# Arduino et GRBL - l'incontournable solution pour piloter une petite CNC

En Juin 2011, je publiais un **premier billet** sur GRBL, qui annonçait une solution très prometteuse de pilotage de CNC par Arduino. Deux an et demi plus tard, GRBL s'est imposé comme une solution très économique, simple d'emploi et malgré tout très puissante pour piloter une CNC. Une petite mise à jour de ce billet s'imposait afin de décrire les avancées du logiciel et voir comment l'installer.

Edit : Billet remis encore à jour au 31 Mai 2015 pour tenir compte des deniers changements....

## Choix de la version

Beaucoup d'améliorations sont régulièrement faites sur GRBL. J'ai donc choisi de documenter comment travailler sur la dernière version extraite de GitHub.

Cette version dispose d'une implémentation particulièrement complète du Gcode, d'un support de 3 boutons (Start / Feedhold / stop), des interrupteurs d'origine machine sur 3 axes, de la gestion du démarrage de la broche, de son sens, de l'arrosage, etc.. Elle a fait l'objet d'optimisations qui permettent de gérer des vitesses de pas jusqu'à 30Khz par axe... Depuis peu, elle supporte également une broche pilotée en vitesse avec du PWM (Pulse Width Modulation).

Impressionnant.

Elle supporte notamment :

- Un fusible ré-armable pour protéger les cartes moteurs
- Un connecteur à vis pour brancher l'alimentation 12 à 36V
- 4 contrôleurs moteur pas à pas
- Le 4e contrôleur peut être câblé soit en 4e axes (support expérimental sous Grbl, via les broches D12 et D13, c'est à dire à la place des 2 broches de commande de la broche), soit pour cloner les signaux d'un axe donné (X, Y ou Z au choix par jumper). C'est très pratique pour piloter une machine qui dispose de 2 moteurs sur un même axe (ex : Shapeoko).
- Supporte des drivers jusqu'au DRV8825, qui supportent le 32e de pas.
- Les signaux de commande moteur sont également tous disponibles sur un connecteur spécifique pour permet le test et la visualisation (ex:: oscilloscope) à des fins de debug
- Support de 3 interrupteurs de détection de limite machine, qui sont disponibles sur 2 connecteurs sur la carte, permettant de câbler un limit+ et un limit - sur chaque axe. A noter qu'il n'y a en fait physiquement qu'un seul signal limite , "Limit X-" et "Limit X+" sont donc une vue de l'esprit, et arrivent sur la même fonction de GRBL : "LimiteX".
- Support de la commande de broche (M/A) et de son sens (CW/CCW) sur 2 broches séparées
- Support d'une commande de pompe d'arrosage (coolant)
- Support pour 3 boutons externes : "Pause/Hold", "Resume/Run" et "Stop"

- Support pour un poussoir d'arrêt d'urgence, câblé sur le Reset de l'Arduino : il n'y a pas plus efficace pour arrêter un usinage en cours...
- Sortie sérielle TTL disponible pour extension ultérieure
- Sortie I2C disponible pour extension ultérieure

Notes importantes :

- Cette shield n'est pas compatible avec la commande de broche en PWM. En effet, lors de l'activation de cette fonction de contrôle PWM, GRBL permute les fonctions "Z limit" et "Spindle Enable". Du coup la sérigraphie de la shield est fausse... Si vous savez vus accomoder de ce petit défaut et que vous avez vraiment besoin de cette fonction, vous trouverez plus d'infos ici : https://github.com/grbl/grbl/wiki/Connecting-Grbl
- Il n'y a pas de sortie dédiée pour mettre un détecteur de longueur d'outil. Il est toutefois possible d'en raccorder un entre les signaux "Scl" et "Gnd" du connecteur I2c. Testé et cela marche... (cf plus bas).
- Pensez à mettre des radiateurs sur les Pollolu, et à les refroidir avec un ventilateur...
- Pour des moteurs de plus de 1.5A, privilégier les drivers "DRV8825" qui permettent de délivrer plus de courant

Cette shield est vendue sur le site de Protoneer en nouvelle Zélande, mais on la trouve désormais également sur le site Banggood ( seule : ICI ou ICI , en pack avec drivers et arduino ICI

# Configuration :

Il va tout d'abord falloir récupérer la dernière version de l'environnement de développement Arduino sur le site officiel (**http://arduino.cc/en/Main/Software** ) et l'installer. Je ne donne pas de mode opératoire détaillé, celui-ci étant déjà très largement documenté.

