

Projet Tutoré 2 : Station de mesures environnementales

Guide de mise en oeuvre

Pierre Gaucher (pierre.gaucher@univ-tours.fr)

Ce document est en cours de rédaction et peut contenir des erreurs. Merci de bien vouloir les signaler afin d'en améliorer le contenu.

Reproduction et diffusion interdites sans autorisation préalable.

1. Table des matières

1.	Introduction	5
1.1.	Volume horaire	5
1.2.	Modalités d'évaluation	5
1.3.	Prérequis	5
1.4.	Organisation du projet	5
2.	Fonctionnalités attendues de l'application développée	6
2.1.	Besoins fonctionnels	6
2.2.	Architecture logicielle de l'application développée	8
3.	Moyens matériels	9
3.1.	Carte microcontrôleur Arduino Mega2560	9
3.1.1.	Entrées – sorties numériques (ou digitales)	12
3.1.2.	Entrées – sorties analogiques	12
3.1.3.	Bus de communication I2C	12
3.1.4.	Bus de communication série	12
3.2.	Shield Grove Mega2560	13
3.3.	Module horloge temps réel RTC	14
3.4.	Capteur BME680	16
3.5.	Module GPS	17
3.6.	Ecran TFT couleur avec dalle tactile	17
4.	Ressources logicielles	18
4.1.	Installation de l'IDE Arduino	18
4.2.	Description des fonctions de l'IDE	20
4.3.	Structure de base d'un programme Arduino	22
4.3.1.	Programme Blink	22
4.3.2.	Exécution du programme sur la carte Arduino	23
5.	Gestion de la liaison série	26
6.	Gestion des interruptions - Timers	27
6.1.	Définition – Principe	27
6.2.	Gestion des interruptions	28
6.2.1.	Gestion des interruptions : principes	28
6.2.2.	Ressources associées au Timer1	29
6.2.3.	Ressources du Timer1 associées à la gestion des interruptions	31
6.2.4.	Gestion de l'interruption Timer1 Overflow : principe de mise en œuvre	32
6.2.5.	Travail à réaliser	34

7.	Mise en œuvre du circuit d'horloge temps réel RTC	34
7.1.	Format et codage des données.....	34
7.2.	Choix types de données et fonctions de manipulation de l'horloge RTC	34
7.2.1.	Types de données	34
7.2.2.	Fonctions de manipulation du circuit d'horloge RTC	35
7.2.3.	Gestion de la communication avec le circuit RTC DS1307	36
7.2.4.	Fonction complémentaire de gestion de calendrier.....	36
8.	Mise en œuvre du module GPS.....	37
8.1.	Données fournies par le module GPS : messages NMEA.....	37
8.1.1.	Format du message \$GPRMC.....	37
8.1.2.	Format du message \$GPGGA.....	38
8.2.	Mode de fonctionnement du module GPS	39
8.3.	Mise en œuvre	39
8.3.1.	Acquisition d'un message NMEA	40
8.3.2.	Parser un message NMEA	40
8.3.3.	Savoir si le module GPS renvoie des trames valides	41
8.3.4.	Choisir le type de message NMEA renvoyé par le module GPS.....	41
8.3.5.	Validation des fonctions associées à l'utilisation du module GPS	41
9.	Mise en œuvre du capteur BME680	41
9.1.	Configuration de l'environnement de programmation.....	42
9.1.1.	Téléchargement de l'archive spécifique pour environnement Arduino.....	42
9.1.2.	Installation des librairies BSEC	43
9.1.3.	Remplacer Arduino-builder	43
9.1.4.	Modifier les options de compilation	43
9.2.	Première mise en œuvre et test du BME680	44
9.2.1.	Fonctions additionnelles	44
9.2.2.	Initialisation des ressources et fonction <i>setup()</i>	44
9.2.3.	Acquisition de mesures depuis le capteur BME680	46
10.	Gestion de la mise à jour de la date et de l'heure	47
10.1.1.	Heure UTC et heure locale	47
10.1.2.	Mise en œuvre : horloge RTC + module GPS + Sortie terminal série.....	49
11.	Utilisation de l'écran TFT.....	49
11.1.	Ecran TFT couleur.....	49
11.1.1.	Système de coordonnées écran et manipulation des couleurs	49
11.1.2.	Configuration matérielle – Interfaçage avec la carte Arduino Mega2560	51
11.1.3.	Mise en œuvre logicielle et test.....	51
11.2.	Affichage de la date et de l'heure	52
11.3.	Affichage des mesures du capteurs BME680.....	54

11.4.	Graphe d'évolution de la pression moyenne	55
11.5.	Finalisation de l'affichage de l'écran principal	55
12.	Dalle tactile	56
12.1.	Système de référence dalle tactile – lien avec référentiel écran TFT	56
12.2.	Configuration matérielle	57
12.3.	Calibration et mise en œuvre de la dalle tactile	58
12.4.	Test écran TFT et dalle tactile	58

1. Introduction

L'objectif du projet tutoré est de développer une application unique afin d'appréhender la démarche de conception d'une application logicielle.

Le travail à réaliser, bien que guidé, nécessitera une part de travail personnel sur :

- La structuration du code ;
- Le choix des structures de données manipulées ;
- L'analyse fonctionnelle et algorithmique des fonctionnalités à implémenter ;
- La conception et la mise en œuvre de tests unitaires.

Le langage support pour la programmation est le langage C, avec pour cible matérielle la plateforme microcontrôleur Arduino Atmega 2560.

Ce projet nécessitera la mise en œuvre matérielle de périphériques. Toutefois, les aspects matériels liés à cette mise en œuvre seront facilitées, afin de se concentrer sur la partie logicielle.

1.1. Volume horaire

Le volume associé est défini ci-dessous :

- 25h de TP encadrés en salle machine.
- 7h de TP en autonomie complète, pendant lesquelles les étudiants travailleront donc seuls.

1.2. Modalités d'évaluation

Le projet est réalisé en binôme. La constitution de chaque binôme est laissée à l'initiative des étudiants.

Les binômes seront constitués lors de la première séance.

L'évaluation pourra prendre en compte la présence lors des séances encadrées.

Les jalons sur lesquels portera l'évaluation seront définis et préciser tout au long de ce document.

1.3. Prérequis

La réalisation de ce projet repose sur les enseignements d'Algorithmique et de Langage C et sur les acquis du projet tutoré 1.

Une condition nécessaire de réussite repose sur la préparation préalable des séances et l'étude des éléments fournis au sein de ce support de TP. D'autres ressources documentaires seront également disponibles en ligne sous Celene.

1.4. Organisation du projet

La toute première partie du projet consistera à prendre en main l'environnement logiciel de développement au travers d'une application simple. En même temps, nous verrons comment tirer parti du mécanisme d'interruption, et qu'elle en est la mise en œuvre logicielle sur la plateforme matérielle cible.

La seconde partie du projet repose sur l'utilisation de ressources matérielles spécifiques qu'il sera nécessaire de maîtriser. Cette maîtrise permettra de définir des blocs fonctionnels de code, qui seront validés séparément.

Enfin la dernière partie devrait être consacrée à « l'assemblage final » des différentes briques logicielles afin de constituer l'application finale.

2. Fonctionnalités attendues de l'application développée

2.1. Besoins fonctionnels

Il s'agit de créer la couche logicielle d'un module matériel qui permettra d'afficher sur un écran couleur, dotée d'une dalle tactile, les informations suivantes :

Affichage des informations de date, heure, heure Eté-Hiver, Fuseau horaire :

- La date en prenant en compte les champs <jour> <quantième> <mois> <Année> (exemple : vendredi 18 septembre 2020). Les données affichées seront rafraîchies périodiquement lorsqu'une modification survient ;
- L'heure en prenant en compte les champs <Heure> : <Minutes> : <Secondes> au format 24h (exemple 18 : 24 : 36 pour 18 heures, 24 minutes et 36 secondes). L'affichage de l'heure sera rafraîchi toutes les secondes ;
- Les informations de date et heure doivent pouvoir être paramétrées à la volée par l'utilisateur, selon que l'on est en heure d'été ou d'hiver. Le mode par défaut est le mode « Heure d'hiver ». En cas de modification par l'utilisateur, la mise à jour de la date et de l'heure heure est automatique, en accord avec le fuseau horaire courant. L'information de mode heure été ou hiver sera affiché sur l'écran ;
- Le fuseau horaire de référence pour la gestion de la date et de l'heure est celui de Paris par défaut. Si besoin, l'utilisateur du module aura la possibilité de changer de fuseau horaire de référence. Dans ce cas, la mise à jour de la date et de l'heure doit être automatique et transparente pour l'utilisateur. La ville de référence du fuseau horaire sera affichée, et son affichage rafraîchi si besoin.

Affichage des informations environnementales : elles sont regroupées en deux familles :

- Famille 1 : paramètres usuels associés à la température, pression atmosphérique, taux d'humidité ;
 - o Température : exprimée en degrés Celsius et affichée sous la forme 21.7 °C ;
 - o Pression : exprimé en hPa (hecto Pascal) et affichée sous la forme 1013.34 hPa ;
 - o Taux d'humidité : taux d'humidité relative exprimé en % et affiché sous la forme 56.3% ;
 - o ΔP Variation de la valeur moyenne de la pression atmosphérique par rapport au créneau horaire précédent. A titre d'illustration, la valeur moyenne de la pression atmosphérique sur le créneau horaire 14h-15h est de 1015.24 hPa. Sur le créneau suivant 15h -16h, la valeur moyenne de la pression atmosphérique est de 1014.32 hPa. Alors $\Delta P = -0.92$ hPa. Cette grandeur est donc rafraîchie toutes les heures ;
 - o Affichage du graphe d'évolution de la valeur moyenne de la pression atmosphérique, par intervalle d'1 heure.
- Famille 2 : paramètres associés à une mesure de la qualité de l'air tels que :
 - o IAQ : Index de Qualité de l'Air ;
 - o IAQAcc : Index de confiance de la mesure de l'IAQ.

L'affichage des mesures environnementales sera rafraîchi toutes les 3s, sauf exception (cf. justification plus loin).

En plus de ces informations environnementales, des données spatiales seront accessibles, en utilisant un module GPS. Au travers de la gestion de ce module GPS, les fonctionnalités offertes sont les suivantes :

- Récupérer la date et l'heure UTC afin de procéder périodiquement au recalage des données du circuit RTC d'horloge temps réel et en cas de mise à jour de l'heure ;
- Récupérer, à la demande de l'utilisateur les données spatio-temporel du lieu où se situe le module, et les afficher, si elles sont disponibles ;
- Indiquer à l'utilisateur, par un affichage adéquat, le fait que le module GPS soit synchronisé.

Les figures ci-dessous résument les différents écrans qui seront proposés à l'utilisateur, sans directives sur leur mise en forme. L'agencement des informations de chaque écran fait partie du travail à réaliser. L'écran principal – Ecran 1 - (cf. Figure 1) offre à l'utilisateur la vue sur l'ensemble des informations à afficher par défaut¹.

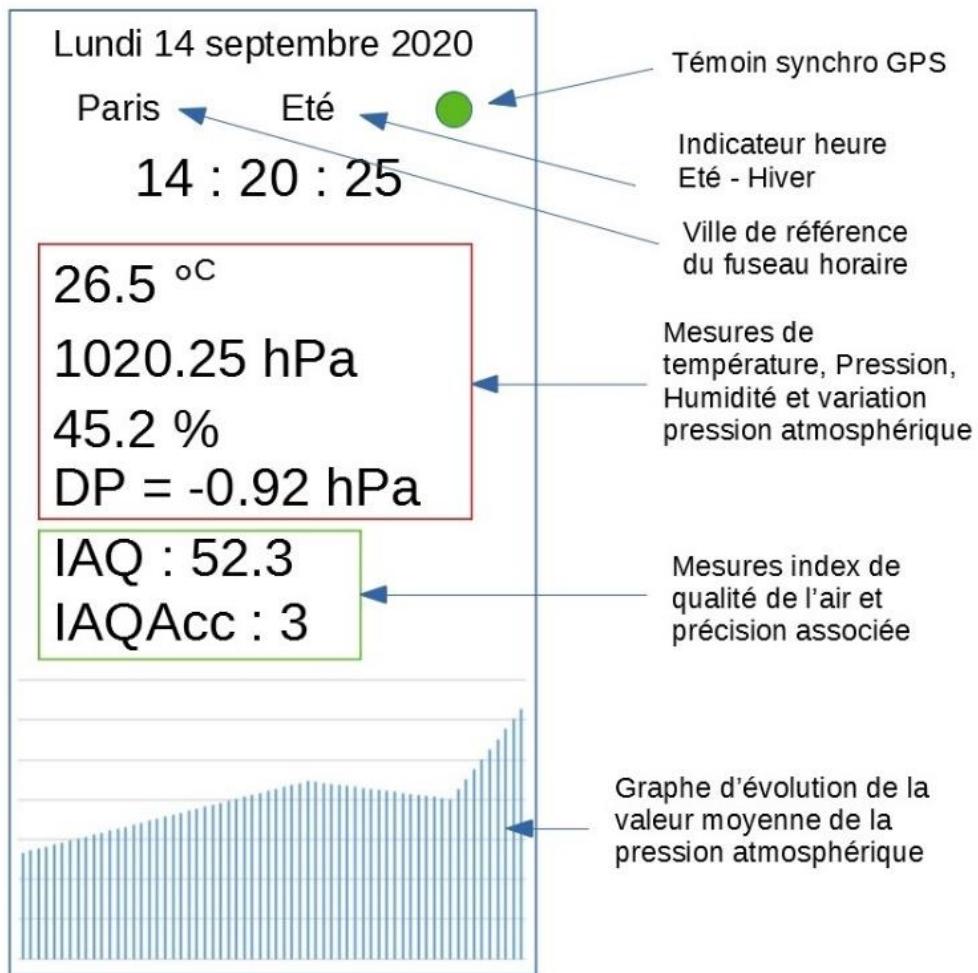


Figure 1 : Ecran principal d'affichage de l'ensemble des grandeurs mesurées

D'autres écrans seront à définir pour :

- Effectuer le choix heure été – heure hiver (Ecran 2) ;
- Effectuer la modification du fuseau horaire de référence (Ecran 3) ;
- Afficher les données spatiales et horaires issues du module GPS (Ecran 4) ;
- Afficher l'ensemble des informations de mesures environnementales (Ecran 5).

L'accès à ces vues s'effectuera depuis l'écran principal, par un menu intermédiaire de choix. Les transitions entre les différents écrans seront gérées au travers d'un écran Menu, via la gestion de la dalle tactile. Le diagramme des transitions entre les différents écrans (ou vues) est défini ci-après (cf. Figure 2)².

¹ La figure 1 n'est donnée qu'à titre d'illustration.

² Time out : dépassement de temps ou temps de repos.

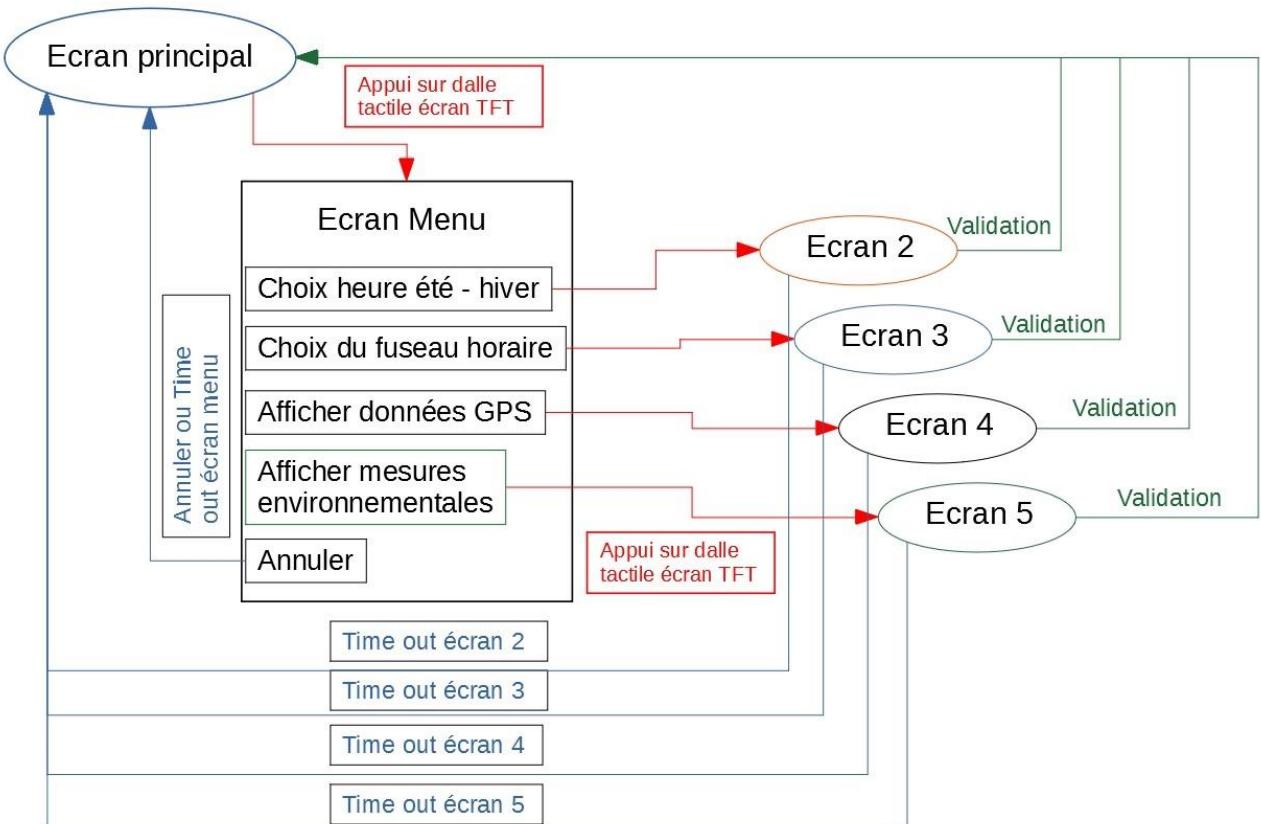


Figure 2 : Graphes de transition entre écran

Le module ne sera pas autonome du point de vue énergétique. Il sera alimenté soit à l'aide d'un port USB, ou bien avec un adaptateur secteur.

Eventuellement, il sera possible de gérer l'intensité du rétroéclairage de l'écran TFT, mais cela reste une option non essentielle.

2.2. Architecture logicielle de l'application développée

Au cours de ce projet, plusieurs fonctions seront développées, associées à des éléments matériels spécifiques ou bien à des fonctionnalités particulières. Afin de structurer le code source de l'application, vous devrez utiliser des modules logiciels indépendants qui devront respecter les règles suivantes :

Votre projet sera défini au sein du répertoire `Station_Mesures`. Il contiendra les fichiers suivants :

- Le fichier principal, qui contiendra implicitement la fonction `main`, sera nommé `Station_Mesures.ino`.
- L'ensemble des fichiers ci-dessous :
 - o `Affichage.cpp` et `Affichage.h`
 - o `BME680_Sensor.cpp` et `BME680_Sensor.h`
 - o `Calendrier.cpp` et `Calendrier.h`
 - o `GPS.cpp` et `GPS.h`
 - o `RTC_DS1307.cpp` et `RTC_DS1307.h`
 - o `TFT_Affichage.cpp` et `TFT_Affichage.h`
 - o `Tactile.cpp` et `Tactile.h`

Le projet est disponible sous la forme de l'archive *Station_Mesures.zip*. Il constitue la trame de base de votre application et peut être modifié si cela est nécessaire.



L'ensemble des fichiers sources doivent posséder l'extension *.cpp*. Bien que la programmation s'appuie sur le langage C, l'utilisation de bibliothèques « Arduino » (écrites dans le formalisme orienté objet) nous imposent d'utiliser cette extension et non pas *.c* uniquement.

L'ouverture du fichier *Station_Mesures.ino* dans l'IDE Arduino provoque l'ouverture de l'ensemble du projet

3. Moyens matériels

Le support matériel utilisé tout au long de ce projet sera composé d'une carte microcontrôleur Arduino, utilisée avec des shields additionnels, destinés à faciliter la mise en œuvre matérielle.

Vous disposez du matériel suivant :

- Une carte Arduino Mega2560 ;
- Un shield RTC DS1307 ;
- Un shield module GPS ;
- Un shield capteur BME680 ;
- Un shield écran TFT + dalle tactile Adafruit 2050 ;
- Une nappe de câbles 20 brins mâle – femelle ;
- Un adaptateur secteur pour alimentation de la carte Arduino sans le câble USB ;
- Une pile 3V CR1225, pour le module RTC DS1307
- Un cordon USB pour alimentation de la carte Arduino et téléchargement du code ;

3.1. Carte microcontrôleur Arduino Mega2560

La cible matérielle sera la carte Arduino Mega2560 Rev3.0 architecturée autour du microcontrôleur Atmega 2560. Les principaux éléments qui composent cette carte sont décrits ci-dessous (cf. Figure 3) et (cf. Figure 4).

Les principales caractéristiques à retenir sont :

- Microcontrôleur : ATmega2560
- Digital I/O Pins : 54, dont 15 permettant la génération d'un signal de sortie PWM³
- Analog Input Pins : 16
- Flash Memory (mémoire de programme) : 256 KB dont 8 KB utilisés pour le bootloader
- SRAM (zone mémoire de stockage des variables globales) : 8 KB
- EEPROM : 4 KB
- Fréquence horloge : 16 MHz
- LED_BUILTIN : Constante symbolique désignant la broche 13 de la led pré câblée
- 4 ports de communication série
- Gestion du bus I2C (broches SCL et SDA)

La documentation constructeur du microcontrôleur est accessible via le lien : <http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf>

Des informations complémentaires sur la carte Arduino Mega2560 peuvent être consultées sur <https://store.arduino.cc/arduino-mega-2560-rev3>.

