

La carte ARDUINO UNO

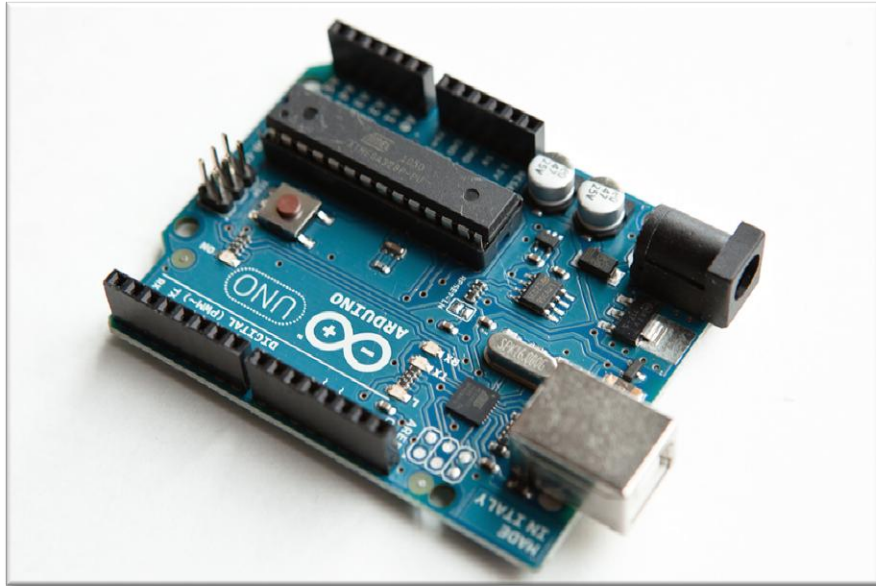



Figure 1 : présentation de la carte utilisée pour le projet ROBOT SUIVEUR DE LIGNE

I. Présentation générale de l'ARDUINO	2
A. La partie logicielle	3
B. La partie matérielle.....	3
1. Alimentation :	4
2. Mémoire.....	4
3. Entrées et sorties	4
4. Communication.....	5
5. Programmation.....	5
6. Reset automatique par Software.....	5
7. Protection de surintensité USB	5
8. Dimensions.....	5
9. Schéma structurel.....	6
II. Présentation de l'Espace de développement Intégré (EDI) Arduino	7
A. Description de l'interface	7
B. Description de la structure d'un programme	8
1. Description générale des parties.....	8
2. Description détaillée des parties.....	9
a. Définition des variables et constantes.....	9
b. Fonction principale : void setup()	10
c. Fonction boucle : void loop()	10
C. Compilation et programmation de l'ARDUINO	10
1. Ecriture de l'algorithme	10
2. Ecriture du programme.....	10
3. Compilation du programme	11
4. Sélection de la cible et du port série	11
5. Transfert du programme vers la carte ARDUINO.....	12

DOSSIER	La carte ARDUINO	
---------	------------------	---

I. Présentation générale de l'ARDUINO

Le système Arduino est une carte électronique basée autour d'un microcontrôleur et de composants minimum pour réaliser des fonctions plus ou moins évoluées à bas coût. Elle possède une interface usb pour la programmer. C'est une plateforme open-source qui est basée sur une simple carte à microcontrôleur (de la famille AVR), et un logiciel, véritable environnement de développement intégré, pour écrire, compiler et transférer le programme vers la carte à microcontrôleur.

Arduino peut être utilisé pour développer des applications matérielles industrielles légères ou des objets interactifs (création artistiques par exemple), et peut recevoir en entrées une très grande variété de capteurs. Arduino peut aussi contrôler une grande variété d'actionneurs (lumières, moteurs ou toutes autres sorties matériels). Les projets Arduino peuvent être autonomes, ou communiquer avec des logiciels sur un ordinateur (Flash, Processing ou MaxMSP). Les cartes électroniques peuvent être fabriquées manuellement ou bien être achetées préassemblées; le logiciel de développement open-source est téléchargeable gratuitement.

