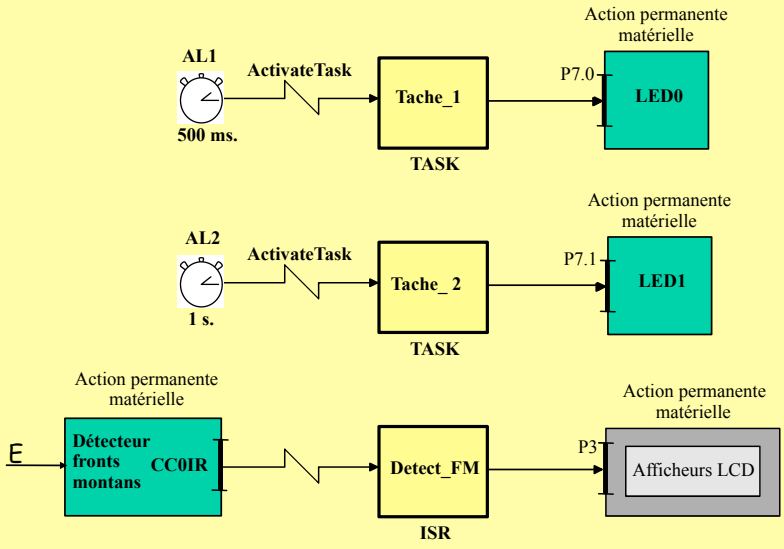
Chapitre 3 -

Développement d'une application

☞ Exemple de problème

Le but de l'exemple est de montrer toute la chaîne de développement en utilisant des tâches, ISR et alarmes (l'exemple en lui-même ne servant à rien).

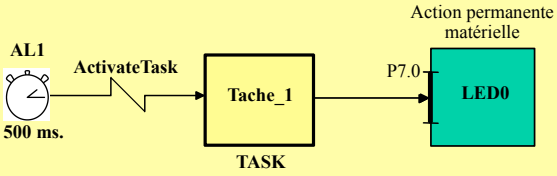


- Compléments pour Tache_1
  - activation par l'alarme AL1 toutes les 500 ms, première activation à 200 ms.
  - initialisation DP7.0 à 1 (sortie, valeur initiale nulle)
  - Tache_1 complémente LED0 à chaque activation

```
TASK (Tache_1)
{
   LED0 = ~ LED0 ;
   TerminateTask ( ) ;
}
```

```
sbit  LED0 = P7^0;

DP7 = 1 ;  // DP7.0 en sortie
LED0 = 0 ; // LED0 éteinte
```

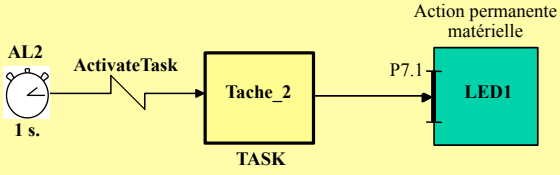


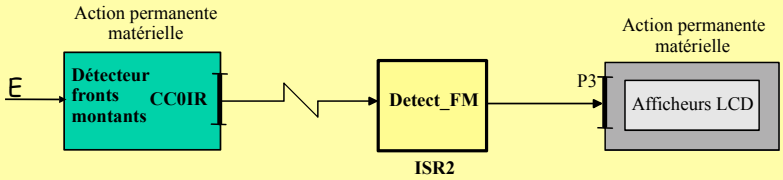
- Compléments pour Tache_2
  - activation par l'alarme AL2 toutes les 1 s., première activation à 200 ms.
  - initialisation DP7.1 à 1 (sortie, valeur initiale nulle)
  - Tache_2 complémente LED1 à chaque activation

```
TASK (Tache_2)
{
   LED1 = ~ LED1 ;
   TerminateTask ( ) ;
}
```

```
sbit  LED1 = P7^1;

DP7 = 2 ;  // DP7.1 en sortie
LED1 = 0 ; // LED1 éteinte
```



