ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

Travaux Pratique :

Commande numérique directe de dispositifs

MAGALINGAM Riot & COUTANSON Florent

10/01/2022

Table des matières

1. Objectifs

- Réaliser une interface de type "console" pour commande le hacheur d'un moteur
- Réaliser la commande des 4 transistors du hacheur en commande complémentaire décalée
- Faire l'acquisition des différents capteurs
- Réaliser l'asservissement en temps réel

2. Console UART

Via le port USB de la carte STM32G431, nous souhaitons avoir une interface permettant d'avoir des informations sur l'état du moteur et l'envoie de commandes. La console Uart permettra une communication utilisateur/système.

- ***2.1. Paramètres de la liaison UART***

Dans un premier temps nous allons paramétrer la liaison UART de la carte STM32-G431RB par l'assistant de STM32 Cube ide.

On réalise un test de bon fonctionnement de l'envoie des données en envoyant un **tick** toutes les secondes, avec la fonction ***HAL_Delay(1000).***

```
const uint8_t tick[]="tick\r\n";
```

```
while (1)
{
    HAL_UART_Transmit(&huart2, tick, sizeof(tick), HAL_MAX_DELAY);
    HAL_Delay(1000);
```

On utilise la fonction Hal Transmit pour envoyer les données.

La liaison uart choisie étant la Uart2, il faut renseigner l'adresse associée, ainsi que la chaine de caractères à afficher et sa longueur.

En testant le code dans la console on a bien le résultat escompté.

```
UART_COM7 (CONNECTED)
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
tick
```

En effet, on remarque que nous avons notre tick qui s'affiche toutes les secondes.

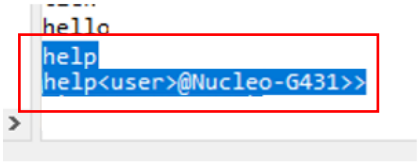
- ***2.2. Affichage de la console***

Dans cette partie, nous allons réaliser l'affichage d'une console uart. Nous avons implanté les fonctions et les variables suivantes :

```
const uint8_t starting[]="hello";
const uint8_t prompt[]="<user>@Nucleo-G431>>";
```

```
HAL_UART_Transmit(&huart2, help, sizeof(help), HAL_MAX_DELAY);
HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);
```

Nous pouvons donc visualiser les messages sur la console à chaque fois que celle-ci est lancée.



- ***2.3. Réalisation de l'écho***

Lors de l'envoie de donnée par l'interface UART, nous ne voyons pas les caractères tapés apparaitre. Il faut faire en sorte de pouvoir voir sur la console chaque caractère que nous envoyons. On cherche donc à réaliser une fonction d'écho sur la STM32 qui renvoie à l'utilisateur les caractères reçus un par un. Nous ne devons pas réaliser, de traitement dans la fonction d'interruption.

Nous implémentons, les variables suivantes :

```
uint8_t uart_rx_buffer[UART_RX_BUFFER_SIZE];
uint8_t uart_tx_buffer[UART_TX_BUFFER_SIZE];
uint8_t uart_cmd_buffer[CMD_BUFFER_SIZE];
uint8_t cmd[CMD_BUFFER_SIZE];
int idx = 0;
int it_uart_rx=0;
```

```
if(it_uart_rx==1){
    // Si validation, it_uart_rx == 1 quand l'utilisateur envoie un caractère
    //ce qui provoque une interruption
    //it_uart_rx = 0 quand on est en attente de l'envoie d'un caractère

}
// Réception d'un nouveau caractère
else{
        cmd[idx] = uart_rx_buffer[0];
        idx++;
        uart_tx_buffer[0]=uart_rx_buffer[0];


HAL_UART_Receive_IT(&huart2, uart_rx_buffer, 1);
it_uart_rx = 0;
```

3. Commande MCC basique

Commande de MCC - niveau basique

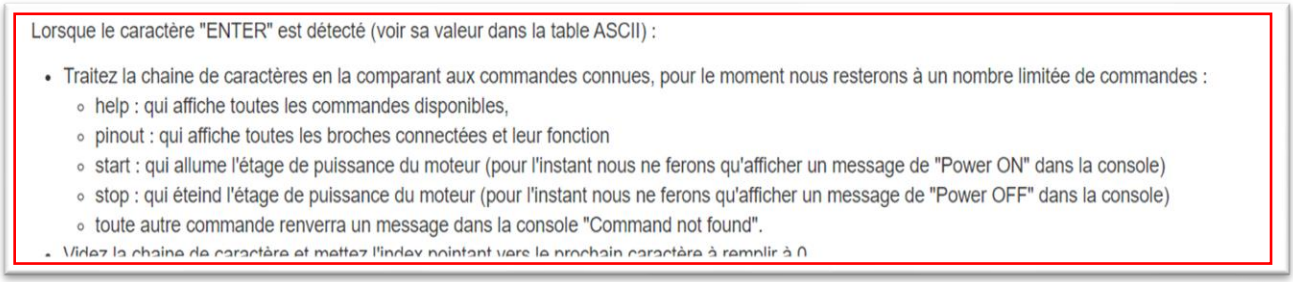
Objectifs :

