MAKERS FOR LIFE

MakAir

Commande du Blower

BOUTRY Loan

Alternant sur le projet MakAir

Ce document fait partie d'une série de livrables concernant la conception et le prototypage d'un respirateur à pression positive continue pour diminuer l'apnée du sommeil d'un patient

Ce livrable a pour but d'expliquer le fonctionnement du blower intégré dans le MakAir. La première partie du document est une analyse des 4 éléments nécessaires pour activer le blower. La deuxième partie portera sur la génération d'une commande PPM et la dernière partie sera sur les résultats de la commande de la turbine.

Table des matières

Table des figures 2
Analyse du système 3
Carte de puissance 3
Carte de contrôle 4
Blower 5
Générateur du signal PPM 6
Génération d'un signal PPM 7
C'est quoi un signal PPM ? 7
Fonction digitalWrite 8
PWM avec un timer/counter 8
Les différentes versions de la commande du Blower 10
Commande simple 10
Variation de la vitesse 10
Annexe 11
Annexe 1 : Librairie « hardwareTimer.h » 11
Annexe 2 : Code de la commande simple 12
Annexe 3 : Variation de la Vitesse 12

Table des figures

Figure 1 : SCP-75-12 3
Figure 2 : Schéma de la carte de contrôle 4
Figure 3 : Blower du MakAir 5
Figure 4 : Caractéristique du blower : pression et débit d'air 5
Figure 5 : Schéma GPIO du STM32F411RE 6
Figure 6 : Schéma complet de la commande du blower 7
Figure 7 : Signal PPM 7
Figure 8 : Code avec la fonction digitalWrite 8
Figure 9 : Principe d'un Timer/Counter 8
Figure 10 : Code de configuration du Timer/Counter 9

Analyse du système

Cette partie énumère les éléments nécessaires au fonctionnement du blower : la carte de puissance, la carte de contrôle, le blower et le générateur du signal PPM. Chaque élément comportera un schéma et des explications sur son rôle dans le système

Carte de puissance

C'est l'élément permettant d'apporter la puissance nécessaire pour que le blower puisse fonctionner dans les conditions optimales. En général, c'est la carte qui va permettre de transformer la tension électrique alternative du secteur (220V ~ 50Hz) en une tension continue (12V dans notre cas). En plus d'apporter la puissance nécessaire à la turbine, nous pouvons coupler le système avec des batteries afin de le rendre autonome en cas d'urgence.

Si nous revenons à notre cas du blower intégré au MakAir, la fiche technique de la carte de contrôle nous indique qu'il peut supporter une tension de 11 à 25V (avec 12V en tension optimale) et tirer un courant continu jusqu'à 3A. Cela nous permet de savoir que l'on doit choisir une carte de puissance avec une sortie de 12 à 24V et ayant un courant ≤ 3A.

Dans notre système nous avons choisi un SCP-75-12 :

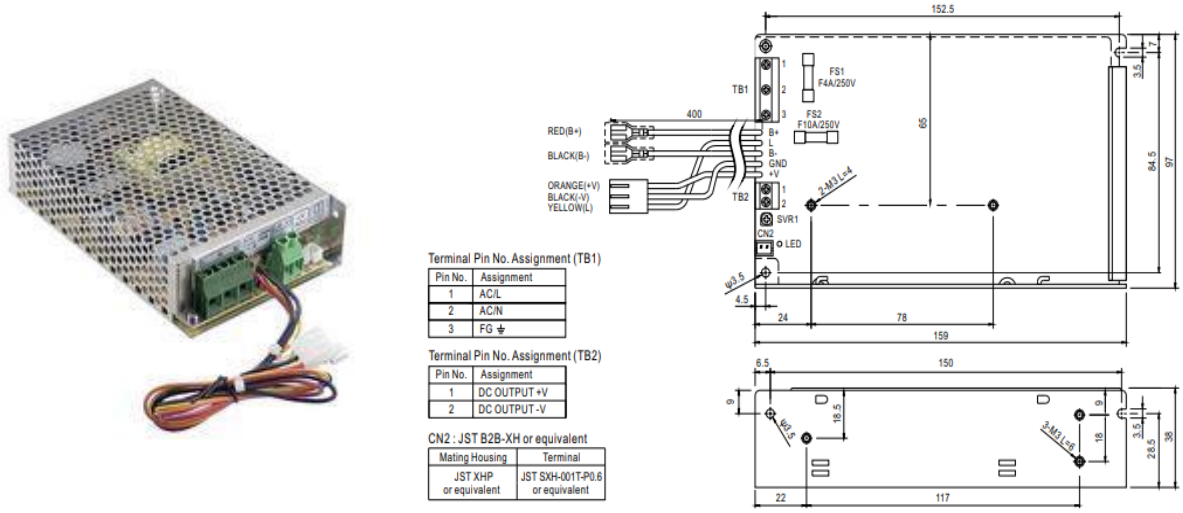

Figure 1 : SCP-75-12

C'est une carte de puissance qui fournit :

Tension nominale	13,8V
Plage de courant	0 ~ 5,4A
Puissance nominale	74,5W

Cette carte sera donc la partie puissance de notre commande du blower.

