

TP commande numérique :

Gozard Matthieu ; Guilloso Kinann

1. Objectifs

A partir d'un hacheur complet et d'une carte Nucleo-STM32G474RE, vous devez :

1. Réaliser un shell pour commander le hacheur, sur la base d'un code fourni
2. Réaliser la commande des 4 transistors du hacheur en commande complémentaire décalée
3. Faire l'acquisition des différents capteurs
4. Réaliser l'asservissement en temps réel

2. Bonnes pratiques

Pour une meilleure lisibilité du code :

- Toutes les variables sont écrites en minuscule avec une majuscule pour les premières lettre à partir du 2eme mot dans le nom de la variable, par exemple : *uartRxBuffer*
- Les directives (macro avec #define) sont écrites en majuscule, par exemple : *#define UART_TX_SIZE*
- Aucun nombre magique ne peut exister, par exemple : remplacer `if(value == 512)` par `if(value == VAL_MAX)` avec *#define VAL_MAX 512*
- Si vous écrivez deux fois le même code, il faut écrire une fonction pour "factoriser" votre code.
- Il faut tester vos appels de fonction, les fonctions HAL renvoient pour la plupart une valeur pour savoir si son exécution s'est déroulée correctement.

3. Github et Doxygen

Le lien de notre github est le suivant :

https://github.com/Kinann75/Commande_numerique_TP

6. Console UART

7. Commande MCC basique

7.1. Génération de 4 PWM

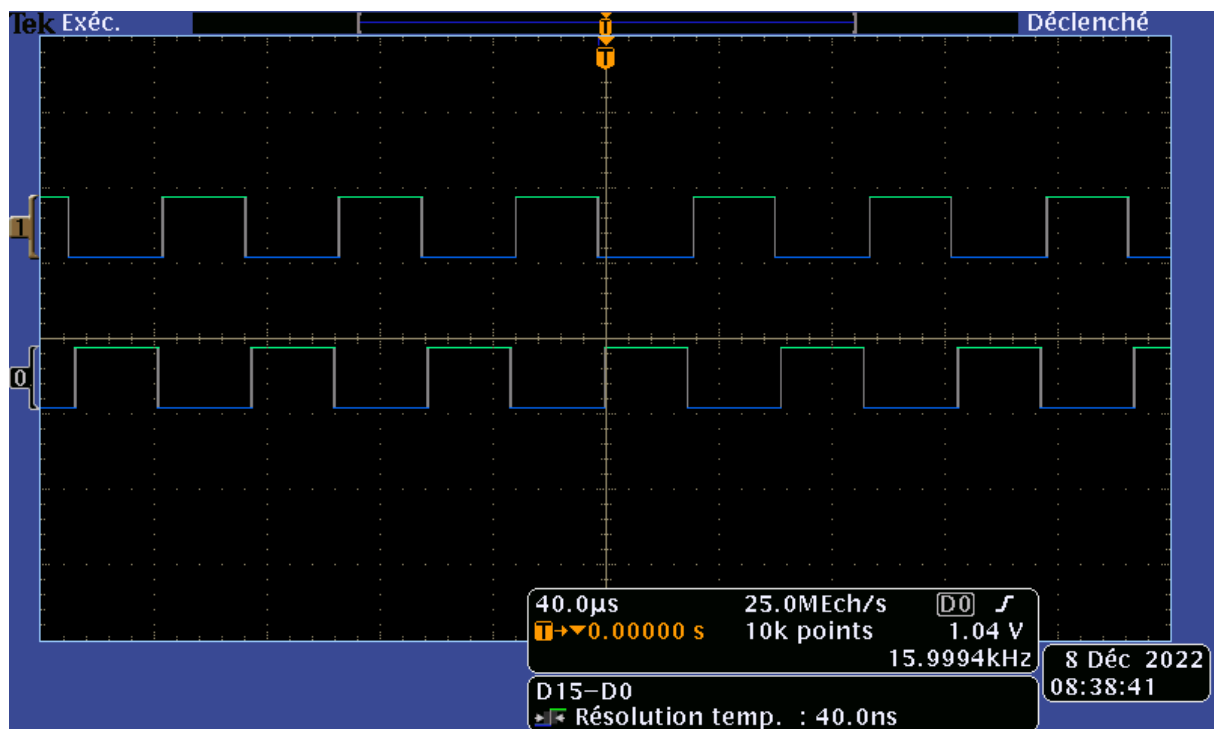
▼ PWM Generation Channel 1 and ...	
Mode	PWM mode 1
Pulse (16 bits value)	1000
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
CHN Polarity	High
CH Idle State	Reset
CHN Idle State	Reset
▼ PWM Generation Channel 2 and ...	
Mode	PWM mode 1
Pulse (16 bits value)	4311
▼ Counter Settings	
Prescaler (PSC - 16 bits val... 0	
Counter Mode	Center Aligned mode1
Dithering	Disable
Counter Period (AutoReload.. 5311	
Internal Clock Division (CKD) No Division	
Repetition Counter (RCR - 1.. 0	
auto-reload preload	Disable

Paramétrage des PWM générés par le timer 1.

