INNOVATIONS... MONTAGES FIABLES... ÉTUDES DÉTAILLÉES... ASSISTANCE LECTEUR

ELECTRONIQUE

ELECTRONIQUE

ET LOISIRS

magazine

Part Dieu
SCI

www.electroniquemagazine.com

n°133 HIVER 2015

Testeur EOBD pour diagnostic auto

- Cours Android VI
- Maîtrisez Arduino III
- Fabriquez un boîtier avec la 3DRAG
- Reconnaissance vocale des couleurs
- Extension GSM/GPRS/GPS pour RaspberryPi

France 7,50 € - DOM 7,50 € - CE 8,00 € - LUX. 8,00 € - NC(S)900FP - Poly.fr 1200CFP - Canada 11,50 $C

N° 133 Décembre 2015

M 04662 - 133 - F: 7,50 € - RD

Sommaire

ARTICLES

Numéro 133
Hiver 2015
100 pages

Les typons des circuits imprimés et les programmes lorsqu'ils sont libres de droits sont téléchargeables

L'EDITO HIVER 2015

Chères lectrices, chers lecteurs,

La Rédaction d'Electronique et Loisirs Magazine vous souhaite de Joyeuses Fêtes de Noël ainsi qu'une Bonne et Heureuse Année 2016.

Pour cette fin d'année nous proposons à nos fidèles lectrices et lecteurs un testeur EOBD pour véhicule qui vous épargnera des factures astronomiques. Ce montage permettra à moindre coût de dépanner votre auto dans de nombreux cas ou tout simplement vérifier l'état de santé de votre véhicule. Il pourra aussi afficher la puissance réelle du moteur, notamment de certains modèles qui seront rappelés pour non-conformité.

Pour les adeptes de l'impression 3D, nous décrivons la fabrication sur mesure d'un boitier en plastique grâce à l'application Tinkercad et la 3DRAG. Nous continuons aussi notre saga sur le RaspberryPi en vous proposant une carte d'extension GSM/GPS/GPRS avec une application distante contrôlée. Pour les personnes ayant des difficultés de vision nous décrivons une reconnaissance vocale des couleurs qui nous l'espérons rendra meilleur leur quotidien.

Enfin nous continuons notre série sur la maîtrise d'Arduino avec les afficheurs LCD et graphiques et la programmation Android avec la gestion directe des appels et des messages ainsi que le contrôle des détecteurs de mouvements et la localisation GPS d'un Smartphone.

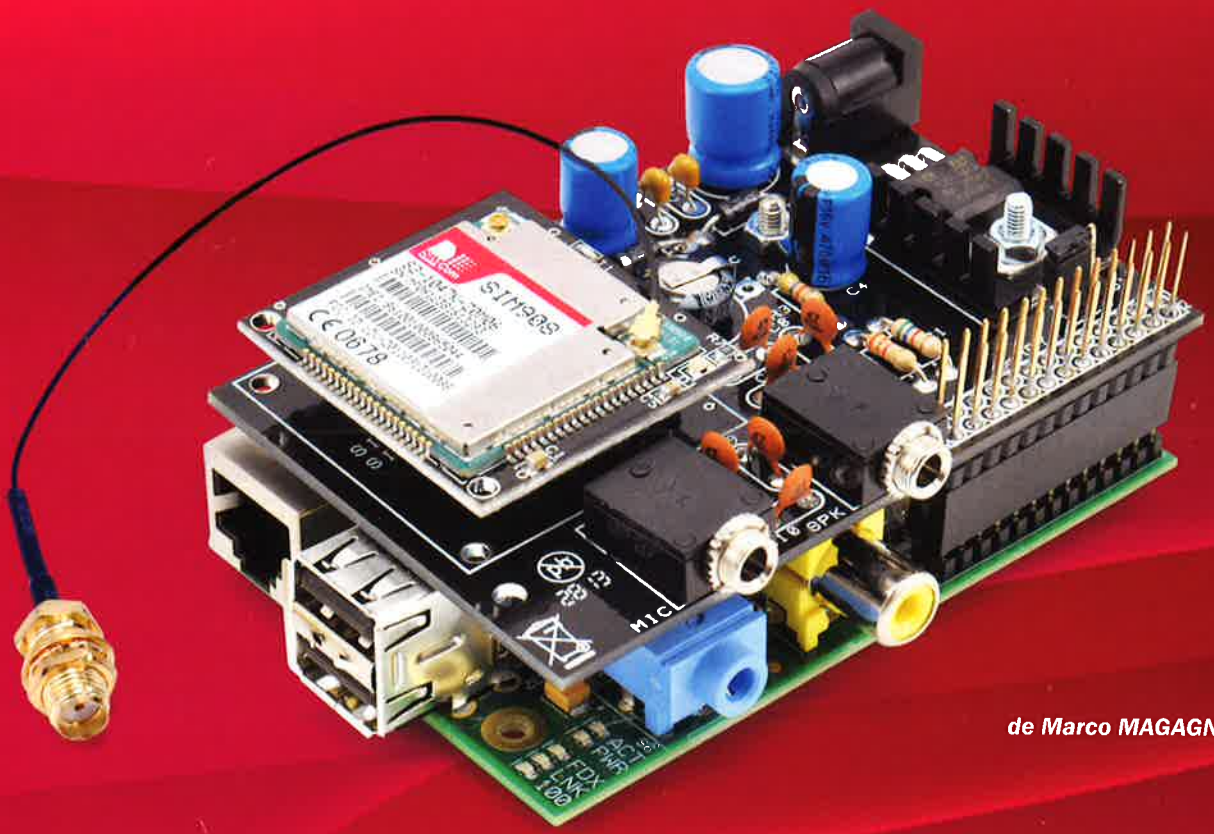
La Rédaction

à l'adresse www.electroniquemagazine.com dans le sommaire de la revue 133 à l'onglet « Télécharger ».

ET1075

Extension GSM/GPRS/GPS pour RaspberryPi

Dans cet article nous vous proposons de réaliser une carte d'extension pour RaspberryPi qui permet de contrôler soit le module SIM900 (GSM/GPRS) soit le module SIM908 (avec la fonction GPS en supplément). Vous pourrez ainsi réaliser des applications qui peuvent être contrôlées ou qui peuvent communiquer à distance à travers les différents réseaux de téléphonie mobile.



de Marco MAGAGNIN

Le système d'exploitation **GNU/Linux** est par vocation orienté vers la connectivité Internet. Les protocoles de transferts des données **TCP/IP** (Transmission Control Protocol/Internet Protocol) sont étroitement liés à l'histoire d'**UNIX** en premier, puis à ses successeurs comme **Linux**.

Cependant cela n'est pas la seule façon pour le système **Linux** de communiquer avec le monde extérieur. Nous allons décrire comment connecter le **RaspberryPi** aux modules de la famille **SIM9XX** via le port série. De cette façon, il est possible d'étendre les fonctionnalités du **RaspberryPi** pour des applications nécessitant une connectivité mobile. De plus en utilisant le module **SIM908** nous disposons d'un **GPS**.

Nous ne devons pas uniquement penser à des applications de connectivité GSM strictement « mobiles », dans les véhicules par exemple, mais aussi, pour des **endroits non connectés à Internet**. Nous pensons tout simplement à un système domotique, ou une alarme ou un système de contrôle d'une installation automatisée.

Dans la plupart des cas ce sont les box ADSL qui procurent la connexion Internet, en cas de coupure de courant il ne serait plus possible de communiquer avec ces systèmes. De plus cela implique un abonnement mensuel élevé pour une utilisation ne nécessitant pas de débit important.

Dans ce cas, la seule façon d'envoyer un message d'avertissement sur l'état du système est le réseau GSM. De toute évidence le système de contrôle doit être équipé d'une batterie de secours pour une grande fiabilité, et doit être capable de communiquer avec **Linux**.

Pour l'instant, revenons à notre carte d'extension **GSM/GPRS** et voyons comment l'utiliser avec le **RaspberryPi**. Comme nous l'avons déjà mentionné dans les précédents articles de la revue dédiés au **RaspberryPi**, les **cartes d'extensions sont conçues pour être enfichées** (empilées) **les unes sur les autres** et peuvent **gérer simultanément plusieurs fonctions différentes**, de manière à **intégrer plusieurs « services »** dans une seule solution intégrée. Par exemple c'est le cas d'un système domotique équipé de capteurs et d'actionneurs et qui utilise un afficheur LCD pour visualiser les états, le même système est réalisable à distance soit par l'intermédiaire d'Internet soit par le réseau GSM.

En plus de cela, il est possible d'ajouter des **services de sécurité pour les personnes âgées**, qui peuvent être activés localement ou à distance, **vocalement** ou par **envoi de SMS** à des **numéros prédéfinis**.

Nous pouvons ajouter la prise d'images avec une caméra vidéo associées soit à l'ouverture d'une porte avec une reconnaissance faciale, soit à des détecteurs de mouvements, avec l'envoi de messages et d'images en cas de doute...

La seule limite est l'imagination ... et la capacité de concevoir et mettre en œuvre des applications. Les applications de cette importance dépassent les capacités des microcontrôleurs en général, bien que ceux-ci peuvent être mis à profit pour gérer les capteurs et les divers périphériques.

Revenons à la pratique, les **différentes cartes connectées** au **RaspberryPi ne doivent pas utiliser les mêmes broches** (pins) à part l'alimentation, afin d'éviter des **conflits non gérés par logiciel**. Notre carte d'extension **GSM/GPRS/GPS** utilise les broches du **port série du connecteur GPIO** du **RaspberryPi** et 2 broches d'entrées/sorties (I/O) pour l'**allumage/extinction** et la **réinitialisation** du module **SIM9XX**.

Le schéma électrique

Le schéma électrique de la carte d'extension **ET1075** (visible en figure A) est **conçu pour connecter les 2 types de modules SIM900** ou **SIM908**.

La **carte permet une alimentation autonome** du **module GSM**, et offre des prises déportées pour l'**entrée micro** et la **sortie audio** sur deux jacks dédiés. Elle permet aussi de connecter convenablement les broches du connecteur **GPIO** du **RaspberryPi** avec celles des modules GSM. Le circuit de l'alimentation est constitué par le **régulateur U1** (7805) et par le réseau de **condensateurs de filtrages** et de stabilisation de la tension (C1, C2, C3 et C4).

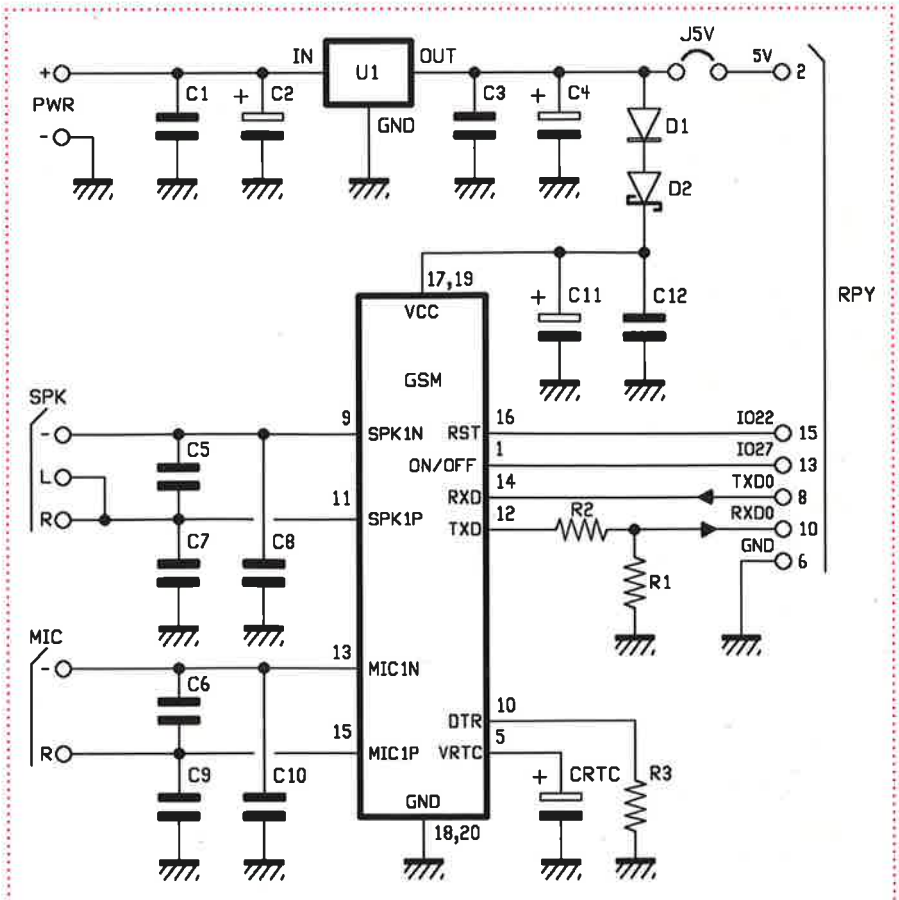


Figure A : schéma électrique de la carte d'extension ET1075.

A l'aide de la diode **D1** et de la diode **Zener D2**, la tension d'alimentation est adaptée à une tension positive (VCC) qui alimente les broches **17** et **19** du module **SIM9XX**. Afin d'éviter des perturbations, les condensateurs **C11** et **C12** filtrent et stabilisent à nouveau la tension VCC. Le pont « **JV5** » permet également d'**alimenter** le **RaspberryPi** à partir de la carte d'extension. La source d'alimentation de la carte d'extension se connecte aux points « PWR+ » et « PWR– » et doit être comprise entre **7 VDC** et **12 VDC**.

La broche **ON/OFF** du module **SIM9XX** est connectée à la broche **13** (GPIO27) du connecteur **GPIO**, elle permet l'activation/désactivation du module. La broche de réinitialisation « **RST** » du module **SIM9XX** est reliée à la broche **15** (GPIO22) du connecteur **GPIO**. Reportez-vous au **brochage du connecteur GPIO** visible en **figure B**.

La **connexion série** s'effectue au moyen des broches **8** (TXD0) et **10** (RXD0) du RaspberryPi. Sur la ligne d'entrée « **RDX0** », le niveau de la tension doit être adapté à **3,3 V** pour être compatible avec le RaspberryPi.

Cela s'effectue au moyen du **diviseur de tension** constitué par les résistances **R1** et **R2**. Aucune adaptation de la tension n'est requise dans le sens opposé, la sortie à 3,3 V du **RaspberryPi** est compatible avec l'entrée du module **SIM9XX**. La ligne « **DTR** » est maintenue constamment à un **niveau bas** par l'intermédiaire de la résistance **R3**. La broche pour l'alimentation tampon de l'horloge temps réel (RTC) est connectée au condensateur « **CRTC** » de forte capacité (0,1 F). Celui-ci permet de garder les paramètres de l'horloge en cas de coupure de courant.

Raspberry Pi B+ J8 Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figure B : pour rappel le brochage du connecteur GPIO du RaspberryPi modèle B+.

Enfin, les lignes d'entrée microphone et de sortie audio sont connectées respectivement aux connecteurs « **MIC** » et « **SPK** » avec chacune son réseau de **condensateurs de filtrages** : C6, C9 et C10 pour l'entrée « MIC » et C5, C7 et C8 pour la sortie « SPK ».

Dans la section « Plan de montage » (voir ci-après), vous pouvez voir l'implantation des différents composants.

En dehors de l'attention habituelle qu'il faut porter lors du montage, il est important de positionner **correctement le connecteur double** (RPY) **de façon à ce que la distance entre le circuit imprimé de la carte d'extension et le circuit imprimé du RaspberryPi permette l'assemblage** en « **sandwich** », dans le cas où vous devez utiliser conjointement d'autres cartes comme la carte « **Afficheur LCD** » décrite dans le numéro **129 d'Electronique et Loisirs Magazine**.

Par conséquent, avant de souder les broches du connecteur, vous devez positionner le connecteur 26 broches en procédant de la manière suivante :

1. placez l'entretoise hexagonale, en insérant l'extrémité filetée du côté soudure dans le trou près du condenseur « CTRC » et fixez-la avec un écrou M3 ;
2. placez le connecteur à 26 broches dans l'emplacement nommé « RPY » en maintenant le côté femelle vers le bas ;
3. insérez la carte sur le RaspberryPi en la plaquant contre l'entretoise. Enfichez les 2 connecteurs 26 broches, et vissez l'entretoise avec une vis M3 ;

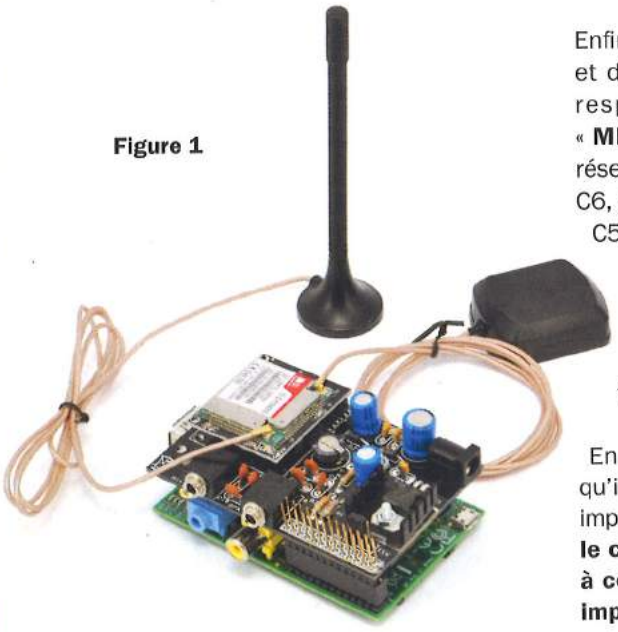


Figure 1

Les modules GSM/GPRS utilisés

La carte d'extension pour RaspberryPi présentée dans cet article peut accueillir soit le module SIM900 (GSM/GPRS) soit le module SIM908 (avec en plus le GPS). Plus précisément, le module SIM900 est un quadri-bande compatible GSM/GPRS, fonctionnant aux fréquences 850/900/1800/1900 MHz, avec SMS et accès à Internet via une pile TCP/IP intégrée. Cette dernière est gérée par un processeur AMR926EJ-S qui s'occupe également de la gestion des communications téléphoniques via les commandes de type « AT ». Le module est compatible « GSM phase 2/2+ », il est de classe 4 (2W) à 850/900 MHz et de classe 1 (1 W) à 1800/1900 MHz.

Le module SIM900 intègre une interface analogique, un convertisseur A/D, une horloge temps réel RTC, un bus SPI, un bus I²C et un module PWM.

