



MAKERS FOR LIFE



MakAir

Interface Homme-Machine

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 la réalisation d'une Interface Homme-Machine sur un système. C'est l'élément important pour l'ergonomie et l'accessibilité de l'utilisateur lors de l'utilisation du système. Nous allons nous concentrer sur 2 éléments : l'Affichage et la Configuration

Table des matières

Table des figures	2
C'est quoi une IHM ?	3
1 ^{ère} composante de l'IHM : L'affichage	4
Mise en place de l'afficheur	4
Affichage d'une chaîne de caractère	5
Intégration de la librairie LCD	5
2 ^{ème} composante de l'IHM : La configuration	7
Mise en place de la configuration	7
Bouton Poussoir : Debounce (Anti rebond)	8
Intégration de l'affichage et de la configuration	9
Annexe	10
Annexe 1 : Tableau des caractères	10
Annexe 2 : Affichage « Hello World ! »	11
Annexe 3 : Configuration d'une variable avec un debounce	11
Annexe 4 : Programme IHM	12

Table des figures

Figure 1 : Schéma de l'afficheur Displaytech 204A	4
Figure 2 : Afficher le symbole "A"	5
Figure 3 : Position (0,0) de l'afficheur	6
Figure 4 : Affichage de "Hello World!"	6
Figure 5 : Schéma Input_pullup	7
Figure 6 : Schéma Input_pulldown	7
Figure 7 : Programme simple de configuration	8
Figure 8 : Principe du rebond sur un bouton poussoir	8
Figure 9 : Programme Debounce	8
Figure 10 : Fonction de configuration	9
Figure 11 : Fonction d'affichage	9

C'est quoi une IHM ?

IHM signifie interface homme-machine et fait référence à un tableau de bord qui permet à un utilisateur de communiquer avec une machine, un programme informatique ou un système. Techniquement, vous pourriez appliquer le terme IHM à n'importe quel écran utilisé pour interagir avec un appareil, mais il est généralement employé pour décrire des écrans utilisés dans les environnements industriels. Les IHM affichent des données en temps réel et permettent à l'utilisateur de contrôler les machines grâce à une interface utilisateur graphique. Elles sont principalement composées de 2 éléments :

- L'Affichage qui va permettre de communiquer avec l'utilisateur afin qu'il puisse analyser les données du système. On peut avoir plusieurs paramètres sur l'affichage :
 - **La technologie d'affichage** : C'est le premier paramètre à prendre en compte puisqu'il est important de savoir si l'on cherche de la performance, de la qualité, ou du low-cost. Nous pouvons retrouver différentes technologies : LCD, OLED, TFT ou encore 7 segments.
 - **La taille de l'écran** : Ce paramètre va dépendre du type d'information qu'il faut communiquer. Si l'on veut afficher des courbes, on va privilégier des écrans plus grands mais forcément plus chers et complexes. Mais si l'on veut juste afficher quelques données concernant le système, on va prendre un écran plus petit.
 - **L'Interface de communication** : le dernier paramètre important est la communication entre le microcontrôleur et l'afficheur puisque c'est par cette voie que les données vont être envoyées et reçues. Cela peut se faire avec une communication en série, une communication I2C ou encore une communication SPI.
 - **Le choix du tactile** : Ce paramètre dépend si l'on veut unir l'affichage et la configuration afin de gagner de l'espace sur le système ou pour avoir IHM plus ergonomique.
- La Configuration est l'autre élément qui compose une IHM. C'est celui-ci qui va permettre à l'utilisateur de modifier des données sur le système afin de modifier son comportement. 2 paramètres sont importants sur la configuration :
 - **La technologie** : Ce paramètre est un choix qui est réalisé lors de la mise en forme du cahier des charges. Dans les systèmes, on a en général 4 composants bien distincts : le bouton poussoir, le potentiomètre, le tactile ou encore le joystick. Ces 4 technologies ont chacune leurs avantages qui dépendent de la donnée qui est modifiée sur le système.
 - Bouton poussoir → Plus fiable et simple d'intégration
 - Potentiomètre → Sortie analogique
 - Tactile → Moins encombrant et intégré à l'afficheur
 - Joystick → Commande direct du système par l'utilisateur
 - **La sortie du composant** : Ce paramètre est lié à la technologie mais il est important de le connaître pour savoir si l'on veut une sortie analogique (qui va sortir des valeurs continues) sur un paramètre (ex : pour modifier le volume d'une enceinte) ou si l'on veut une sortie analogique sur plusieurs paramètres (ex : modifier l'axe X et Y d'un système).