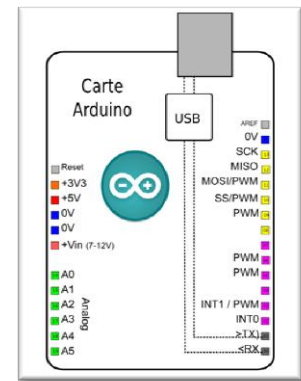


Figure 2 : carte ARDUINO UNO

Synthèse des caractéristiques

Microcontrôleur	Atmega328
Tension de fonctionnement	5V
Tension d'alimentation (recommandée)	7-12V
Tension d'alimentation (limites)	6-20V
Broches E/S numériques	14 (dont 6 disposent d'une sortie PWM)
Broches d'entrées analogiques	6 (utilisables en broches E/S numériques)
Intensité maxi disponible par broche E/S (5V)	40 mA (ATTENTION : 200mA cumulé pour l'ensemble des broches E/S)
Intensité maxi disponible pour la sortie 3.3V	50 mA
Intensité maxi disponible pour la sortie 5V	Fonction de l'alimentation utilisée - 500 mA max si port USB utilisé seul
Mémoire Programme Flash	32 KB (Atmega328) dont 0.5 KB sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	2 KB (Atmega328)
Mémoire EEPROM (mémoire non volatile)	1 KB (Atmega328)
Vitesse d'horloge	16 MHz

Qu'est-ce qu'un microcontrôleur ?

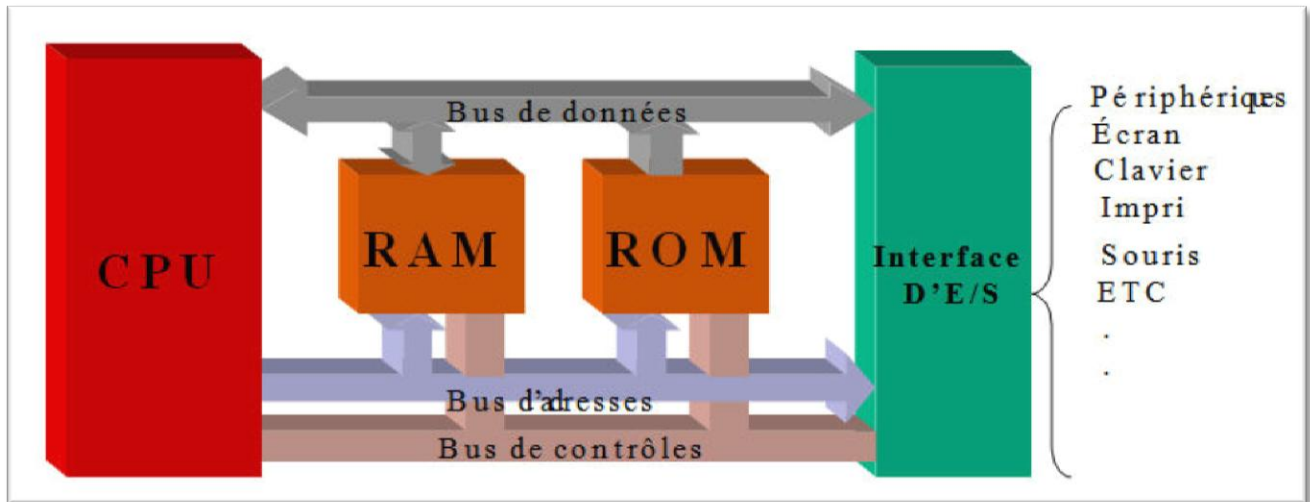


Figure 3 : schéma simplifié du contenu type d'un microcontrôleur.

Un **microcontrôleur** est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties. Les microcontrôleurs se caractérisent par un plus haut degré d'intégration, une plus faible consommation électrique (quelques milliwatts en fonctionnement, quelques nanowatts en veille), une vitesse de fonctionnement plus faible (quelques mégahertz à quelques centaines de mégahertz) et un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels.

A. [La partie logicielle](#)

Le logiciel de programmation des modules Arduino est une application Java, libre et multi-plateformes, servant d'éditeur de code et de compilateur, et qui peut transférer le firmware et le programme au travers de la liaison série (RS232, Bluetooth ou USB selon le module). Il est également possible de se passer de l'interface Arduino, et de compiler les programmes en ligne de commande.

Le langage de programmation utilisé est le C++, compilé avec `avr-g++`, et lié à la bibliothèque de développement Arduino, permettant l'utilisation de la carte et de ses entrées/sorties. La mise en place de ce langage standard rend aisé le développement de programmes sur les plates-formes Arduino, à toute personne maîtrisant le C ou le C++.

