Conseils pour la réalisation du projet

1 Mise en place des cahiers des charges, spécifications :

Il faut avancer dans l'analyse du projet pour être capable d'établir les trois points suivants, en s'aidant des schémas ci-dessus, du texte du sujet et des textes des Tps préparatoires (qui vous donnent la culture manquante à la réalisation du projet).

- cahier des charges pour les parties électroniques,
- choix des périphériques sur lesquels brancher le hardware,
- cahier des charges des comportements des briques logicielles.

Il faut reprendre les schémas ci-dessus pour faire le point sur les informations qui s'échangent entre entités différentes (le bloc de contrôle au sol, le robot, la supervision, les obstacles, l'opérateur) pour voir comment cela va se traduire au niveau électronique hardware. Il est alors extrêmement important de se poser les questions suivantes pour chaque signal : qui parle à qui ? Dans quel sens l'information circule-t-elle ? On traduira cela par des flèches orientées.

Dans le cas où des informations sont transmises au micro-contrôleur, le but est de bien comprendre les informations qu'il faut réussir à reconstituer logiciellement, et la temporalité d'arrivée de ces informations. On pourra alors définir dans quel type de variable ces informations seront stockées quand elles sont complètes et quels sémaphores seront nécessaires pour prévenir une autre brique logicielle indépendante d'une information complète disponible. On pourra aussi définir les variables qui permettent de stocker une information en cours de réception ou d'envoi, ainsi que les variables de contrôle qui permettent de savoir où l'on en est de l'envoi ou de la réception.

A ce stade vous dessinerez juste des blocs pour les modules électroniques (sans aucun schéma, ni détail dedans) en précisant juste en entrée et en sortie de bloc des chronogrammes comportant les indications sur la grandeur qui contient l'information :

- Est ce l'amplitude du signal qui contient l'information importante ?
- Est ce un niveau logique qui contient l'information importante ?
- Est ce un événement qui contient l'information importante ?
- Est ce une durée entre deux événements qui contient l'information importante ?
- Est ce le nombre d'événements pendant une certaine durée qui contient l'information ?

Vous remarquerez la différence entre le concept de niveau logique et d'événement qui caractérise un changement de niveau logique.

Vous pouvez constater, dans cette liste, l'absence systématique du mot « temps » qui est flou, on lui préférera la notion d'instant et de durée. Le fait de se poser ce genre de questions permet de savoir si on doit utiliser une entrée analogique, une entrée GPIO simple, une interruption externe, une entrée de capture, une entrée de comptage externe, une entrée de réception série...

Ces phrases donnent aussi des indications sur la technique à employer pour récupérer l'information importante. On se posera la question du périphérique à mettre en œuvre et surtout s'il est nécessaire que les événements importants soient gérés par interruption ou par scrutation.

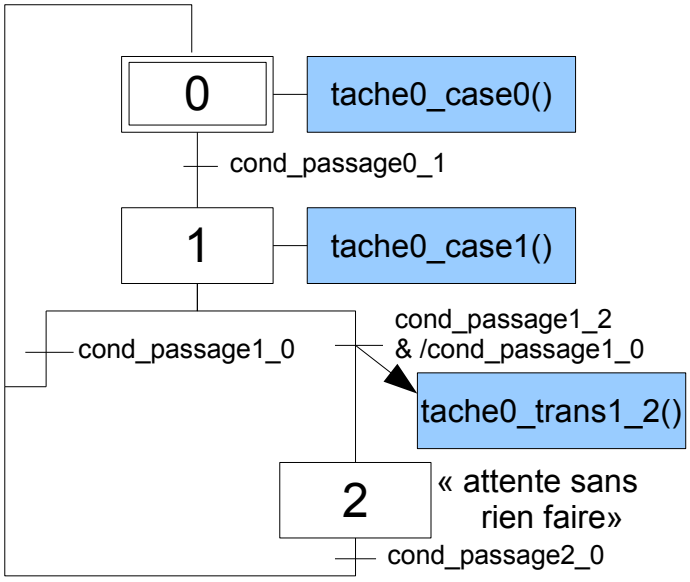
En se posant la même question sur les actions à générer, les informations que le micro-contrôleur doit transmettre, la manière dont il doit le transmettre, pour choisir les types de sorties du micro-contrôleur, on pourra ainsi déterminer si l'on doit utiliser une sortie de type analogique (convertisseur numérique analogique), sortie GPIO, PWM, sortie OUTPUT COMPARE.

