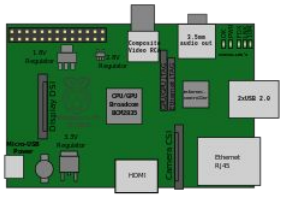
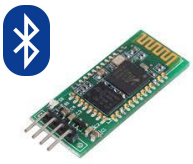
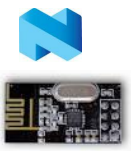
Arduino Bases

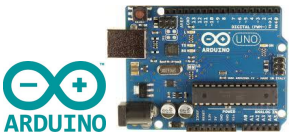
matériel et programmation

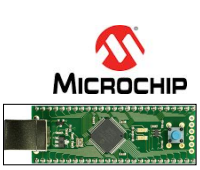
Préambule

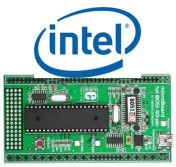










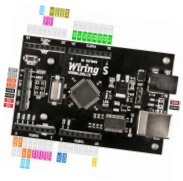


Plan

- Quelques mots des origines
- Anatomie de la carte Arduino
- Environnement matériel
- Environnement logiciel
- Premier programme
- Quelques éléments de langage

Arduino Origines

- En 2003 Hernando Barragan (à l'IDII - Interaction Design Institute Ivrea) a entrepris pour sa thèse de fin d'étude la conception de Wiring (http://wiring.org.co/) une carte de prototypage utilisant le langage de programmation inspiré de Processing (https://www.processing.org/) conçu pour des designers graphiques à partir de 2001 par Casey Reas et Ben Fry (au MIT).



- En 2005, un équipe de professeurs et d'étudiants de l'IDII (David Mellis, Tom Igoe, Gianluca Martino, David Cuartielles, Massimo Banzi et Nicholas Zambetti) ont développé le projet Arduino en s'inspirant de Wiring et de Processing.
- La grande idée est de faire simple et ouvert ! Arduino est distribué sous licence Creative Commons Attribution Share-Alike 2.5 Les schémas ainsi que les typons de circuits sont disponibles. Le code source de l'environnement de programmation et les bibliothèques embarquées sont disponibles sous licence LGPL. Le but de ces licences est de favoriser la diffusion et la création.
- http://day.arduino.cc/ : 2 avril 2016 événement mondial

http://owni.fr/2011/12/16/arduino-naissance-mythe-bidouille/
http://www.flossmanualsfr.net/arduino/ch002_historique-du-projet-arduino

Historique du projet Arduino

- Le projet Arduino est issu d'une équipe d'enseignants et d'étudiants de l'école de Design d'Interaction d'Ivrea 1 (Italie). Ils rencontraient un problème majeur à cette période (avant 2003 - 2004) : les outils nécessaires à la création de projets d'interactivité étaient complexes et onéreux (entre 80 et 100 euros). Ces coûts souvent trop élevés rendaient difficile le développement par les étudiants de nombreux projets et ceci ralentissait la mise en œuvre concrète de leur apprentissage.
- Jusqu'alors, les outils de prototypage étaient principalement dédiés à l'ingénierie, à la robotique et aux domaines techniques. Ils étaient puissants, mais leurs processus de développement étaient longs et difficiles à apprendre et à utiliser pour les artistes, les designers d'interactions et, plus généralement, pour les débutants.
- Leur préoccupation se concentra alors sur la réalisation d'un matériel moins cher et plus facile à utiliser. Ils souhaitaient créer un environnement proche de Processing, ce langage de programmation développé dès 2001 par Casey Reas 2 et Ben Fry, deux anciens étudiants de John Maeda au M.I.T., lui-même initiateur du projet DBN 3.
- En 2003, Hernando Barragan, pour sa thèse de fin d'études, avait entrepris le développement d'une carte électronique dénommée Wiring, accompagnée d'un environnement de programmation libre et ouvert. Pour ce travail, Hernando Barragan réutilisait les sources du projet Processing. Basée sur un langage de programmation facile d'accès et adaptée aux développements de projets de designers, la carte Wiring a donc inspiré le projet Arduino (2005).
- Comme pour Wiring, l'objectif était d'arriver à un dispositif simple à utiliser, dont les coûts seraient peu élevés, les codes et les plans « libres » (c'est-à-dire dont les sources sont ouvertes et peuvent être modifiées, améliorées, distribuées par les utilisateurs eux-mêmes) et, enfin, « multi-plates-formes » (indépendant du système d'exploitation utilisé).
- Conçu par une équipe de professeurs et d'étudiants (David Mellis, Tom Igoe, Gianluca Martino, David Cuartielles, Massimo Banzi ainsi que Nicholas Zambetti), l'environnement Arduino est particulièrement adapté à la production artistique ainsi qu'au développement de conceptions qui peuvent trouver leurs réalisations dans la production industrielle.
- Le nom Arduino trouve son origine dans le nom du bar dans lequel l'équipe avait l'habitude de se retrouver. Arduino est aussi le nom d'un roi italien, personnage historique de la ville « Arduin d'Ivrée », ou encore un prénom italien masculin qui signifie « l'ami fort ».

1. L'école « Interaction Design Institute Ivrea » (IDII) est aujourd'hui située à Copenhagen sous le nom de « Copenhagen Institute of Interaction Design ». ↑
2. Casey Reas était lui-même enseignant à IVREA, pour Processing, à cette époque. ↑
3. Design By Numbers, un langage de programmation spécialement dédié aux étudiants en arts visuels. ↑

Ressources documentaires

Il existe beaucoup, beaucoup de ressouces : soyez curieux, consultez-les !

- http://arduino.cc/
  Site officiel
- http://www.mon-club-elec.fr/
  Référence Arduino français
- http://eskimon.fr/
  Tutoriel détaillé (ex: Site-du-Zéro / OpenClassrooms)
- http://fr.wikiversity.org/wiki/Micro_contrôleurs_AVR/Arduino
  Tutoriel plus spécialisé
- http://www.flossmanualsfr.net/arduino/
  Tutoriel un autre....
- http://www.pmclab.fr/Les transparents de ce cours ont été inspirés, Le fablab de l'UPMC utilisateur des platines Arduino
- https://www.google.fr/search?q=arduino
  les forums et les bibliothèques de fonctions sont innombrables

et évidemment http://fr.wikipedia.org/wiki/Arduino !