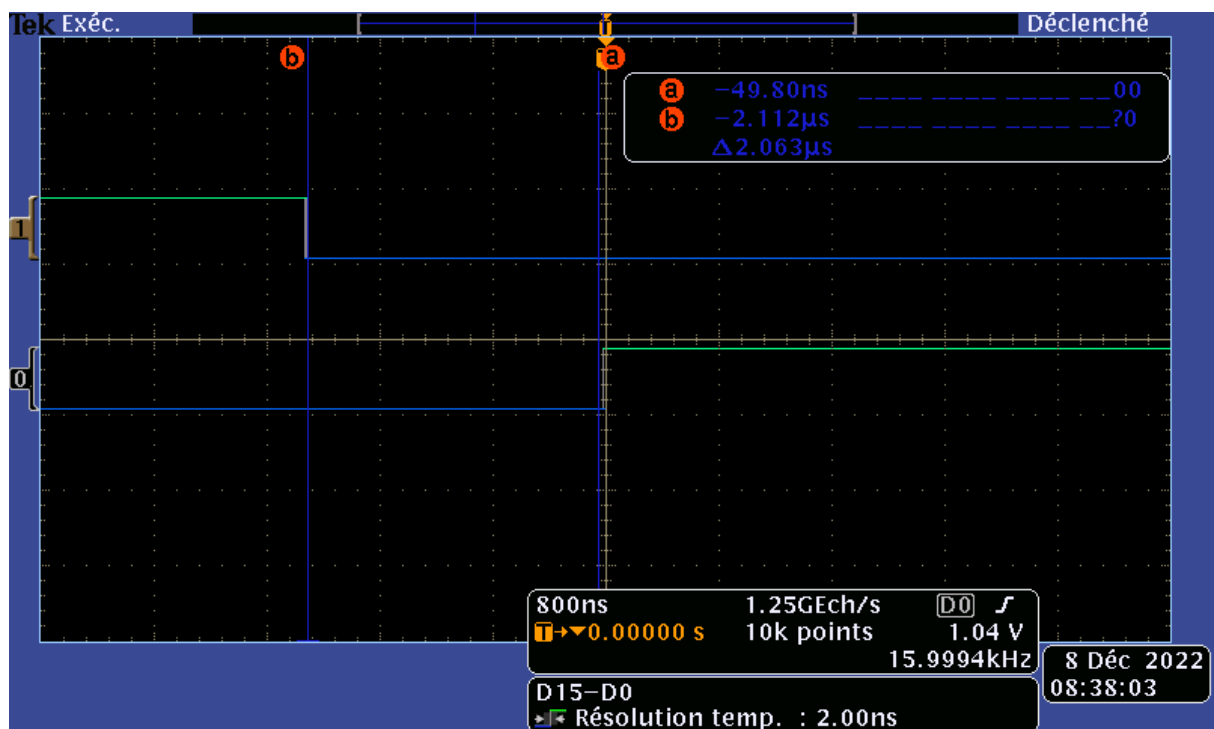
Pour choisir la counter période nous utilisons l'expression suivante.
 $f_{clk}/f_{pwm} = 1/(1+PSC)(1+CCR)$ (en prenant $PSC=0$, ce qui nous permettra d'avoir une plus grande précision sur "Pulse"). On trouve alors $CCR=5311$.

Dead Time 205

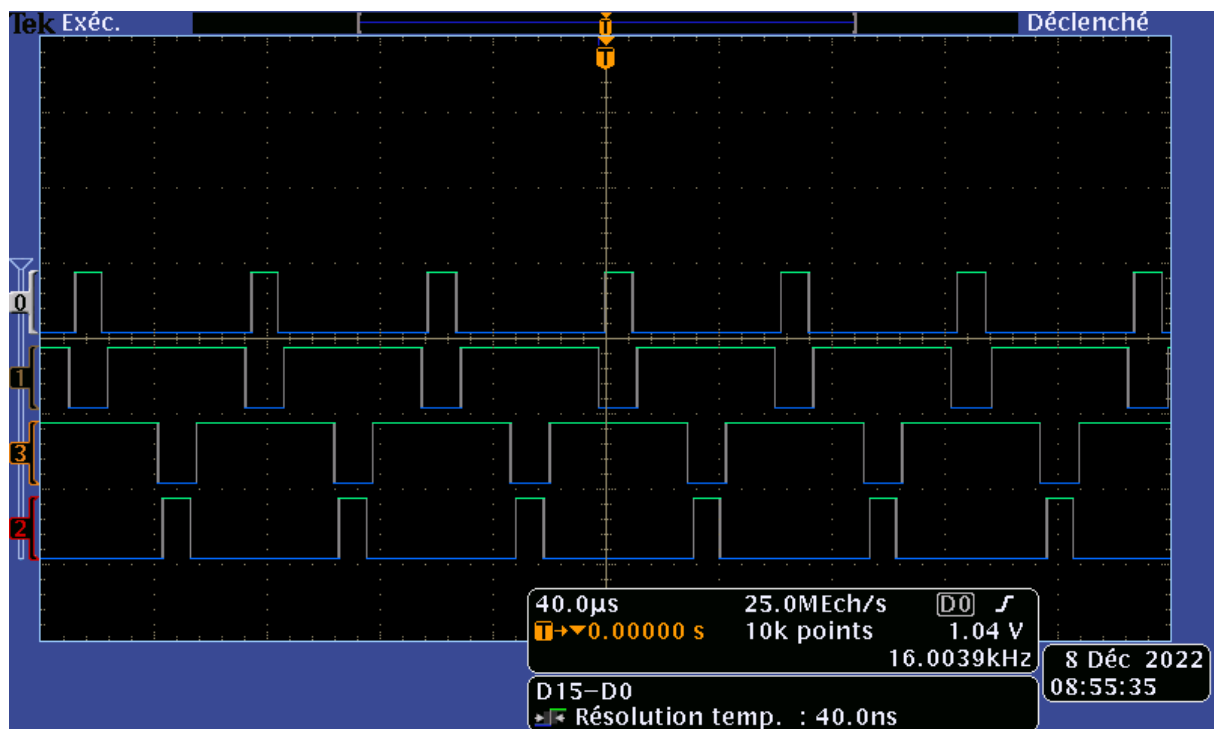
Choisir Dead Time à 205 nous permet d'avoir un temps mort de 2us.



Affichage à l'oscilloscope des signaux PWM1 et PWM1N.



Affichage à l'oscilloscope des signaux PWM1 et PWM1N avec un temps mort de 2µs.



