Chapitre 16 - Présentation du module Arduino Uno et de l'environnement de développement intégré (IDE)

Présentation du module Arduino Uno

Comme nous utiliserons les modules Arduino pour nous initier à l'utilisation des microcontrôleurs, nous allons commencer par regarder les caractéristiques du module Arduino Uno.

Tension d'alimentation

Nous savons que le module Uno est architecturé autour d'un microcontrôleur 8 bits de marque Atmel, modèle ATmega328P. Celui-ci fonctionne avec une tension comprise entre 1,8 et 5,5 V (données constructeur) ; retenons le 5 V, ce qui permettra d'être compatible avec les signaux TTL (voir le chapitre 8). C'est ce choix qui a été fait par Arduino (5 V), en conséquence, cette tension doit se trouver sur le module Uno. Plusieurs possibilités existent :

- Alimentation par la prise USB de votre ordinateur : c'est le cas le plus simple, le module Uno est relié à l'ordinateur par un câble USB. C'est le cas lors de la phase de mise au point du programme. L'interface USB délivre la tension de 5 V nécessaire et une sécurité existe sur le module Uno pour couper le courant au cas où celui-ci dépasserait 500 mA sur le port USB.
- Alimentation extérieure : le module Uno dispose d'une entrée jack qu'on peut raccorder à un adaptateur transformant le courant EDF en courant continu (adaptateur AC-DC). La prise jack doit avoir 2,1 mm de diamètre et **avoir le pôle + en son centre**. On peut aussi raccorder une batterie sur cette prise jack.
- Les bornes d'une batterie peuvent être raccordées au connecteur marqué Vin et au connecteur marqué GND. Dans ce cas, la borne négative de la batterie est raccordée à GND (la masse) et la borne positive est raccordée à Vin (tension d'entrée).

La figure 16.1 montre la prise jack qui est entourée en violet et les connecteurs GND et Vin du connecteur d'alimentation entouré en vert sont repérés par une flèche violette. Dans le cas d'une alimentation extérieure par la prise jack ou sur les connecteurs Vin et GND, la tension recommandée doit être comprise entre 7 et 12 V (au-dessous de 7 V, la tension de 5 V nécessaire au microcontrôleur peut ne pas être atteinte et au-dessus de 12 V, le régulateur de tension, fabriquant le 5 V nécessaire au microcontrôleur, risque de chauffer).

Une tension de 3,3 V est également fabriquée par le module Uno à partir du 5 V dont il dispose ; cette tension peut être récupérée sur le connecteur d'alimentation sur la broche repérée par une flèche noire mais le courant délivré ne dépasse pas 50 mA.

La tension de 5 V est aussi récupérable sur la broche marquée 5 V (à droite de la flèche noire sur la figure 16.1).

Attention : il ne faut jamais alimenter le module Uno par ces deux broches (5V et 3,3 V) car cela surpasse les régulateurs de tension et peut endommager gravement le module Uno. En conséquence, retenir que le module Uno doit être alimenté **via le câble USB ou par la prise jack**.