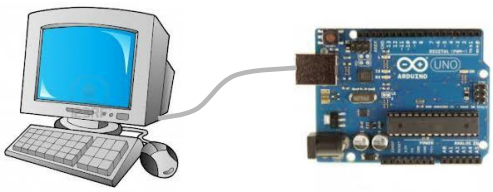
Bibliographie

Il existe aussi pleins de livres sur l'environnement Arduino, ils donnent une vision plus complète et plus détaillés et sont souvent liés à des sites contenant des tutoriels avec les sources

- Le grand livre d'Arduino (Eyrolles)
- Arduino Cookbook (O'Reilly)
- Arduino pour les nuls (First Interactive)
- Arduino - Maîtrisez sa programmation et ses cartes d'interface (Dunod)

Carte Arduino

- Microcrontrôleur AVR (ATmega328 / ATmega2560 / ATmega168) et composants complémentaires pour la programmation et l'interfaçage avec d'autres circuits.
- Chaque module possède au moins un régulateur linéaire 5 V et un oscillateur à quartz 16 MHz ou 8MHz
- Programmation par un bootloader à travers un port série RS-232 utilisant une connexion USB qui peut être sur une carte fille.
- La plupart des entrées/sorties du microcontrôleur sont déportées vers des connecteurs femelle HE14 situés sur le dessus de la carte, les modules d'extension venant s'empiler sur l'Arduino.

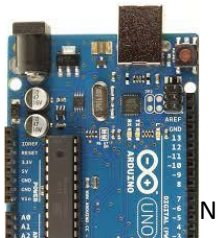


Cartes Arduino …

http://arduino.cc/en/Products.Compare

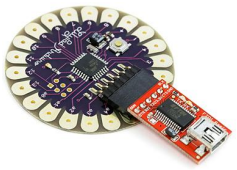
< 3.5 €

Pro-Mini -- LilyPad
ATmega168 - 16kB - 8MHz
14 ES num - 6 PWM - 6 An



Nano
ATmega328 - 32kB - 16MHz
14 ES num - 6 PWM - 8 An

< 3 €



UNO :
ATmega328 - 32kB + 2kB - 16MHz
14 ES num - 6 PWM - 6 An

Intel Galieo2
SoC 32b (pentium)

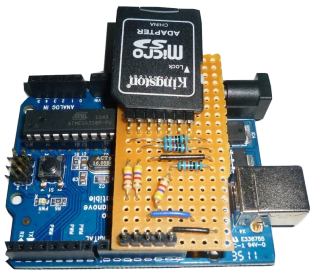
Mega :
ATmega 2560 - 256kB + 8kB - 16MHz
54 ES num - 14 PWM - 16 An

Cartes Arduino …

- 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 pour la programmation et l'interfaçage avec d'autres circuits. Chaque module possède au moins un régulateur linéaire 5 V et un oscillateur à quartz 16 MHz (ou un résonateur céramique).
- Le microcontrôleur est préprogrammé avec un bootloader pour qu'un programmateur dédié ne soit pas nécessaire.
- Les modules sont programmés par une connexion série RS-232. Les premiers Arduino possédaient un port série remplacé par l'USB. Certains modules n'ont pas d'interface de programmation, elle est alors sur un module USB-série dédié.
- L'Arduino utilise la plupart des entrées/sorties du microcontrôleur pour l'interfaçage avec les autres circuits. L'aduino uno possède 14 entrées/sorties numériques, dont 6 peuvent produire des signaux PWM, et 6 entrées analogiques. Les connexions utilisent des connecteurs femelle HE14 situés sur le dessus de la carte, les modules d'extension venant s'empiler sur l'Arduino.
- Certains modules utilisent des connecteurs mâle pour une utilisation aisée avec des plaques de test.

Arduino Sheilds

< 40€
(Amazon)



http://shieldlist.org/

Arduino Shield List

Pin usage details for 317 shields from 125 makers, and counting!

Search: GO

> Home

- 4D Systems (4)
- Adafruit Industries (9)
- AeroQuad (2)
- Andre Concalves (2)
- antrax Datentechnik (1)
- Applied Platonics (1)
- ArduCapSense (1)
- Arduino (3)
- Argent Data Systems (1)
- AsyncLabs (3)
- Batsocks (2)
- Ben Combee (1)
- Bhasha Technologies (2)
- Bliptronics (2)
- Blushing Boy (1)
- Carlos Neves (1)
- Chesters Garage (1)
- Chips To Bits (1)
- Circuit Ideas Design (1)
- Circuits At Home (2)
- CISECO (4)
- Collin Schulz (1)
- Conductive Resistance (1)
- Control Connection (1)
- Creatron Inc (1)
- Critical Velocity (5)
- Critter and Guitari (1)
- Curious Inventor (2)
- CuteDigi (8)
- Damien Good (1)
- Dexter Industries (1)
- DFRobot (16)
- Dr Michael Kroll (1)
- Dreaming Robots (1)
- DroidBuilder (1)
- DSS Circuits (1)
- Dynamic Perception (1)
- Emartee (1)
- Eric Rogers (1)
- EtherMania (1)
- Evil Mad Science (2)
- Excamera Labs (1)
- Faz Jaxton (1)
- FlamingoEDA (1)
- Freetronics (12)
- Futura Elettronica (2)
- Galileo 7 (4)
- GeekOnFire (1)
- GfxHax (1)
- GinSing (1)
- Gravitech (1)
- Homeroasters (1)
- HW Kitchen (1)
- ITead Studio (6)
- Jee Labs (1)
- Jimmie Rodgers (1)
- John Liu (3)
- Knutsel (1)
- Lars Schumann (3)
- Libelium (6)
- LinkSprite (5)
- Liquidware (12)
- Logos Electromechanical (1)
- Low Voltage Labs (1)
- Luke Weston (1)
- macetech (3)
- Maker Shed (1)
- Mark Sproul (1)
- Max Pierson (1)
- Mayhew Labs (2)
- MCI Electronics (4)
- McLaughlin Engineering (1)
- MightyOhm (2)
- Mitek (1)
- Modern Device (1)
- Multilogica (1)
- Narbotic Instruments (1)
- Neuroelec (2)
- NKC Electronics (4)
- Nootropic Design (3)
- North And Nash (1)
- Nu Electronics (9)
- Ocean Controls (1)
- Open Electronics (1)
- PDK Solutions (1)
- Photoduino (1)
- Pololu (1)
- Practical Maker (5)
- Protuino (2)
- Ray's Hobby (1)
- Renbotics (2)
- RepRap Research Foundation (1)
- Ro-Bot-X Designs (1)
- Robot Power (1)
- RobotPirate (1)
- Rocket Scream (1)
- Rogue Robotics (1)
- Rugged Circuits (6)
- Samurai Circuits (3)
- Scattered Mind (1)
- Schmelle2 (6)
- Seeed Studio (14)
- Shieldstudio (2)
- SK Pang Electronics (3)
- Small Room Labs (1)
- Smart Energy Groups (1)
- Snootlab (6)
- Solarbotics (2)
- SonikTech (1)
- Sparkfun (26)
- SpikenzieLabs (4)
- Sunhayato (2)
- SWFLTEK (1)
- Synthetos (1)
- Unified Microsystems (1)
- Unsped (1)
- Watterott (3)
- Wavical Technologies (1)
- Wayne and Layne (1)
- Wicked Device (3)
- Wingshield Industries (1)
- Wise Time (2)
- Yawp (2)
- Zach Hoeken (1)