- Générer 4 PWM en complémentaire décalée pour contrôler en boucle ouverte le moteur en respectant le cahier des charges,
- Inclure le temps mort,
- Vérifier les signaux de commande à l'oscilloscope,
- Prendre en main le hacheur,
- Câbler correctement la STM32 au hacheur
- Générer le signal de commande "start" en fonction de la datasheet
- Faire un premier essai de commande moteur

- ***3.1. Commandes accessibles & création d'un menu***

Dans cette partie, nous devons pouvoir taper des commandes pour activer les actions définies par le cahier des charges. L'objectif, est de garder l'***Echo*** actif lors de la réception des caractères. Il faut donc stocker les données obtenues dans une variable de type ***char,*** avec un pointeur qui s'incrémentera à chaque caractère reçu. Il est important de savoir que notre fonction ***écho***, s'intitule ***IT***.

Voici les instructions données par le cahier des charges :


Lorsque le caractère "ENTER" est détecté (voir sa valeur dans la table ASCII) :

- Traitez la chaine de caractères en la comparant aux commandes connues, pour le moment nous resterons à un nombre limitée de commandes :
  - help : qui affiche toutes les commandes disponibles,
  - pinout : qui affiche toutes les broches connectées et leur fonction
  - start : qui allume l'étage de puissance du moteur (pour l'instant nous ne ferons qu'afficher un message de "Power ON" dans la console)
  - stop : qui éteind l'étage de puissance du moteur (pour l'instant nous ne ferons qu'afficher un message de "Power OFF" dans la console)
  - toute autre commande renverra un message dans la console "Command not found".
- Videz la chaine de caractère et mettez l'index pointant vers le prochain caractère à remplir à 0

Figure 1 : Cahier des charges

On définit, les fonctions de la manière suivante :

```
/* USER CODE BEGIN PV */
const uint8_t starting[]="hello";
const uint8_t prompt[]="<user>@Nucleo-G431>>";
const uint8_t new_line[]="\r\n";
const uint8_t tick[]="tick\r\n";
char input[32];
int idxCmd;
const uint8_t help[] = "pinout, starting, start, stop, plus, moins, speed";
const uint8_t pinout[] = " PWM: Yellow phase top pin 12<=>Pa8, Red phase top pin :
const uint8_t start[] = "on";
const uint8_t stop[] = "off";
const uint8_t not_found[] = "error";
```

La commande strncmp retourne le nombre de caractères qui diffèrent de l'entrée, cette fonction nous permet donc de programmer certaines commandes qui seront reconnues lorsqu'elles seront correctement entrées par l'utilisateur.

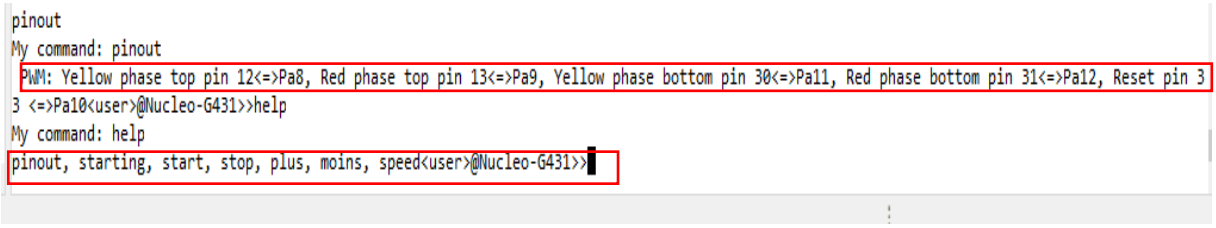
```
HAL_UART_Transmit(&huart2, new_line, sizeof(new_line), HAL_MAX_DELAY);
HAL_UART_Transmit(&huart2, "My command: ", 13, HAL_MAX_DELAY);
HAL_UART_Transmit(&huart2, cmd, idx, HAL_MAX_DELAY);
idx=0;
HAL_UART_Transmit(&huart2, new_line, sizeof(new_line), HAL_MAX_DELAY);

if(strncmp(cmd,"help",4)==0){
    HAL_UART_Transmit(&huart2, help, sizeof(help), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);

}
else if (strncmp(cmd,"pinout",6)==0){
    HAL_UART_Transmit(&huart2, pinout, sizeof(pinout), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);

}
else if (strncmp(cmd,"start",4)==0){
    HAL_UART_Transmit(&huart2, power_on, sizeof(power_on), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);
    HAL_GPIO_WritePin(Reset_GPIO_Port, Reset_Pin, SET);
    HAL_Delay(1);
    HAL_GPIO_WritePin(Reset_GPIO_Port, Reset_Pin, RESET);
    button_it=0;
}
else if(strncmp(cmd,"stop",4)==0){
    TIM1->CCR1=(TIM1->ARR)/2;
    TIM1->CCR2=(TIM1->ARR)/2;
    HAL_UART_Transmit(&huart2, power_off, sizeof(power_off), HAL_MAX_DELAY)
    HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);
```