Le module est alimenté avec une tension continue comprise entre 3,4 V et 4,5 V, il consomme un courant maximal de 0,3 A (en continu) pendant une transmission. Il est équipé d'une prise pour une antenne externe.

Le module SIM908 intègre un récepteur GPS (avec une antenne séparée) à 42 canaux, sa sensibilité est de -160 dB en « tracking » (suivi) et de -143 dB en acquisition. Sa précision de localisation est de 2,5 m seulement. Son temps de démarrage à chaud (hot start) est d'une seconde et de 30 secondes pour le démarrage à froid (cold start). Le jeu de commandes AT est étendu pour l'accès au GPS.

NB : le terme « démarrage à froid » décrit la performance d'un récepteur GPS à sa mise sous tension lorsqu'il n'y a aucune donnée de navigation disponible, par exemple lors de la première mise en service. Le démarrage à froid signifie que le récepteur n'a pas encore d'almanach à jour en mémoire (satellites en vue), ni de données sur les orbites des satellites, ni de position géographique initiale, ni de temps de référence.

Au cours du démarrage à froid, le récepteur choisit automatiquement un ensemble de satellites et dédie un canal individuel à la recherche du signal de chaque satellite de la constellation en appliquant éventuellement des décalages « Doppler » sur ces fréquences.

Si aucun des satellites n'est acquis après une période prédéterminée, appelée « Time Out », le récepteur choisit un nouvel ensemble de recherche de satellites puis, il répète le processus jusqu'à ce que les données d'un premier satellite soient acquises.

Pendant que des satellites sont acquis, le récepteur rassemble automatiquement les données de l'almanach qu'il commence à enregistrer. Le récepteur utilise l'information obtenue lors de l'acquisition des données d'un satellite spécifique pour éliminer de l'ensemble des recherches les satellites au-dessous de l'horizon. Cette stratégie améliore l'acquisition des données des satellites additionnels exigés pour effectuer le premier calcul d'une position.

Les ensembles de recherche de démarrage à froid sont établis pour s'assurer qu'au moins 3 satellites sont acquis au cours des deux premières périodes d'arrêt. Dès que 4 satellites sont acquis, le récepteur calcule une première localisation. Un récepteur GPS réalise typiquement un démarrage à froid en moins d'une minute.

Le « démarrage à chaud » s'applique lorsque l'almanach, la position, l'heure, les données d'éphéméride dans la mémoire sont valides. La stratégie de recherche lors d'un démarrage à chaud identifie les satellites que le récepteur s'attend à trouver grâce aux données de l'almanach, de la position initiale et de l'heure. Les données dans la mémoire sont considérées à jour et valides, le temps d'acquisition est en général de moins de 30 secondes.

4. une fois les cartes alignées et espacées correctement, vous pouvez souder sur les broches du connecteur mâle.

Toujours dans le souci d'assurer un empilement des cartes, **placez soigneusement le dissipateur du régulateur U1, de sorte qu'il ne touche pas une éventuelle carte** montée au-dessus.

Utilisation pratique de la carte

La première étape consiste à monter un des modules GSM compatibles sur la carte d'extension et ensuite, si cela n'est pas déjà fait, d'insérer la carte sur le connecteur du RaspberryPi. **Assurez-vous que le côté soudure de la carte n'entre pas en contact avec la partie métallique des connecteurs USB et/ou Ethernet**.

En cas de doute, protégez les connecteurs avec de l'isolant. Connectez les périphériques, le réseau et l'alimentation du **RaspberryPi** de manière habituelle. Connectez l'alimentation de la carte ET1075 et ... mettez sous tension.

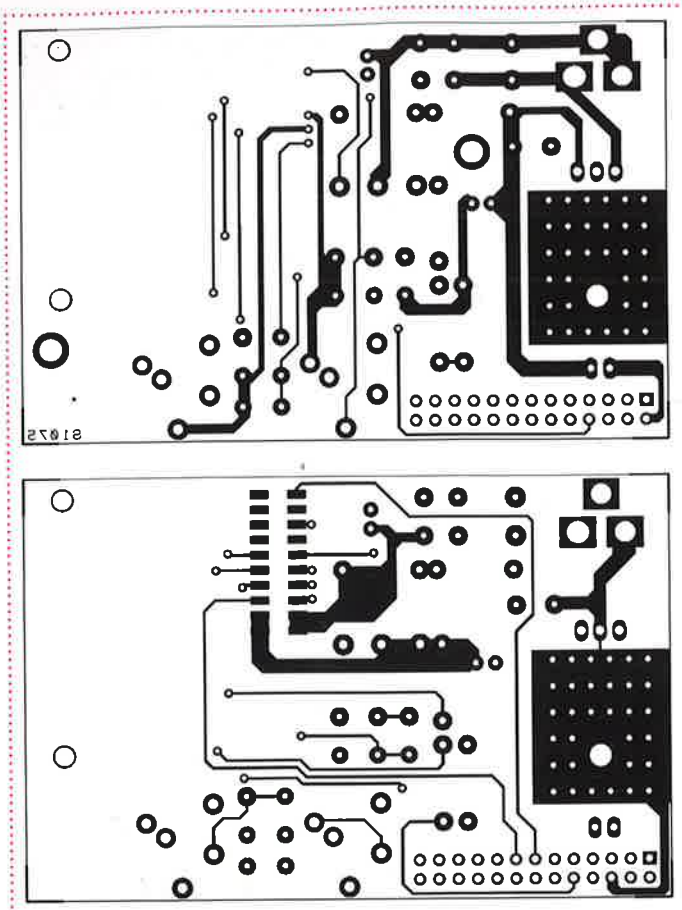


Figure C : circuit imprimé à l'échelle 1 : 1 de la carte d'extension côté soudures.

Liste des composants du ET1075

R1.....5,6 kΩ
R2.....3,3 kΩ
R3.....4,7 kΩ
C1.....100 nF multicouche
C2.....470 µF/25V électrolytique
C3.....100 nF multicouche
C4.....470 µF/16V électrolytique
C5.....47 pF céramique
C6.....47 pF céramique
C7.....47 pF céramique
C8.....47 pF céramique
C9.....47 pF céramique
C10....47 nF céramique
C11....470 µF/16V électrolytique
C12....100 nF multicouche
U1.....7805

Figure D : circuit imprimé à l'échelle 1 : 1 de la carte d'extension côté composants.

Dans cet article, nous vous proposons une **approche didactique de l'utilisation de la carte GSM** afin de vous donner la possibilité d'expérimenter autant que possible avant de commencer à écrire des applications plus complexes.

Rappelons que l'utilisation de **Linux** permet de réaliser des applications qui effectuent de nombreuses tâches, comme par exemple le partage des ressources de la carte GSM.

Par exemple, nous pouvons écrire une application qui attend un événement externe tel que l'arrivée d'un appel vocal ou d'un message SMS, mais qui soit également capable de réagir à un événement interne telle que l'activation des entrées/sorties (I/O) numériques et analogiques, ou la demande de passer un appel vocal ou d'envoyer un texto à un ou plusieurs numéros. L'application peut aussi envoyer un fichier ou un mail.

En attendant, vous pouvez acquérir les données provenant de la section GPS du module. Pour ce faire, comme nous l'avons déjà décrit pour la carte « Afficheur LCD » (reportez-vous au numéro 129), nous devons concevoir une architecture modulaire qui puisse partager des ressources sans créer de conflit. Il en résulte une architecture composée de différents types de modules, tels que les modules de base (systèmes de fichiers, protocoles réseaux), qui s'interfacent et contrôlent les ressources, les modules périphériques (carte mère, carte vidéo, carte réseau...), dont chacun est spécialisé dans une fonction déterminée et qui peut être rappelé dans des conditions spécifiques. Les modules périphériques communiquent avec les modules de base afin de définir les priorités et éviter les conflits.

Pour cela, vous avez besoin d'apprendre dans les moindres détails les exigences fonctionnelles de l'application, c'est pour cette raison qu'avant d'entrer dans la conception d'applications client/serveur que nous étudierons

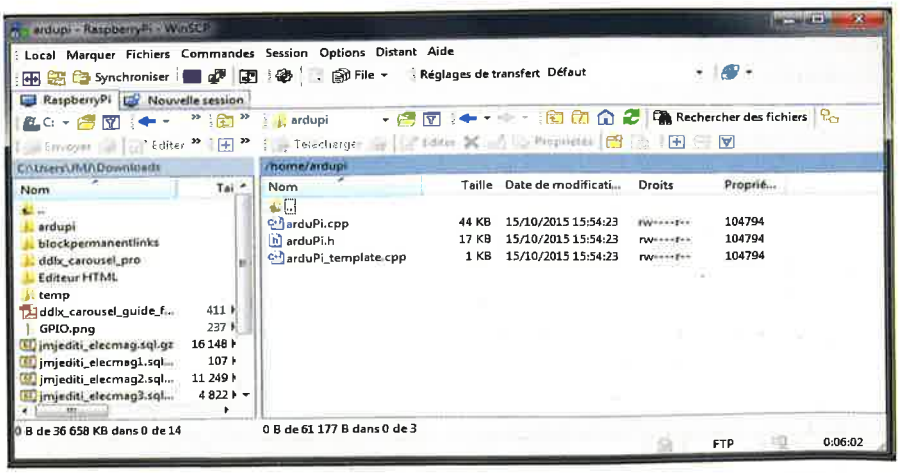


Figure 2 : ici le contenu du dossier « ardupi » visualisé à l'aide de WinSCP.

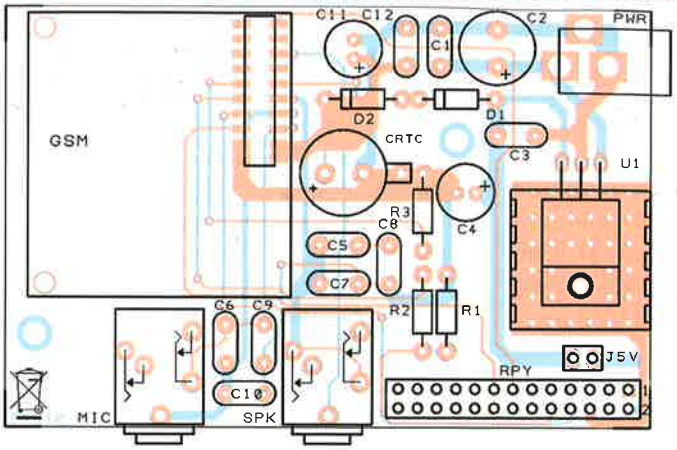


Figure E : schéma de câblage des composants de la carte d'extension.

D1..... 1N4007
D2..... 1N5819
CRTC.. condensateur bas profil 0,1 F
Divers

Prise alimentation pour ci
Prise jack femelle stéréo 3,5 mm pour ci (x 2)
Dissipateur
Vis 8 mm 3 MA
Vis 12 mm 3 MA
Ecrou 3 MA (x 2)
Jumper
Barrette mâle pôle
Prise MMS SMT 2 x 10 pôles au pas de 2 mm
Connecteur 13 x 2 pour RaspberryPi
Entretoise M/F 18 mm

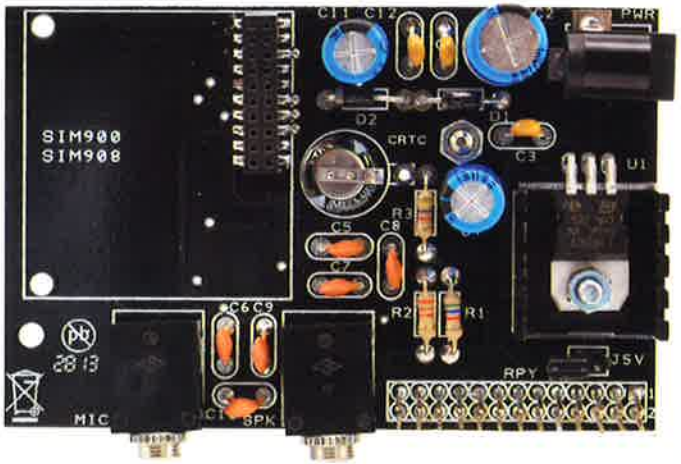


Figure F : photo de l'un de nos prototypes de la carte d'extension GSM/GPRS/GPS.

dans les prochains numéros, nous vous proposons quelques expérimentations qui vous permettront une utilisation souple de la carte d'extension.

La première expérimentation permet de **réutiliser des petits programmes écrits pour Arduino en les modifiant**, cela vous familiarisera avec la carte Arduino dont vous pouvez trouver depuis quelques temps dans cette revue un cours complet. Nous allons par exemple réutiliser un programme de gestion des modules GSM développé pour Arduino. Le programme utilise la **librairie** « **arduPi** » développée par « Libelium Comunicaciones Distribuidas S.L » disponible sous la licence « GNU General Public License ».

La seconde expérimentation est **d'utiliser un mode direct**. Pour cela nous nous connectons au port série avec un programme **TTY** et nous **pilotons directement le module GSM** en lui envoyant et en recevant des commandes de gestion. Cela nous permet de tester des séquences de commandes nécessaires pour réaliser une fonction spécifique, sans devoir écrire et tester à chaque fois un programme.

Envoyer un SMS de type Arduino

Nous allons voir comment installer et utiliser la librairie « **arduPi** », pour cela il suffit de la télécharger sur notre site web : **www.raspberrypi.electronique-magazine.com/telechargement.html**

Nous proposons **deux librairies** : une pour le **RaspberryPi** et l'autre pour le **RaspberryPi 2**. Décompressez l'archive dans le dossier « **/home** ». Pour plus de commodité, mettons dans un même dossier (que nous nommons « ardupi » par exemple) les 3 fichiers ainsi extraits.

Avant de compiler la librairie, ouvrons le fichier « **arduPi.cpp** » et, si la **ligne 64** faisant référence au port série est différente, remplacez- la par ligne suivante :

```
serialPort="/dev/ttyAMA0";
```

Maintenant, allons dans le dossier « ardupi » (voir la figure 2) avec la commande :

```
cd /home/ardupi
```

et compilons la librairie avec la commande (voir la figure 3) :

```
g++ -c ardupi.cpp -o ardupi.o
```

En ce qui concerne les programmes nous proposons 3 modules.

Figure 3 : ici la compilation de la librairie.

Deux permettront d'allumer et d'éteindre le module GSM et seront très utiles à l'avenir. Le 3ème permet d'envoyer un SMS à un téléphone mobile.

Pour **activer le module GSM nous avons besoin de mettre à un niveau haut la broche « GPIO27 » pendant 2 secondes**, puis la **ramener à un niveau bas**. Pour l'éteindre **nous effectuons la même opération** lorsque le module est allumé.

Pour la **réinitialisation du module GSM**, il est nécessaire **d'amener et de maintenir à un niveau haut pendant une demi-seconde la broche « GPIO22 », puis de la ramener à un niveau bas**. Avant tout, pour pouvoir utiliser le port série, nous devons effectuer une **modification dans le processus** de « **boot** » (démarrage) afin de **désactiver la console série** qui, dans ce cas, accaparerait le port série.

Listing 1

```
/*
* GSMallumage
*/

//Include ArduPi library
#include "arduPi.h"

int resetModulePin = 9;
int onModulePin = 8;

void switchModuleOn(){
   digitalWrite(onModulePin,HIGH);
   delay(2000);
   digitalWrite(onModulePin,LOW);
}

void resetModule(){
   digitalWrite(resetModulePin,HIGH);
   delay(500);
   digitalWrite(resetModulePin,LOW);
   delay(100);
}

int main (){
   Serial.begin(115200);
   delay(2000);
   pinMode(resetModulePin, OUTPUT);
   pinMode(onModulePin, OUTPUT);
   Serial.flush();
   printf ("zero\n");
   Serial.print("AT");
   delay(1000);
   if (Serial.available()==0)
     {
     printf ("uno\n");
     resetModule();
     delay(2000);
     }
   Serial.print("AT");
   delay(1000);
   if (Serial.available()==0)
     {
     printf ("due\n");
     switchModuleOn();
     }
        return (0);
```

Désactiver la console série

Presque toutes les distributions **Linux** activent au démarrage les fonctionnalités de la console, c'est-à-dire la possibilité de communiquer via une ligne de type série avec un terminal distant (réel ou virtuel) afin de gérer l'ensemble du système à travers cette interface de communication.

C'est un héritage du temps où l'on se connectait à un ordinateur via un modem en utilisant les lignes téléphoniques standards. Parfois, c'est encore la seule possibilité d'accéder à des systèmes embarqués ou à des serveurs.

Pour désactiver la console, utilisez le logiciel « WinSCP » (reportez-vous aux numéros 124 et suivants d'Electronique et Loisirs Magazine pour une utilisation détaillée de ce programme), allez dans le dossier « **boot** » avec la commande :

```
cd /boot
```

et éditez le fichier « **cmdline.txt** » en faisant un double-clic dessus, et dont le chemin est « **/boot/cmdline.txt** ». Supprimez l'expression suivante en accordant une attention aux paramètres de configuration (voir la figure 4) :

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```

ci-dessus c'est l'expression à supprimer, surlignée en bleu en figure 4. Enregistrez le fichier et fermez-le.

Exemples de programmes (ON / OFF du module GSM et d'envoi de SMS)

Examinons la logique du programme de l'**allumage du module GSM**, dont vous pouvez voir le code source dans le **Listing 1**. Un premier examen nous montre que le programme a été écrit pour exécuter une seule séquence d'instructions, contrairement aux programmes pour Arduino, qui s'exécutent à l'infini dans une boucle de type « loop() ». Cela permet d'utiliser des programmes individuels comme module d'une application de niveau supérieur.

Le processus de mise sous tension du module GSM comprend les étapes suivantes :

- contrôle, à travers l'envoi d'une série de commandes de type « AT », permettant de vérifier que le module GSM n'est pas déjà allumé ;
- si la réponse est négative (à l'envoi d'une commande AT dans une situation normale, le module GSM retourne la chaîne « OK »), en deuxième contrôle est exécutée une procédure de réinitialisation (« RESET ») avec l'envoi d'une deuxième commande « AT » ;
- en cas d'une nouvelle réponse négative, le processus d'allumage est effectué. Il consiste à mettre à un niveau haut pendant 2 secondes la broche « GPIO27 ».

La nécessité des **contrôles préliminaires est due au fait que le processus d'allumage et d'extinction du module GSM est le même**, donc sans ces contrôles, si le programme d'allumage est exécuté et que le module GSM est déjà allumé, il s'éteint. Pour compiler le programme « **GSMallumage** » nous utilisons la commande :

```
g++ -lrt -lpthread GSMallumage.cpp ardupi.o -o GSMallumage
```

Figure 4 : l'expression surlignée en bleu doit être supprimée.

