TP1 : La station de fabrication de béton

Introduction

Tout automatisme industriel doit être évidement réalisé avec un automate proprement dit (Schneider, Siemens,...). Cependant, la conception d'un automate étant réalisée à partie de μC, il est intéressant, d'un point de vue pédagogique, d'étudier le codage d'un automate en langage C et c'est l'objet de ces Travaux Pratiques.

La maquette de Travaux Pratique modélise la fabrication de béton, par un dosage de sable et de ciment. Ce processus est modélisé par le Grafcet de la figure 1.

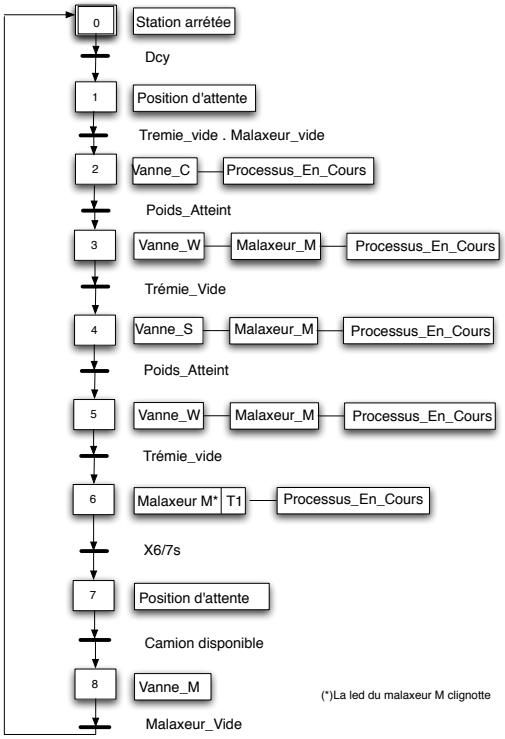


Figure 1: Grafcet

- La maquette comporte 6 sorties, des entrées vu du coté micro : $PA_0, PA_1, PA_2, PA_3, PA_4, ALARM$
  Elles correspondent aux interrupteurs de la carte. Ces derniers permettent de valider les différentes transitions du Grafcet. On utilisera des puissances de 2 dans les étiquettes car si l'on avait choisi un nombre. Par exemple considérons que l'on lit 5 sur $PINC$. $5 = 4 + 1 = 3 + 2$, il y a alors une ambiguïté sur les entrées (du micro) possibles.

- La maquette comporte 8 entrées, des sorties vues du coté micro : $PB_0, PB_1, PB_2, PB_3, PB_4, PB_5, PB_6, PB_7$
  Elles permettent d'activer des actions sur les étapes du Grafcet. Elles serviront aussi à commander les leds simulant le processus de fabrication de béton. Les étiquettes attachées à ces entrées correspondront à des nombres croissants.

1 Initialisations

Nous allons utiliser l'environnement Arduino. Sélectionner l'icône $ArduinoGE_3$. [1]

1. Positionner les options dans le menu "outils" :
   - Type de carte : NG or older
   - Processeur : Processeur $ATMEGA_8$
   - Port : choisissez le port $COM1$
   - Programmateur : $USBASP$

   Pour compiler vous utiliserez dans le menu "croquis" ou "fichier" : Téléverser avec un programmeur

2. Dans votre fichier C principal, éditer les étiquettes suivantes :

```
#define DEPART_CYCLE   1     // entrees PORTC
#define TREMIE_VIDE    2  // entrees PORTC
#define MALAXEUR_VIDE  4  // entrees PORTC
#define POIDS_ATTEINT  8  // entrees PORTC
#define CAMION_DISPO   16  // entrees PORTC
#define TEMPO          32  // entrees PORTC

#define POSITION_ATTENTE     0  //  Attention  PORTB !
#define VANNE_C              0  //  Sorties PORTD
#define VANNE_S              1  //  Sorties PORTD
#define MALAXEUR_M           3  //  Sorties PORTD
#define VANNE_W              4  //  Sorties PORTD
#define VANNE_M              5  //  Sorties PORTD
#define PROCESSUS_COURS      6  //  Sorties PORTD
#define STATION_ARRETEE      7  //  Sorties PORTD
```

3. Effectuez les câblages des sorties de la maquette sur les ports $PORTC$ et $PORTD$ de la carte μC :
   - PA_0 : Départ cycle sur PC_0 ;
   - PA_1 : Trémie vide sur PC_1 ;
   - PA_2 : Malaxeur vide sur PC_2 ;
   - PA_3 : Poids atteint sur PC_3 ;
   - PA_4 : Camion disponible sur PC_4 ;
   - $ALARM$ sur PD_2.