B. [La partie matérielle](#)

Un module Arduino est généralement construit autour d'un microcontrôleur ATMEL AVR (Atmega328 ou Atmega2560 pour les versions récentes, Atmega168 ou Atmega8 pour les plus anciennes), et de composants complémentaires qui facilitent la programmation et l'interfaçage avec d'autres circuits. Chaque module possède au moins un régulateur linéaire 5V et un oscillateur à quartz 16 MHz (ou un résonateur céramique dans certains modèles). Le microcontrôleur est pré-programmé avec un boot loader de façon à ce qu'un programmeur dédié ne soit pas nécessaire.

Les modules sont programmés au travers une connexion série RS-232, mais les connexions permettant cette programmation diffèrent selon les modèles. Les premiers Arduino possédaient un port série, puis l'USB est apparu sur les modèles Diecimila, tandis que certains modules destinés à une utilisation portable se sont affranchis de l'interface de programmation, relocalisée sur un module USB-série dédié (sous forme de carte ou de câble).

L'Arduino utilise la plupart des entrées/sorties du microcontrôleur pour l'interfaçage avec les autres circuits. Le modèle Diecimila par exemple, possède 14 entrées/sorties numériques, dont 6 peuvent produire des signaux PWM, et 6 entrées analogiques. Les connexions sont établies au travers de connecteurs femelle HE14 situés sur le dessus de la carte, les modules d'extension venant s'empiler sur l'Arduino. Plusieurs sortes d'extensions sont disponibles dans le commerce.

1. Alimentation :

La carte Arduino UNO peut être alimentée par l'USB ou par une alimentation externe. La source est sélectionnée automatiquement.

La tension d'alimentation extérieure (hors USB) peut venir soit d'un adaptateur AC-DC ou de piles. L'adaptateur peut être connecté grâce à un 'jack' de 2.1mm positif au centre. Le raccordement vers un bloc de piles peut utiliser les bornes Gnd et Vin du connecteur d'alimentation (POWER). La carte peut fonctionner à l'aide d'une tension extérieure de 7 à 12 volts. Les broches (pins) d'alimentation sont les suivantes :

- VIN. La tension d'entrée positive lorsque la carte Arduino est utilisée avec une source de tension externe (à distinguer du 5V de la connexion USB ou autre source 5V régulée). Vous pouvez alimenter la carte à l'aide de cette broche, ou, si l'alimentation est fournie par le jack d'alimentation, accéder à la tension d'alimentation sur cette broche.
- 5V. La tension régulée utilisée pour faire fonctionner le microcontrôleur et les autres composants de la carte (pour info : les circuits électroniques numériques nécessitent une tension d'alimentation parfaitement stable dite « tension régulée » obtenue à l'aide d'un composant appelé un régulateur et qui est intégré à la carte Arduino). Le 5V régulé fourni par cette broche peut donc provenir soit de la tension d'alimentation VIN via le régulateur de la carte, ou bien de la connexion USB (qui fournit du 5V régulé) ou de tout autre source d'alimentation régulée.
- 3V3. Une alimentation de 3.3V fournie par le circuit intégré FTDI (circuit intégré faisant l'adaptation du signal entre le port USB de votre ordinateur et le port série de l'Atmega) de la carte est disponible : ceci est intéressant pour certains circuits externes nécessitant cette tension au lieu du 5V). L'intensité maximale disponible sur cette broche est de 50mA
- GND. Broche de masse (ou 0V).

2. Mémoire

L' Atmega328 a 32 KB de mémoire (dont 0.5 KB pour le bootloader). Il a également 2 KB de SRAM et 1 KB de mémoire non volatile EPROM (qui peut être écrite et lue grâce à la librairie 'EEPROM').

3. Entrées et sorties

Chacune des 14 broches numériques de la Uno peut être utilisée en entrée (input) ou en sortie (output), en utilisant les fonctions `pinMode()`, `digitalWrite()`, et `digitalRead()`.

Elles fonctionnent en logique TTL (0V-5V) ; chacune pouvant fournir (source) ou recevoir un courant maximal de 40 mA et dispose si besoin est d'une résistance interne de 'pull-up'.

En outre, certaines broches ont des fonctions spécialisées :

- Serial : broche 0 (RX) et broche1 (TX). Permet de recevoir (RX) et de transmettre (TX) des données séries TTL. Ces broches sont raccordées à leurs homologues sur le chip Atmega8U2 spécialisé dans la conversion USB-to-TTL série.
- Interruptions externes 2 et 3. Ces broches peuvent être configurées pour déclencher une interruption sur une valeur LOW, sur un front montant ou descendant, ou encore sur le changement de valeur. (voir la fonction `attachInterrupt()` pour des détails).
- PWM : 3, 5, 6, 9, 10, and 11. Output 8-bit de PWM avec la fonction `analogWrite()`.
- SPI : 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ces broches fournissent le support de communication SPI en utilisant la 'library' spécialisée
- LED : 13. Il y a une LED connectée à la broche numérique 13.