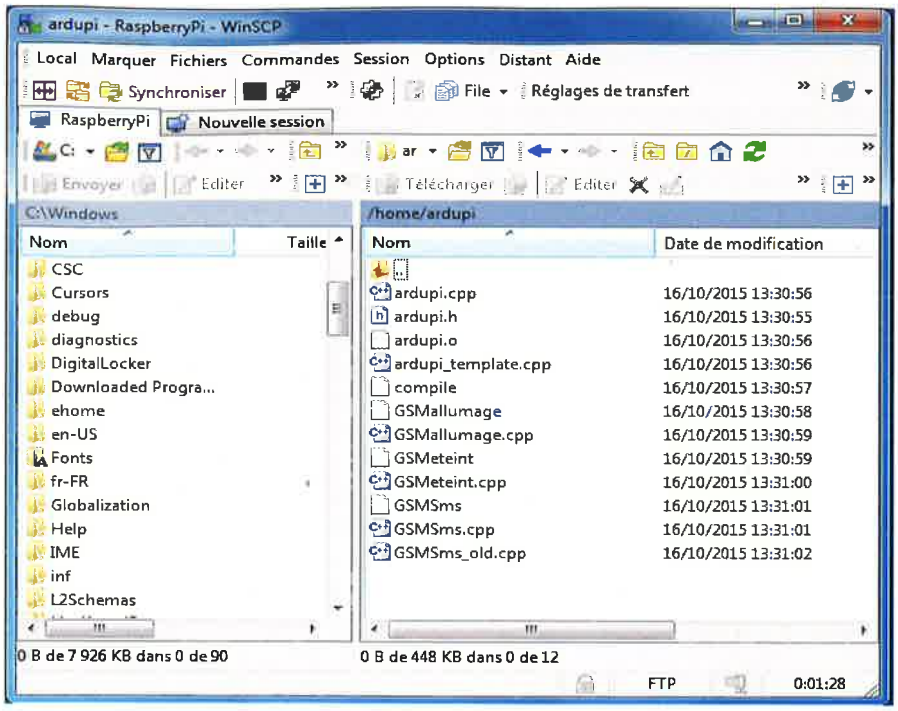


Figure 5 : contenu du dossier « ardupi » après l'écriture et la compilation des programmes.

Pour exécuter le programme après l'avoir placé dans le dossier « **ardupi** » avec la commande :

cd /home/ardupi

tapez la commande suivante :

./GSMallumage

Vous pouvez suivre la procédure d'allumage du module GSM en observant la **LED verte qui**, à la mise sous tension, s'allume pendant environ **2 secondes** et puis s'éteint. Ensuite **elle clignote** à des intervalles d'environ une **demi-seconde** pour signaler que le module est à la **recherche du réseau GSM**. Lorsque le **réseau est accroché**, la **LED clignote** à une fréquence beaucoup plus faible, environ **une fois par seconde**.

Dans le **Listing 2**, vous pouvez voir le programme de **désactivation** (extinction) du **module GSM**, la logique est similaire au programme précédent :

- contrôle par l'envoi d'une série de commandes « AT » afin de vérifier que le module n'est pas déjà éteint ;
- en cas de réponse négative, comme deuxième contrôle, une procédure de réinitialisation est effectuée avec l'envoi d'une deuxième commande « AT » ;
- en cas de réponse positive, la procédure d'extinction est exécutée, elle consiste à mettre à un niveau haut pendant 2 secondes la broche « GPIO27 ».

Encore une fois l'objet des vérifications préliminaires est dû au fait que

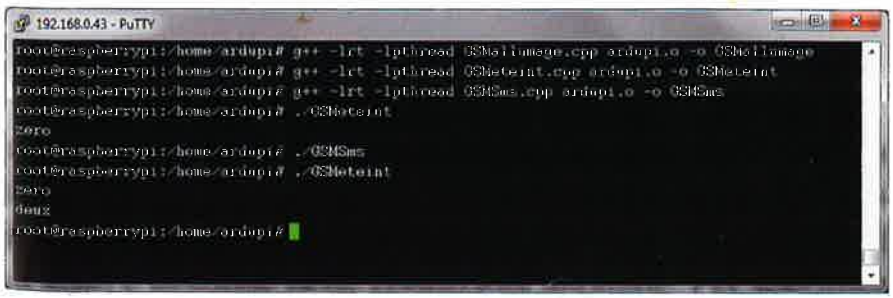


Figure 6 : séquence des commandes de compilation et d'exécution des différents programmes du dossier « ardupi ».

les processus de mise sous tension et hors tension sont les mêmes, si vous exécutez le programme de mise hors tension sans contrôles et que le module GSM est déjà éteint, il s'allume. Vous obtenez l'effet inverse à celui désiré.

Pour compiler le programme « **GSMeteint** » utilisez la commande :

g++ -lrt -lpthread GSMeteint.cpp ardupi.o -o GSMeteint

Listing 2

```
/*
* GSMeteint
*/

//Include ArduPi library
#include "arduPi.h"

int resetModulePin = 9;
int onModulePin = 8;

void switchModuleOff(){
  digitalWrite(onModulePin,HIGH);
  delay(2000);
  digitalWrite(onModulePin,LOW);
}

void resetModule(){
  digitalWrite(resetModulePin,HIGH);
  delay(500);
  digitalWrite(resetModulePin,LOW);
  delay(100);
}

int main (){
  Serial.begin(115200);
  delay(2000);
  pinMode(resetModulePin, OUTPUT);
  pinMode(onModulePin, OUTPUT);
  Serial.flush();
  printf ("zero\n");
  Serial.print("AT");
  delay(1000);
  if (Serial.available()==0)
    {
    printf ("uno\n");
    resetModule();
    delay(2000);
    }
  Serial.print("AT");
  delay(1000);
  if (Serial.available()>0)
    {
    printf ("due\n");
    switchModuleOff();          //
    }
        return (0);
}
```

Pour exécuter le programme, après l'avoir placé dans le dossier « **ardupi** », tapez la commande :

```
./GSMeteint
```

Dans le **Listing 3**, vous pouvez voir le programme qui permet d'envoyer un SMS.

- la variable « **phone_number[]** » adapte le numéro de téléphone à appeler ;
- la commande « **AT+CMGF=1** » définit le mode SMS au format texte ;
- la commande « **AT+CMGS=** » envoie un message SMS défini dans l'instruction suivante ;
- enfin, la chaîne « **CTRL-Z** » est envoyée pour terminer l'appel.

Pour compiler le programme, tapez la commande suivante :

```
g++ -lrt -lpthread GSMSms.cpp arduPi.o -o GSMSms
```

Pour exécuter le programme tapez la commande :

```
./GSMSms
```

Avant de lancer le programme, il est préférable d'exécuter d'abord le programme d'allumage du module GSM, afin de vous assurer que le module GSM est bien opérationnel.

Lorsque vous développerez une application de contrôle, ça sera à cette dernière de gérer la séquence correcte d'appel des modules (de Linux) avec leurs contrôles relatifs des états et la gestion des conditions anormales d'un module GSM par exemple.

En figure 5 vous pouvez voir le contenu du dossier après l'écriture et la compilation des programmes. En figure 6 vous pouvez voir la séquence des commandes de compilation et d'exécution de l'allumage du module GSM, l'envoi de SMS et l'extinction du module.

Nous allons étudier maintenant la seconde expérimentation qui permet d'interagir directement avec le port série grâce à un programme d'émulation **TTY** « **CuteCom** ». Puisque nous avons configuré le **RaspberryPi** pour être géré à distance via un protocole de type « **SSH** », continuons dans cette direction afin d'utiliser l'**émulateur TTY** de cette même manière.

Installation de l'émulateur TTY série

Pour la gestion du port série, nous avons choisi d'utiliser le programme « CuteCom ». Nous l'installons avec les commandes habituelles :

```
apt-get update
apt-get upgrade
apt-get install cutecom
```

Installation de Xming

« **Xming** » est un serveur « **X** » Open Source pour **Windows** qui permet de **visualiser sur le bureau d'un PC Windows des fenêtres graphiques d'applications qui fonctionnent sous Linux à distance.** Ce n'est pas l'ensemble du bureau qui est transféré, mais seulement les fenêtres graphiques d'applications individuelles fonctionnant sur des systèmes distants.

Vous pouvez imaginer les anecdotes racontées dans le monde des administrateurs systèmes, laisser la possibilité d'exécuter des commandes dans une fenêtre d'administration du système à partir d'un système différent ! Toutefois, la fonctionnalité est très intéressante.

« **X** » est une application basée sur le modèle « **client/serveur** » qui permet d'exporter sur une machine distante l'interface graphique (Front-End) d'une application fonctionnant sous un système différent. Elle s'appuie sur le protocole « **SSH** » et sur l'application « **Putty** » (ou Kitty) que nous avons présenté dans les précédents articles de la revue traitant du **RaspberryPi** (à partir du numéro 124).

La dénomination « **client /serveur** » provoque souvent une **confusion** pour les utilisateurs de l'application « **X** » parce que les termes apparaissent inversés.

Sur le bureau Windows de l'**utilisateur** est le « **serveur** » qui est en cours d'exécution et il est utilisé par les **applications** (« **client** »). L'application « X » offre une visualisation des programmes, il se comporte comme un « serveur », les applications qui sont exploitées via

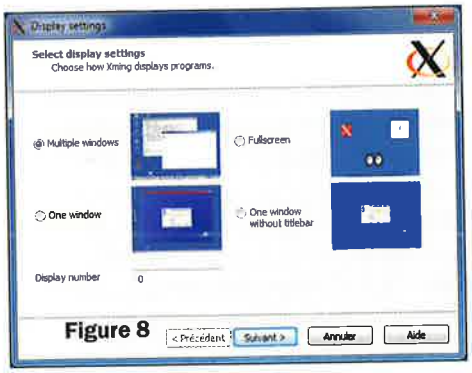


Figure 8

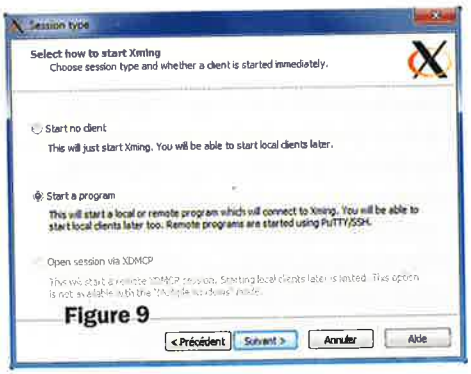


Figure 9

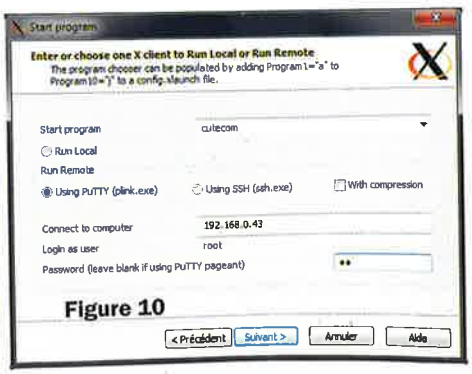


Figure 10

Figure 7 : installation de l'émulateur TTY série.

Listing 3

```
/*
* GSMSms
*/

//Include ArduPi library
#include "arduPi.h"

int timesToSend = 1;                        // Nombre de SMS à envoyer
int count = 0;
int ok = 0;
int numCar = 0;

char phone_number[]="3.....9";              // Numéro de téléphone du destinataire

void setup(){
  Serial.begin(115200);  // UART baud rate
  delay(2000);
  Serial.println("AT+CMGF=1");              // configure le mode SMS au format texte
  delay(100);
}

void loop(){
  while (count < timesToSend){
    delay(1500);
    Serial.print("AT+CMGS=\"");             // envoie le SMS au numéro
    Serial.print(phone_number);
    Serial.println("\"");
    while(Serial.read()!='>');
    Serial.print("Se arriva funziona");     // corps du SMS
    delay(500);
    Serial.write(0x1A);                     // envoie ++
    Serial.write(0x0D);
    Serial.write(0x0A);
    delay(5000);
    count++;
  }
}

int main (){
        setup();
        while(1){
                loop();
        if (count == timesToSend)
        {
        break;
        }
        }
        return (0);
}
```

une connexion distante utilisent ces services en tant que « client ».

Le protocole de communication entre le « serveur » et le « client » utilise le réseau de manière transparente. Les deux peuvent résider sur une même machine ou sur des machines différentes, avec des architectures différentes et des systèmes d'exploitation différents.

Le « serveur » et le « client » peuvent également communiquer en toute sécurité sur le réseau via un tunnel crypté qui utilise souvent le protocole « SSH ».

Le serveur « **X** » peut être :

- un **programme du système** qui contrôle la sortie vidéo d'un ordinateur ;

Figure 11

- un **composant matériel dédié** (cela peut être un ordinateur avec seulement les ressources matérielles nécessaires pour exécuter le serveur « X » et visualiser des applications fonctionnant sur des serveurs) ;
- une **application** qui affiche des données dans une fenêtre d'un autre système graphique.

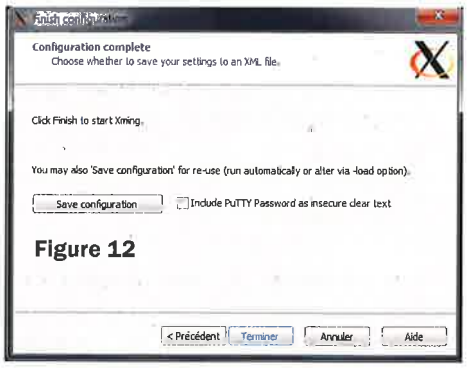


Figure 12

« **Xming** » est téléchargeable à l'adresse :

http://sourceforge.net/projects/xming/

Nous le mettrons aussi sur notre site dédié au **RaspberryPi** (voir plus haut dans l'article). Le téléchargement et l'installation s'effectuent comme toute autre application Windows.

Une fois le programme installé, allez dans le menu « **Démarrer** » → « **Tous les programmes** » → « **Xming** » puis cliquez sur « **XLaunch** ». Vous devez obtenir une fenêtre similaire à celle de la figure 8, refermez l'application en cliquant sur le croix rouge en haut à droite de la fenêtre.

Vous trouverez aussi dans le menu le programme « **Xming** » qui permet d'**exécuter à distance des applications individuelles dans des fenêtres séparées de Windows**.

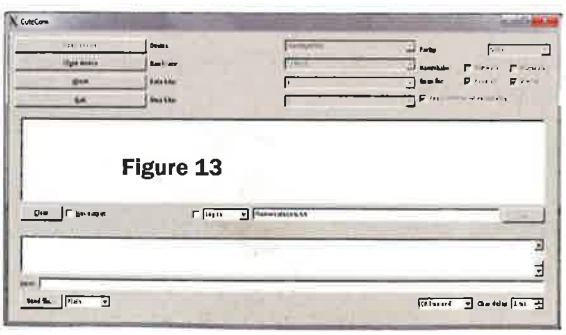


Figure 13

Une fois « Xming » installé, lancez « **CuteCom** » à distance sur le bureau de votre PC. Ouvrez « **XLaunch** » (figure 8) et sélectionnez « **Multiple Windows** » et cliquez sur « **Suivant** ». Dans la fenêtre suivante sélectionnez « **Start a program** » (figure 9) et cliquez encore sur « **Suivant** ».

Dans la nouvelle fenêtre sélectionnez « **Using PuTTY (pink.exe)** ». Dans le champ « **Start program** » sélectionnez « **CuteCom** », dans le champ « **Connect to computer** » insérez l'adresse IP du RaspberryPi. Dans notre cas l'IP est 192.168.0.43.

Dans le champ « **Login as user** » tapez « **root** » et dans le champ « **Password** » votre mot de passe (pi dans notre cas). Cliquez sur « **Suivant** » vous devez obtenir la figure 11.

Cochez la case « **Clipboard** » si cela n'est pas déjà fait et cliquez de nouveau sur « **Suivant** », vous arrivez sur la fenêtre de la figure 12 où vous cliquez sur « **Terminer** ». Répondez « **Ok** » et « **No** » aux **messages suivants** et attendez. Au bout de quelques instants vous obtenez la fenêtre de la figure 13.

Gestion du module GSM par l'émulateur TTY

La fenêtre de « **CuteCom** » est divisée en **plusieurs** « **zones** ».

En haut se trouvent les champs et les boutons pour configurer, activer et désactiver la communication série. En bas se trouve un « champ texte » qui permet de saisir des commandes à envoyer sur le port série, et qui garde et affiche l'historique des commandes. Tout en bas des commandes de configuration permettent de :

- activer ou désactiver un fichier journal (fichier log) ;
- ajouter la commande « fin de ligne » ;
- transmettre et/ou recevoir des données au format ASCII ou hexadécimal.

Enfin, dans le milieu de la fenêtre, nous pouvons voir la zone où sont mis en évidence les messages entrants en provenance du port série.

La première chose à effectuer est de configurer le nom du périphérique (device) et son chemin du port série auquel nous voulons le connecter. Tapez la commande :

```
/dev/tty/AMA0
```

Si des valeurs prédéfinies ne sont pas trouvées, nous écrivons directement la valeur dans le champ de configuration.

Effectuons la configuration suivante pour le port en insérant les valeurs suivantes : **115200** pour la vitesse, **8 bits** pour la longueur des caractères, **1 pour le bit d'arrêt** (stop bit) et « **none** » pour la **parité** c'est-à-dire pas de parité.

Nous ne cochons aucune case pour l'« handshake » (procédure d'établissement d'une communication entre deux matériels ou logiciels).

Par contre pour l'option « **Open for** » nous **cochons les cases** « **Reading** » (lecture) et « **Writing** » (écriture). Les cases « **Hex output** » et « **Log out** » ne doivent pas être cochées.

Dans le **menu déroulant en bas à gauche** à côté le bouton « **Send file** », sélectionnons « **Plain** », en face à droite sélectionnons « **CRline end** » dans le menu déroulant et enfin dans le menu « **Char delay** » réglons la valeur à **1 ms**.

Maintenant, nous pouvons commencer les tests. Tout d'abord vérifiez que le module **GSM SIM9XX a bien une carte SIM insérée**, qu'**il est allumé et connecté au réseau mobile** en vérifiant que la **LED verte clignote lentement**. Sinon, allumons-le en tapant la commande suivante :

```
./GSMallumage
```

Ensuite, appuyons sur le bouton « **Open device** » dans le coin supérieur gauche de la fenêtre principale de « CuteCom ».

Si tout est en ordre, nous apercevons la zone de configuration en haut devenir « grisée » et nous pouvons commencer à envoyer des commandes en les insérant dans le champ en dessous.