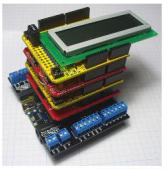
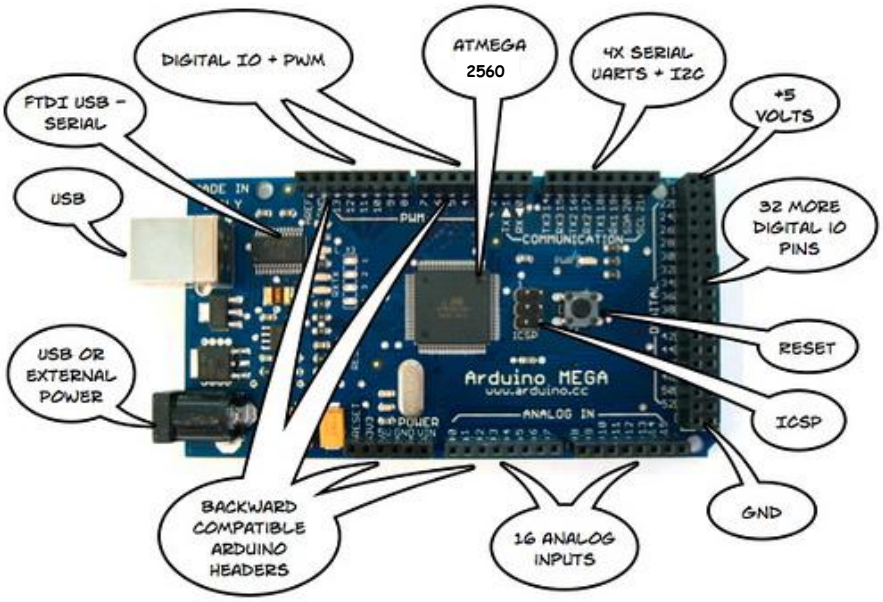
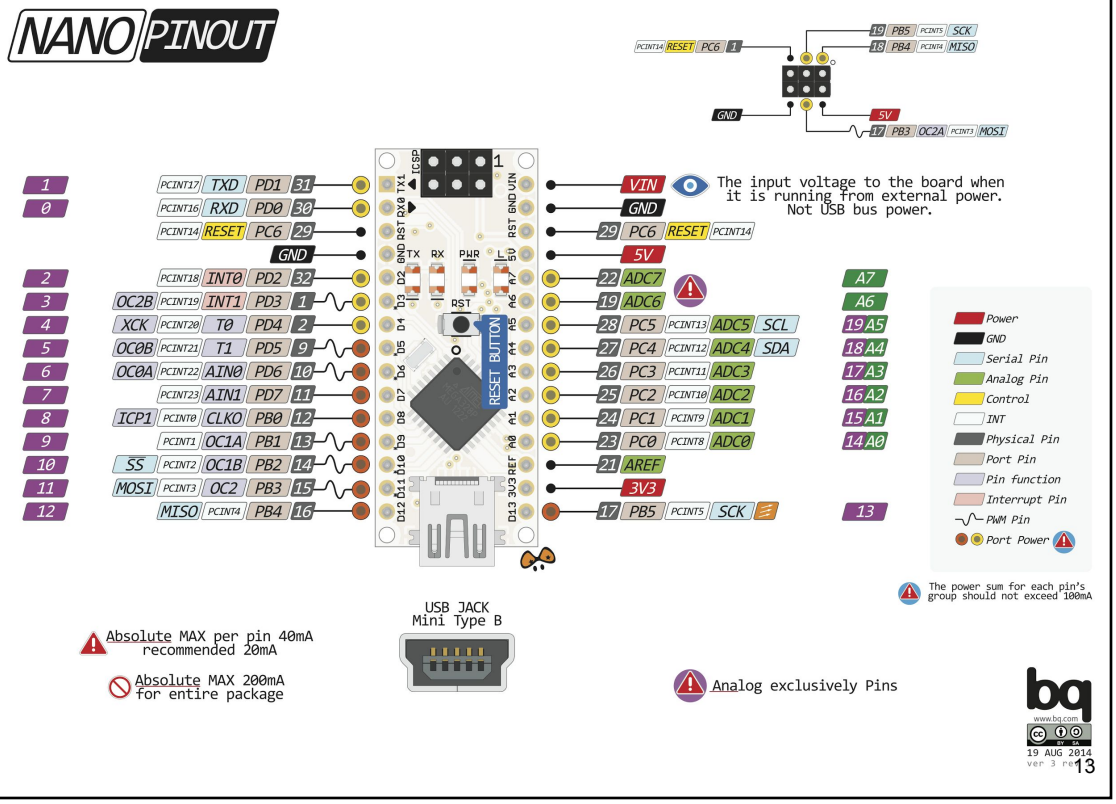


