Pensivy Romain
Noel Hugo

TP commande MCC et asservissement

lien git: https://github.com/HugoNoel/commande-et-asservissement.git

Console UART

Nous avons réalisé une console UART.
Nous utilisons notamment les interruptions sur UART pour reconnaître l’écriture au clavier.
Cette console gère l’écriture, l'exécution d’une commande avec la touche ENTER ainsi que la suppression de caractère avec la touche DELETE.
Les commandes implémentées sont les suivantes :

- pinout : repérer les broche du nucléo connectées au hacheur
- speed=xx : mettre la vitesse à la valeur xx
- start : lancer les PWM et le hacheur pour faire tourner le moteur
- stop : arrêter le moteur

Elles seront détaillées par la suite.
Toute autre commande sera indiquée comme non reconnue car non implémentée.

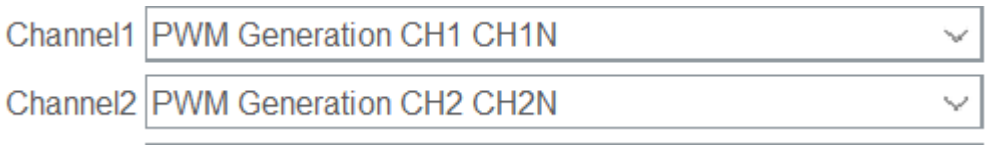
Commande MCC basique

Notre objectif est de générer 4 PWM en complémentaire décalée. Nous devons respecter le cahier des charges suivant:

- Fréquence de la PWM : 16kHz
- Temps mort minimum : 2us
- Résolution minimum : 10 bits.

Génération des PWM

Dans un premier temps, sur STM32CubeIDE nous activons le timer 1. Nous utiliserons le channel 1 en mode “PWM Generation CH1 CH2N” et le channel 2 en mode “PWM Generation CH1 CH2N”.



Une résolution de 10 bits (=1024) impose l’ARR à 1023. On doit donc déterminer le prescaler .

Sachant que $f_{PWM} = \frac{fsys}{(1+PRE).(1+ARR)} \Leftrightarrow PRE = \frac{fsys}{f_{PWM}.(1+ARR)} - 1 = \frac{170.10^6}{16.10^3 \times 10244}$ -1 $\simeq 9.4$

Sachant que l’on préférera une fréquence de PWM inférieur ou égale à 16kHz, on choisit de prendre 10 comme valeur de prescaler.

Pour le temps mort, nous devons nous référer au TIMx break and dead-time register du reference manual. La valeur que l’on doit écrire dans STM32CubeIDE est DTG[7:0].

Bits 7:0 **DTG[7:0]**: Dead-time generator setup
This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.
DTG[7:5]=0xx => DT=DTG[7:0]x t_{dtg} with $t_{dtg}=t_{DTS}$.
DTG[7:5]=10x => DT=(64+DTG[5:0])xt_{dtg} with $T_{dtg}=2xt_{DTS}$.
DTG[7:5]=110 => DT=(32+DTG[4:0])xt_{dtg} with $T_{dtg}=8xt_{DTS}$.
DTG[7:5]=111 => DT=(32+DTG[4:0])xt_{dtg} with $T_{dtg}=16xt_{DTS}$.

avec $T_{DTG} = \frac{1}{f_{SYS}} = \frac{1}{170.10^6}$
On veut donc que DT soit égal à 2us ce qui n’est possible que pour DTG[7:5]=110.

