6.cpp
[ 58%] /Users/pierremolinaro/treel-tools/gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-eabi-gcc -mthumb -
mcpu=cortex-m4 -Os -Wall -Wextra -Werror -Wreturn-type -Wformat -Wshadow -Wsign-compare -Wpointer-arith -
Wparentheses -Wcast-align -Wcast-qual -Wwrite-strings -Wswitch -Wswitch-enum -Wuninitialized -Wsign-conversion
-ffunction-sections -fdata-sections -Wno-unused-parameter -Wshadow -std=c++14 -fno-rtti -fno-exceptions -
Woverloaded-virtual -Weffc++ -fno-threadsafe-statics -Wmissing-declarations -Wsuggest-override -c sources/
start-teensy-3-6.cpp -o zBUILDS/start-teensy-3-6.cpp.o -DSTATIC= -I zSOURCES -I /Volumes/dev-svn/GITHUB/real-
time-kernel-teensy/solutions/03-software-modes -I sources -MD -MP -MF zBUILDS/start-teensy-3-6.cpp.o.dep
```

CENTRALE NANTES

Utilisation pratique des modes logiciels (1/2)

La mise en œuvre pratique des modes logiciels est illustrée avec le mode **BOOT** ; on procèdera de manière analogue avec les autres modes.

En-tête d'une fonction sans argument :

En-tête de la fonction sans annotation de mode	`void f (void)`
En-tête de la fonction avec annotation de mode	`void f (BOOT_MODE)` **Ce qu'il faut écrire dorénavant**
En-tête de la fonction, vue par le compilateur lors de la vérification de mode	`void f (const BOOT_mode_class inSoftwareMode)`
En-tête de la fonction, vue par le compilateur lors de la génération du code	`void f (void)`

Vous n'avez pas à écrire ceci, c'est une simple illustration du travail du précompilateur

Et une fonction avec au moins un argument :

Noter la caractère de soulignement à la fin de **BOOT_MODE_** et l'absence de virgule

En-tête de la fonction sans annotation de mode	`void f (argument1)`
En-tête de la fonction avec annotation de mode	`void f (BOOT_MODE_ argument1)` **Ce qu'il faut écrire dorénavant**
En-tête de la fonction, vue par le compilateur lors de la vérification de mode	`void f (const BOOT_mode_class inSoftwareMode, argument1)`
En-tête de la fonction, vue par le compilateur lors de la génération du code	`void f (argument1)`

Vous n'avez pas à écrire ceci, c'est une simple illustration du travail du précompilateur

Utilisation pratique des modes logiciels (2/2)

Pour l'instruction d'appel de fonction, on utilise l'annotation **`MODE`** (ou **`MODE_`** si il y des arguments), et ce quelque soit le mode de la fonction.

Appel d'une fonction sans argument :

Sans annotation de mode	`f ()`
Avec annotation de mode	`f (MODE)` **Ce qu'il faut écrire dorénavant**
Appel de la fonction, vu par le compilateur lors de la vérification de mode	`void f (inSoftwareMode)` **Vous n'avez pas à écrire ceci, c'est une simple illustration du travail du précompilateur**
Appel de la fonction, vu par le compilateur lors de la génération du code	`f ()`

Et L'appel d'une fonction avec au moins un argument :

Noter la caractère de soulignement à la fin de **`MODE_`** et l'absence de virgule

Sans annotation de mode	`void f (argument1)`
Avec annotation de mode	`void f (MODE_ argument1)` **Ce qu'il faut écrire dorénavant**
Appel de la fonction, vu par le compilateur lors de la vérification de mode	`void f (inSoftwareMode, argument1)` **Vous n'avez pas à écrire ceci, c'est une simple illustration du travail du précompilateur**
Appel de la fonction, vu par le compilateur lors de la génération du code	`void f (argument1)`

Travail à faire

Dupliquer le répertoire de l'étape précédente et renommez-le par exemple **03-software-modes**.

Ajoutez aux sources le fichier **software-modes.h**.

Ajouter dans annotations de mode dans les fichiers d'en-tête suivants (et les fichiers C++ correspondants) :

- **setup-loop.h**, fonctions **setup** et **loop** ;
- **start-teensy-3-6.h**, fonctions **boot** et **init** ;
- **time.h**, fonctions **startSystick** et **busyWaitDuring**.

CENTRALE NANTES