Photo: John Boxall

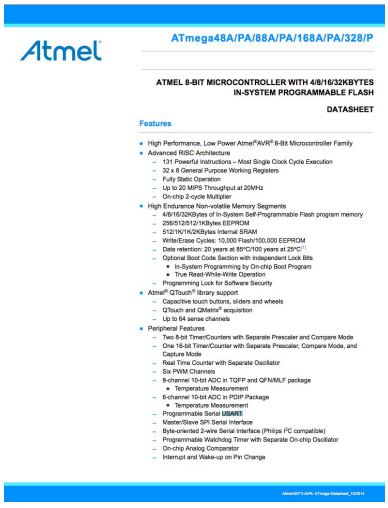
Arduino ATmega2560 : Vue d'ensemble





Arduino ATmega 2560 : Caractéristiques

Microcontrôleur :
http://www.atmel.com/ATmega328P_datasheet.pdf


Atmel

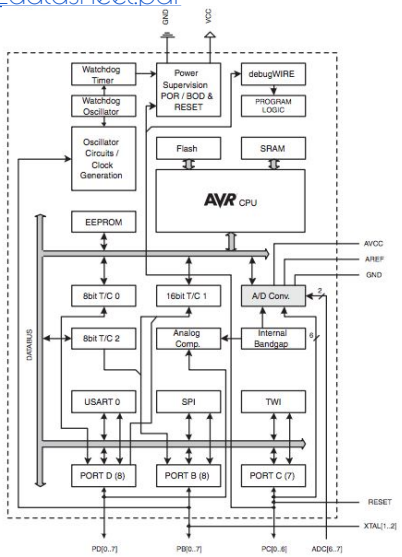
ATmega48A/PA/88A/PA/168A/PA/328/P

ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH

DATASHEET

Features

- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/1K/1K Bytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change



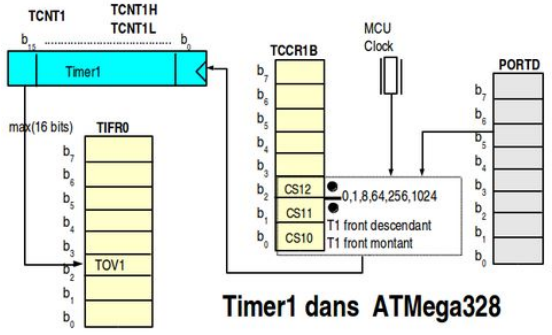
Programmation des composants internes

Tous les composants internes se programment !

- Ils faut lire leur spécification dans la documentation
- Chaque registre est adressable (l'assembleur n'est pas une fatalité)

```
TCCR1A = 0;
TCCR1B = 0;
TCNT1 = 34286;
TCCR1B |= (1 << CS12);
TIMSK1 |= (1 << TOIE1);
```

- Nous allons voir aussi qu'il existe des bibliothèques de fonctions permettant d'éviter de rentrer dans ces détails !

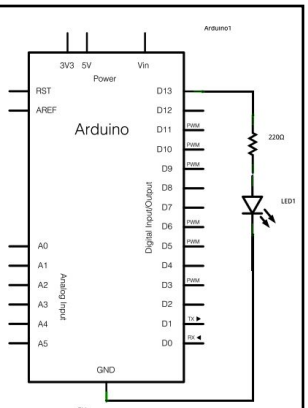


http://fr.wikiversity.org/wiki/Micro_contrôleurs_AVR/Arduino

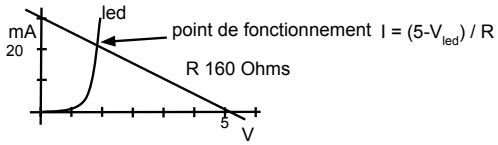
`analogWrite(broche, valeur)`

Qui produit un signal numérique (0-1-0) dont la valeur moyenne est comprise
entre 0 et 5V et donc une valeur analogique après filtrage.

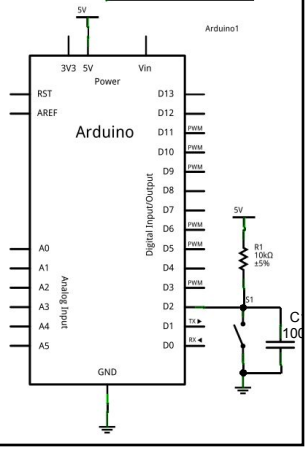
B-A-BA électronique

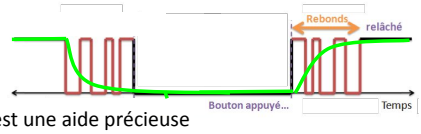


- Allumer une LED
  - Il faut que D13 soit programmé en sortie
  - que la tension programmée soit 5V
  - La tension aux bornes de LED allumée est 1.8V et le courant max est de 20mA
  - La résistance limite le courant R = $(5\text{-}V_{led})$ / I_{led}



- lire un bouton poussoir
  - Lorsque l'interrupteur est ouvert D2 est à 5V
  - Lorsqu'il se ferme la capacité se décharge
  - en R1*C1 = 10^4 (Ohms) * 10^{-7} (F) = 10^{-3} (s) = 1 ms (66%)



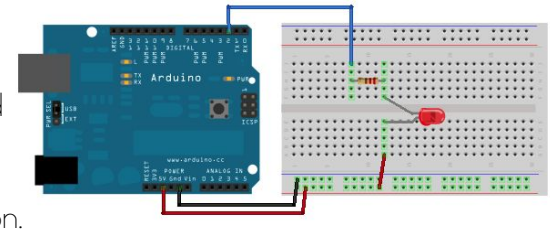


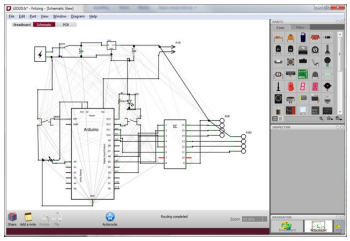
http://eskimon.fr/ est une aide précieuse

