



MAKERS FOR LIFE



MakAir

Intégration du système

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 l'intégration des composants du système afin d'avoir une fonction générale du système qui consiste à envoyer de l'air en continu.

Table des matières

Table des figures	2
C'est quoi une intégration ?	2
Les étapes de l'intégration	3
Etape 1 : Adapter les différents programmes et l'assembler	3
Etape 2 : Modifier le programme	5
Etape finale : Tests unitaires	6
Etape bonus : Création de fichier .h et .cpp	8
Annexe	11

Table des figures

Figure 1 : Intégration des composants du Respirateur PPC	2
Figure 2 : Test unitaire du débitmètre	6
Figure 3 : Test unitaire du blower	7
Figure 4 : Test unitaire du capteur de pression	7
Figure 5 : Test unitaire de l'afficheur LCD	8
Figure 6 : création des 2 fichiers .h et .cpp	8
Figure 7 : Fichier .h	9
Figure 8 : fichier .cpp	9
Figure 9 : Dossier du composant programmé	10
Figure 10 : Programme principal	10

C'est quoi une intégration ?

L'intégration lors d'un projet consiste à réunir au sein d'un même système, des parties développées de façon séparées. Cette partie est très importante puisqu'elle va être votre structure finale de votre système. Il est important de faire attention à chaque fonction intégrée, d'ensuite d'adapter toutes les fonctions et d'enfin faire des tests unitaires pour savoir si chaque fonction du système fonctionne. En règle générale, cette partie va être la plus longue s'il n'y a pas eu de réflexion d'adaptabilité de chaque fonction. Nous allons voir les étapes de l'intégration.

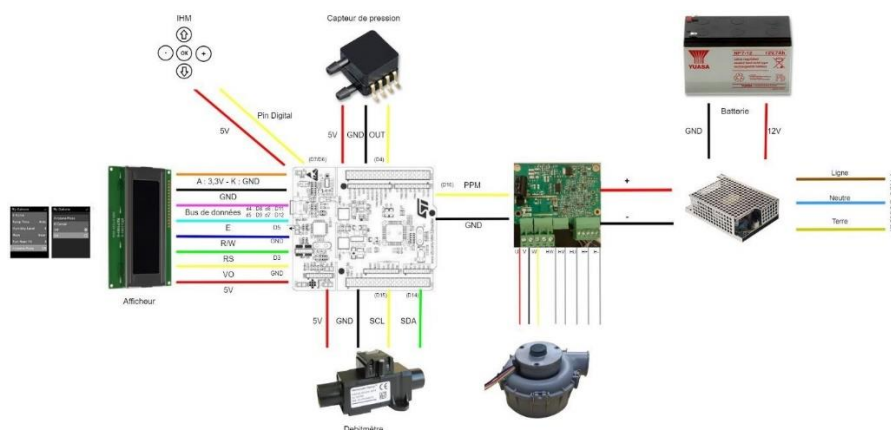


Figure 1 : Intégration des composants du Respirateur PPC

Les étapes de l'intégration

Nous allons avoir 4 étapes lors de cette intégration : Adapter les différentes fonctions, assembler les fonctions sur un programme, Ajuster le programme pour avoir un résultat final et réaliser des tests unitaires sur chaque fonction.

Etape 1 : Adapter les différents programmes et l'assembler

La première étape est de modifier les différents programmes du système pour éviter des erreurs de variables, d'association de pins ou encore des fonctions ambiguës. Nous allons montrer quelques exemples d'adaptation à réaliser :

- **HardwareTimer** : Activer des timers différents pour chaque composant du système

```
HardwareTimer* hardwareTimer1; // Timer Blower command
HardwareTimer* hardwareTimer2; // Timer Debitmetre command
HardwareTimer* hardwareTimer3; // Timer Pressure command
```

Ici on peut voir que l'on crée 3 timers afin de séparer le Blower, le débitmètre et le capteur de pression. Cela va permettre de gérer la périodicité de chaque composant et de pouvoir gérer leur mode d'activation. En réalisant cette manipulation, il faut donc prendre en compte différents timers et channels du microcontrôleur :

```
hardwareTimer1 = new HardwareTimer(TIM4); //Sélectionne le timer 4
hardwareTimer2 = new HardwareTimer(TIM1); //Sélectionne le timer 1
hardwareTimer3 = new HardwareTimer(TIM2); //Sélectionne le timer 2

#define TIM_CHANNEL_ESC_BLOWER 1 //Channel 1 de la pin D10
#define TIM_CHANNEL_AIRFLOW 4 //Channel 4 de la pin D15
#define TIM_CHANNEL_PRESSURE 2 //Channel 2 de la pin D3
```

On associe différents timers pour éviter des confusions pour le microprocesseur et qu'une erreur se produise lors de l'exécution du code. Ensuite les channels sont choisis en fonction de l'association de la pin (voir la fiche pédagogique du Blower). En séparant les différentes fonctions et les timers associées, on va éviter des erreurs d'activation des composants.