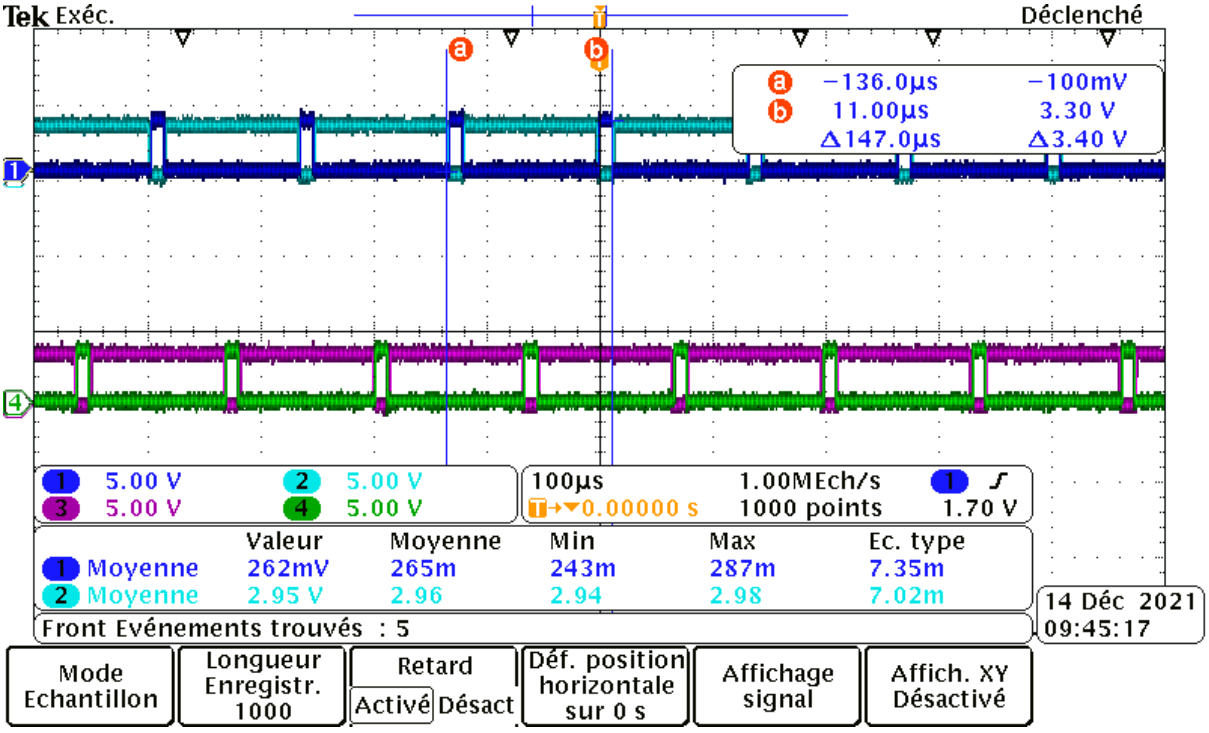
Soit $2.10^{-6} = (32 + DTG[4:0]) \times \frac{8}{170.10^6} \Leftrightarrow DTG[4:0] = \frac{2.10^{-6} \times 170.10^6}{8} - 32 = 10.5$

Sachant que l'on préférera un temps mort plus grand ou égal que 2 us, on choisit DTG[4:0] = 11 soit en binaire → 01011
Donc DTG[7:0]= 1100 1011 = 203.