L'environnement étant installé, nous allons ensuite paramétrer l'ordinateur pour pouvoir utiliser le compilateur C pour AVR en ligne de commande. Cette manipulation est nécessaire, car le programme que nous allons envoyer à l'Arduino n'est pas un sketch Arduino, mais bien un programme en C AVR, optimisé pour le processeur. Mais afin de nous éviter d'installer une chaine de compilation complète, nous allons utiliser cette qui est installée dans l'environnement Arduino, et qui est utilisée lors de la compilation de Sketches de façon transparente par celui-ci.

Pour ce faire, il va nous falloir ajouter ce chemin à la variable d'environnement $PATH de notre OS.

**S**ur Mac, éditer le ficher .bash_profile dans la racine de votre compte et ajouter les lignes suivantes :

```
# pour utiliser l'environnement Arduino comme compilateur avr

export
PATH=$PATH:/Applications/Arduino.app/Contents/Resources/Java/hardware/tools
/avr/bin
```

puis lancer la main la commande pour éviter de relancer un shell

```
export
PATH=$PATH:/Applications/Arduino.app/Contents/Resources/Java/hardware/tools
/avr/bin
```

**Sur Windows**, faire :

- Appuyer simultanément sur les touches "Windows" + "Pause"
- aller dans l'onglet "Avancé",
- cliquez sur le bouton "variables d'environnement"
- Modifier la variable d'environnement PATH pour ajouter ";C:\Program Files\arduino-0022\hardware\tools\avr\bin;C:\Program Files\arduino\hardware\tools\avr\utils\bin" à la fin.
- Noubliez pas le ';' de séparation :

**Sur Linux** éditer le ficher .bash_profile dans la racine de votre compte et ajouter de quoi appeler le sous répertoire bin contenant les binaires du compilateur AVR.

# Récupération du firmware

Nous allons récupérer les sources à partir de la branche "Master" de GRBL (**https://github.com/grbl/grbl** ). Pour ce faire, **téléchargez l'archive** et la décompresser sur le bureau par exemple.

# Configuration

Nous allons avoir 4 fichiers à configurer (dont un à créer):

- defauts/ma_machine.h : Fichier à créer, avec une configuration propre à "ma machine"
- default.h , afin d'ajouter l'appel du fichier de configuration correspondant à "ma machine"
- config.h pour de multiples raisons :
- Afin d'activer la configuration que nous venons de créer "(ma machine")
- Afin que la machine, après homing, soit en X=0,Y=0
- Enfin, afin de désactiver la gestion de commande de broche en PWM (activé par défaut)
- Makefile afin de paramétrer l'interface de programmation pour qu'elle puisse utiliser le booloader Arduino. Ceci nous évitera d'utiliser un programmateur spécialisé comme un AVRISP, AVRTInyUSB ou USBasp.

# default.h

Ouvrez d'abord default.h. Afin de travailler proprement, nous allons ajouter un bloc avec defaut_GENERIC :

```
#ifdef DEFAULTS_MAMACHINE

  // Grbl  default settings for My machine

  #include "defaults/defaults_mamachine.h"

#endif
```

```
#ifdef DEFAULTS_GENERIC

...
```