- **Communication I2C multi-slaves** : Mettre en place une communication I2C avec un maître et deux esclaves (débitmètre et capteur de pression)

```
//Communication I2C multi-slaves
TwoWire myWire1(PIN_I2C_SDA, PIN_I2C_SCL);
TwoWire myWire2(PRESSURE_SDA, PRESSURE_SCL);
```

Cette manipulation se fait avec la fonction TwoWire qui permet de créer plusieurs slaves dans une communication I2C, cela va donc remplacer l'objet commun « Wire » par « myWire1 » et « myWire2 » qui vont chacun représenter un composant : débitmètre et capteur de pression. Ensuite, on met en place le remplacement dans chaque fonction :

```
myWire1.begin(); //Initialisation de la communication I2C avec le débitmètre
myWire2.begin(); //Initialisation de la communication I2C avec le capteur de pression
```

On remarque qu'il n'y a plus besoin de la fonction Wire.setSDA et Wire.setSCL puisque les pins sont déjà associés au moment de la création de l'objet TwoWire.

Voici un exemple d'une partie du code « sensor_Init() » avec l'objet myWire1 :

```
myWire1.begin();
myWire1.beginTransmission(MFM_HONEYWELL_HAF_I2C_ADDRESS);
myWire1.write(0x02); // Force reset
uint8_t txOk = myWire1.endTransmission();
myWire1.end();
```

Un autre exemple avec une partie du code « Pressure() » avec l'objet myWire2 :

```
myWire2.beginTransmission(id);
int stat = myWire2.write (cmd, 3); // write command to the sensor
stat |= myWire2.endTransmission();
delay(10);
myWire2.requestFrom(id, 7); // read back Sensor data 7 bytes
```

Après avoir modifier toutes les fonctions concernant la communication I2C, vous pouvez communiquer avec 2 slaves.

- **Vérifier toutes les variables** : Observer s'il y a des variables avec le même nom ou encore des variables ambiguës entre chaque composant

```
////Variables Blower////
static int speedBlower = MAX_BLOWER_SPEED;

////Variables Débitmètre////
uint8_t slpm = 0;
int debit[2];
int32_t mfmInspiratoryLastValue = 0;
int32_t mfmInspiratoryAirFlow = 0;
union {
    uint16_t i;
    int16_t si;
    unsigned char c[2];
} mfmLastData;

////Variables Pressure////
uint8_t id = 0x28; // i2c address pressure sensor
uint8_t data[7]; // holds output data
uint8_t cmd[3] = {0xAA, 0x00, 0x00}; // command to be sent
double press_counts = 0; // digital pressure reading [counts]
double temp_counts = 0; // digital temperature reading [counts]
double pressure = 0; // pressure reading [bar, psi, kPa, etc.]
double temperature = 0; // temperature reading in deg C
double outputmax = 15099494; // output at maximum pressure [counts]
double outputmin = 1677722; // output at minimum pressure [counts]
double pmax = 60; // maximum value of pressure range [bar, psi, kPa, etc.]
double pmin = -60; // minimum value of pressure range [bar, psi, kPa, etc.]
double percentage = 0; // holds percentage of full scale data
char printBuffer[200], cBuff[20], percBuff[20], pBuff[20], tBuff[20];
```

Dans notre cas, une réflexion a déjà eu lieu lors de la conception de chaque fonction pour avoir des variables explicites et éviter des variables identiques entre chaque programme. Mais c'est un important de faire attention pour éviter des variables identiques contradictoires entre 2 fonctions ou encore des variables avec des valeurs variant aléatoirement lors de l'exécution du programme. Il y a des exemples de noms de variables qu'il faut faire attention comme les noms avec une seule lettre (ex : la variable « i ») ou encore les noms vagues (ex : la variable « data »).