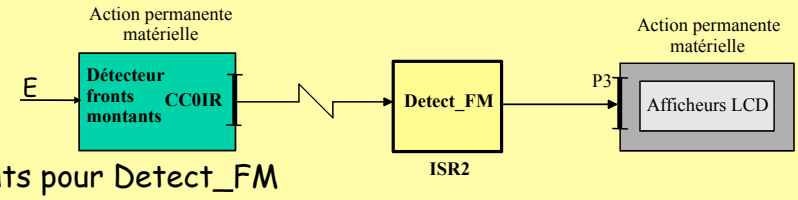


- Compléments pour Detect_FM
  - activation de l'ISR via l'interruption du détecteur de transition CC0IO, en fonction du signal externe E
  - initialisation du détecteur de transition et de l'afficheur LCD
  - Detect_FM affiche le nombre courant de fronts détectés

```
ISR (Detect_FM)
{
   static int N=0 ;

   N++;
   printf("\r%d", N) ;
   TerminateISR2 ( ) ;
}
```

```
CCM0 = 1 ; //front montant CC0IO
CC0IC = 0x50 ; // IT locale valide
                // ILVL=4, GLVL=0
lcd_Init ( ) ;   // init. Afficheur LCD
```

- Compléments pour Detect_FM
  - activation de l'ISR via l'interruption du détecteur de transition CC0IO, en fonction du signal externe E
  - initialisation du détecteur de transition et de l'afficheur LCD
  - Detect_FM affiche le nombre courant de fronts détectés

Le signal E est connecté sur le canal 0 (bit 0 du port P2).

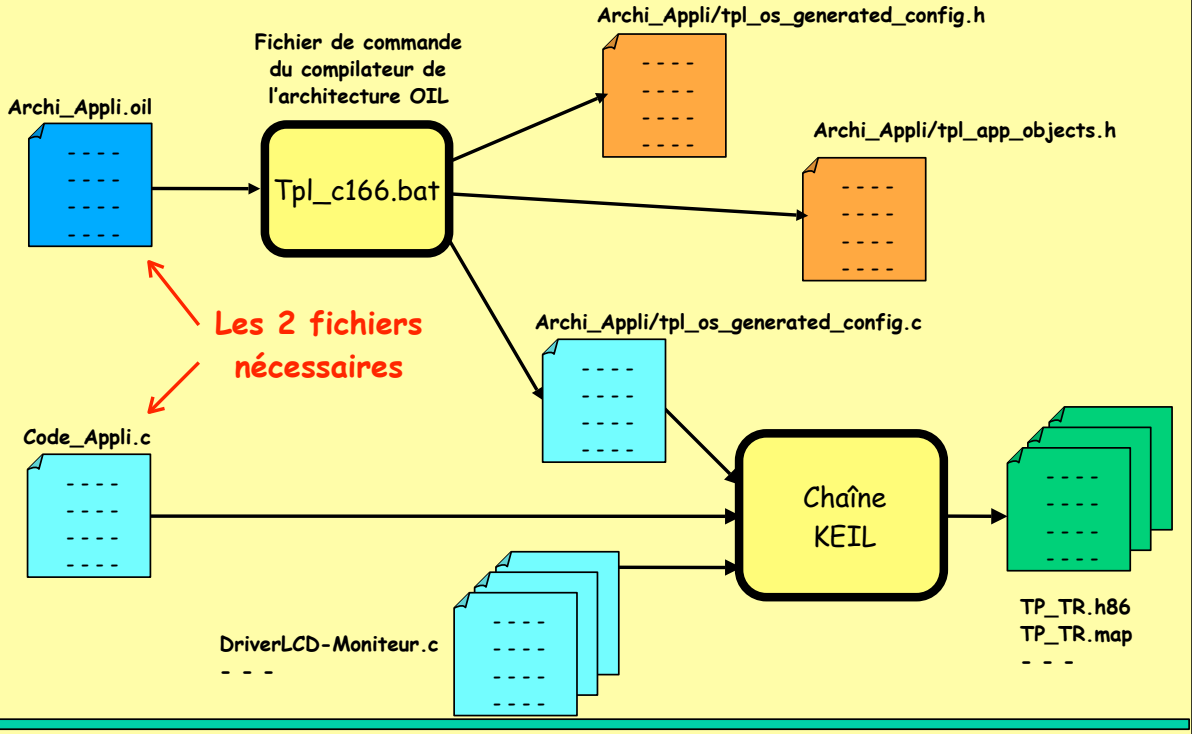
CCM0 = 0000 0000 0000 0001
CCM0 contrôle les canaux 0 à 3 (CC0IO à CC3IO). Pour chaque canal 4 bits définissent :
- La fonction (entrée de capture ou sortie de comparaison)
- le timer associé (si on l'utilise) T0 (bit à 0) ou T1 (bit à 1)