Voici un extrait de notre console :



Le tableau ci-dessous, résume les différentes fonctions :

Fonction	Description
Help	Affiche toutes les commandes disponibles
Start	ON de l'étage de puissance
Stop	OFF de l'étage de puissance et affiche sur le terminal
Pinout	Affichage le branchement des pins de la carte
Not_found	Affiche un message d'erreur si les caractères sont différents des caractères définies

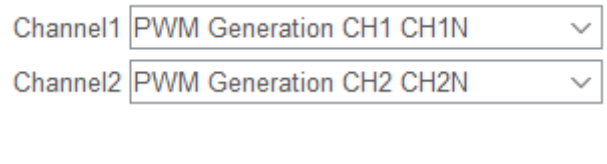
- **3.2. Génération de 4 PWM**

L'objectif est de générer 4 PWM, afin de commander le moteur MCC. Les 4 signaux seront générés via ***TIM1***.

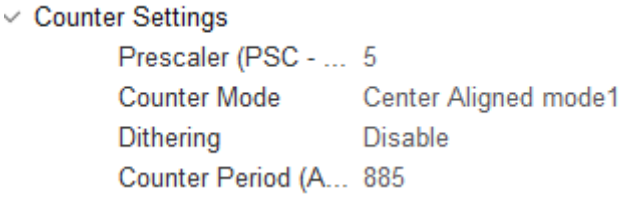
Cahier des charges :

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

L'assistant Stm 32 cube ide nous permet de choisir judicieusement nos paramètres pour obtenir une PWM correspondant au critères ci-dessus.



Pin Na...	Signal on ...
PA8	TIM1_CH1
PA9	TIM1_CH2
PA11	TIM1_CH1N
PA12	TIM1_CH2N



$$f=\frac{170*10^6}{2*(ARR+1)*(PSC+1} = 16kHz$$

le x2 provient du fait qu'on est en center aligned mode donc on compte jusqu'à ARR+1 et on décompte jusqu'à 0, la résolution est légèrement inférieure à 10 bit = 1024.

Le code ci-dessous nous permet de changer la valeur de CCR et donc choisir le rapport cyclique de notre pwm.

```
TIM1->CCR1 = val;
TIM1->CCR2 = TIM1->ARR - val;
```

Cela dit, il n'est pas très aisé de choisir le rapport cyclique ainsi, il faudrait ramener cela de 0 à 100 :

```
TIM1->CCR1=(a*(TIM1->ARR))/100;
TIM1->CCR2=((100-a)*(TIM1->ARR))/100;
```

On génère des PWM en commande complémentaire décalée, puis on visualise à de l'oscilloscope. La figure suivante, représente ces 4 signaux en branchant les sondes sur les pins indiquées.