³ PWM : Pulse Width Modulation ou Modulation par Largeur d'Impulsion (MLI). Procédé qui permet de générer une tension continue par exemple.

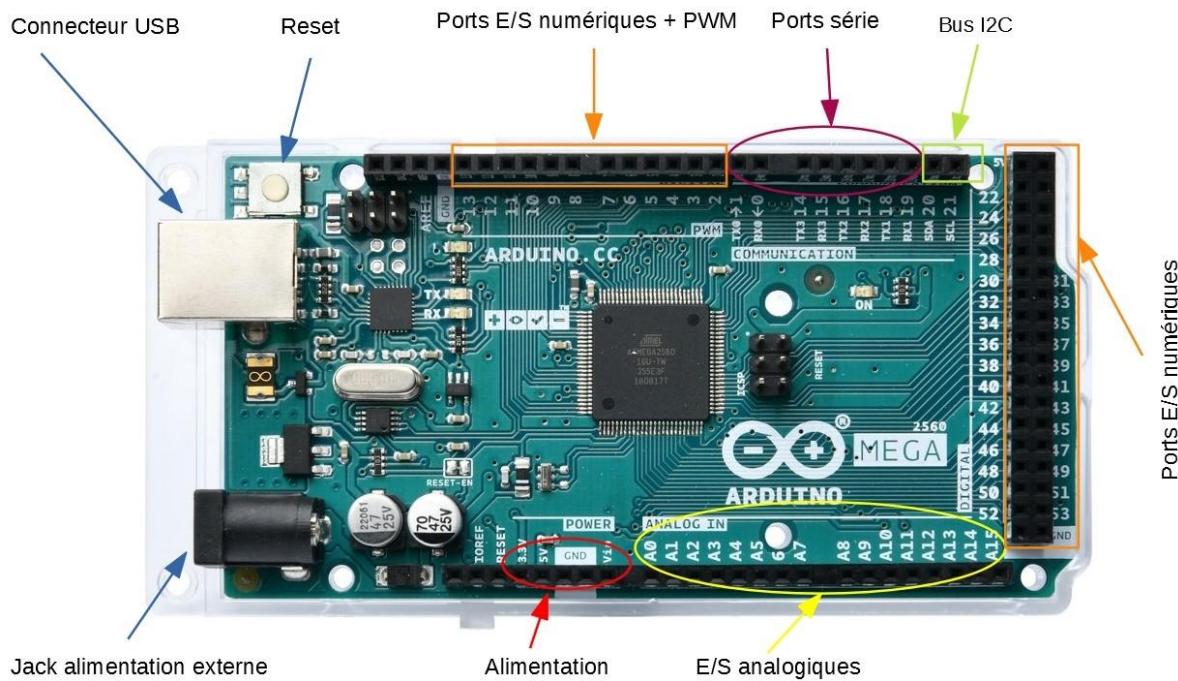


Figure 3 : Carte Arduino Mega2560 Rev 3.0

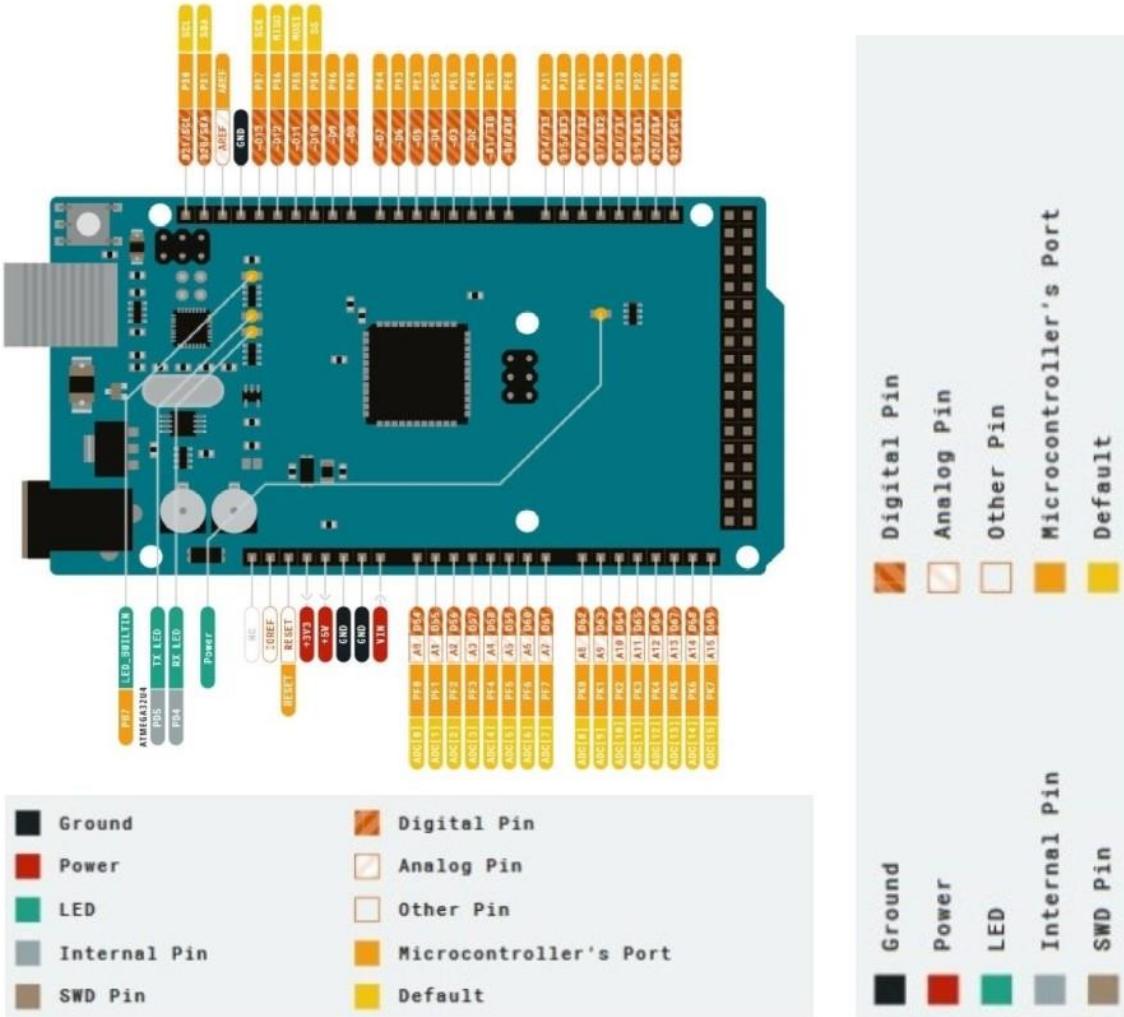


Figure 4 : Typologie fonctionnelle des entrées - sorties de la carte Arduino Mega2560 Rev 3.0

La figure ci-dessous permet d'identifier l'ensemble des broches du microcontrôleur, avec en rouge la nomenclature dédiée au « monde Arduino ». Ce schéma permet également de préciser le rôle de chacune des broches du point de vue fonctionnel (cf. <https://www.arduino.cc/en/Hacking/PinMapping2560>) et (cf. Figure 5).

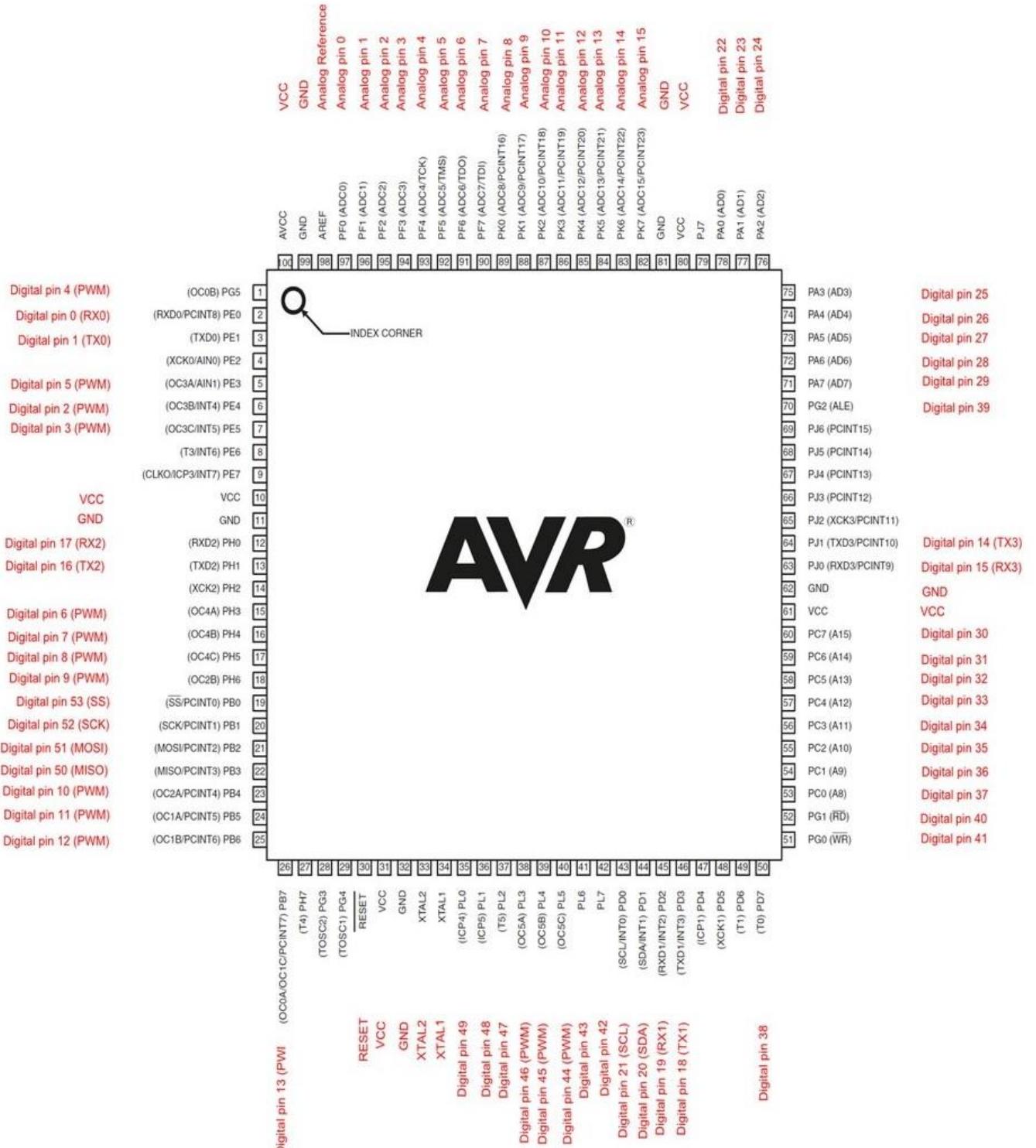


Figure 5 : Nomenclature « Arduino » des broches Arduino Mega2560

Parmi les broches que nous utiliserons, on retrouve les broches d'E/S digitales, ou analogiques.

3.1.1. Entrées – sorties numériques (ou digitales)

Au nombre de 45, elles correspondent aux broches D2 à D13 et D22 à D54. Elles peuvent être utilisées aussi bien en entrée qu'en sortie. Il est impératif de spécifier avant toute utilisation quel sera le « sens » d'utilisation, par une initialisation matérielle gérée à l'aide de la fonction (*pinMode*). Lorsque cette première étape est réalisée, alors il est possible de lire l'état d'une broche, ou bien de définir son état, en utilisant les fonctions *digitalWrite()* ou *digitalRead()* sous l'environnement Arduino. Une broche numérique ne peut avoir que deux états logique, haut ou bas, correspondant à une tension respectivement de 5V ou 0V.

La documentation des fonctions de gestion des ports d'E/S numériques est accessible via les liens suivants :

- *pinMode()* : initialiser le mode de fonctionnement d'une broche
 - o <https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/>
- *digitalWrite()* : définir l'état d'une broche numérique
 - o <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/>
- *digitalRead()* : lire l'état d'une broche numérique
 - o <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/>

En plus de ces fonctionnalités, il est possible de générer des signaux PWM sur les broches D2 à D13 et D44 à D46 à l'aide de la fonction *analogWrite()*.

(cf. <https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/>).

3.1.2. Entrées – sorties analogiques

Elles correspondent aux broches A0 jusqu'à A15. Il peut être associé à chacune de ces 16 broches un convertisseur analogique / numérique sur 10 bits. La conversion analogique – numérique s'effectue à l'aide de la fonction *analogRead()*

(cf. <https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>).

3.1.3. Bus de communication I2C

Les broches D20 (SDA) et D21 (SCL) jouent un rôle particulier lors de l'utilisation du bus de communication I2C. Ce bus utilise 2 fils, SCL et SDA. SDA permet de gérer l'échange des données qui transitent sur le bus tandis que SCL permet de gérer l'horloge qui assure la synchronisation des échanges de données. Le dialogue avec un périphérique I2C nécessite l'utilisation de la bibliothèque *Wire* dont la documentation est disponible sur <https://www.arduino.cc/en/Reference/Wire>.

3.1.4. Bus de communication série

La carte ArduinoMega2560 possède 4 ports de communication, référencés TX0- RX0 à TX3 – RX3. L'affectation des broches est définie au sein du tableau ci-dessous (cf. Tableau 1).

	Serial pins	Serial1 pins	Serial2 pins	Serial3 pins
Mega2560	0(RX), 1(TX)	19(RX), 18(TX)	17(RX), 16(TX)	15(RX), 14(TX)

Tableau 1 : affectation des broches aux ports série de la carte Mega2560

Le port de communication de base Serial (broches 0 et 1) est affecté à la communication entre la carte et un PC (et aussi via un terminal série) au travers d'une communication USB. Ce port série est également affecté à la gestion du téléchargement du code dans la mémoire du microcontrôleur. Tout

périphérique connecté sur ce port série provoquera une erreur lors du téléchargement d'un programme.

L'utilisation de la communication série nécessite l'utilisation de la bibliothèque Serial (inclus dans l'environnement de développement Arduino) et, à minima, des méthodes de base suivantes :

- *begin()* : initialiser les paramètres de la liaison série ;
- *read()* : lecture d'un caractère sur la liaison série ;
- *write()* : écriture (envoi) d'un caractère sur la liaison série
- *print()* : affichage de données au sein d'un terminal série
- *println()* : affichage de données au sein d'un terminal série et retour à la ligne suivante

dont la description est accessible depuis :

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

3.2. Shield Grove Mega2560

Il s'agit d'une carte d'interface destinée à faciliter la mise en œuvre matérielle de périphériques d'E/S sur la carte Arduino utilisée (cf. Figure 6) et (cf. https://wiki.seeedstudio.com/Grove-Mega_Shield/).

Elle dispose d'un format de connectique spécifique.

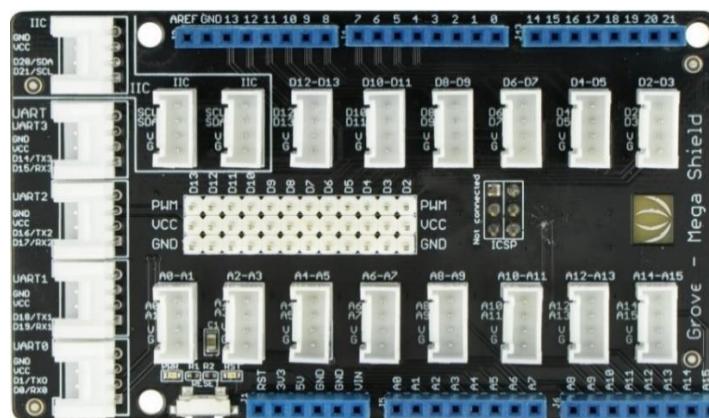


Figure 6 : Shield de connexion Grove pour Mega2560

Du point de vue fonctionnel (cf. Figure 7 et cf. Figure 8), nous retrouvons l'accès :

- Aux ports d'E/S numériques ;
- Aux ports d'E/S analogiques
- Au bus de communication I2C, série, ...

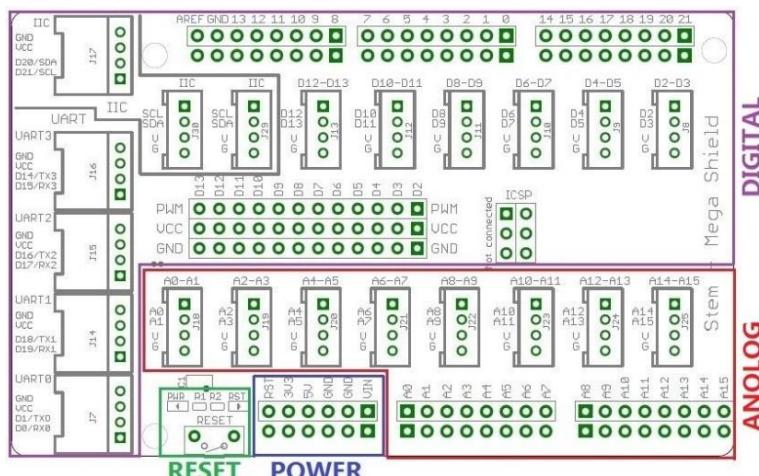


Figure 7: Fonctionnalités accessibles sur shield Grove Mega2560 (1)

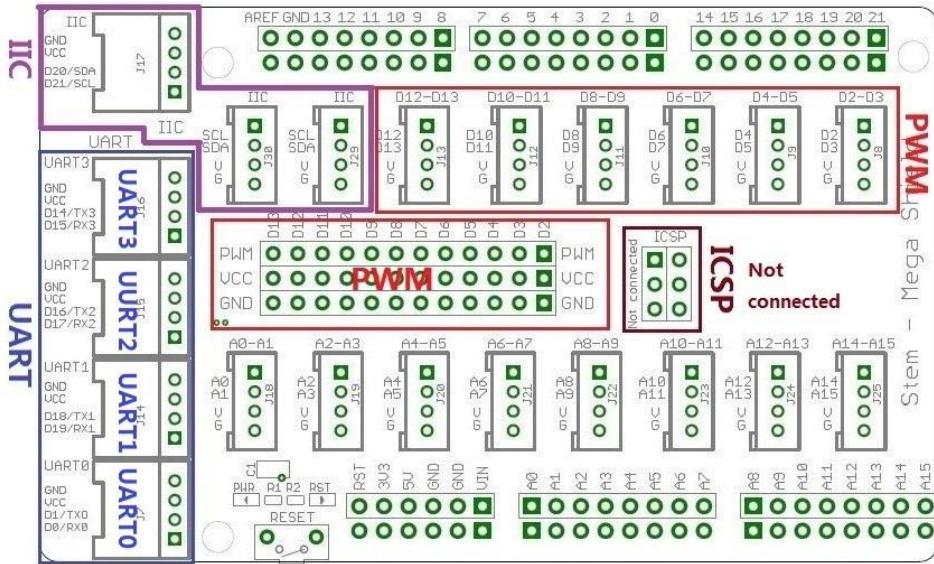


Figure 8: Fonctionnalités accessibles sur shield Grove Mega2560 (2)

Cette carte viendra s'enficher sur la carte Arduino comme illustré ci-dessous (cf. Figure 9).

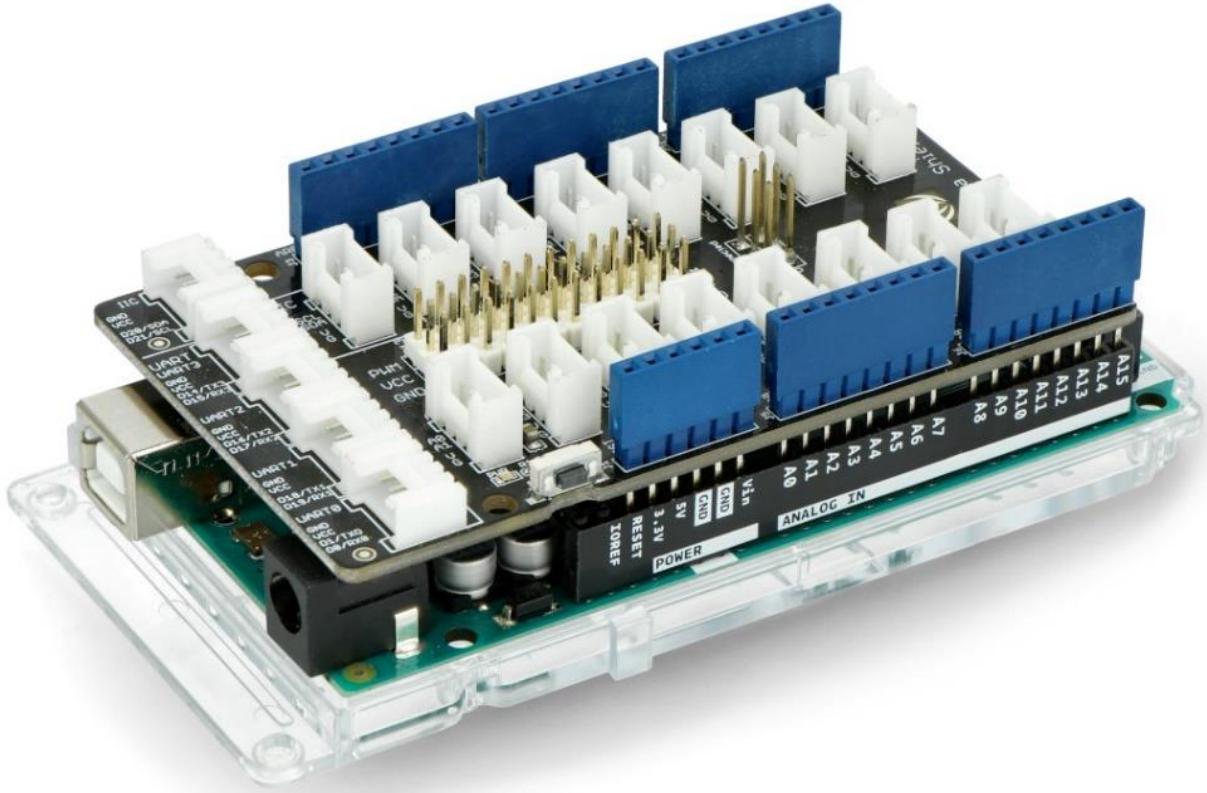


Figure 9: Carte Arduino Mega2560 et shield Grove

3.3. Module horloge temps réel RTC

Les fonctionnalités de gestion du temps vont s'appuyer sur la gestion d'un circuit d'horloge temps réel (RTC – Real Time Clock) DS1307, compatible avec la connectique Grove. Pour cela, nous utiliserons le circuit RTC DS1307, monté sur un shield spécifique (cf. Figure 10).