Ici pour le canal 0, on le fait travailler en simple détecteur de transition, la fonction « capture » même si elle est toujours en service ne nous intéresse pas.
0: puisque qu'il faut bien mettre quelque chose, mais on n'utilise pas T0
001 = détection de front montant

CC0IC = 0 1 0100 00
Raz de l'IT par 0 dans CCOIR, Enable IT par 1 dans CCOIE,
ILVL = 4 (niveau global à 4) et GLVL = 0 (pas d'intérêt)

L'ISR2 compte dans une variable statique le nombre de fronts détectés, et l'affiche via printf (putchar pilote directement l'afficheur LCD).

☞ Les fichiers et les outils dans le flot de développement



☞ 4 grandes étapes

Étape 1 : dans cette étape vous allez « coder » votre application dans 2 fichiers (ce seront les 2 seuls)

Archi_Appli.oil

Écriture du fichier de description de l'architecture de l'application, écrit en langage OIL (OSEK Implementation Language)

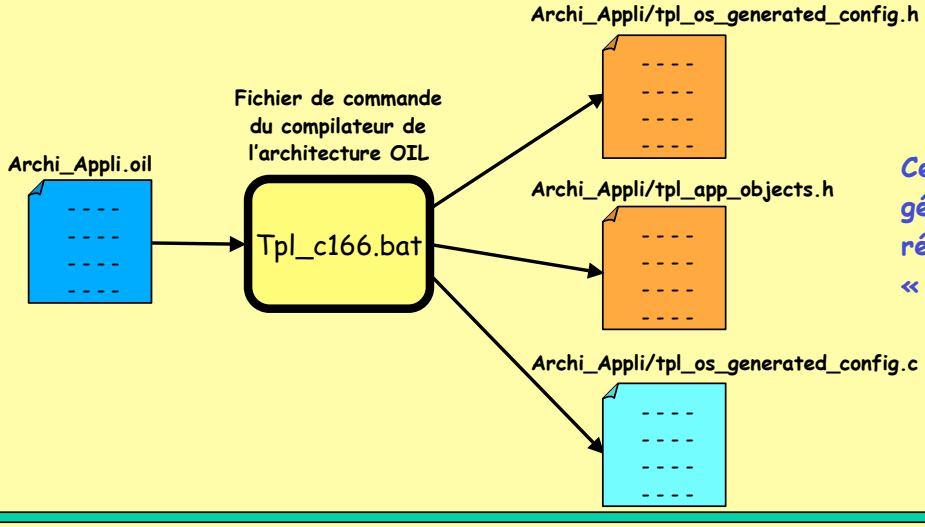
Code_Appli.c

Écriture du fichier des algorithmes des tâches, et des ISR, écrit en langage C

☞ 4 grandes étapes

Étape 2 : dans cette étape vous allez compiler l'architecture pour produire 2 fichiers contenant les structures de données de votre application temps réel.

