Architecture logicielle d'une application complexe temps réel en Informatique Industrielle

CESI 2ème année

2020 - 2021

1- Organisation logicielle de gestion d'un système temps réel sans moniteur temps réel

2- Machine à états finis et codage

A. Nketsa

1- Organisation logicielle de gestion d'un système temps réel en informatique industrielle et/ou embarquée sans moniteur temps réel

Un système est dit temps réel s'il est réactif. C'est-à-dire qu'il est capable de traiter toutes les sollicitations significatives du système dans le temps imparti.
Pour gérer ce type de système, il existe différentes méthodes :

- l'utilisation d'un moniteur temps réel qui gère et ordonnancent les tâches de manière à respecter toutes les contraintes de temps et de précision des calculs.
- l'organisation judicieuse des tâches permettant aussi de respecter les mêmes contraintes. Mais dans ce cas, le nombre de tâches doit être réduit.

Le point le plus important dans la gestion de ces systèmes est de pouvoir maîtriser le temps. Cela passe par la réduction des boucles d'attente passive. Par exemple, faire une boucle pour attendre qu'un capteur devienne actif est une perte de temps pour le microprocesseur.
Il est évident dans ce cas que le microprocesseur perd son temps à attendre alors qu'il peut s'occuper d'autres tâches. C'est pourquoi dans la gestion des systèmes temps réel, on utilise un moniteur qui a pour rôle essentiel la distribution du temps processeur à différentes tâches selon des priorités. En effet si une tâche doit passer son temps à attendre un événement, alors elle en informe le moniteur qui la met en attente et donne le processeur à une autre. Si le moniteur est préemptif alors la tâche n'a même pas besoin de l'informer. Dans ce cas, le moniteur prend autoritairement la main au bout d'un certain temps ou si un événement majeur est intervenu et que des taches prioritaires sont prêtes à utiliser le processeur.

Une autre solution plus simple de gestion consiste à décomposer son application en un ensemble de **tâches non bloquantes**. Cette situation est très utile dans les systèmes à base de microcontrôleur sans système d'exploitation et de les enchaîner en termes d'exécution dans une boucle infinie. Pour réaliser, ce type de gestion, il faut que les tâches aient une part d'autonomie pour faire leur gestion interne. La seule contrainte à respecter est que le temps de cycle soit inférieur à la plus petite durée entre deux événements consécutifs sur une même entrée.

Une méthode pour donner de l'autonomie à un programme est de le décrire par une machine à états ou un Grafcet. Par ailleurs, la mise en œuvre de ce modèle doit être non bloquante. C'est-à-dire, connaissant l'état de l'évolution, on est capable d'exécuter une partie de la tâche globale et on rend la main pour d'autres tâches. En particulier, si on attend un événement, on teste s'il est survenu ou pas. S'il y a l'occurrence de l'événement, on applique le traitement associé et on passe au prochain état de traitement. S'il n'y en a pas, on passe la main. Car il n'y a pas d'activité pour l'instant.
Il arrive aussi que l'on utilise une base de temps pour synchroniser les cycles de programme.

Nous allons utiliser cette solution dans la mise en œuvre des systèmes en informatique industrielle et/ou embarquée que nous allons réaliser.
Pour cela, nous allons utiliser une horloge temps réel qui répartira l'attribution du processeur aux différentes tâches. Cette méthode est appropriée pour l'automatique car le quantum de temps peut être considéré comme la période d'échantillonnage.

Les systèmes considérés en informatique industrielle et/ou embarquée sont des systèmes de contrôle-commande.

Dans la mise en œuvre des systèmes de **contrôle-commande**,
1- La structure du système comporte 2 parties :
   a) le bloc de commande qui élabore des ordres en fonction de l'environnement
   b) la partie opérative (système commandé) qui renseigne sur l'environnement et qui reçoit les ordres envoyés par le bloc de commande. Ces ordres déclenchent des fonctions dont les résultats sont endogènes (internes : variables modifiées, périphériques internes commandés, etc.) ou exogènes (externes : sorties physiques, etc.)