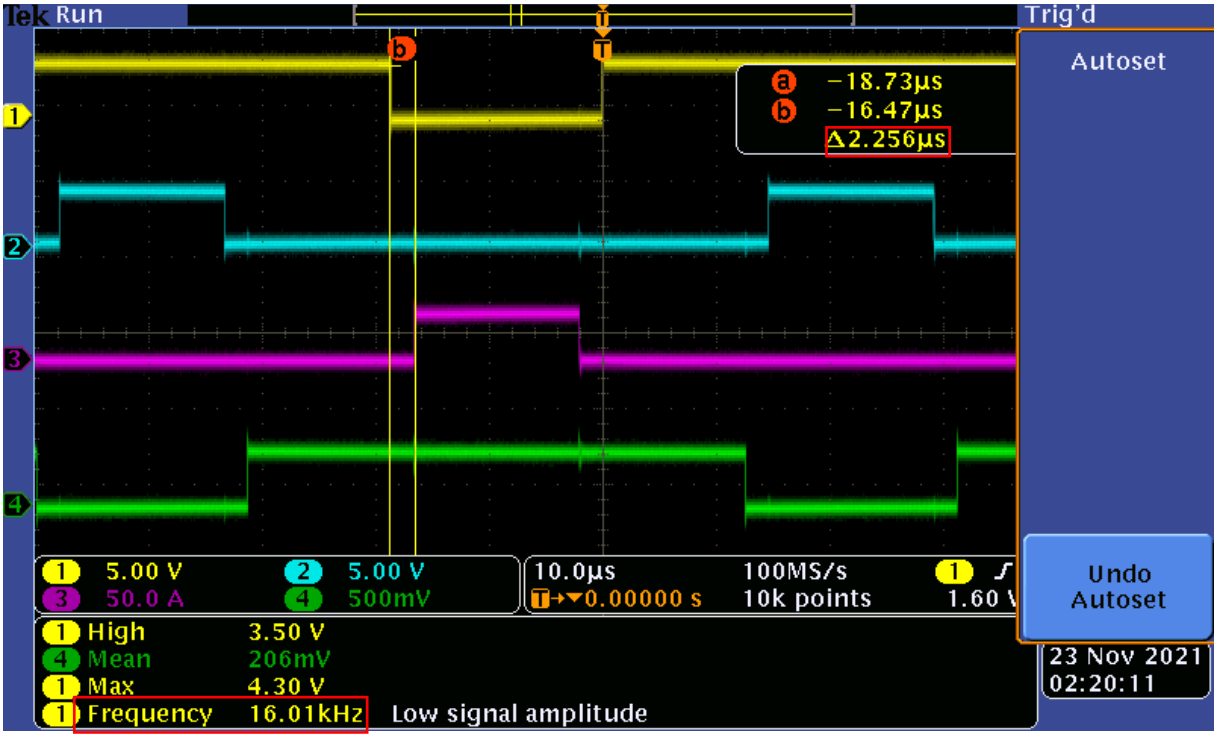


Figure 2: Les 4 PWM, issues de TIM1

Les 4 PWM permettent de contrôler l'ouverture ou la fermeture des transistors de la machine à courant continu et ainsi de contrôler la tension moyenne appliquée aux bornes du moteur et donc la vitesse de rotation. (E=Vdc(2α-1)).

La fréquence de découpage est bien à 16kHz. Deux signaux, correspondent à TOP, ils ne sont pas à l'état haut en même temps. Ils comprennent un « temps mort » pour tenir compte du temps de commutation, permet d'éviter que les transistors TOP et BOTTOM d'un même bras ne conduisent simultanément.

Les signaux observés à l'oscilloscope nous permettent de mesurer le temps mort qui est bien en adéquation avec le cahier des charges.
Les deux autres signaux TOP comprennent également des signaux complémentaires (BOTTOM).

Les signaux BOTTOM des signaux 1 et 2 sont respectivement les signaux 3 et 4. On notera que l'amplitude des signaux est de 5V.

- ***3.3. Prise en main du hacheur et câblage***

Le hacheur utilisé comprend de nombreuses broches d'entrée/sortie. A partir de la datasheet du hacheur, nous allons réaliser la liste des broches à alimenter, à relier à la masse et de commande PWM.

Voici la datasheet du hacheur :