La carte Uno a 6 broches d'entrée analogiques, A0 à A5, chacune avec 10 bits de résolution (1024 valeurs différentes).

Par défaut les mesures sont effectuées de la masse à 5V (valeur de référence), mais il est possible de spécifier la valeur de référence en utilisant la broche VREF et la fonction `analogReference()`.

En outre, certaines broches ont des fonctions spécialisées :

- I2C : 4 (SDA) and 5 (SCL). Permettent le support du bus I2C (TWI) en utilisant le 'library' Wire.

Autres broches sur la carte :

- AREF. Tension de référence déjà mentionnée.
- Reset. Permet au niveau bas (LOW° de faire un reset du contrôleur. Elle est utilisée typiquement pour monter un bouton 'reset' aux cartes additionnelles ('shields') bloquant celui de la carte principale.

4. [Communication](#)

La carte Arduino Uno a de nombreuses possibilités de communications avec *l'extérieur*. L'Atmega328 possède une communication série UART TTL (5V), grâce aux broches numériques 0 (RX) et 1 (TX).

Un contrôleur Atmega8U2 sur la carte, gère cette communication série vers l'USB et apparaît comme un port de communication virtuel pour le logiciel sur l'ordinateur.

Le firmware de l'8U2 utilise le protocole USB, et aucun driver externe n'est nécessaire.

Windows a cependant besoin d'un fichier .inf, à l'installation. Le logiciel Arduino possède un logiciel série (Telnnet) intégré permettant l'envoi et la réception de texte. Les DELs RX et TX sur la carte clignoteront pour indiquer la transmission de données vers l'ordinateur.

Une librairie 'SoftwareSerial' permet la transmission de données série à partir de chacune des broches numériques du Uno.

L'Atmega328 supporte le bus I2C (TWI) et le protocole de communication synchrone maître-esclave SPI. Le logiciel Arduino inclut un ensemble de fonctions pour mettre en œuvre l'un ou l'autre.

5. [Programmation](#)

La carte Arduino Uno peut être programmée directement avec « l'Arduino software ». L'Atmega328 sur la carte Uno est pré programmé avec un 'bootloader' qui permet de charger le code d'une nouvelle application sans utiliser un programmeur hardware externe. Il communique avec un ordinateur en utilisant le protocole STK500 d'ATMEL.

Mais vous pouvez programmer le contrôleur de la carte en utilisant le port ICSP (In-Circuit Serial Programming).

Le code source du firmware du contrôleur auxiliaire Atmega8U2 est disponible.

6. [Reset automatique par Software](#)

Il est possible d'effectuer un reset via le logiciel ARDIONO. En effet, la ligne DTR sur l'Atmega8U2 est connectée à la ligne Reset de l'Atmega328 à travers une capacité. Lorsque cette ligne est amenée à l'état logique 0, un signal Reset est envoyé au contrôleur.

7. [Protection de surintensité USB](#)

La carte Arduino Uno possède une protection par fusible pour le port USB si un courant de plus de 500mA est demandé. La déconnexion durera tant que la source de consommation excessive n'aura pas cessé.

8. [Dimensions](#)

Les longueur et largeur maximales du PCB sont de 6,9 et 5,3 cm respectivement.

9. Schéma structurel

Arduino™ UNO Reference Design

References Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Arduino reserves the right to change specifications and product descriptions without notice. Do not finalize a design with this information. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