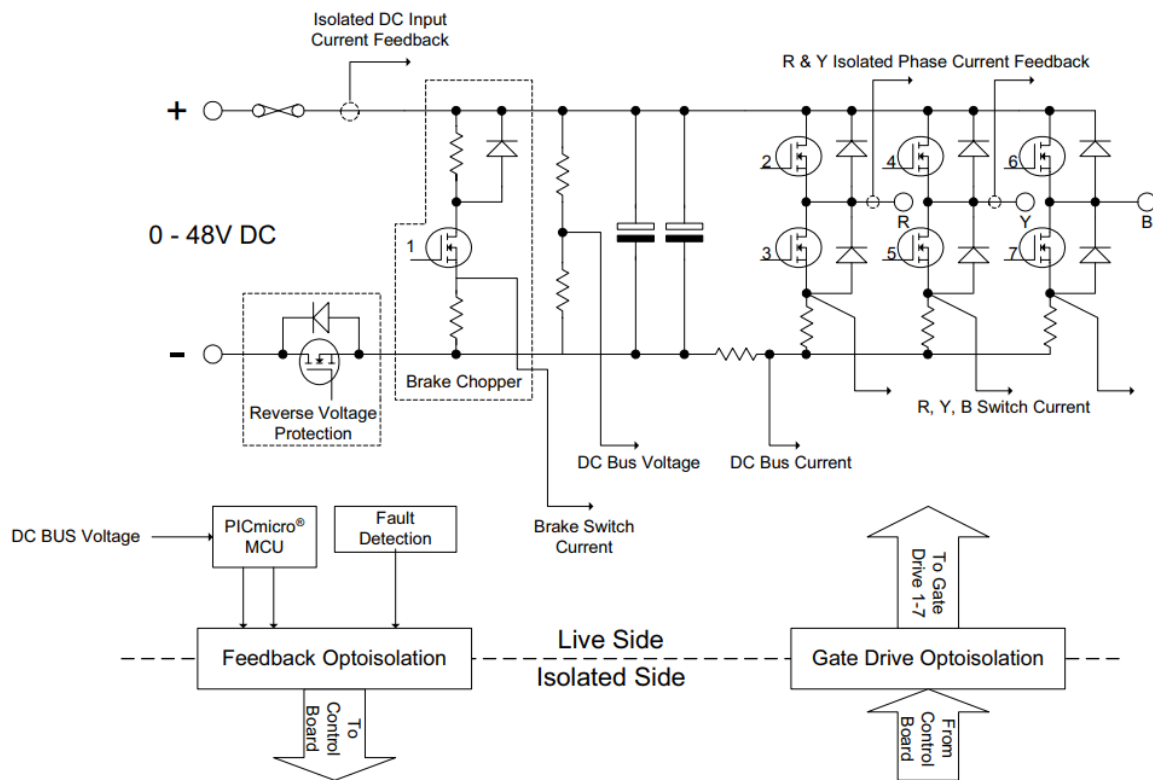
Affichage à l'oscilloscope des signaux PWM1, PWM1N, PWM2 et PWM2N.

7.2. Prise en main 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 ($\pm 2\%$)	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	ISO_SCLK	Output	Yes
15	Active Low Fault	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 ($\pm 2\%$)	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	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 ($\pm 2\%$)	ISO_+5V	Input	Yes

USER SIGNAL CONNECTOR PINOUT (37-PIN, D-TYPE)

En rouge seront tous les pins que nous devons connecter. Ceux-ci comprennent les 4 signaux PWM que nous avons générés précédemment et qui seront connectés sur les pins Red_top, Red_bottom, Yellow_top et Yellow_bottom. Le signal PWM1 sera envoyé sur red_top, PWN1N sur Red_bottom, PW2 sur Yellow_top et PMW2N sur Yellow_bottom. Enfin on générera un signal GPIO pour l'envoyer sur le pin ISO_RESET. Les alimentations 5V et GND sont déjà reliées sur le PCB, nous n'avons donc pas à nous en occuper.



MC1L 3-PHASE LOW VOLTAGE POWER MODULE BLOCK DIAGRAM

7.3. Commande start

Voici ce qu'il va nous falloir générer pour ISO_RESET (à partir d'un GPIO) d'après la datasheet "Power Module 70097A".

- Reset the system by activating the active high ISO_RESET line. The ISO_RESET line is on pin 33 of the 37-pin, D-type (see **Section 1.8 "User Signal Connector Pinout (37-Pin, D-Type)"**). On the dsPICDEM MC1 Motor Control Development Board, this signal is routed to pin 14 of the 30F6010 dsPIC device, which is on Port RE9. The minimum pulse width for the reset is 2 μ s. The RESET should be done in coordination with the SPI™ handling routine of the dsPIC device to ensure correct synchronization of the serial interface providing the isolated voltage feedback (see **Section 1.5.7.2 "Isolated Voltage Feedback"**). The system is now ready to use.

Nous devons générer ce signal de deux façons : après avoir appuyer sur le bouton bleu de la carte, ou en écrivant "power on"

```

while (1)
{
    if (flag==1)
    {
        HAL_GPIO_WritePin(ISO_RESET_GPIO_Port, ISO_RESET_Pin,GPIO_PIN_SET);
        for(i=0;i<1000;i++)
        {
        }
        HAL_GPIO_WritePin(ISO_RESET_GPIO_Port, ISO_RESET_Pin,GPIO_PIN_RESET);
        flag=0;
    }
}

186 /* USER CODE BEGIN 4 */
187 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
188 {
189     flag=1;
190 }

```

On a finalement mis la boucle for de 0 à 60. Nous avons ainsi généré le signal ISO_RESET comme voulu. Nous avons également ajouté ce code dans la fonction moterPowerOn pour pouvoir générer le signal ISO_RESET depuis le shell.