Pin	Function	Net Name	Input/Output	Isolated
1	Not Used	—	—	—
2	Yellow Phase Shunt Current Feedback	Y_SHUNT	Output	No if LK20 Fitted
3	DC Bus Shunt Current Feedback	BUS_SHUNT	Output	No If LK22 Fitted
4	Not Used	—	—	—
5	Yellow Phase Voltage Feedback	Y_VPH_SENSE	Output	No If LK25 Fitted
6	Blue Phase Back EMF crossing	B_CROSSING	Output	No if LK27 Fitted
7	Red Phase Back EMF crossing	R_CROSSING	Output	No if LK29 Fitted
8	Rectifier Output Voltage (|VAC|) Feedback	|VAC|_SENSE	Output	No if LK31 Fitted
9	Analog +5V from control PCB (±2%)	ISO_A+5V	Input	Yes
10	PFC Switch Firing Command	CMD_PFC	Input	Yes
11	Blue Phase Top Switch Firing Command	CMD_B_TOP	Input	Yes
12	Yellow Phase Top Switch Firing Command	CMD_Y_TOP	Input	Yes
13	Red Phase Top Switch Firing Command	CMD_R_TOP	Input	Yes
14	Active Low Serial Clock	$\overline{\text{ISO\_SCLK}}$	Output	Yes
15	Active Low Fault	$\overline{\text{FAULT\_ISO}}$	Output	Yes
16	Yellow Phase Hall Current Sensor Feedback	Y_HALL	Output	Yes
17	PFC Hall Current Sensor Feedback	PFC_HALL	Output	Yes
18	Digital GND from control PCB	ISO_GND	Input	Yes
19	Digital +5V from control PCB (±2%)	ISO_+5V	Input	Yes
20	Blue Phase Shunt Current Feedback	B_SHUNT	Output	No if LK19 Fitted
21	Red Phase Shunt Current Feedback	R_SHUNT	Output	No if LK21 Fitted
22	Brake Chopper Switch Shunt Current Feedback	BRAKE_SHUNT	Output	No if LK23 Fitted
23	Blue Phase Voltage Feedback	B_VPH_SENSE	Output	No if LK24 Fitted
24	Red Phase Voltage Feedback	R_VPH_SENSE	Output	No If LK26 Fitted
25	Yellow Phase Back EMF crossing	Y_CROSSING	Output	No if LK28 Fitted
26	DC Bus Voltage Feedback	BUS_SENSE	Output	No if LK30 Fitted
27	Analog GND from control PCB	ISO_AGND	Input	Yes
28	Brake Chopper Switch Firing Command	CMD_BRAKE	Input	Yes
29	Blue Phase Bottom Switch Firing Command	CMD_B_BOT	Input	Yes
30	Yellow Phase Bottom Switch Firing Command	CMD_Y_BOT	Input	Yes
31	Red Phase Bottom Switch Firing Command	CMD_R_BOT	Input	Yes
32	Active Low Serial Data	$\overline{\text{ISO\_DATA}}$	Output	Yes
33	Fault Reset Command	ISO_RESET	Input	Yes
34	Not Used	—	—	—
35	Red Phase Hall Current Sensor Feedback	R_HALL	Output	Yes
36	Digital GND from control PCB	ISO_GND	Input	Yes
37	Digital +5V from control PCB (±2%)	ISO_+5V	Input	Yes

Figure 3 : Caractéristique du hacheur

Hacheur	Carte MicroP	Fonction
12 Yellow Phase Top ch1	PA8	CH1
13 Red Phase Top ch2	PA9	CH2
30 Yellow Phase Bottom	PA11	CH1N
31 Red Phase Bottom	PA12	CH2N
33 Fault Reset Command	PA10	Fault reset command

Ce tableau, représente les différentes les différentes connections entre les PINs du hacheur et de notre microP. Par la suite, nous affections le branchement.

Voici le branchement suivant :

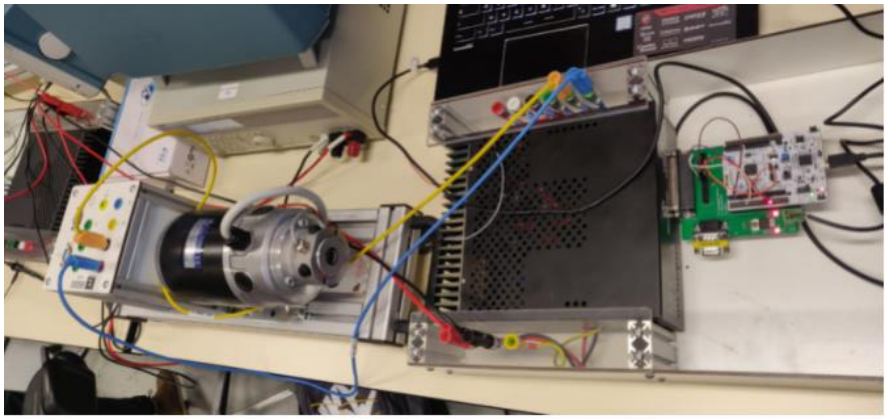


Figure 4 : Branchement

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.