Carte de contrôle

C'est l'élément qui va permettre de gérer la puissance à envoyer à la turbine et de contrôler le fonctionnement et le sens du blower grâce à 6 transistors MOSFET. Pour comprendre simplement le système : le blower est composé d'un moteur brushless. Celui-ci est un moteur sans balai c'est-à-dire qu'à la place des balais nous avons 3 bobines avec 2 transistors MOSFET par bobine. Ça permet de contrôler le sens du moteur en fonction des bobines actives et des transistors. Un élément peut se rajouter au moteur pour connaître le sens et la vitesse. Dans notre cas, c'est un capteur à effet hall intégré au moteur qui nous renvoie ces données à la carte.
La carte de contrôle va aussi réguler la vitesse du blower selon la consigne envoyée par un signal PPM depuis le microcontrôleur. Cette régulation met en oeuvre des asservissements complexes faisant intervenir le courant passant dans chaque phase (mesuré par des capteurs de courant), et de la position du moteur (mesurée par des capteurs à effet Hall). Elle est réalisée de façon autonome par la carte de contrôle et ne rentre pas dans le cadre de cette étude.

En général, le blower est fourni avec sa carte de contrôle puisqu'elle est compatible seulement avec celle-ci. C'est notre cas avec le blower intégré au MakAir :

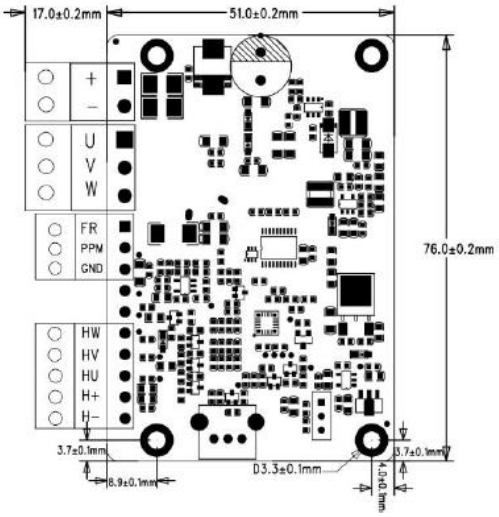

Figure 2 : Schéma de la carte de contrôle

+ -	Sortie + de la carte de puissance (SCP-75-12) Sortie – de la carte de puissance
U V W	12 V pour l'entrée du blower (red) GND pour l'entrée du blower (black) Signal de contrôle du blower (yellow)
FR PPM GND	 Signal PPM de 50Hz envoyé par le générateur GND du générateur de PPM
HW HV HU H+ H- □GND	 Signaux de contrôle du sens et de la vitesse du blower

Nous savons aussi que la variation de la vitesse de blower se fait entre 1,08 ms et 1,86 ms pour le « High Level Pulse » du signal PPM

Blower

C'est l'élément central qui va nous permettre d'envoyer de l'air en pression positive continue. Celui-ci comporte un moteur brushless (sans balai) qui va aspirer l'air environnant pour l'envoyer sous un certain débit. C'est un composant généralement utilisé dans les aspirateurs, VMC ou encore dans les respirateurs médicaux.
Le blower intégré au MakAir est un cas un peu particulier puisque nous avons peu d'informations. Ce que l'on sait :

Plage de tension	6 à 30V
Plage de courant	< 1,4A
Plage de vitesse	0 à 33000 rpm
Pression en sortie	1000 Pa

Ces informations nous permettent de déduire qu'il est plutôt adapté à notre système de respirateur PPC contre l'apnée du sommeil.

On peut aussi retrouver la connectique de la turbine ci-dessous :

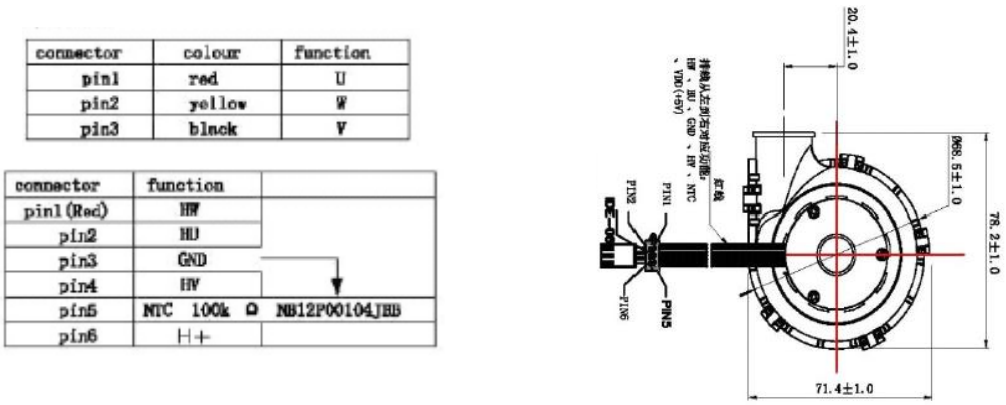

Figure 3 : Blower du MakAir

La caractéristique entre le débit d'air et la pression est donnée pour connaître les points extrêmes du blower (pression max et débit max). Nous pouvons la retrouver ci-dessous :