Pour cela vous utilisez le fichier de commande Tpl_c166.bat

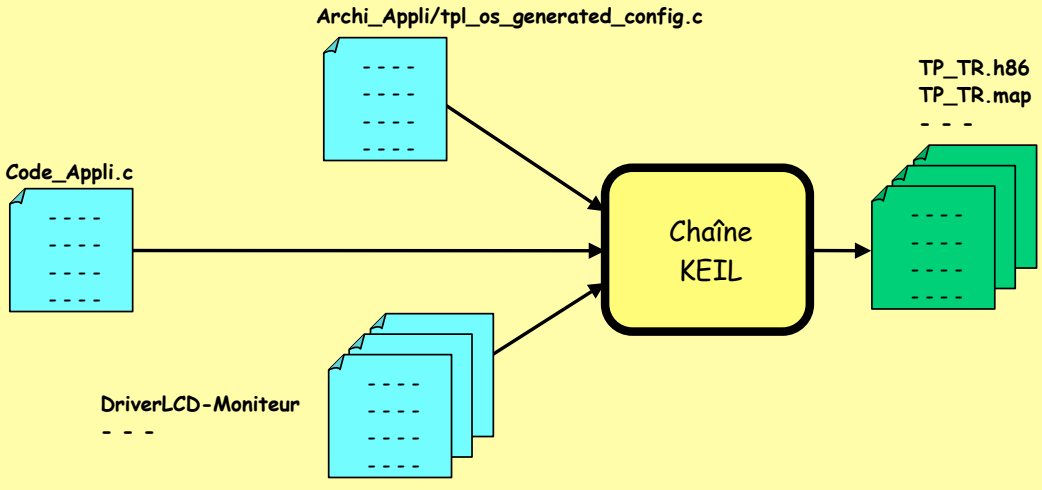


Ces 3 fichiers sont générés dans un répertoire nommé « Archi_Appli »

☞ grandes étapes

Étape 3 : dans cette étape vous compilez l'ensemble des fichiers C et assembleur pour générer (entre autres) l'exécutable.

Pour cela vous utilisez la chaîne de développement KEIL.



☞ **4 grandes étapes**

Étape 4 : dans cette étape vous faites le téléchargement de votre exécutable sur la carte cible, et faites la mise au point.

Pour cela vous utilisez la chaîne de développement KEIL.

Pour se remémorer l'utilisation de la chaîne, voir 2 documents :

- l'annexe : « Résumé sur l'utilisation de la chaîne KEIL »

- sur le serveur, le fichier pdf du tutorial de 1e année

« TP 1e année - Tutorial Keil.pdf »

☞ **Ecriture du fichier de description de l'architecture**

```
OIL_VERSION = "2.5" : "test" ;

IMPLEMENTATION trampoline {
};
CPU test {
  APPMODE std {
  };

  TASK Tache_1 {
    PRIORITY = 10;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
    USRSTACKSIZE = 512;
    SYSSTACKSIZE = 128;
  };

  TASK Tache_2 {
    PRIORITY = 5;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
    USRSTACKSIZE = 512;
    SYSSTACKSIZE = 128;
  };

  ISR Detect_FM {
    CATEGORY = 2;
    PRIORITY = 15;
    TRAP = 16;  // CAPCOM0
    USRSTACKSIZE = 512;
    SYSSTACKSIZE = 128;
  };

  COUNTER general_counter {
      TICKSPERBASE = 1;
      MAXALLOWEDVALUE = 2048;
      MINCYCLE = 20;
    };

    ALARM AL1 {
      COUNTER = general_counter;
      ACTION = ACTIVATETASK {
          TASK = Tache_1;
      };
      AUTOSTART = TRUE {
          ALARMTIME = 200;
          CYCLETIME = 500;
          APPMODE = std;
          };
    };

    ALARM AL2 {
      COUNTER = general_counter;
      ACTION = ACTIVATETASK {
          TASK = Tache_2;
      };
      AUTOSTART = TRUE {
          ALARMTIME = 200;
          CYCLETIME = 1000;
          APPMODE = std;
          };
    };
};
```

Tache_1 priorité 10

Tache_2 priorité 5

Liens AL1 avec Tache_1 et general_counter, Période 500 ms, Démarrage à 200 ms.