# defaults/defaults_mamachine.h

Dupliquez le fichier defaults_generic.h du sous répertoire /defaults et nommez le defaults_mamachine.h ou créez le votre en fonction de votre configuration. Voici ce que cela donne pour une machine réalisée au Fablab Labsud :

```
  // Grbl for Labsud CNC

  #define MOTOR_STP 200 // number of step per turn

  #define MICRO_STP 16  // Micro stepping 1/16 step

  #define SCREW_PITCH_MM 1.25 // Screw pitch in mm

  //

  #define DEFAULT_X_STEPS_PER_MM (MICRO_STP*MOTOR_STP/SCREW_PITCH_MM)

  #define DEFAULT_Y_STEPS_PER_MM (MICRO_STP*MOTOR_STP/SCREW_PITCH_MM)

  #define DEFAULT_Z_STEPS_PER_MM (MICRO_STP*MOTOR_STP/SCREW_PITCH_MM)

  #define DEFAULT_X_MAX_RATE 380.0 // mm/min

  #define DEFAULT_Y_MAX_RATE 380.0 // mm/min

  #define DEFAULT_Z_MAX_RATE 220.0 // mm/min

  #define DEFAULT_X_ACCELERATION (50.0*60*60) // 50*60*60 mm/min^2 = 50
mm/sec^2

  #define DEFAULT_Y_ACCELERATION (50.0*60*60) // 50*60*60 mm/min^2 = 50
mm/sec^2

  #define DEFAULT_Z_ACCELERATION (50.0*60*60) // 50*60*60 mm/min^2 = 50
mm/sec^2

  #define DEFAULT_X_MAX_TRAVEL 200.0 // mm

  #define DEFAULT_Y_MAX_TRAVEL 200.0 // mm

  #define DEFAULT_Z_MAX_TRAVEL 80.0 // mm

  #define DEFAULT_STEP_PULSE_MICROSECONDS 10

  #define DEFAULT_STEPPING_INVERT_MASK 0

  #define DEFAULT_DIRECTION_INVERT_MASK 1

  #define DEFAULT_STEPPER_IDLE_LOCK_TIME 25 // msec (0-254, 255 keeps
steppers enabled)
```

```
  #define DEFAULT_STATUS_REPORT_MASK
((BITFLAG_RT_STATUS_MACHINE_POSITION)|(BITFLAG_RT_STATUS_WORK_POSITION))

  #define DEFAULT_JUNCTION_DEVIATION 0.02 // mm

  #define DEFAULT_ARC_TOLERANCE 0.002 // mm

  #define DEFAULT_REPORT_INCHES 0 // false

  #define DEFAULT_AUTO_START 1 // true

  #define DEFAULT_INVERT_ST_ENABLE 0 // false

  #define DEFAULT_INVERT_LIMIT_PINS 0 // false

  #define DEFAULT_SOFT_LIMIT_ENABLE 0 // false

  #define DEFAULT_HARD_LIMIT_ENABLE 0  // false

  #define DEFAULT_HOMING_ENABLE 1  // false

  #define DEFAULT_HOMING_DIR_MASK 3 // move positive dir

  #define DEFAULT_HOMING_FEED_RATE 220.0 // mm/min

  #define DEFAULT_HOMING_SEEK_RATE 300.0 // mm/min

  #define DEFAULT_HOMING_DEBOUNCE_DELAY 250 // msec (0-65k)

  #define DEFAULT_HOMING_PULLOFF 1.0 // mm

#endif
```

Dans cette configuration, j'ai activé la gestion des interrupteurs de détection d'origine machine (homing), et paramétrés le nombre de pas par mm de chaque axe aindi que les accélérations et course de la machine (200 X 200 X 80mm). Les moteurs sont en 1/16 de pas.

# config.h

Nous allons ensuite editer config.h, afin d'activer cette nouvelle configuration : Cela se fait très simplement en remplacant

```
#define DEFAULTS_SHERLINE_5400
```

par

```
#define DEFAULTS_MYMACHINE
```

Il faut également, dans le même fichier, désactiver le PWM :

```
//#define VARIABLE_SPINDLE // Default enabled. Comment to disable.
```

Enfin, il faut activer le forçage de l'origine en 0,0 après homing:

```
#define HOMING_FORCE_SET_ORIGIN // Uncomment to enable.
```

# Makefile

Enfin, nous allons éditer le fichier Makefile afin d préciser comment nous allons programmer la carte. Pour ce faire, il va nous falloir :

Remplacer la ligne :

```
PROGRAMMER = -c arduino -P usb
```

par (sous windows)

```
PROGRAMMER = -C "C:\Program Files\arduino-
0022\hardware\tools\avr\etc\avrdude.conf" -c stk500v1 -P COM5 -b115200
```

ou (sous OSX)

```
PROGRAMMER = -c arduino -P /dev/tty.usbserial-A7006R7r -b115200 (pour une
Arduino Uno)
```

OU

```
PROGRAMMER = -c ardjino -P /dev/tty.usbmodem1421 -b57600  (pour une Arduino
Duelinanove)
```

Bien sur le nom du port série (derrière option -P) est à remplacer en fonction de votre config...

**Note importante : La vitesse de programmation (-b57600) dépend de la carte. Elle est de 57600 pour une Duemilanove ou équivalent (carte avec un composant série "FTDI232") et de 115200 pour une carte Arduino Uno ou équivalent (carte avec composant série ATMEGA8U2).**