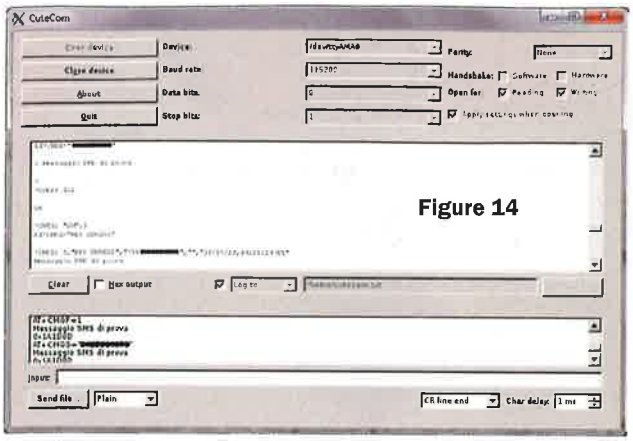


Figure 14

Commençons par la commande « **AT** », nous la tapons dans le champ approprié et appuyons sur le bouton « **Send file** ».

Normalement nous devons recevoir la réponse « **OK** », si cela n'est pas le cas il se peut que le module ne soit pas en « mode de commande ». Sinon activons le « mode commande » (Command Mode) avec la commande « **+++** ».

Si nous nous référons à la documentation technique du module **SIM9XX**, nous pouvons tester l'effet de chaque commande, soit pour la configuration du module, soit pour l'exécution des séquences de commandes de fonctionnement.

Voici un exemple, connectons un haut-parleur à la sortie audio et un microphone à la prise MIC, ou utilisons un casque avec un micro ayant des prises séparées.

Essayons d'envoyer et de recevoir un appel. Tout d'abord tapons la commande :

AT+CALM=0

qui permet d'activer la sonnerie (nous pouvons également définir la mélodie à l'aide du manuel technique).

Maintenant, tapons la commande

ATD <numéro de téléphone>

suivie par un point-virgule (;).

Par exemple : **ATD + 336 ..... 7;**

Lorsque le téléphone appelé accepte l'appel, nous pouvons parler à notre interlocuteur.

Pour mettre fin à l'appel de notre côté, tapons la commande :

ATH

Si un téléphone externe nous appelle, nous entendons la sonnerie dans le haut-parleur et à chaque sonnerie nous voyons le message « RING » (sonnerie) s'afficher dans la zone centrale de la fenêtre de « CuteCom ».

Pour répondre à l'appel entrant, tapons la commande :

ATA

Pour envoyer un SMS, tapons selon la séquence ci-après les commandes :

AT+CMGF=1
AT+CMGS="<numéro de téléphone>"

Ne pas oublier de saisir le numéro de téléphone entre guillemets. Attendons le message de retour : **">"**

Tapons le texte du message, par exemple :

Message SMS de test

cette fois sans les guillemets, enfin sélectionnons l'avant dernier champ en bas à droite « Hex input » et insérons :

0x1A1D0D

puis cliquons sur « Send file » (c'est l'équivalent de CTRL-Z pour fermer un message de type SMS).

Pour continuer, nous devons reporter la valeur de l'avant-dernier champ en bas à droite dans le champ « CR line end ».

Pour lire la liste des messages SMS reçus et non lus, tapons la commande :

AT+CMGL="REC UNREAD"

Nous obtenons la figure 14.

+CMGL: 3,"REC UNREAD","+336.....1","","15/07/23, 14:29:24+08"

Message SMS de test

où « 3 » correspond à l'index du message. Nous pouvons lire le message individuellement, en utilisant l'index avec la commande :

AT+CMGR=3

Enfin, si vous avez installé un module **SIM908** équipé d'un GPS, vous pouvez essayer de lire la position (avec l'antenne extérieure) en utilisant la commande :

AT+CGPSINF=0

Vous obtenez comme résultat une chaîne composée des champs : mode, longitude, latitude, altitude, date/heure, TTFF (Time To First Fix), le nombre de satellites, la vitesse et la direction de la boussole.

Vous pouvez choisir d'utiliser les commandes natives afin de créer des fonctionnalités que vous pourrez ensuite encoder dans des programmes structurés.

Nous approfondirons par la suite la mise en œuvre d'applications structurées et modulaires.

Nous étudierons aussi dans le « Cours Arduino » des exemples de gestion de modules GSM que vous pourrez adapter au RaspberryPi.

Dans le prochain numéro, nous vous proposerons de réaliser une carte d'extension pour RaspberryPi pouvant gérer jusqu'à 64 sorties et 64 entrées digitales grâce au bus I^2C.

Les typons des circuits imprimés sont téléchargeables gratuitement sur notre site :

www.electroniquemagazine.com

dans le sommaire détaillé de la revue numéro **133** section « **Télécharger** ».

Les librairies sont téléchargeables à l'adresse suivante :
www.raspberrypi.electroniquemagazine.com/telechargement.html

ET849

Dans cet article, nous vous proposons de construire un outil de diagnostic pour les unités électroniques de contrôle des véhicules. Cet outil affiche les principaux paramètres de fonctionnement du moteur à travers un PC et l'état des systèmes afin de réduire l'émission de polluants. Il permettrait aussi la reprogrammation, mais là c'est un autre domaine...

Testeur EOBD pour dlagnostlque auto

d'Alessandro SOTTOCORNOLA

Combien de fois avez-vous entendu parler de normes « **Euro 1** », « **Euro 2** », « **Euro 3** », etc. ? Certainement lorsque les journaux télévisés annoncent certaines restrictions de circulation dues à la pollution ambiante ou lorsque vous décidez d'acheter un nouveau véhicule neuf.

Les normes « **Euro** » sont des règlementations de l'**Union Européenne** qui fixent les **limites maximales de rejet des polluants des véhicules essence ou diesel** et permettent de définir si une voiture pollue.

La norme « **Euro 1** » a imposé l'utilisation du **catalyseur,** aussi bien sur les moteurs essence à injection électronique que sur ceux sans gestion électronique. Elle a aussi introduit l'utilisation de la vanne « **EGR** » pour les moteurs diesels.

Celle-ci permet la **recirculation des gaz d'échappement** afin de réduire les émissions d'oxydes d'azote. La norme « **Euro 2** » impose pour les véhicules essence et diesel une injection à commande électronique et un catalyseur.

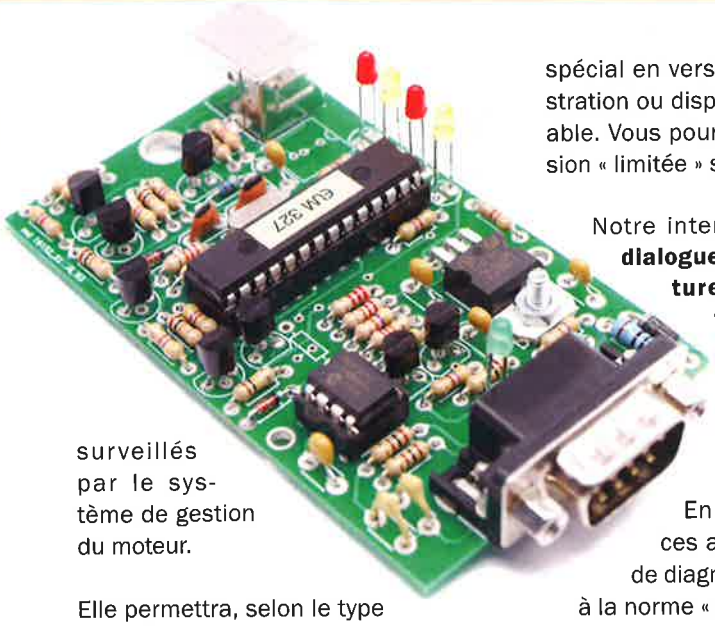
La véritable avancée en matière de norme antipollution apparaît avec la norme « **Euro 3** », qui ne concernait pas l'ajout de systèmes d'antipollution, mais la possibilité d'**avertir le**

conducteur de la défaillance d'un de ces systèmes, à savoir la sonde « **lambda** » du catalyseur ou la vanne « **EGR** ».

Le système de diagnostic des dispositifs de réduction des émissions de polluants est appelé « **EOBD** » (Enhanced On Board Diagnostic) et, dans les véhicules respectant la norme « **Euro 3**, » est intégrée une unité centrale électronique de gestion du moteur appelée « **OBD** » qui utilise le même connecteur.

Cette unité centrale, en plus de dialoguer avec les différents capteurs et dispositifs du véhicule, peut **communiquer avec des outils de diagnostics** appropriés utilisés dans les ateliers mécaniques. Ainsi pour le réparateur, avec une simple valise diagnostique ou un PC équipé d'une interface appropriée, il devient relativement facile de détecter les éventuelles pannes ou dysfonctionnements du moteur.

Dans ces pages, nous vous proposons de **réaliser une interface capable de communiquer avec le système** « **EOBD** » de votre voiture. Elle pourra interroger l'unité centrale à l'aide d'un PC et vérifier les paramètres et l'état de fonctionnement des principaux organes



surveillés par le système de gestion du moteur.

Elle permettra, selon le type d'unité centrale de votre véhicule, du logiciel téléchargé et des restrictions imposées par les constructeurs, d'afficher les données concernant le nombre de tours moteur, la température de l'air et de l'eau, la quantité d'air aspirée, le fonctionnement de la vanne EGR, et beaucoup d'autres données spécifiques à chaque véhicule.

Elle permet aussi de découvrir, après l'énième retour chez le mécanicien, où se situe le problème. Cette situation n'est pas rare, en particulier si la voiture est sous garantie. C'est ce qui est arrivé à une personne proche de notre entourage, l'analyse d'un véhicule avec notre système de diagnostic a montré un problème dans le débitmètre, ce que le concessionnaire Opel, après de nombreuses interventions, n'avait pas remarqué...

Notre interface est le lien entre l'unité centrale du véhicule et le PC, dans lequel vous aurez à installer un logiciel spécial en version limitée de démonstration ou disponible à un prix abordable. Vous pourrez télécharger la version « limitée » sur notre site.

Notre interface est capable de **dialoguer avec toutes les voitures fabriquées à partir de 2001** en ce qui concerne les **modèles essence** et depuis **2003** pour les **modèles diesel**.

En effet c'est à partir de ces années que le système de diagnostic a été standardisé à la norme « **EOBD** » qui prévoit une **connexion normalisée** ainsi qu'un protocole de communication valable pour tous les véhicules.

Mais pourquoi cela ? Certainement pour aider les mécaniciens et les électriciens auto. En effet, utiliser un connecteur et un protocole de communication propriétaire pour chaque constructeur, aurait obligé les fabricants d'outils de diagnostic de produire autant de modèles qu'il y a de types de véhicules, mais aussi les garagistes à s'équiper d'autant de différents outils, ce qui financièrement n'est pas viable.

Cependant la standardisation, réalisée à partir de la norme « **Euro 3** », a principalement permis aux organismes chargés des contrôles de la pollution, lorsqu'une voiture « fumait trop », d'analyser et de vérifier si les systèmes de dépollution étaient opérationnels ou si une défaillance se produisait.

En résumé, la norme « **Euro 3** » a contraint les constructeurs automobiles à produire des véhicules avec un port de communication standard, grâce auquel les **divers organismes de contrôle peuvent effectuer un diagnostic avec un seul appareil**. Il y a quelques années les forces de l'ordre vérifiaient la pollution des véhicules diesel en introduisant un opacimètre en sortie des gaz d'échappement sur le bord de la route.

Les voitures à essence depuis 2001 et les diesels depuis 2003 sont équipées d'unités de contrôle qui communiquent avec l'outil de diagnostic à l'aide d'un bus **CAN**. Aujourd'hui il existe 5 bus de communication différents :

- **SAE-J1850 PWM** (General Motors) ;
- **SAE-J1850 VPW** (Ford) ;
- **ISO-9141**(Chrysler, véhicules européens et asiatiques) ;
- **ISO-14230** (appelé Keyword Protocol 2000 ou KWP 2000, nouvelle version du ISO-9141 plus rapide) ;
- **ISO-15765-4 CAN BUS** (équipe déjà de nombreux véhicules, toutes marques confondues, standard sur tous les modèles à partir de 2008).

Des subtilités se situent au niveau du codage du bus pour la transmission des informations entre le calculateur et l'outil de diagnostic. Le codage de ces informations n'est pas propriétaire ni secret, il est standardisé par un organisme de normalisation (SAE ou ISO) et ces informations sont publiques.

Mais alors faut-il un appareil par protocoles ? À l'heure actuelle, le fait que plusieurs protocoles coexistent n'est pas réellement un problème dans la

Notre circuit est une interface qui permet d'interroger, en utilisant un logiciel spécial, l'unité centrale d'un véhicule à l'aide d'un PC.

mesure où il existe des composants électroniques capables de tous les décoder.

D'autre part, le bus **CAN** développé par les Allemands est obligatoire sur tous les véhicules à partir de 2008. Le dernier composant capable de décoder tous les protocoles actuels se nomme le « **ELM327 révision 1.2a** ».

Ces différents protocoles de communication ont été adoptés par les différents constructeurs au cours de ces dernières années. Mais cela ne signifie pas qu'avec un outil standard vous pouvez avoir accès à l'ensemble des paramètres d'un véhicule donné, vous ne pouvez avoir accès qu'à ce que la loi vous permet.

D'après les tests que nous avons effectués sur les véhicules de différents constructeurs (Ford, Mercedes, Volkswagen, Audi, Opel, etc.), nous avons remarqué que certains donnent accès à presque toutes les données (c'est le cas d'Opel et de Ford) tandis que les marques premiums Allemandes telles qu'Audi sont avares en informations.

L'accès à des données confidentielles et la modification de certains paramètres de fonctionnement, tels que l'élimination des codes d'erreurs ou la modification du kilométrage, sont généralement réservés au personnel autorisé et vous ne pouvez pas y accéder.

Cela a été voulu en partie par les constructeurs automobiles et par le législateur afin d'éviter la désactivation des dispositifs d'antipollution (par exemple, la déconnexion de la vanne « **EGR** » augmente les performances du moteur mais aussi la pollution) ainsi que la reprogrammation de la gestion du moteur permettant une augmentation de puissance.

Nous vous conseillons donc, **lorsque vous achetez une voiture d'occasion, de vérifier l'historique des contrôles techniques**. Pour cela, il suffit de contacter l'« **OTC** » (Organisme Technique Central, c'est une division de l'UTAC) qui collecte les déclarations des passages au « CT » de tous les véhicules en France des différents réseaux. Il suffit d'envoyer à cet organisme (UTAC - OTC

Le circuit intégré ELM327

Pour communiquer avec l'unité centrale d'un véhicule, notre analyseur utilise un microcontrôleur PIC personnalisé, dénommé **ELM327**, et dont le « firmware » est capable de communiquer avec un bus de type « **CAN** », « **SAE** » et dispose d'une liaison de type bus « **K-Line** ». Cela est possible par l'intermédiaire d'une interface matérielle simple qui est visible sur le schéma électrique de notre projet (voir la figure A). Grâce à ces 3 types de connexions qui sont à la norme « **OBD2** », **notre interface peut gérer jusqu'à 12 protocoles différents**, donc pratiquement tous les véhicules respectant la norme « **EURO3** » produits depuis 2001 pour l'essence et depuis 2003 pour le diesel. Les 3 bus de communication sont générés par des broches spécifiques qui sont les suivantes :

- broches 3, 4, 11, 13 et 14 pour l'interface « SAEJ1850 » ;
- broches 12, 21 et 22 pour le bus « K-Line » (normes ISO9141 et ISO14230) ;
- broches 23 et 24 pour le bus « CAN ».

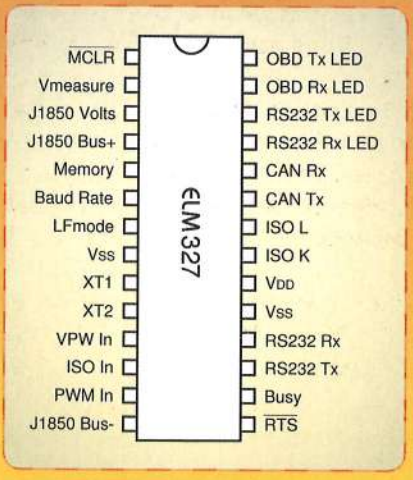
Quels que soient le protocole et l'interface utilisés pour le dialogue, les données lues sur la prise « **EOBD** » sont envoyées vers l'interface **série/USB** (FT232RL) via la broche « TX » (17) du microcontrôleur. Les données envoyées à partir du PC vers l'unité centrale du véhicule (interrogations de communications et paramètres) sont reçues sur la broche « RX » (18) à travers la même interface série/USB.

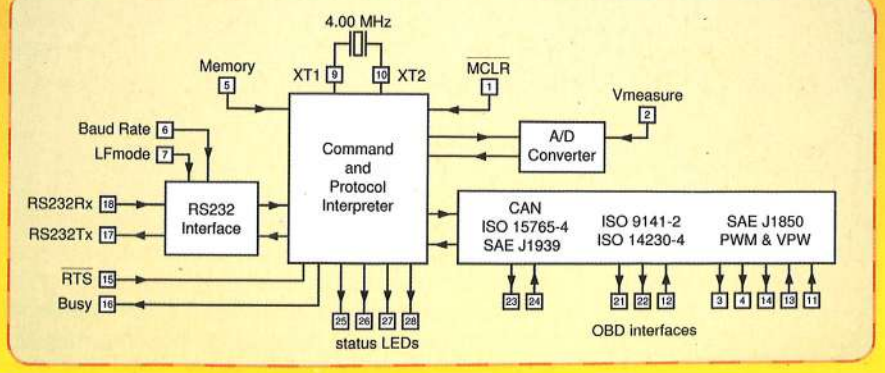
La vitesse de communication série est déterminée par la broche « BAUD » (6) qui est lue immédiatement après la mise sous tension. Si elle est à un niveau haut, la vitesse de communication est de 38400 bps, si elle est à un niveau logique 0, la vitesse est réduite à 9600 bauds. Dans notre cas la vitesse sera toujours de 38400 bps.

Une autre fonction du « ELM327 » est la la mesure de la tension d'alimentation via la broche 2 à l'aide du convertisseur A/N interne.

Quatre broches, configurées pour fonctionner en mode « sink » (drainent le courant), permettent de gérer 4 LED de signalisation. LD4 visualise l'état de la broche « RX », LD3 signale l'activité de la broche « TX », LD2 signale la réception des données de l'« OBD » et LD1 signale l'émission des données vers l'« OBD ».