2- L'exécution d'un programme sur un microcontrôleur est séquentielle.
C'est-à-dire que le microcontrôleur ne peut pas effectuer deux instructions en même temps. Ceci est moins vrai aujourd'hui parce qu'il existe des microcontrôleurs multi-cœur. Nous allons quand même considérer le cas d'un microcontrôleur à un cœur car les principes que nous proposons pour la mise en œuvre restent valables mais à une vitesse d'exécution plus élevée.

3- Le principe de fonctions non bloquantes permet de mettre en place un mécanisme d'échantillonnage qui permet de réaliser un pseudo-parallélisme dans l'exécution du bloc de commande et de la partie opérative.

Le pseudo-parallélisme consiste à exploiter la vitesse d'exécution des instructions par rapport au temps de réponse du système commandé pour donner **l'impression de faire plusieurs tâches en même temps**.

Une fonction non bloquante est une fonction dont on peut borner le temps d'exécution. En d'autres termes, la fonction ne contient pas de boucle infinie (par exemple l'attente d'un événement dont le temps d'occurrence n'est pas défini (borné)).

Ce pseudo-parallélisme peut se faire avec la notion de cycle de fonctionnement. On peut rapprocher cette notion de cycle de fonction à celle d'une période d'horloge dans le cas de la mise en œuvre d'une machine à états en électronique numérique. Pour cela, le système fonctionne sur la base d'un cycle et le début du cycle est considéré comme le font montant de l'horloge.

Un cycle correspond à :
- la prise en compte des entrées,
- l'exécution de toutes les fonctions prévues (bloc de commande, partie opérative (actions, etc.)),

Pendant un cycle :
- toutes les fonctions prévues sont exécutées dans les temps impartis,
- toutes les variations des entrées sont prises en compte et traitées pendant le cycle suivant. Pour cela, on lit les entrées au début du cycle et on actualise les sorties externes à la fin du cycle. Ceci permet de garantir la réactivité du système et d'éviter des incohérences entre des entrées pour un même cycle.

Cependant une entrée est prise en compte si son changement est maintenu jusqu'au début du prochain cycle. Dans ce cas, des modifications trop rapides peuvent ne pas être prises en compte.

Deux situations peuvent en résulter :
a) soit les modifications de trop courte durée sont considérées comme des parasites, donc elles n'ont pas à être traitées
b) soit on peut mettre en place, un mécanisme de mémorisation de toute modification pendant un cycle.

Nous allons considérer le cas a)

On peut ajouter des fonctions supplémentaires tant que la réactivité est garantie. La réactivité correspond au fait que toutes les modifications acceptées sont prises en compte et traitées pendant le cycle.

Compte tenu de ces contraintes, nous souhaitons proposer une **architecture logicielle générique des systèmes de contrôle commande** de manière à permettre son adaptation la plus simple possible. Pour cela, **l'approche modulaire (orientée composants)** semble naturellement indiquée. La mise en œuvre comportera des **composants non bloquants** parce que tous les événements seront échantillonnés sur la base du cycle de fonctionnement.

Signalons que cette approche permet d'exécuter plusieurs modèles (systèmes) en pseudo-parallélisme. Il suffit de les appeler séquentiellement ou à des instants différents en fonction de la durée d'exécution de chaque modèle.

Cette approche nous permet d'avoir la même structure de programme (ou de description) en langage C ou assembleur que pour le VHDL

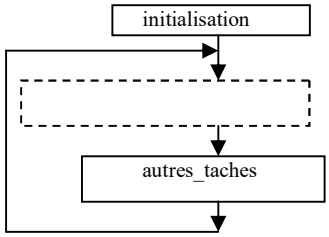
Le cycle peut être :

- **asynchrone**, c'est-à-dire sans base de temps
- **synchrone**, c'est-à-dire avec une base de temps

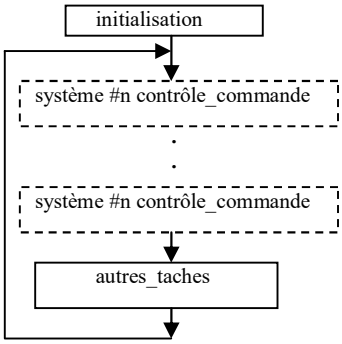
Remarque :
Autres_taches sont d'autres fonctions supplémentaires non bloquantes que l'on peut faire exécuter en pseudo parallélisme tant que la réactivité du système de contrôle-commande reste garantie

- soit sans une base de temps (asynchrone), c'est-à-dire qu'à la fin de l'exécution de toutes les fonctions prévues, le processus recommence. C'est le temps d'exécution des fonctions prévues qui définit la durée du cycle.