Liens AL2 avec Tache_2 et general_counter, Période 1 s., Démarrage à 200 ms.

Lien avec l'IT n°16, priorité 15

☞ Ecriture du fichier des algorithmes C de l'application

```
#include "../os/tpl_os.h"
#include "tpl_os_generated_configuration.h »
#include "tpl_os_wrappers.h"
#include <stdio.h>

#include <C167CS.h>

sbit LED=P7^0;
sbit LED1=P7^1;

void InitApp(void)
{
     DP7 = 0x03;
     LED0 = 0; LED1 = 0;
     CCM0 = 1;
     CC0IC = 0x50;
     lcd_Init ( );
}

void main (void)
{
    InitApp ( ) ;
    StartOS ( OSDEFAULTAPPMODE ) ;
}

TASK (Tache_1)
{
    LED0 = ~ LED0;
    TerminateTask ( );
}

TASK (Tache_2)
{
    LED1 = ~ LED1;
    TerminateTask ( );
}

ISR (Detect_FM)
{
    static  int  N=0;

    N++;
    printf("\r%d", N);
    TerminateISR2 ( );
}
```

Ne pas changer !

Le fichier C167CS.h contient la définition de l'ensemble des ports et bits du C167. Il permet d'utiliser directement les noms symboliques des E/S, comme DP7 (Direction Port 7) ou CCM0 (Capture Compare Mode 0) ou CCOIC (Capture Compare chanel 0 Interrupt Control).

Cette routine d'initialisation est appelée dans votre fonction principale

Démarrage de l'exécutif

☞ L'organisation proposée (fournie lors du 1e TP) est la suivante :

TP_MC_TR ← Votre répertoire de travail (nom libre)

- App
  - DriverLCD
    - DriverLCD-Moniteur.c
    - DriverLCD-Moniteur.h

    ← Les utilitaires pour l'afficheur et le compilateur C
  - Archi_Appli.oil
  - Code_Appli.c

    ← Vos 2 fichiers de votre application
  - Goil.exe ← Le compilateur OIL
  - Tpl_c166.bat ← Le fichier de commande pour compiler l'architecture
- Os

  --- tous les fichiers .c et .h du RTOS pour diverses cibles ---
- Templates

  --- structures de données utiles à la génération de l'application ---

Trampoline.uv2 ← Le fichier du projet KEIL

☞ Après compilation de l'architecture :

TP_MC_TR
- App
  - DriverLCD
    - DriverLCD-Moniteur.c
    - DriverLCD-Moniteur.h
  - Archi_Appli.oil
  - Code_Appli.c
  - Goil.exe
  - Tpl_c166.bat
  - Archi_Appli ← Le répertoire généré par la compilation
    - tpl_os_generated_configuration.c
    - tpl_os_generated_configuration.h
    - tpl_app_objects.h