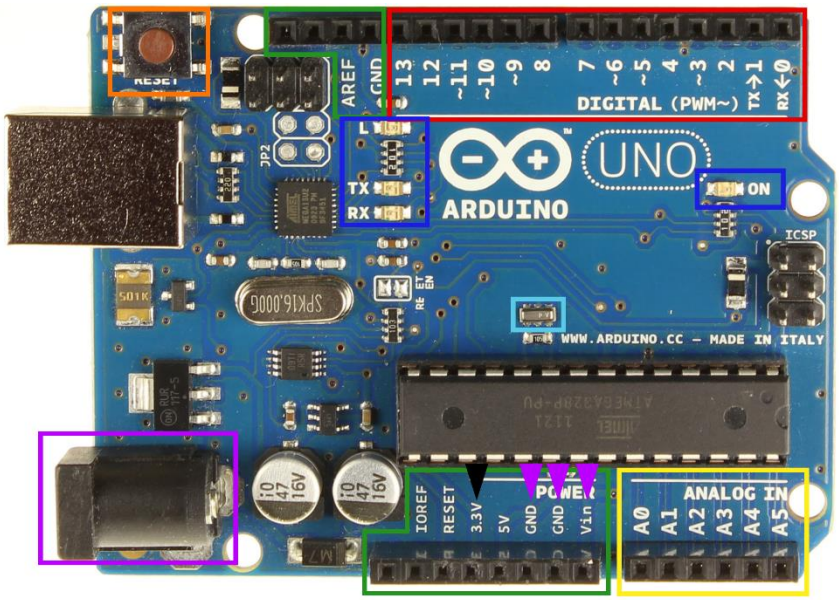

Figure 16. 1

Fréquence de l'horloge

Le microcontrôleur ATmega328P peut travailler jusqu'à une fréquence de 16 Mhz. Un quartz de 16 Mhz permet de régler précisément l'horloge ; il est repéré en bleu clair sur la figure 16.1. Plus la fréquence horloge d'un microcontrôleur est élevée, et plus le microcontrôleur travaille rapidement (nous en reparlerons ultérieurement).

Mémoire

Le microcontrôleur ATmega328P dispose de 32 k-octets de mémoire qui peut être programmée ou reprogrammée autant qu'on le veut ; cette mémoire ne s'efface pas quand on coupe l'alimentation du module. Sur ces 32 KB (B pour byte ou octet en anglais), 0,5 sont utilisés par le **bootloader** qui est un petit programme servant à programmer le microcontrôleur. On en reparlera lors de l'étude de la programmation. Ces 32 KB de mémoire servent à stocker le programme ; un bon réflexe est d'apprendre à l'économiser chaque fois que c'est possible car l'inconvénient des microcontrôleurs par rapport aux microprocesseurs est la mémoire réduite qu'ils peuvent gérer.

Le microcontrôleur dispose également de 2 KB de mémoire vive SRAM (qui s'efface à la mise hors tension) et de 1 KB de mémoire EEPROM qu'on peut écrire électriquement et qui ne s'efface pas à la mise hors tension. Pour utiliser la mémoire EEPROM, une bibliothèque (en anglais library) est mise à notre disposition, ce qui nous évite d'avoir à écrire un programme pour cela.

Entrées-sorties

Chaque broche d'entrée-sortie du microcontrôleur opère en 5 V et peut recevoir (en entrée) ou délivrer (en sortie) 20 mA en opération normale ; ceci est suffisant pour allumer une LED mais pas pour autre chose de plus gourmand. On peut tout de même aller jusqu'à un maximum de **40 mA à ne jamais dépasser** sous peine d'endommager le microcontrôleur de façon irrémédiable.

De plus, le microcontrôleur ne peut pas fournir plus de 200 milliampères au total. Il convient donc de **limiter l'intensité utilisée sur chaque broche pour ne pas dépasser le total cumulé sur l'ensemble des broches (200 mA)**. Dans ce cas, il peut être nécessaire d'amplifier les signaux délivrés et on verra comment faire dans la suite du cours.

Chaque E/S dispose en interne d'une résistance de pull-up déconnectée par défaut mais qu'on peut connecter par logiciel ; ceci nous fait encore économiser des composants extérieurs au module. Nous verrons ultérieurement comment utiliser ces résistances de pull-up et à quoi cela correspond.

Voyons quelques spécificités de certaines entrées-sorties numériques :

- 0 et 1 : servent à envoyer des signaux (données série) à l'ordinateur ou à en recevoir de sa part. Pour cette raison, on évitera de les utiliser dans un programme et on les gardera **pour communiquer**, ce qui est bien utile pour effectuer la mise au point d'un programme comme on le verra plus loin.
- 2 et 3 : ces deux entrées peuvent être programmées pour effectuer des interruptions. On verra à quoi servent les interruptions dans les chapitres sur la programmation.
- 3, 5, 6, 9, 10 et 11 : peuvent délivrer un signal numérique de type PWM, ce qui permet de jouer sur le rapport cyclique (voir le chapitre 12).
- 13 : cette sortie est connectée à une LED présente sur le module, permettant déjà de vérifier le fonctionnement du module avec le programme donné en exemple « Blink » permettant de faire clignoter une LED (celle-ci en l'occurrence).

D'autres spécificités seront citées ultérieurement ; pour l'instant, contentons-nous de l'essentiel. Les entrées sorties numériques sont entourées en rouge sur la figure 16.1 et les entrées analogiques sont entourées en jaune.

Bouton poussoir de reset

Le module Uno comporte un bouton poussoir permettant de reseter le système au cas où un programme serait déficient. Il est entouré en orange sur la figure 16.1. Certaines cartes shield, comme elles s'enfichent sur le module, peuvent condamner l'accès au bouton poussoir reset ; on peut alors en rajouter un en le connectant à la broche reset située à gauche de la flèche noire sur la figure 16.1 (un état LOW sur cette broche correspond à un reset).