Pour la réalisation du respirateur PPC, on choisit un afficheur LCD pour sa simplicité d'utilisation et pour le besoin du système (4 informations max à afficher). Pour la configuration, on choisit des boutons poussoirs pour la simplicité d'intégration et la fiabilité de la technologie.

1^{ère} composante de l'IHM : L'affichage

Mise en place de l'afficheur

On va commencer par la mise en place de l'affichage. Comme dit précédemment, nous choisissons un afficheur LCD (Displaytech 204A) avec une communication en série. Voici son schéma :

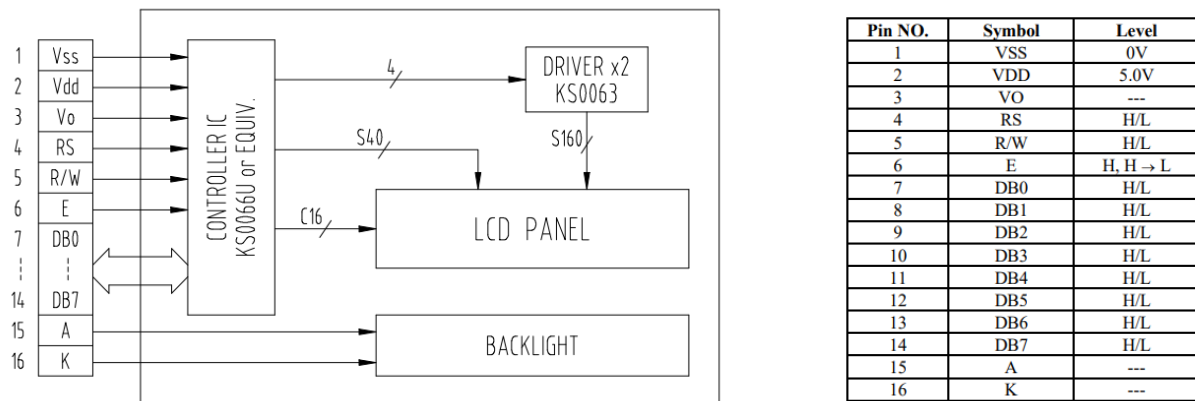


Figure 1 : Schéma de l'afficheur Displaytech 204A

D'un point de vue pédagogique, la communication en série permet d'observer les différents signaux importants pour transmettre des données entre un composant et un microcontrôleur. Ces signaux ont chacun un rôle :

- **VSS** : C'est la masse du composant (GND)
- **VDD** : C'est la tension d'alimentation (5V)
- **VO** : C'est la tension d'entrée du LCD (5V)
- **RS** : C'est le signal qui définit si l'afficheur est en émission ou en réception (D2)
- **R/W** : C'est le signal qui définit si le composant est en écriture ou en lecture. Dans notre cas, nous voulons seulement afficher des données donc on met ce signal sur GND (en mode écriture)
- **E** : C'est le signal d'activation de l'afficheur (D5)
- **DB0...DB7** : Ce sont les signaux de données qui vont écrire sur l'afficheur la chaîne de caractères. (D8, D9, D11, D12)
- **A/K** : Ce sont les signaux de la diode du LCD. Celui-ci permet d'allumer le fond d'écran de l'afficheur. A = Anode (5V) et K = Cathode (GND), pour compléter, il est important de mettre une résistance afin de gérer la luminosité de l'affichage.

Les symboles qui vont être affichés par l'afficheur dépendent de la valeur des signaux « DB ». Par exemple si l'on veut afficher un A, il faut que DB0..3 = 0100 et DB4..7 = 0001 :

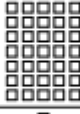
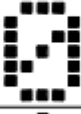
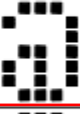



upper 4 bit lower 4 bit	0000	0010	0011	0100
0000	CG RAM (1)			
0001	(2)			

Figure 2 : Afficher le symbole "A"

Vous pouvez retrouver le tableau entier en *Annexe 1*.

Les éléments expliqués sont les principaux paramètres à comprendre afin d'afficher des caractères sur un afficheur LCD. Maintenant, nous pouvons programmer un simple programme qui va afficher une chaîne de caractères.