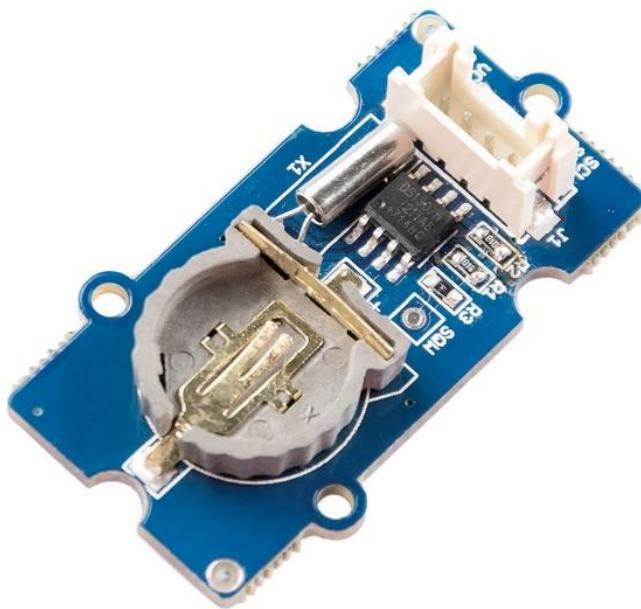


Figure 10: Shield du circuit RTC DS1307

De plus, le shield pourra être équipé d'une pile de sauvegarde permettant ainsi au circuit DS1307 de maintenir à jour ses registres internes de date et heure en l'absence d'alimentation.

En terme de fonctionnalités, le circuit DS1307 permet de gérer un calendrier (jour – mois – année – jour de la semaine), y compris les années bissextiles, ainsi qu'une horloge (hh : mm : ss) au format 12h ou 24h.

Ce circuit RTC communique via le bus I2C. Son adresse I2C est \$68.

Les différents champs de date et d'heure sont organisés selon le schéma mémoire suivant. Chaque champ est codé sur 1 octet (cf. Figure 11) L'organisation de chaque champ est définie selon la figure ci-dessous (cf. Figure 12).

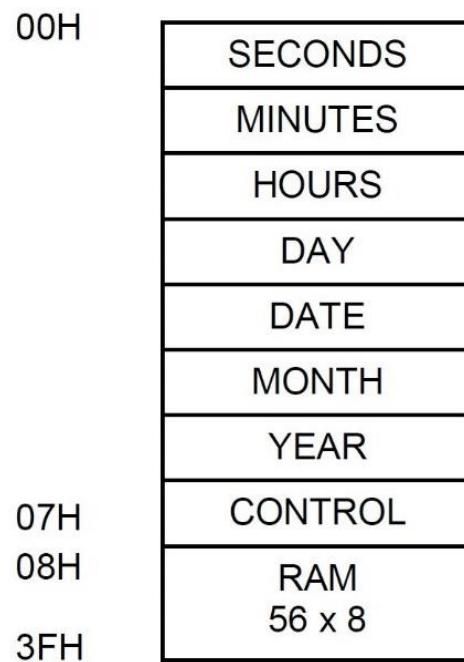


Figure 11: DS1307- Carte mémoire des données de date et jour

		BIT7							BIT0
00H	CH		10 SECONDS		SECONDS				00-59
	X		10 MINUTES		MINUTES				00-59
	X	12 24	10 HR A/P	10 HR		HOURS			01-12 00-23
	X	X	X	X	X	DAY			1-7 01-28/29 01-30 01-31
	X	X	10 DATE		DATE				
	X	X	X	10 MONTH	MONTH				01-12
			10 YEAR		YEAR				00-99
07H	OUT	X	X	SQWE	X	X	RS1	RS0	

Figure 12 : DS1307 - Formatage de chaque octet

Le contenu de la date et de l'heure est codé en « Decimal Codé Binaire » (DCB). Le bit 7 du champ des secondes (CH) permet d'activer l'oscillateur du circuit d'horloge. Pour que l'horloge s'incrémente, il est nécessaire que CH = 0.

Références documentaires :

- <https://datasheets.maximintegrated.com/en/ds/DS1307.pdf>
- <https://wiki.seeedstudio.com/Grove-RTC/>

3.4. Capteur BME680

Le capteur BME680 de Bosch permet d'effectuer des mesures de la température extérieure, du taux d'humidité et de la pression atmosphérique. En plus de ces capacités, il est possible d'effectuer une mesure de la qualité de l'air en obtenant une valeur d'IAQ (Index Air Quality) ainsi que la valeur de paramètres supplémentaires associés à l'estimation de la mesure de l'IAQ. L'ensemble des informations techniques est disponible à partir du lien ci-dessous, aussi bien sur la partie caractéristiques / fonctionnalités que pour la mise en œuvre pratique et logicielle.

<https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors-bme680/#documents>

Ce capteur peut communiquer avec une plateforme microcontrôleur en s'interfaisant avec le bus I2C. Son adresse I2C est figée et fixée à \$76 ou \$77, en fonction de la configuration matérielle du capteur (liée à la broche SDO).

En ce qui concerne sa mise en œuvre matérielle, nous utiliserons un shield compatible avec la connectique Grove (cf. Figure 13), qui prend en compte l'ensemble des éléments matériels pour son interfaçage avec la carte Arduino Mega2560.

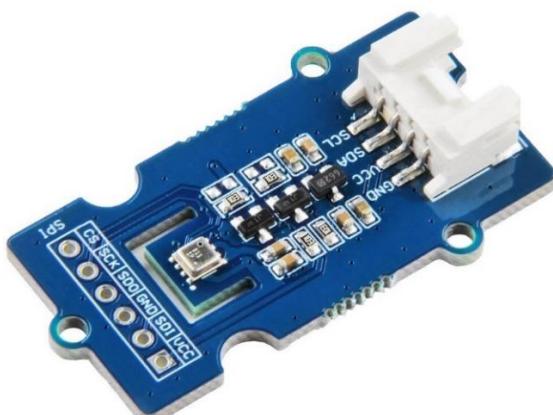


Figure 13: Shield BME680 Grove

L'ensemble des informations techniques sont disponibles depuis https://wiki.seeedstudio.com/Grove-Temperature_Humidity_Pressure_Gas_Sensor_BME680/.

3.5. Module GPS

Le module GPS utilisé est conçu autour du module SIM28. Il est mis en œuvre sur un shield (cf. Figure 14) qui permet la connexion au format Grove sur un port série. Une documentation plus complète est accessible via le lien suivant (cf. <https://www.seeedstudio.com/Grove-GPS-Module.html>) Il supporte la réception des trames NMEA⁴.

Afin d'améliorer la qualité de réception du signal GPS, le module utilisé est doté d'une antenne externe.

Il est configurable :

- Choix de la vitesse de transmission du port série. Par défaut : 9600 bauds ;
- Choix des types de messages NMEA reçus.

Le paramétrage du module GPS s'effectue par des commandes « ASCII »⁵, envoyées sous forme de chaîne de caractères sur la liaison série à laquelle il est connecté.



Figure 14: Module Grove GPS SIM28

3.6. Ecran TFT couleur avec dalle tactile

L'interface utilisateur sera accessible à l'aide d'un écran TFT couleur muni d'une dalle tactile. Le shield écran est le shield Adafruit 2050 (cf. Figure 15).

Un guide de mise en œuvre est également disponible sur :

<https://www.digikey.co.uk/htmldatasheets/production/1843711/0/0/1/3-5-320x480-Color-TFT-Touchscreen-Breakout.pdf>

Les principales caractéristiques de l'écran sont résumées ci-dessous.

- Ecran couleur RGB 16 bits, 320 x 480 pixels, diagonale de 3.5 pouces ;
- Contrôleur graphique HXD8357D ;
- Dalle tactile résistive ;
- Bus de données 8 bits ;
- Gestion possible du rétroéclairage.

L'utilisation de l'écran TFT et de la dalle tactile nécessitera l'utilisation de librairies spécifiques. Ce point sera abordé ultérieurement.

⁴ NMEA - National Marine & Electronics Association

⁵ ASCII : American Standard Code for Information Interchange

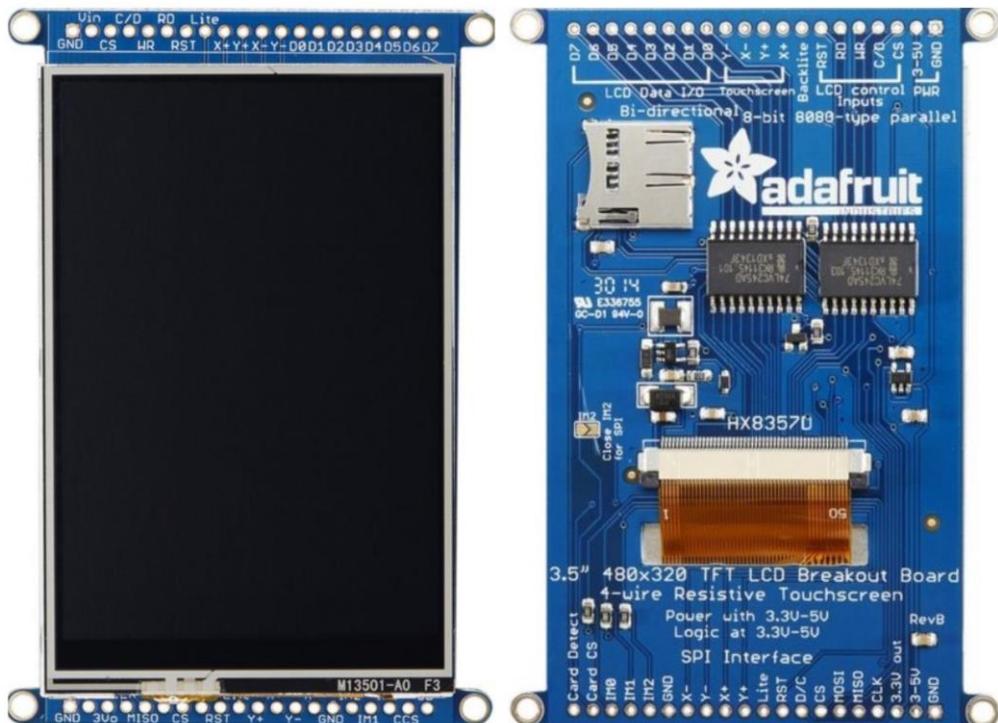


Figure 15 : Ecran TFT couleur 3.5'' et dalle tactile résistive

4. Ressources logicielles

L'environnement de programmation sera l'IDE Arduino. La version de l'environnement Arduino choisie est la version **1.8.8**, afin de s'assurer que l'ensemble des librairies logicielles utilisées seront correctement supportées.

Par la suite, lors du lancement de l'IDE, il ne faudra pas effectuer la mise à jour vers une version plus récente.



Cet environnement intègre un éditeur de texte, l'appel à un compilateur et éditeur de liens. Il permet également de gérer le téléchargement du code à exécuter dans la mémoire du microcontrôleur.

La structure générale d'un programme au sein de cet environnement sera également abordée.

4.1. Installation de l'IDE Arduino

L'installation de la version 1.8.8 s'effectue depuis la page de téléchargement du site Arduino (cf. <https://www.arduino.cc>). Afin de d'installer une version antérieure à la version la plus récente, il est nécessaire d'aller sur la page :

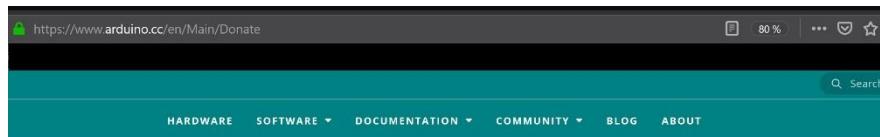
<https://www.arduino.cc/en/Main/OldSoftwareReleases#previous> (cf. Figure 16).

L'IDE peut être installé sous environnement Windows, Linux ou Mac OS X. Par la suite, nous nous limiterons à un environnement Windows. Il existe deux possibilités d'installation :

- soit en utilisant l'installateur Windows fourni (cf. Figure 17);
 - ou bien en téléchargeant l'archive qui contiendra l'ensemble des fichiers et répertoires requis à l'utilisation de l'IDE (cf. Figure 18). Une fois l'archive décompressée, il suffit de placer l'ensemble des fichiers et répertoires, sans rien modifier, dans le répertoire de votre choix.

1.8.11	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM 32 Linux ARM 64	Source code on Github
1.8.10	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM 32 Linux ARM 64	Source code on Github
1.8.9	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM 32 Linux ARM 64	Source code on Github
1.8.8	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM 32 Linux ARM 64	Source code on Github
1.8.7	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM 32 Linux ARM 64	Source code on Github
1.8.6	Windows Windows Installer	MAC OS X	Linux 32 Bit Linux 64 Bit Linux ARM	Source code on Github

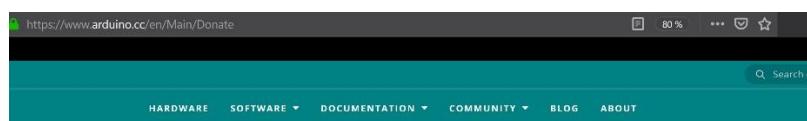
Figure 16: page de téléchargement IDE Arduino 1.8.8



Contribute to the Arduino Software

Consider supporting the Arduino Software by contributing to its development. (US tax payers, please note this contribution is not tax deductible). Learn more on how your contribution will be used.

Figure 17: Téléchargement de l'installateur Windows IDE Arduino



Contribute to the Arduino Software

Consider supporting the Arduino Software by contributing to its development. (US tax payers, please note this contribution is not tax deductible). Learn more on how your contribution will be used.

Figure 18: Téléchargement de l'archive de l'IDE Arduino

Remarques :

- Utiliser l'installateur Windows conduira à intégrer l'environnement de programmation au sein de l'OS, avec mise à jour des clés de registres. L'installation d'une autre version nécessitera alors de désinstaller celle déjà installée. Un répertoire supplémentaire *Arduino\librairies* est également créé, et contiendra les bibliothèques de base associées à l'IDE. Cela facilite la gestion des bibliothèques en cas de réinstallation de l'IDE.
- Utiliser l'archive complète permet de faire cohabiter différentes versions de l'IDE.

En utilisant l'installateur Windows, vous devez avoir un répertoire dont le contenu correspond à celui de la figure ci-dessous (cf. Figure 19).

Nom	Modifié le	Type	Taille
drivers	02/07/2020 15:23	Dossier de fichiers	
examples	02/07/2020 15:23	Dossier de fichiers	
hardware	02/07/2020 15:23	Dossier de fichiers	
java	02/07/2020 15:23	Dossier de fichiers	
lib	02/07/2020 15:23	Dossier de fichiers	
libraries	02/07/2020 15:23	Dossier de fichiers	
reference	02/07/2020 15:23	Dossier de fichiers	
tools	02/07/2020 15:23	Dossier de fichiers	
tools-builder	02/07/2020 15:23	Dossier de fichiers	
arduino	06/12/2018 11:25	Application	395 Ko
arduino.l4j	06/12/2018 11:25	Paramètres de configuration	1 Ko
arduino_debug	06/12/2018 11:25	Application	393 Ko
arduino_debug.l4j	06/12/2018 11:25	Paramètres de configuration	1 Ko
arduino-builder	14/06/2017 11:31	Application	3 220 Ko
libusb0.dll	06/12/2018 11:24	Extension de l'application	43 Ko
msvcp100.dll	06/12/2018 11:24	Extension de l'application	412 Ko
msvcr100.dll	06/12/2018 11:24	Extension de l'application	753 Ko
revisions	06/12/2018 11:24	Document texte	88 Ko
uninstall	02/07/2020 15:23	Application	404 Ko
wrapper-manifest	06/12/2018 11:25	Document XML	1 Ko

Figure 19: Répertoire d'installation de l'IDE Arduino

De plus, un répertoire *Arduino* est généralement créé à la racine de votre lecteur de données. Il contiendra les fichiers des librairies qui ne sont pas installées lors de l'installation de l'IDE Arduino.

4.2. Description des fonctions de l'IDE

Pour ouvrir et lancer l'environnement de programmation, il faut cliquer sur le raccourci  qui permet l'accès aux différentes fonctionnalités de l'éditeur. Il est probable qu'une version plus récente soit disponible, mais il ne faut pas procéder à l'upgrade (cf. Figure 20) (cf. Figure 21).

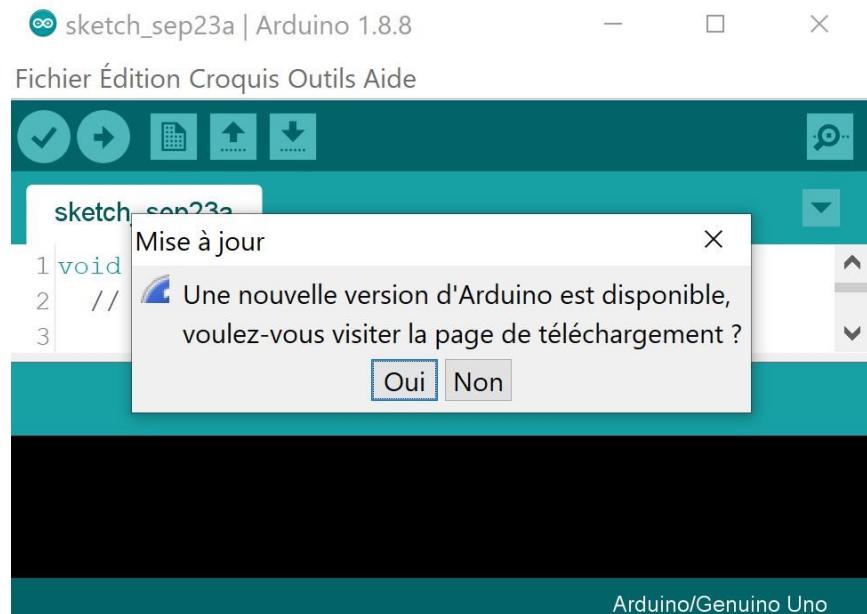


Figure 20: Ouverture de l'IDE Arduino

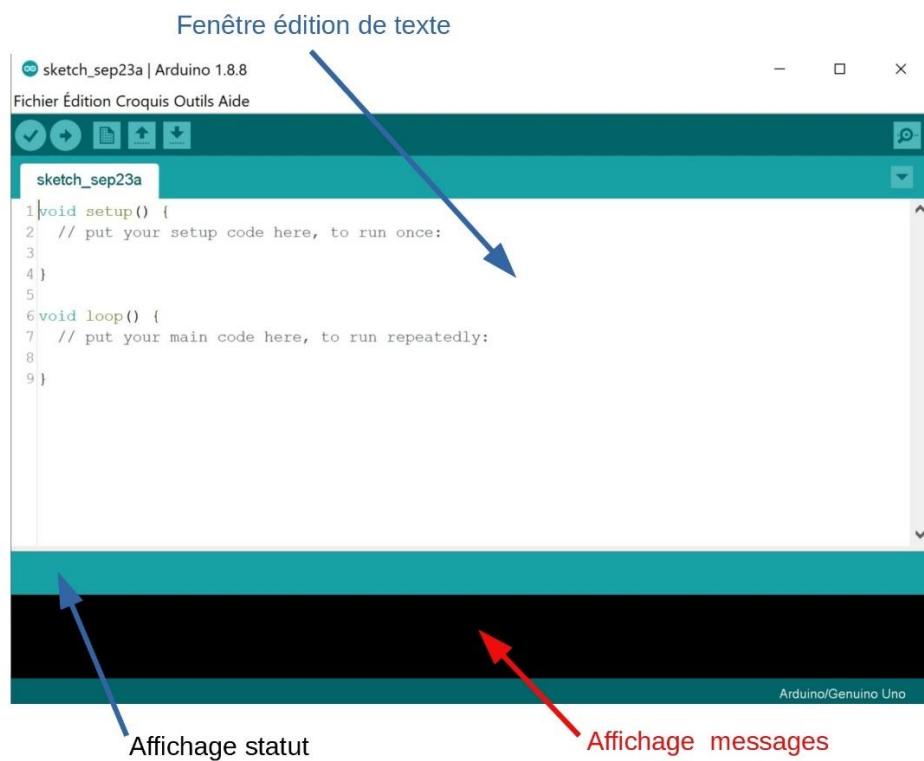


Figure 21: Fenêtre de l'éditeur IDE Arduino

Cet environnement dispose de fonctions également accessibles par des raccourcis présents dans la barre contenant les icônes suivantes (cf. Figure 22) :

- « Vérifier » : permet de détecter des erreurs de syntaxe d'un programme ;
- « Téléverser » : permet de transférer le code exécutable d'un programme vers la mémoire du microcontrôleur ;
- « Nouveau » : pour ouvrir un nouveau croquis ;
- « Ouvrir » : pour ouvrir un croquis déjà existant ;
- « Enregistrer » : pour sauvegarder un croquis ;

- « Moniteur Série » : permet d'ouvrir une fenêtre texte pour afficher des messages texte via le port série.