LED de contrôle

Le module Uno dispose de plusieurs LED qui permettent de contrôler le bon déroulement des opérations. La LED ON permet de vérifier que le module est bien alimenté, les LED Tx et Rx clignotent lorsque le module communique par données série avec l'ordinateur auquel il est relié. Enfin, la LED L est reliée à la sortie 13 du microcontrôleur.

Les LED sont entourées en bleu sombre sur la figure 16.1.

À retenir du module Arduino Uno :

- Comment alimenter le module Uno et surtout ce qu'il ne faut pas faire.
- Les quantités et types de mémoire mise à votre disposition.
- La limitation en courant de chaque E/S et ce qu'il ne faut pas dépasser.
- La spécificité de certaines broches d'E/S.

Présentation de l'environnement de développement intégré (IDE)

Après la présentation de la partie matérielle du module (le hardware), il est temps maintenant d'examiner le logiciel qui permet de programmer le microcontrôleur (le software).

Ce logiciel de programmation est appelé IDE (Integrated Development Environment) ou encore Environnement de Développement Intégré. En effet, ce petit logiciel vous permet de tout avoir sous la main pour programmer le microcontrôleur : d'une part, la possibilité d'éditer vos propres programmes en écrivant les différentes lignes d'instructions, d'autre part la possibilité de vérifier qu'il n'y a pas d'erreurs de syntaxe, enfin la possibilité de transférer ce programme directement en langage machine dans la mémoire du microcontrôleur. La dernière opération suppose donc que le logiciel soit capable de transformer ce que vous avez écrit en une suite d'octets qui seront « compris » du microcontrôleur (le fameux langage machine **propre à chaque microcontrôleur**) ; on y reviendra par la suite.

Installation de l'IDE sur votre ordinateur

Je ne vais pas développer cette installation en détail car cela a été expliqué dans ce forum de nombreuses fois ou bien sur le site www.locoduino.org auquel vous pouvez vous référer.

La première chose à faire est de se connecter au site www.arduino.cc et de choisir la page Download (téléchargement) ou bien si vous préférez, d'aller directement à cette adresse internet : https://www.arduino.cc/en/Main/Software?setlang=en

Comme vous le constatez, nous sommes actuellement à la version 1.6.12 du logiciel (novembre 2016) et celle-ci est disponible aussi bien pour Windows que Mac OS X 10.7 ou bien encore Linux 32 ou 64 bits. Pour Windows, vous avez deux choix possibles : soit installer le logiciel sur votre ordinateur, soit obtenir une version sous forme d'un fichier ZIP qui, une fois les fichiers extraits, vous donnera un répertoire comprenant tout ce qui est nécessaire au fonctionnement du logiciel lui-même présent dans le répertoire. Dans ce cas, vous n'avez pas besoin d'être administrateur puisque rien ne s'installe sur le micro et le répertoire obtenu peut être copié sur une clé pour pouvoir travailler n'importe où. Vous pouvez aussi télécharger une version antérieure du logiciel, mais ceci ne présente pas un grand intérêt.

L'environnement de développement dans sa version 1.6.12

La figure 16.2 montre l'environnement de travail de l'IDE version 1.6.12 prêt à travailler avec le module Uno relié à l'ordinateur par le câble USB ; deux fonctions sont déjà présentes car elles sont **obligatoires** dans l'écriture d'un programme. Ce sont la **fonction setup** et la **fonction loop** et il n'y a plus qu'à les compléter avec ce qu'on veut faire.

Le logiciel vous propose un nom pour le programme que vous allez écrire, basé sur la date courante (ici sketch_nov12a entouré en vert, sketch signifiant **croquis** en anglais, nom donné à un programme dans l'environnement Arduino). Vous pouvez bien sûr le modifier et lui donner un nom plus explicite (faire Fichier > Enregistrer sous). La première ligne (entourée en jaune) vous donne différents menus possibles, puis la ligne du dessous (entourée en rouge) vous offre quelques icônes que vous utiliserez souvent (Vérifier, Téléverser, Nouveau, Ouvrir, Enregistrer et tout à droite ouverture du Moniteur série, option qui est décrite plus loin).