- **Vérifier les pins utilisés** : Observer les pins activant les différents composants s'ils ne sont pas semblables.

```
#define PIN_ESC_BLOWER PB6 // D10 / TIM4_CH1
#define PIN_I2C_SDA PB9
#define PIN_I2C_SCL PB8
#define PRESSURE_SDA PB3
#define PRESSURE_SCL PB10

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

Nous avons eu la même réflexion au préalable que les variables afin d'éviter d'utiliser les mêmes pins du microcontrôleur entre chaque composant. C'est pour cela qu'il est important de faire un schéma de prototypage qui va regrouper les composants du système et leur associer des pins du microcontrôleur. Cela est aussi indispensable pour avoir une communication I2C à multi-slaves puisqu'on va attribuer des pins SDA et SCL différents pour chaque capteur : PB9 et PB8 pour le débitmètre et PB3 et PB10 pour le capteur de pression. On peut voir aussi que l'on ajoute un afficheur LCD qui comporte de nombreux pins, il faut donc faire attention à lui attribuer des pins non utilisés.

Normalement avec une réflexion en préalable sur la conception de chaque programme et l'adaptation lors de l'assemblage des codes, vous devez avoir aucune ou très peu d'erreurs lors de la compilation du code d'intégration.

Etape 2 : Modifier le programme

Cette étape va être de modifier le programme afin de mettre en place les différentes fonctions du système. Dans notre cas, cette étape est assez courte mais dans des systèmes plus complexes, celle-ci peut être assez longue afin de répondre aux besoins du cahier des charges imposés au système.

- **Mettre en place les fonctions dans le setup() ou loop()**

Dans notre cas, cela va être rapide puisque toutes nos fonctions sont intégrées dans des timers, donc il n'y aura pas de fonctions dans le loop() et il faudra mettre en place les fonctions d'initialisation des capteurs et des timers dans le setup() comme ci-dessous :