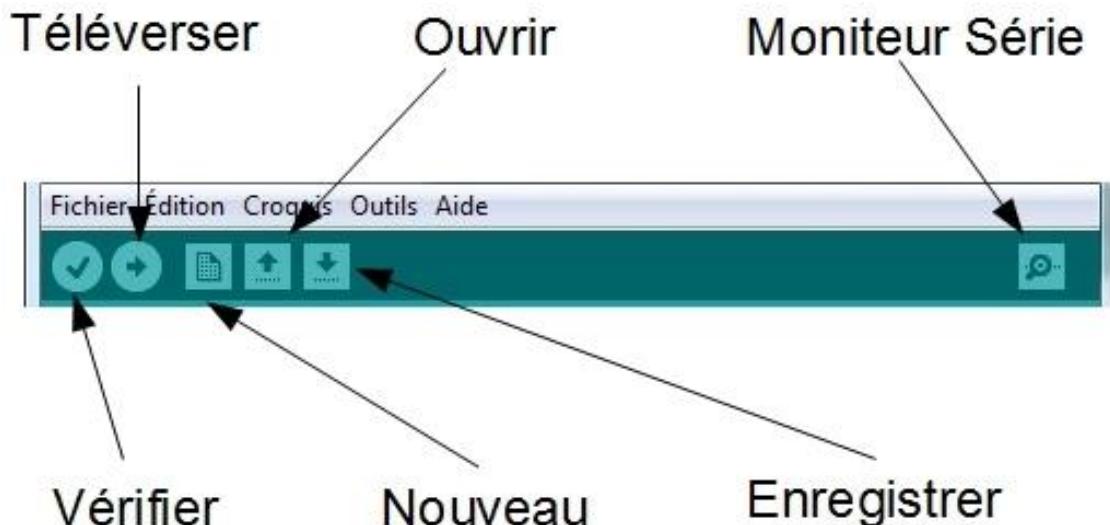


Figure 22: Raccourcis IDE Arduino

4.3. Structure de base d'un programme Arduino

4.3.1. Programme Blink

La carte Arduino Mega2560 possède une led pré câblée connectée sur le port 13. Il est possible d'accéder au sein de la base des exemples au programme *Blink* (cf. Figure 24) qui permet de faire clignoter cette led. Afin d'accéder à ce programme :

Fichier → Exemples → 01. Basics → Blink (cf. Figure 23) et (cf. Figure 24).

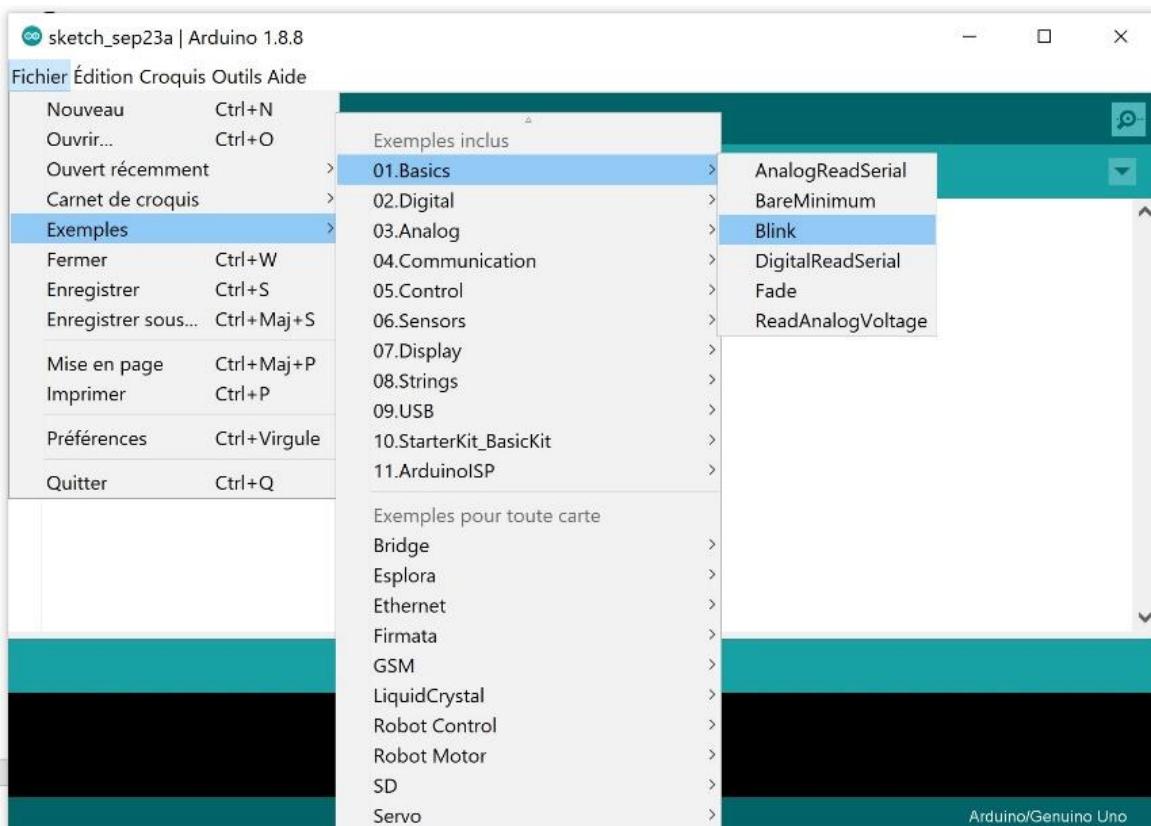


Figure 23: Ouverture de Blink

```

1 // 
2 Blink
3
4 Turns an LED on for one second, then off for one second, repeatedly.
5
6 Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
7 it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
8 the correct LED pin independent of which board is used.
9 If you want to know what pin the on-board LED is connected to on your Arduino
10 model, check the Technical Specs of your board at:
11 https://www.arduino.cc/en/Main/Products
12
13 modified 8 May 2014
14 by Scott Fitzgerald
15 modified 2 Sep 2016
16 by Arturo Guadalupi
17 modified 8 Sep 2016
18 by Colby Newman
19
20 This example code is in the public domain.
21
22 http://www.arduino.cc/en/Tutorial/Blink
23 */
24
25 // the setup function runs once when you press reset or power the board
26 void setup() {
27   // initialize digital pin LED_BUILTIN as an output.
28   pinMode(LED_BUILTIN, OUTPUT);
29 }
30
31 // the loop function runs over and over again forever
32 void loop() {
33   digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
34   delay(1000);                      // wait for a second
35   digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
36   delay(1000);                      // wait for a second
37 }

```

Figure 24: Programme Blink

Un programme Arduino est structuré autour de deux fonctions de base, *setup* et *loop*.

La fonction *setup* est exécutée **une seule fois** lors du lancement du programme. Elle contient toutes les initialisations.

La fonction *loop* est exécutée de façon répétitive. C'est en fait une « boucle infinie ». **Aucune** fonction *main* n'apparaît.

L'accès aux ressources documentaires de cet exemple de programme s'effectue par le lien : <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink>, en particulier :

- Fonction *loop()*
- Fonction *setup()*
- Fonction *delay()*
- Fonction *digitalWrite()*
- Fonction *pinMode()*

4.3.2. Exécution du programme sur la carte Arduino

Avant de lancer l'exécution du programme sur la carte Arduino Mega2560, deux étapes préalables sont nécessaires :

1. Définir la carte microcontrôleur cible Outils → Type de carte et sélectionner la carte Arduino Mega2560 (cf. Figure 25).

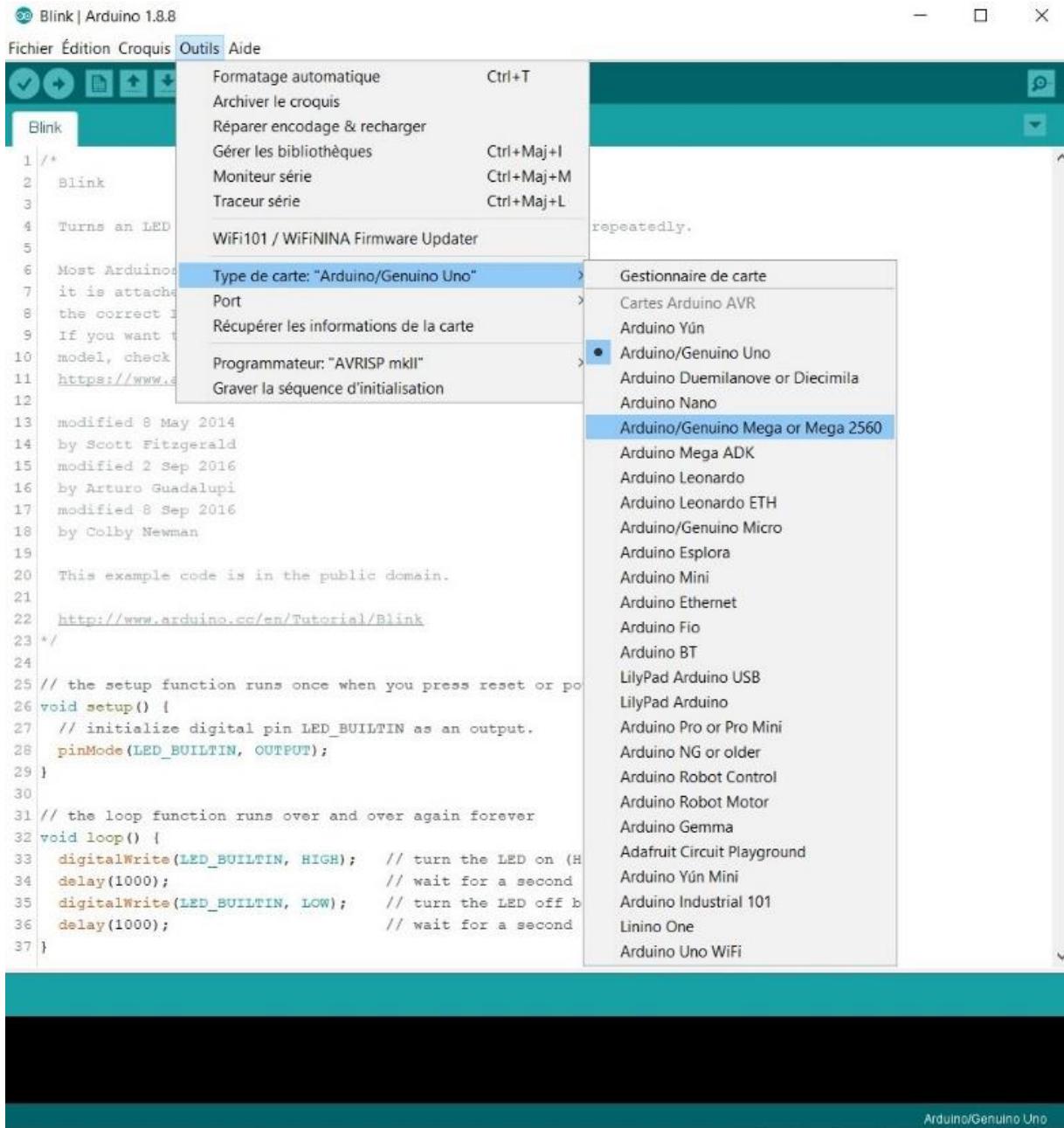


Figure 25: Sélection de la carte microcontrôleur cible

2. Il est ensuite possible de vérifier que le code compile sans erreur, en utilisant le raccourci « Vérifier ». Si la syntaxe du programme est correcte, un message s'affichera dans la fenêtre des messages, sinon, ce seront des messages d'erreurs (cf. Figure 26).
3. Lorsque le code est compilé sans erreur, il est possible de le télécharger dans la mémoire de programme du microcontrôleur. Auparavant, il faut configurer l'environnement de programmation en spécifiant le port COM ouvert lors de la connexion de la carte sur un port USB avec Outils → Port (cf. Figure 27). Le port COM ouvert avec la carte microcontrôleur peut être identifié à l'aide du gestionnaire de périphériques, en développant l'onglet Ports (COM et LPT) (cf. Figure 28).

The screenshot shows the Arduino IDE interface with the 'Blink' sketch open. The code is displayed in the main editor window, which includes comments explaining the purpose of the sketch and the history of its modifications. Below the code, a message indicates that compilation has completed successfully. A status bar at the bottom provides information about the core being used and the port selected.

```

/*
  Blink
  Turns an LED on for one second, then off for one second, repeatedly.

Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to the correct LED pin independent of which board is used.

If you want to know what pin the on-board LED is connected to on your Arduino model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products

modified 8 May 2014
by Scott Fitzgerald
modified 2 Sep 2016
by Arturo Guadalupi
modified 8 Sep 2016
by Colby Newman

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/Blink
*/
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}

```

Compilation terminée.

Archiving built core (caching) in: C:\Users\gaucher\AppData\Local\Temp\arduino_cache_428933\core\core_arduino_ Le croquis utilise 1460 octets (0%) de l'espace de stockage de programmes. Le maximum est de 253952 octets. Les variables globales utilisent 9 octets (0%) de mémoire dynamique, ce qui laisse 8183 octets pour les variab Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560)

Figure 26: Vérification code programme Blink

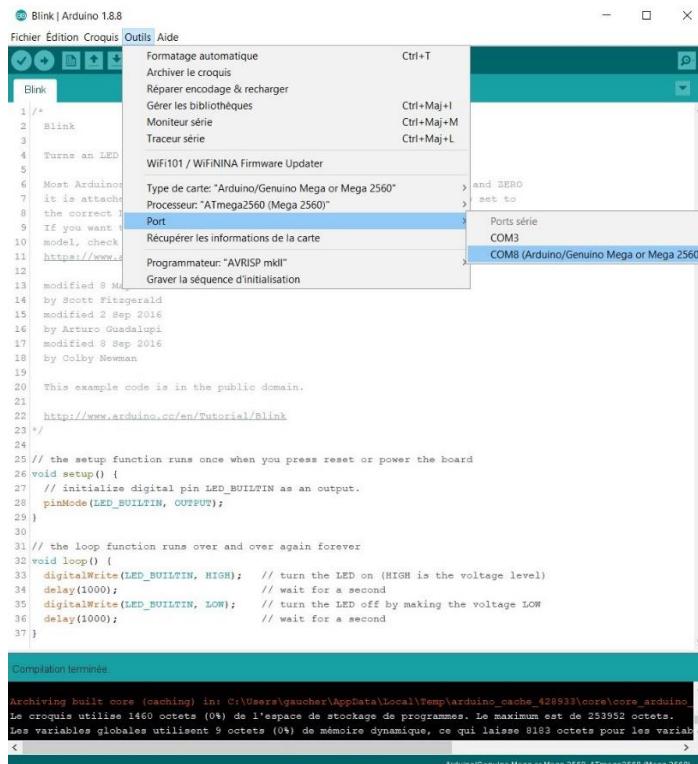


Figure 27 : Initialisation du port COM au sein de l'IDE Arduino

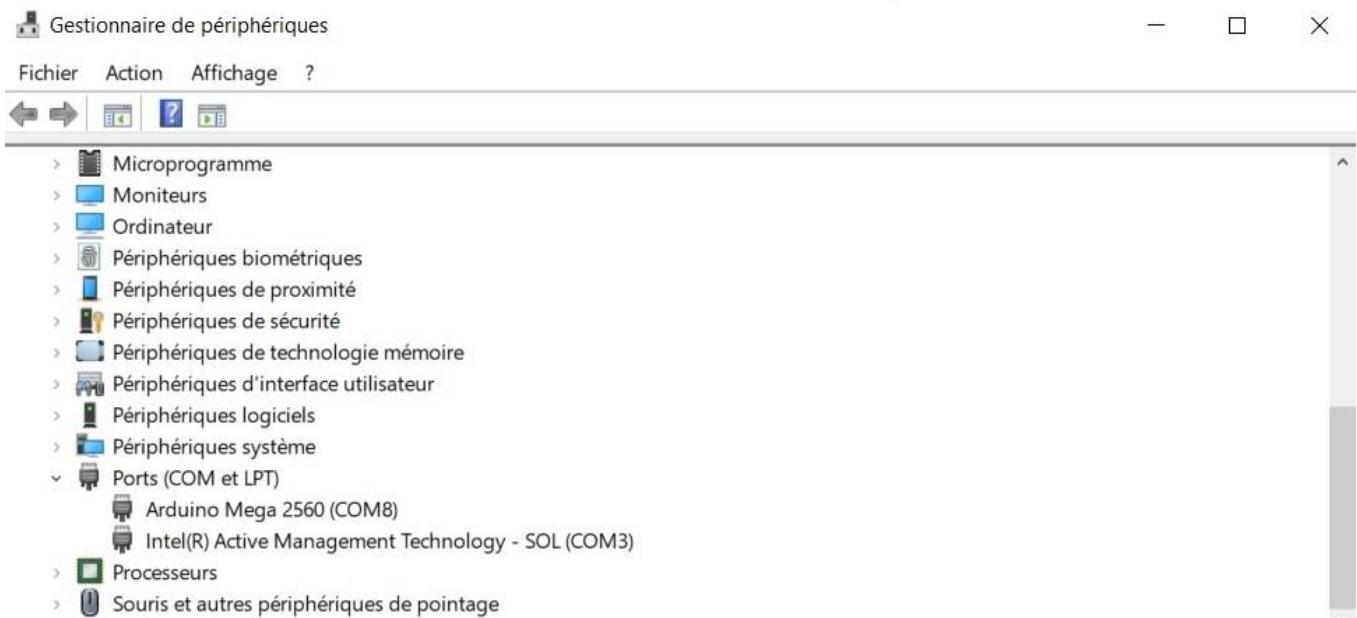


Figure 28: Gestionnaire de périphériques

4. Le raccourci « *Téléverser* » permet alors de procéder à la compilation du code, **puis** à son téléchargement au sein de la mémoire de programme du microcontrôleur. Lorsque le téléchargement est achevé, l'exécution du programme débute.

Remarques :

Lorsqu'un programme a été téléchargé une fois :

- Il est possible d'en provoquer une nouvelle exécution en appuyant sur le bouton reset de la carte microcontrôleur.
- Lorsque l'on connecte la carte microcontrôleur sur un port USB, le dernier programme téléchargé s'exécute.

5. Gestion de la liaison série

La carte Arduino Mega2560 possède 4 ports de communication série (cf. §3.1.4). Afin de pouvoir tracer des événements lors de l'exécution d'un programme, il est utile de pouvoir afficher des messages sous la forme de chaîne de caractères au sein d'un terminal série, accessible depuis l'IDE. Dans ce cas, le port série par défaut est *Serial*.

Afin de pouvoir utiliser ce terminal série, il est nécessaire d'initialiser la vitesse de transmission en utilisant la méthode *begin()* au sein de la fonction *setup()*.

(cf. <https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/>)

Une fois cette initialisation effectuée, l'affichage de message est provoqué par l'appel à la méthode *print()* ou *println()*.

(cf <https://www.arduino.cc/reference/en/language/functions/communication/serial/print/>)

(cf. <https://www.arduino.cc/reference/en/language/functions/communication/serial/println/>)

L'ouverture du terminal série de l'IDE s'effectue en cliquant sur le raccourci accessible dans la fenêtre de l'IDE. Cela provoque également l'exécution du programme, comme après un reset de la carte. Il faut ensuite veiller à ce que le paramétrage de la liaison série et celui de la fenêtre du terminal série qui s'ouvre soient identiques.

Mise en œuvre :

Modifier le programme *Blink* afin d'afficher au sein du terminal série de l'IDE l'état de la led à l'aide de messages que vous définirez (cf. Figure 29). Pensez à renommer le fichier avant sauvegarde.

Remarque :

- Les autres ports série Serial1, Serial2 et Serial3 peuvent être utilisés pour connecter tout périphérique communicant via ce type d'interface. Toutes les méthodes vues précédemment peuvent être utilisées. En revanche, il faut bien penser à initialiser chaque port série avec la bonne vitesse de transmission.

The screenshot shows the Arduino IDE interface. The main window displays the code for the *Blink_Modif* sketch:

```
1 //  
2 Blink  
3  
4 Turns an LED on for one second, then off for one second, repeatedly.  
5  
6 Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO  
7 it is attached to digital pin 13, on MRR1000 on pin 6. LED_BUILTIN is set to  
8 the correct LED pin independent of which board is used.  
9 If you want to know what pin the on-board LED is connected to on your Arduino  
10 model, check the Technical Specs of your board at:  
11 https://www.arduino.cc/en/Main/Products  
12  
13 modified 8 May 2014  
14 by Scott Fitzgerald  
15 modified 2 Sep 2016  
16 by Arturo Guadalupi  
17 modified 8 Sep 2016  
18 by Colby Newman  
19  
20
```

Below the code editor is the Serial Monitor window titled "COM8 (Arduino/Genuino Mega or Mega 2560)". It shows the output of the sketch, which alternates between "Led On : 1s" and "Led Off : 1s". The monitor includes controls for automatic scrolling, timestamping, and baud rate selection (9600 baud).

At the bottom of the IDE, a status bar provides memory usage information:

Téléversement terminé
Le croquis utilise 2626 octets (1%) de l'espace de stockage de programmes. Le maximum est de 253952 octets.
Les variables globales utilisent 212 octets (2%) de mémoire dynamique, ce qui laisse 7980 octets pour les variab

Figure 29: Utilisation du terminal série IDE Arduino

6. Gestion des interruptions - Timers

6.1. Définition – Principe

Une interruption est constituée par un événement externe au programme en cours d'exécution. Lorsqu'une interruption survient, le programme en cours d'exécution est interrompu et le

microcontrôleur exécute alors un programme particulier appelée **routine d'interruption**, avant de redonner la main au programme interrompu.