Il nous faut maintenant pouvoir modifier le rapport cyclique (alpha) des PWM depuis le shell. Voilà ce que nous avons réalisé :

```

53 const uint8_t alpha[]="rapport cyclique\r\n";

168     else if((strcmp(argv[0],"alpha")==0))
169     {
170         HAL_UART_Transmit(&huart2, alpha, sizeof(alpha), HAL_MAX_DELAY);
171         set_alpha(atoi(argv[1]));
172     }

/**
 * @brief Set the motor speed
 * @param a : set alpha to the value of a in %
 * @retval None
 */
void set_alpha(int a)
{
    TIM1->CCR1=a*(TIM1->ARR)/100;
    TIM1->CCR2=TIM1->ARR-a*(TIM1->ARR)/100;
}

```

7.4. Premiers tests

Nous avons brancher le moteur en respectant les données constructeur du moteur et réalisé les tests suivants :

- Rapport cyclique de 50% : le moteur ne tourne pas, la tension aux bornes du moteur est de 0V.
- Rapport cyclique de 70% : le moteur tourne correctement.
- Rapport cyclique de 100% : le moteur tourne à sa vitesse maximale.
- Rapport cyclique de 0% : le moteur tourne à sa vitesse maximale dans l'autre sens.

7.5. Définition de la vitesse

Dans motor.c on a écrit :

```
43 void motorSetSpeed(int speed) {
44     int alpha=0;
45     HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin); // just for test, you can delete it
46     alpha=( (speed+3000)/6000)*100;
47     set_alpha((int)alpha);
48 }
```

Dans shell.c on a écrit :

```
174     else if ((strcmp(argv[0], "set")==0) && (strcmp(argv[1], "speed")==0))
175     {
176         HAL_UART_Transmit(&huart2, setspeed, sizeof(setspeed), HAL_MAX_DELAY);
177         motorSetSpeed(atoi(argv[2]));
178     }
```

8. Capteurs de courant et position.

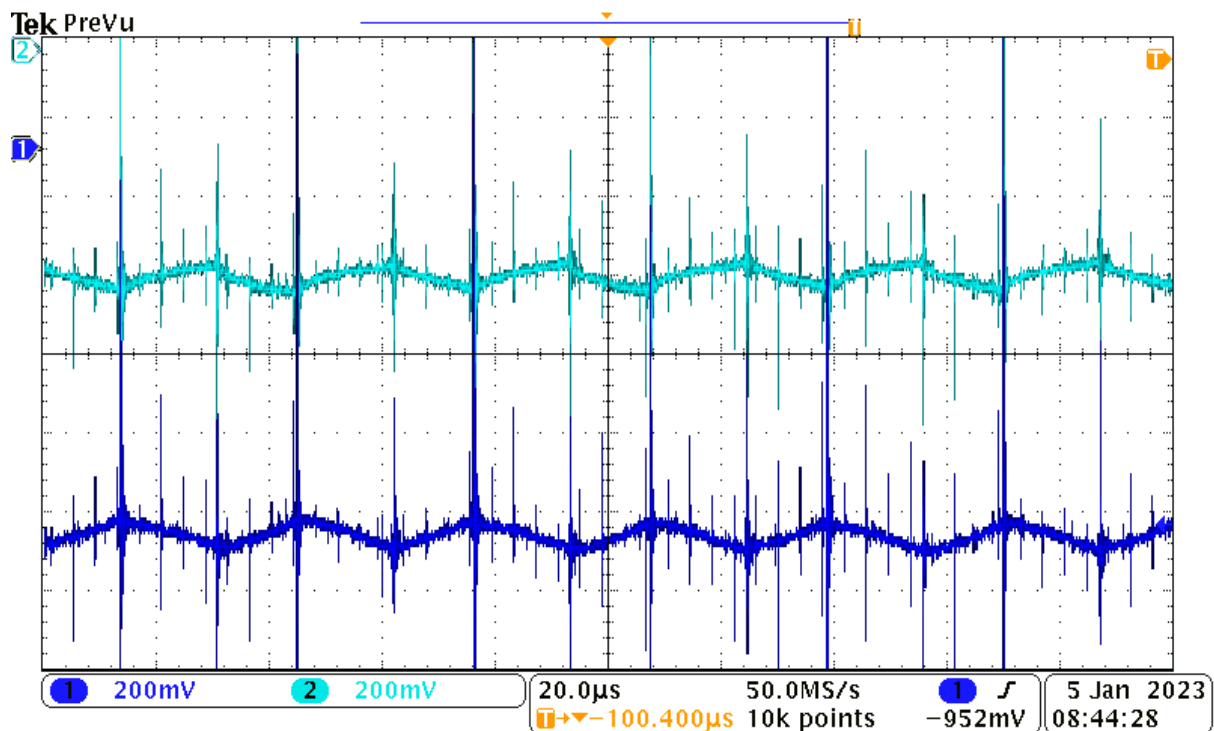
Objectif :

- Mettre en place la mesure de courant à partir de la résistance de shunt.

- Mettre en place la mesure de vitesse du moteur à partir de la roue codeuse.

8.1. Mesure de courant

On observe les tensions obtenues aux pins 16 et 35 du connecteur 37 broches qui correspondent à “yellow phase hall current sensor” et “red phase hall current sensor”:



Affichage à l'oscilloscope des signaux aux pins 16 (en vert) et 35 (en bleu).

Nous observons que les signaux sont fortement bruités, néanmoins, on observe bien les évolutions du courant (en dent de scie). La fréquence des signaux est calée sur la fréquence du timer 1 soit sur une fréquence de 16kHz.

Nous allons maintenant configurer l'ADC1 et le DMA afin de faire l'acquisition de 10 valeurs de courant par période.