```
void setup() {
  Serial.begin(115200);
  myWire1.begin(); //Initialisation de la communication I2C avec le débitmètre
  myWire2.begin(); //Initialisation de la communication I2C avec le capteur de
pression
  lcd.begin(20, 4); //Initialisation de l'afficheur LCD
  sensor_Init();
  PWM_Blower();
  Airflow_Timer();
  Pressure_Timer();
}
```

Ensuite les fonctions seront activées par les timers, on peut aussi modifier celle-ci pour configurer leur périodicité par exemple si l'on veut que le débitmètre nous renvoie le débit d'air toutes les secondes et le capteur de pression nous renvoie la pression toutes les 0,2s :

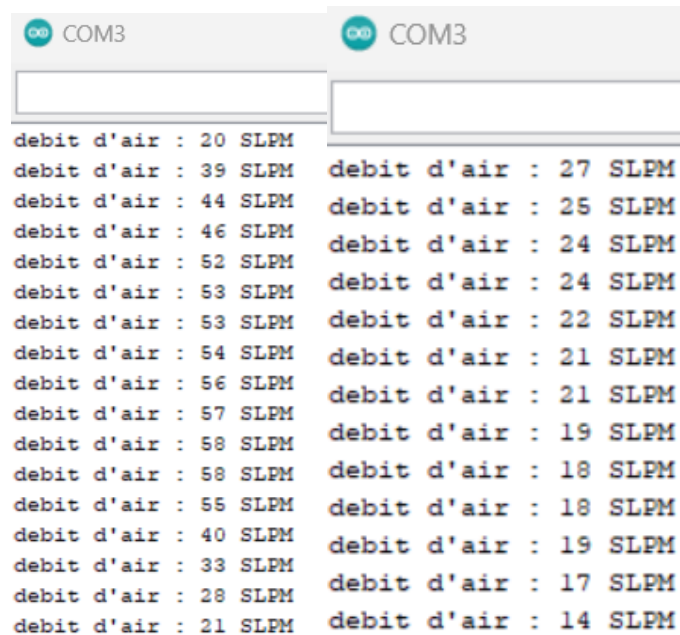
```
#define AIRFLOW_PERIOD 1000000 // 1Hz
#define PRESSURE_PERIOD 200000 // 5Hz
```

La configuration des timers se fait en général en fonction du cahier des charges du système. Maintenant que le programme du système est bien modifié et intégré, il faut réaliser des tests unitaires chaque fonction.

Etape finale : Tests unitaires

Cette étape est la plus importante puisque c'est là où vous allez pouvoir valider le système conçu. Ici, on va tester les différents composants du système pour voir si l'on répond aux fonctionnalités imposées par le cahier des charges. On va donc tester le blower, le débitmètre, le capteur de pression et l'afficheur.

- **Débitmètre** : Nous allons tester ce composant en observant une variation du débit d'air. On va donc regarder sur le terminal Arduino si l'on voit le débit d'air varié lors ce qu'on souffle de l'air à l'intérieur :



COM3	COM3
debit d'air : 20 SLPM	debit d'air : 27 SLPM
debit d'air : 39 SLPM	debit d'air : 25 SLPM
debit d'air : 44 SLPM	debit d'air : 24 SLPM
debit d'air : 46 SLPM	debit d'air : 24 SLPM
debit d'air : 52 SLPM	debit d'air : 22 SLPM
debit d'air : 53 SLPM	debit d'air : 21 SLPM
debit d'air : 53 SLPM	debit d'air : 21 SLPM
debit d'air : 54 SLPM	debit d'air : 19 SLPM
debit d'air : 56 SLPM	debit d'air : 18 SLPM
debit d'air : 57 SLPM	debit d'air : 18 SLPM
debit d'air : 58 SLPM	debit d'air : 19 SLPM
debit d'air : 58 SLPM	debit d'air : 17 SLPM
debit d'air : 55 SLPM	debit d'air : 14 SLPM
debit d'air : 40 SLPM	
debit d'air : 33 SLPM	
debit d'air : 28 SLPM	
debit d'air : 21 SLPM	

Figure 2 : Test unitaire du débitmètre

On remarque que le débit d'air varie en fonction de l'air qui rentre dans le débitmètre. Cela nous permet de valider ce composant.

- **Blower** : Nous allons tester ce composant en modifiant sa vitesse et s'il envoie une quantité d'air différente à chaque modification. Pour cela, on va regarder la variation du débit d'air dans le débitmètre en fonction de la vitesse du blower :

COM3	COM3	COM3
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 36 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 36 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 36 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 36 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 36 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 10 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 36 SLPM
debit d'air : 9 SLPM	debit d'air : 19 SLPM	debit d'air : 35 SLPM

Figure 3 : Test unitaire du blower

On remarque que le débit d'air varie en fonction de la vitesse du blower. Cela nous permet de valider ce composant.

- **Capteur de pression** : Nous allons tester ce composant en observant une variation de la pression. On va donc regarder sur le terminal Arduino si l'on voit la pression variée.

COM3	
debit d'air : 9 SLPM	debit d'air : 10 SLPM
0.028 KPa; 50.019 %	debit d'air : 9 SLPM
debit d'air : 9 SLPM	-0.042 KPa; 49.972 %
debit d'air : 9 SLPM	debit d'air : 9 SLPM
0.020 KPa; 50.013 %	debit d'air : 9 SLPM
debit d'air : 9 SLPM	-0.043 KPa; 49.972 %
debit d'air : 9 SLPM	debit d'air : 9 SLPM
0.021 KPa; 50.014 %	debit d'air : 9 SLPM
debit d'air : 9 SLPM	-0.054 KPa; 49.964 %
debit d'air : 10 SLPM	debit d'air : 9 SLPM
-0.011 KPa; 49.993 %	debit d'air : 9 SLPM
debit d'air : 9 SLPM	-0.059 KPa; 49.960 %
debit d'air : 9 SLPM	debit d'air : 9 SLPM
-0.013 KPa; 49.991 %	-0.072 KPa; 49.952 %
debit d'air : 9 SLPM	debit d'air : 10 SLPM
debit d'air : 9 SLPM	debit d'air : 9 SLPM
-0.033 KPa; 49.978 %	-0.090 KPa; 49.940 %

Figure 4 : Test unitaire du capteur de pression

On observe donc que la pression peut varier lorsque le système est en marche. Cela nous permet de valider ce composant.

- **Afficheur LCD** : Nous allons tester ce composant en affichant le débit d'air et la pression en temps réels.

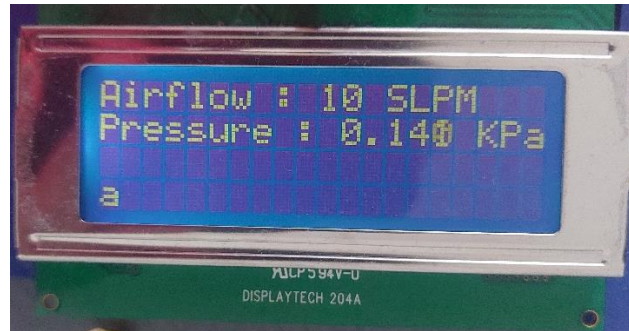


Figure 5 : Test unitaire de l'afficheur LCD


On observe que l'on peut afficher le débit d'air et la pression en temps réels sur l'afficheur LCD. Cela nous permet de valider ce composant.

Tous les tests unitaires sont valides donc on peut valider notre intégration du système puisque chaque fonction fonctionne correctement lors de l'exécution du programme.

Etape bonus : Création de fichier .h et .cpp

Cette étape est optionnelle dans l'intégration du système mais elle est importante à réaliser sur des systèmes plus complexes pour rendre le programme principal plus modulable et plus portable. En règle générale, cette étape est réalisée dans un milieu industriel et professionnel.

Cette étape consiste à prendre chaque composant et de réaliser des fichiers séparés pour chacun d'eux qui vont définir les fonctions et les variables liées. Par exemple, on peut prendre le débitmètre et créer un fichier « airflowSensor.h » et un fichier « airflowSensor.cpp » :

- **Créer un fichier sur Arduino IDE** : Il vous suffit de prendre votre fichier débitmètre et d'appuyer sur ce bouton : 

Pour chaque fonction, il faut créer 2 fichiers (.h et .cpp) comme ci-dessous :

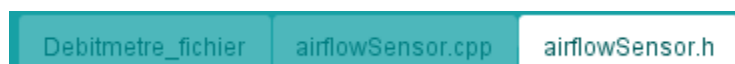


Figure 6 : création des 2 fichiers .h et .cpp

- **Fichier airflowSensor.h** : ce fichier signifiant **header**, il va regrouper les définitions de chaque fonction ainsi que les définitions de structures ou de classes. On va donc définir le nom de la fonction, ce qui rentre à l'intérieur (les paramètres) et ce qui en sort (la variable de retour). Ainsi, votre programme principal aura une idée de comment fonctionne votre fonction comme s'il s'adressait à une boîte noire. On peut aussi définir des variables liées au composant. Voici l'exemple sur le header du débitmètre :

Debitmetre_fichier	airflowSensor.cpp	airflowSensor.h
--------------------	-------------------	-----------------