Affichage d'une chaîne de caractère

Pour afficher une chaîne de caractères, il y a plusieurs étapes :

- *Intégration de la librairie LCD*
- *Initialisation de l'afficheur et de la communication*
- *Afficher la chaîne de caractères sur une position choisie*

Intégration de la librairie LCD

L'intégration d'une librairie dans Arduino IDE est expliquée dans la fiche « Tuto STM32duino ». Il faut suffir de récupérer la librairie : « LiquidCrystal » est de l'installer sur l'IDE. Ensuite pour l'intégrer à votre programme, il vous faut écrire cette ligne de code :

```
#include <LiquidCrystal.h>
```

Initialisation de l'afficheur et de la communication

L'initialisation de l'afficheur et de la communication est importante pour définir les pins de contrôle de l'afficheur et créer le bus de données avec le microcontrôleur. Cela va se faire par 2 instructions :

- `LiquidCrystal lcd(PA10, PB4, PA9, PC7, PA7, PA6);` : Cette instruction va permettre de définir les pins de l'afficheur. La définition dans notre cas se fait sur cet ordre : (RS, E, DB4, DB5, DB6, DB7). Il existe d'autres définitions que vous pouvez retrouver dans la librairie « LiquidCrystal.h ».
- `lcd.begin(20, 4);` : Cette instruction permet d'initier la communication entre l'afficheur LCD et le microcontrôleur. On peut voir aussi qu'il faut définir la taille des caractères de l'afficheur (ici 20x4 caractères). On met cette instruction dans le setup

Afficher la chaîne de caractères sur une position choisie

La dernière étape est d'afficher la chaîne de caractères sur une position de l'afficheur. Pour cela il nous faut seulement 2 instructions :

- `lcd.setCursor(0, 0);` : Cette instruction permet de sélectionner la position où l'on veut afficher la chaîne de caractères. Ici on choisit la position 0/20 et 0/4 ce qui correspond à la position en haut à gauche.

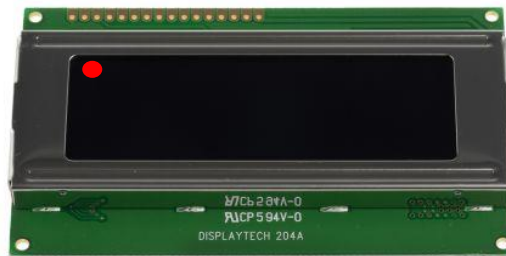


Figure 3 : Position (0,0) de l'afficheur

Si l'on veut afficher en bas de l'écran, on va mettre `lcd.setCursor(0,4)`.

- `lcd.print("Hello World!");` : Cette instruction va permettre d'écrire la chaîne de caractères à afficher sur l'écran LCD. Il suffit de mettre en guillemets la chaîne de caractères (String) à écrire. Si c'est une variable à afficher, il suffit juste d'enlever les guillemets et de mettre le nom de la variable : `lcd.print(variable);`

En écrivant ces instructions, vous allez pouvoir avoir un premier résultat qui est l'affichage de « Hello World! » et la valeur de votre variable sur votre afficheur :



Figure 4 : Affichage de "Hello World!"

Maintenant que notre afficheur fonctionne, nous pouvons passer à la configuration

2^{ème} composante de l'IHM : La configuration

Mise en place de la configuration

Pour notre IHM, nous choisissons des boutons poussoirs pour leur simplicité d'intégration et leur fiabilité. C'est aussi un composant simple à mettre en place au niveau Software puisqu'il faut simplement initier et lire sa broche de contrôle :

- `pinMode(PB5, INPUT_PULLUP);` : Cette instruction va nous permettre d'initialiser la pin PB5 sur le mode INPUT_PULLUP. Ce mode est adapté pour ce genre de composants. Sa particularité peut se représenter par un schéma électrique :

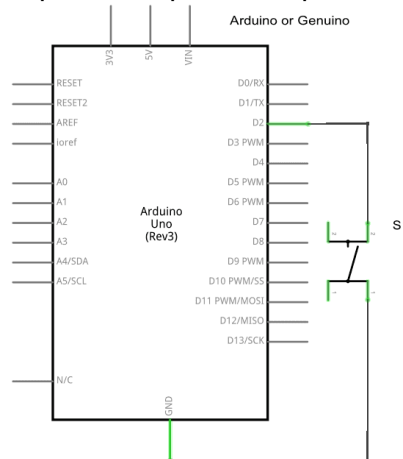


Figure 5 : Schéma Input_pullup

Dès que le bouton sera appuyé, le pin sera au niveau bas car il sera tiré par la masse (GND). Un autre mode existe, c'est le mode INPUT_PULLDOWN qui est le même principe mais cette fois la broche va être tiré par le 5V quand le bouton est appuyé :