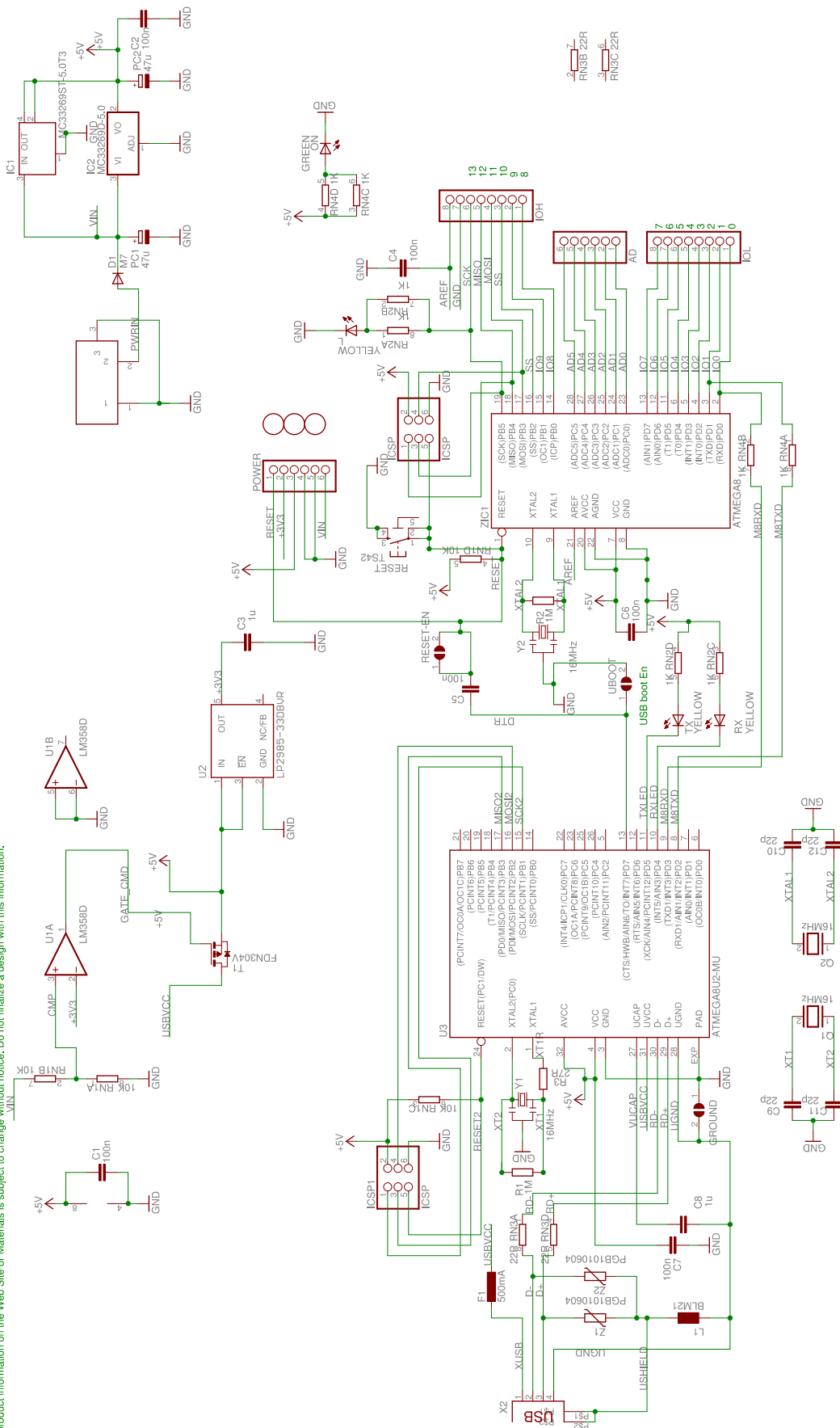


Figure 4 : schéma structurel de l'ARDUINO UNO

II. Présentation de l'Espace de développement Intégré (EDI) Arduino

A. Description de l'interface

Le logiciel Arduino a pour fonctions principales :

- de pouvoir écrire et compiler des programmes pour la carte Arduino
- de se connecter avec la carte Arduino pour y transférer les programmes
- de communiquer avec la carte Arduino

Cet espace de développement intégré (EDI) dédié au langage Arduino et à la programmation des cartes Arduino comporte :

- une **BARRE DE MENUS** comme pour tout logiciel une interface graphique (GUI),
- une **BARRE DE BOUTONS** qui donne un accès direct aux fonctions essentielles du logiciel et fait toute sa simplicité d'utilisation,
- un **EDITEUR** (à coloration syntaxique) pour écrire le code de vos programmes, avec onglets de navigation,
- une **ZONE DE MESSAGES** qui affiche indique l'état des actions en cours,
- une **CONSOLE TEXTE** qui affiche les messages concernant le résultat de la compilation du programme
- un **TERMINAL SERIE** (fenêtre séparée) qui permet d'afficher des messages textes reçus de la carte Arduino et d'envoyer des caractères vers la carte Arduino. Cette fonctionnalité permet une