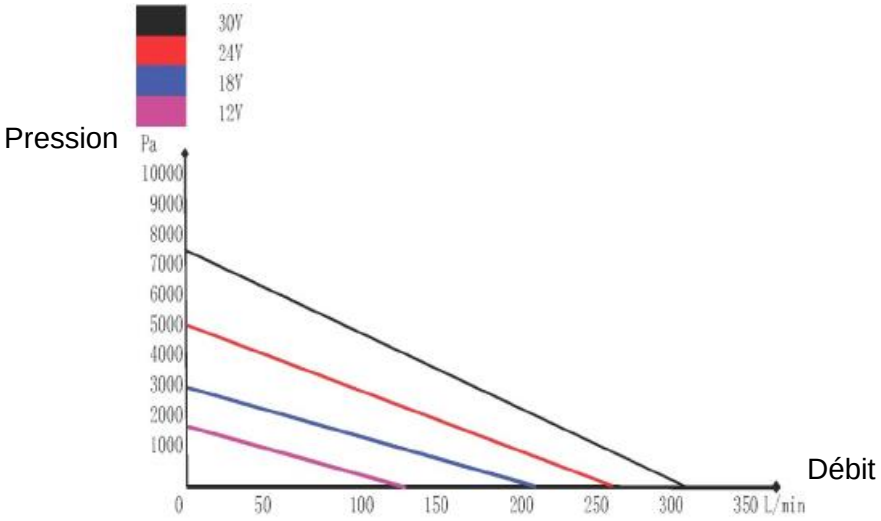

Figure 4 : Caractéristique du blower : pression et débit d'air

Générateur du signal PPM

C'est l'élément essentiel pour gérer la vitesse de la turbine. Dans notre cas, le signal PPM va être généré par un microcontrôleur STM32F411RE. Une des sorties va générer un signal PPM avec des outils intégrés au microcontrôleur : par une sortie numérique ou par timer/counter générant un signal PWM.

Les caractéristiques du microcontrôleur STM32F411RE sont intéressantes car très performantes pour notre système :

- Processeur 32 bits
- Tension d'alimentation de 1,7 à 3,6V
- 512 Kbytes de mémoire Flash et 128 Kbytes de mémoire RAM
- 11 timers : six de 16 bits et deux de 32 bits avec chacun quatre channels Input Compare/Output Compare/PWM/Pulse counter. Ensuite deux watchdog timers et un SysTick timer
- Serial wire debug (SWD) pour le debug
- ADC de 12 bits avec 16 channels
- 81 I/O ports avec une capacité d'interruption
- 3 interfaces I^2C, 3 interfaces USART et 5 interfaces SPI

Pour notre prototypage de la commande du blower, nous avons choisi le port PB6 (D10) pour générer le signal PPM. Le port comporte les fonctions ci-dessous :