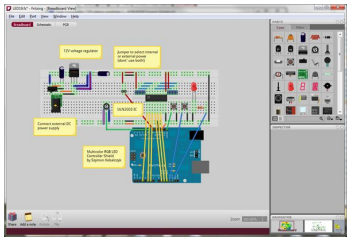
Environnement de développement matériel

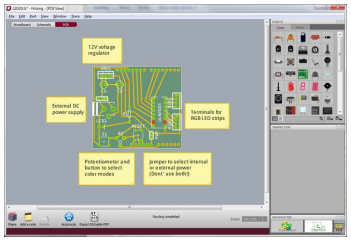
Pour utiliser la carte dans un environnement réel, il faut connecter les ports avec des composants externes.

- On peut utiliser une breadboard et des cables de connexion.
- http://fritzing.org/home/ permet de simplifier la conception.





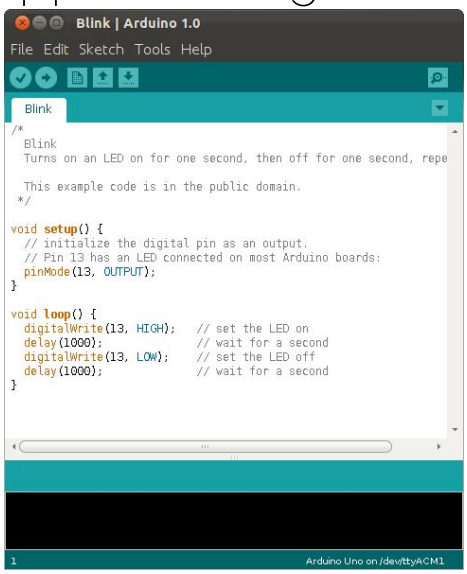




Environnement de développement logiciel

L'environnement de développement (IDE) http://arduino.cc/en/main/software

- écrit en Java (linux, windows, mac os)
  - éditeur de code
  - compilateur
  - programmateur
  - terminal de commande
- Il est possible de compiler et de charger les programmes en lignes de commande.



Langage de programmation

- C++, compilé avec `avr-g++`
- bibliothèque de développement Arduino nommée Wiring pour le contrôle des composants internes du micro-controleur.
- Un programme Arduino se nomme **sketch**, composé au minimum de 2 fonctions
  - setup() exécutée une fois pour initialiser les composants et les variables
  - loop() exécutée en boucle jusqu'à l'extinction de la carte

Menus de la fénêtre Arduino

http://www.positron-libre.com/robotique/robots/boe-shield-bot/notice/premier-programme.php

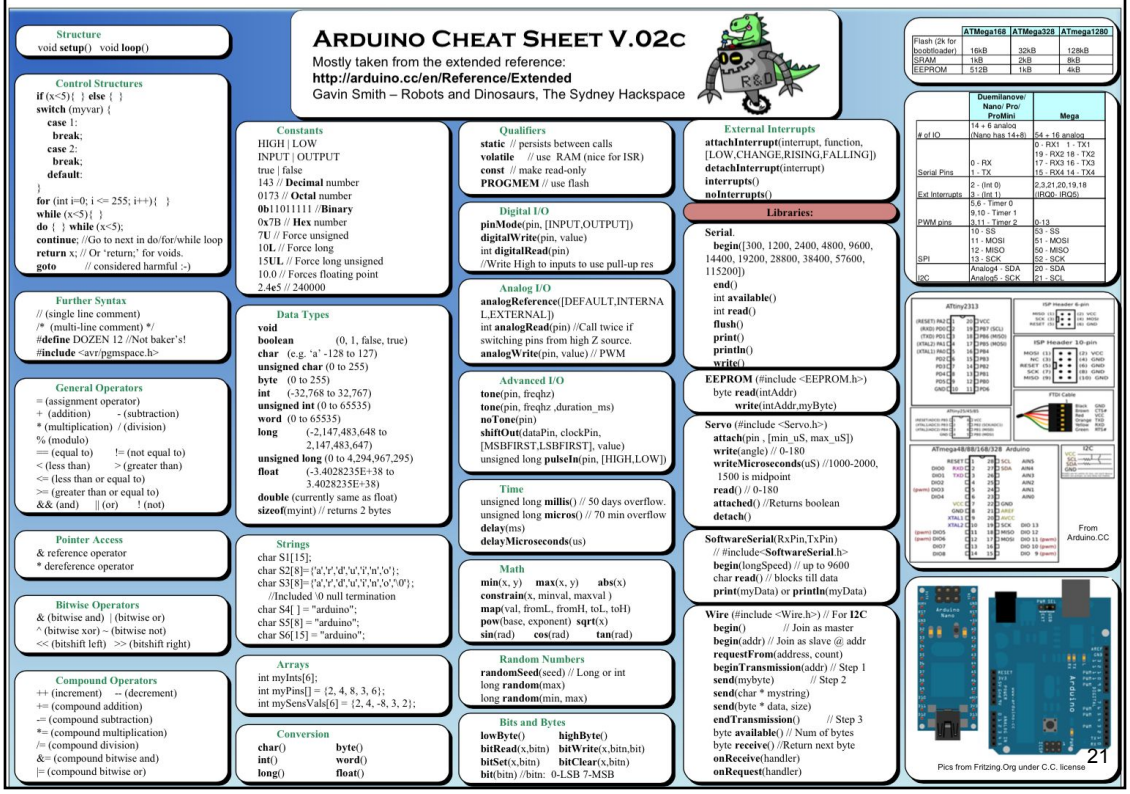
Programmation en ligne de commande

La chaîne de compilation d'Arduino utilise gcc, il est tout à fait possible de se passer de l'IDE proposé afin d'utiliser un environnement plus classique.