# Compilation et programmation du firmware

Lancer une ligne de commande (commandes MSDOS, ou terminal sur OSX ou Linux), et se placer dans le répertoire dans lequel vous avez décompressé et adapté le logiciel, puis faire "make"

```
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c main.c -o main.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c motion_control.c -o motion_control.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c gcode.c -o gcode.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c spindle_control.c -o spindle_control.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c coolant_control.c -o coolant_control.o
```

```
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c serial.c -o serial.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c protocol.c -o protocol.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c stepper.c -o stepper.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c eeprom.c -o eeprom.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c settings.c -o settings.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c planner.c -o planner.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c nuts_bolts.c -o nuts_bolts.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c limits.c -o limits.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c print.c -o print.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-c report.c -o report.o

avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -I. -ffunction-sections
-o main.elf main.o motion_control.o gcode.o spindle_control.o
coolant_control.o serial.o protocol.o stepper.o eeprom.o settings.o
planner.o nuts_bolts.o limits.o print.o report.o -lm -Wl,--gc-sections

rm -f grbl.hex

avr-objcopy -j .text -j .data -O ihex main.elf grbl.hex

avr-size --format=berkeley main.elf

   text          data     bss     dec     hex filename

  29842             4    1414   31260    7a1c main.elf
```

Ensuite, lancer la programmation de la carte avec un "make flash"

```
avrdude -c stk500v1 -P /dev/tty.usbserial-A7006R7r -b115200 -p atmega328p -
B 10 -F -U flash:w:grbl.hex:i



avrdude: AVR device initialized and ready to accept instructions



Reading | ################################################## | 100% 0.01s



avrdude: Device signature = 0x1e950f
```

```
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be
performed

         To disable this feature, specify the -D option.

avrdude: erasing chip

avrdude: reading input file "grbl.hex"

avrdude: writing flash (29846 bytes):


Writing | ################################################## | 100% 8.59s


avrdude: 29846 bytes of flash written

avrdude: verifying flash memory against grbl.hex:

avrdude: load data flash data from input file grbl.hex:

avrdude: input file grbl.hex contains 29846 bytes

avrdude: reading on-chip flash data:


Reading | ################################################## | 100% 6.35s


avrdude: verifying ...

avrdude: 29846 bytes of flash verified


avrdude: safemode: Fuses OK


avrdude done.  Thank you.
```

## TESTS DE BON FONCTIONNEMENT DU FIRMWARE

Lancer l'environnement arduino, sélectionnez le port série sur lequel est raccordé la carte et lançez la console série à l'aide de l'icone totalement à droite. Passez la vitesse à 115200 bauds en bas à droite et appuyez sur le bouton reset de l'Arduino.

GRBL devrait vous accueillir avec sa bannière :

```
Grbl 0.9c (20131231) ['$' for help]

['$H'|'$X' to unlock]
```

Remarque importante : le texte "'$X' to unlock' est affiché car le homing est activé. Tant que la recherche d'origine machine ne sera pas faite, il ne sera pas possible de lancer un programme d'usinage. Onc si vous n'avez pas de switch, il vaudra mieux la désactiver en mettant fans default.h :

```
  #define DEFAULT_HOMING_ENABLE 0  // false
```

Cela peut être également fait à l'aide de l'interpréteur en tapant :

```
$26=1
```

Vous pouvez jouer avec l'interpréteur en tapant **$** et entrée pour avoir l'aide :

```
$$ (view Grbl settings)

$# (view # parameters)

$G (view parser state)

$N (view startup blocks)

$x=value (save Grbl setting)

$Nx=line (save startup block)

$C (check gcode mode)

$X (kill alarm lock)

$H (run homing cycle)

~ (cycle start)

! (feed hold)

? (current status)

ctrl-x (reset Grbl)
```

La commande **$$** permet de vérifier que les réglages par défaut que vous avez créé pour votre machine ont bien étés récupérés :

```
$0=1600.000 (x, step/mm)

$1=1600.000 (y, step/mm)

$2=1600.000 (z, step/mm)

$3=500.000 (x max rate, mm/min)

$4=500.000 (y max rate, mm/min)

$5=500.000 (z max rate, mm/min)

$6=10.000 (x accel, mm/sec^2)

$7=10.000 (y accel, mm/sec^2)
```

```
$8=10.000 (z accel, mm/sec^2)

$9=200.000 (x max travel, mm)

$10=200.000 (y max travel, mm)

$11=100.000 (z max travel, mm)

$12=10 (step pulse, usec)

$13=250.000 (default feed, mm/min)

$14=0 (step port invert mask, int:00000000)

$15=192 (dir port invert mask, int:11000000)

$16=25 (step idle delay, msec)

$17=0.020 (junction deviation, mm)

$18=0.005 (arc tolerance, mm)

$19=3 (n-decimals, int)

$20=0 (report inches, bool)

$21=1 (auto start, bool)

$22=0 (invert step enable, bool)

$23=0 (invert limit pins, bool)

$24=1 (soft limits, bool)

$25=1 (hard limits, bool)

$26=1 (homing cycle, bool)

$27=0 (homing dir invert mask, int:00000000)

$28=25.000 (homing feed, mm/min)

$29=500.000 (homing seek, mm/min)

$30=250 (homing debounce, msec)

$31=1.000 (homing pull-off, mm)

ok
```