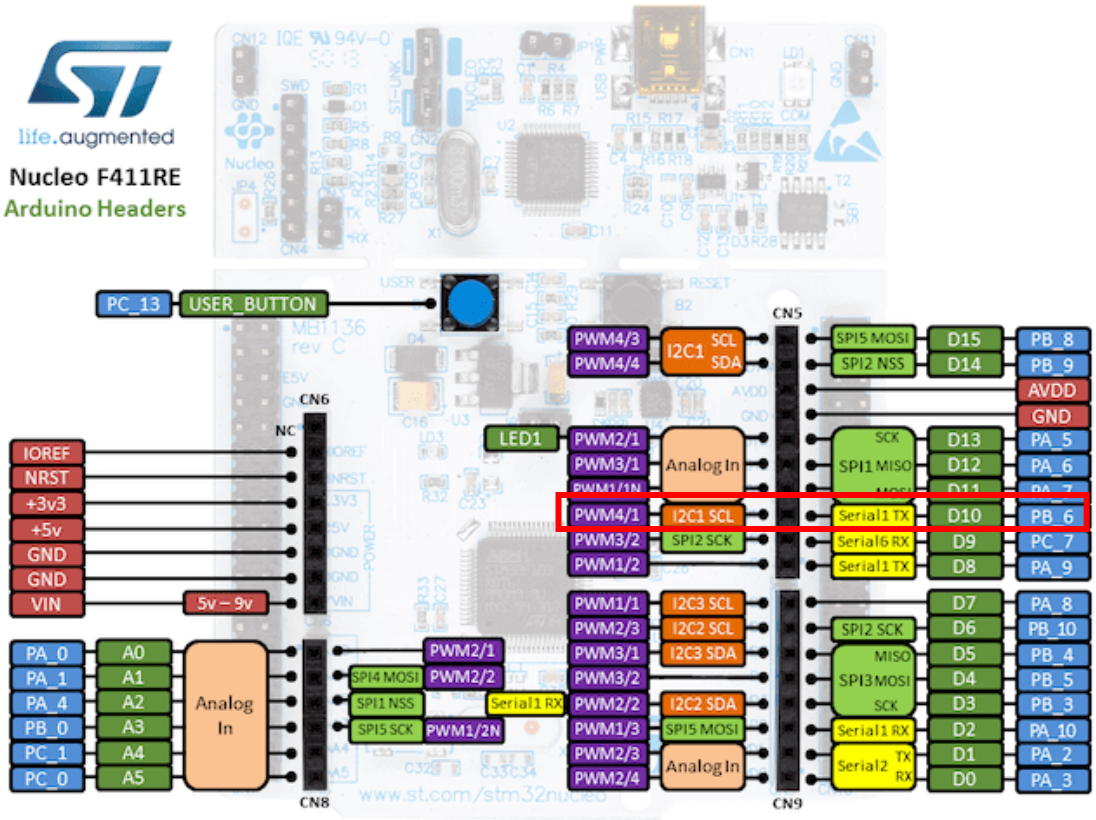

Figure 5 : Schéma GPIO du STM32F411RE

Il est très important de faire attention au timer/counter et au channel PWM correspondant au port choisi. Sur le port D10, nous sommes sur le TIM4 channel 1.

Si on assemble les 4 éléments de la commande du blower, on peut en déduire ce schéma :

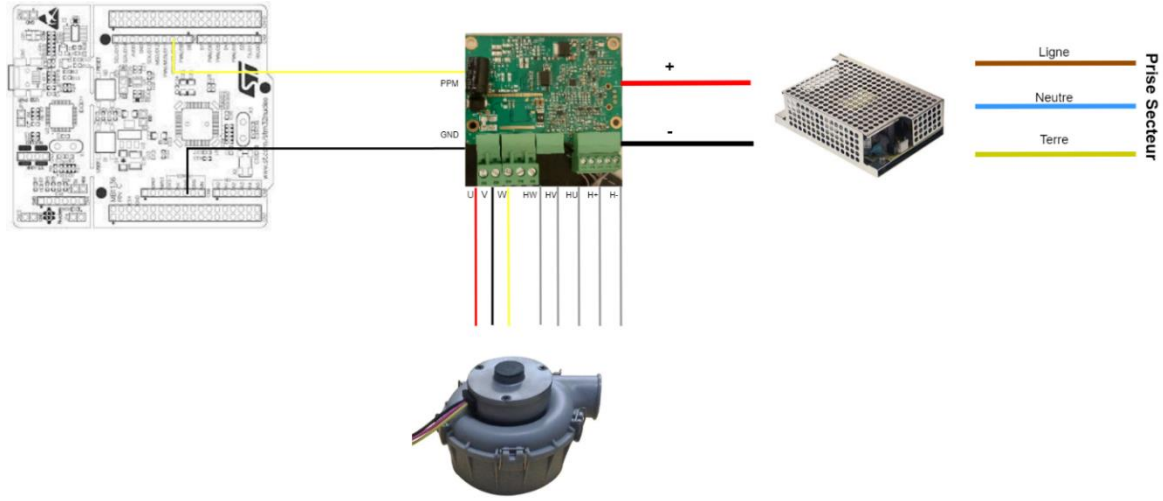

Figure 6 : Schéma complet de la commande du blower

Maintenant que nous avons présenté tous nos éléments, nous allons expliquer comment générer un signal PPM pour activer et contrôler la vitesse.

Génération d'un signal PPM

Cette partie va introduire les fonctions nécessaires pour générer un signal PPM de 50 Hz sur le port PB6. Il y a 2 manières différentes : avec une fonction simple sur une sortie numérique et avec la génération d'une PWM avec un timer/counter.

C'est quoi un signal PPM ?

Un signal PPM (Pulse-Position Modulation) est comparable a un signal PWM sauf qu'à la place d'utiliser un rapport cycle de 0 à 100%, on va varier le signal sur une intervalle de rapport cyclique beaucoup plus courte. Le signal PPM est dépendant d'un signal carré puisque celui-ci est sa référence en période.

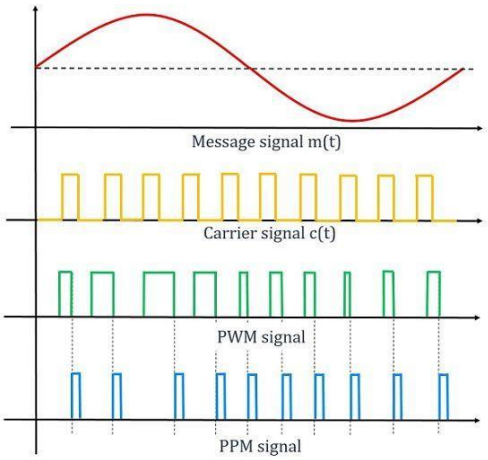

Figure 7 : Signal PPM

C'est un signal souvent utilisé pour la modulation pour faire une liaison « point-à-point » entre un émetteur et récepteur par exemple entre une télécommande et un drone.