```

#include <Arduino.h>

#define MFM_HONEYWELL_HAF_I2C_ADDRESS 0x49
#define PIN_I2C_SDA PB9
#define PIN_I2C_SCL PB8

void Airflow_init();

void Airflow_read(void);

uint32_t getSerialNumber(void);
int32_t getAirflow(void);
int32_t getAirflowRaw(void);

void afficherAirflow();
void afficherAirflowRaw();
void afficherSerialNumber();

```

Figure 7 : Fichier .h

- **Fichier airflowSensor.cpp** : Ce fichier est une extension .cpp (pour C Plus Plus → C++). Il regroupe l'implémentation des fonctions c'est-à-dire que c'est dans ce fichier que vous allez écrire le contenu de chaque fonction. La première manipulation à faire dans ce fichier est d'inclure : `#include "airflowSensor.h"` et ensuite vous écrivez le contenu des fonctions comme l'exemple de la fonction « Airflow_init() » ci-dessous :

Debitmetre_fichier	airflowSensor.cpp	airflowSensor.h
--------------------	-------------------	-----------------

```

#include <Arduino.h>
#include <Wire.h>
#include "airflowSensor.h"

////Variables Débitmètre////
uint8_t slpm = 0;
int debit[2];
int32_t mfmInspiratoryLastValue = 0;
int32_t mfmInspiratoryAirFlow = 0;
union {
    uint16_t i;
    int16_t si;
    unsigned char c[2];
} mfmLastData;
uint32_t serialNumber;

void Airflow_init(){
    Wire.setSDA(PIN_I2C_SDA);
    Wire.setSCL(PIN_I2C_SCL);
    Wire.begin();
    Wire.beginTransmission(MFM_HONEYWELL_HAF_I2C_ADDRESS);
    Wire.write(0x02); // Force reset
    uint8_t txOk = Wire.endTransmission();
    Wire.end();
    delay(30);