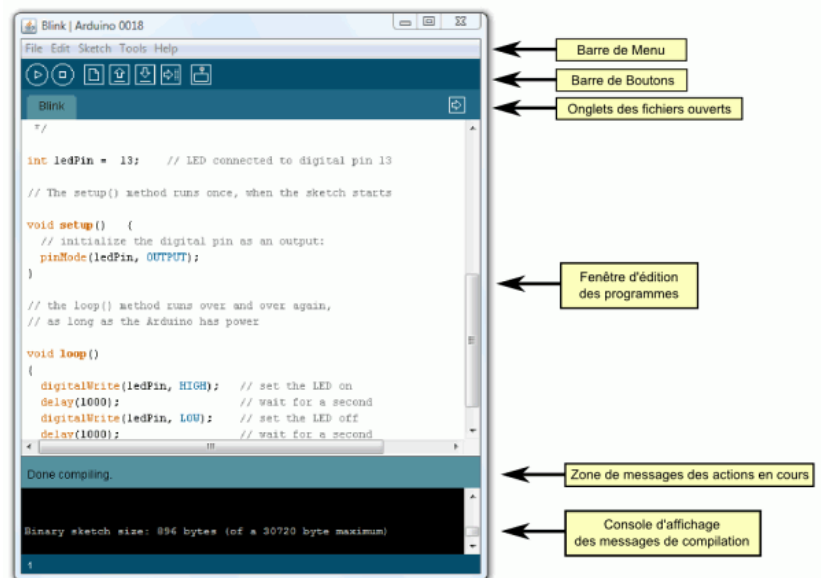


Figure 5 : présentation des éléments de l'ARDUINO software

mise au point facilitée des programmes, permettant d'afficher sur l'ordinateur l'état de variables, de résultats de calculs ou de conversions analogique-numérique : un élément essentiel pour améliorer, tester et corriger ses programmes.

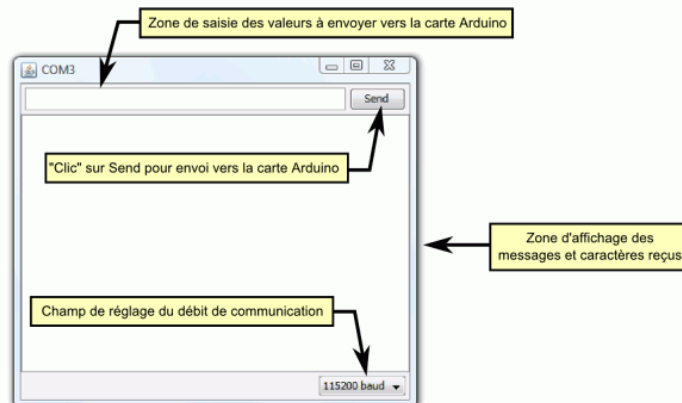


Figure 6 : module TERMINAL SERIE

B. Description de la structure d'un programme

1. Description générale des parties

Un programme utilisateur Arduino est une suite d'instructions élémentaires sous forme textuelle, ligne par ligne. La carte lit puis effectue les instructions les unes après les autres, dans l'ordre défini par les lignes de code, comme mors d'une programmation classique. Cette structure se décompose en trois parties :

- Description des constantes et variables du programme
- Fonction principale : configuration des entrées/sorties et éléments à configurer (cette partie ne sera exécutée qu'une seule fois) dans la partie *VOID SETUP()*
- Fonction boucle : description du fonctionnement général du programme (gestion des interactions entre les entrées/sorties) dans la partie *VOID LOOP()*



Figure 7 : fenêtre graphique de l'EDI

Remarque :

Il est possible d'ajouter des commentaires au programme. Pour cela on peut procéder de deux manière :

- à la fin de la ligne en ajoutant « // »
- en encadrant les commentaires entre « /* » et « */ »

2. Description détaillée des parties

a. Définition des variables et constantes

Dans cette partie, on déclare des éléments utilisés tout au long du programme : les constantes (statiques) et les variables (dynamiques). Ce sont des emplacements mémoire utilisés pour stocker des données (des valeurs) utilisables dans la suite du programme.

Variable : Une variable peut aussi bien représenter des données lues ou envoyées sur un des ports analogiques ou numériques, *une étape de calcul* pour associer ou traiter des données, que le numéro 'physique' de ces entrées ou sorties. Une "variable" n'est donc pas exclusivement un paramètre variant dans le programme. On la déclare de la façon suivante :

```
TYPE_DE_LA_DONNEE          NOM_DE_LA_DONNEE
```

Exemple : `int led`

Constante : Une constante est une variable dont la valeur est inchangeable lors de l'exécution d'un programme. On la déclare de la façon suivante :

```
CONST          TYPE_DE_LA_DONNEE          NOM_DE_LA_DONNEE
```

Exemple : `const int led`

Les différents types de données:

En programmation informatique, un type de donnée, ou simplement type, définit les valeurs que peut prendre une donnée, ainsi que les opérateurs qui peuvent lui être appliqués.