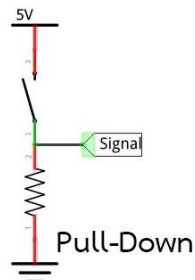


Figure 6 : Schéma Input_pulldown

On choisit le mode INPUT_PULLUP pour une question de sécurité puisque l'on ne peut pas avoir un signal GND et non avec une tension d'alimentation.

- `reading = digitalRead(PB5);` : Cette instruction permet de lire l'état de la broche D4. Avec le mode INPUT_PULLUP, le bouton est appuyé lors que l'on récupère un 0 (état bas).

Avec ces lignes de code, vous pouvez configurer des variables comme-ci-dessous :

```
void setup(){
    pinMode(PB5, INPUT_PULLUP);
}

void loop(){
    reading = digitalRead(PB5);
    if (reading == 0) {
        value -= variation;
    }
}
```

Figure 7 : Programme simple de configuration

Bouton Poussoir : Debounce (Anti rebond)

Vous pouvez maintenant configurer des variables. Mais vous allez rencontrer un problème particulier au bouton poussoir : c'est le rebond (bounce) qui va déclencher plusieurs fois le bouton malgré un appui de l'utilisateur :

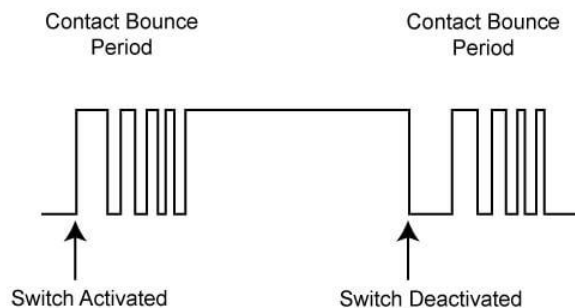


Figure 8 : Principe du rebond sur un bouton poussoir

C'est un problème qu'il faut résoudre pour avoir une IHM fiable. Pour cela il faut réaliser une fonction anti rebond (debounce) au niveau software. Cela peut se faire au niveau électronique mais c'est plus complexe pour notre système. Voici la fonction « debounce » :

```
int reading = 0;
int buttonState = 1; // état courant du bouton
int lastButtonState = 0; // état précédent du bouton
unsigned long lastDebounceTime = 0; // le temps précédent du changement d'état du bouton
unsigned long debounceDelay = 50; // Delai de l'anti rebond, l'augmenter si il y a encore des rebonds sur le bouton

lastDebounceTime = debounceTime;
reading = digitalRead(PB5);
if (reading != lastButtonState) {
    // réinitialisation de l'anti-rebond
    lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) > debounceDelay) {
    // Condition du changement d'état du bouton
    if (reading != buttonState) {
        buttonState = reading;

        // Instruction exécutée si le bouton est LOW
        if (buttonState == 0) {
            value -= variation;
        }
    }
}
lastButtonState = reading;
```

Figure 9 : Programme Debounce

Le principe est assez simple : On va récupérer le temps de « bounce » du bouton et grâce au seuil défini par la variable « debounceDelay », on va garder l'état actuel du bouton pendant le temps de « bounce » récupéré afin que le système ne prenne pas en compte les rebonds du bouton. La fonction s'active lorsqu'il y a un changement d'état sur le bouton.

En intégrant cette fonction, vous n'allez plus avoir de problème de rebond sur votre bouton et améliorer l'IHM. Maintenant nous allons intégrer les 2 composantes

Intégration de l'affichage et de la configuration

L'intégration est assez simple puisque ce sont 2 composantes indépendantes de chacune, il suffit de faire une fonction la configuration et une fonction pour l'affichage comme-ci-dessous :

```
double debounceLess(int buttonPin, int debounceTime, double value, double variation, double
limite) {
    pinMode(buttonPin, INPUT_PULLUP);
    lastDebounceTime = debounceTime;
    reading = digitalRead(buttonPin);
    if (reading != lastButtonState) {
        // réinitialisation de l'anti-rebond
        lastDebounceTime = millis();
    }
    if ((millis() - lastDebounceTime) > debounceDelay) {
        // Condition du changement d'état du bouton
        if (reading != buttonState) {
            buttonState = reading;
            // Instruction exécutée si le bouton est LOW
            if (buttonState == 0) {
                value -= variation;
            }
        }
    }
    lastButtonState = reading;
    if (value <= limite) value = limite; //Limitation de la valeur min
    less = value;
    return less;
}
```