Fonction digitalWrite

Une première approche pour générer notre signal PPM consiste à utiliser la fonction digitalWrite, en activant la sortie pendant un court instant.
Cette fonction est très simple à mettre en place mais le principal problème c'est que si nous ajoutons d'autres fonctions (débitmètre, afficheur LCD, …), cette fonction risque d'occuper le microcontrôleur tout le temps à cause des « delayMicroseconds ». Donc la fonction digitalWrite nous permet d'activer le blower et de réguler sa vitesse mais sans pouvoir ajouter d'autres fonctions importantes pour notre système.

```
#define PIN_ESC_BLOWER PB6  // D10

void setup() {

 pinMode(PIN_ESC_BLOWER,OUTPUT);

void loop() {
 digitalWrite(PIN_ESC_BLOWER, LOW);
 Min = delayMicroseconds(1750); ou Max = delayMicroseconds(50);
 digitalWrite(PIN_ESC_BLOWER, HIGH);
 Min = delayMicroseconds(250); ou Max = delayMicroseconds(1950);
}
```

Figure 8 : Code avec la fonction digitalWrite

Précision sur la fonction AnalogWrite :

Le blower va pouvoir être activé mais ne va pas pouvoir être commandé en vitesse à cause de la fréquence PWM en sortie de la fonction AnalogWrite. La cause est dû à la précision de la génération du signal PPM de 50 Hz puisque la sortie numérique ne peut générer qu'une PWM de 1,1 kHz. Cela n'est pas suffisant pour avoir une précision de variation du « High Level Pulse » inférieure 1 ms (pour varier entre 1,08 et 1,86 ms).

Avec une sortie numérique qui génère une fréquence de 1,1 kHz, la précision de la variation du rapport cyclique est de 0,9 ms. Donc on peut en déduire que c'est trop grand pour faire varier le signal entre 1,08 et 1,86 ms.

PWM avec un timer/counter

Cette fonction est plus complexe mais permet d'avoir une commande beaucoup plus précise sur la vitesse du blower. Nous allons utiliser un élément important d'un processeur, c'est le Timer/Counter qui permet généralement de déclencher une instruction/fonction lorsque le Timer a atteint le Flag que l'on a défini lors de la configuration.

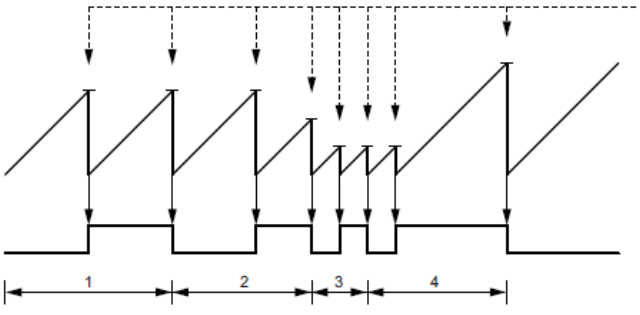

Figure 9 : Principe d'un Timer/Counter

Les Timer/Counter ont généralement 4 modes d'utilisation : IC (Input Compare), OC (Output Compare), PWM et Pulse counter. C'est le mode PWM qui nous intéresse dans notre cas puisque celui-ci va générer un signal PWM en fonction de la configuration attribuée.

Pour utiliser le timer/counter sur le STM32F411RE, nous allons prendre la librairie « hardwareTimer » pour avoir des fonctions configurant le timer/counter sur le port PB6 (celui-ci correspond au TIM4 channel 1). Voici comment se présente le code :

```
#define ESC_PPM_PERIOD 20000 // 50Hz
#define MIN_BLOWER_SPEED 300u
#define MAX_BLOWER_SPEED 1800u
#define DEFAULT_BLOWER_SPEED 900u
#define PIN_ESC_BLOWER  PB6  // D10 / TIM4_CH1
#define TIM_CHANNEL_ESC_BLOWER 1 //Channel 1 de la pin D10
#define BlowerSpeed2MicroSeconds(value) map(value, 0, 1800, 1000, 1950) //Intervalle [1ms à
1,95 ms] sur la variation de vitesse du blower

#include "HardwareTimer.h"

HardwareTimer* hardwareTimer1;  // Timer Blower command

hardwareTimer1 = new HardwareTimer(TIM4); //Sélectionne le timer 4 pour la pin D10
hardwareTimer1->setOverflow(ESC_PPM_PERIOD, MICROSEC_FORMAT); //Définit la période/fréquence
de la PWM
hardwareTimer1->refresh(); //réninitialise le timer
hardwareTimer1->setMode(TIM_CHANNEL_ESC_BLOWER, TIMER_OUTPUT_COMPARE_PWM1, PIN_ESC_BLOWER);
//Permet de mettre le mode PWM sur le timer (Autres modes : Input Capture/Ouput Capture/Pulse
Counter
    // Set PPM width to 1ms
hardwareTimer1->setCaptureCompare(TIM_CHANNEL_ESC_BLOWER,
BlowerSpeed2MicroSeconds(DEFAULT_BLOWER_SPEED), MICROSEC_COMPARE_FORMAT); //Définit le rapport
cyclique de la PWM
hardwareTimer1->resume(); //Lance le timer
```