4. Effectuez les câblages des entrées de la maquette sur les ports $PORTD$ et $PORTB$ de la carte μC :
   - PB_0 : Position d'attente sur PB_0 ;
   - PB_1 : Vanne C sur PD_0 ;
   - PB_2 : Vanne S sur PD_1 ;
   - PB_3 : Malaxeur M sur PD_3 ;

[1] Deux versions logicielles d'Arduino cohabitent sur les PC et nous allons utiliser la 1.6.4 (Icone ArduinoGE3) pour tous les Travaux Pratique de première année.

- PB_4 : Vanne W sur PD_4 ;
- PB_5 : Vanne M sur PD_5 ;
- PB_6 : Processus en cours sur PD_6 ;
- PB_7 : Station arrêtée sur PD_7.

5. Relier la masse de la carte micro (chercher la patte GND) à celle de l'alimentation.

6. Fonction *transition* : Ecrire la fonction *etape1()*, compiler et téléverser et tester pour différentes valeurs de conditions.

```
void transition(int condition) {
  int capteur=0;
  // if (condition == TEMPO) {??}  else
  do {
    capteur = PINC;
    capteur = capteur & condition;
  } while(capteur != condition);
}

void etape0(void){PORTD=(1<<STATION_ARRETEE);}

int main(void)
{
   DDRC = 0x00;          // port C en entree
   DDRB = 0x01;          // port B : PB0 en sortie
   DDRD = 0b11111011;    // port D en sortie sauf  PD2
                         // PD2 : entree d'interruption externe
  do {
    etape0();
    transition(DEPART_CYCLE+CAMION_DISPO);
    etape1();
    transition(MALAXEUR_VIDE);
  } while(1);            // le G7 est en boucle infinie
}
```

2 Le Grafcet

- Compléter la fonction *transition* pour aussi prendre en compte la temporisation entre les étapes X_6 et X_7 à la place d'une condition logique. Le clignotement de la led $MALAXEUR\_M$ de l'étape X_6 doit être réalisé à avec une fréquence de $1hz$. La temporisation pourra être gérée par la fonction $\_delay\_ms()$.
- Compléter le programme principal pour mettre en oeuvre le Grafcet intégralement (voir la figure 1).

3 Arrêt d'urgence

Modifier le programme de façon à pouvoir traiter un arrêt d'urgence. On considèrera le signal d'alarme comme un front descendant. L'évènement lié à l'interruption se nomme $INT0\_vect$. Le type d'événement susceptible de déclencher une interruption externe se programme à l'aide des bits ISC_{01}, ISC_{00} du registre $MCUCR$. Tandis que cette interruption se valide à l'aide du bit INT_0 du registre $GICR$.

Le sous-programme d'interruption lié à l'arrêt d'urgence devra sauvegarder l'état des leds puis il devra faire clignoter l'ensemble des leds commandables pendant 5 secondes, puis reprendre le Grafcet là, où il a été interrompu avec les leds qui seront restaurées. Le délai de clignotement sera gérée par la fonction $\_delay\_ms()$.

4 Processus En cours

Nous allons modifier la prise en compte de la led PROCESSUS_EN_COURS. Celle-ci va dorénavant, clignoter tout au long du déroulement des étapes X_2, X_3, X_4, X_5, X_6. Pour le délai, que vous fixerez avec une valeur quelconque, du clignotement vous utiliserez l'interruption de débordement du timer 0. En plus de l'usage du pré-diviseur de fréquence, vous devrez re-diviser la fréquence de clignotement à l'aide d'une variable statique dans le sous-programme d'*interruption*. Vous activerez cette interruption, à l'aide de son bit de masquage, à l'étape 2 et vous la désactiverez à l'étape 7.

Rappels :

- Le bit de masquage se nomme $TOIE_0$ et appartient au registre $TIMSK$.
- La valeur initiale de comptage du timer 0 est faite dans le registre $TCNT_0$.
- $TCCR_0$ contrôle le pré-diviseur de fréquence.
- L'interruption d'Overflow du timer zéro, se nomme $TIMER0\_OVF\_vect$.