    uint32_t sn = 0;
    Wire.begin();
    Wire.beginTransmission(MFM_HONEYWELL_HAF_I2C_ADDRESS);
    uint8_t rxcount = Wire.requestFrom(MFM_HONEYWELL_HAF_I2C_ADDRESS, 2);
    sn = Wire.read();
    sn <<= 8;
    sn |= Wire.read(); // first transmission is serial number register 0
    sn <<= 8;
    delay(2); // if you do not wait, sensor will send again register 0
    rxcount += Wire.requestFrom(MFM_HONEYWELL_HAF_I2C_ADDRESS, 2);
    sn |= Wire.read();
    sn <<= 8;
    sn |= Wire.read(); // second transmission is serial number register 1
    serialNumber = sn;
    Wire.end();
}

```

Figure 8 : fichier .cpp

- **Lier et Tester les fichiers au programme principal** : Maintenant que vos définitions et vos fonctions sont implémentées, il ne reste plus qu'à les intégrer au programme de votre composant afin de les tester. Tout d'abord, vérifier si votre dossier est constitué de tous les fichiers :

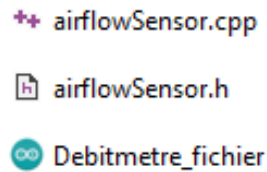


Figure 9 : Dossier du composant programmé

Ensuite vous intégrez `#include "airflowSensor.h"` dans le programme principal et vous n'avez plus qu'à faire appel aux fonctions dans le programme dans `setup()`, `loop()` ou encore dans un timer comme ci-dessous :

```
#include <Arduino.h>
#include <Wire.h>
#include "HardwareTimer.h"
#include "airflowSensor.h"

#define AIRFLOW_PERIOD 200000 // 5Hz
#define TIM_CHANNEL_AIRFLOW 4 //Channel 4 de la pin D15

HardwareTimer* hardwareTimer2; // Timer Debitmetre command

void setup() {
    Serial.begin(115200);
    Airflow_init();
    Airflow_Timer();
}

void loop() {

}

void airFlow(){
    Airflow_read();
    afficherAirflow();
}

void Airflow_Timer() {
    hardwareTimer2 = new HardwareTimer(TIM1); //Sélectionne le timer 4 pour la pin D10
    hardwareTimer2->setOverflow(AIRFLOW_PERIOD, MICROSEC_FORMAT); //Définit la période/fréquence
    de la PWM
    hardwareTimer2->refresh(); //réinitialise le timer
    hardwareTimer2->setMode(TIM_CHANNEL_AIRFLOW, TIMER_OUTPUT_COMPARE); //Permet de mettre le
    mode PWM sur le timer (Autres modes : Input Capture/Output Capture/One-pulse
    // Set PPM width to 1ms
    hardwareTimer2->setCaptureCompare(TIM_CHANNEL_AIRFLOW, 0, PERCENT_COMPARE_FORMAT);
    //Définit le rapport cyclique de la PWM
    hardwareTimer2->attachInterrupt(TIM_CHANNEL_AIRFLOW, airFlow); //Active la fonction à chaque
    interruption
    hardwareTimer2->resume(); //Lance le timer
}
```

Figure 10 : Programme principal

Lors de la compilation, Il peut arriver que le lien avec les symboles/librairies Arduino ne se fasse pas correctement. Dans ce cas, rajoutez `#include <Arduino.h>` au début de votre fichier .h ou .cpp.

Maintenant, quand vous allez compiler, le compilateur va aller chercher le fichier pointé par le « **include** », le compiler puis le lier dans votre programme principal. Pour rendre votre système totalement modulable, il vous reste à le faire pour les autres composants : Blower et Capteur de pression.

Annexe

Programme d'intégration du système

```
#define ESC_PPM_PERIOD 20000 // 50Hz
#define AIRFLOW_PERIOD 100000 // 10Hz
#define PRESSURE_PERIOD 200000 // 5Hz
#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 TIM_CHANNEL_AIRFLOW 4 //Channel 4 de la pin D15
#define TIM_CHANNEL_PRESSURE 2 //Channel 2 de la pin D3
#define BlowerSpeed2MicroSeconds(value) map(value, 0, 1800, 1000, 1950)
//Intervalle [1ms à 1,95 ms] sur la variation de vitesse du blower
#define MFM_HONEYWELL_HAF_I2C_ADDRESS 0x49 //Adresse du debitmetre
#define PIN_I2C_SDA PB9
#define PIN_I2C_SCL PB8
#define PRESSURE_SDA PB3
#define PRESSURE_SCL PB10

#include <Wire.h>
#include "HardwareTimer.h"
#include <LiquidCrystal.h>

HardwareTimer* hardwareTimer1; // Timer Blower command
HardwareTimer* hardwareTimer2; // Timer Debitmetre command
HardwareTimer* hardwareTimer3; // Timer Pressure command

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

//Communication I2C multi-slaves
TwoWire myWire1(PIN_I2C_SDA, PIN_I2C_SCL);
TwoWire myWire2(PRESSURE_SDA, PRESSURE_SCL);