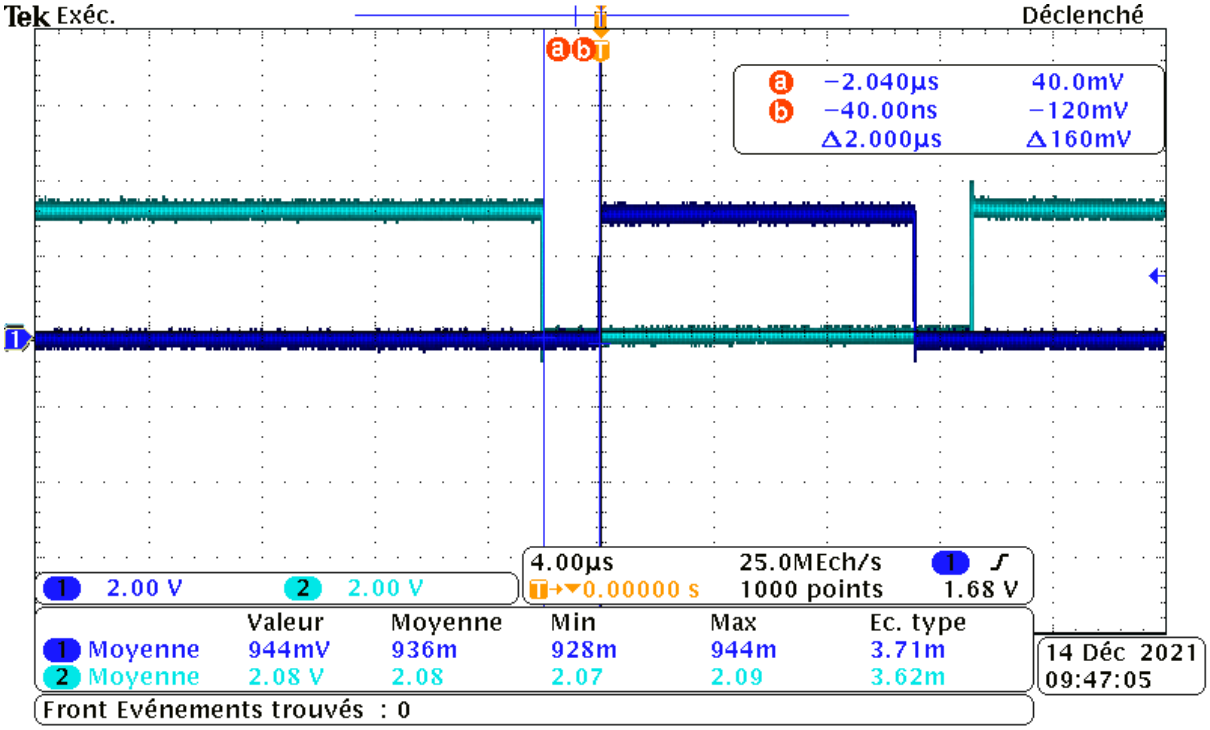
Dead Time 203

Affichage

Après avoir écrit le code pour générer les PWM en utilisant les paramètres précédant, nous les affichons



Nous obtenons au dessus, 4 PWM cohérentes. Elles forment deux paires de complémentaires et ces paires sont déphasées d'une demi-période.



En zoomant sur une paire, nous lisons un dead time de 2us, ce qui valide notre configuration.

Pinout

Rôle	Pin microcontroleur	Broche arduino	Pin hacheur
Start	PA4	A2	33→Fault Reset Command
CH1	PC0	A5	13→Red Phase Top
CH1N	PA7	D11	31→Red Phase Bottom
CH2	PC1	A4	12→Yellow Phase Top
CH2N	PB0	A3	30→Yellow Phase Bottom
+5V		5V	19→Digital +5V
GND		GND	18→ Digital GND

Le hacheur doit être alimenté en 48V. Cette alimentation n'est pas réversible. Lors d'un freinage, elle ne pourra pas encaisser le courant pouvant être de l'ordre de l'ampère. Pour cela, on met en parallèle une résistance qui consommera ce courant.

On doit réaliser séquence d'allumage afin de reset le système pour ne pas endommager le système. Pour cela il faut mettre le pin 33 "ISO_RESET" à l'état haut pendant au moins 2 us puis le remettre à l'état bas.