    } ← Les 3 fichiers engendrés par la compilation de l'architecture
- Os

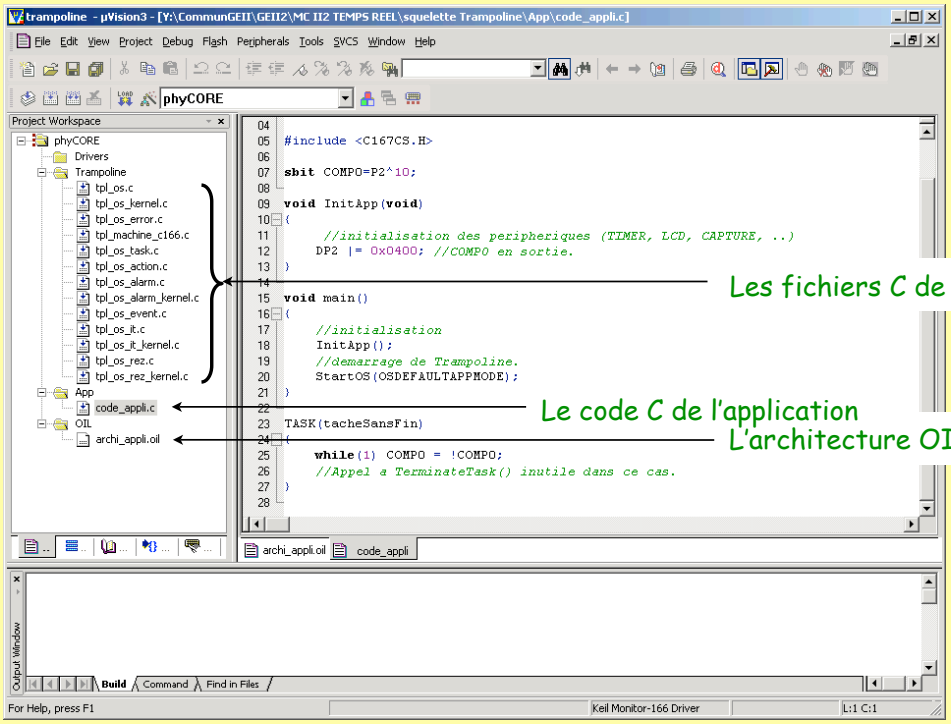
  --- tous les fichiers .c et .h du RTOS pour diverses cibles ---
- Templates

  --- structures de données utiles à la génération de l'application ---

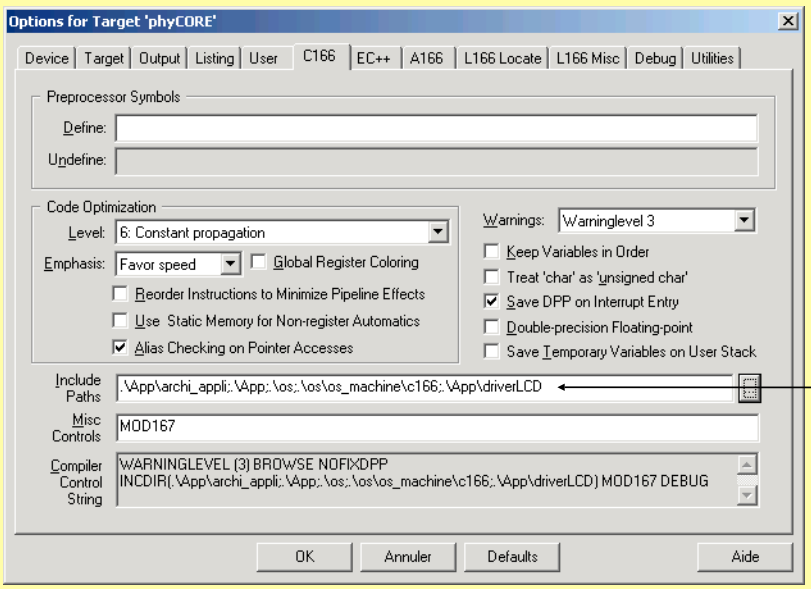
  Trampoline.uv2

La compilation de la structure rajoute les fichiers mentionnés dans un répertoire (du même nom que le fichier de structure).

☞ Organisation du projet KEIL



☞ Configuration des chemins d'accès pour les inclusions



Les chemins d'accès possibles pour la recherche des fichiers à inclure
(ne pas toucher !)

☞ Que faire pour développer une nouvelle application ?