Pour un microcontrôleur, plusieurs sources d'interruptions peuvent être considérées :

- Interruptions externes :
 - o Changement de l'état d'une broche ;
 - o Détection d'un front montant et / ou descendant sur une broche ;
- Interruptions internes :
 - o Sur un événement particulier associé à un timer ;
 - o Lors d'une Conversion Analogique Numérique ;
 - o Associée à la gestion d'une liaison série, ...

Par la suite, nous allons nous intéresser uniquement à la gestion des interruptions générées par les timers, et plus particulièrement aux interruptions sur débordement du timer (Timer Overflow). Cette gestion des interruptions associées aux timers nécessitera de manipuler explicitement des registres du microcontrôleur.

6.2. Gestion des interruptions

6.2.1. Gestion des interruptions : principes

Le microcontrôleur Atmega2560 possède un registre particulier dénommé SREG (Status REGister) (cf. Figure 30).

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Figure 30: Structure du registre de statut SREG

Il intervient sur la gestion globale des interruptions, quelle qu'en soit la source. Afin d'autoriser la prise en compte d'une interruption, il est nécessaire que le bit I soit positionné à 1. Lorsqu'une interruption survient, ce bit I passe à 0. A la fin du traitement de l'interruption, il est impératif de le reconfigurer à 1, **sans modifier** la valeur des autres bits du registre SREG.

La gestion des interruptions obéit aux règles générales définies ci-dessous. Lors de la détection d'une interruption valide :

- L'instruction en cours est achevée ;
- Le bit I du registre SREG est positionné à 0, inhibant toute nouvelle prise en compte d'une autre interruption ;
- Le compteur ordinal est sauvegardé dans la pile (la sauvegarde des autres registres est du ressort de l'utilisateur, notamment pour le registre SREG) ;
- Le vecteur d'IT correspondant à la source de l'interruption est chargé dans le compteur ordinal ;
- La routine associée à l'interruption s'exécute ;

- Juste avant de sortir de la routine d'interruption, il faut de nouveau autoriser la prise en compte des interruptions ;
- La sortie de la routine d'interruption provoque la reprise du programme interrompu ;

Le microcontrôleur Atmega2560 possède une table des vecteurs d'interruptions où sont définis l'ensemble des interruptions qui sont gérées, en précisant la ressource et le type d'événement pouvant provoquer l'IT⁶ (cf. Figure 31 – extrait de la doc Atmel du Atmega2560). Parmi cet ensemble d'IT, nous allons nous focaliser sur l'IT associée au dépassement du compteur du Timer1 (Timer1 Overflow – vecteur IT 21).

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	PCINT0	Pin Change Interrupt Request 0
11	\$0014	PCINT1	Pin Change Interrupt Request 1
12	\$0016 ⁽³⁾	PCINT2	Pin Change Interrupt Request 2
13	\$0018	WDT	Watchdog Time-out Interrupt
14	\$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	\$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	\$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	\$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	\$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	\$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	\$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	\$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	\$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	\$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	\$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	\$0030	SPI, STC	SPI Serial Transfer Complete
26	\$0032	USART0 RX	USART0 Rx Complete
27	\$0034	USART0 UDRE	USART0 Data Register Empty

Figure 31: Extrait de la table des vecteurs d'interruptions Atmega2560

6.2.2. Ressources associées au Timer1

Le microcontrôleur possède 6 timers qui se distinguent par la taille du registre de comptage associé. Ainsi, nous avons :

- Timer0 et Timer2 : registre de comptage sur 8 bits. On désigne alors ces timers par des timers 8 bits.
- Timer1, Timer3, Timer4 et Timer5 : registre de comptage sur 16 bits. On désigne alors ces timers par des timers 16 bits.

Le mode de fonctionnement de chacun de ces timers est associé à un ensemble de registres qui leur sont propres. Par la suite, nous ne nous intéresserons qu'au Timer1, y compris en ce qui concerne la gestion des interruptions.

⁶ IT : InTeruption

La structure d'un timer 16 bits est décrite ci-dessous (cf. Figure 32) :

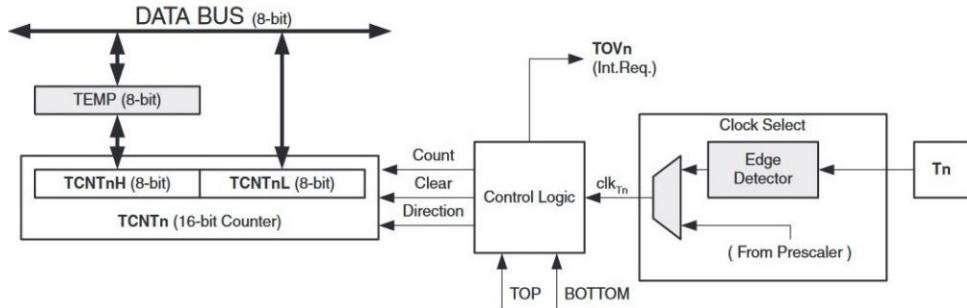


Figure 32: Structure générale Timer 16 bits

On distingue les éléments suivants :

- Le registre de comptage TCCNT1 (16 bits), accessible en lecture **et** écriture ;
- L'unité de contrôle logique, à laquelle sont associés les registres de contrôle TCCR1A, TCCR1B, et TCCR1C, sur 8 bits. Ces registres permettent de définir le mode de fonctionnement du Timer1, ainsi que la valeur de pré diviseur (prescaler) ;
- Des registres liés à la gestion des interruptions : TIFR1 and TIMSK1.

La fréquence de comptage est liée à la fréquence d'horloge du microcontrôleur, soit $F_{CPU} = 16MHz$ ou une période de base de $T_{CPU} = .6.28 \cdot 10^{-8}s$ (ou 62.5 ns).

Mode de fonctionnement du Timer1 : nous supposons par la suite que nous utiliserons le Timer1 :

- En mode normal (cf. Annexe)
- Valeur minimale du registre de comptage : 0x0000
- Valeur maximale du registre de comptage : 0xFFFF
- Le registre de comptage s'incrémentera d'une unité toutes les Clk_{T1} secondes. Cette valeur dépend de T_{CPU} et de la valeur du prescaler ;
- Prise en compte des interruptions sur débordement du registre de comptage du Timer1.

Registre TCCR1A

La structure du registre TCCR1A est disponible ci-dessous (cf. Figure 33) :

TCCR1A – Timer/Counter 1 Control Register A

Bit (0x80)	7	6	5	4	3	2	1	0
Read/Write	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10
Initial Value	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 33: Structure du registre TCCR1A

Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A

Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B

Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C

Bit 1:0 – WGM11:0: Waveform Generation Mode

Pour la configuration de ces bits, se référer à l'annexe, Table 17-3 et 17-2.

Registre TCCR1B

Sa structure est représentée ci-dessous (cf. Figure 34).

TCCR1B – Timer/Counter 1 Control Register B

Bit	7	6	5	4	3	2	1	0
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figure 34: Structure du registre TCCR1B

- Bit 7 – ICNC1: Input Capture Noise Canceled
- Bit 6 – ICES1: Input Capture Edge Select
- Bit 5 – Reserved Bit
- Bit 4:3 – WGM13:2: Waveform Generation Mode
- Bit 2:0 – CS12:0: Clock Select

Pour la configuration de ces bits, se référer à l'annexe, Table 17-6 et 17-2.
Les bits ICN1C et ICES1 seront initialisés avec la valeur 0.

Registre TCCR1C

Sa structure est définie ci-dessous (cf. Figure 35).

TCCR1C – Timer/Counter 1 Control Register C

Bit	7	6	5	4	3	2	1	0
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–
Read/Write	W	W	W	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

Figure 35: Structure du registre TCCR1C

- Bit 7 – FOCnA: Force Output Compare for Channel A
- Bit 6 – FOCnB: Force Output Compare for Channel B
- Bit 5 – FOCnC: Force Output Compare for Channel C

Dans le mode d'utilisation choisi pour le Timer1, l'ensemble des bits de ce registre seront initialisés à 0.

6.2.3. Ressources du Timer1 associées à la gestion des interruptions

Les deux registres associés au Timer1 pour la gestion des interruptions sont TIMSK1 (Timer/Counter1 Interrupt Mask Register) et TIFR1 (Timer/Counter1 Interrupt Flag Register). Leur structure respective est présentée ci-dessous (cf. Figure 36 et Figure 37).

TIMSK1 – Timer/Counter 1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figure 36: Structure registre TIMSK1

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
0x16 (0x36)	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figure 37: Structure du registre TIFR1

Au sein de ces deux registres, seuls les bits TOIE1 et TOV1 nous intéressent dans la gestion des interruptions sur Overflow.

[...] Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Overflow interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 101) is executed when the TOV1 Flag, located in TIFRn, is set [...] (cf. documentation Atmel).

[...] Bit 0 – TOV1: Timer/Counter1, Overflow Flag

The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 17-2 on page 145 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location [...] (cf. documentation Atmel).

6.2.4. Gestion de l'interruption Timer1 Overflow : principe de mise en œuvre

Cette gestion s'effectuera en deux étapes. La première consistera à initialiser les registres de configuration du Timer1. Cette initialisation sera effectuée une seule fois, au sein de la fonction *setup()*. La seconde consiste à écrire la routine d'interruption qui sera appelée lors du débordement du registre de comptage.

Initialisation

- Inhiber toutes les interruptions ;
- Initialiser le contenu du registre TCCR1A
- Initialiser le contenu du registre TCCR1B afin d'arrêter le compteur
- Initialiser le contenu du registre TCCR1C
- Initialiser le bit TOV1 du registre TIFR1 à 1
- Initialiser le bit TOEI1 à 1
- Initialiser la valeur du registre de comptage TCNT1
- Autoriser toutes les interruptions
- Initialiser le registre TCCR1B pour définir la valeur de prescaler et démarrer le compteur

Remarques :

- Pour inhiber et autoriser les interruptions de façon globale, vous disposez des fonctions :
 - o `interrupts()`
 - o `nointerrupts()`
- (cf. <https://www.arduino.cc/reference/en/language/functions/interrupts/interrupts/>)
 (cf. <https://www.arduino.cc/reference/en/language/functions/interrupts/nointerrupts/>)

Ecriture de la routine d'interruption

L'écriture de la routine d'interruption associée au débordement du registre ce comptage du Timer1 possède la structure ci-dessous :

```
ISR (TIMER1_OVF_vect)
{
    Bloquer les IT du Timer1 ;
    Sauver le contenu du registre SREG ; // Optionnel
    Code associé au traitement de l'IT ;
    {
        }

    Gestion du registre TIFR1 et du bit TOEV1 ;
    Initialiser le contenu du registre de comptage TCNT1 ; // Si besoin
    Autoriser les IT Timer : gestion du registre TIFR1 ;
    Restaurer le contenu du registre SREG ; // Optionnel
    Autoriser toutes les interruptions : gestion du contenu du registre SREG ;
}
```

Remarques :

1. L'exécution d'une routine d'interruption doit être aussi rapide que possible, afin de ne pas masquer la prise en compte d'autres interruptions. Les traitements réalisés seront donc « simples ».
2. Il est possible de modifier le contenu d'une variable globale au sein d'une routine d'interruption. Dans ce cas, il faut appliquer le modificateur *volatile* à la variable concernée.
3. L'appel aux méthodes *print()* et *println()* est prohibé, sous peine de « planter l'exécution » de la routine d'IT, voire du programme.
4. L'accès aux registres du microcontrôleur s'effectue de façon « classique ». En effet, les noms des registres du microcontrôleur sont reconnus comme des identifiants de variables. Par exemple, pour initialiser la valeur du registre de comptage du Timer1 avec la valeur entière ABCD exprimée en hexadécimal, l'instruction suivante est valide :

TCNT1 = 0xABCD ;

Initialisation de la valeur du registre de comptage

Le mode de fonctionnement du Timer1 sera le mode normal (cf. Annexe). Le registre de comptage est sur 16 bits. La période d'incrément du registre de comptage est Clk_{T1} . En l'absence de valeur de prescaler (valeur égale à 1), cette période de comptage correspond à $1/\text{F}_{\text{CPU}}$ ($\text{F}_{\text{CPU}} = 16\text{MHz}$). Si l'on définit une valeur de prescaler, alors $\text{Clk}_{T1} = \text{valeur prescaler} * 1/\text{F}_{\text{CPU}}$). On souhaite qu'une IT sur débordement du registre de comptage survienne toutes les T ms. L'événement provoquant l'IT survient lorsque le registre de comptage passe de la valeur 0xFFFF à 0X0000.

Question1 : combien faut-il cumuler de période de comptage Clk_{T1} pour atteindre la durée T ?
Question2 : en déduire alors la valeur d'initialisation du registre de comptage.

Répondre à ces deux questions pour $T = 1\text{s}$ et $T = 0.5\text{s}$

6.2.5. Travail à réaliser

A partir du code du programme *Blink* étudié précédemment, écrire le programme *Blink_ITTimer1*, implémentant les fonctionnalités décrites ci-dessous :

- Faire clignoter la led connectée sur le port D13, en utilisant la fonction `delay()`, pour la faire clignoter toutes les 3 secondes (allumer 3s et éteinte 3s);
- Faire clignoter une autre led, connectée sur le port numérique de votre choix, toutes les secondes (allumer 1s et éteinte 1s) puis réessayer (allumer 0.5s et éteinte 0.5s), en utilisant le mécanisme de gestion des IT du Timer1 sur débordement du registre de comptage.

7. Mise en œuvre du circuit d'horloge temps réel RTC

7.1. Format et codage des données

Le circuit d'horloge temps réel utilisé a été présenté dans la partie 3.3. L'ensemble des données manipulées est décrit au sein de la même partie 3.3 (cf. Figure 12). Elles sont organisées sous la forme de registres de 8 bits, d'adresse adjacente. Le premier registre, celui des secondes, possède l'adresse \$00. Seuls les registres d'adresse \$00 à \$06 nous intéressent. Le contenu de chaque registre est codé en « Décimal Codé Binaire » ou DCB. Ainsi, les chiffres des unités et des dizaines sont codés en binaire sur 4 bits. Par exemple, si le contenu du registre des minutes contient 47 (minutes) :

Unité : 7mn soit 0111 en binaire

Dizaine : 4 dizaine de minutes, soit en binaire 0100

et donc le contenu du registre des minutes du circuit DS1307 sera : 0100 0111.

Le registre des secondes d'adresse \$00 contient :

- Pour les 7 premiers bits, le codage BCD de la valeur des secondes.
- Le bit de poids le plus fort (bit7)(CH) permet d'arrêter ou de lancer l'horloge.

Le registre des heures d'adresse \$02 contient :

- Bit6 : choix mode 12 ou 24h. Positionné à 0, le mode 24h est sélectionné.
- Bit5 :0 : en mode 24h, il représente la valeur des heures.

Le registre d'adresse \$03 contient le jour de la semaine.

- Seuls les bits 0:2 sont utilisés.
- Le lundi correspondant à la valeur 1.
- Le mardi correspondant à la valeur 2...
- Le dimanche correspondant à la valeur 7.

Le registre d'adresse \$06 représente l'année. Seules les unités et les dizaines sont codées, de 00 à 99. Ainsi, l'année 2015 sera codée 15.

Par la suite, nous adopterons le mode 24h pour le fonctionnement de l'horloge.

7.2. Choix types de données et fonctions de manipulation de l'horloge RTC

7.2.1. Types de données

Afin de faciliter par la suite la manipulation des données concernant la date et l'heure, il faut au préalable créer des types de données sous la forme de définition de type. Pour cela, on considère les regroupements suivants :

Type associé à l'heure : constitué des champs suivants.

 seconde ;
 minute ;

heure ;

Type associé à la date : constitué des champs suivants.

```
jour_semaine; // pour {lundi (1),...,dimanche(7)}  
jour_mois; // {1, ...,31}  
mois; // {1, ...,12}  
annee ; // {00, ..., 99}
```

Type associé à l'horloge RTC : constitué des champs suivants.

Type associé à l'heure
Type associé à la date

En fonction de l'étendue de la valeur de chaque champ, et de la définition des types de base disponibles sous environnement Arduino (cf. <https://www.arduino.cc/reference/en/#variables>) :

- Pour chacun des champs, choisissez le type de donnée le mieux adapté ;
- Au sein du fichier d'en-tête *RTC_DS1307.h*, implémenter les définitions de types ci-dessus.

7.2.2. Fonctions de manipulation du circuit d'horloge RTC

Deux fonctions principales sont à considérer :

- Initialiser la date et l'heure du circuit RTC ;
- Récupérer la date et l'heure du circuit RTC ;
- En plus d'une 3^{ème} qui consistera à afficher sur le terminal série la date et l'heure sous la forme :

Lundi 5 octobre 2020 17 : 52 : 14

En plus de ces trois fonctions, il faut en prévoir deux autres qui permettront :

- La conversion « décimal vers bcd » ;
- La conversion « bcd vers décimal ».

Pour l'ensemble des fonctions de manipulation de l'horloge RTC, vous trouverez ci-dessous leurs spécifications :

- Initialiser la date et l'heure du circuit RTC :
 - o Identificateur : *setDateDs1307*
 - o Paramètres d'entrée : Type associé à l'horloge RTC
 - o Valeur renournée : aucune
- Récupérer la date et l'heure :
 - o Identificateur : *getDateDs1307*
 - o Paramètres d'entrée : aucun
 - o Valeur renournée : Type associé à l'horloge RTC
- Afficher la date et l'heure sur le terminal série :
 - o Identificateur : *Affiche_date_heure*
 - o Paramètres d'entrée : Type associé à l'horloge RTC
 - o Valeur renournée : aucune

7.2.3. Gestion de la communication avec le circuit RTC DS1307

Le circuit de l'horloge RTC gère l'échange des données sur le bus I2C. La communication sur bus I2C et de type « maître – esclave ». La carte Arduino joue le rôle de maître et le circuit RTC celui d'esclave.

Sous l'IDE Arduino, nous disposons de la librairie Wire qui facilite la gestion de la communication sur le bus I2C.(cf. <https://www.arduino.cc/en/Reference/Wire>).

L'utilisation de cette librairie nécessite d'inclure le fichier *Wire.h* à l'aide de la directive :

```
#include <Wire.h>
```

La seconde étape permet de définir la carte Arduino en tant que maître du bus en utilisant la méthode *begin()* de la librairie *Wire*. L'appel à cette méthode sera effectué une seule fois, au sein de la fonction *setup()*. Lorsque cette initialisation est réalisée, alors la communication sur le bus I2C peut démarrer.

Le dialogue avec un périphérique I2C débute toujours par un début de transmission. Ce début de dialogue entre la carte Arduino et le périphérique s'effectue à l'aide de la méthode *beginTransmission()* (cf. <https://www.arduino.cc/en/Reference/WireBeginTransmission>). Puis l'on se positionne sur le premier registre du circuit DS1307 avec la méthode *write()* (cf. <https://www.arduino.cc/en/Reference/WireWrite>). A partir de cet instant, 2 scenarii possibles :

- Acquisition de la date et de l'heure :
 - o Fermer la connexion avec le périphérique I2C esclave en utilisant la méthode *endTransmission()* (cf. <https://www.arduino.cc/en/Reference/WireEndTransmission>) ;
 - o La carte Arduino « maître » émet une requête vers le circuit RTC afin de lire le contenu des registres, en utilisant la méthode *requestFrom()* (cf. <https://www.arduino.cc/en/Reference/WireRequestFrom>).
 - o Il suffit ensuite de lire le contenu des registres en utilisant la méthode *read()* (cf. <https://www.arduino.cc/en/Reference/WireRead>).
- Mise à jour de la date et de l'heure :
 - o Ecrire successivement le contenu de chacun des registres en utilisant la méthode *write()*.
 - o Terminer avec *endTransmission()*.

Depuis ces éléments, finaliser l'écriture des fonctions décrites à la section 7.2.2 au sein du programme Arduino *RTC_DS1307*. Afin de vous aider, vous disposer d'une bibliothèque pour le circuit DS1307, disponible sous Celene. A partir des fichiers .cpp et .h, extraire les portions de code qui vous permettront d'implémenter les fonctions souhaitées.

7.2.4. Fonction complémentaire de gestion de calendrier

Lors de l'initialisation des registres du circuit RTC DS1307, il peut être utile de pouvoir déterminer le jour de la semaine (lundi, mardi, ..., dimanche) en fonction de la date, du mois et de l'année. Une méthode est proposée et décrite via :

https://fr.wikibooks.org/wiki/Curiosit%C3%A9s_math%C3%A9matiques/Trouver_le_jour_de_la_semaine_avec_une_date_donn%C3%A9e

On pourra se référer à la méthode 2.

Implémenter alors cette méthode et vérifier que le résultat retourné est compatible avec les conventions du circuit DS1307 pour la gestion du jour de la semaine. Effectuer les adaptations nécessaires si besoin.

8. Mise en œuvre du module GPS

Le module GPS utilisé a été présenté dans la partie § 3.5. Il permet de réceptionner des trames NMEA, qui contiennent des informations relatives à la position géographique, la date ou l'heure... La mise en œuvre de ce module GPS doit permettre de répondre aux besoins ci-dessous :

- Permettre le recalage de la date et de l'heure du module d'horloge temps réel de façon automatique, lorsque les données issues du module GPS sont valides ;
- Permettre à l'utilisateur d'afficher un ensemble de données telles que :
 - o Heure UTC ;
 - o Date ;
 - o Latitude ;
 - o Longitude ;
 - o Altitude.
- En option, permettre de gérer la date et l'heure selon différents fuseaux horaires.