5 Compte rendu

- Une introduction (0,5 page max) décrit le but du TP.
- Chaque question du TP doit donner lieu à un texte décrivant le test et les résultats d'exécution accompagnés quand cela est nécessaire par un algorithme ou un organigramme. Vous préciserez comment vous avez testé votre code et vous donnerez sur Uncloud quand cela est pertinents des extraits vidéos correspondant à des résultats d'exécution.
- La conclusion (1 page max) se définit comme une recherche d'une application industrielle des notions abordées en TP. Cette conclusion doit faire clairement apparaître une recherche documentaire.
- En annexe, donnez votre code commenté

TP2 : Commande d'un Moteur Pas à Pas

Introduction

Ce moteur comporte 48 pas correspondant à des secteurs angulaires identifiés sur une corolle. La maquette supportant le moteur pas à pas comprend :

- 4 bits de commande ;
- Une borne COMPT qui passe à *un* dès que l'optocoupleur détecte la position 1 du moteur pas à pas. Cet état est mémorisé grâce à une bascule D (74HCT74). Non utilisée dans le TP.
- La borne Reset (a coté de compt) permettant de remettre à zéro la bascule D. Non utilisée dans le TP.
- Un interrupteur marche/arrêt permettant de couper l'alimentation du moteur.
- Un potard permet de délivre une tension variable $V \in [0, 5]$ entre la borne se trouvant près de l'optocoupleur et la masse. On trouve aussi un connecteur d'alimentation et de masse.

Dans ce sujet on abordera la commande du moteur pas à pas.

1 Initialisations

1. Effectuer les câblages suivant :
   - PB_0 sur la borne 0 ;
   - PB_1 sur la borne 1 ;
   - PB_2 sur la borne 2 ;
   - PB_3 sur la borne 3 ;
   - Brancher un fil volant sur PB_5;
2. Relier la masse de la carte à celle de l'alimentation.
3. Relier une sonde d'oscilloscope sur une des commandes de bobines du moteur.

2 Commande moteur

2.1 mondelai

Ecrire une fonction *mondelai* qui réalise un délai qui est un multiple entier de 1 milliseconde. Pour cela vous appellerez la fonction $\_delay\_ms(1)$ dans une boucle for.

2.2 Mode 1 : Pas entier, une bobine

En utilisant *mondelai*, écrire une fonction appelée dans le *main* pour réaliser une fonction permettant de faire tourner le moteur infiniment, en pas entier à vitesse constante dans le sens horaire de 48 pas. Tous les 4 pas commuter le bit PD_0. Faire varier le délai et observer l'image de la vitesse sur bit PD_0 pour obtenir la vitesse maximale à l'oscilloscope. Calculer la vitesse maximale obtenue en tours/min.

2.3 Mode 2 : Pas entier, deux bobines

En utilisant *mondelai*, écrire une fonction appellée dans le *main* pour réaliser une fonction permettant de faire tourner le moteur infiniment, en pas entier, deux bobines par deux bobines, à vitesse constante dans le sens horaire de 48 pas. Tous les 4 pas commuter le bit PD_0. Faire varier le délai et observer l'image de la vitesse sur bit PD_0 pour obtenir la vitesse maximale à l'oscilloscope. Calculer la vitesse maximale obtenue en tours/min.

2.4 Mode 3 : demipas

En utilisant *mondelai*, écrire une fonction appelée dans le *main* pour réaliser une fonction permettant de faire tourner le moteur de infiniment, en demi-pas dans le sens trigonométrique de 48 pas. Tous les 8 demi-pas (4 pas) commuter le bit PD_0. Faire varier le délai et observer l'image de la vitesse sur bit PD_0 pour obtenir la vitesse maximale à l'oscilloscope. Calculer la vitesse maximale obtenue en tours/min.

2.5 Modes et vitesse maximale

- Quel est le mode qui permet d'atteindre la plus grande vitesse maximale ?
- Trouver une explication physique de la vélocité d'un mode par rapport aux autres ?

2.6 changement de mode

A l'aide du bit PB_5 défini en entrée, ré-écrire vos fonctions pour permettre de changer de mode : Pas entier sens horaire vers le mode demi-pas sens trigonométrique (et vice-versa). Pour que la prise en compte du changement de sens puisse se faire entre deux pas, on va réaliser un test de PB_5 dans la boucle de commande des pas et on sortira en case de test positif (c.a.d. si PB_5 indique qu'il faut changer de sens) de cette boucle par l'instruction break.

2.7 Convertisseur Analogique Numérique

Relier la borne du potard à PC_0.

- Initialiser le CAN en mode en 8 bits et utiliser le mode free-running sans timer ni interruption.
- Utiliser la valeur de conversion pour déterminer le délai qui réglera la vitesse du moteur.