Structure pour un seul modèle:



Structure pour plusieurs modèles (pseudo-parallélisme):

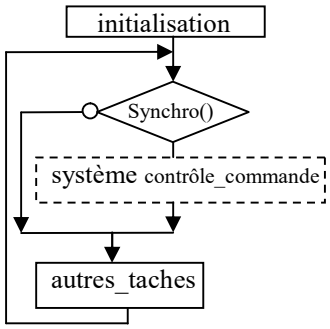


- soit avec une base de temps (synchrone), c'est-à-dire que l'on définit la durée d'un cycle d'exécution en prenant en compte la réactivité du système. Dans ce cas, toutes les fonctions nécessaires pendant un cycle doivent avoir le temps d'être complètement exécutées.

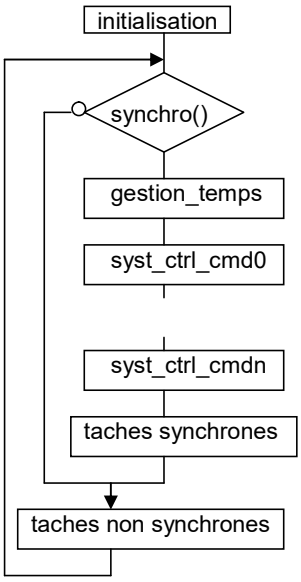
Pour cela, on définit une fonction de temporisation, synchro() qui retourne 1 chaque que fois que le temps imparti est écoulé.

L'avantage de cette approche est que l'on peut contrôler le temps d'exécution. Dans le cas où ce temps est dépassé, on peut déclencher une alarme pour signaler le mauvais fonctionnement.

Structure pour un seul modèle:



Structure pour plusieurs modèles (pseudo-parallélisme):

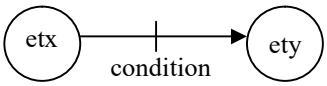


Rappel Modélisation par machine à états

Définition

Une machine à états est une représentation des systèmes séquentiels ayant au plus un état actif à la fois.

Représentation

1- Les états sont représentés par des ronds numérotés	
2- Cas particulier : état initial L'état initial est représenté par un double rond	
3- Le passage d'un état à un autre est appelé transition. Elle est représentée par une flèche étiquetée par une condition	

4- Règle de représentation

Si plusieurs transitions partent d'un même état, les conditions associées doivent être exclusives 2 par 2.

Les conditions sont exclusives si Cij . Cik = 0 (fig 1)

Pour rendre les conditions exclusives, on peut définir des priorités entre ces conditions (fig 2)

On ne sait pas si Cij et Cik sont exclusives
On les rend exclusives en définissant une priorité entre les deux conditions comme ci-contre

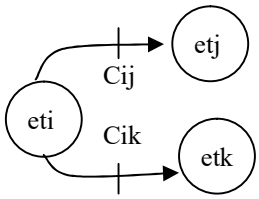


fig 1

Exclusion des conditions

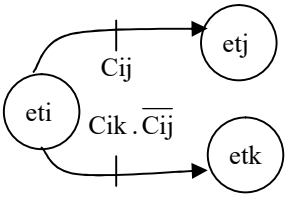


fig 2

Cij et plus prioritaire que Cik . $\overline{Cij}$

Règles d'évolution

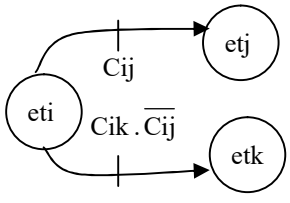
1- Un seul état de la mae doit être actif à la fois

2- L'état initial est actif à la mise sous tension ou au démarrage.

3- On détermine l'état actif puis on évalue les conditions associées à chaque transition de cet état.
Puisque les conditions associées sont exclusives, une seule de ces conditions **peut être** vraie.
Pour la transition dont la condition associée est vraie, de façon indivisible :
- on désactive l'état actif (état amont de la transition)
- et on active l'état aval de la transition.