Les valeurs peuvent être remplacées, en tapant $xx=valeur ou xx est le numéro de variable. Le firmare répond par ok quand la commande est comprise.

```
$21=0

ok
```

# Logiciel de pilotage

Avec GRBL, il va nous falloir un logiciel qui permette d'envoyer le GCODE à la carte. j'en ai testé plusieurs, avec des comportements allant de erratiques à franchement bizarre.

La solution la plus fonctionnelle que j'ai trouvée est une évolution de Universal Gcode Sender (développé initialement par l'auteur de GRBL), qui est e cours de développement par Winder.

On la trouve sur **https://github.com/winder/Universal-G-Code-Sender** Aller en bas de page sur Downloads et télécharger la version 2.0 "Nightly Build" en cliquant sur le lien "Classic GUI".

Elle est écrite en Java, ce qui en permet l'execution sur PC / Mac / Linux. Il va falloir installer une machine Java ( **https://java.com/en/download/** ) et lancer le logiciel après décompression à l'aider du fichier start.bat sous windows ou start.sh sous linux ou OSX. Il faudra certainemetn rendre start.sh executable (chmod +x start.sh).

Sous OSX il se peut que vous rencontriez des problèmes, car java est déjà installé, dans une version plus ancienne par Apple. Si l'on installe Java pour OSX à partir de java.com, il n'est pas activé par défaut.

Le plus simple pour executer le Gcode sender sur OSX consiste à editer le fichier start.sh et remplacer :

```
case "$platform" in

        mac)

                java  -Xdock:name=UniversalGCodeSender -jar -Xmx256m
$rootdir/UniversalGcodeSender*.jar

        ;;

        linux)

                java -jar -Xmx256m $rootdir/UniversalGcodeSender*.jar

        ;;

esac
```

Par

```
case "$platform" in

        mac)

                /Library/Internet\ Plug-
Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  -
Xdock:name=UniversalGCodeSender -jar -Xmx256m
$rootdir/UniversalGcodeSender*.jar

        ;;

        linux)

                java -jar -Xmx256m $rootdir/UniversalGcodeSender*.jar
```

```
        ;;

esac
```

Il faut ensuite sélectionner le port série, la vitesse (115200) et cliquer connect. La bannière

de GRBL devrait apparaitre en bas de l'écran.

# Essais

Le firmware étant opérationnel, il ne reste qu'à raccorder la carte à la machine et passer aux essais réel...

GRBL Gcode sender 2.0 permet dans le 4e onglet de mettre des macros. Cette fonction va nous être utile pour mettre un détecteur de longeur d'outils... On peut en fabriquer facilement un avec un bout de circuit imprimé vierge et une pince croco (cf **http://www.autoleveller.co.uk/cnc-probe-guide/** ) ou en acheter un pour 7 EUROS ( **http://www.banggood.com/CNC-router-engraving-MACH3-machine-Tool-Setting-Auto-Check-instrument-p-939657.html**

Ca doit être raccordé entre la broche SCL et GND du connecteur I2C de la Grbl Shield, ou sur la broche A5 de l'arduino.

Ensuite, il suffit d'assigner une macro de recherche d'origine sur l'une des macros libre :

```
G38.2 Z-25 F30;
```

Ici la machine descend en recherche de la probe à 30 mm par minute, jusqu'à "-25mm". Elle s'arrêtera sitot le contact établi sur la probe ou à -25mm si pas de probe rencontrée.

Une deuxième macro fera le job pour le point zero du Z à partir de la position courante, en tenant compte de l'épaissseur du capteur :

```
G92 Z1.65;

ici pour un morceau de circuit imprimé de 16/10e + 35 um de cuivre
```

ou

```
G92 Z20;
```

pour la sonde de banggood, qui fait 20mm d'épaisseur

Apparemment, ces réglages ne sont as sauvés. Il faut editer le fichier JSON de préférences du logiciel (UniversalGcodeSender.json ) et l'ajouter dedans:

```
...
  "customGcode1": "G0 X0 Y0;",
  "customGcode2": "G38.2 Z-30.00 F30;",
```

```
  "customGcode3": "G92 Z1.65;G0 X-20.00",
  "customGcode4": "G92 Z20.00;G0 X-20.00;",
  "customGcode5": "",
...
```