Figure 10 : Code de configuration du Timer/Counter

Avant de commencer, il est conseillé d'aller regarder la librairie « hardwareTimer.h » afin de voir toutes les méthodes de la librairie et les différents attributs (Voir Annexe 1). Nous allons décrire les différentes fonctions pour connaître leurs paramètres et rôles :

- HardwareTimer* hardwareTimer1 : Cette commande permet de faire appel à la librairie hardwareTimer.h à travers un objet. C'est grâce à cet objet que l'on va pouvoir utiliser les fonctions de la librarie « hardwareTimer »

- hardwareTimer1 = new HardwareTimer(TIM4) : Cette commande permet d'instancier l'objet HardwareTimer en sélectionnant TIM4

- hardwareTimer1->setOverflow(ESC_PPM_PERIOD, MICROSEC_FORMAT) : Cette commande permet de configurer la période du timer, dans notre cas ça va définir la période du signal PWM à 50 Hz.

- hardwareTimer1->refresh() : Cette commande met à jour tous les registres du timer

- hardwareTimer1->setMode(TIM_CHANNEL_ESC_BLOWER,TIMER_OUTPUT_COMPARE_PWM1, PIN_ESC_BLOWER) : Cette commande permet de sélectionner le channel du timer, le mode du timer (PWM dans notre cas) et sur quel port (PB6).

- hardwareTimer1->setCaptureCompare(TIM_CHANNEL_ESC_BLOWER, BlowerSpeed2MicroSeconds(DEFAULT_BLOWER_SPEED), MICROSEC_COMPARE_FORMAT) : Cette commande permet de sélectionner le channel et définir le Flag (la limite) du timer. Cela va définir le rapport cyclique de notre signal PWM et donc de réaliser un

signal PPM à laquelle son High Level Pulse va varier entre 1,06 ms et 1,86 ms (c'est BlowerSpeed2MicroSeconds qui va permettre cette variation)

- hardwareTimer1->resume() : Cette commande permet de lancer le timer et d'activer le channel du timer sélectionné.

En configurant tous les paramètres nécessaires, vous avez un signal PWM de 50 Hz qui est généré sur le port PB6. Vous pouvez modifier le rapport cyclique en variant la valeur dans BlowerSpeed2MicroSeconds. A présent, vous pouvez commander la vitesse du blower.

Les différentes versions de la commande du Blower

Cette partie montre les différentes versions de la commande du Blower : Une commande simple du blower, la variation de la vitesse avec des boutons poussoirs.

Commande simple

Vous pouvez retrouver le code en Annexe 2

Cette version est une commande simple du blower c'est-à-dire que l'on définit la vitesse de la turbine avant la compilation du code. Nous aurons juste un débit d'air envoyé constant.

Variation de la vitesse

Vous pouvez retrouver le code en Annexe 3

Cette version est une commande permettant de varier la vitesse du blower avec des boutons poussoirs. On ajoute donc 2 boutons au système : un bouton pour augmenter la vitesse (PB10) et un bouton pour la diminuer (PA8). On ajoute une fonction anti-rebond à chaque bouton pour commander la turbine plus précisément et éviter les valeurs non désirées. Dans cette version nous ajoutons cette commande au timer :

- hardwareTimer1->attachInterrupt(TIM_CHANNEL_ESC_BLOWER,expiration) : Cette commande permet de déclencher la fonction expiration à chaque fois que le timer atteint son Flag.

D'autres versions peuvent être ajoutées comme l'ajout d'un débitmètre ou encore le calcul du débit d'air envoyé par la turbine affiché sur un écran LCD.