gvim ou emacs	Edition du code
avr-g++	Compilation et édition de liens
avr-ar	Archive de la bibliothèque
avrdude	Chargement du programme
make	Automatisation du processus de compilation
screen ou minicom	Communication série avec l'Arduino

Aide mémoire

http://www.cheat-sheets.org/



Gestion de temps 1/2

Besoins

- Faire une attente entre deux actions (action1 - wait 1s - action2)
- Faire une action périodiquement (faire toutes les 20ms : action)
- Faire une séquence d'actions à des dates précises
  (à chaque date[i] faire action[i])
- Mesurer le temps qui sépare deux actions
- Veillez qu'une action ait été effectuée avant une échéance.
- Démarrer une action dans un délai maximum après un événement.

	delay attente active	time counter scrutation	timers interruption
Attente entre 2 actions	oui mais	oui pas précis	oui
Action périodique	non	oui pas précis	oui
Séquences d'actions temporelles	oui	oui pas précis	oui
Action avant échéance	non	oui pas précis	oui
Action après événement	non	oui pas précis	oui

Gestion de temps 2/2

Attente active : le programme n'avance plus mais le processeur est toujours sensible aux interruptions.

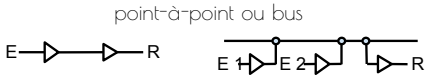
- delay(unsigned long ms)
  - 1ms à 4Gi ms (= 50 jours)
- delayMicroseconds(unsigned int us)
  - 3 µs à 16883µs (= 16ms)

Time Counter (utilise le timer0 interne)

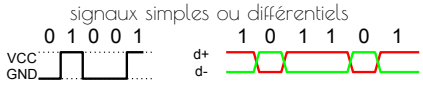
- unsigned long millis()
  - retourne le nombre de ms écoulées depuis le début (max 50j)
- unsigned long micros()
  - retourne le nombre de µs écoulées depuis le début (max 1h11m)

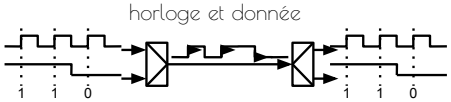
Liaisons série 1/3

- RS232
  - full duplex,
  - pas de signal d'horloge
  - 2 data (3 fils mininum : RX, TX, GND),
  - signal non différentiel
  - point à point
  - de 75 bits/s à 115 kb/s
- I2C ls / hs
  - half duplex
  - horloge et data séparé (3 fils : SDA, SCL, GND),
  - signal non différentiel,
  - bussé
  - 100 kb/s à 3.4 Mb/s



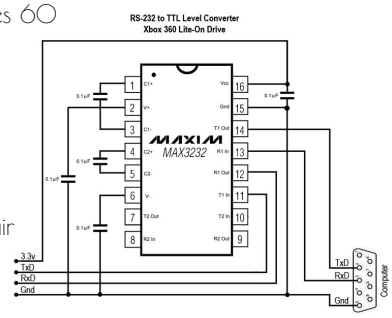
- SPI
  - full duplex,
  - horloge et data séparés
    (4 fils mininum : SCLK, MISO, MOSI, SS)
  - signal non différentiel
  - point à point
  - adhoc jusqu'à 100Mb/s



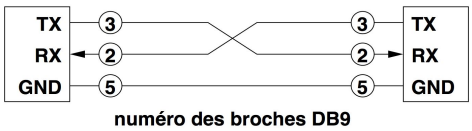


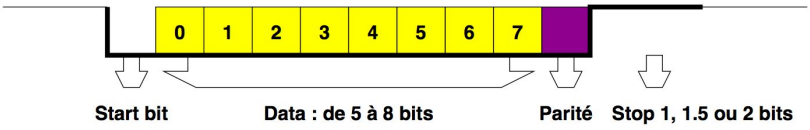
Liaisons série 2/3

- Protocole faible débit, simple et très diffusé, datant des années 60
- Pas d'horloge: l'émetteur et le récepteur s'entendent avant.
- Protocole *handshake* optionnel : RTS (request), CTS (clear), ...
- Liaison point-à-point, pas de notion d'adresse.
- Trame de données de 5 à 8 bits avec parité.
- La parité est optionnelle:
  parité paire:
    nombre de 1 de la donnée et du bit de parité doit être pair
  parité impaire: c'est le contraire



- RS232 prévoit plusieurs types de cablages:
  le cablage null-modem définit la communication entre 2 terminals
- au minimum 3 fils : TX, RX et GND
- on peut avoir besoin d'un convertisseur
  de niveaux électriques :
  - 0 logique : +8 à +12V
  - 1 logique : -8 à -12V





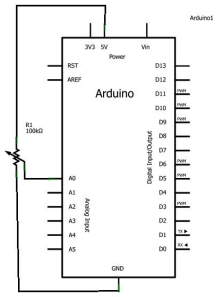
Liaisons série 3/3

Arduino nano série à un port serial

- serial.begin(speed[,config]) → speed: 300..9600..115200, config: SERIAL_8N1
- serial.end() → ferme la connexion
- serial.available() → rend le nombre de char en attente (jusqu'à 64)
- serial.read() → retire et rend le caractère en attente ou -1
- serial.peek() → rend le caractère en attente ou -1
- serial.flush() → vide le buffer en sorties
- serial.print(var[,base|frac]) → imprime la variable (int, char, string)
  base : BIN, OCT, DEC, HEX
  frac : nombre de chiffres après la virgule
- serial.println() → comme print + '\n' (ascii 10)
- serial.write(val) → val entier ou string rend le nombre de char écrits
- serial.write(buf, len) → buf pointeur, len size
- serialEvent() → appelée quand il y a une donnée présente

Conversion Analogique Numérique 1/3

- 10-bit Resolution
- 1 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13µs - 260µs Conversion Time
- Up to 76.9kSPS (Up to 15kSPS at Maximum Resolution)
- 8 Multiplexed Single Ended Input Channels
- 1 Comparator (**comparaison de deux entrées**)
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete



Conversion Analogique Numérique 2/3

anologReference(REF)