Au-dessous, vous avez l'espace de travail (entouré en bleu) et comme je l'ai dit, les deux fonctions obligatoires sont déjà présentes. La fonction setup, ou plutôt void setup, est la fonction qui vous permet d'initialiser le microcontrôleur. La fonction void loop vous permet de mettre votre programme, celui que vous allez écrire et mettre au point. Ces deux fonctions sont capables de recevoir **n'importe quelle instruction du langage de base d'Arduino**. Elles travaillent donc de la même façon excepté une différence fondamentale : la fonction void setup **est exécutée une fois seulement** alors que la fonction void loop est **exécutée de façon répétitive**, chaque fois que le programme est terminé, il reprend au départ en boucle (loop signifie boucle en anglais). Ceci est résumé par les deux commentaires en

anglais que vous pouvez d'ailleurs supprimer pour mettre vos propres commentaires (lignes commençant par //, once signifiant une fois, repeatedly signifiant de façon répétée).

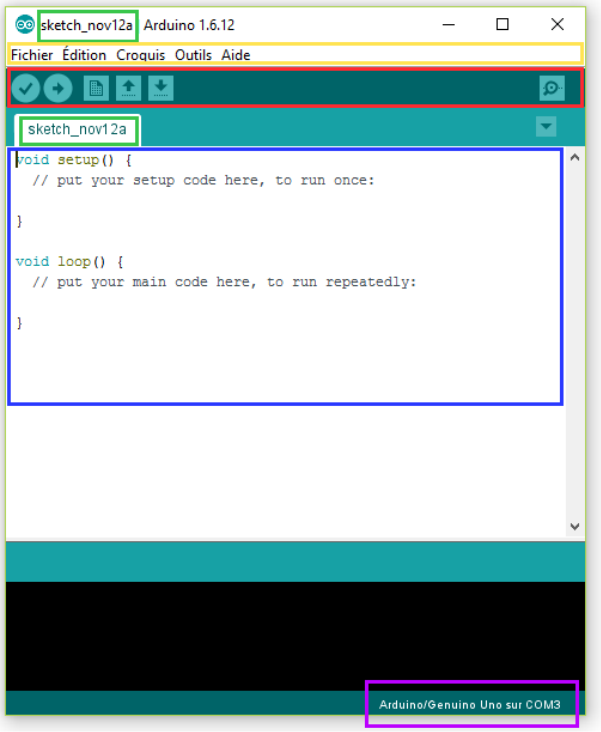

Figure 16. 2

Enfin, en bas de la fenêtre du logiciel, on retrouve que celui-ci est prêt à opérer avec un module Arduino ou Genuino Uno qui est d'ailleurs relié sur le port COM 3 (voir ce qui est entouré en violet). Plutôt que perdre du temps à développer ce que je viens d'évoquer, je vous propose de manipuler un peu le logiciel pour vous l'approprier.

Fonction setup et fonction loop

La fonction setup est exécutée une seule fois ; on l'utilise donc en général **pour initialiser le microcontrôleur avant de le faire travailler**. On pourrait donc penser que cette fonction n'est faite que pour cela et qu'elle ne peut admettre que des instructions d'initialisation. En réalité, comme je l'ai dit plus haut, la fonction setup peut recevoir toutes les instructions du langage Arduino : on peut donc l'utiliser pour initialiser (si nécessaire, mais en général c'est souvent le cas) puis exécuter une partie de programme qui ne doit être réalisée **qu'une seule fois**. Pour bien voir cela, je vous propose d'aller ouvrir le programme « toneMelody » donné en exemple. Aller dans le menu :

Fichier > Exemples > 02.Digital > toneMelody

Une nouvelle fenêtre s'ouvre avec le programme toneMelody. Ce qu'on remarque, c'est que la fonction loop ne comporte aucune instruction, comme le montre la figure 16.3. En effet, ce programme permet de jouer une petite mélodie sur un haut-parleur relié à la sortie 8 (le montage est décrit sur le site d'Arduino). En étant inclus dans la fonction setup, le programme ne joue la mélodie qu'une seule fois au lieu de la jouer en boucle (ce qui finirait d'ailleurs par vous agacer car c'est loin d'être du Mozart !).