Commande

"start":

Cette commande permet d'allumer l'étage de puissance et affiche sur le terminal "Power ON"

```
if ((strncmp(Commande,START,i)==0) && (i==strlen(START))){
    printf("\r\nPower ON\r\n");
    Start();
    commande_existe=1;
}
```

Pour démarrer le système:

```
void Start(){
    TIM1->CCR1 = (int) ((TIM1->ARR)*50)/100;
    TIM1->CCR2 = (TIM1->ARR) - TIM1->CCR1;
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_Delay(1);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
    ON=1;
}
```

On fixe le rapport cyclique sur CH1. Les autres rapport étant déduit de celui-ci

On démarre les 4 PWM

procédure de start

"stop":

Cette commande sert à éteindre l'étage de puissance et affiche sur le terminal "Power OFF"

```
if ((strncmp(Commande,STOP,i)==0) && (i==strlen(STOP))){
    printf("\r\nPower OFF\r\n");
    Stop();
    commande_existe=1;
}
```

Pour arrêter le système:

```
void Stop(){
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
    HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_1);
    HAL_TIMEx_PWMN_Stop(&htim1, TIM_CHANNEL_2);
    ON=0;
}
```

On arrête les 4 PWM

“help”:

Cette commande permet d’afficher sur le terminal toutes les commandes disponibles.

“pinout”:

Cette commande permet d’afficher sur le terminal le branchement des pins de la carte vers le hacheur.

Bouton exti:

On peut générer le start ou stop en appuyant sur le bouton bleu grâce à l’interruption EXTI. Dans le handler de l’interruption on ne fais que changer la valeur de la variable global exti_it=1 afin de savoir quand est ce que le bouton a été pressé:

```
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */
    exti_it=1;
  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
```

On traite ensuite dans la boucle while l’interruption:

```
if (exti_it==1){
    if(ON==0){
        printf("\r\nPower ON\r\n");
        Start();
    }else{
        printf("\r\nPower OFF\r\n");
        Stop();
    }
    i=0;
    Ligne_Init(&huart2);
    exti_it=0;
}
```

La variable global ON permet de savoir si le système est en marche (ON=1) ou à l'arrêt (ON=0). Si le système est en marche, on le stop et vice versa.

“speed=xx”

Cette commande permet de régler la vitesse du moteur en écrivant le rapport cyclique souhaité. “speed=50” réglera le rapport cyclique à 50% et donc le moteur sera à l'arrêt. Entre 50 et 100 le moteur tournera dans un sens et entre 50 et 100 dans l’autre sens.

Pour détecter la commande, on ne regardera que les 6 premiers caractères afin de pouvoir réaliser la comparaison.

```
if (strncmp(Commande,SPEED,6)==0){
    Set_SPEED(i);
    commande_existe=1;
}
```

On va récupérer la valeur de la vitesse dans les autres caractères et réaliser la conversion char vers int grâce à la fonction atoi.

```
void Set_SPEED(int i){
    int length = i - strlen(SPEED);

    char valeur[length];
    for (int j=0; j<length; j++){
        valeur[j]=Commande[6+j];
    }

    int a = atoi(valeur);
```

La variable a contient la nouvelle valeur de vitesse.
Tout d'abord, on vérifie que la vitesse est valable c'est à dire comprise entre 0 et 100. Si ce n’est pas le cas, on affiche sur l'écran que la vitesse acquise n’est pas bonne.

```
if(a>0 || a<100){
```

changement de vitesse }

```
else{
    printf("La valeur de vitesse n'est pas valable. Elle doit être comprise entre 1 et 99.=%d\r\n", a);
}
```

On a remarqué que si on veut mettre une vitesse importante d’un coup, le moteur se bloque. En effet, faire cela provoque un appel de courant trop important. Sachant que le hacheur est protégé, il va se bloquer et le moteur se bloquera. Il faut donc faire un traitement pour éviter cet appel de courant trop important.
Pour une différence positive entre la nouvelle commande de vitesse et la vitesse actuelle de plus de 5%, on fera évoluer la vitesse de +5% et on ajoute un delay pour ne pas générer d’appel de courant trop important. On modifie la vitesse en modifiant directement les registres CCR.

```
if(a-V>5){
 int n = (a-V)/5;
 int inter=V;


 for (int j=0; j<n+1; j++){
   inter=inter+5;
   TIM1->CCR1 = (int) ((TIM1->ARR)*inter)/100;
   TIM1->CCR2 = (TIM1->ARR) - TIM1->CCR1;

   HAL_Delay(150);
   }
}
```

avec **V** la commande vitesse actuelle
inter la vitesse en temps réel du moteur
a la nouvelle commande moteur