Sachant que les blocs électroniques sont connectés sur un micro-contrôleur, on fera particulièrement attention à la compatibilité en matière de gamme de tension (il faut rester dans la gamme [0 3V3]).

2 Techniques de programmation utiles :

- **Machine à état pour réaliser une tâche séquentielle sans être bloquant :**

On utilise cette manière de coder uniquement dans les cas où un programme nécessiterait d'attendre des événements qui provoquerait une immobilisation dans le code. Seulement si on est confronté à ce genre de problème, on utilise cette technique de programmation. On dessine au préalable cette machine à état sous la forme d'un grafcet simple en numérotant les états, et en décrivant toutes les conditions de passage d'un état à l'autre. La variable clef pour piloter la machine à états est la variable de type unsigned char etat_mae0. Elle sera déclarée en static dans la fonction et ainsi elle ne sera visible que dans la fonction. **Une interruption est dans un organigramme indépendant, jamais connecté à l'organigramme d'un programme principal ou au grafcet d'une tâche. Les grafcets ne seront utilisés que pour les machines à états. Une machine à état peut aussi être très pratique dans une interruption pour ne pas réaliser la même chose à chaque passage dans l'interruption, les conditions de passage pouvant être alors toujours vraies, de manière qu'on change d'étape à chaque passage en interruption.**




```
void tache0(void)
{static unsigned char etat_mae0=0; //cette initialisation définit l'étape initiale (encadrée en double)
 pin_espion_gestion_tache0=1;//quand le code est fonctionnel, on met en commentaire
switch(etat_mae0)
  {   case 0: // commentaire décrivant le role de l'étape 0
        tache0_case0(); // s'il n y a pas grand chose écrire le code directement
        pin_espion_case0=1; //aide à la vérification du code sans l interrompre,
        if(cond_passage0_1) {etat_mae0=1;pin_espion_case0=0;}
   break;
   case 1 :  // commentaire décrivant le role de l'étape 1
        tache0_case1(); // s'il n y a pas grand chose écrire le code directement
        pin_espion_case1=1;
        if(cond_passage1_0) {etat_mae0=0;pin_espion_case1=0;break;}  // prioritaire  par break!
        if(cond_passage1_2) {tache0_trans1_2(); etat_mae0=2;pin_espion_case1=0;}
                        // fonction exécutée une fois seulement au changement d'étape
   break;
   case 2:  // commentaire décrivant le rôle de l'étape 2 : ici état d attente, on ne fait rien
        pin_espion_case2=1;
```

```
        if(cond_passage2_0) {etat_mae0=0;pin_espion_case2=0;}
  break ;
 }
// on rend la main au main()
 pin_espion_gestion_tache0=0;//quand le code est fonctionnel, on met en commentaire
}
```

Les conditions de passage sont parfois des événements mis à jour par les interruptions (comme par exemple un temps d'attente que l'on met à jour à la transition et qui est décompté en interruption), parfois on passe automatiquement à l'état suivant sans condition de passage, mais c'est l'exécution de la machine à état qui est conditionnée à l'apparition d'un sémaphore (voir ci dessous).
RAPPEL : LA MACHINE A ETATS N'EST UTILE :

- QUE POUR LES FONCTIONS AVEC ATTENTE D'EVENEMENTS, fonctions qu'on aurait tendance à programmer sinon avec des lignes de code comportants des boucles d'attente : while(test) ; //attente de l'arrivée de ….
- QUE POUR LES FONCTIONS OU ON EXECUTE UN MORCEAU DE CODE DIFFERENT A CHAQUE APPEL.