NOM DU TYPE	VALEUR MIN/MAX	TAILLE EN MEMOIRE
VALEURS BINAIRES		
boolean	0/1	1 octet
VALEURS NUMERIQUES ENTIERES SIGNEES		
int	-32 768 / +32 767	2 octets
long	-2 147 483 648 / +2 147 483 647	4 octets
VALEURS NUMERIQUES ENTIERES NON SIGNEES		
byte	0 / +255	1 octet
unsigned int	0 / +65535	2 octets
word	0 / +65535	2 octets
unsigned long	0 / +4 294 967 295	4 octets
VALEURS NUMERIQUES A VIRGULE		
float	-3.4028235E+38 / +3.4028235E+38	4 octets
double	-3.4028235E+38 / +3.4028235E+38	4 octets
CARACTERES		
char	-128 / +127 (ASCII)	1 octet

b. Fonction principale : void setup()

Cette fonction n'est exécutée qu'une seule fois au démarrage du programme. Elle permet la configuration des entrées et sorties de la carte. Les broches numériques de l'Arduino peuvent aussi bien être configurées en entrées qu'en sorties. Ici on a configuré LED_Pin_13 en sortie.

pinMode (nom, état) est une des quatre fonctions relatives aux entrées et sorties numériques que nous verrons plus bas.

```
void setup()
{
    ici se trouve la configuration des entrées et des sorties
}
```

c. Fonction boucle : void loop()

Cette fonction loop() (boucle en anglais) fait exactement ce que son nom suggère et s'exécute en boucle sans fin, permettant à votre programme de s'exécuter. Dans cette boucle, on définit les opérations à effectuer.

La fonction loop() est obligatoire, même vide, dans tout programme.

```
void loop()
{
    ici se trouve la description générale du programme en boucle
}
```

C. Compilation et programmation de l'ARDUINO

L'écriture d'un programme se déroule en plusieurs étapes.

1. Ecriture de l'algorithme

L'algorithme est une méthode pour résoudre un problème. L'algorithme est un moyen pour le programmeur de présenter son approche du problème à d'autres personnes. En effet, un algorithme est l'énoncé dans un langage bien défini d'une suite d'opérations permettant de répondre au problème donné. Un algorithme doit donc être compréhensible même par un non-informaticien.

Avant d'écrire un programme, il est donc **nécessaire** d'avoir un algorithme.

2. Ecriture du programme

La rédaction du programme se fait bien sur directement en rapport avec l'algorithme ci-dessus. Il faut absolument penser à mettre des commentaires compréhensifs par le non programmeur. Détaillé le programme et le partitionner en bloc logiques.

La rédaction du programme se fait dans la partie rayée ci-dessous :

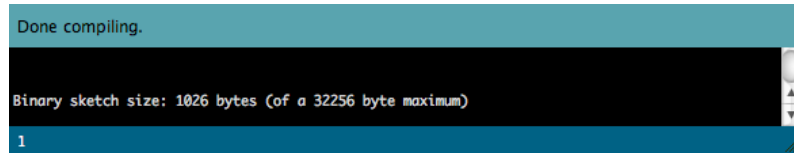
```
DEL_clignotante | Arduino 0023
DEL_clignotante
/*
 * Ce programme permet de faire clignoter la DEL présente sur la broche 13
 * de la carte ARDUINO toutes les secondes
 */
int LED_Pin_13=13;
void setup() {
    pinMode(LED_Pin_13, OUTPUT); // Configuration de la broche 13 en sortie
}
void loop() {
    digitalWrite(LED_Pin_13, HIGH); // Fixe la sortie 13 au niveau HAUT (allume la DEL)
    delay(1000); // Fixe une 1 seconde (1000ms) d'attente
    digitalWrite(LED_Pin_13, LOW); // Fixe la sortie 13 au niveau BAS (éteint la DEL)
    delay(1000); // Fixe une 1 seconde (1000ms) d'attente
}
```

3. Compilation du programme

Dans cette partie, on vérifie si le code contient des erreurs de syntaxes. En cas d'anomalie de compilation, le compilateur renseigne sur le type d'erreur et la ligne où elle se trouve.