Comme l'organisation et la génération de l'application sont des choses complexes, il vous est conseillé de suivre la démarche suivante :

1- Copier les 2 fichiers qui définissent votre application, dans le seul objectif d'en faire une sauvegarde

Archi_Appli.oil --> TP1.oil (par exemple)

Code_Appli.c --> TP1.c (par exemple)

2- Modifier les contenus des 2 fichiers

Archi_Appli.oil et Code_Appli.c

pour définir votre nouvelle application.

Pour chaque nouveau thème de TP, il est plus intéressant de reprendre l'ossature du premier TP (les risques d'erreurs sont nombreux compte-tenu du volume de fichiers et des configurations à faire). En fait, au 1° TP on donne la description d'un système complètement configuré. Elle contient deux fichiers « squelettes » pour la structure et le code C. Ainsi la chaîne est assez facile à utiliser.

Ainsi, aucune reconfiguration n'est nécessaire !

Annexe : les objets OIL

Mémo OIL pour Trampoline/OSEK - cible C167 (courtoisie Mikaël Briday)

La description OIL d'une application OSEK/VDX est composée d'un ensemble **d'objets OIL**. Chaque objet est décrit par un ensemble **d'attributs**. Outre les attributs standard, il est possible de rajouter des attributs spécifiques à une implémentation. Certains attributs utilisent des **paramètres**.

Cette description succincte reprend les différents objets utiles pour les TP et est basée sur la **norme OIL 2.5** disponible sur http://www.osek-vdx.org (67 pages).

☞ Les bases du fichier OIL

Base d'un fichier de description OIL

Un fichier OIL commence toujours par définir la version utilisée, où description est une chaîne de caractères:

OIL_VERSION = "2.5" : "description" ;

On trouve ensuite une partie implémentation pour définir des valeurs par défaut des objets, où restreindre les plages de valeurs de certains paramètres: priorité sur 8 bits, taille de pile de 128 octets par défaut, … Cette partie sera toujours vide dans notre cas:

IMPLEMENTATION trampoline {};

Le dernier élément d'une description OIL est le conteneur CPU. Celui-ci va contenir tous les objets des applications développées (TASK, RESOURCE, ALARM, …):

```
CPU test {
        // liste des objets.
};
```

Objets OIL

Les objets OIL définissant les caractéristiques de l'application (tâches, ressources, alarmes, …) sont listés dans l'élément CPU de la description OIL. Chaque objet a un certain nombre **d'attributs** qui sont listés pour le C166. Certains attributs ont des **paramètres** qui sont précisés entre accolade (voir les exemples).

☞ Objets « APPMODE » et « TASK »

APPMODE

Cet objet définit les différents modes pour l'application (debug, production, ..). Au moins un APPMODE doit être définit dans un CPU (pas d'attributs dans les TP).

APPMODE std {};

TASK

- **PRIORITY** (entier) : priorité de la tâche
- **AUTOSTART** (FALSE, TRUE) : Si l'attribut est TRUE, la tâche est dans l'état READY au démarrage de l'OS. Il faut préciser en paramètre les modes d'application (APPMODE) ;
- **ACTIVATION** (entier) : nombre maximal d'activations d'une tâche à un instant donné ;
- **SCHEDULE** (NON,FULL) : Définit si la tâche est préemptible (FULL) ou non (NON). Si elle n'est pas préemptible, elle ne peut pas utiliser de ressources internes ;
- **RESOURCE** (nom) : Ressource utilisée par la tâche (ceci est utile pour déterminer la priorité de la ressource avec PCP);
- **EVENT** (nom) : Evénements qui peuvent être utilisés par une tâche étendue ;
- **USRSTACKSIZE** (entier) : Taille de la pile utilisateur pour la cible C166. La pile utilisateur stocke les variables locales ;
- **SYSSTACKSIZE** (entier) : Taille de la pile système pour la cible C166. La pile système est utilisée pour les appels de fonctions. Cette pile utilise la mémoire interne du processeur (3ko sur le C167CS).