Fichier .ioc :

The screenshot shows the STM32CubeMX GUI with the 'NVIC Settings' tab selected. The 'NVIC Interrupt Table' is displayed, showing the configuration for the DMA1 channel1 global interrupt and the ADC1 and ADC2 global interrupt. The DMA1 channel1 global interrupt is enabled with a preemption priority of 0 and a sub priority of 0. The ADC1 and ADC2 global interrupt is disabled with a preemption priority of 0 and a sub priority of 0.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 channel1 global interrupt	<input checked="" type="checkbox"/>	0	0
ADC1 and ADC2 global interrupt	<input type="checkbox"/>	0	0

Categories A->Z

- COMP5
- COMP6
- COMP7
- ⚠ DAC1
- DAC2
- DAC3
- DAC4
- OPAMP1
- ⚠ OPAMP2
- OPAMP3
- OPAMP4
- ⚠ OPAMP5
- OPAMP6

Timers

- ⚠ HRTIM1
- LPTIM1
- RTC
- ⚠ TIM1
- TIM2**
- TIM3
- TIM4
- TIM5
- TIM6
- TIM7
- ⚠ TIM8
- TIM15

Mode

Slave Mode Disable

Trigger Source Disable

Clock Source Internal Clock

Configuration

Reset Configuration

Parameter Settings
User Constants
NVIC Settings
DMA Settings

Configure the below parameters :

Counter Settings

Prescaler (PSC - 16 bits value)

0

Counter Mode

Up

Dithering

Disable

Counter Period (AutoReload Register - 32 bits value)

531

Internal Clock Division (CKD)

No Division

auto-reload preload

Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Enable (Trigger delayed for master/slaves simultaneous start)

Trigger Event Selection TRGO

Update Event

Fichier main.C :

```
main.c x NUCLEO-G474R... shell.c startup_stm... usart.c usart.h shell.h tim.c »%
51
52 /* Private variables -----*/
53
54 /* USER CODE BEGIN PV */
55
56 extern uint8_t uartRxReceived;
57 extern uint8_t uartRxBuffer[UART_RX_BUFFER_SIZE];
58 extern uint8_t uartTxBuffer[UART_TX_BUFFER_SIZE];
59 uint8_t flag=0;
60 uint8_t flagADC=0;
61 uint32_t ADC_Buffer[20];
62 char chaine[30];
63 /* USER CODE END PV */
```

On commence par créer un tableau ADC_Buffer dans lequel seront stockées les valeurs récupérées par le capteur de courant. On crée aussi une variable flagADC initialisée à 0 dont on verra l'utilité dans notre code plus tard.

```
main.c x NUCLEO-G474R... shell.c startup_stm... usart.c usart.h shell.h tim.c »%
108 MX_TIM2_Init();
109 /* USER CODE BEGIN 2 */
110 HAL_UART_Receive_IT(&huart2, uartRxBuffer, UART_RX_BUFFER_SIZE);
111 HAL_Delay(1);
112 shellInit();
113
114 HAL_TIM_Base_Start(&htim2);
115
116 if(HAL_OK!=HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED))
117 {
118     Error_Handler();
119 }
120 if(HAL_OK!=HAL_ADC_Start_DMA(&hadc1, ADC_Buffer, 10)){
121     Error_Handler();
122 }
123
124
125
```

On initialise le timer 2, l'ADC1 et le DMA dans "USER CODE BEGIN 2" à l'aide des fonctions HAL associées.

```

139  /* USER CODE BEGIN WHILE */
140  while (1)
141  {
142      if (flag==1)
143      {
144          HAL_GPIO_WritePin(ISO_RESET_GPIO_Port, ISO_RESET_Pin,GPIO_PIN_SET);
145          for(i=0;i<70;i++)
146          {
147              }
148          HAL_GPIO_WritePin(ISO_RESET_GPIO_Port, ISO_RESET_Pin,GPIO_PIN_RESET);
149          flag=0;
150      }
151      // SuperLoop inside the while(1), only flag changed from interrupt could launch functions
152      if(uartRxReceived){
153          if(shellGetChar()){
154              shellExec();
155              shellPrompt();
156          }
157          uartRxReceived = 0;
158      }
159
160      if (flagADC==1)
161      {
162          //sprintf(chaine,"resultat de conversion : %i\r\n", ADC_Buffer[0]);
163          //HAL_UART_Transmit(&huart2, (uint8_t *)chaine, strlen(chaine), HAL_MAX_DELAY);
164          flagADC=0;
165      }
166

```

Dans le while (1), si le flagADC a été mis à 1 par le code que l'on verra ci-dessous, on réinitialise flagADC à 0.

FlagADC est mis à 1 une fois que ADC_Buffer est rempli avec les valeurs fournies par le capteur.

Fichier shell.c :