La réception et l'émission des données de l'« OBD » désignent l'échange des données indifféremment pour les 3 bus (« SAE », « K-line » ou « CAN »).





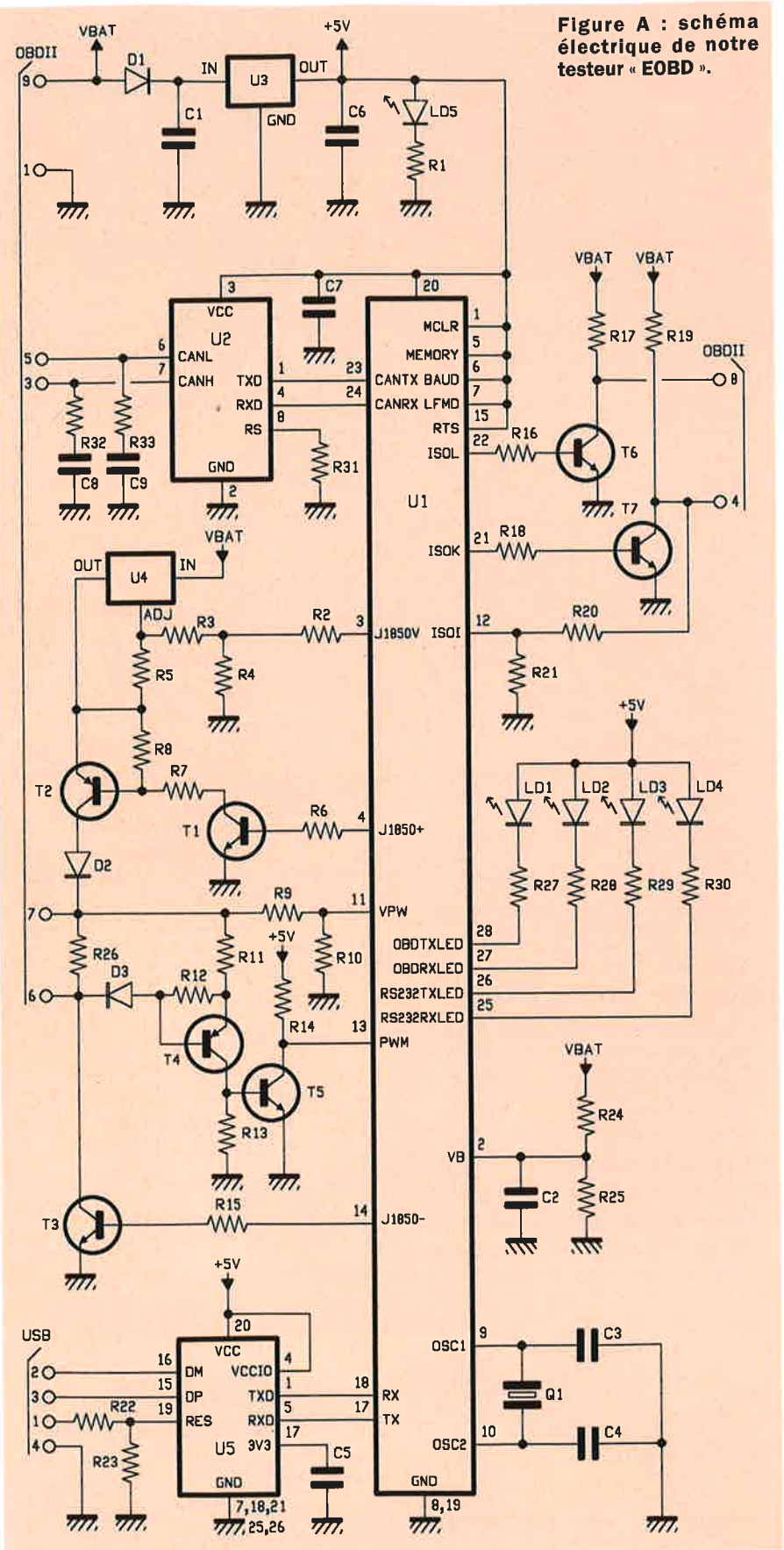


Figure A : schéma électrique de notre testeur « EOBD ».

- Autodrome de Linas-Montlhéry - BP 20212 - 91311 Montlhéry Cedex) une copie du certificat d'immatriculation et votre carte d'identité. Vous recevrez la liste de tous les « CT » du véhicule, avec la date, le centre contrôleur, le numéro de rapport, le kilométrage relevé, et le résultat du contrôle, ainsi que les adresses des centres de contrôle ayant vérifié le véhicule.

Notre analyseur

Le diagnostic est réalisé à l'aide d'une interface matérielle (hardware), d'un logiciel spécialisé et bien sûr d'un PC. L'interface est assurée par un **convertisseur multistandard** qui adapte les signaux des différents bus **CAN-Bus/SAEJ1850/ISO9141-2** vers une interface **USB**. Il est alors possible de **lire les données** à partir du connecteur « **EOBD** » et de les adapter sous forme de signaux compatibles **USB**.

Bien sûr, cela n'est pas facile car notre interface doit communiquer avec l'unité centrale des véhicules et doit être en mesure de gérer au moins 3 des 5 protocoles utilisés par les unités à bord des voitures.

Mais pourquoi autant de protocoles ? L'explication réside dans le fait qu'à l'origine chaque constructeur disposait de sa propre connexion pour leurs unités de contrôle. Ainsi sont nés les différents protocoles qui ont ensuite été unifiés par les normes « ISO » et « SAE », en établissant un cahier des charges précis en termes de signaux de synchronisation, de niveaux de tensions, mais aussi en termes de connectique et cela a amené au connecteur « OBD ».

Ce terme veut dire précisément « On Board Diagnostic » c'est-à-dire « Diagnostic embarqué » et indique si le système de gestion du moteur est défaillant ou pas.

De nos jours, le **BUS-CAN** est largement le plus utilisé dans les systèmes de diagnostics, ces systèmes sont appelés « **OBD2** ». Mais avec l'arrivée de la norme « **Euro 3** » et des suivantes, ces systèmes de diagnostics ont évolué et sont désormais appelés « **EOBD** ».

Bien que partageant le même connecteur, les systèmes « **OBD** », « **OBD2** et « **EOBD** » peuvent travailler avec les différents protocoles de communication suivants : le « K-Line » (variante des normes « ISO 9141-2 » et

« ISO 14230-4 », également appelé « KWP2000 ») ; le FORD-SAE (J1850 PWM et VPW) et ceux basés sur le CAN (ISO15765-4 et SAE J1939).

C'est pour cette raison qu'un système de diagnostic valable doit être capable de reconnaître au minimum des données provenant de l'une de ces 3 normes. Il doit ensuite traiter les informations et les convertir au format série/USB avec une synchronisation exacte.

La gestion des protocoles de communication avec l'« **EOBD** » est confiée à un circuit intégré spécifique dénommé « **ELM327** » qui est un **microcontrôleur PIC** de chez **Microchip**, disposant d'une interface appropriée et programmé pour gérer les **12 protocoles**, y compris ceux du « CAN », du « J1850 » et de l'« ISO9141-2 ». Le microcontrôleur adapte ensuite tous les signaux de ces différents protocoles en signaux standards compatibles TTL.

Pour des raisons de sécurité et des dommages que pourrait créer un microcontrôleur mal programmé à l'électronique d'un véhicule (la facture peut monter à plusieurs milliers d'euros), le « **ELM327** » est disponible dans le commerce déjà programmé en usine.

A travers l'UART interne dont il dispose, il restitue les données lues à partir de la prise « **EOBD** » vers un ordinateur de type PC.

Les lignes d'émission/réception sont gérées par un circuit intégré « FT232RL » qui convertit les signaux TTL en USB à la norme 2.0. Le logiciel qui fonctionne sur l'ordinateur déchiffre les données entrantes par la prise USB et affiche les résultats à l'écran.

Le schéma électrique

Examinons la composition du schéma électrique, le cœur du montage est le microcontrôleur programmé en usine et dénommé « **ELM327** ». Il dispose d'un nombre important de broches assurant les différentes fonctions d'entrées/sorties (I/O).

Le microcontrôleur communique avec la prise USB, par l'intermédiaire d'un

Les voitures deviennent de plus en plus autonomes !

Le jour où nous monterons à bord de notre voiture et qu'elle nous demandera « Indiquez votre destination », se rapproche de plus en plus. Cela a été démontré par des expériences récentes menées sur des prototypes de véhicules robotisés, capables de conduire eux-mêmes dans la circulation et d'atteindre l'objectif demandé par le conducteur. C'est le cas de la voiture sans conducteur de Google, appelée « Google Car », qui est un projet de voiture autonome développé par Google. Le système de pilotage automatique utilise un « lidar » (télédétection par laser utilisant la lumière), une caméra, des radars, un récepteur GPS et des capteurs sur les roues motrices. Le véhicule doit, une première fois, être conduit de manière ordinaire sur le trajet qu'il empruntera ensuite de manière autonome, afin qu'une équipe vérifie que le véhicule a enregistré tous les signaux importants le long du trajet. Ensuite, le véhicule n'a plus qu'à gérer les modifications de signalisation pendant son trajet autonome. Une des limitations du système est son incapacité à agir suivant les gestes d'un agent de police faisant la circulation.

Un projet intéressant a été développé par des ingénieurs Japonais de la firme « ZMP Inc », il s'agit de la « RoboCcar » un prototype entièrement fonctionnel pouvant se déplacer de manière autonome sur une route réelle. La « RoboCar » est de taille modeste : 2,40 mètres de longueur, 1,60 mètre de hauteur et 1 mètre de largeur. Elle ne peut servir qu'à déplacer une personne sans bagages et sur de faibles distances. Deux caméras CCD regardent vers l'avant et fournissent une vision en relief à l'ordinateur de bord. Un système optique à laser, installé sur l'avant, repère les obstacles proches, tout comme les 8 capteurs infrarouges. Elle est dotée en plus d'un système inertiel qui mesure les accélérations et permet ainsi de situer la position du véhicule à 1 ou 2 centimètres près. L'ordinateur de bord fonctionne sous Linux.

Une autre expérience intéressante a été menée par NISSAN au « CEATEC » de Tokyo, la marque a présenté le concept « EPORO » qui correspond à une petite voiture avec une seule roue. Cela ressemble plus à un robot autonome, mais ce concept démontre la possibilité de construire des véhicules qui peuvent voyager en groupe, tout en évitant les obstacles, mais aussi éviter une collision les uns avec les autres, tout comme les poissons dans l'océan. Le projet vise à la production de véhicules qui ne peuvent pas entrer en collision, cela améliore le trafic et réduit le nombre d'accidents.

circuit intégré « FT232RL », utilisé dans sa configuration classique avec le générateur d'horloge interne.

Ce circuit intégré permet de convertir des signaux séries vers un port de communication USB. Sa vitesse de transmission peut être paramétrée et il est compatible avec les interfaces USB 1.0 et USB 2.0. L'interface « série/USB » est donc réalisée par le convertisseur U5 qui est connecté aux broches « RX » et « TX » de l'UART du microcontrôleur U1.

La communication avec la prise « **EOBD** » du véhicule est réalisée via **3 interfaces** différentes. La 1ère interface CAN est réalisée par le convertisseur « CAN-BUS/TTL » qui est un circuit **MCP2551** (U2). C'est un **transmetteur** « **CAN** » (émetteur/récepteur) compatible avec la norme « **ISO11898** » et

capable de convertir les signaux d'un bus « CAN » en signaux compatibles TTL et vice versa.

Les lignes « **TXD** » et « **RXD** » du circuit U2 sont utilisées comme interface, respectivement pour les broches « **CANTX** » et « **CANRX** » du microcontrôleur « ELM327 » réservées pour la communication avec la prise « EOBD ».

Les broches 3, 4, 11, 13 et 14 du microcontrôleur U1 gèrent le bus d'interface selon le protocole « **SAE J1850** », souvent intégré dans le connecteur « EOBD » et utilisé avant la standardisation dans certains outils de diagnostics. Ce bus existe en 2 versions. L'une fournit un signal de données à 41,6 kbps de type « **PWM** » (modulation en largeur d'impulsion) et fonctionne sur 2 fils en mode différentiel, tandis que l'autre génère un signal à 10,4 kbps de type « **VPW** » (largeur d'impulsion variable) sur un seul fil unique (cela permet des connexions avec des longueurs de câble allant jusqu'à 35 mètres avec un maximum de 32 nœuds).

Dans notre montage, la broche 3 (J1850V) de **U1** fournit au régulateur U4 (LM317) la **tension de référence** nécessaire pour que dernier génère une **tension correcte** afin d'alimenter la **broche 7** du connecteur « **OBD2** ». La tension de référence appliquée au régulateur dépend du signal appliqué sur la base de T1 à travers R6 (broche 4 ou « J1850+ »).

Pour être précis, lorsque la broche « J1850+ » est à un niveau haut, le transistor T1 devient conducteur et le collecteur de T2 fournit à travers la diode D2 une tension positive sur la broche 7 du connecteur « OBD2 ». Si la broche « J1850+ » (4 de U1) est à un niveau bas, alors T1 est bloqué et donc T2 aussi, de sorte que la tension sur la cathode de D2 soit proche de 0 V.

La broche 14 (« J1850- ») du circuit « ELM327 », c'est-à-dire de U1, est reliée à travers R15 à la base du transistor T3 et fournit des impulsions à la broche 6 du connecteur « OBD2 ». De cette tension (broche 6) est dérivé un potentiel qui est ensuite utilisé par la broche 13 de U1 (entrée PWM) et la broche 11 (entrée VPW). L'interface « J1850 » est une alternative au « CAN BUS ».

La dernière connexion de la prise « **EOBD2** », utilisée pour communiquer avec l'« ECU » du véhicule, est l'interface « **K-Line** » (ISO9141-2 ou ISO14230-4) gérée via les broches 12 (ISOI), 21 (ISOK) et 22 (ISOL) du microcontrôleur « ELM327 ».

En particulier, la broche 4 du connecteur « **EOBD2** » est utilisée en mode **bidirectionnel**. Elle se comporte comme une sortie (OUTPUT) lorsque T7 est conducteur, et comme une entrée (INPUT) lorsque T7 est bloqué. Les broches « **ISOK** » et « **ISOL** » sont par contre **2 lignes de sortie** dirigées vers l'unité de contrôle du véhicule.

La broche « **IOSI** » fonctionne en **entrée** et est gérée en interne par un comparateur capable de lire les niveaux logiques intermédiaires lorsque le transistor T7 est bloqué (la ligne « ISOK » est à un niveau logique bas 0).

La broche 4 du connecteur « OBD2 » reçoit les impulsions de la ligne de transmission du bus « ISO ». De la même manière l'interface « K-Line » est une alternative au « CAN-BUS ». Enfin la broche 2 (VB) du circuit « ELM327 » est reliée au convertisseur A/D afin de mesurer la tension d'alimentation.

A propos de l'alimentation, notez que toute l'interface est alimentée par la tension de 12 V fournie par le connecteur « OBD2 » du véhicule. Pour être exact, la broche 9 amène le 12 V positif (VBAT) et la broche 1 distribue la masse (GND).

La tension « VBAT » est directement utilisée par l'interface « K-Line » à l'aide des 2 transistors et par l'interface « J1850 » à travers le régulateur U4. La tension « VBAT » est aussi appliquée au diviseur de tension formé par les résistances R24 et R25, afin d'être mesurée par le convertisseur A/D interne du ELM327.

Le régulateur U3 (7805) fournit une tension stabilisée de 5 V alimentant le convertisseur « CAN-BUS/TTL » (U2), le convertisseur « série/USB » (U5), le microcontrôleur « ELM327 » et les LED de signalisation.

Fonctionnement du circuit

Une fois notre montage connecté à la prise diagnostic du véhicule, le circuit s'initialise (après l'initialisation des I/O, le microcontrôleur « ELM327 » allume

de manière séquentielle les 4 LED pour indiquer qu'il est prêt) et tente de dialoguer avec le système « EOBD ».

Plus précisément, il **vérifie lequel des 3 bus possibles existe** et s'il est **actif**, comme par exemple le bus « CAN » ou le « SAE J1850 » ou encore le « K-Line ». Lors de la connexion, le montage interroge l'unité centrale du véhicule et vérifie le protocole utilisé.

Si le matériel est reconnu et que les signaux sont corrects mais que l'électronique du véhicule adopte un protocole inconnu par rapport aux 12 gérés par le circuit « ELM327 », celui-ci envoie sur le port USB le message « **Protocole non supporté** » qui sera affiché sur l'écran de l'ordinateur dans la fenêtre principale.

Pour un bus de type « SAE J1850 », le circuit « ELM327 » vérifie dans l'ordre la présence du signal « PWM » (pour une liaison dont le protocole est de type « PWM ») et du signal « VPW » (pour un protocole de type « VPW »).

Après la détection du type de connexion, le circuit est prêt à recevoir les commandes séries provenant du PC, elles doivent être conformes au format des commandes « AT ».

Le logiciel de gestion

À ce stade, vous devez installer le programme sur votre PC. Pour nos tests, nous avons utilisé une version de démonstration nommée « **ScanMaster-ELM-DEMO** » et que vous pouvez télécharger gratuitement sur notre site **www.electroniquemagazine.com** dans le sommaire détaillé du numéro **133** à l'onglet « **Télécharger** ». Vous y trouverez aussi les typons du circuit imprimé et une liste des codes d'erreurs (voir plus haut).

Une fois le programme installé, vous devez le lancer et aller dans le menu « **Options** » en haut à gauche. Sélectionnez « **Langage** ». Une fenêtre s'ouvre nommée « **GUI Langage** », choisissez « **French** » dans le menu déroulant. Fermez et redémarrez le programme. Les principaux menus doivent normalement s'afficher en Français.

Une fois que vous avez connecté l'interface au PC, en lançant le programme, vous obtenez la fenêtre principale visible en figure 1. La fenêtre de dialogue **affiche l'interface détectée et l'état de la connexion avec l'interface**, soit **elle est correcte soit elle est en défaut**. Plus précisément, dans notre cas, les informations concernent l'état de la connexion, le type d'« ECU » détectée et le protocole utilisé (la figure 1 indique un bus CAN identifié avec la norme ISO 15765-4). Si la communication est impossible, les messages suivants apparaissent « **Unrecognized ECU** » (« ECU » non reconnue) ou « **Unrecognized Protocol** » (protocole non reconnu).