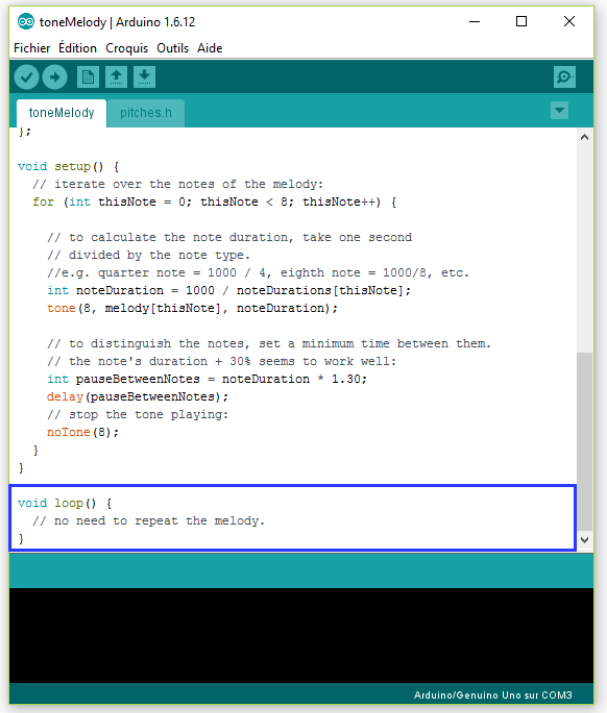

Figure 16. 3

Nous allons maintenant ouvrir un autre programme donné en exemple, le programme Blink (clignoter en anglais). Aller dans le menu :

Fichier > Exemples > 01.Basics > Blink

La figure 16.4 montre l'ensemble du programme qui est constitué de la fonction setup (entourée en rouge) et de la fonction loop (entourée en bleu).

```
// 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
}
```

Figure 16. 4

Comme on le voit, la fonction setup ne fait qu'initialiser le microcontrôleur pour mettre sa broche N° 13 en sortie avec l'instruction pinMode(LED_BUILTIN, OUTPUT); (LED_BUILTIN est la LED reliée à la sortie 13, OUTPUT signifie sortie). Ceci n'est fait **qu'une seule fois, au tout début du programme**. Par contre, la fonction loop est exécutée en boucle : ici elle allume la LED reliée à la sortie 13 (digitalWrite(LED_BUILTIN, HIGH) ;), attend un peu (delay(1000) ;), éteint la LED (digitalWrite(LED_BUILTIN, LOW) ;), attend un peu. Comme ces instructions sont répétées

continuellement (jusqu'à ce qu'on débranche l'alimentation du module), le LED clignote indéfiniment à la fréquence de 0,5 Hz (puisque la période est au total de 2 secondes).

Maintenant que le programme Blink est chargé, téléversez-le dans votre module et observez la LED clignoter. Si votre LED clignotait déjà avant (les modules sont livrés avec ce programme dans le microcontrôleur), changez les deux valeurs de delay (mettez 500 pour avoir une fréquence de 1 Hz) puis téléversez votre nouveau programme : vous constaterez que la LED clignote deux fois plus rapidement.

À retenir sur l'IDE d'Arduino :

- Les fonctions setup et loop peuvent admettre toutes les instructions du langage Arduino.
- La fonction setup n'est exécutée qu'une seule fois.
- La fonction loop est exécutée en boucle de façon répétitive.
- La fonction setup est généralement utilisée pour initialiser le microcontrôleur.
- La fonction loop est utilisée pour le programme principal qui doit s'exécuter sans arrêt, c'est-à-dire refaire continuellement les mêmes choses.

Présentation du moniteur série

L'IDE d'Arduino est livré avec un moniteur ou si vous préférez, une fenêtre qu'on peut faire apparaître et dans laquelle Arduino peut afficher des informations : valeur d'une variable, résultat d'un calcul, etc. Ce moniteur permet d'ailleurs de communiquer **dans les deux sens** avec Arduino puisqu'on peut écrire dans une zone spécifique et envoyer des données à Arduino.

Nous allons voir comment utiliser ce moniteur et surtout comprendre son intérêt.

Affichage du moniteur

L'icône située la plus à droite permet de faire apparaître le moniteur, à la condition qu'une carte Arduino soit bien reliée à l'ordinateur via un câble USB, comme le montre la figure 16.5.

Figure 16. 5

La figure 16.6 montre l'aspect de ce moniteur sur l'écran de l'ordinateur.