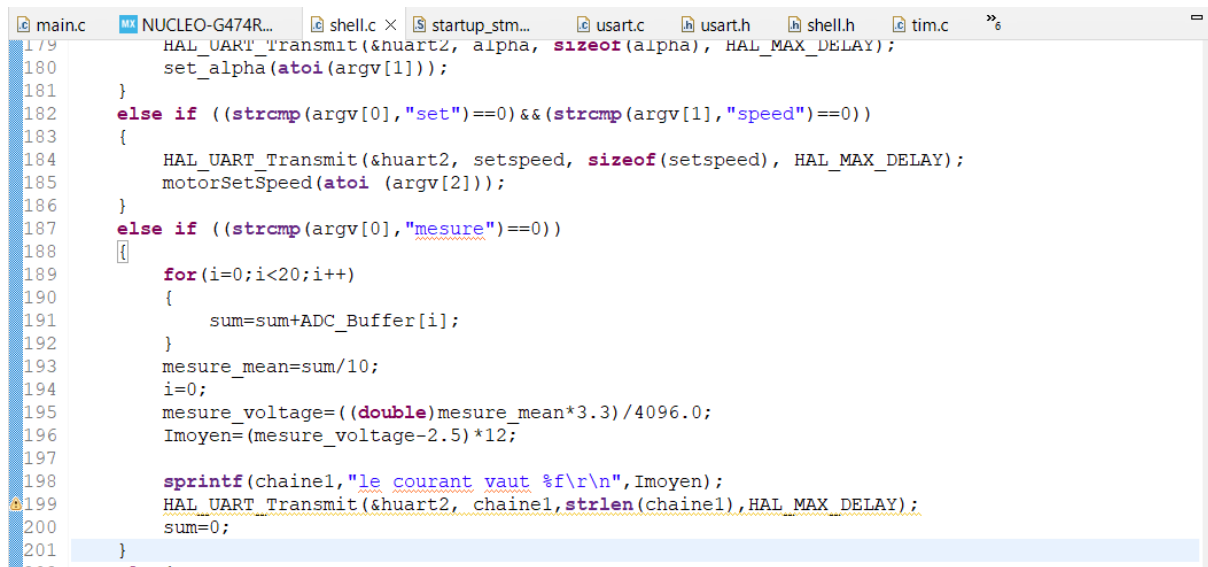
Nous allons maintenant créer une commande “mesure” dans le shell qui nous permettra d’observer la valeur du courant après conversion.

```

38  "\r\n| Pinout used |"
39  "\r\n*-----*\r\n"
40  "PC13 : blue button\r\n"
41  "PC3 : ISO_RESET\r\n"
42  "PA2 : USART2_TX\r\n"
43  "PA3 : USART2_RX\r\n"
44  "PA5 : LED\r\n"
45  "PA8 : TIM1_CH1\r\n"
46  "PA9 : TIM1_CH2\r\n"
47  "PA11 : TIM1_CH1N\r\n"
48  "PA12 : TIM1_CH2N\r\n";
49  const uint8_t powerOn[]="Switch on motor control\r\n";
50  const uint8_t powerOff[]="Switch off motor control\r\n";
51  const uint8_t motorSpeedInst[]="Enter a motor speed such as \"set speed <int>\"\r\n";
52  const uint8_t cmdNotFound[]="Command not found\r\n";
53  const uint8_t alpha[]="rapport cyclique\r\n";
54  const uint8_t setspeed[]="modification de la vitesse\r\n";
55  const uint8_t mesure[]="mesure du courant\r\n";
56
57
58  char cmdBuffer[CMD_BUFFER_SIZE];
59  extern uint8_t uartRxBuffer[UART_RX_BUFFER_SIZE];
60  extern uint16_t ADC_Buffer[20];

```

Nous créons donc une const mesure[] et récupérons en variable externe ADC_Buffer.



```
179 HAL_UART_Transmit(&huart2, alpha, sizeof(alpha), HAL_MAX_DELAY);
180 set_alpha(atoi(argv[1]));
181 }
182 else if ((strcmp(argv[0], "set")==0) && (strcmp(argv[1], "speed")==0))
183 {
184     HAL_UART_Transmit(&huart2, setspeed, sizeof(setspeed), HAL_MAX_DELAY);
185     motorSetSpeed(atoi(argv[2]));
186 }
187 else if ((strcmp(argv[0], "mesure")==0))
188 {
189     for(i=0; i<20; i++)
190     {
191         sum=sum+ADC_Buffer[i];
192     }
193     mesure_mean=sum/10;
194     i=0;
195     mesure_voltage=((double)mesure_mean*3.3)/4096.0;
196     Imoyen=(mesure_voltage-2.5)*12;
197
198     sprintf(chain1, "le courant vaut %f\r\n", Imoyen);
199     HAL_UART_Transmit(&huart2, chain1, strlen(chain1), HAL_MAX_DELAY);
200     sum=0;
201 }
```

Ensuite nous faisons une moyenne des 10 valeurs récupérées dans ADC_Buffer (une "case" sur deux ne récupère pas de données et reste à 0). Pour passer de la valeur numérique fournie par le capteur à la tension réelle fournie, nous la multiplions par 3,3 (valeur max de courant que peut fournir le capteur) et la divisons par 2¹² (ou 4096 qui correspond à la précision de la mesure). Enfin, nous trouvons la valeur du courant moyen correspondant via la datasheet du capteur. Pour finir, nous affichons la valeur sur le shell sur demande.

Nous remarquons une erreur de mesure. Effectivement, lorsque le courant est à 0, la valeur mesurée est environ de 0,25A. Cette erreur est due aux erreurs cumulées du capteur de courant et de la mesure par l'ADC.

Mesure de la vitesse :

Fichier .ioc :

TIM3 Mode and Configuration

Mode	
Channel1	Disable
Channel3	Disable
Channel4	Disable
Combined Channels	Encoder Mode
Use ETR as Clearing Source	Disable
<input type="checkbox"/> XOR activation <input type="checkbox"/> One Pulse Mode	

Configuration	
Reset Configuration	
<input checked="" type="checkbox"/> Parameter Settings <input checked="" type="checkbox"/> User Constants <input checked="" type="checkbox"/> NVIC Settings <input checked="" type="checkbox"/> DMA Settings <input checked="" type="checkbox"/> GPIO Settings	
Configure the below parameters :	
<input type="text" value="Search (Ctrl+F)"/>	<input type="button" value="↶"/> <input type="button" value="↷"/> <input type="button" value="i"/>
Slave Mode Preload Activation Disable	
____ Parameters for Channel 1 ____	
Polarity	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter	0
____ Parameters for Channel 2 ____	
Polarity	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter	0

On commence par activer TIM 5 en interruption à une fréquence de 10Hz et TIM 3 qui gèrera le compte des fronts montants et descendants sur CNT envoyés par la roue codeuse.

Fichier main.c :

```

247 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
248 {
249     /* USER CODE BEGIN Callback 0 */
250
251     /* USER CODE END Callback 0 */
252     if (htim->Instance == TIM6) {
253         HAL_IncTick();
254     }
255     /* USER CODE BEGIN Callback 1 */
256     if (htim->Instance == TIM5) {
257         valeur_timer = TIM3->CNT;
258         vitesse = (((valeur_timer - 32000) * 60) / 4096) / PERIODE_VITESSE;
259         TIM3->CNT = 32000;
260     }
261     /* USER CODE END Callback 1 */
262 }

```

A l'interruption du TIM 5, on calcule la vitesse en tour par minute. Pour cela, on récupère la valeur de CNT du TIM 3 qui "compte" le nombre de front montant et descendant et ajoute 32000 (la moitié de la valeur maximale de CNT afin d'être centré autour d'une valeur non nulle et de pouvoir calculer des vitesses négatives). On divise enfin cette valeur par la période durant laquelle on a compté le nombre de front montant et descendant "PERIODE_VITESSE"/60 ("PERIODE_VITESSE est en s). On remet enfin CNT sur sa valeur centrée 32000 et on recommence.