☞ Objet « TASK »

Exemple

```
TASK tacheAffichage {
        PRIORITY = 2;
        AUTOSTART = TRUE {
                APPMODE = std;
        };
        SCHEDULE = FULL;
        ACTIVATION = 3;
        RESOURCE = resource1;
        RESOURCE = resource2;
        EVENT = event1;
        USRSTACKSIZE = 256;
        SYSSTACKSIZE = 256;
};
```

☞ Objet « ISR »

ISR (Interrupt Service Routine)

- **CATEGORY** (entier) : Catégorie de l'ISR:
  - 1 : Interruption rapide, n'interfère pas avec OSEK ;
  - 2 : Interruption qui peut effectuer un appel système (démarrer une tâche, …).
- **PRIORITY** (entier) : priorité de l'interruption au sens OSEK (indépendant de la priorité matérielle!!) ;
- **RESOURCE** (nom) : ressource utilisée par l'ISR ;
- **USRSTACKSIZE** (entier) : taille de la pile utilisateur pour la cible C166 ;
- **SYSSTACKSIZE** (entier) : taille de la pile système pour la cible C166 ;
- **TRAP** (entier) : index de l'interruption dans le vecteur d'interruption matérielle ;

```
ISR AppuiBouton {
        CATEGORY = 2;
        PRIORITY = 30;
        RESOURCE = ressource1;
        USRSTACKSIZE = 256;
        SYSSTACKSIZE = 256;
        TRAP = 16;      // CAPCOM 0
};
```


☞ Objets « EVENT » et « RESOURCE »

EVENT

- **MASK** (entier ou AUTO): défini le masque utilisé par les tâches.

```
EVENT event1 {
        MASK = AUTO;
};
```

RESOURCE

- **RESOURCEPROPERTY** (INTERNAL,STANDARD): définit le mode de fonctionnement de la ressource. La ressource peut être interne (INTERNAL) ou non (STANDARD);

```
RESOURCE resource1 {
        RESOURCEPROPERTY = STANDARD;
};
```

☞ Objet « COUNTER »

COUNTER

- **TICKSPERBASE** (entier): Nombre d'impulsion du timer matériel nécessaires pour faire progresser le compteur d'une unité (i.e. une impulsion compteur). Pour le C166 à 40MHz, **l'impulsion timer est fixé à 1ms**;
- **MAXALLOWEDVALUE** (entier): Valeur maximale que peut prendre le compteur (afin d'optimiser les tailles des données);
- **MINCYCLE** (entier): définit le nombre minimal d'impulsions compteur pour une alarme périodique reliée au compteur.

```
COUNTER generalCounter {
        TICKSPERBASE = 1;
        MAXALLOWEDVALUE = 65535;
        MINCYCLE = 128;
};
```


☞ Objet « ALARM »

ALARM

- **COUNTER** (nom): nom du compteur associé à l'alarme (un compteur par alarme);
- **ACTION** (nom): action à réaliser quand l'alarme est déclenchée :
  - **ACTIVATETASK**: activation d'une tâche. Le paramètre associé est le nom de la tâche;
  - **SETEVENT**: déclenchement d'un évènement. Le paramètre associé est le nom de l'événement;
  - **ALARMCALLBACK**: appel d'une fonction utilisateur. Le paramètre associé est le nom de la fonction appelée.
  - **AUTOSTART** (FALSE, TRUE): L'alarme est dans l'état READY au démarrage de l'OS. Si l'attribut est TRUE, il faut préciser en paramètre le/les modes d'application APPMODE, la date de la première échéance ALARMTIME, et la période de l'alarme CYCLETIME. Les temps sont en nombre d'impulsions compteur.

☞ Objet « ALARM »

Exemple

```
ALARM alarmTest {
        COUNTER = generalCounter;
        ACTION = ACTIVATETASK {
                TASK = tachePeriodique;
        };
        AUTOSTART = TRUE {
                ALARMTIME = 1;
                CYCLETIME = 5000;
                APPMODE = std;
        };
};
```