Annexe

Annexe 1 : Librairie « hardwareTimer.h »

Définition des modes du timer :

```
typedef enum {
  TIMER_DISABLED,                       // == TIM_OCMODE_TIMING           no output, useful for only-
interrupt
  // Output Compare
  TIMER_OUTPUT_COMPARE,                 // == Obsolete, use TIMER_DISABLED instead. Kept for
compatibility reason
  TIMER_OUTPUT_COMPARE_ACTIVE,          // == TIM_OCMODE_ACTIVE           pin is set high when counter ==
channel compare
  TIMER_OUTPUT_COMPARE_INACTIVE,        // == TIM_OCMODE_INACTIVE         pin is set low when counter ==
channel compare
  TIMER_OUTPUT_COMPARE_TOGGLE,          // == TIM_OCMODE_TOGGLE           pin toggles when counter ==
channel compare
  TIMER_OUTPUT_COMPARE_PWM1,            // == TIM_OCMODE_PWM1             pin high when counter < channel
compare, low otherwise
  TIMER_OUTPUT_COMPARE_PWM2,            // == TIM_OCMODE_PWM2             pin low when counter < channel
compare, high otherwise
  TIMER_OUTPUT_COMPARE_FORCED_ACTIVE,   // == TIM_OCMODE_FORCED_ACTIVE    pin always high
  TIMER_OUTPUT_COMPARE_FORCED_INACTIVE, // == TIM_OCMODE_FORCED_INACTIVE  pin always low

  //Input capture
  TIMER_INPUT_CAPTURE_RISING,           // == TIM_INPUTCHANNELPOLARITY_RISING
  TIMER_INPUT_CAPTURE_FALLING,          // == TIM_INPUTCHANNELPOLARITY_FALLING
  TIMER_INPUT_CAPTURE_BOTHEDGE,         // == TIM_INPUTCHANNELPOLARITY_BOTHEDGE

  // Used 2 channels for a single pin. One channel in TIM_INPUTCHANNELPOLARITY_RISING another channel in
TIM_INPUTCHANNELPOLARITY_FALLING.
  // Channels must be used by pair: CH1 with CH2, or CH3 with CH4
  // This mode is very useful for Frequency and Dutycycle measurement
  TIMER_INPUT_FREQ_DUTY_MEASUREMENT,

  TIMER_NOT_USED = 0xFFFF  // This must be the last item of this enum
} TimerModes_t;
```

Définition des unités de la période du timer :

```
typedef enum {
  TICK_FORMAT, // default
  MICROSEC_FORMAT,
  HERTZ_FORMAT,
} TimerFormat_t;
```

Définition du format pour configurer le rapport cyclique :

```
typedef enum {
  TICK_COMPARE_FORMAT = 0x80, // default
  MICROSEC_COMPARE_FORMAT,
  HERTZ_COMPARE_FORMAT,
  PERCENT_COMPARE_FORMAT, // used for Dutycycle
} TimerCompareFormat_t;
```

Méthodes importantes :

```
HardwareTimer(TIM_TypeDef *instance);
void resume(void); // Resume counter and all output channels
void refresh(void);
void setOverflow(uint32_t val, TimerFormat_t format = TICK_FORMAT); // set AutoReload register
depending on format provided
void setMode(uint32_t channel, TimerModes_t mode, uint32_t pin);
void setCaptureCompare(uint32_t channel, uint32_t compare, TimerCompareFormat_t format =
TICK_COMPARE_FORMAT);  // set Compare register value of specified channel depending on format
provided
void attachInterrupt(callback_function_t callback); // Attach interrupt callback which will be
called upon update event (timer rollover)
```

Annexe 2 : Code de la commande simple

```
#define ESC_PPM_PERIOD 20000 // 50Hz
#define MIN_BLOWER_SPEED 300u
#define MAX_BLOWER_SPEED 1800u
#define DEFAULT_BLOWER_SPEED 900u
#define PIN_ESC_BLOWER  PB6  // D10 / TIM4_CH1
#define TIM_CHANNEL_ESC_BLOWER 1 //Channel 1 de la pin D10
#define BlowerSpeed2MicroSeconds(value) map(value, 0, 1800, 1000, 1950) //Intervalle [1ms à
1,95 ms] sur la variation de vitesse du blower

#include "HardwareTimer.h"

HardwareTimer* hardwareTimer1;  // Timer Blower command

void setup() {
  // put your setup code here, to run once:
 PWM_Blower();
}

void PWM_Blower(){

 pinMode(PIN_ESC_BLOWER, OUTPUT);
 hardwareTimer1 = new HardwareTimer(TIM4); //Sélectionne le timer 4 pour la pin D10
 hardwareTimer1->setOverflow(ESC_PPM_PERIOD, MICROSEC_FORMAT); //Définit la période/fréquence
de la PWM
 hardwareTimer1->refresh(); //réninitialise le timer
 hardwareTimer1->setMode(TIM_CHANNEL_ESC_BLOWER, TIMER_OUTPUT_COMPARE_PWM1, PIN_ESC_BLOWER);
//Permet de mettre le mode PWM sur le timer (Autres modes : Input Capture/Ouput Capture/Pulse
Counter
    // Set PPM width to 1ms
 hardwareTimer1->setCaptureCompare(TIM_CHANNEL_ESC_BLOWER,
BlowerSpeed2MicroSeconds(DEFAULT_BLOWER_SPEED), MICROSEC_COMPARE_FORMAT); //Définit le rapport
cyclique de la PWM
hardwareTimer1->resume(); //Lance le timer

}
```