Shell.c :

```
205     else if (strcmp(argv[0], "showspeed")==0)
206     {
207         sprintf(chaine2, "la vitesse est de %d tr.min \r\n", vitesse);
208         HAL_UART_Transmit(&huart2, chaine2, strlen(chaine2), HAL_MAX_DELAY);
209     }
210 }
```

On a également créé une commande shell "showspeed" qui permet à l'utilisateur de demander la valeur de la vitesse du moteur à tout moment.

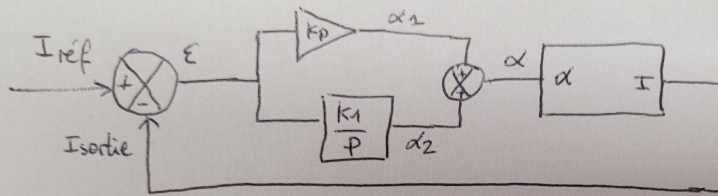
```
--
user@Nucleo-STM32G4RB31>>alpha 56
rapport cyclique
user@Nucleo-STM32G4RB31>>mesure
le courant vaut 2.851758 A
user@Nucleo-STM32G4RB31>>showspeed
la vitesse est de 170 tr.min
```

Voilà une capture du fonctionnement de notre shell.

La vitesse affichée sur la sonde tachymétrique était de 180 tr/min, soit une erreur relative de 5%.

Asservissement en courant du système :

Voici le système que nous essayons de modéliser.



Avec $\alpha_1 = K_p \cdot E$

$$\text{et } \alpha_2[n] = \alpha_2[n-1] + \frac{K_1 \cdot T_e}{2} \cdot (E[n] + E[n-1])$$

Enfin $\alpha = \alpha_1 + \alpha_2$

On a :

- K_p : coefficient proportionnel.
- K_1 : coefficient de l'action intégrale.
- α le rapport cyclique.
- T_e : la période d'échantillonnage du courant (1/160 kHz).

Voilà ce que nous avons fait :

Dans le fichier main.c :

```
47 #define I_CONSIGNE 3
48 #define P 0.0002
49 #define I 3
```

On commence par définir le courant de consigne que l'on souhaite avoir pour le moteur ainsi que P : le coefficient proportionnel et I : coefficient intégral.


```

73 float erreur[2]={0};
74
75 float alpha1=0;
76
77 float alpha2[2]={0};
78 float alpha_corrige[2]={0};
79 int32_t valeur_timer;
80 double Current_loop_buffer[3];
81 extern int sum;
82 extern double Imoyen;
83 extern double mesure_mean;
84 extern double mesure_voltage;
85
86 int i=0;
87 int j=0;

```

On crée les variables dont on aura besoin.
 Ensuite, dans la boucle while on écrit le code ci-dessous :

```

195     for (j=0;j<20;j++)
196     {
197         sum=sum+ADC_Buffer[j];
198     }
199     mesure_mean=sum/10;
200     j=0;
201     courant_buffer[0]=courant_buffer[1];
202     mesure_voltage=((double)mesure_mean*3.3)/4096.0;
203     Imoyen=(mesure_voltage-2.5)*12;
204     courant_buffer[1]=Imoyen;
205     sum=0;
206     erreur[0]=erreur[1];
207     alpha2[0]=alpha2[1];
208     erreur[1]=I_CONSIGNE-courant_buffer[1];
209     alpha_corrige[0]=alpha_corrige[1];
210     alpha1=erreur[1]*P;
211     alpha2[1]=alpha2[0]+((I*Te)/2)*(erreur[1]-erreur[0]);
212     alpha_corrige[1]=alpha_corrige[0]+alpha1+alpha2[1];
213     if (alpha_corrige[1]>50)
214     {
215         alpha_corrige[1]=50;
216     }
217     else if(alpha_corrige[1]<-50)
218     {
219         alpha_corrige[1]=-50;
220     }
221     set_alpha((50+(int)alpha_corrige[1]));
222
223

```

On commence avec la boucle for et les 6 lignes suivantes par mesurer le courant dans le moteur (de la même façon que ce que nous avons fait précédemment dans

la partie "mesure de courant"). Ensuite, nous calculons les α_1 et α_2 comme ce qui avait été décrit sur notre schéma et obtenons un $\alpha_{\text{corrigé}}$.

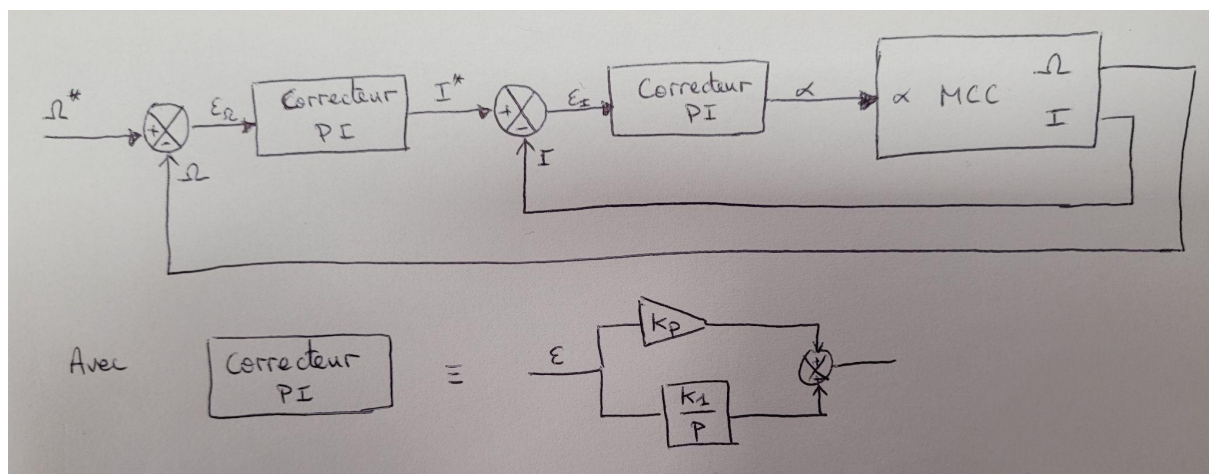
Nous utilisons ensuite la fonction set_alpha avec $50 + \alpha_{\text{corrigé}}$.