- **Communication par sémaphores software (l'équivalent des drapeaux d'événements que l'on trouve dans les registres xxxIR des différents périphériques)**

Le but est de déclencher des tâches à des instants bien particuliers. Il est classique de rester le moins longtemps possible en interruption, de juste récuperer une information importante, et de prévenir que cette information est disponible.

```
volatile unsigned char flag_active1 = 0; // sémaphore manipulé en IT, il doit être déclaré volatile
volatile unsigned char flag_active2 = 0; // sinon le compilateur peut décider de ne pas le tester
                                         // car jamais mis à jour dans le main => tâches non exécutées
void gere_it1() __irq
{pin_presence_it1=1 ;//pour visualiser en temps réel le passage dans l'it1
// ce qui doit être fait en urgence est fait ici
….....
flag_active1=1; //activation à chaque interruption d'une tâche via un sémaphore
if(test) {flag_active2=1;}//activation conditionnelle d'un sémaphore qui activera une tâche une fois
//acquittement de l'it1
….....
pin_presence_it1=0 ;
}
void main(void)
{
init_proc();
init_var_globales();
acquitter_autoriser_it();

while(1)
 { pin_beep_vie_main ^=1; //permet de voir que le main s execute !
  tache0(); //toujours executée
  if(flag_active1==1) {tache1();flag_active1=0;}//tache exécutée une fois après chaque IT
  if(flag_active2==1) {tache2();flag_active2=0;} // tache exécutée une fois après certaines IT
 }
}
```

- **gestion de tableau circulaire pour gestion de flux**

Le but est de permettre à deux processus travaillant à des rythmes différents de communiquer. Par exemple, une fonction doit envoyer un très gros message à rythme très lent (à cause d'une transmission lente) et on ne veut pas immobiliser le processeur dans cette tache à attendre que le périphérique soit libre. On va donc utiliser une fifo dont la taille permet le stockage d'un message complet d'un seul coup et le message est transmis octet par octet dès que possible par une autre fonction qui gère le périphérique dès qu'il est libre. Le nombre d'octets à transmettre en moyenne doit être compatible avec le périphérique et sa vitesse de transmission

```
//************************************************************
//************ variables nécessaires ****************************
//************************************************************
// le mot clef volatile est indispensable si les variables sont manipulées en interruption
        #define TAILLE_FIFO 64
        #define MASQUE_FIFO 63
        unsigned char fifo[TAILLE_FIFO] ;   // le tableau dont la taille doit etre suffisante
        volatile unsigned char pw_fifo = 0; // index d'ecriture dans la fifo
        volatile unsigned char pr_fifo = 0; // index de lecture dans la fifo
        volatile unsigned char nb_fifo_libre = TAILLE_FIFO ; // places encore libres dans la fifo

//**** ecriture brutale du message entier dans la fifo si la place le permet******************
// ex : envoi d'un message de 3 octets (imaginez plus grand, il suffit de completer cette routine)
//*****************************************************************************
void envoi_message_taille_3_octets(void)
{   if (nb_fifo_libre>2) //vérifier qu'on peut écrire le message d'un coup dans la fifo.
        {fifo[pw_fifo]='A';             //ecriture du premier octet
        // incrementation d index et rebouclage en debut de fifo (buffer circulaire)
         pw_fifo++;  if(pw_fifo>=TAILLE_FIFO) {pw_fifo=0;}
        fifo[pw_fifo]='B';              // ecriture du second octet
        pw_fifo++;  if(pw_fifo>=TAILLE_FIFO) {pw_fifo=0;} //index suivant
        fifo[pw_fifo]='C';              //ecriture du troisieme octet
        pw_fifo++; if(pw_fifo>=TAILLE_FIFO) {pw_fifo=0;}//index suivant
        nb_fifo_libre -= 3;             // la fifo a perdu en place libre ...
}       }
//*****************************************************************************
//**** expédition du message lentement des que le périphérique série est disponible *******
//*****************************************************************************
// on redéfinit le test du flag indiquant que l on peut écrire un octet dans la liaison série
// il faut remplacer par le bon registre de status et par le bon masque indiquant le bit utile
// ces define changent d un processeur à l'autre, mais me permettent d'écrire ici un code générique...
// VOUS UTILISEREZ DIRECTEMENT LES REGISTRES DU PROCESSEUR
// Status de la liaison série, le registre dans lequel on trouve le bit qui indique qu'on peut transmettre
        #define STATUS_RXTX  .....
// Masque permettant de tester le bit en question
        #define MASQUETX  (1<<.....)
// le test proprement dit
        #define TX_LIBRE  (STATUS_RXTX & MASQUETX)
// le registre dans lequel écrire pour
        #define TX_BUF_TRANSMIT   .....
// on peut transmettre si le périphérique est disponible et si la fifo contient au moins
// un octet à transmettre. Dans la boucle while(1) du main, on scrute alors
// s'il y a quelque chose à envoyer et si la place est libre.
```

```
void  main(void)
{init_proc() ;
 while(1)
        { tache0() ;//une tache toujours exécutée
          // un programme d'interruption (par exemple) décide qu'il faut envoyer  un message
           if(flag_active_envoi==1) {envoi_message_taille_3_octets(); flag_active_envoi=0;}
          // on vide la fifo dès que possible...
           if (TX_LIBRE && (nb_fifo_libre<TAILLE_FIFO) ) {envoi_octet_mes();}
        }
}

void envoi_octet_mes(void)
{
 // faire transmettre le caractere le plus ancien par le périphérique
        TX_BUF_TRANSMIT=fifo[pr_fifo];
// dire qu'une place de plus est dispo dans la fifo
        nb_fifo_libre++;
// pointer le caractere suivant en rebouclant en fin de tableau
// il faut donc incrementer l index de lecture et le remettre à zero en fin de tableau
        pr_fifo++; if (pr_fifo>=TAILLE_FIFO) {pr_fifo=0;}
// si on a la chance d'avoir une puissance de 2 comme taille, on peut utiliser un masque :
// pr_fifo &= MASQUE_FIFO ; // ex: faire un et logique avec 63 transforme le chiffre 64 en 0
// si les index sont des unsigned char et que le tableau fait 256 octets, il n'y a pas de test à faire.
}
```