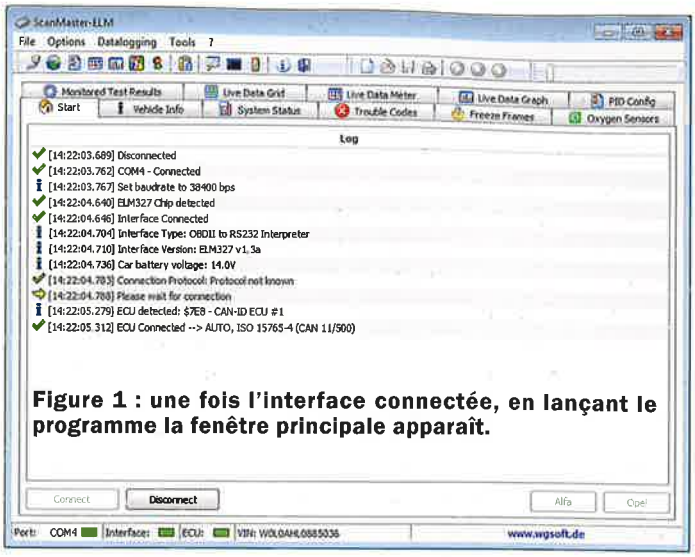


Figure 1 : une fois l'interface connectée, en lançant le programme la fenêtre principale apparaît.

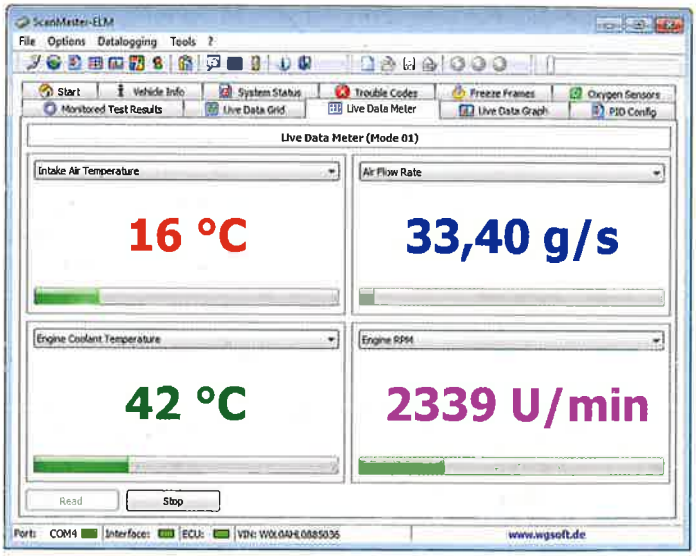


Figure 2 : le programme affiche en temps réel les paramètres sélectionnés à l'aide des menus contenus dans chaque onglet.

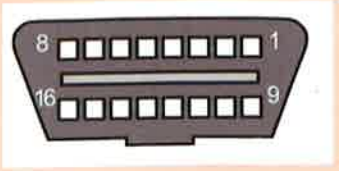
Une fois la connexion établie, en cliquant sur les différents onglets vous accédez aux différentes fonctions du programme. Une fonction très intéressante est celle correspondant à l'onglet « **Compteur de Données (dynamiques)** » (« Live Data Meter »).

En cliquant sur l'onglet, 4 rectangles apparaissent avec un chiffre au milieu, le logiciel peut afficher en temps réel l'un des paramètres sélectionnés dans le menu déroulant qui se situe au-dessus de chaque nombre (voir la figure 2). Dans ce mode, vous pouvez choisir quel paramètre afficher en continu.

Une autre fonction très intéressante qui résume tous les paramètres disponibles par l' « ECU » à travers le connecteur « OBD2 » est la fonction « **Grille de Données (dynamiques)** » (« Live Data Grid ») accessible via l'onglet portant le même nom (voir la figure 3). Ici, nous observons en temps réel les principaux paramètres du moteur.

La dernière fonction que nous analysons est celle qui permet de visualiser les codes d'erreurs liés à des anomalies détectées par l'unité de contrôle et qui sont mémorisées. Cette fonction est accessible via l'onglet « **Codes Erreurs** » (« Trouble Codes ») (voir la figure 4). Une liste exhaustive est disponible sur notre site comme expliqué plus haut.

Le connecteur EOBD



Le connecteur **OBD2** supporte les bus « **CAN** », « **SAE** » et « **K-Line** » ; le brochage est le suivant :

2 : J1850 Bus+ (correspond à la broche 7 de notre circuit) ;
4 : Masse du châssis (correspond à la broche 1 de notre circuit) ;
5 : Signal Ground ;
6 : CAN-Bus High (J-2284) (correspond à la broche 1 de notre circuit) ;
7 : K-Line ISO 9141-2, et ISO/DIS 14230-4 ligne K (correspond à la broche 4 de notre circuit) ;
10 : J1850 Bus– (correspond à la broche 6 de notre circuit) ;
14 : CAN-Bus Low (J-2284) (correspond à la broche 5 de notre circuit) ;
15 : K-Line ISO 9141-2 et ISO/DIS 14230-4, lignes L et I (correspond à la broche 8 de notre circuit) ;
16 : Tension de la batterie (correspond à la broche 9 de notre circuit).

Les bus implémentés sont le « SAE J1850 » (broches PWM et VPW), le « BDLC-D » (Byte Data Link Controller), l'« ISO9141-2/ ISO14230-4 K-Line KWP1281/KWP2000 », et le bus CAN (SAE J2284/ ISO15765). Dans certains cas, le bus « LIN » (SAE J2602) est intégré mais il est facultatif.

Fonctionnement de la détection EOBD

Le **calculateur** regarde en permanence la régularité du **signal volant moteur**. Les ratés de combustion provoquent un **acyclisme** moteur (donc une chute de couple). L'observation d'une perturbation du signal du volant moteur (variation de la période) permet la reconnaissance d'une mauvaise combustion sur un ou plusieurs temps moteurs. Cette surveillance permet de distinguer 2 types de défaut :

1) Les faibles taux de ratés de combustion entraînent un dépassement des seuils de pollution EOBD. Ils provoquent l'allumage du voyant EOBD (MIL) si la détection a lieu durant trois roulages consécutifs. Le voyant s'éteint après 40 montées en température du moteur (la faute reste inscrite dans la mémoire de l'ECU) si le problème n'est plus présent. A plus ou moins longue échéance selon l'utilisation du véhicule cela amène à une destruction du catalyseur.

2) Les forts taux de ratés de combustion entraînent la destruction très rapide du catalyseur.

Le diagnostic de la sonde à oxygène est de détecter un dysfonctionnement qui provoquerait un dépassement du seuil EOBD des émissions polluantes. Le test de cette sonde est réalisé dans certaines conditions et doit aboutir à la lecture d'un signal sinusoïdal dont l'amplitude est d'environ 0,9 V et la fréquence aux environs de 3 Hz. Le diagnostic du catalyseur permet de vérifier la capacité de stockage en oxygène du catalyseur et d'indiquer son état. Lorsque le catalyseur vieillit, sa capacité de stockage en oxygène diminue, en même temps que sa capacité à traiter les polluants.

Liste des composants du ET849

R1.....470 Ω
R2.....470 Ω
R3.....470 Ω
R4.....470 Ω
R5.....240 Ω
R6.....4,7 kΩ
R7.....4,7 kΩ
R8.....10 kΩ
R9.....10 kΩ
R10....22 kΩ
R11....10 kΩ
R12....100 kΩ
R13....10 kΩ
R14....4,7 kΩ
R15....4,7 kΩ
R16....2,2 kΩ
R17....510 Ω 1/2 W
R18....2,2 kΩ
R19....510 Ω 1/2 W
R20....47 kΩ
R21....22 kΩ

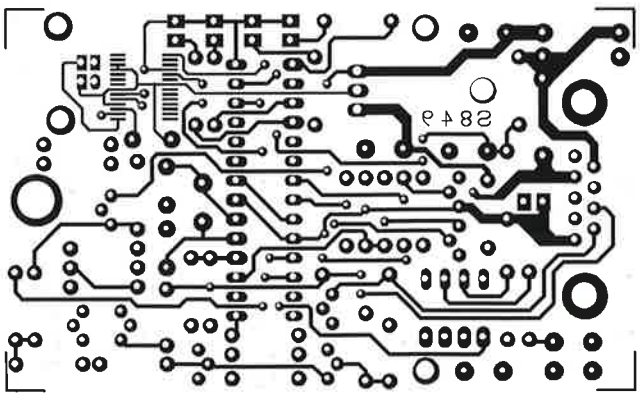


Figure B : circuit imprimé à l'échelle 1 : 1 côté soudures du testeur EOBD.

R22....4,7 kΩ
R23....10 kΩ
R24....47 kΩ
R25....10 kΩ
R26....*voir texte
R27....470 Ω
R28....470 Ω
R29....470 Ω

R30....470 Ω
R31....4,7 kΩ
R32....100 Ω
R33....100 Ω
C1.....100 nF multicouche
C2.....100 nF multicouche
C3.....27 pF céramique
C4.....27 pF céramique

Le test consiste donc à créer des variations importantes des créneaux de richesse, afin d'envoyer plus d'oxygène dans le catalyseur. Si le catalyseur est bon, il absorbera l'oxygène qu'on lui envoie et la tension de la sonde O2 aval restera à une valeur moyenne (signal quasiment plat), par contre si le catalyseur est usé, il rejettera l'oxygène qu'il ne pourra pas stocker et la sonde génèrera un signal.

Les codes d'erreurs

Les codes d'erreurs sont des codes alphanumériques qui identifient un problème rencontré par un ou plusieurs systèmes contrôlés par l'ordinateur de bord. Chaque code constitue un message qui décrit le circuit, le composant ou le système dans lequel le problème a été décelé. Les codes d'erreurs « OBD2 » sont composés de 5 caractères, soit une lettre suivie de 4 chiffres.

Le 1er caractère est une lettre qui identifie le « système principal » d'où provient le problème (carrosserie, châssis, groupe motopropulseur ou réseau).

P : Groupe motopropulseur
B : Carrosserie
C : Châssis
U : Réseau

Le 2ème caractère est un chiffre qui identifie le type de code (code générique ou propre au fabricant).

NB : les codes d'erreurs ont été standardisés pour tous les fabricants de véhicules. Les normes s'appliquant aux codes des problèmes génériques, ainsi que les définitions, sont établies conjointement par la Society of Automotive Engineers (SAE) et l'ISO. Les codes propres aux différents fabricants sont des codes qui sont édités par le fabricant du véhicule. Les fabricants peuvent dépasser les diagnostics obligatoires de l'ordinateur de bord pour faciliter encore plus le diagnostic de leurs systèmes.

Le 3ème caractère est un chiffre qui identifie le système ou le sous-système spécifique où se situe le problème.
Les 4ème et 5ème caractères sont des chiffres qui identifient quelle section du système pose problème.

Exemple pour le code **P0301**
P : Groupe motopropulseur
0 : code générique (1 pour un code constructeur)
3 : Système de contrôle de la combustion : ratés de combustion
01 : sur le cylindre n°1

Nous proposerons en téléchargement sur notre site une liste des codes de base, cela prendrait plusieurs pages de la revue et n'apporterait que peu d'intérêt.

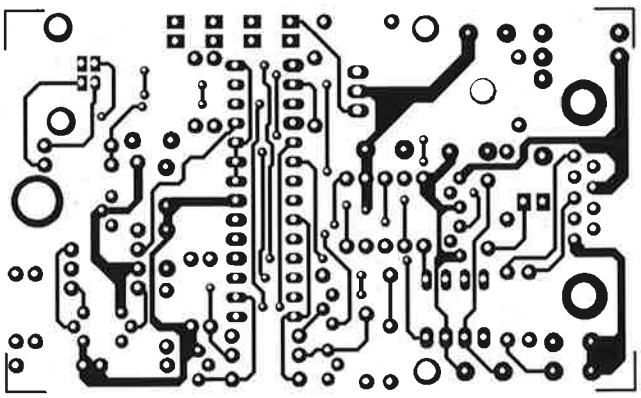


Figure C : circuit imprimé à l'échelle 1 : 1 côté composants du testeur EOBD.

C5 ..... 100 nF multicouche
C6 ..... 100 nF multicouche
C7 ..... 100 nF multicouche
C8 ..... 560 pF céramique
C9 ..... 560 pF céramique
Q1 ..... quartz 4 MHz
U1 ..... ELM327 (programmé en usine)

U2 ..... MCP2551-E/P
U3 7805
U4 ..... LM317LZ
U5 ..... FT232RL
D1 ..... 1N4007
D2 ..... 1N4148
D3 ..... 1N4148
LD1 .... LED 3 mm rouge

LD3 .... LED 3 mm rouge
LD2 .... LED 3 mm jaune
LD4 .... LED 3 mm jaune
LD5 .... LED 3 mm verte
T1 ..... 2N3904
T2 ..... 2N3906
T3 ..... 2N3904
T4 ..... 2N3906
T5 ..... 2N3904
T6 ..... 2N3904
T7 ..... 2N3904
USB .... embase USB type B
OBD2 . prise DB9 mâle pour ci

Divers

Vis 10 mm 3 MA
Ecrous 3 MA
Support ci 2 x 4 broches
Support ci 2 x 14 broches
Coffret référence SC-700
Câble M/DB9F

Réalisation pratique

À ce stade, nous devons construire notre montage et le connecter à l'ordinateur. La fabrication est assez simple, vous devez graver un circuit imprimé double face.

Pour cela vous devez d'abord télécharger sur notre site **www.electroniquemagazine.com** dans le sommaire détaillé du numéro 133 à l'onglet « Télécharger » les typons des circuits. Nous proposons aussi la version GERBER.

Une fois les circuits gravés et les trous correctement percés, commencez par souder les « vias » à l'aide de morceaux de pattes de composants afin de relier les pistes de la couche inférieure avec celles de la couche supérieure aux endroits prévus. A ce point, vous devez souder le **composant CMS FT232RL** du côté soudures (face d'en dessous).

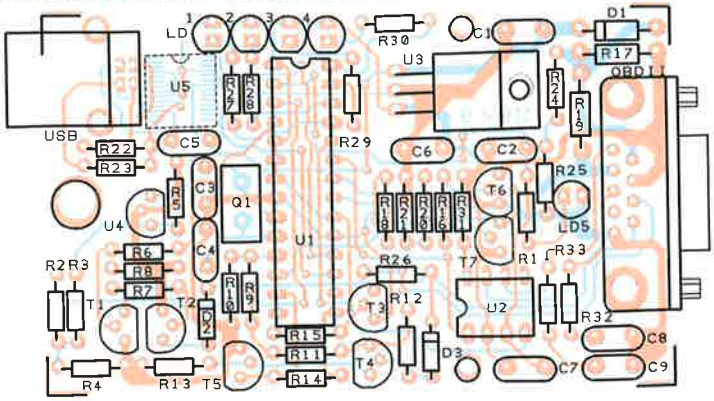


Figure D : implantation des composants du testeur EOBD.

Ce composant exige une exigence particulière, vous devez utiliser un fer à souder avec une pointe fine et de la soudure d'un diamètre de 0,5 mm.

Tout d'abord positionnez correctement le circuit en lui appliquant une pointe de colle qui le maintiendra une fois en place puis, **soudez une broche vers l'extérieur, laissez refroidir et soudez la broche en diagonale la plus éloignée par rapport à la précédente**.

Entre chaque soudure **laissez refroidir** puis continuez de proche en proche en utilisant le moins de soudure possible.

Le bus pour l'automobile

Dès l'apparition des systèmes de gestions électroniques des moteurs, les différents constructeurs automobiles ont développé des méthodes sophistiquées de diagnostics et de protocoles de communications propriétaires (propre à chaque constructeur). Il y a seulement une quinzaine d'années que s'est posée la question au niveau international de standardiser les protocoles de communications avec les différents véhicules présents sur le marché, de manière à être en mesure de réaliser des diagnostics dans les ateliers sur tous types de véhicules. De nos jours il n'existe pas encore de réelle unification, mais les protocoles utilisés par les différents constructeurs sont soumis aux spécificités « ISO ». Les différents types de bus de communication les plus communs présents aujourd'hui dans les automobiles sont :

- le **BUS CAN** (Controlled Area Network) ou « ISO 15765 » a été développé par Bosch en 1986. C'est un bus série conçu pour les réseaux internes des véhicules ; la communication s'effectue au moyen d'une paire de fils torsadés. Le bus est spécialement conçu pour offrir une haute immunité aux interférences électromagnétiques. Dans un véhicule, il peut y avoir plusieurs instances de bus CAN. Par exemple le contrôle des vitres et des sièges est normalement géré par un bus CAN à faible vitesse, tandis que la gestion du moteur et de la commande des freins nécessite un bus CAN très rapide. Le BUS CAN est également utilisé dans les automatismes industriels, dans l'aérospatiale et la domotique. La vitesse maximale prise en charge est égale à 1 Mbps pour des distances inférieures à 40 m, 125 kbps pour des distances jusqu'à 500 m, et 50 kbps pour des distances jusqu'à 1 km. Dernièrement, le BUS CAN a été adopté comme standard pour tous les véhicules (aux États-Unis c'est le cas depuis 2008) afin de simplifier les outils de diagnostics Les connexions du BUS CAN sur le connecteur « OBD2 » sont les suivantes :

  broche 6 = CAN High ;
  broche 14 = CAN Lox ;
  broche 4 = masse du véhicule ;
  broche 16 = positif de la batterie du véhicule.

- le bus « **LIN** » (Local Interconnect Network ou Réseau Internet Local) est un bus système série utilisé dans les véhicules automobiles récents. Ce protocole, développé par le consortium « LIN », est un bus série conçu pour la communication entre les capteurs intelligents et les dispositifs actionneurs (par exemple l'amortissement piloté) dans un véhicule. La première version a été réalisée en 1999. Les membres du consortium sont les fabricants suivants : BMW, DaimlerChrysler, Audi, Volvo, Motorola, Volkswagen. Une des caractéristiques importantes de ce protocole est que sa mise en œuvre ne nécessite sur le plan physique qu'un seul fil alimenté en 12 V (le châssis du véhicule est utilisé comme signal de retour). Les systèmes d'un véhicule particulier exploitant le bus « LIN » sont les suivants : climatiseurs, sièges, portières, toit ouvrant, essuie-glaces intelligents, capteurs de pluie et de température. Pour ces types d'applications il n'est pas nécessaire d'avoir une vitesse de communication élevée. La vitesse maximale de transmission sur un bus « LIN » est

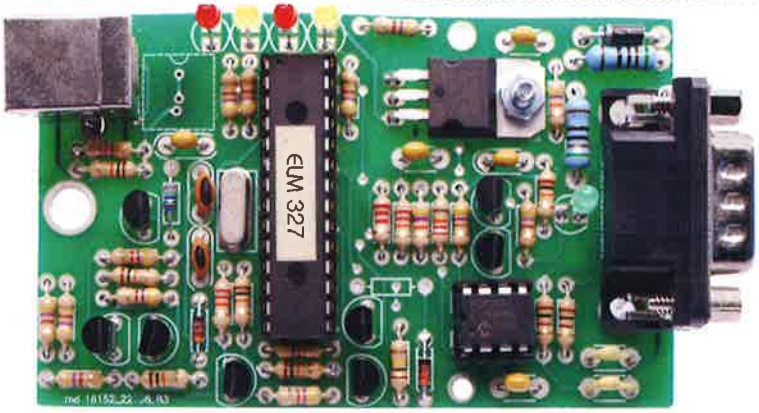


Figure E : photo de l'un de nos prototypes du testeur EOBD.