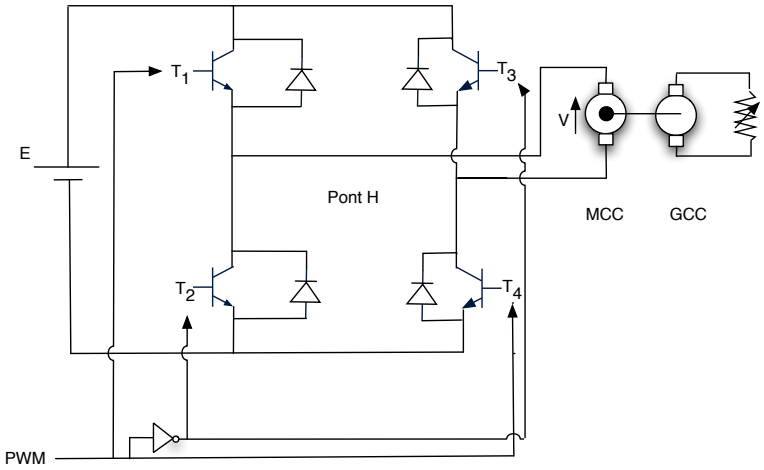
3 Compte rendu

- Une introduction (0,5 page max) décrit le but du Travaux Pratique.
- Chaque question du Travaux Pratique doit donner lieu à un texte décrivant le test et les résultats d'exécution accompagnés quand cela est nécessaire par un algorithme ou un organigramme. Vous préciserez comment vous avez testé votre code (résultats visuels, éventuellement mesures à l'oscilloscope) et vous donnerez quand cela est possible des extraits quantitatifs ou visuels (photos, gifs) des résultats d'exécution.
- Conclusion : Rappels des éléments vus dans le TP.
- Recherche documentaires sur les questions suivantes : Quelles sont les différents types de moteurs électriques ? Quels sont les avantages et inconvénients des Moteur Pas à Pas ? ? Quels sont les caractéristiques limites des Moteur Pas à Pas ?
- En annexe, donnez votre code commenté

TP3 : Commande d'une Machine courant continu

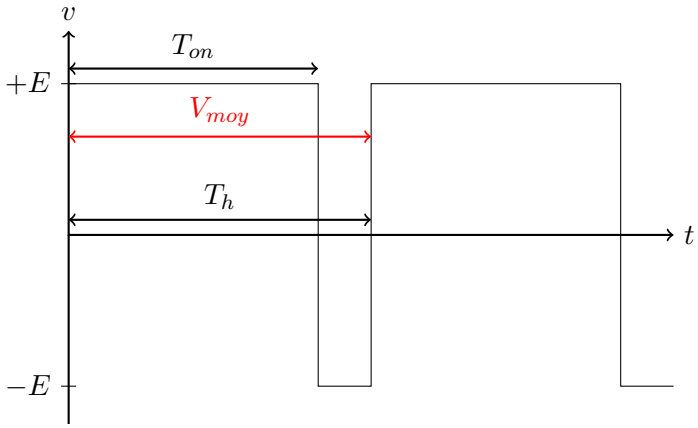
Introduction

La figure 1 décrit une Machine à Courant Continu(MCC) commandée par le biais d'un hacheur. Une génératrice est couplée au moteur, elle se comporte comme un frein grâce à un ensemble de résistances variables. Le schéma de principe du hacheur est donné dans la figure suivante :



On a 4 transistors qui sont appairés en 2 couples (T_1, T_4) et (T_2, T_3). Ces deux couples sont reliés à une même ligne de commande PWM, mais le second couple à travers une porte inverseuse (74LS04). Donc suivant le niveau logique sur cette ligne on a soit (T_1, T_4) passant soit (T_2, T_3) passant. Si (T_1, T_4) sont passant alors (T_2, T_3) sont bloqués et on a une tension de $+E$ aux bornes du moteur, sinon on a $-E$.

L'objet de ces Travaux Pratique est de réaliser une commande en boucle ouverte de la MCC. Soit T_h une période appelée période de hachage, le rapport cyclique $\frac{T_{ON}}{T_h}$ permet la commande de la MCC.
En hachant la tension de commande E, la PWM permet de produire une tension moyenne V_{moy}, que l'on peut calculer en faisant la somme des aires, comme le montre la figure suivante :



La somme des aires pour une période T_h produit le calcul de l'intégrale suivante :

$$V_{moy} = \frac{1}{T_h}\left(\int_0^{T_{on}} E.dt + \int_{T_{on}}^{T_h} -E.dt\right)$$