- ***3.4. Commande start***

Le hacheur a besoin d'une séquence de d'amorçage pour obtenir une tension de sortie. Il existe deux types de séquence d'allumage via le GPIO :

- Sur l'appui du bouton bleu de la carte avec une gestion d'interruption lors de l'appui (EXTI)
- Sur la réception en uart de la commande "start"

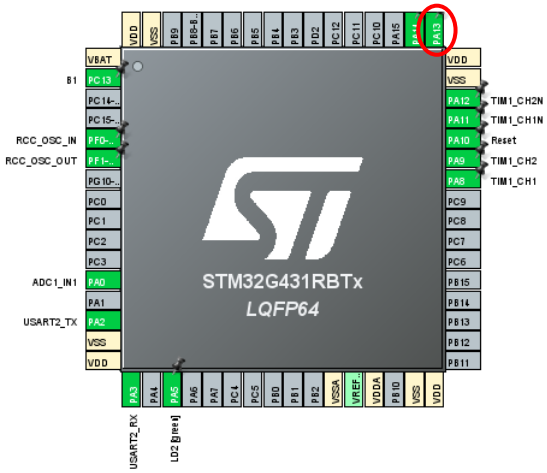
Le code sera établi de manière à activer la séquence d'allumage du moteur via l'entrée de la commande start et via le bouton bleu de la carte Nucleo-G431RB. Le GPIO relatif à ce bouton est le GPIO PC13.

❖ ***Appui sur le bouton***

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

```
if (button_it==1){
    HAL_GPIO_WritePin(Reset_GPIO_Port, Reset_Pin, SET);
    HAL_Delay(1);
    HAL_GPIO_WritePin(Reset_GPIO_Port, Reset_Pin, RESET);
    button_it=0;
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
```

Pour activer le bouton bleu, nous l'avons activé via le logiciel STM32 :



Par la suite, nous avons testé la fonction à l'aide de l'oscilloscope.

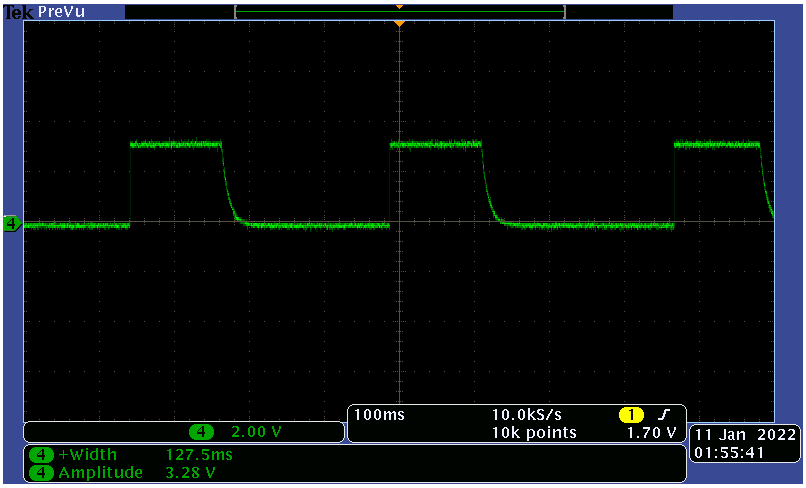


Figure 5 : Test de la fonction bouton

On remarque que à chaque appuie sur le bouton, le signal passe à 1. On en déduit donc que la fonction de start du bouton bleu fonctionne correctement.

On peut faire la même chose en tapant start :

```
else if (strncmp(cmd,"start",4)==0){
    HAL_UART_Transmit(&huart2, start, sizeof(start), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);
    HAL_GPIO_WritePin(Reset_GPIO_Port, Reset_Pin, SET);
    HAL_Delay(1);
    HAL_GPIO_WritePin(Reset_GPIO_Port, Reset_Pin, RESET);
```

- **3.5.** ***Définition de la vitesse :***

Nous avons branché le moteur en respectant les données constructeur du moteur. Pour contrôler la vitesse du moteur, nous allons envoyer une séquence via la liaison UART (par l'USB) de la forme :

```
speed=XXXX
```

L'objectif de cette commande permet de régler la vitesse du moteur en écrivant le rapport cyclique souhaité Par exemple, "speed=50" permet de régler le rapport cyclique à 50% et donc le moteur sera à l'arrêt. Le moteur tourne dans le sens horaire pour une vitesse comprise entre [50 :75], et tourne dans le sens anti horaire pour une vitesse comprise entre [35 :50]. Voici la fonction speed :

```
else if(strncmp(cmd,"plus",4)==0){
    val = (val + 100)%(TIM1->ARR);
    TIM1->CCR1 = val;
    TIM1->CCR2 = TIM1->ARR - val;
}
else if(strncmp(cmd,"moins",5)==0){
                    val = (val - 100)%(TIM1->ARR);
                    TIM1->CCR1 = val;
                    TIM1->CCR2 = TIM1->ARR - val;
}
else if(strncmp(cmd,"speed",5)==0){
    int a = (atoi(cmd+5));
    if(a<=0) a=1;
    if(a>=100) a=99;
    TIM1->CCR1=(a*(TIM1->ARR))/100;
    TIM1->CCR2=((100-a)*(TIM1->ARR))/100;
    HAL_UART_Transmit(&huart2, prompt, sizeof(prompt), HAL_MAX_DELAY);
```

Avec atoi, on prend en considération les caractères après les 5 premiers quand on reconnait la commande speed, ainsi une fois la valeur stockée dans a, on obtient le rapport à appliquer à Arr pour la valeur de CCR.