8.1. Données fournies par le module GPS : messages NMEA

Le module GPS envoie périodiquement sur la liaison série des trames ou messages NMEA. Une trame ou message NMEA est composé de caractères ASCII qui forment une chaîne de caractères. Leur format répond aux règles générales suivantes de formatage :

- Le début du message est défini par le caractère '\$' ;
- La fin du message est toujours définie par les caractères <CR><LF> ;
- Chaque message est formé de champs, composés de caractères ASCII, séparés par une virgule ;
- Après le dernier champ, et avant les caractères <CR><LF> se trouve le résultat du calcul d'un checksum composé du caractère '*' suivi de 2 caractères ASCII qui représentent sa valeur. La valeur du checksum est calculée en considérant l'ensemble des caractères situés entre le caractère '\$' et '*' et en effectuant un 'OU exclusif' de tous les caractères.

Le module GPS peut délivrer plusieurs types de messages. Parmi ceux-ci, nous utiliserons les messages \$GPGGA et \$GPRMC. La trame \$GPRMC sera utilisée de façon standard pour le recalage des données de date et heure, la gestion de l'heure Eté – Hiver et la gestion des fuseaux horaires. Le message \$GPGGA sera lui utilisé pour accéder à des données complémentaires, non disponibles dans le message \$GPRMC.

8.1.1. Format du message \$GPRMC

Le format d'un message \$GPRMC est illustré ci-dessous :

\$GPRMC,093415.000,A,4721.8924,N,00041.1105,E,0.42,318.99,100517,,D*68

L'analyse des différents champs est disponible dans le tableau ci-dessous (cf. Tableau 2).

Id champs	Nom	Exemple	Unité	Description
0	ID message	\$GPRMC		
1	UTC time	093415.000		hhmmss.000
2	Statut	A		A = data valid ou V=data not valid
3	Latitude	4721.8924		ddmm.mm
4	Indicateur latitude N/S	N		N = Nord ou S = South
5	Longitude	00041.1105		ddmm.mm

6	Indicateur longitude E/W	E		E = East ou W = West
7	Speed over ground	0.42	knots	
8	Course over ground	318.99	degrés	
9	Date	100517		ddmmyy
10	Magnetic variation		degrés	
11	Est/West indicator			
12	Mode	D		A = Autonomous D = DGPS
13	Checksum	68		
14	<CR><LF>			Terminaison de fin de message

Tableau 2 : Exemple de message \$GPRMC

Parmi les éléments de ce message NMEA nous n'en exploiterons que quelques uns, listés ci-dessous.

- ID message : \$GPRMC
- UTC time⁷ : 093415 pour 09h 34 minute et 15 secondes
- Statut : A pour signifier que les données du message sont valides
- Latitude : 47°21' et 0.8924' N
- Longitude : 00°41' et 0.1105' E
- Date : 10.05.2017

8.1.2. Format du message \$GPGGA

Le format d'un message \$GPGGA est illustré ci-dessous (cf. Tableau 3):

\$GPGGA,093415.000,4721.8924,N,00041.1105,E,2,08,1.03,57.6,M,47.9,M,0000,0000*56

Id champs	Nom	Exemple	Unité	Description
0	ID message	\$GPGGA		
1	UTC time	093415.000		hhmmss.000
2	Latitude	4721.8924		ddmm.mm
3	Indicateur latitude N/S	N		N = Nord ou S = South
4	Longitude	00041.1105		ddmm.mm
5	Indicateur longitude E/W	E		E = East ou W = West
6	Position fix indicator	2		0 : fix not available 1 : GPS SPS mode, fix valid 2 : Differential GPS, SPS mode, fix valid
7	Satellites used	08		
8	HDOP	1.03		Horizontal Dilution of Precision
9	MSL altitude	57.6	mètres	

⁷ Universel Temps Coordonné : échelle de temps adoptée comme base du temps civil international.

10	Units	M	mètres	
11	Geoid separation	47.9	mètres	Geoid to ellipsoid separation Ellipsoid altitude = MSL altitude + geoid separation
12	Units	M	mètres	
13	Age of Diff. Corr.	0000	sec	Null field when DGPS is not used
14	Diff. Ref. Station ID	0000		
15	Cheksum	*56		
16	<CR><LF>			Terminaison de fin de message

Tableau 3: Exemple de message \$GPGGA

Ce type de message nous sera utile pour récupérer l'altitude du lieu par l'intermédiaire du champ MSL altitude, exprimé en mètres.

8.2. Mode de fonctionnement du module GPS

Le module GPS communique via une liaison série. Par défaut, la vitesse de communication est configurée à 9600 Bauds. Dès sa mise sous tension, le module envoie des messages, valides ou non.

Le second point concerne les messages NMEA délivrés par le module. Par défaut, le module délivre les messages de type \$GPGLL, \$GPRMC, \$GPVTG, \$GPGGA, \$GPGSA, \$GPGSV et \$GPZDA, avec une périodicité d'une seconde. Il est possible de paramétriser le module GPS afin de ne recevoir qu'un certain type de trame, selon une périodicité donnée. Pour cela, il faut transmettre au module GPS une commande dite « PMTK », spécifique au module GPS utilisé. Par la suite, nous utiliserons de façon principale les messages \$GPRMC, avec une périodicité d'une seconde, et de façon secondaire, le message \$GPGGA.

Pour n'utiliser que le message de type \$GPRMC, avec une périodicité d'une position, il faudra envoyer la commande suivante au module GPS :

Pour n'utiliser que le message de type \$GPGGA, avec une périodicité d'une position, il faudra envoyer la commande suivante au module GPS :

Dans tous les cas, le module GPS renvoie un accusé de réception dont le format est spécifié ci-dessous :

"\$PMTK001,Cmd,Flag*Checksum<CR><LF>"

Le champ Cmd désigne le numéro de message PMTK (soit ici 314).

Le champ Flag permet d'avoir un retour sur la prise en compte de la commande :

Flag = 0 commande non valide

Flag = 1 commande non supportée

Flag = 1 commande non supportée
Flag = 2 Commande valide, mais l'action demandée a échoué

Flag = 3 Commande valide et action demandée exécutée correctement

8.3 Mise en œuvre

Le module GPS communique via une liaison série. Par défaut, il est paramétré pour fonctionner à la vitesse de 9600 Bauds. Il sera connecté sur la carte Arduino Mega2560 au port série 1.

Un seul type de message NMEA est transmis par le module GPS, soit des messages de type \$GPRMC ou bien de type \$GPGGA. Par défaut, ce seront les messages de type \$GPRMC qui seront transmis.

Dès que le module GPS est sous tension, il renvoie des messages NMEA, qu'ils soient valides ou non.

Du point de vue fonctionnel, la gestion du module GPS nécessitera l'usage de trois fonctions de base :

- Effectuer l'acquisition d'un message NMEA ;
- Parser un message NMEA valide, afin d'en extraire les champs à utiliser par la suite ;
- Savoir si le module GPS est synchronisé, c.à.d. qu'il renvoie des messages valides.

8.3.1. Acquisition d'un message NMEA

L'acquisition d'un message NMEA répond aux impératifs définis ci-dessous :

- Le message NMEA sera stocké dans un buffer, sous la forme d'une chaîne de caractères ;
- L'acquisition du message NMEA débute lorsque le caractère '\$' est détecté et s'achève lorsque le caractère '*' est détecté.
- Le temps d'acquisition du caractère '\$' de début de message NMEA sera limité dans le temps. Lorsque ce temps sera dépassé, alors la procédure d'acquisition du message devra être interrompue.

Ecrire la fonction *GetGPS_Msg()* qui permet de stocker dans un buffer un message NMEA. Vérifier que les messages reçus sont conformes en les affichant sur le terminal série.

Remarque : La synchronisation du module GPS n'est pas immédiate. En fonction des conditions de réception du signal GPS, la réception de messages NMEA valides peut prendre jusqu'à plusieurs minutes. Soyez patient(es).

8.3.2. Parser un message NMEA

Parser un message NMEA consiste à récupérer sous forme de chaîne de caractères l'ensemble des champs du message. La fonction qui permettra de parser un message NMEA doit pouvoir prendre en compte aussi bien les messages \$GPRMC et \$GPGGA.

Paramètres d'entrée : le message NMEA à parser ;

Paramètres de sortie : l'ensemble des champs du message sous forme de chaîne de caractères.

Exemple :

Message NMEA : \$GPGGA,132428.000,4721.8640,N,00041.1144,E,1,7,1.24,59.7,M,47.9,M,*,*

Sorties :

- "\$GPGGA"
- "132428.000"
- "4721.8640"
- "N"
- "00041.1144"
- "E"
- "1"
- "7"
- "1.24"
- "59.7"
- "M"
- "47.9"

- "M"
- ""
- "*"

Ecrire la fonction *GPS_msg_parse()* qui à partir d'un message NMEA de type \$GPRMC ou \$GPGGA permet de récupérer l'ensemble des champs sous forme de chaîne de caractères.

8.3.3. Savoir si le module GPS renvoie des trames valides

Lorsque le module GPS commence à être alimenté, il peut mettre un certain temps avant de retourner des trames dont le contenu est valide. Avant d'exploiter le contenu des champs d'un message NMEA, il faut donc vérifier que le module GPS renvoie des trames valides. Cette information est disponible dans chaque message dans le champ *Position fix indicator* ou *Statut* (cf. section 8.1).

Ecrire la fonction *Test_Synchro_GPS()* qui permet de savoir si un message de type \$GPGGA ou \$GPRMC est valide, et donc de savoir si le module GPS est synchronisé. Cette fonction retournera un booléen.

8.3.4. Choisir le type de message NMEA renvoyé par le module GPS

Afin de faciliter le choix du type de message que le module GPS doit renvoyer, écrire la fonction *Choix_Msg_NMEA()* telle que :

- Paramètre d'entrée : choix du message \$GPGGA ou \$GPRMC, type à définir.
- Valeur renournée : aucune

8.3.5. Validation des fonctions associées à l'utilisation du module GPS

A partir du squelette de programme disponible, le compléter afin de valider les fonctionnalités des quatre fonctions décrites ci-dessus. Vous devrez définir les tests unitaires que vous mettrez en place et en fournir le résultat.

9. Mise en œuvre du capteur BME680

Le capteur BME680 de Bosch va nous permettre d'effectuer l'acquisition des grandeurs suivantes :

- Mesure de température en °C
- Mesure du taux relatif d'humidité en %
- Mesure de la pression atmosphérique en hPa
- Mesure d'un Index de Qualité de l'AIR (IAQ)
- Indicateur de confiance sur la mesure de l'IAQ

Des mesures complémentaires pourront être obtenues comme :

- Concentration des COV – Composants Organo Volatils
- Concentration en CO2

Sa mise en œuvre reposera sur des bibliothèques logicielles fournies par Bosch, en particulier pour la mesure de l'IAQ, mais aussi pour les autres mesures usuelles, ainsi que la mesure des COV et du CO2.

9.1. Configuration de l'environnement de programmation

De par la complexité du capteur BME680, sa mise en œuvre s'effectuera en utilisant les bibliothèques logicielles fournies par Bosch. Pour cela, nous nous appuierons sur l'environnement logiciel BSEC – Bosch Software Environmental Cluster. Cet environnement est disponible sous la forme de l'archive *BSEC-Arduino-library-master* disponible depuis à partir de <https://github.com/BoschSensortec/BSEC-Arduino-library>⁸.

Avant de récupérer des mesures depuis le capteur, il est impératif de configurer l'environnement de programmation Arduino. Cette configuration spécifique est due à l'utilisation de la librairie précompilée. Les étapes érites ci-dessous supposent que l'IDE Arduino a été installé en utilisant l'installateur Windows.

Il est également possible de se référer au guide d'intégration (fichier Pdf) téléchargeable à l'adresse :

https://community.bosch-sensortec.com/varuj77995/attachments/varuj77995/bst_community-mems-forum/2753/1/Integration%20Guidelines%20for%20Arduino%20platforms.pdf

La procédure décrite, bien que spécifique pour l'IDE Arduino 1.8.5 est également valide pour la version 1.8.8. Les points abordés ci-dessous vous permettront de configurer votre environnement sur une machine personnelle.

9.1.1. Téléchargement de l'archive spécifique pour environnement Arduino

Depuis le lien fourni ci-dessus (cf. note8), il est possible de télécharger l'archive de la librairie au format .zip (cf. Figure 38).

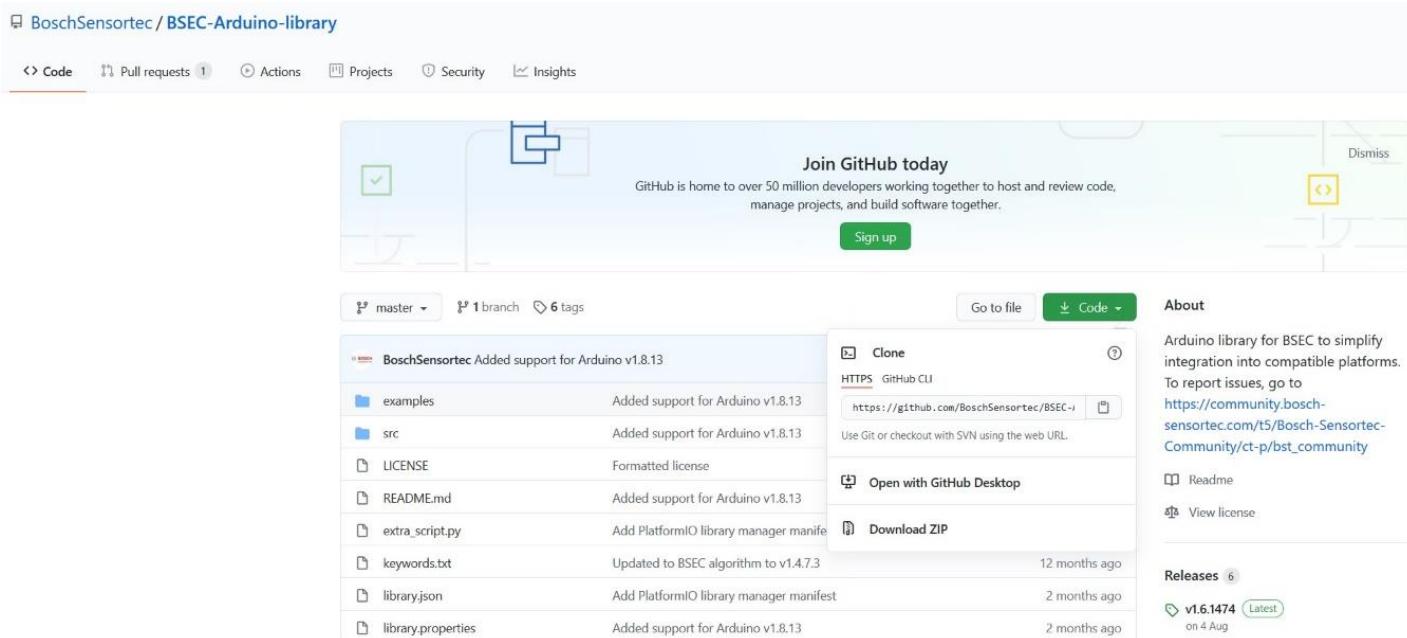


Figure 38: Téléchargement de l'archive la librairie BSEC

Une fois l'archive décompressée, il faut placer l'ensemble des répertoires et fichiers au sein du répertoire *bsec*. Au sein du répertoire *src\atmega2560* se trouve la librairie précompilée *libalgobsec.a*. Il s'agit d'une librairie statique, qui contient du code objet spécifique en vue d'être exécuté sur le microcontrôleur Atmega2560.

⁸ Cette archive est spécifique à une utilisation du BME680 dans l'environnement Arduino, pour la cible Atmega2560.

9.1.2. Installation des librairies BSEC

L’installation de la librairie BSEC sous l’IDE Arduino s’effectue en copiant le contenu du répertoire bsec dans le répertoire *Arduino\libraries*. Si l’IDE était ouvert, alors il faut le fermer puis le relancer afin de prendre en compte l’ajout de la bibliothèque.

9.1.3. Remplacer Arduino-builder

La génération du code exécutable ainsi que le téléchargement du code dans la mémoire de programme du microcontrôleur sont gérés par l’utilitaire *arduino-builder*. Cet utilitaire se situe dans le répertoire d’installation de l’IDE (cf. Figure 19). Dans sa version de base, *arduino-builder* ne gère pas l’utilisation de librairie précompilée. Il faut donc le remplacer par une autre version, *arduino-builder-PR219*. Le lien de téléchargement est fourni ci-après :

downloads.arduino.cc/PR/arduino-builder/arduino-builder-219.zip

Il permet de télécharger l’archive *arduino-builder-219.zip*. Après l’avoir décompressé, sélectionner l’application *arduino-builder* selon l’OS installé sur la machine (cf. Figure 39) ,

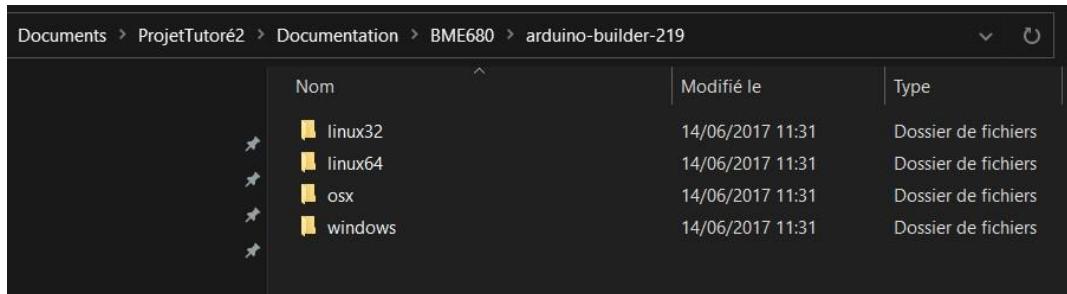


Figure 39: Choix de *arduino-builder* selon l’OS de la machine cible

et remplacer celui présent lors de l’installation par cette nouvelle instance.

9.1.4.Modifier les options de compilation

La génération du code exécutable s’appuie sur un fichier texte qui définit :

- L’ensemble des étapes de compilation du code source, ainsi que les options de compilation ;
- L’ensemble des étapes d’édition de liens, avec les options rattachées.

Toutes ces spécifications sont regroupées au sein du fichier *platform.txt*, situé dans le répertoire d’installation de l’IDE Arduino et plus précisément *Arduino\hardware\arduino\avr* (cf. Figure 40).

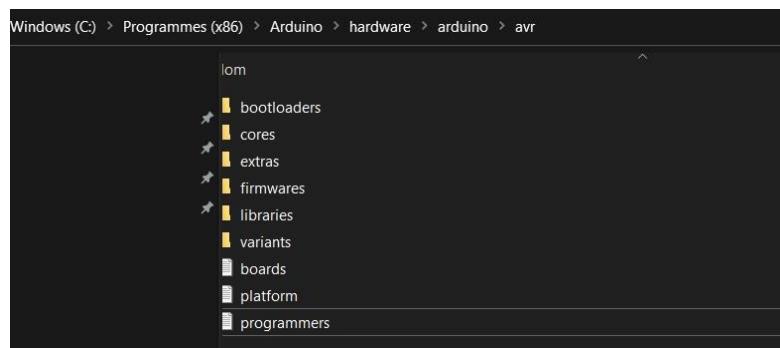


Figure 40: Emplacement du fichier *platform.txt*

Il faut modifier la section ## *Combine gc-sections, archives, and objects* et placer l'option {compiler.c.elf.extra-flags} plutôt vers la fin. Ainsi, la version initiale :

```
## Combine gc-sections, archives, and objects
recipe.c.combine.pattern="{compiler.path}{compiler.c.elf.cmd}"      {compiler.c.elf.flags}      -
mmcu={build.mcu}  {compiler.c.elf.extra_flags} -o  "{build.path}/{build.project_name}.elf"
{object_files} "{build.path}/{archive_file}" "-L{build.path}" -lm
```

deviendra :

```
## Combine gc-sections, archives, and objects
recipe.c.combine.pattern="{compiler.path}{compiler.c.elf.cmd}"      {compiler.c.elf.flags}      -
mmcu={build.mcu}      -o      "{build.path}/{build.project_name}.elf"      {object_files}
{compiler.c.elf.extra_flags} "{build.path}/{archive_file}" "-L{build.path}" -lm
```

Remarques : En ce qui concerne la mise en œuvre sur les machines de l'école, la machine virtuelle dont vous disposez intègre déjà ces modifications, sauf pour la gestion de la librairie *bsec*.

9.2. Première mise en œuvre et test du BME680

Des exemples de programmes de mise en œuvre du capteur BME680 sont disponibles au sein du répertoire *bsec\examples* créé lors de l'installation de la librairie *bsec*. Nous allons nous intéresser au programme *basic* qui offre toutes les fonctionnalités de base pour effectuer par la suite l'acquisition des mesures à réaliser. Les copies d'écran ci-dessous n'en sont qu'une version modifiée destinées à expliciter les principaux éléments.

9.2.1. Fonctions additionnelles

Vous disposerez des deux fonctions *checkIaqSensorStatus()* (cf. Figure 41) et *errLeds()* (cf. Figure 42) qui permettent respectivement :

- De vérifier l'état du capteur et de renvoyer un code d'erreur si nécessaire sur le terminal série;
 - o En cas d'erreur, la led connectée sur le port 13 clignote, dans le cas contraire, elle reste allumée.
- De faire clignoter la led connectée sur le port 13.

Ces deux fonctions sont définies via leur prototype.