Prédétermination

- Retrouver les expressions littérales de T_{on} et de $\frac{T_{on}}{T_h}$ en fonction de V_{moy}, E, T_h et de V_{moy} en fonction de T_{on}, T_h, E.
- Déterminer les valeurs numériques de $\frac{T_{on}}{T_h}$ pour obtenir les tensions moyennes de 28 v et -25 v sachant que $E = 35v$.
- Etablir dans une feuille de calcul (libreOffice) la formule de cours, qui lie OCR_1A, F_{hash}, N et F_{clkio}. N ne prenant que cinq valeurs $(1, 8, 64, 256, 1024)$, trouver tous les couples (N,OCR_1A) valides pour établir une fréquence de hachage à 15 khz. Quel est le meilleur couple et pourquoi ?
- Etablir les valeurs de $OCR1_B$ pour $28v$ et $-25v$ en considérant que $N = 1$ et que $f_h = 15$ kh.

Branchements

- Brancher PB_2 sur l'entrée numérique de la maquette ;
- Brancher la masse de la carte $ATMEGA_8$ à une masse de la maquette.
- Placer le commutateur sur entrée numérique (maquette éteinte).

1 Commande de la MCC

1.1 Profils de tensions

On veut dans un premier temps commander la MCC avec une commande avec le profil de tensions suivant :

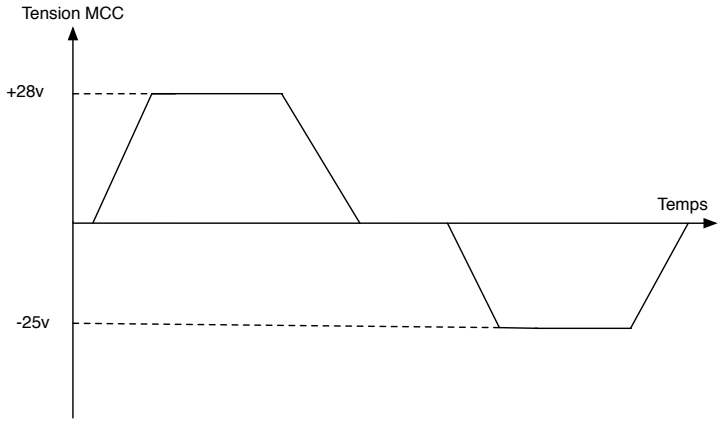


Figure 2: Profil de vitesse

- Cette courbe de tensions permet de modéliser une accélération dont vous fixerez la pente par la fonction _delay_ms, suivie d'une vitesse constante suivie d'une décélération et un petit palier à vitesse constante. Une forme symétrique avec des tensions négatives termine le cycle.
- Pour cela, vous utiliserez le $TIMER_1$ en mode "phase et fréquence correcte", on utilisera la patte $OC1_B$ pour générer la PWM. On appliquera une fréquence de hachage de $15khz$. Les deux plateaux de la figure 1.1 correspondent respectivement aux tensions $28v$ et $-25v$.

- Visualisez la vitesse et le courant à l'oscilloscope. Appliquez une charge et déterminer comment varient la vitesse et le courant.

1.2 Commande par un potard analogique

On va commander maintenant la MCC à l'aide du seul potard de la carte μC et à l'aide du module de conversion analogique. Le potard devra commander le moteur entre les tensions $[-35, +35]$. Ecrivez trois versions de la conversion en utilisant les modes suivants :

1. Utiliser une conversion sur 10 bits en mode interruptif échantillonné : Attention que la relance de la conversion soit bien faite dans l'IT de débordement du Timer et non pas dans l'IT du Can. On utilisera la fréquence du timer $f_{ech} = 15khz$ pour échantillonner l'ADC. On réalise donc 2 interruptions : L'interruption de débordement du Timer 1 pour lancer la conversion et l'interruption de fin de conversion qui permettra de lire le résultat de la conversion. La valeur lue servira à définir la variation de vitesse du moteur.

2. Utiliser une conversion sur 8 bits, sans interruption, en mode attente active : Boucle d'attente scrutative du bit $ADCS$ (appartenant au registre $ADCSRA$) dans le programme principal.

Pour chacun des 2 modes on écrira une fonction $Init\_CAN$ dédiée.

1.3 Compte rendu

Une introduction (1 page max) décrit le but du TP. La conclusion (2 pages max) se définit comme une recherche d'une application industrielle des notions abordées en TP. Faites dans votre conclusion une recherche sur les différents types de moteurs électriques. Dans votre compte-rendu vous préciserez comment vous avez testé votre code. Vous donnerez des résultats visuels des courbes obtenues à l'oscilloscope (mode run) pour différentes charges.