Une fois le CMS soudé, montez les composants en commençant par les résistances et les diodes, ensuite les supports des circuits intégrés et les condensateurs non polarisés et polarisés (attention au sens). Continuez avec les transistors, les LED, le connecteur USB ainsi que le régulateur en le vissant à 90° sur le circuit. Pour cela aidez-vous des figures B, C, D et E.

Il ne vous reste plus qu'à fabriquer le câble : EOBD/DB9 en reliant les contacts correspondants. Le plan de câblage du brochage du câble est donné dans l'encadré intitulé « Construisons le câble EOBD ». Notez que vous pouvez acquérir le câble déjà monté.

Vérifiez encore une fois qu'il n'y a pas d'erreurs dans l'implantation des composants et dans le brochage du câble.

Complétez l'assemblage en insérant les circuits intégrés dans leurs supports en respectant l'orientation. Avant de connecter l'interface à un véhicule,

égale à 19200 bps avec une longueur de câble allant jusqu'à 40 mètres, mais il prend également en charge les vitesses suivantes : 2400 bps et 9600 bps. D'un point de vue électrique, le bus « LIN » est basé sur une interface de type UART à 8 bits, et prend en charge une architecture de type « Maître » unique/ « Esclaves » multiples (Single MASTER/Multi SLAVE). Avec cette architecture, l'ajout ou la suppression d'un nœud n'a des conséquences que sur le nœud « MASTER » (Maître) ; les nœuds « SLAVES » (Esclaves) ne sont pas affectés. Une autre caractéristique du bus « LIN » est que le nœud « MASTER » est capable de synchroniser tous les nœuds « SLAVES », par conséquent il est inutile d'utiliser un oscillateur ou une horloge externe. Pour toutes ces raisons, le bus « LIN » peut être considéré comme une solution à faible coût.

- le bus « **SAE J1850** » : il a été développé en 1994, et est principalement utilisé pour des applications de diagnostics et de partages de données dans les véhicules lorsqu'ils roulent ou lorsqu'ils sont en maintenance dans un atelier. Il existe deux types de bus « SAE J1850 » : un basé sur une ligne de données différentielle (2 fils) capables d'atteindre une vitesse de 41,6 kbps avec une modulation de type « PWM » (modulation en largeur d'impulsion) ; l'autre basé sur l'utilisation d'un seul fil pouvant atteindre une vitesse de transmission de 10,4 kbps avec un système de transmission de type « VPW » (largeur d'impulsion variable). Les connexions de la version « PWM » sont : broche 2 = Bus+ et broche 10 = Bus–. Dans ce bus le niveau logique haut correspond à une tension de « + 5 V » et les messages de diagnostics ont une longueur de 12 octets (bytes). En ce qui concerne la version à un seul fil (« VPW »), l'unique connexion de données est la broche 2 (Bus +). Le niveau logique haut correspond à 7 V et le seuil de basculement entre les niveaux est de 3,5 V.

- le bus « **ISO 9141-2 K-Line** » adopté par Chrysler et les constructeurs Européens et Asiatiques et le bus « ISO 14230-4 » dont la nouvelle version est l'« ISO9141-2 » sont des bus de communication à 2 fils, utilisés principalement dans les véhicules produits jusqu'en 2002, et dont les 2 lignes sont « K » et « L ». La vitesse de communication est de l'ordre de 50 kbps. L'« ISO9141-2 » est de type asynchrone et sa vitesse de transmission est de 10,4 kbps. Cela correspond en termes de vitesse de communication à une connexion RS232, « les signaux transitent dans un seul sens ». La broche active est la broche 7 (K-Line), la broche 15 (L-ligne) est facultative. La 1ère, lorsqu'elle est au repos, est à un niveau haut (positif de la batterie). Les messages de diagnostics ont une longueur de 12 octets.

Quant à l'« **ISO14230** », également connu sous le nom **KWP2000** (Keyword Protocol 2000), les connexions sont identiques. La vitesse de communication varie entre 1,2 kbps et 10,4 kbps. Les messages peuvent contenir jusqu'à 255 octets.

En dehors de cela, un bus d'une nouvelle conception a été développé, il se nomme « FlexRay ». Il a été créé en 1999 par un consortium de constructeurs automobiles qui est dissous aujourd'hui. Parmi ses caractéristiques intéressantes, il dispose de 2 canaux de communication, chacun capable d'atteindre une vitesse de 10 Mbps (les canaux peuvent être utilisés ensemble de façon à introduire un mécanisme de redondance, ce qui rend le bus « FlexRay » extrêmement rapide et fiable, ou de manière indépendante l'un par rapport à l'autre ce qui permet d'obtenir une vitesse maximale de 20 Mbps). Sa tolérance aux pannes et l'existence d'un mécanisme de synchronisation entre les différents nœuds du réseau en font un bus très fiable.

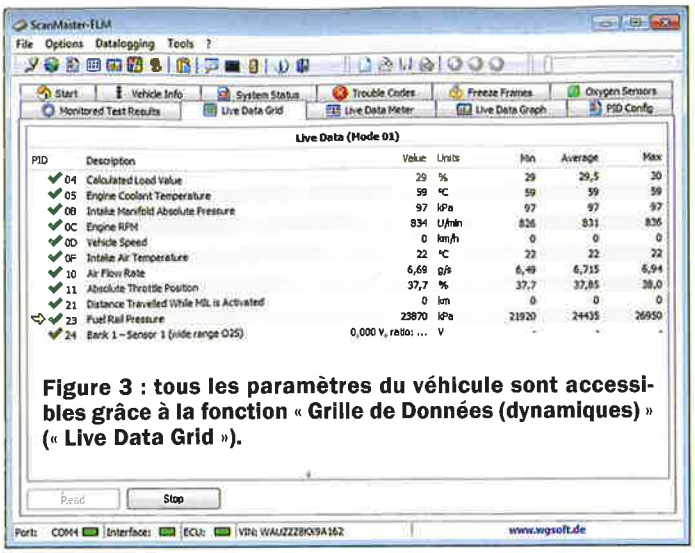


Figure 3 : tous les paramètres du véhicule sont accessibles grâce à la fonction « Grille de Données (dynamiques) » (« Live Data Grid »).

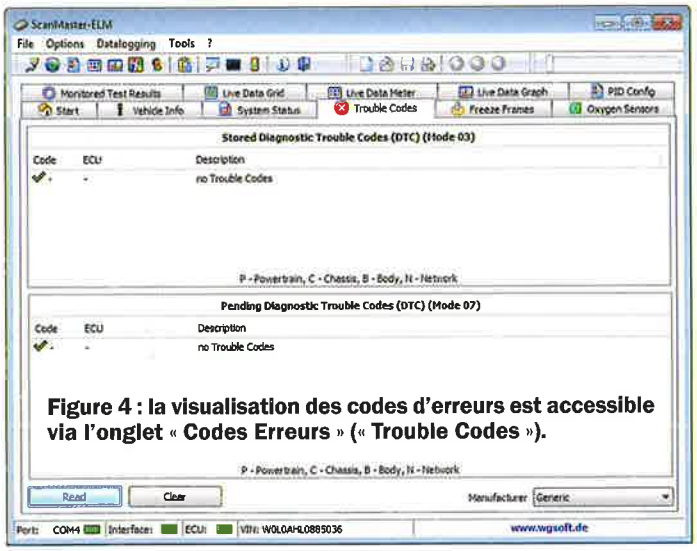


Figure 4 : la visualisation des codes d'erreurs est accessible via l'onglet « Codes Erreurs » (« Trouble Codes »).

branchez le connecteur USB à un PC et vérifiez que le logiciel « ScanMaster-ELM-DEMO » détecte bien l'interface.

Cherchons la prise « EOBD » dans le véhicule

Avant de pouvoir utiliser l'interface avec un type de véhicule donné, vous devez vérifier où le connecteur « EOBD » du véhicule se situe. En règle générale il se trouve dans l'habitacle, à proximité des pédales ou de la colonne de direction, masqué par un cache.

Par exemple dans les Ford Mondeo et Fiesta 2 il se situe en bas à gauche, dans une Mercedes Classe C il se trouve à côté de la colonne de direction juste au-dessus des pédales.

Dans une Audi A3 ou A4 il se situe de manière identique à une classe C mais caché par un couvercle.

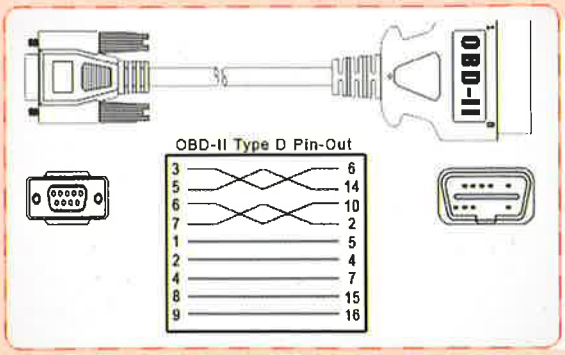
Dans une Volkswagen Golf, la prise diagnostique se situe à gauche du volant près de la poignée d'ouverture du capot moteur.

Une caractéristique particulière à l'Opel Astra produite de 2005 à aujourd'hui, est que le connecteur « EOBD » est placé près du levier de vitesses, dans un logement approprié situé dans le tunnel central. Quel que soit le véhicule testé, nous vous recommandons de connecter l'interface au connecteur « EOBD » de la voiture et le port USB à l'ordinateur, ensuite vous tournez la clé de contact.

Les typons des circuits imprimés, les fichiers GERBER, le logiciel Scan Master Demo et la liste des codes d'ereurs sont téléchargeables gratuitement sur notre site :
www.electroniquemagazine.com dans le sommaire détaillé de la revue numéro **133** section « **Télécharger** ».

Construisons le câble EOBD

Notre interface est équipée d'un connecteur femelle de type DB9 d'un côté et d'un connecteur « EOBD » mâle à l'autre extrémité. Si vous voulez, vous pouvez préparer votre câble en respectant les connexions représentées sur le schéma ci-dessus.



ET 771

Reconnaissance des couleurs avec processeur vocal

Ce montage est capable de reconnaître la couleur d'un objet au moyen d'un convertisseur couleur/fréquence directement géré par un microcontrôleur. Après l'acquisition de la couleur, le microcontrôleur effectue un traitement et programme un synthétiseur vocal pour annoncer la couleur détectée.

de Fabio Riscica

Ce que nous allons vous décrire dans ces pages est quelque chose de vraiment spécial, que nous n'avons jamais publié auparavant. Il s'agit d'un appareil qui peut être utilisé pour offrir une plus grande autonomie aux personnes malvoyantes, notamment à celles ayant des difficultés à percevoir les différentes couleurs, c'est-à-dire les daltoniens. Le daltonisme est une anomalie de la rétine oculaire qui provoque une déficience de la perception des couleurs.

Nous allons donc vous présenter dans cet article un **montage vocal de reconnaissance des couleurs**. Il est doté d'un haut-parleur, et il peut être utilisé à des fins didactiques mais aussi pour rendre des services dans la vie quotidienne.

En principe, ce **type de dispositif est très utile pour le renforcement de l'autonomie des personnes ayant des déficiences visuelles**.

Les modèles disponibles sur le marché ont des dimensions similaires à celles d'un GSM et permettent de connaitre la couleur d'un objet (vêtement, meuble, etc.) grâce à un « œil artificiel » capable de détecter la totalité du spectre des couleurs visibles. L'utilisation est généralement très simple. Il suffit d'appliquer le capteur de l'instrument sur l'objet dont nous voulons connaître la couleur, ensuite il faut appuyer sur un bouton pour détecter la couleur et lancer la synthèse vocale.

Pour une personne ayant une vue très dégradée, être en mesure de reconnaître la couleur d'un objet est certainement utile. Par exemple, connaitre la couleur d'un vêtement permet à la personne de les choisir elle-même. Cela peut aussi aider pour le stockage de produits alimentaires dans des récipients de différentes couleurs.

Malheureusement, le coût de ces appareils commerciaux est assez élevé, à cause de la technologie nécessaire, mais aussi pour des raisons commerciales. Par conséquent, ces dispositifs se vendent difficilement et sont rarement attribués aux personnes concernées.

Notre reconnaissance vocale des couleurs n'est pas la plus performante en termes de nombre de couleurs détectées et de nuances distinguées par rapport aux appareils disponibles dans le commerce.

Cependant, elle fonctionne selon les mêmes principes utilisés par les produits commerciaux et représente une excellente application didactique pour la compréhension du fonctionnement théorique de ces dispositifs, tout en assurant la possibilité d'une utilisation pratique.

Le cœur de ce montage est le **convertisseur de couleur/fréquence** « **TCS3200-DB** » fabriqué par la firme « **Parallax** » (www.parallax.com), qui **génère une onde carrée de fréquence proportionnelle à l'intensité lumineuse des 3 composantes RVB des couleurs détectées**.

Un microcontrôleur acquiert et traite le signal en fréquence, afin qu'il puisse être reconnu et classé selon les critères humains.

Ensuite, il pilote un circuit à **synthèse vocale** « **SpeakJet** » qui, relié à un amplificateur et un haut-parleur, reproduit la **séquence des allophones** (en phonétique un allophone ou variante est une réalisation phonique d'un phonème, en particulier un son) qui compose la phrase relative de la couleur détectée (en **Anglais**). Voyons maintenant comment concevoir ce dispositif.

Schéma de principe de la reconnaissance vocale des couleurs

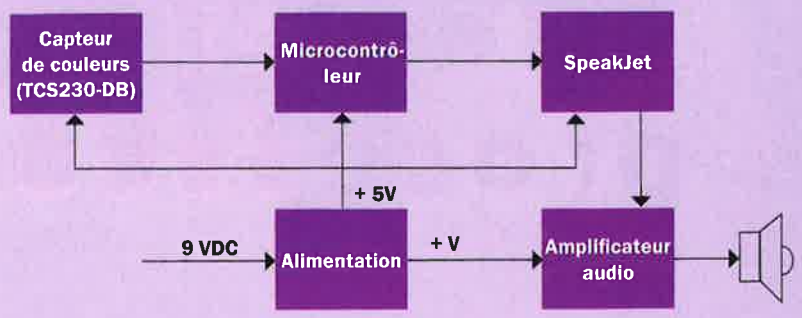


Figure 1 : Le microcontrôleur communique avec le capteur de couleurs et pilote le synthétiseur vocal qui prononce la couleur détectée en Anglais. Le bloc d'alimentation fournit 5 VDC pour les circuits logiques, et une tension légèrement inférieure à 9 VDC pour l'amplificateur audio.

Le schéma de principe

Le schéma de principe est représenté en figure 1. Comme vous pouvez le voir, la reconnaissance des couleurs peut être divisée en **5 blocs**, dont le **1er est le capteur de couleurs**, à savoir un circuit électronique basé sur la conversion de la couleur en une fréquence.

Il s'agit du circuit « **TCS3200-DB** » auquel nous consacrerons un paragraphe. Ce capteur communique de manière bidirectionnelle avec le microcontrôleur. Dans la pratique, le « **TCS3200-DB** » envoie les **informations de la couleur** « **vue** » en fonction des requêtes du microcontrôleur. Celui-ci, après avoir traité l'information reçue par le capteur de couleurs, génère une séquence de commandes qu'il envoie au **3ème bloc**, c'est-à-dire au synthétiseur vocal « **SpeakJet** » que nous étudierons dans un autre paragraphe.

Le **synthétiseur annonce oralement via le haut-parleur la couleur détectée par le capteur**. La sortie du « **SpeakJet** » n'est pas assez puissante pour piloter directement le haut-parleur, nous devons donc l'amplifier à l'aide d'un **amplificateur audio** qui alimente un HP d'une puissance de 1 W / 8 Ω. Cet étage est le **4ème bloc**.

Il ne reste plus que le **5ème bloc** qui est l'**étage d'alimentation**.

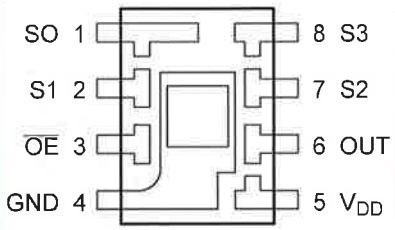


Figure 2 : brochage du circuit de reconnaissance des couleurs TCS3200-DB.

Il est alimenté par une tension extérne de **9 VDC** et fournit 2 tensions « **V+** » et « **5 V** » nécessaires pour alimenter le montage. La tension « **V+** » est réservée à l'amplificateur audio et la tension « **5 V** » alimente le reste du montage.

Le convertisseur couleur/fréquence TCS3200-DB

Ce capteur intègre un système de **photodiodes et un convertisseur couleur/fréquence dans un seul circuit intégré CMOS** selon le brochage représenté en figure 2. La tension d'alimentation peut être comprise entre 2,7 VDC et 5,5 VDC.