Annexe 3 : Variation de la Vitesse

```
#define ESC_PPM_PERIOD 20000 // 50Hz
#define MIN_BLOWER_SPEED 300u
#define MAX_BLOWER_SPEED 1800u
#define DEFAULT_BLOWER_SPEED 900u
#define PIN_ESC_BLOWER  PB6  // D10 / TIM4_CH1
#define TIM_CHANNEL_ESC_BLOWER 1 //Channel 1 de la pin D10
#define BlowerSpeed2MicroSeconds(value) map(value, 0, 1800, 1000, 1950)
//Intervalle [1ms à 1,95 ms] sur la variation de vitesse du blower
#include "HardwareTimer.h"
#include <LiquidCrystal.h>

HardwareTimer* hardwareTimer1;  // Timer Blower command

LiquidCrystal lcd(PB3, PB4, PA9, PC7, PA7, PA6);
//int i = 0;
static int speedBlower = DEFAULT_BLOWER_SPEED;

///////Debounce/////////
int reading, reading2 = 0;
int buttonState = 1;             // the current reading from the input pin
int lastButtonState = 0;  // the previous reading from the input pin
int buttonState2 = 1;             // the current reading from the input pin
int lastButtonState2 = 0;  // the previous reading from the input pin
// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an int.
unsigned long lastDebounceTime = 50;  // the last time the output pin was toggled
unsigned long debounceDelay = 0;    // the debounce time; increase if the output
flickers
unsigned long lastDebounceTime2 = 50;  // the last time the output pin was toggled
```

```
unsigned long debounceDelay2 = 0;    // the debounce time; increase if the output
flickers
//////////////////////

void setup() {
  // put your setup code here, to run once:
 Serial.begin(115200);
 lcd.begin(20, 4);
 pinMode(PA8, INPUT_PULLUP); //D7
 pinMode(PB10, INPUT_PULLUP); //D6
 PWM_Blower();
}

void loop() {
  // put your main code here, to run repeatedly:
reading = digitalRead(PA8);
reading2 = digitalRead(PB10);
}

void PWM_Blower(){
 pinMode(PIN_ESC_BLOWER, OUTPUT);
 hardwareTimer1 = new HardwareTimer(TIM4); //Sélectionne le timer 4 pour la pin D10
 hardwareTimer1->setOverflow(ESC_PPM_PERIOD, MICROSEC_FORMAT); //Définit la
période/fréquence de la PWM
 //hardwareTimer1->setOverflow(50, HERTZ_FORMAT);
 hardwareTimer1->refresh(); //réninitialise le timer
 hardwareTimer1->setMode(TIM_CHANNEL_ESC_BLOWER, TIMER_OUTPUT_COMPARE_PWM1,
PIN_ESC_BLOWER); //Permet de mettre le mode PWM sur le timer (Autres modes : Input
Capture/Ouput Capture/One-pulse
    // Set PPM width to 1ms
 hardwareTimer1->setCaptureCompare(TIM_CHANNEL_ESC_BLOWER,
BlowerSpeed2MicroSeconds(DEFAULT_BLOWER_SPEED), MICROSEC_COMPARE_FORMAT);
 //hardwareTimer1->setCaptureCompare(TIM_CHANNEL_ESC_BLOWER, 10 ,
PERCENT_COMPARE_FORMAT); //Définit le rapport cyclique de la PWM
 hardwareTimer1->attachInterrupt(TIM_CHANNEL_ESC_BLOWER,expiration); //Active la
fonction à chaque interruption
 hardwareTimer1->resume(); //Lance le timer
}

void expiration(){
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;
      // only toggle the LED if the new button state is HIGH
      if (buttonState == 0) {
          speedBlower -= 10;
      }
    }
  }
  lastButtonState = reading;

  if (reading2 != lastButtonState2) {
    // reset the debouncing timer
    lastDebounceTime2 = millis();
  }
  if ((millis() - lastDebounceTime2) > debounceDelay2) {
    // if the button state has changed:
    if (reading2 != buttonState2) {
      buttonState2 = reading2;
      // only toggle the LED if the new button state is HIGH
      if (buttonState2 == 0) {
          speedBlower += 10;
      }
```

```
    }
  }
  lastButtonState2 = reading2;
    /*if (digitalRead(PA8) == 0) {
        speedBlower -= 10;
     }
     if (digitalRead(PB10) == 0) {
        speedBlower += 10;
     }*/

     if(speedBlower <= 100)  speedBlower = 100;
     if(speedBlower >= 1800)  speedBlower = 1800;
     lcd.setCursor(0,0);
     lcd.print(speedBlower);
     hardwareTimer1->setCaptureCompare(TIM_CHANNEL_ESC_BLOWER,
BlowerSpeed2MicroSeconds(speedBlower), MICROSEC_COMPARE_FORMAT);
     hardwareTimer1->resume(); //Lance le timer
}
```