Pour une consigne de -3A, voici ce que l'on obtient :

```
COM3 x
user@Nucleo-STM32G4RB31>>mesure
le courant vaut -2.513965 A
user@Nucleo-STM32G4RB31>>mesure
le courant vaut -3.103711 A
user@Nucleo-STM32G4RB31>>mesure
le courant vaut -3.345410 A
user@Nucleo-STM32G4RB31>>mesure
le courant vaut -2.629980 A
```

Asservissement en vitesse :

Voici le schéma de l'asservissement global que nous souhaitons obtenir :



Dans le fichier main.c :

```
46 #define V_CONSIGNE 300
47 #define I_CONSIGNE 0
48 #define P 0.0002
49 #define I 3
50 #define Pv 0.0002
51 #define Iv 3
```

On a mis lconsigne=0 mais on ne l'utilisera plus pour le moment. On ajoute désormais une consigne sur la vitesse et les coefficients proportionnel et intégral pour la vitesse (Pv et Iv).

```
74 float erreur_vitesse[2]={0};
75 float alpha2_v[2]={0};
76 float alpha1_v=0;
77 float alpha_corrige_v[2]={0};
78 extern double courant_buffer[2];
79 float erreur[2]={0};
80
81 float alpha1=0;
82
83 float alpha2[2]={0};
84 float alpha_corrige[2]={0};
85 int32_t valeur_timer;
86 double Current_loop_buffer[3];
87 extern int sum;
88 extern double Imoyen;
89 extern double mesure_mean;
90 extern double mesure_voltage;
91 int flag_vitesse=0;
92 int flag_asserv=0;
```

On crée les variables alpha1_v, alpha2_v, alpha_corrige_v et un flag vitesse que nous allons utiliser par la suite.

```
170 erreur_vitesse[0]=erreur_vitesse[1];
171 alpha2_v[0]=alpha2_v[1];
172 alpha_corrige_v[0]=alpha_corrige_v[1];
173 erreur_vitesse[1]=V_CONSIGNE-vitesse;
174 alpha1_v=erreur_vitesse[1] *Pv;
175 alpha2_v[1]=alpha2_v[0]+((Iv*Te)/2)*(erreur_vitesse[1]-erreur_vitesse[0]);
176 alpha_corrige_v[1]=alpha_corrige_v[0]+alpha1_v+alpha2_v[1];
```

Dans la boucle while :

```

212     if(flag_vitesse==1){
213         erreur_vitesse[0]=erreur_vitesse[1];
214         alpha2_v[0]=alpha2_v[1];
215         alpha_corrige_v[0]=alpha_corrige_v[1];
216         erreur_vitesse[1]=V_CONSIGNE-vitesse;
217         alpha1_v=erreur_vitesse[1]*Pv;
218         alpha2_v[1]=alpha2_v[0]+((Iv*Te)/2)*(erreur_vitesse[1]-erreur_vitesse[0]);
219         alpha_corrige_v[1]=alpha_corrige_v[0]+alpha1_v+alpha2_v[1];
220         flag_vitesse=0;
221     }
222
223
224     if (flagADC==1){
225         for(j=0;j<20;j++)
226         {
227             sum=sum+ADC_Buffer[j];
228         }
229         mesure_mean=sum/10;
230         j=0;
231         courant_buffer[0]=courant_buffer[1];
232         mesure_voltage=((double)mesure_mean*3.3)/4096.0;
233         Imoyen=(mesure_voltage-2.5)*12;
234         courant_buffer[1]=Imoyen;
235         sum=0;
236         erreur[0]=erreur[1];
237
238         alpha2[0]=alpha2[1];
239         //erreur[1]=I_CONSIGNE-courant_buffer[1];
240         erreur[1]=alpha_corrige_v[1]-(float)courant_buffer[1];
241         alpha_corrige[0]=alpha_corrige[1];
242         alpha1=erreur[1]*P;
243         alpha2[1]=alpha2[0]+((I*Te)/2)*(erreur[1]-erreur[0]);
244         alpha_corrige[1]=alpha_corrige[0]+alpha1+alpha2[1];
245         if (alpha_corrige[1]>50)
246         {
247             alpha_corrige[1]=50;
248         }
249         else if(alpha_corrige[1]<-50)
250         {
251             alpha_corrige[1]=-50;
252         }
253         set_alpha((50+(int)alpha_corrige[1]));
254         flagADC=0;
255     }

```

Le flag_vitesse est mis à 1 lorsqu'une mesure de vitesse est effectuée (à une fréquence de 10Hz) et flagADC est mis à 1 lorsqu'une mesure de courant est effectuée (à une fréquence de 16kHz).

L'asservissement en vitesse sort un alpha_corrigé_v qui correspond en fait au courant de commande auquel on asservit notre courant.

Dans notre exemple, la vitesse est bien asservie à 300 tr/min. Néanmoins le temps de réponse est très long, environ 30 secondes. Nous devons régler nos coefficients de Pv et Iv. Nous allons également ajouter un anti-windup afin que le courant donné par alpha2_v ne dépasse pas un certain courant max (que l'on définira à 5A).

```
46 #define V_CONSIGNE 500
47 #define I_CONSIGNE 0
48 #define P 0.0002
49 #define I 3
50 #define Pv 0.0002
51 #define Iv 3
52 #define D 0
53 #define Te 0.00000625
```