Lorsque cet écart est plus faible que 5%, on peut appliquer la nouvelle commande de vitesse directement.

```
TIM1->CCR1 = (int) ((TIM1->ARR)*a)/100;
TIM1->CCR2 = (TIM1->ARR) - TIM1->CCR1;
printf("\r\nCCR1=%d\r\n",(int)TIM1->CCR1);
printf("\r\nCCR2=%d\r\n",(int)TIM1->CCR2);

V=a;
```

On modifie la variable globale V, la commande vitesse actuelle, qui prend comme valeur la nouvelle valeur de vitesse que l'on vient d'attribuer.

Pour une différence négative entre la nouvelle commande de vitesse et la vitesse actuelle de plus de 5%, on fera évoluer la vitesse de -5% et on ajoute le delay.

```
else if(a-V<5){
    int n = (a-V)/5;
    int inter=V;
    for (int j=0; j<n+1; j++){
                inter=inter-5;
                TIM1->CCR1 = (int) ((TIM1->ARR)*inter)/100;
                TIM1->CCR2 = (TIM1->ARR) - TIM1->CCR1;

                HAL_Delay(150);
                }
}
```

Si la commande entrée ne correspond à aucune déja existante, on envoie le message suivant:

```
if(commande_existe==0){
    printf("\r\nCette commande n'existe pas\r\n");
}
```

Mesure de Vitesse et de courant

Mesure de vitesse

PA0 et PA1 sont associés au timer 2 mis en mode encodeur. A chaque pas de codeur, le timer s'incrémente. Pour une rotation dans le sens inverse, le timer se décrémente.

Pinout

Rôle	Pin microcontroleur	Broche arduino	Pin hacheur
Encodeur phase A	PA0	A1	A
Encodeur phase B	PA1	A0	B

En lisant la valeur de CNT de TIM2 nous connaissons directement la valeur de la vitesse en pulse par seconde. Pour déterminer cette valeur en tour/s il suffit de compter le nombre de pulse par tour.
Nous avons pris du temps pour la détermination de la vitesse lors du TP2. En effet, nous avons été retardés pendant cette séance car notre codeur incrémental était défectueux. Nous ne l'avons remarqué que tardivement. Celui-ci ne générait pas des signaux en forme de créneaux mais plus des raies d'impulsion. Cela avait pour conséquence de perturber l'incrémentation du timer en mode encodeur, à tel point que la valeur de vitesse lue à chaque seconde pouvait sembler complètement aléatoire.

Nous avons donc dû changer de moteur.
On ajoute alors la commande pour avoir la vitesse du moteur.

"get_speed"

Cette commande permet d'afficher sur le terminal la vitesse actuelle du moteur.

```
if ((strncmp(Commande,GET_SPEED,i)==0) && (i==strlen(GET_SPEED))){
    Get_Speed();
    commande_existe=1;
}
```

Afin de déterminer la vitesse du moteur, on fixe la valeur du compteur à ARR/2 pour ne pas avoir de valeur négative. On stocke cette valeur dans une variable. Après un délais, le nombre de tick effectué est la valeur du compteur moins la valeur de départ du compteur.

```
void Get_Speed(){
    int deb= (TIM2->ARR)/2;
    TIM2->CNT=deb;
    HAL_Delay(250);
    vitesse=((int)(TIM2->CNT)-deb)/256;

    printf("La vitesse est %d tour/sec\r\n", vitesse);
    //Ligne_Init(&huart2);
}
```

On sait qu' un tour du moteur correspond à 1024 ticks. Pour avoir la vitesse en nombre de tour par seconde, on divise le nombre de ticks par 1024. Sachant qu'on comptait le nombre de ticks pendants un quart de seconde, on multiplie ce résultat par 4. Ce qui ramène à seulement divisé par 256.