4- on recommence en 3-

Exemple :



Considérons les cas suivants avec comme état actif eti :
1- si état, eti, est actif et Cij = 0 et Cik =0 alors pas d'évolution
2- si état, eti, est actif et cij =1 et Cik=*
alors on désactive eti et on active etj qui devient l'état actif.
3- si état, eti, est actif et cij =0 et Cik=1
alors on désactive eti et on active etk qui devient l'état actif.

Les actions
Une machine à états est utilisée pour commander d'autres dispositifs.

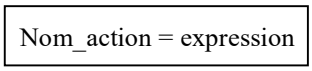
Pour cela, on définit 2 types d'actions :
- les actions à mise à zéro implicite (AMZI)
- les actions à mise à zéro explicite (AMZE) : ce sont des actions nécessitant une mémorisation

Nous allons simplifier ces actions en considérant comme en électronique numérique que la plupart de ces actions sont binaires.

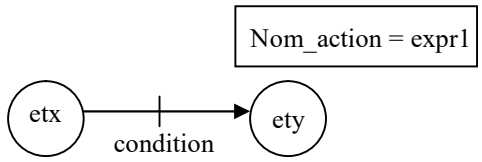
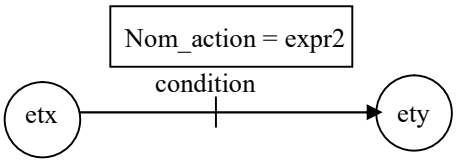
Par exemple une action de comptage peut se faire comme en VHDL, en calculant en_cpt comme une action AMZI. Le compteur étant un composant qui exploitera cette commande.

AMZI
Les actions à mise à zéro implicite sont des actions pour lesquelles on indique sur la mae uniquement les activations. Ce sont des actions combinatoires.

Représentation

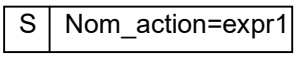
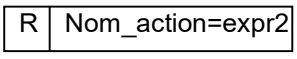
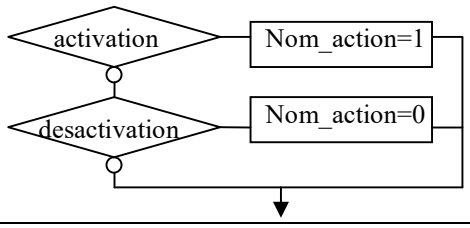


Les actions peuvent être placées :

Sur les états	Sur les transitions
 **Equation :** Nom_action = ety . expr1	 **Equation :** Nom_action = etx . condition .expr2

AMZE
Les actions à mise à zéro explicite sont des actions pour lesquelles on indique sur la mae uniquement les activations et les désactivations. Ce sont des actions à mémoire.
Les activations et désactivations sont des AMZI et peuvent être placées sur état ou sur transition

Représentation

Equation activation : 	**Equation désactivation :** 
1- On calcule activation et désactivation 2- On calcule l'action mémorisée en appliquant l'organigramme ci_contre	

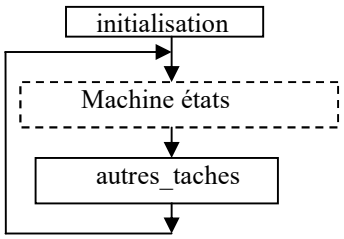
Mise en œuvre

Pré-requis : Notions de masque et calcul des masques
Opérateurs logiques et opérateurs booléens

Canevas de codage d'une machine à états en langage assembleur

1- Nous considérons qu'un microcontrôleur peut gérer plusieurs commandes dans un cycle de traitement. Pour cela, les fonctions associées doivent être non bloquantes. C'est-à-dire que dans ces fonctions, on ne trouvera ni de boucle non maîtrisée, ni de boucle d'attente d'une entrée.

2- La structure du programme principal est une boucle infinie dans laquelle différentes machines à états sont appelées.

Le cycle peut être défini par l'organigramme ci-contre : Un ou plusieurs fonctions non bloquantes dont la MAE Autres_tâches sont d'autres fonctions supplémentaires non bloquantes que l'on peut faire exécuter en pseudo parallélisme tant que la réactivité des machines à états non bloquantes reste garantie.	

Pour cela, l'approche modulaire (**basée composants**) est naturellement indiquée.

La mise en œuvre de la mae sera non bloquante parce que tous les événements seront échantillonnés sur la base du cycle de fonctionnement.