9.2.2. Initialisation des ressources et fonction *setup()*

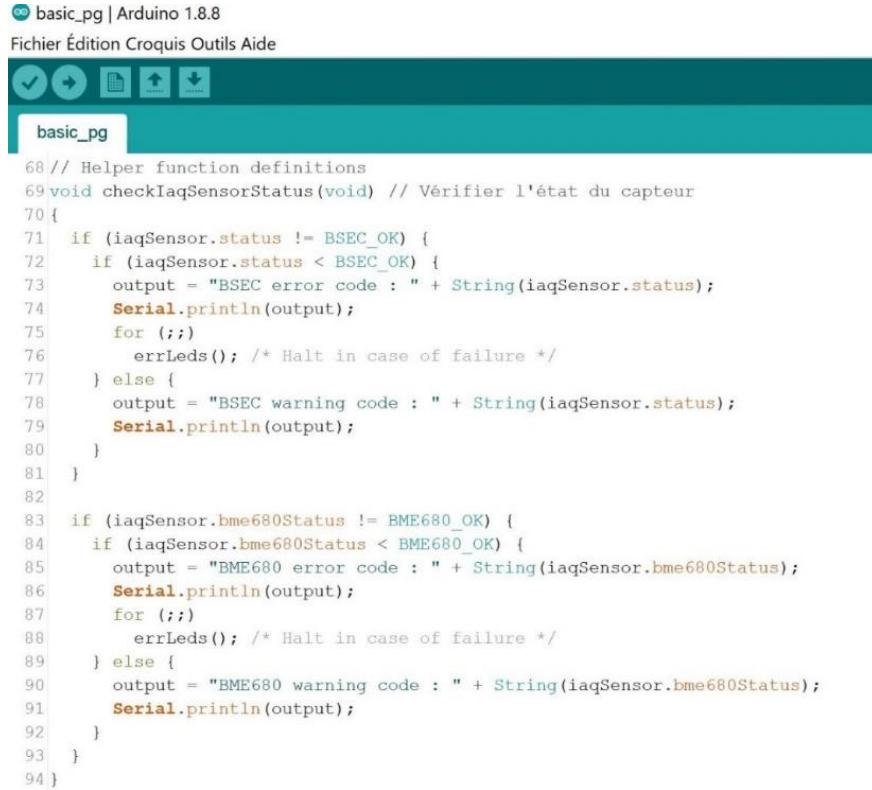
Avant d'utiliser le capteur, il est impératif de créer une instance de l'objet de type *Bsec*. Cette instance est *iaqSensor*. Elle représente l'objet capteur BME680. A partir de cette instance *iaqSensor*, il sera possible ensuite d'appeler les méthodes associées à la gestion du capteur (cf. Figure 43).

L'initialisation du mode de fonctionnement du capteur peut alors débuter au sein de la fonction *setup()*.(cf. Figure 44).

- A la ligne 17, la méthode *begin()* permet de spécifier l'adresse I2C du capteur utilisé et d'initialiser la communication I2C entre le microcontrôleur et le capteur.
- Les lignes 18-21 permettent d'afficher la version de la librairie utilisée et de vérifier l'état du capteur.

Les lignes 23-37 permettent de gérer un capteur virtuel, en spécifiant la liste des services qui lui sont associés, puis en souscrivant aux services déclarés. Cette souscription permet de définir le mode de

fonctionnement du capteur, par la constante *BSEC_SAMPLE_RATE_LP*. Ainsi, le capteur fonctionnera en mode basse puissance (Low Power). Ce mode particulier permettra d'obtenir une mesure toutes les 3 secondes. A ce stade, le capteur est prêt à être utilisé (cf. Figure 44).



```

basic_pg | Arduino 1.8.8
Fichier Édition Croquis Outils Aide
basic_pg
68 // Helper function definitions
69 void checkIaqSensorStatus(void) // Vérifier l'état du capteur
70 {
71   if (iaqSensor.status != BSEC_OK) {
72     if (iaqSensor.status < BSEC_OK) {
73       output = "BSEC error code : " + String(iaqSensor.status);
74       Serial.println(output);
75       for (;;)
76         errLeds(); /* Halt in case of failure */
77     } else {
78       output = "BSEC warning code : " + String(iaqSensor.status);
79       Serial.println(output);
80     }
81   }
82
83   if (iaqSensor.bme680Status != BME680_OK) {
84     if (iaqSensor.bme680Status < BME680_OK) {
85       output = "BME680 error code : " + String(iaqSensor.bme680Status);
86       Serial.println(output);
87       for (;;)
88         errLeds(); /* Halt in case of failure */
89     } else {
90       output = "BME680 warning code : " + String(iaqSensor.bme680Status);
91       Serial.println(output);
92     }
93   }
94 }
```

Figure 41: Mise en œuvre du capteur BME680 - Check capteur

```

96 void errLeds(void) // Gestion de la led sur port 13 en cas d'erreur capteur BME680
97 {
98   pinMode(LED_BUILTIN, OUTPUT);
99   digitalWrite(LED_BUILTIN, HIGH);
100  delay(100);
101  digitalWrite(LED_BUILTIN, LOW);
102  delay(100);
103 }
```

Figure 42: Gestion Led en cas d'erreur sur capteur BME680

```

1 #include "bsec.h"
2
3 // Fonctions prototypes
4 void checkIaqSensorStatus(void);
5 void errLeds(void);
6
7 // Créer l'objet iaqSensor de la classe Bsec
8 Bsec iaqSensor;
9
10 String output;
11
12 void setup(void)
13 {
14   Serial.begin(115200);
15   Wire.begin();
16
17   iaqSensor.begin(BME680_I2C_ADDR_PRIMARY, Wire); // Associer iaqSensor avec l'I2C du capteur et le bus I2C
18   // Afficher la version de la librairie Bosch utilisée
19   output = "\nBSEC library version " + String(iaqSensor.version.major) + "." + String(iaqSensor.version.minor) + "." + String(iaqSensor.version.major_bugfix) + "." + String(iaqSensor.version.minor_bugfix);
20   Serial.println(output);
21   checkIaqSensorStatus(); // Vérifier l'état du capteur
```

Figure 43: Instance du capteur et version librairie BSEC

```

23 // Créer la liste des services
24 BSEC_VIRTUAL_SENSOR_T sensorList[10] = {
25   BSEC_OUTPUT_RAW_TEMPERATURE,
26   BSEC_OUTPUT_RAW_PRESSURE,
27   BSEC_OUTPUT_RAW_HUMIDITY,
28   BSEC_OUTPUT_RAW_GAS,
29   BSEC_OUTPUT_IAQ,
30   BSEC_OUTPUT_STATIC_IAQ,
31   BSEC_OUTPUT_CO2_EQUIVALENT,
32   BSEC_OUTPUT_BREATH_VOC_EQUIVALENT,
33   BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,
34   BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY,
35 };
36 // S'abonner à la liste des services + configuration du mode de fonctionnement du capteur
37 iaqSensor.updateSubscription(sensorList, 10, BSEC_SAMPLE_RATE_LP);
38 // Vérifier l'état du capteur
39 checkIaqSensorStatus();
40
41 // Print the header : en tête des données acquises dans la boucle loop()
42 output = "Timestamp [ms], raw temperature [°C], pressure [hPa], raw relative humidity [%], gas [Ohm], IAQ, IAQ accuracy, temperature [°C], relative humidity [%], Static IAQ, CO2 equivalent, breath VOC equivalent";
43 Serial.println(output);

```

Figure 44: Mode de fonctionnement du capteur BME680

9.2.3. Acquisition de mesures depuis le capteur BME680

L’acquisition des mesures depuis le capteur BME680 s’effectue en continu au sein de la fonction *loop()* (cf. Figure 45). Les données recueillies sont ensuite affichées dans le terminal série. Il est alors possible de vérifier que l’acquisition des données s’effectue bien toutes les 3 secondes.

```

46 void loop(void)
47 {
48   unsigned long time_trigger = millis();
49   if (iaqSensor.run()) { // If new data is available
50     output = String(time_trigger);
51     output += ", " + String(iaqSensor.rawTemperature);
52     output += ", " + String(iaqSensor.pressure);
53     output += ", " + String(iaqSensor.rawHumidity);
54     output += ", " + String(iaqSensor.gasResistance);
55     output += ", " + String(iaqSensor.iaq);
56     output += ", " + String(iaqSensor.iaqAccuracy);
57     output += ", " + String(iaqSensor.temperature);
58     output += ", " + String(iaqSensor.humidity);
59     output += ", " + String(iaqSensor.staticIaq);
60     output += ", " + String(iaqSensor.co2Equivalent);
61     output += ", " + String(iaqSensor.breathVocEquivalent);
62     Serial.println(output);
63   } else {
64     checkIaqSensorStatus();
65   }
66 }

```

Figure 45: Acquisition des mesures depuis capteur BME680

Par la suite, nous ne voulons mesurer que les grandeurs suivantes :

- Température : cf. ligne 51 ;
- Pression atmosphérique : cf. ligne 52 ;
- Taux d’humidité : cf. ligne 58 ;
- Mesure de l’IAQ : cf. ligne 55 ;
- Mesure de la précision de l’IAQ : cf. ligne 56 ;
- Mesure taux de CO2 : cf. ligne 60 ;
- Mesure taux de COV : cf. ligne 61.

Le tableau ci-dessous précise les unités de mesures et l’intervalle de valeurs pour chaque mesure effectuée (cf. Tableau 4).

A partir du code du programme *basic* , en extraire les éléments utiles et compléter le squelette de programme fourni pour n’afficher que les grandeurs ci-dessous au sein du terminal série. Vous

pourrez valider le bon fonctionnement du capteur en affichant sur le terminal série les mesures effectuées.

	Unité	Plage	Format
Température	°C	[-40.0, ,85.0]	Réel
Pression atmosphérique	hPa	[300.00, 1100.00]	Réel
Taux humidité	%	[0.0, ,100.0]	Réel
IAQ	-----	[0.0, ,500.0]	Réel
IAQ_Accuracy	-----	{0, ,3}	Entier
CO2	ppm		Réel
COV	ppm		Réel

Tableau 4: : Format et unités des mesures du capteur BME680

10.Gestion de la mise à jour de la date et de l'heure

La mise à jour de la date et de l'heure du circuit d'horloge RTC va s'effectuer en utilisant les données de date et heure fournies par le module GPS, au sein d'un message de type \$GPRMC. Cette mise à jour ne peut s'effectuer que lorsque le module GPS est synchronisé. On distingue 2 cas où la mise à jour est réalisée :

1. Au démarrage de l'application, la date et l'heure sont initialisées au vendredi 1 janvier 2010, 00 : 00 :00. Dès que le module GPS est synchronisé, alors on effectue la mise à jour de l'horloge RTC à l'heure et à la date courantes. Si la mise à jour échoue, alors, il faut recommencer.
2. De façon périodique, le circuit d'horloge RTC doit être recalé sur l'heure (et aussi la date), en raison d'une dérive liée au manque de stabilité dans le temps du circuit RTC. La périodicité T_{MaJ_RTC} retenue est de l'ordre de quelques heures (toutes les 4 à 6 heures semble un bon compromis). A partir du lancement de l'application, lorsque la durée T_{MaJ_RTC} s'est écoulée, alors la procédure de mise à jour de date et heure de l'horloge doit s'effectuer. Si elle échoue, alors, il faut recommencer.

Dans tous les cas, le rafraîchissement de l'affichage de l'heure et de la date n'est pas interrompu.



10.1.1. Heure UTC et heure locale

Le module GPS fournit l'heure UTC – Universel Temps Coordonné⁹, mais aussi la date par rapport à cette référence horaire. Le passage de l'heure UTC à l'heure locale s'effectue en prenant en compte simultanément deux corrections :

1. La correction liée au fuseau horaire, UTC+X ou UTC-Y, où X et Y représentent le décalage horaire par rapport à l'heure UTC. Le décalage horaire est de la forme hh : mm.
2. La correction horaire liée à l'heure d'été¹⁰. Cela consiste à corriger l'heure locale. En heure dite d'hiver, la correction est nulle, tandis que pour l'heure d'été, il faut rajouter une heure à l'heure locale.

Aussi, l'heure locale se calcule depuis l'heure UTC par la relation :

$$\text{Heure_locale} = \text{Heure_UTC} + \text{Correction_UTC} + \text{Correction_Ete-Hiver}.$$

⁹ Pour approfondir la notion de temps UTC : https://fr.wikipedia.org/wiki/Temps_universel_coordonn%C3%A9

¹⁰ Heure d'été : voir aussi https://fr.wikipedia.org/wiki/Heure_d%27%C3%A9t%C3%A9.

En fonction du moment auquel s'effectue cette correction pour obtenir l'heure locale, la date locale peut aussi être affectée. Il est alors possible d'avancer ou de reculer d'un jour (mais aussi d'un mois ou d'une année !).

Illustrations :

Heure UTC	Date UTC	Correction UTC	Correction Eté - Hiver	Heure locale	Date locale
22 : 28 : 52	Mardi 08.12.2020	+01 : 00 : 00 (Paris)	00 : 00 : 00	23 : 28 : 52	Mardi 08.12.2020
22 : 28 : 52	Mardi 20.10.2020	+01 : 00 : 00 (Paris)	01 : 00 : 00	00 : 28 : 52	Mercredi 21.10.2020
22 : 28 : 52	Mardi 20.10.2020	+05 : 30 : 00 (New Delhi)	00 : 00 : 00	03 : 58 : 52	Mercredi 21.10.2020
22 : 28 : 52	Mardi 20.10.2020	-05 : 00 : 00 (Montréal)	00 : 00 : 00	17 : 28 : 52	Mardi 20.10.2020
22 : 28 : 52	Mardi 20.10.2020	+11 : 00 : 00 (Sydney)	00 : 00 : 00	09 : 28 : 52	Mercredi 21.10.2020
08 : 28 : 52	Mardi 20.10.2020	-10 : 00 : 00 (Hawaï)	00 : 00 : 00	22 : 28 : 52	Lundi 19.10.2020

Par défaut, le fuseau horaire utilisé sera celui de Paris, France, en heure d'hiver, avec la correction UTC +01 :00 :00.

Par la suite, afin de pouvoir manipuler facilement des fuseaux horaires différents et gérer la mise à jour du fuseau horaire courant, utilisé pour afficher la date et l'heure, nous utiliserons le type de données constitué des champs ci-dessous :

- pays : tableau de caractères, 25 éléments ;
- ville : tableau de caractères, 20 éléments ;
- decalage_horaire : type heure

Ecrire la fonction *Correction_Hheure_Date()* qui recalculera la date et l'heure locales en fonction de la date et heure UTC, de la correction UTC et de la correction Eté-Hiver.

- Paramètres d'entrée :
 - o Date et Heure UTC
 - o Correction UTC
 - o Correction Eté-Hiver
- Valeur renournée : la date et l'heure locales

Effectuer des tests unitaires afin de valider l'ensemble des cas pouvant se présenter (dont gestion année bissextile, mois à 30 ou 31 jours, mois de février, ...).

Ecrire la fonction *Extract_date_heure_from_GPS()* qui permettra de récupérer la date et l'heure depuis un message de type \$GPRMC.

- Paramètres d'entrée :
 - o Chaine de caractères du champ date du message \$GPRMC
 - o Chaine de caractères du champ date du message \$GPRMC
- Valeur renournée : la date et l'heure UTC

Depuis l'acquisition d'un message de type \$GPRMC, vérifier que vous obtenez le résultat attendu.

10.1.2. Mise en œuvre : horloge RTC + module GPS + Sortie terminal série

Cette partie constitue la synthèse de la gestion du circuit d'horloge RTC, couplée à l'utilisation du module GPS pour la gestion de la mise à jour de la date et de l'heure. On pourra s'appuyer sur les fonctionnalités développées précédemment.

Vous disposez du squelette de programme *Station_Mesures_V0* dont vous devez compléter les différentes parties.

La date et l'heure **seront affichées au sein du terminal série**, avec un rafraîchissement toutes les secondes. De plus, les indications concernant la ville de référence du fuseau horaire, l'indicateur heure Eté-Hiver et l'état de la synchronisation du module GPS seront affichés dans le terminal série.

 Afin d'anticiper sur la suite, en particulier l'affichage des données sur l'écran TFT, veillez à bien séparer les fonctions de traitements de vos données de celles liées à la gestion de l'affichage.

A partir du squelette du projet *Station_Mesures*, finaliser le projet ***Station_Mesures_V01*** qui vous permettra de valider :

- L'acquisition périodique de la date et de l'heure depuis le module horloge RTC ;
- De vérifier que la mise à jour de l'heure s'effectue correctement dès que le module GPS est synchronisé ;
- D'afficher au sein du **terminal série** la date, l'heure, l'état de synchronisation du module GPS, la ville du fuseau horaire de référence et l'indicateur Eté-Hiver.

Lorsque vous aurez validé le fonctionnement du programme *Station_Mesures_V01*, **le compléter** afin de gérer l'affichage sur le **terminal série** des données acquises depuis le capteur BME680. Le rafraîchissement de l'affichage des données issues du BME680 sera effectué toutes les 3s (période d'acquisition des mesures du capteur). Sauvez alors votre programme pour obtenir le programme ***Station_Mesures_V02***.

Vous disposez maintenant de l'ensemble des fonctions de gestion du circuit RTC DS1307, du module GPS et du capteur BME680. Nous allons maintenant gérer l'affichage de l'ensemble des informations sur l'écran TFT couleur dans une première étape. Ensuite, nous verrons comment exploiter les capacités de la dalle tactile.

11. Utilisation de l'écran TFT

L'écran utilisé a déjà été présenté plus haut (cf. §3.6). Nous détaillerons ici les éléments de mise en œuvre de l'affichage ainsi que ceux associés à l'utilisation de la dalle tactile.

11.1. Ecran TFT couleur

11.1.1. Système de coordonnées écran et manipulation des couleurs

L'écran utilisé est un écran couleur RGB 16 bits (les couleurs sont codées sur 18 bits au format RGB). Sa résolution est :

- Selon X : 320 pixels
- Selon Y : 480 pixels,

L'élément de base de l'affichage est le pixel. La position d'un pixel au sein de l'écran est définie par le couple (x,y) de coordonnées entières, dans le système de coordonnées (X, Y) par défaut décrit par la Figure 46 ci-dessous.

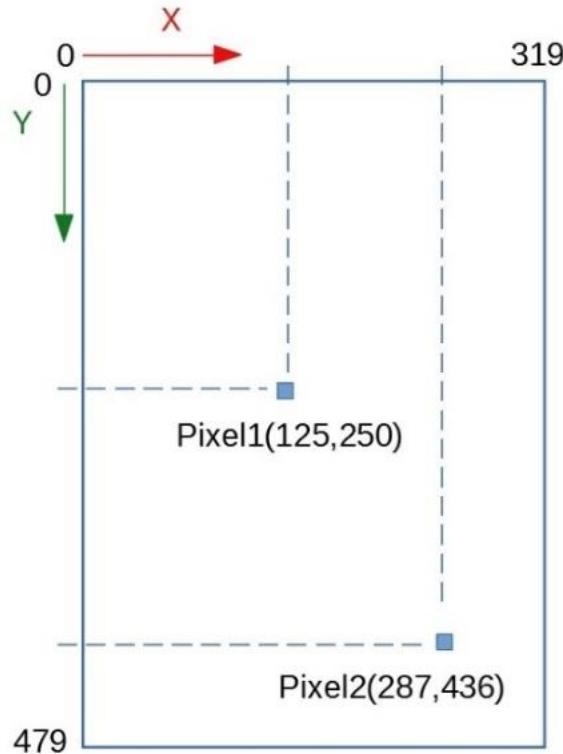


Figure 46: Système de coordonnées écran TFT

La couleur de chaque pixel est définie par un ensemble de 2 octets, en représentation non signée, selon la convention RGB. Le niveau des trois composantes de couleur R (Red) G (Green) B (Blue) est codé selon la convention suivante (cf. Figure 47) (Source : <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-gfx-graphics-library.pdf>) . La manipulation des couleurs sera facilitée par l'utilisation de constantes symboliques telles que :

```
#define BLACK 0x0000
#define BLUE 0x001F
#define RED 0xF800
```

Les constantes symboliques associées à plusieurs couleurs sont fournies au sein d'un fichier d'entête.

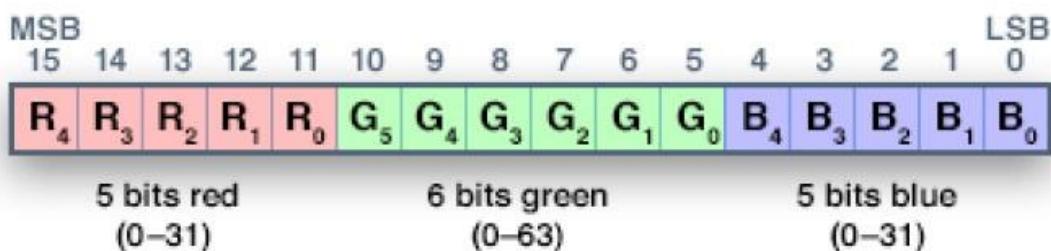


Figure 47: Convention de codage des couleurs en RGB

11.1.2. Configuration matérielle – Interfaçage avec la carte Arduino Mega2560

L'écran possède un contrôleur qui gère l'affichage de chaque pixel (position et couleur). L'échange des données entre la carte Arduino et le contrôleur s'effectuera en mode parallèle sur 8 bits. Le shield écran possède des entrées sorties (cf. Figure 48) :

- Pour la gestion du mode 8 bits parallèle et l'envoi des données vers le contrôleur graphique, les broches [D0.....D7] ;
- Les broches de contrôle [RST, RD, WR, C/D, CS] ;
- La gestion du rétroéclairage [Backlite].

en plus de l'alimentation : broches du bloc PWR [3-5V, GND].