////Variables Blower////
static int speedBlower = MAX_BLOWER_SPEED;

////Variables Débitmètre////
uint8_t slpm = 0;
int debit[2];
int32_t mfmInspiratoryLastValue = 0;
int32_t mfmInspiratoryAirFlow = 0;
union {
    uint16_t i;
    int16_t si;
    unsigned char c[2];
} mfmLastData;

////Variables Pressure////
uint8_t id = 0x28; // i2c address pressure sensor
uint8_t data[7]; // holds output data
uint8_t cmd[3] = {0xAA, 0x00, 0x00}; // command to be sent
double press_counts = 0; // digital pressure reading [counts]
double temp_counts = 0; // digital temperature reading [counts]
double pressure = 0; // pressure reading [bar, psi, kPa, etc.]
double temperature = 0; // temperature reading in deg C
double outputmax = 15099494; // output at maximum pressure [counts]
double outputmin = 1677722; // output at minimum pressure [counts]
double pmax = 60; // maximum value of pressure range [bar, psi, kPa, etc.]
double pmin = -60; // minimum value of pressure range [bar, psi, kPa, etc.]
```

```

double percentage = 0; // holds percentage of full scale data
char printBuffer[200], cBuff[20], percBuff[20], pBuff[20], tBuff[20];

void setup() {
    Serial.begin(115200);
    myWire1.begin(); //Initialisation de la communication I2C avec le débitmètre
    myWire2.begin(); //Initialisation de la communication I2C avec le capteur de
    pression
    lcd.begin(20, 4); //Initialisation de l'afficheur LCD
    sensor_Init();
    PWM_Blower();
    Airflow_Timer();
    Pressure_Timer();
}

void loop() {

}

void airFlow() { //lire le débit
    myWire1.begin();
    uint8_t readCount = myWire1.requestFrom(MFM_HONEYWELL_HAF_I2C_ADDRESS, 2);
    mfmLastData.c[0] = myWire1.read(); //Lecture des données
    mfmLastData.c[1] = myWire1.read();
    // Wire.endTransmission() send a new write order followed by a stop. Useless and
    the
    // sensor often nack it.
    myWire1.end();

    mfmInspiratoryLastValue = (uint32_t) (mfmLastData.c[1] & 0xFFu); // masquage
    0b00000000000000000000000011111111 & données du registre 1 (falcutatif)
    mfmInspiratoryLastValue |= (((uint32_t)mfmLastData.c[0]) << 8) & 0x0000FF00u; //
    masquage 0b00000000000000000111111110000000 & données du registre 0 (falcutatif)

    // Theoretical formula: Flow(slp) = 200*((rawvalue/16384)-0.1)/0.8
    // float implementation, 1 liter per minute unit
    mfmInspiratoryAirFlow = 200 * (((uint32_t)mfmInspiratoryLastValue / 16384.0) -
    0.1) / 0.8;
    // (Output value in SLPM)

    // fixed float implementation, 1 milliliter per minute unit
    //mfmInspiratoryAirFlow = (((10 * mfmInspiratoryLastValue) - 16384) * 1526);
    Serial.print("debit d'air : "); //affichage du débit sur le terminal
    Serial.print(mfmInspiratoryAirFlow);
    Serial.println(" SLPM");
    lcd.setCursor(0, 0); //affichage du débit sur le LCD
    lcd.print("Airflow : ");
    lcd.print(mfmInspiratoryAirFlow);
    lcd.print(" SLPM");
}

void sensor_Init() { //lire le numéro de série

    myWire1.begin();
    myWire1.beginTransmission(MFM_HONEYWELL_HAF_I2C_ADDRESS);
    myWire1.write(0x02); // Force reset
    uint8_t txOk = myWire1.endTransmission();
    myWire1.end();
    delay(30);

    uint32_t sn = 0;
    myWire1.begin();
    myWire1.beginTransmission(MFM_HONEYWELL_HAF_I2C_ADDRESS);
    uint8_t rxcount = myWire1.requestFrom(MFM_HONEYWELL_HAF_I2C_ADDRESS, 2); // read
    back Sensor data 2 bytes
    sn = myWire1.read();
    sn <<= 8;
    sn |= myWire1.read(); // first transmission is serial number register 0

```