Pour lancer la compilation, il faut appuyer sur . A ce moment-là, le bouton devient jaune et la zone de message affiche « Compiling » indiquant que la compilation est en cours.

Si la compilation se déroule sans erreur, le message « Done compiling » apparaît, suivi de la taille du programme.



Un compilateur est un programme informatique qui traduit un langage (appelé le langage source) en un autre (le langage cible), généralement dans le but de créer un fichier exécutable.

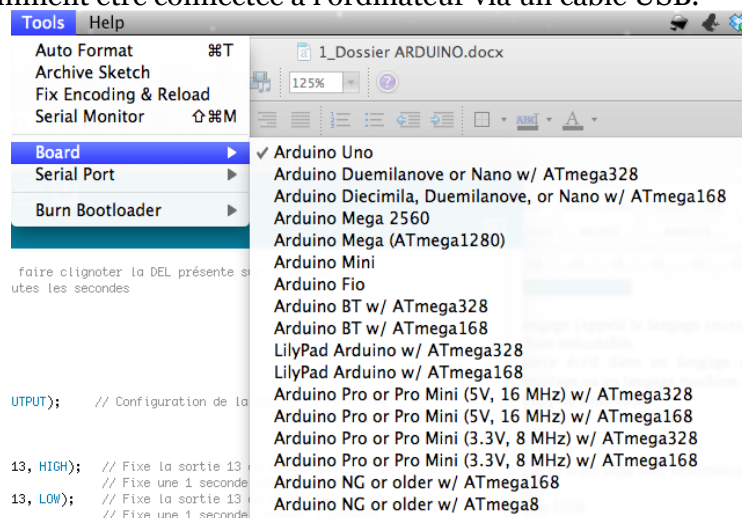
Un compilateur sert le plus souvent à traduire un code source écrit dans un langage de programmation en un autre langage, habituellement un langage d'assemblage ou un langage machine.

Le programme en langage machine produit par un compilateur est appelé code objet.

4. Sélection de la cible et du port série

Avant de transférer le programme vers la carte Arduino, il faut, si ce n'est déjà fait, sélectionner la bonne carte Arduino (la bonne cible) depuis le menu **Tools>Board** (Outils>Carte).

La carte doit évidemment être connectée à l'ordinateur via un câble USB.



Vous devez également sélectionner le bon port série depuis le menu **Tools > Serial Port** (Outils > Port Série).


Remarque :

Selon le système d'exploitation le nom du port série peut différer :

- Sous Mac, sélectionner le port /dev/tty.usbserial-1B1 (pour une carte USB)
- Sous Windows, sélectionner le port COM1, COM2 (pour une carte série) ou COM4 ou supérieur (pour une carte USB)
- Sous Linux, sélectionner le port /dev/ttyUSB0, /dev/ttyUSB1 ou équivalent.

5. Transfert du programme vers la carte ARDUINO

Une fois que vous avez sélectionné le bon port série et la bonne carte Arduino, cliquez sur le bouton

UPLOAD  (Transfert vers la carte) dans la barre d'outils, ou bien sélectionner le menu **File>Upload to I/O board** (Fichier > Transférer vers la carte).

Sur la plupart des cartes, vous devez voir les LEDs des lignes RX et TX clignoter rapidement, témoignant que le programme est bien transféré. Durant le transfert, le bouton devient jaune et le logiciel Arduino affiche un message indiquant que le transfert est en cours.

Sources

SUR INTERNET:

<http://www.arduino.cc/>

<http://www.mon-club-elec.fr/>

<http://fr.wikipedia.org/wiki/Arduino>

BIBLIOGRAPHIE:

Christian Tavernier, *Arduino Maîtriser sa programmation et ses cartes d'interface (shields)*, Paris, 2011, DUNOD

Annexes

Erreurs de syntaxes les plus courantes :

En phase d'écriture et de mise au point de votre code, répétez régulièrement les compilations comme indiqué ici afin de détecter les erreurs et corrigez-les dès qu'elles apparaissent. Les erreurs sont parfois subtiles à retrouver. Les plus fréquentes en pratique :

- oubli d'une accolade de fermeture ou d'ouverture
- oubli d'un ; de fin de ligne, ajout d'un ; après une instruction #include ou #define
- utilisation du signe = au lieu du signe == dans une condition If ou une boucle while
- utilisation d'une variable non déclarée, etc...
- Si vous ne savez pas d'où vient votre erreur, utiliser des // avant les lignes suspectes, et recompiler. Avec de l'expérience, vous aurez de moins en moins de messages d'erreur