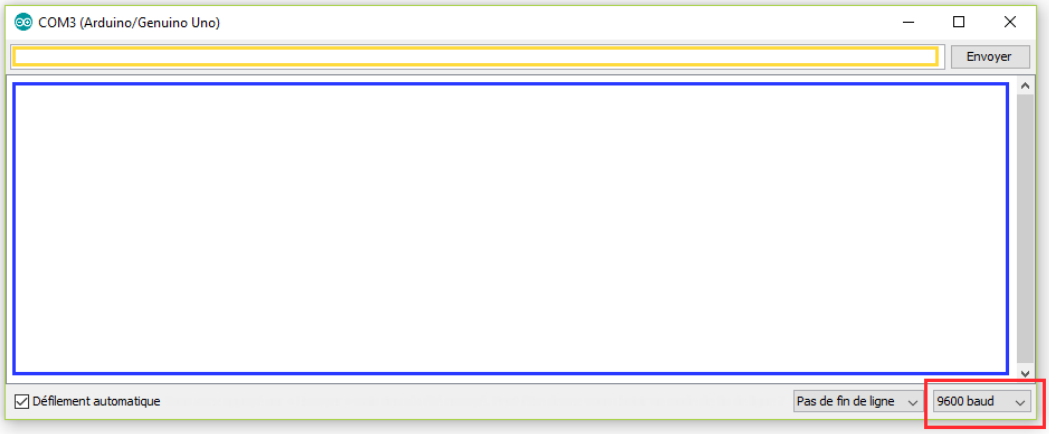

Figure 16. 6

La première zone (en jaune), à gauche du bouton « Envoyer » est la zone dans laquelle on écrit une donnée qu'on veut envoyer à Arduino. La zone la plus importante (en bleu) est la zone dans laquelle Arduino affichera ses résultats. En bas à droite, un bouton fléché (en rouge) permet de régler le débit des transmissions entre Arduino et l'ordinateur ; par défaut, il est réglé sur une vitesse de transmission de 9600 baud. La transmission des données a lieu sous forme série (les bits des octets sont envoyés les uns après les autres) par l'intermédiaire des broches 0 et 1 du module (encore appelées Rx et Tx pour réception et transmission). Nous verrons en tant voulu comment réaliser une communication série dans un programme grâce aux fonctions Serial.begin, Serial.print, Serial.println ou Serial.write.

Mise au point d'un programme

Parfois, le programme que nous avons écrit compile normalement mais le résultat n'est pas celui attendu lorsque nous l'exécutons. Il peut alors être difficile de trouver ce qui ne va pas. Pour y arriver, on peut introduire des affichages de données à certains endroits du programme afin de vérifier l'évolution de certaines variables, puis retirer ces lignes d'instructions une fois que le programme est au point et fonctionne normalement.

Cela permet de voir à quel endroit le programme s'éloigne du fonctionnement attendu et donc de remédier au problème. Par exemple, en faisant afficher un compteur de boucle, on peut vérifier que celle-ci est parcourue en entier. En faisant afficher une variable, on peut vérifier qu'il n'y a pas de dépassement de capacité, ce qui conduirait à des calculs faux par la suite. Enfin, en cas de tests, on peut voir par où le programme passe et ainsi vérifier que cela correspond à la logique qu'on veut suivre.

Le moniteur en modélisme ferroviaire

En général, une application destinée au modélisme ferroviaire n'aura pas besoin du moniteur. En effet, Arduino réagit à des capteurs disposés sur le réseau pour commander des accessoires du réseau (aiguilles, feux, moteurs, etc.). Il est donc en quelque sorte autonome et n'a pas besoin d'ordinateur. En dehors de la phase de mise au point du programme, comme nous l'avons défini au paragraphe précédent, il est rare d'avoir besoin d'afficher des données sur l'écran de l'ordinateur. De plus, pour faire cela, le module doit rester branché à l'ordinateur via le câble USB, ce qui n'est guère pratique. La meilleure raison pour ne pas utiliser le moniteur en modélisme ferroviaire, c'est que les affichages sur écran par les fonctions Serial.print() prennent du temps et les utiliser pourraient ralentir le microcontrôleur dans son travail et lui faire louper des événements sur le réseau.

À retenir sur le Moniteur Série :

- L'IDE contient un moniteur qui permet de dialoguer avec Arduino.
- On affiche le moniteur avec l'icône la plus à droite, à la condition q'un module soit relié à l'ordinateur via le câble USB.
- En modélisme ferroviaire, le moniteur est utilisé pour mettre au point les programmes qui ne se comportent pas comme on voudrait.
- Les communications doivent être initialisées dans le setup avec la fonction Serial.begin.
- Les données sont affichées avec les fonctions du type Serial.print.