Par la suite nous avons réalisé différents tests pour différentes valeurs de rapport cyclique :

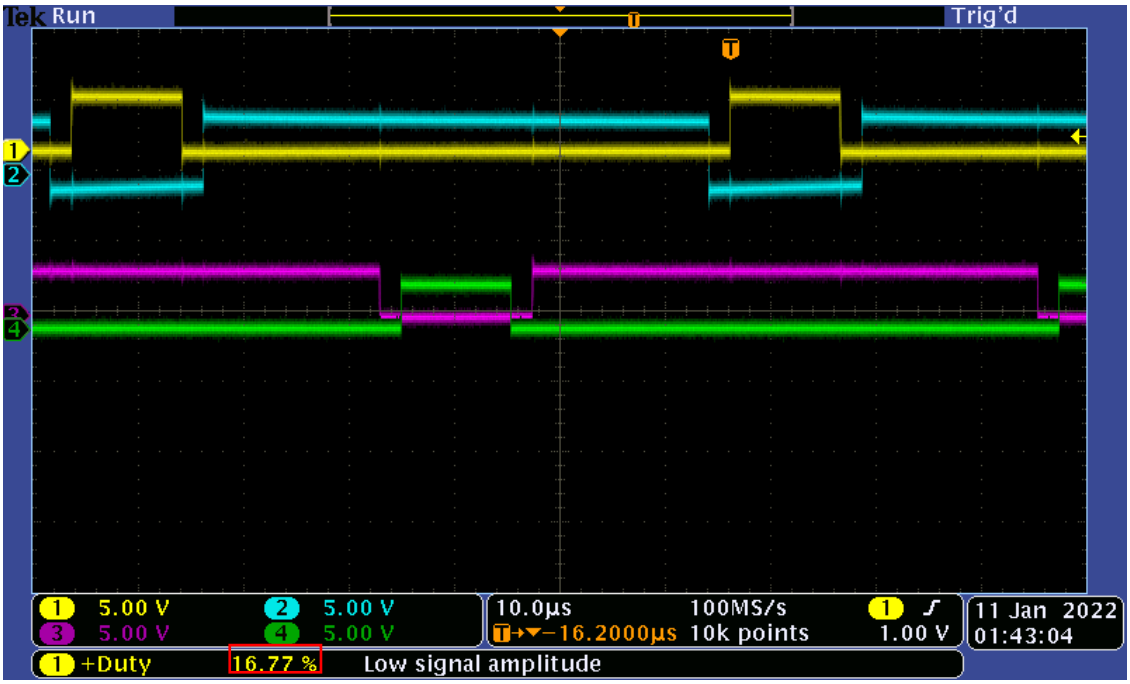


Figure 6 : Rappor cyclique = 17%

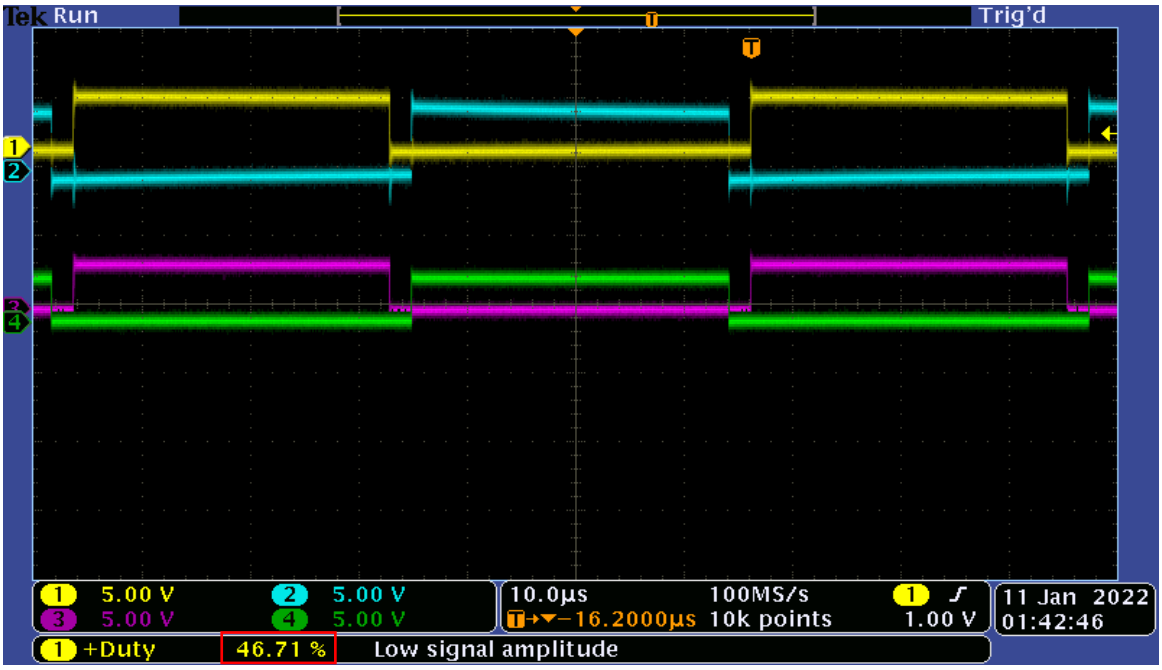


Figure 7 : Rapport cyclique 45%

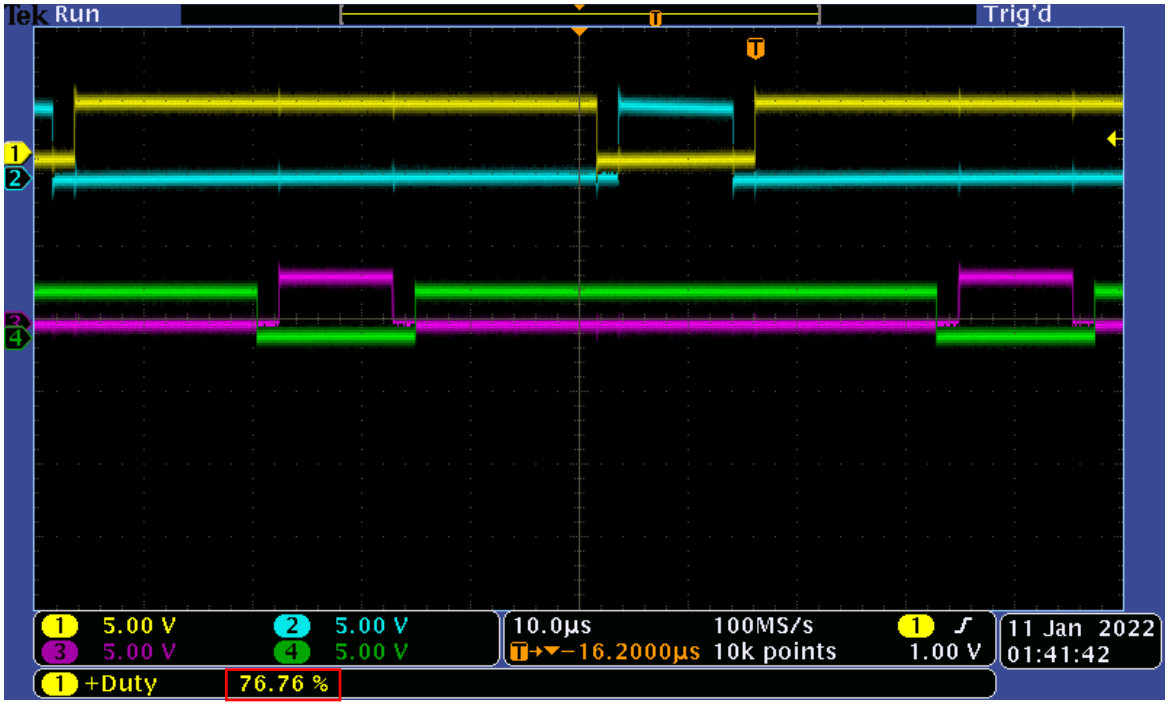


Figure 8 : Rapport cyclique 75%

On constate que notre réglage de vitesse fonctionne aussi bien dans les deux sens, les temps morts sont bien respectés.

Cependant on note que lorsque l'on souhaite commander le moteur il est difficile de changer la commande de vitesse (donc le rapport cyclique) trop brusquement car cela provoque un appel de courant trop important.

Afin d'améliorer ce point nous avons travailler sur la réalisation d'une boucle de courant à l'aide d'un ADC. Nous ne somme cependant pas parvenu à des résultats concluants par manque de temps.