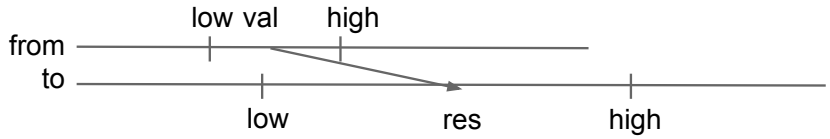
REF : DEFAULT, INTERNAL1V1, EXTERNAL

int analogRead(int pin)

0 à 7

resultat sur 10bits 0 à 1023

int map(value, fromLow, fromHigh, toLow, toHigh)

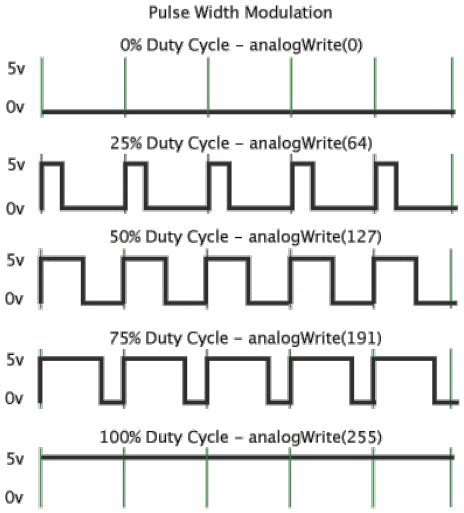


Conversion Numérique Analogique 3/3

L'Arduino ne dispose pas de convertisseur mais en numérique.
On peut produire un signal modulé en largeur d'impulsion : PWM

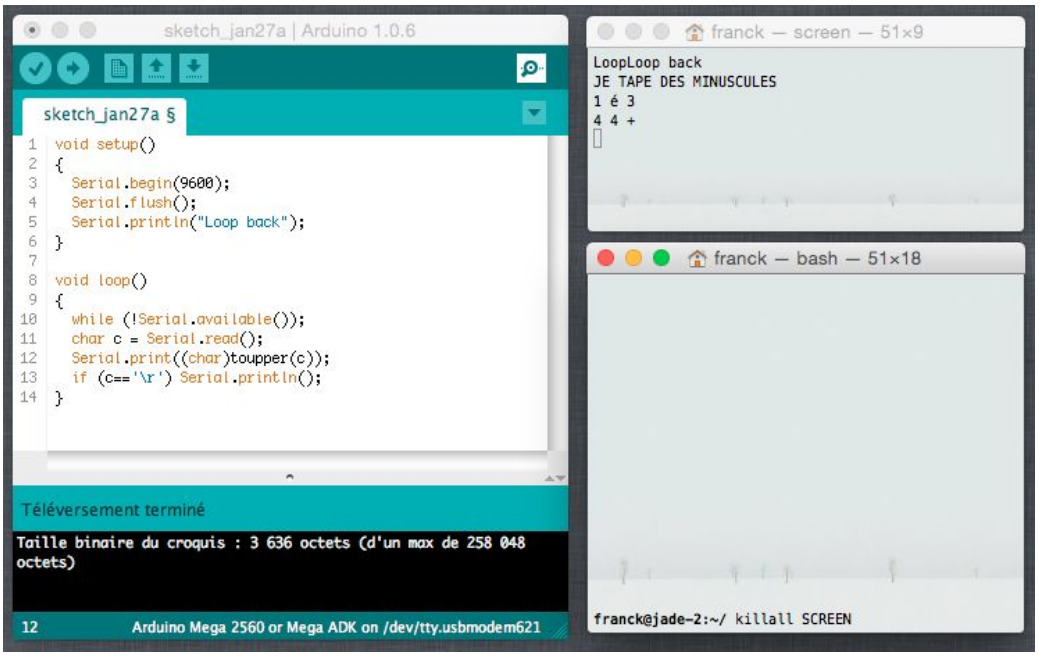
En Arduino, la fonction
`void analogWrite(int pin, int value)`

- Pin
  - Arduino nano 3, 5, 6, 9, 10, 11
- Fréquence
  - 490Hz
- Précision
  - 8 bits = 256 états différents



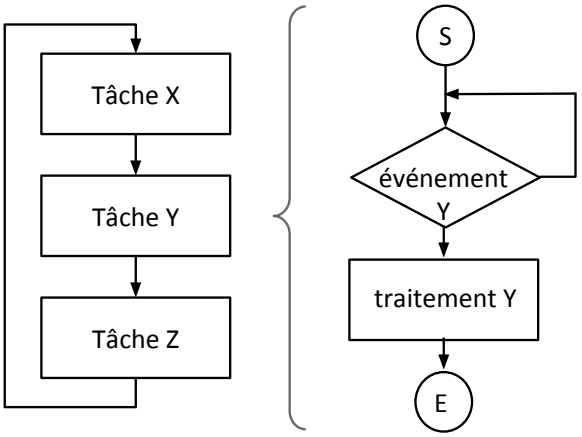
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.AnalogWrite

Exemple de programme



Problème : exécution de plusieurs tâches

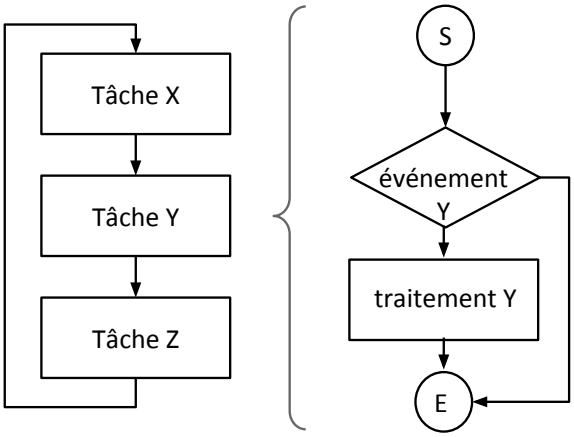
Un grand nombre des tâches destinées à un microcontrôleur dépendent d'événements externes périodiques ou non.



Si l'évément est long à venir cela bloque l'exécution de toutes les tâches.