3 En présentiel pour discuter d'un soucis en électronique :

- Présentez le contexte, nommez le module sur lequel vous travaillez, la fonctionnalité que vous cherchez à mettre en place, le test, la vérification que vous cherchez à mettre en œuvre.
- Venez avec un schéma complet de la chaîne électronique, pas simplement une fonction isolée, les erreurs venant souvent de la cascade des circuits et non d'un circuit seul.
- Caractérisez le problème auquel vous êtes confrontés, en expliquant à l'aide de tout document utile, en particulier des chronogrammes, la documentation des composants utilisés. Vous décrirez ce que vous pensiez observer, et la différence avec la réalité observée. Le but est de mettre en place une démarche comportant les étapes suivantes :
  - Hypothèse de l'origine du problème rencontré.
  - Proposition d'une expérience d'arbitrage pour valider, invalider cette hypothèse en tirant les conséquences de cette hypothèse (prouver qu'une idée est fausse, c'est aussi avancer).
  - Prédiction d'interprétation des résultats de l'expérience à conduire, en anticipant comment régler les appareils de mesure.
  - Réalisation de l'expérience, comparaison avec les résultats possibles prédits
  - Conclusion sur l'hypothèse, soit retour au début, soit recherche de solution.
- Pour tout ce qui est interfaçage, choix de périphérique au niveau micro-contrôleur, vous présenterez un schéma clair, qui précise l'étage d'adaptation, le chronogramme clef que vous pensez avoir à l'entrée ou la sortie du micro-contrôleur, le nom de la patte choisie, la fonctionnalité choisie (ce qui permet de déterminer la valeur à mettre dans le registre PINSEL) et vous dessinerez un bloc représentant le périphérique à mettre en œuvre.