Figure 10 : Fonction de configuration

Un ajout a été fait sur la limite de la valeur à configurer afin d'éviter de configurer des valeurs trop importantes pour le système. Nous avons aussi intégré des paramètres à la fonction qui permet de configurer la fonction (broche du bouton, debounce, variable à configurer, variation de la valeur lors de l'appui et limite de la valeur à configurer).

Concernant l'affichage, nous avons une fonction simple :

```
void Affichage() {
    lcd.setCursor(0, 0);
    lcd.print("Debit: ");
    lcd.print(getAirflow());
    lcd.print(" SLPM");
    lcd.setCursor(0, 1);
    lcd.print("Pression: ");
    lcd.print(getPres());
    lcd.print(" mbar");
}
```

Cette fonction permet d'afficher le débit d'air et la pression envoyés par les capteurs. Nous avons choisi d'intégrer cette fonction dans Timer (TIM3) afin de gérer la vitesse d'affichage du LCD (5 Hz). La configuration du Timer est trouvable dans *Annexe 4*.

Figure 11 : Fonction d'affichage

Toutes les fonctions de l'IHM sont programmées, nous avons donc une Interface Homme-Machine utilisable et répondant au cahier des charges du système. Vous pouvez retrouver tous les codes en Annexe 2, 3 et 4.

Annexe

Annexe 1 : Tableau des caractères

upper 4 bit lower 4 bit		0000	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)															
0001	(2)															
0010	(3)															
0011	(4)															
0100	(5)															
0101	(6)															
0110	(7)															
0111	(8)															
1000	(1)															
1001	(2)															
1010	(3)															
1011	(4)															
1100	(5)															
1101	(6)															
1110	(7)															
1111	(8)															

Annexe 2 : Affichage « Hello World ! »

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(PA10, PB4, PA9, PC7, PA7, PA6); //Attribution de l'Afficheur

int variable = 24;

void setup() {
  // put your setup code here, to run once:
  lcd.begin(20, 4); //Initialisation de l'afficheur LCD
}

void loop() {
  // put your main code here, to run repeatedly:
  lcd.setCursor(0, 0);
  lcd.print("Hello World!");
  lcd.setCursor(0, 1);
  lcd.print(variable);
}
```

Annexe 3 : Configuration d'une variable avec un debounce

```
int reading = 0;
int buttonState = 1;           // état courant du bouton
int lastButtonState = 0;       // état précédent du bouton
unsigned long lastDebounceTime = 0; // le temps précédent du changement d'état du bouton
unsigned long debounceDelay = 50; // Delai de l'anti rebond, l'augmenter si il y a encore des rebonds sur le bouton

int value = 0;
int variation = 1;

void setup(){
  pinMode(PB5, INPUT_PULLUP);
}

void loop(){
  lastDebounceTime = debounceTime;
  reading = digitalRead(PB5);
  if (reading != lastButtonState) {
    // réinitialisation de l'anti-rebond
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // Condition du changement d'état du bouton
    if (reading != buttonState) {
      buttonState = reading;
      // Instruction exécutée si le bouton est LOW
      if (buttonState == 0) {
        value -= variation;
      }
    }
  }
  lastButtonState = reading;
}
```

Annexe 4 : Programme IHM

```
#include <Arduino.h>
#include <LiquidCrystal.h>
#include "pressureSensor.h"
#include "airflowSensor.h"