Sol. :si les tâches coopèrent, on attend pas

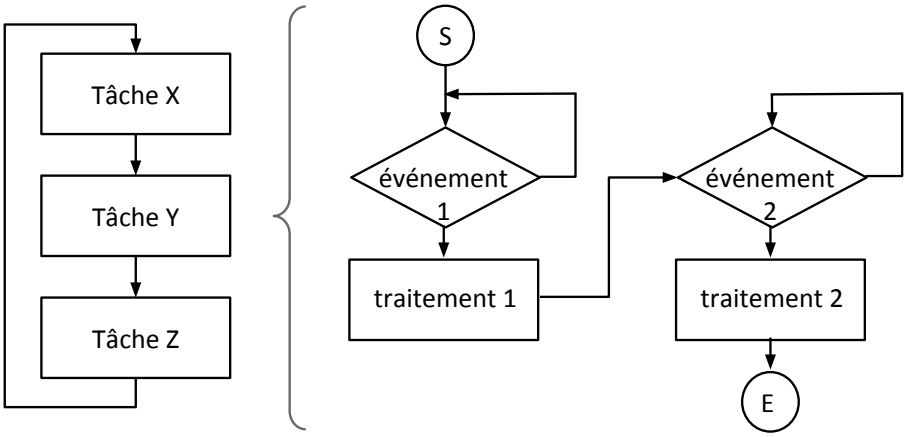
Si on peut garantir que les traitements sont bornés, alors il n'est pas obligatoire d'attendre l'événement.



Les tests des taches vont être multiplés. Les tâches ne se gènent plus.

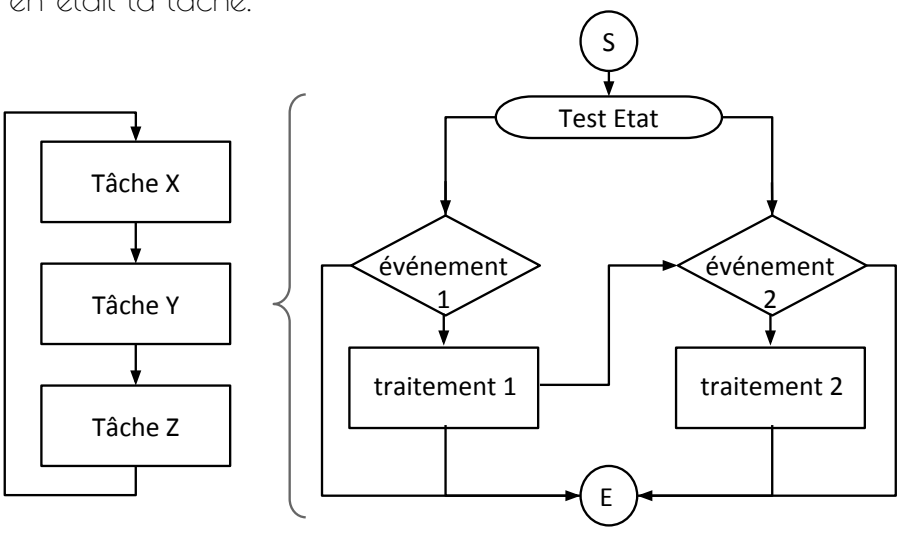
Une tâche peut avoir plusieurs traitements

Quand l'événement 1 arrive, il faut exécuter le traitement 1
Quand l'événement 2 arrive, il faut exécuter le traitement 2

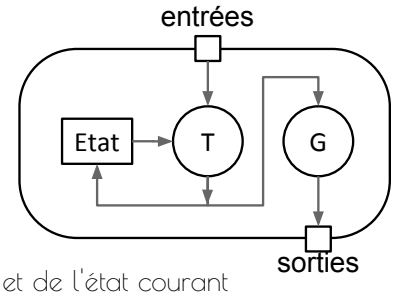


Solution: les tâches doivent avoir des états

Si on n'attend pas les événements, il faut se souvenir où en était la tâche.



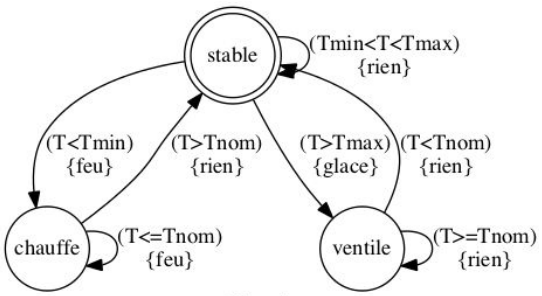
FSM : Machine à états finis



Une FSM (automate) est défini par :

- un nombre d'états finis
- un état initial
- un état courant
- une fonction de transition d'états définissant l'état futur à partir des entrées et de l'état courant
- une fonction de génération définissant les sorties à partir de l'état courant ou futur

On décrit le comportement d'uns FSM avec son graphe de transition d'états



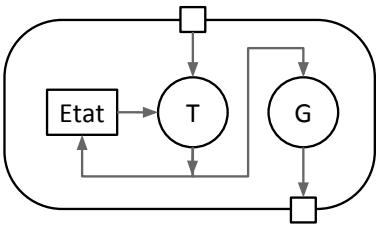
Climatiseur

INPUTS
3 températures : Tmin - Tnom - Tmax (p.ex. 18° 19° 20°)
la tempéraure T

ACTIONS
feu : {ventilateur = OFF; radiateur = ON;}
glace : {ventilateur = ON; radiateur = OFF;}
rien : {ventilateur = OFF; radiateur = OFF;}

Une tâche est une machine d'états finis

Une tâche à la forme suivante



Aller à l'état courant (p. ex. Etat E0)
E0:
    Si événement attendu
    Alors Faire
        G : traitement
        T : en fonction des entrées
            - Calcul de l'état suivant
    Fin faire
E1:
    [...]

Aller à l'état courant (p. ex. Etat E0)
E0:
    Si événement attendu
    Alors Faire
        en fonction des entrées
            - T : Calcul de l'état suivant
            - G : traitement
    Fin faire
E1:
    [...]