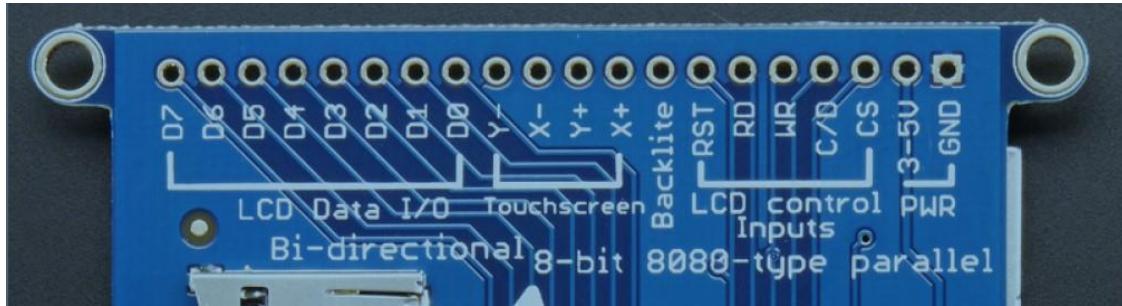


Figure 48: GPIO de l'écran TFT pour le mode 8 bits

La connexion de l'écran TFT sur la carte Arduino s'effectue selon les indications du tableau ci-dessous (cf. Tableau 5). La dalle tactile n'est pas utilisée pour l'instant.

	Shield écran	Carte Arduino Mega2560
LCD Data I\O	D0	Pin 22
	D1	Pin 23
	D2	Pin 24
	D3	Pin 25
	D4	Pin 26
	D5	Pin 27
	D6	Pin 28
	D7	Pin 29
LCD Control	RST	A4
	RD	A3
	WR	A2
	C/D	A1
	CS	A0
PWR	3-5V	5V
	GND	Gnd
Rétroéclairage	Backlite	Pin 46 (pwm)

Tableau 5: Connexion du Shield écran sur carte Arduino Mega2560 (1)

11.1.3. Mise en œuvre logicielle et test

L'utilisation de l'écran TFT nécessite de faire appel à 2 bibliothèques spécifiques, l'une pour gérer la couche matérielle, la seconde pour permettre l'accès à des fonctions graphiques de haut niveau.

La gestion de la couche matérielle s'appuie sur la librairie **Adafruit_TFTLCD**.

L'accès à des fonctions graphiques de haut niveau impose l'utilisation de la librairie **Adafruit_GFX**. La documentation de cette bibliothèque est accessible via le lien :

<https://cdn-learn.adafruit.com/downloads/pdf/adafruit-gfx-graphics-library.pdf?timestamp=1602659067>

et permet d'accéder à la documentation des fonctions graphiques de base.

Lorsque la librairie *Adafruit_TFTLCD* est installée, des exemples d'utilisation de l'écran sont disponibles. Parmi les programmes accessibles, vous pouvez tester *graphictest*. Vous procéderez aux modifications nécessaires concernant la définition des broches de contrôle de l'écran TFT.

 L'écran TFT utilisé ne dispose pas de buffer. Aussi, lorsque l'on souhaite rafraîchir son contenu, l'on dispose de deux possibilités :

- 1. Effacer tout le contenu de l'écran avant rafraîchissement, mais cela prend du temps ;
- 2. Ne rafraîchir l'affichage que des éléments qui ont fait l'objet d'une modification. Dans ce cas, il faut d'abord « effacer » l'élément déjà affiché puis réécrire au même emplacement la nouvelle donnée. **Cette méthode qui devra être privilégiée.**

11.2. Affichage de la date et de l'heure

Pour l'heure, le format d'affichage adopté est hh : mm : ss, par exemple 17 : 03 : 28 pour 17h 3 minutes et 28 secondes. Il est rafraîchi toutes les secondes.

Pour la date, le format d'affichage est <Jour> <Quantième> <Mois> <Année> par exemple Vendredi 1 janvier 2010. Il est rafraîchi toutes les secondes. **Vous avez toute latitude pour définir la disposition des différents éléments sur l'écran TFT.**

Au démarrage de l'application certains champs possèdent une valeur par défaut, définies ci-dessous. La périodicité de mise à jour de l'affichage de chaque champs est également précisée, tout comme le format d'affichage (cf. Tableau 6).

Eléments à afficher	Format	Valeur par défaut	Périodicité de rafraîchissement
Date	Jour Quantième mois Année	Vendredi 1 janvier 2010	Toutes les secondes
Heure	hh : mm : ss	00 : 00 : 00	Toutes les secondes
Mode heure été - hiver	Ete ou Hiver	Hiver	Si modification
Fuseau horaire courant	Chaîne de caractères	Paris	Si modification
Témoin synchro module GPS	A définir par utilisateur	Non synchronisé	Si modification

Tableau 6 : Valeur par défaut champs, format d'affichage et périodicité de rafraîchissement (1)

Ecrire la fonction *TFT_Affiche_Date()* qui permettra d'afficher sur l'écran TFT la date selon le format spécifié.

- Paramètres d'entrée :
 - o La date courante ;
 - o La date précédente.
- Valeur retournée : aucune.

Ecrire la fonction *TFT_Affiche_Heure()* qui permettra d'afficher sur l'écran TFT l'heure selon le format spécifié.

- Paramètres d'entrée :
 - o L'heure courante ;
 - o L'heure précédente.
- Valeur retournée : aucune.

Ecrire la fonction *TFT_Affiche_Indicateur_Ete_Hiver()* qui permettra de savoir sur l'écran TFT si l'on est en heure d'hiver ou d'été.

- Paramètres d'entrée :
 - o Indicateur Eté – Hiver courant ;
 - o Indicateur Eté – Hiver précédent.
- Valeur retournée : aucune.

Ecrire la fonction *TFT_Affiche_ville_ref_fuseau_horaire()* qui permettra d'afficher sur l'écran TFT la ville du fuseau horaire de référence.

- Paramètres d'entrée :
 - o Ville du fuseau horaire de référence courant ;
 - o Ville du fuseau horaire de référence précédent.
- Valeur retournée : aucune.

Ecrire la fonction *TFT_Affiche_Etat_Synchro_GPS()* qui permettra d'afficher sur l'écran TFT l'état de synchronisation du module GPS.

- Paramètres d'entrée :
 - o Etat synchro module GPS ;
 - o Valeur retournée : aucune.

Ecrire alors le programme *Station_Mesures_V1* qui vous permettra de vérifier :

- Que la gestion de tous les éléments d'affichage (date, heure, état synchro GPS, ville fuseau horaire de référence, indicateur Eté - Hiver) sur l'écran TFT est correcte.
- Que la mise à jour de la date et de l'heure s'effectue correctement en utilisant le module GPS.

A cette étape, l'ensemble des données de date et d'heure est totalement géré.

11.3. Affichage des mesures du capteurs BME680

L'affichage des données sur l'écran principal va être maintenant complété avec celles issues du capteur BME680 :

- Valeur de la température ;
- Valeur de la pression atmosphérique ;
- Valeur du taux d'humidité ;
- Valeur de l'index de qualité de l'air ;
- Valeur de la précision sur la mesure de qualité de l'air.

Eléments à afficher	Format	Valeur par défaut	Péodicité de rafraîchissement
Température	21.2°C	1 ^{ère} mesure effectuée	Toutes les 3 secondes
Pression	1013.34 hPa	1 ^{ère} mesure effectuée	Toutes les 3 secondes
Taux humidité relative	45.6%	1 ^{ère} mesure effectuée	Toutes les 3 secondes
IAQ	53.2	1 ^{ère} mesure effectuée	Toutes les 3 secondes
IAQ_Accuracy	2	1 ^{ère} mesure effectuée	Toutes les 3 secondes

Du fait des particularités de l'écran utilisé, en particulier du point de vue du rafraîchissement des éléments affichés, il est indispensable de ne modifier dans les valeurs à afficher que les digits qui sont modifiés. Par exemple, pour la pression atmosphérique :

- Mesure courante : 1023.12 hPa
- Mesure précédente : 1022.98 hPa (qui est celle affichée sur l'écran TFT)

alors l'affichage de la mesure courante nécessite de ne modifier que les digits surlignés en jaune. Concrètement, les digits non modifiés sont réécrits tels quels, tandis que pour les digits à modifier, il faut :

- « Effacer » le digit à modifier de la mesure précédente ;
- Ecrire le nouveau digit de la mesure courante au même emplacement ;

D'autre part, il va falloir gérer l'affichage de valeurs réelles ou entière, positives ou négatives, avec des parties entières ou décimales dont le nombre de chiffres est variable. Lorsque le digit de rang le plus élevé de la partie entière est nul, alors il n'est pas affiché. Ainsi, à titre d'exemple, si la valeur de la pression atmosphérique est 998.21hPa, il faudra l'afficher sous cette forme et non sous la forme 0998.21hPa.

Eléments à afficher	Format	Partie entière	Partie décimale	Données signées
Température	21.2°C	2 digits	1 digit	Oui
Pression	1013.34 hPa	4 digits	2 digits	Non
Taux humidité relative	45.6%	2 digits	1 digit	Non
IAQ	53.2	3 digits	1 digit	Non
IAQ_Accuracy	2	1 digit	0 digit	Non

Enfin, il va falloir définir sous une forme unique l'emplacement où s'effectuera l'affichage de chaque mesure sur l'écran TFT. Pour les données réelles, la référence de position sera donnée par la position du point décimal sur l'écran TFT, et l'affichage des digits s'effectuera par rapport à cette référence, aussi bien pour la partie entière que pour la partie décimale.

Au final, nous devons écrire une fonction d'affichage de valeurs numériques qui puisse être utilisée pour l'affichage des mesures issues du capteur BME680. Les paramètres d'entrée de cette fonction seront :

- La valeur de la mesure courante à afficher (dans [-9999.99 ; 9999.99])
- La valeur de la mesure précédente déjà affichée ;
- Le nombre de digits de la partie décimale. Valeur max : 2. Si 0, valeur entière
- Le nombre de digits de la partie entière. Valeur max : 4
- La couleur d'affichage sur écran TFT. Codée sur 16 bits
- La taille des caractères
- pos_ref_x : position de référence selon x sur afficheur TFT du point décimal
- pos_ref_y : position de référence selon y sur afficheur TFT du point décimal
- L'unité : chaîne de caractères pour afficher l'unité de la mesure (hPa, %,...) si besoin
- Le signe : flag pour gérer l'affichage du signe de la mesure. Si 0 : pas d'affichage du signe de la mesure

La fonction ne retournera aucune valeur.

Ecrire la fonction *TFT_Affichage()* qui permettra de gérer l'affichage des données sur l'écran TFT des données réelles essentiellement. A partir de cette fonction, vous devrez gérer l'affichage de l'ensemble des données de mesure du capteur BME680. Si besoin, vous pouvez également écrire des fonctions additionnelles (s'appuyant sur la fonction générique *TFT_Affichage()*) qui vous permettront de gérer plus facilement l'affichage des différentes mesures.

La validation des affichages sera effectué au sein du programme *Station_Mesures_V1*.

11.4. Graphe d'évolution de la pression moyenne

En cours de rédaction

11.5. Finalisation de l'affichage de l'écran principal

En cours de rédaction

L'écran principal (cf. Figure 1) doit permettre d'accéder aux informations suivantes :

- Affichage de la date ;
- Affichage de l'heure ;
- Affichage du mode heure été - hiver
- Affichage du fuseau horaire courant sous la forme de la ville de référence du fuseau horaire ;
- Affichage de l'état de synchronisation du module GPS ;
- Affichage de la température en °C, avec affichage des 1/10 de °C ;
- Affichage de la pression en hPa, avec affichage des dixièmes et centièmes d'hPa ;
- Affichage du taux d'humidité relative, avec affichage des dixièmes ;
- Affichage de la valeur de l'IAQ ;
- Affichage de la valeur de IAQ_Accuracy ;

- Affichage de la variation de la valeur moyenne de la pression atmosphérique sur une amplitude d'une heure ;
- Affichage du graphe d'évolution de la valeur moyenne de la pression atmosphérique, par tranche d'une heure, sur une amplitude d'au moins 48h.

En cours de rédaction - En cas d'appui sur n'importe quel point de la dalle tactile, alors on affiche le contenu de l'écran correspondant à Ecran Menu (cf. Figure 2). Si l'on valide « Annuler » alors on revient à l'écran principal. En l'absence de toute action, au bout d'un temps à définir, on revient sur l'écran principal.

Au démarrage de l'application certains champs possèdent une valeur par défaut, définies ci-dessous. La périodicité de mise à jour de l'affichage de chaque champ est également précisée, tout comme le format d'affichage (cf.).

Eléments à afficher	Format	Valeur par défaut	Périodicité de rafraîchissement
Date	Jour Quantième mois Année	Vendredi 1 janvier 2010	Toutes les secondes
Heure	hh : mm : ss	00 : 00 : 00	Toutes les secondes
Mode heure été - hiver	Ete ou Hiver	Hiver	Si modification
Fuseau horaire courant	Chaine de caractères	Paris	Si modification
Température	21.2°C	1 ^{ère} mesure effectuée	Toutes les 3 secondes
Pression	1013.34 hPa	1 ^{ère} mesure effectuée	Toutes les 3 secondes
Taux humidité relative	45.6%	1 ^{ère} mesure effectuée	Toutes les 3 secondes
IAQ	53.2	1 ^{ère} mesure effectuée	Toutes les 3 secondes
IAQ_Accuracy	2	1 ^{ère} mesure effectuée	Toutes les 3 secondes
Témoin synchro module GPS	A définir par utilisateur	Non synchronisé	Si modification
Variation valeur moyenne pression	± 1.2 hPa	0.0	Toutes les heures
Graphe évolution pression moyenne	Graphique		Toutes les heures

12.Dalle tactile

Le shield écran Adafruit 2050 possède une dalle tactile résistive qui permettra d'interagir avec l'application. Elle est placée au-dessus de l'écran TFT. Elle permet de détecter la pression du doigt ou d'un stylet.

<https://cdn-learn.adafruit.com/downloads/pdf/adafruit-3-5-color-320x480-tft-touchscreen-breakout.pdf?timestamp=1602648354>

Il s'agit d'un dispositif indépendant de l'écran TFT, dont le référentiel spatial est indépendant de celui de l'écran TFT.

L'utilisation de la dalle tactile repose sur l'utilisation de la bibliothèque *touchscreen*.

12.1. Système de référence dalle tactile – lien avec référentiel écran TFT

La surface de la dalle tactile et celle de l'écran TFT sont superposées selon la figure ci-dessous (cf. Figure 49).

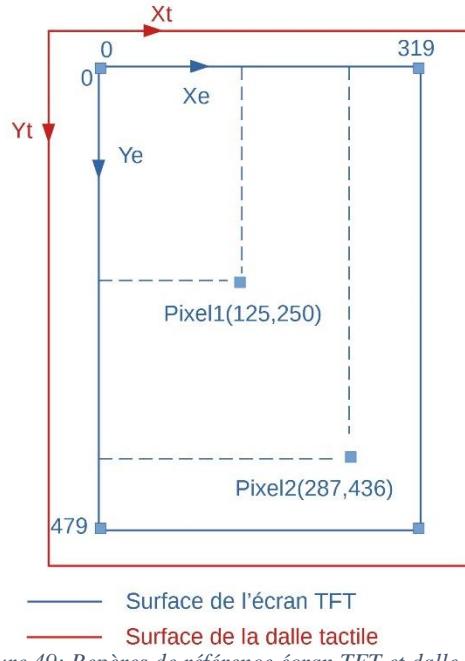


Figure 49: Repères de référence écran TFT et dalle tactile

Le repère de référence de l'écran TFT est défini par (X_e , Y_e), tandis que celui de la dalle tactile est défini par (X_t , Y_t).

Les données issues de la dalle tactile, suite à une pression exercée sur cette dernière, sont constituées par un triplet (x_t , y_t , z_t) défini dans le repère (X_t , Y_t) où :

x_t : coordonnées selon X_t du point d'appui sur la dalle tactile ;

y_t : coordonnées selon Y_t du point d'appui sur la dalle tactile ;

z_t : mesure de la pression exercée par l'appui sur la dalle tactile.

Dans le repère (X_e , Y_e) de l'écran TFT, la position d'un pixel est définie par le couple (x_e , y_e) où :

x_e : coordonnée selon X_e d'un pixel ;

y_e : coordonnée selon Y_e d'un pixel.

Soient $TSMINX$, $TSMAXX$, $TSMINY$, $TSMAXY$, définis dans le repère (X_t , Y_t) de la dalle tactile tels que :

- La valeur de $TSMINX$ correspond à un pixel de l'écran TFT de coordonnée $x_e = 0$;
- La valeur de $TSMAXX$ correspond à un pixel de l'écran TFT de coordonnée $x_e = 319$;
- La valeur de $TSMINY$ correspond à un pixel de l'écran TFT de coordonnée $y_e = 0$;
- La valeur de $TSMAXY$ correspond à un pixel de l'écran TFT de coordonnée $y_e = 479$;

Les repères de référence de l'écran TFT et de la dalle tactile possédant la même orientation, connaissant les valeurs de $TSMINX$, $TSMAXX$, $TSMINY$, $TSMAXY$, il est alors possible d'exprimer (x_e , y_e) en fonction de (x_t , y_t) par un « simple » changement d'échelle. Pour cela, il est possible d'utiliser la fonction `map()` dont la description est disponible sur <https://www.arduino.cc/reference/en/language/functions/math/map/>.

12.2. Configuration matérielle

L'utilisation de la dalle tactile nécessite de câbler les broches Y_- , X_- , Y_+ et X_+ du shield écran (cf. Figure 48) sur les entrées – sorties de la carte Arduino Mega2560. Le tableau ci-dessous fournit les broches de la carte microcontrôleur qui seront utilisées (cf. Tableau 7).

Shield écran	Carte Arduino Mega2560
--------------	------------------------

Touchsreen	Y-	Pin 45
	X-	A5
	Y+	A6
	X+	Pin 44

Tableau 7: Connexion du Shield écran sur carte Arduino Mega2560 (2)

12.3. Calibration et mise en œuvre de la dalle tactile

A partir du programme *touchscreenemoshield_V0* fourni, vérifier que la dalle tactile est active et que vous pouvez visualiser les valeurs du triplet (x_t , y_t , z_t) dans le terminal série. Relever alors, pour votre shield écran, les valeurs de TSMINX, TSMAXX, TSMINY, TSMAXY.

Ecrire alors un second programme *touchscreenemoshield_V1* qui inclura le mapping entre les valeurs de (x_t , y_t) et celles de (x_e , y_e) et affichera alors les coordonnées écran correspondant au point d'appui sur la dalle tactile.

Ecrire la fonction *GetPointDalleTactile()* qui renverra une donnée de type *TSPoint*. Si un point de la dalle tactile est activé, alors les valeurs renvoyées correspondront au triplet (x_e , y_e , z_t). Dans le cas contraire ce sera le triplet (0, 0, 0) qui sera retourné.

12.4. Test écran TFT et dalle tactile

Afin de tester le bon fonctionnement de l'écran TFT conjointement avec la dalle tactile, vous disposez du programme *tftpaint* qui vous permettra de valider le bon fonctionnement du shield écran. Il sera nécessaire de le modifier pour tenir compte de branchements du shield écran sur la carte Arduino. Ce même programme constitue une base d'exemples de l'utilisation des fonctions graphiques de la librairie Adafruit_GFX.

Annexe

1. Timer1 : mode normal :

[...] The simplest mode of operation is the Normal mode ($WGMn3:0 = 0$). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value ($MAX = 0xFFFF$) and then restarts from the BOTTOM (0x0000). In normal operation the Timer/Counter Overflow Flag (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOVn Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime [...] (cf. Documentation Atmel de l'Atmega 2560).

2. Timer1 : configuration des bits du registre TCCR1A (cf. documentation Atmel)

Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

3. Timer 1 : configuration des bits du registre TCCR1A et TCCR1B (cf. documentation Atmel)

Table 17-2. Waveform Generation Mode Bit Description⁽¹⁾

Mode	WGMr3	WGMr2 (CTCn)	WGMr1 (PWMr1)	WGMr0 (PWMr0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Note: 1. The CTCn and PWMr1:0 bit definition names are obsolete. Use the WGMr2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

If external pin modes are used for the Timer/Counter, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

4. Structure du projet – Arborescence des répertoires

Structure de l'archive mise en ligne.

Nom	Type	Taille compressée	Protégé pa...	Taille	Ratio	Modifié le
Affichage.cpp	Fichier CPP	1 Ko	Non	1 Ko	57 %	27/11/2020 16:14
Affichage.h	Fichier H	1 Ko	Non	1 Ko	26 %	27/11/2020 16:06
BME680_Sensor.cpp	Fichier CPP	1 Ko	Non	1 Ko	42 %	27/11/2020 16:06
BME680_Sensor.h	Fichier H	1 Ko	Non	1 Ko	28 %	27/11/2020 16:06
Calendrier.cpp	Fichier CPP	1 Ko	Non	1 Ko	34 %	27/11/2020 16:06
Calendrier.h	Fichier H	1 Ko	Non	1 Ko	12 %	27/11/2020 16:06
GPS.cpp	Fichier CPP	1 Ko	Non	1 Ko	40 %	27/11/2020 16:43
GPS.h	Fichier H	1 Ko	Non	1 Ko	30 %	27/11/2020 16:06
RTC_DS1307.cpp	Fichier CPP	1 Ko	Non	1 Ko	41 %	27/11/2020 16:14
RTC_DS1307.h	Fichier H	1 Ko	Non	1 Ko	37 %	27/11/2020 16:06
Station_Mesures.ino	Arduino file	1 Ko	Non	2 Ko	61 %	27/11/2020 16:17
Tactile.cpp	Fichier CPP	1 Ko	Non	1 Ko	4 %	27/11/2020 16:06
Tactile.h	Fichier H	1 Ko	Non	1 Ko	32 %	27/11/2020 16:06
TFT_Affichage.cpp	Fichier CPP	1 Ko	Non	1 Ko	6 %	27/11/2020 16:06
TFT_Affichage.h	Fichier H	1 Ko	Non	1 Ko	50 %	27/11/2020 16:07