#define IHM_PERIOD 200000
#define TIM_CHANNEL_IHM 1

HardwareTimer* hardwareTimer4; // Timer Pressure command
LiquidCrystal lcd(PA10, PB4, PA9, PC7, PA7, PA6); //Attribution de l'Afficheur
/////////Anti-rebond/////////
int reading, reading2, reading3 = 0;

int buttonState = 1;           // état courant du bouton
int lastButtonState = 0; // état précédent du bouton
int buttonState2 = 1;
int lastButtonState2 = 0;
int buttonState3 = 1;
int lastButtonState3 = 0;

unsigned long lastDebounceTime = 0; // le temps précédent du changement d'état du bouton
unsigned long debounceDelay = 50;   // Delai de l'anti rebond, l'augmenter si il y a encore
des rebonds sur le bouton
unsigned long lastDebounceTime2 = 0;
unsigned long debounceDelay2 = 50;
unsigned long lastDebounceTime3 = 0;
unsigned long debounceDelay3 = 50;
/////////

static double less, more, configuration = 0; //Valeurs des modes des boutons

double debounceLess(int buttonPin, int debounceTime, double value, double variation, double
limite) {
    pinMode(buttonPin, INPUT_PULLUP);
    lastDebounceTime = debounceTime;
    reading = digitalRead(buttonPin);
    if (reading != lastButtonState) {
        // réinitialisation de l'anti-rebond
        lastDebounceTime = millis();
    }
    if ((millis() - lastDebounceTime) > debounceDelay) {
        // Condition du changement d'état du bouton
        if (reading != buttonState) {
            buttonState = reading;
            // Instruction exécutée si le bouton est LOW
            if (buttonState == 0) {
                value -= variation;
            }
        }
    }
    lastButtonState = reading;
    if (value <= limite) value = limite; //Limitation de la valeur min
    less = value;
    return less;
}

double debounceMore(int buttonPin, int debounceTime, double value, double variation, double
limite) {
    pinMode(buttonPin, INPUT_PULLUP);
    lastDebounceTime2 = debounceTime;
    reading2 = digitalRead(buttonPin);
    if (reading2 != lastButtonState2) {
        // réinitialisation de l'anti-rebond
        lastDebounceTime2 = millis();
    }
    if ((millis() - lastDebounceTime2) > debounceDelay2) {
        // Condition du changement d'état du bouton
        if (reading2 != buttonState2) {
            buttonState2 = reading2;
            // Instruction exécutée si le bouton est LOW
            if (buttonState2 == 0) {
                value += variation;
            }
        }
    }
}
```

```

    }
}
lastButtonState2 = reading2;
if (value >= limite) value = limite; //Limitation de la valeur min
more = value;
return more;
}

int debounceConfig(int buttonPin, int debounceTime, int etat, int etatfutur) {
    pinMode(buttonPin, INPUT PULLUP);
    lastDebounceTime3 = debounceTime;
    reading3 = digitalRead(buttonPin);
    if (reading3 != lastButtonState3) {
        // réinitialisation de l'anti-rebond
        lastDebounceTime3 = millis();
    }
    if ((millis() - lastDebounceTime3) > debounceDelay3) {
        // Condition du changement d'état du bouton
        if (reading3 != buttonState3) {
            buttonState3 = reading3;
            // Instruction exécutée si le bouton est LOW
            if (buttonState3 == 0) {
                etat = etatfutur;
            }
        }
    }
    lastButtonState3 = reading3;
    configuration = etat;
    return configuration;
}

void IHM_Timer() { // Cette fonction va permettre de générer le du timer pour la fonction du
capteur de pression
    lcd.begin(20, 4); //Initialisation de l'afficheur LCD
    hardwareTimer4 = new HardwareTimer(TIM3); //Sélectionne le timer 2
    hardwareTimer4->setOverflow(IHM_PERIOD, MICROSEC_FORMAT); //Définit la période/fréquence du
timer
    hardwareTimer4->refresh(); //réinitialise le timer
    hardwareTimer4->setMode(TIM_CHANNEL_IHM, TIMER_OUTPUT_COMPARE); //Permet de mettre le mode
OC sur le timer (Autres modes : Input Capture/PWM/One-pulse
    hardwareTimer4->setCaptureCompare(TIM_CHANNEL_IHM, 0, PERCENT_COMPARE_FORMAT); //Définit le
début du compteur
    hardwareTimer4->attachInterrupt(TIM_CHANNEL_IHM, Affichage); //Active la fonction à chaque
interruption
    hardwareTimer4->resume(); //Lance le timer
}

void Affichage() {
    lcd.setCursor(0, 0);
    lcd.print("Debit: ");
    lcd.print(getAirflow());
    lcd.print(" SLPM");
    lcd.setCursor(0, 1);
    lcd.print("Pression: ");
    lcd.print(getPres());
    lcd.print(" mbar");
}

```

Vous pouvez retrouver tous les codes sur le lien Github :

<https://github.com/makers-for-life/makair-cpap/tree/master/Firmware-CPAP>