```

    sn <=&= 8;
    delay(2); // if you do not wait, sensor will send again register 0
    rxcount += myWire1.requestFrom(MFM_HONEYWELL_HAF_I2C_ADDRESS, 2);
    sn |= myWire1.read();
    sn <=&= 8;
    sn |= myWire1.read(); // second transmission is serial number register 1
    Serial.print("Numéro de série : ");
    Serial.println(sn);
    myWire1.end();
}

void Pressure() { //Lire la pression
    myWire2.beginTransmission(id);
    int stat = myWire2.write (cmd, 3); // write command to the sensor
    stat |= myWire2.endTransmission();
    delay(10);
    myWire2.requestFrom(id, 7); // read back Sensor data 7 bytes
    int i = 0;
    for (i = 0; i < 7; i++) {
        data [i] = myWire2.read();
    }
    press_counts = data[3] + data[2] * 256 + data[1] * 65536; // calculate digital
    pressure counts
    temp_counts = data[6] + data[5] * 256 + data[4] * 65536; // calculate digital
    temperature counts
    temperature = (temp_counts * 200 / 16777215) - 50; // calculate temperature in
    deg c
    percentage = (press_counts / 16777215) * 100; // calculate pressure as percentage
    of full scale
    //calculation of pressure value according to equation 2 of datasheet
    pressure = ((press_counts - outputmin) * (pmax - pmin)) / (outputmax - outputmin)
+ pmin;
    dtostrf(press_counts, 4, 1, cBuff); // dtostrf() may be the function you need if
    you have a floating point value that you need to convert to a string.
    dtostrf(percentage, 4, 3, percBuff);
    dtostrf(pressure, 4, 3, pBuff);
    dtostrf(temperature, 4, 3, tBuff);
    /*
        The below code prints the raw data as well as the processed data
        Data format : Status Register, 24-bit Sensor Data, Digital Counts, percentage
    of full scale
        pressure,
        pressure output, temperature
    */
    sprintf(printBuffer, " % x\t % 2x % 2x % 2x\t % s\t % s\t % s\t % s \n", data[0],
    data[1], data[2], data[3], cBuff, percBuff, pBuff, tBuff);
    Serial.print(pBuff); //affichage de la pression et du taux de la plage de mesure
    sur le terminal
    Serial.print(" KPa; ");
    Serial.print(percBuff);
    Serial.println(" % ");
    lcd.setCursor(0, 1); //affichage de la pression sur le LCD
    lcd.print("Pressure : ");
    lcd.print(pBuff);
    lcd.print(" KPa");
    delay(10);
}

void PWM_Blower() { //Génération du signal PPM
    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(); //réinitialise 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/Output 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->resume(); //Lance le timer
}

void Airflow_Timer() { //Génération du timer pour la fonction du débitmètre
    hardwareTimer2 = new HardwareTimer(TIM1); //Sélectionne le timer 1
    hardwareTimer2->setOverflow(AIRFLOW_PERIOD, MICROSEC_FORMAT); //Définit la
période/fréquence du timer
    hardwareTimer2->refresh(); //réinitialise le timer
    hardwareTimer2->setMode(TIM_CHANNEL_AIRFLOW, TIMER_OUTPUT_COMPARE); //Permet de
mettre le mode OC sur le timer (Autres modes : Input Capture/PWM/One-pulse
    // Set PPM width to 1ms
    hardwareTimer2->setCaptureCompare(TIM_CHANNEL_AIRFLOW, 0 ,
PERCENT_COMPARE_FORMAT); //Définit le début du compteur
    hardwareTimer2->attachInterrupt(TIM_CHANNEL_AIRFLOW, airFlow); //Active la
fonction à chaque interruption
    hardwareTimer2->resume(); //Lance le timer
}

void Pressure_Timer() { //Génération du timer pour la fonction du capteur de
pression
    hardwareTimer3 = new HardwareTimer(TIM2); //Sélectionne le timer 2
    hardwareTimer3->setOverflow(PRESSURE_PERIOD, MICROSEC_FORMAT); //Définit la
période/fréquence du timer
    hardwareTimer3->refresh(); //réinitialise le timer
    hardwareTimer3->setMode(TIM_CHANNEL_PRESSURE, TIMER_OUTPUT_COMPARE); //Permet de
mettre le mode OC sur le timer (Autres modes : Input Capture/PWM/One-pulse
    // Set PPM width to 1ms
    hardwareTimer3->setCaptureCompare(TIM_CHANNEL_PRESSURE, 0 ,
PERCENT_COMPARE_FORMAT); //Définit le début du compteur
    hardwareTimer3->attachInterrupt(TIM_CHANNEL_PRESSURE, Pressure); //Active la
fonction à chaque interruption
    hardwareTimer3->resume(); //Lance le timer
}

```