L'échelle de la fréquence de sortie peut être paramétrée à l'aide de niveaux logiques appliqués sur les entrées de contrôle S0 et S1. Si ces entrées sont toutes les deux à un

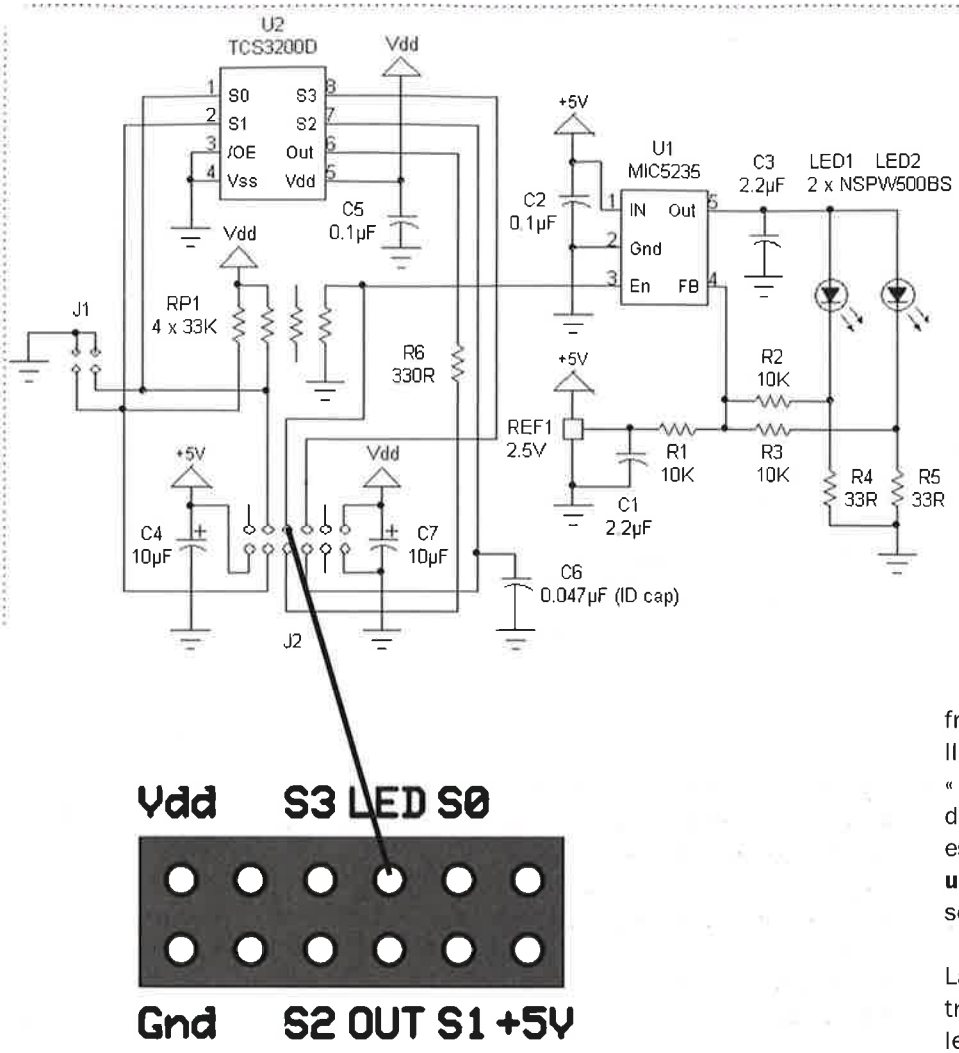


Figure 3 : schéma électrique interne du module TCS3200-DB. Le brochage en bas représente le connecteur situé à l'arrière du module. Attention le sens des broches est inversé par rapport au schéma. Le trait indique la broche « LED » sur le schéma.

niveau logique bas, le circuit est en veille (« POWER DOWN »). Si S0 et S1 sont toutes les deux à un niveau logique haut, la fréquence de sortie correspond à l'échelle complète (100 %). Si S0 est à un niveau logique haut et S1 à un niveau logique bas, la fréquence de sortie correspond à 20 % de l'échelle complète. Enfin, si S0 est à un niveau logique bas et S1 à un niveau logique haut, la fréquence de sortie correspond à 2 % de l'échelle complète.

Les entrées et la sortie numériques peuvent être directement interfacées à un microcontrôleur ou un autre circuit. La broche « **OUTPUT ENABLE** » (/OE) met dans un état de haute impédance la sortie « **fréquence** » (OUT), une condition nécessaire pour piloter en parallèle plusieurs microcontrôleurs.

Le **convertisseur couleur/fréquence** comprend une matrice de **8 x 8 photodiodes**. **16 photodiodes** ont des **filtres bleus**, **16** ont des **filtres verts**, **16** ont des **filtres rouges** et les **16** dernières n'ont **aucun filtre**. Les **4 groupes de photodiodes sont entrecoupés afin de minimiser l'effet de non-uniformité du rayonnement incident** (en d'autres termes, cela augmente l'efficacité de la reconnaissance de la couleur).

Les **16 photodiodes** de chaque **groupe de même couleur** sont connectées en **parallèle**, et **chaque groupe** de photodiode peut être **sélectionné** grâce aux entrées **S2** et **S3**. Si S2 et S3 sont à un niveau logique bas, le groupe de photodiodes rouges est sélectionné, si S2 est à un niveau logique bas et S3 à un niveau logique haut, cela correspond au groupe bleu. Si S2 est à un niveau logique haut et S3 à un niveau logique bas, le groupe sélectionné est le groupe sans filtre. Si enfin S2 et S3 sont à un niveau logique haut, c'est le groupe vert qui est sélectionné.

Le module est visible en figure 4 et contient le circuit convertisseur couleur/fréquence ainsi que les photodiodes. Il est disponible auprès de la société « **Parallax** » au prix de 55 € et le lien direct pour le module « **TCS3200-DB** » est : **https://www.parallax.com/product/28302** (au moment où ces pages sont écrites).

La figure 3 représente le schéma électrique interne du module qui intègre les **2 LED à haute luminosité utilisées comme sources lumineuse**. Si la broche « **LED** » du module est mise à un niveau bas, elle **active les 2 LED**. La sortie « **fréquence** » (OUT) est **activée** si la broche « **OE** » se trouve à un **niveau bas**.

Dans la pratique, le fonctionnement du capteur peut être résumé de la manière suivante : après avoir sélectionné l'échelle appropriée de la fréquence

Figure 4 : ici le module hybride TCS3200-DB utilisé dans ce projet de reconnaissance vocale des couleurs.

de sortie à l'aide des broches S0 et S1, il suffit d'activer les 2 LED haute luminosité afin que les photodiodes des groupes rouge, vert ou bleu acquièrent le rayonnement lumineux réfléchi par l'objet « observé ».

Il en résulte un signal de sortie dont la fréquence est directement proportionnelle à l'intensité des couleurs rouge, verte ou bleue de l'objet. Si vous activez les photodiodes sans filtre, le capteur acquiert la totalité du spectre du rayonnement réfléchi et vous pouvez réaliser par exemple le calibrage de la couleur blanche.

Classification des couleurs

Déterminer une couleur en fonction de la fréquence qu'elle émet peut paraître facile, mais décrire une couleur à une personne qui ne la visualise pas correctement s'avère être un peu plus compliqué. Le système de détection des couleurs du capteur **TCS3200-DB** est de type « **RVB** » et ne correspond pas directement à une classification de type perceptive basée sur le système « **HSL** » (Hue, Saturation, Lightness ou Teinte, Saturation, Luminosité).

La classification des couleurs est basée sur la relation entre les émotions humaines et les couleurs, d'après **Johannes Itten** (1888 - 1967 peintre et un enseignant Suisse).

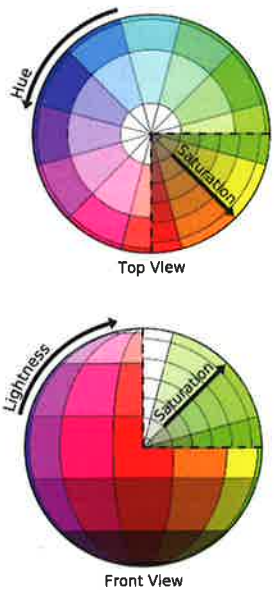


Figure 5 : la sphère des couleurs.

Il enseignait la couleur et la forme, et créa des variantes du cercle chromatique en utilisant le système « **HSL** ».

Il sélectionna 12 teintes fondamentales, imposa 5 niveaux de luminance et 3 niveaux de saturation pour chaque couleur. Il obtint un total de 180 couleurs de base, représentées sur la surface d'une sphère (voir la figure 5).

Les 12 couleurs pures sont placées le long du cercle équatorial, la luminance varie le long des méridiens, et la saturation augmente lorsque le rayon croît.

Les couleurs opposées les unes aux autres sont situées à des positions diamétralement opposées. Cette disposition permet une identification objective des particularités des couleurs, en identifiant 5 types différents de contrastes : le contraste de la teinte, le contraste clair/foncé, le contraste chaud/froid, le contraste complémentaire et le contraste de saturation.

Les contrastes mentionnés ci-avant peuvent être utilisés pour définir une représentation du contenu d'une image. Cette représentation, cependant, est fortement conditionnée par la culture et les expériences individuelles.

Mais d'abord nous devons donner quelques définitions pour bien situer le problème de détection des couleurs selon la technique « **HSL** ».

La **classification des couleurs** est en fait la **colorimétrie** ou **mesure des couleurs**. Une **couleur** est représentée dans un espace à **3 dimensions** selon 3 critères qui sont : la **teinte**, la **saturation** et la **luminance**.

- La **teinte** d'une couleur est le nom que l'on donne à la famille dans laquelle elle s'insère, avec plus ou moins de précision et de nuances. Nous pouvons cependant classer les teintes en deux catégories :

  - les couleurs spectrales que nous pouvons observer dans un arc-en-ciel c'est-à-dire les couleurs rouge, orange, jaune, vert, cyan, bleu et violet ainsi que toutes les nuances intermédiaires ;

  - les couleurs non-spectrales issues d'un mélange bleu-rouge ou violet-rouge c'est-à-dire les nuances de pourpres et de magentas.

- La **saturation** est plus fréquemment appelée **pureté** dans le domaine de la peinture, elle décrit la vivacité d'une couleur. Dans le cas des couleurs spectrales, ce terme est tout à fait adapté puisque une couleur saturée à 100 % correspond à une couleur pure, c'est-à-dire une onde monochromatique, autrement dit, une onde parfaitement sinusoïdale (de longueur d'onde unique). On parle de façon analogue d'un son pur en acoustique. Une couleur est vive lorsque sa saturation est élevée. En revanche, une couleur pale ou terne est peu saturée. Le blanc et toutes les nuances de gris ont une saturation nulle. Dans le système « **HSL** » que nous utilisons en informatique, les couleurs sont affectées d'une saturation comprise entre 0 et 255. Cependant une couleur ayant une saturation de 255 n'est pas pure, car les couleurs primaires utilisées ne sont pas pures. Les couleurs saturées à 100 % ne se rencontrent ni dans la nature ni dans l'environnement quotidien.

- La **luminance** est une grandeur photométrique qui mesure l'**éclat d'une source lumineuse** étendue, c'est-à-dire d'une surface. Elle décrit donc la luminosité, la clarté, la luminance relative ou encore le

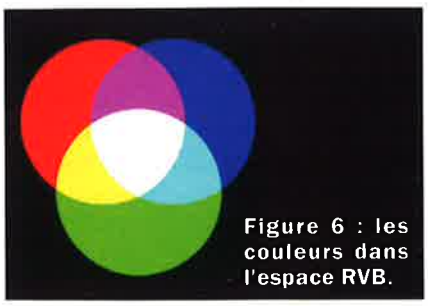
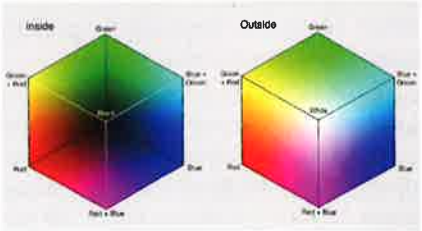


Figure 6 : les couleurs dans l'espace RVB.

Figure 7 : le cube RVB.



facteur de luminance dont les définitions diffèrent légèrement. Dans tous les cas, une couleur qui a une luminance élevée sera vive si sa saturation est élevée, et claire si sa saturation est faible. Une couleur qui a une luminance faible est une couleur sombre. Dans le système « **HSL** » la luminosité prend la valeur « 0 » pour le noir et jusqu'à « 255 » pour le blanc. En colorimétrie, on utilise souvent le facteur de luminance allant de 0 pour le noir jusqu'à 100 % pour le blanc.

Le modèle « **RVB** » est un système de **codage informatique des couleurs basé sur les 3 couleurs primaires** :

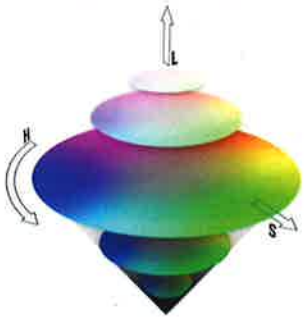


Figure 8 : les couleurs selon le cône HSL.

Rouge (Red), **Vert** (Green) et le **Bleu** (Blue). Ainsi une couleur peut être reconstituée par une synthèse additive de ces 3 couleurs primaires.

Toute image peut être décomposée, à travers des filtres, **dans ces trois couleurs de base**. Celles-ci, lorsqu'elles sont mélangées, peuvent couvrir l'ensemble du spectre visible.

Le modèle « **RVB** » est un **modèle additif** dans lequel la combinaison des 3 couleurs avec leur intensité maximale crée la couleur blanche (toute la lumière est réfléchie).

La **combinaison de 2 couleurs** donne le **cyan**, le **magenta** et le **jaune** (voir la figure 6).

L'espace « **RVB** » peut être représenté par un **cube** (voir la figure 7), dont les bords sont rouge, jaune, vert, bleu, cyan, magenta, noir et blanc. Chacune des 3 valeurs « **RVB** » est comprise entre **0** et **255**.

Cependant l'échelle « **RVB** » ne permet pas de classer la couleur selon un modèle de perception (ce qui est le cas pour le modèle « HSL ») car il est orienté matériel.

Le modèle « **HSL** » utilise un espace de couleur ayant une déformation non linéaire du cube des couleurs du système « **RVB** ». Il comprend 3 coordonnées : la **teinte** (Hue), la **saturation** (Saturation) et la **luminosité** (Lightness).

La teinte définit la composante chromatique, c'est-à-dire la fréquence de l'onde lumineuse, et représente un angle qui peut varier de **0°** à **360°**. Typiquement un angle de **0°** correspond à la couleur **rouge**, **60°** à la **jaune**, **120°** à la **verte**, **180°** à la **bleue**, **240°** à la couleur **magenta**. Les autres couleurs sont réparties linéairement entre ces angles.

La **saturation définit la façon dont la couleur est proche du gris** : **0** indique la couleur **grise**, tandis que la valeur **1** représente la couleur **primaire pure**. L'échelle est divisée en **100 intervalles** (valeurs), c'est une façon de l'exprimer en pourcentage.

La **luminosité représente la lumière moyenne de la couleur**. Comme pour la saturation, la valeur est comprise entre 0 et 1 avec une échelle de 100 valeurs. La diminution de la luminosité réduit la valeur des couleurs primaires, mais en les maintenant dans une même proportion.

L'espace « **HSL** » est représenté graphiquement par un double cône (voir la figure 8) dans lequel les 2 points extrêmes correspondent aux couleurs noire et blanche. Le **paramètre angulaire** correspond à la **teinte**, la **saturation** est

Listing 1

```
#define MAX(a,b) (((a)>(b)) ? (a):(b))
#define MIN(a,b) (((a)<(b)) ? (a):(b))

// Convertit le format RVB
// dans la plage de 0 à 1 au format
// HSL, toujours dans la plage de 0 à 1.

void ToHSL(const float red, const float green, const float blue, float& hue, float&
saturation, float& luminance)
 {
  float fmax, fmin;
  fmax=MAX(MAX(red, green), blue);
  fmin=MIN(MIN(red, green), blue);
  luminance = fmax;
  if (fmax > 0)
    saturation = (fmax-fmin)/fmax;
  else
    saturation = 0;
  if (saturation == 0)
    hue = 0;
  else
    {
      if (fmax == red)
        hue = (green - blue) /
        (fmax - fmin);
      else if (fmax == green)
        hue = 2 + (blue - red) /
        (fmax - fmin);
      else
        hue = 4 + (red - green) /
        (fmax - fmin);
      hue = hue / 6;
      if (hue < 0) hue += 1;
    }
 }
```

Tableau 1 - Quelques phonèmes du circuit SpeakJet

Voyelles			Consonne		
Longue A - "Gate" & "Ate"	\EYIY		V - "Vest" & "Even"	\VV	
Longue E - "See" & "Even"	\IY		Z - "Zoo" & "Zap"	\ZZ	
Longue I - "Sky" & "Five"	\OHIH	voir aussi IE	ZH - "Azure" & "Treasure"	\ZH	
Longue O - "Comb" & "Over"	\OW	voir aussi OA	TH - "There" & "That"	\DH	

Voyelles avec R			Consonne fricative		
R - "Ray" & "Brain"	\RR		H - "Help" & "Hand"	\HE	en avant
AIR - "Hair" & "Stair"	\EYRR		H - "Hoe" & "Hot"	\HO	en arrière
AR - "Part" & "Farm"	\AWRR	voir aussi IE	WH - "Who" & "Whale"	\WH	
EAR - "Clear" & "Hear"	\IYRR	voir aussi OA	F - "Food" & "Effort"	\FF	

NB : les exemples sont en Anglais car le circuit SpeakJet prononce en Anglais.

Tableau 2 - Exemples de commandes MSA

Dec.	SpeakJet
000	Pause 0
001	Pause 1
002	Pause 2
003	Pause 3
004	Pause 4
005	Pause 5
006	Pause 6
007	Son suivant rapide
008	Prononce le son suivant lentement
014	Prononce le son suivant avec un ton haut
015	Prononce le son suivant avec un ton bas
016	Attend
020	Volume, X

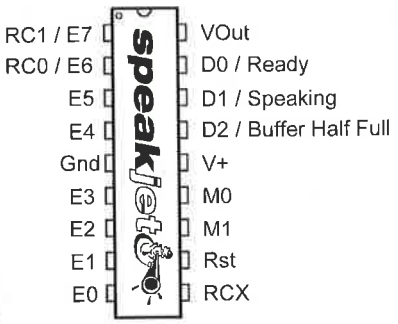


Figure 9 : brochage du synthétiseur vocal « SpeakJet ».

Tableau 3 - Liste partielle des allophones

Code	Phonème	Exemples de mots	Msec.	Type de phonème
128	IY	See, Even, Feed	70	Voyelle longue
129	IH	Sit, Fix, Pin	70	Voyelle courte
130	EY	Hair, Gate, Beige	70	Voyelle longue
131	EH	Met, Check, Red	70	Voyelle courte
132	AY	Hat, Fast, Fan	70	Voyelle courte
133	AX	Cotten	70	Voyelle courte
134	UX	Luck, Up, Uncle	70	Voyelle courte
135	OH	Hot, Clock, Fox	70	Voyelle courte
136	AW	Father, Fall	70	Voyelle courte
137	OW	Comb, Over, Hold	70	Voyelle longue
138	UH	Book, Could, Should	70	Voyelle courte
139	UW	Food, June	70	Voyelle longue
140	MM	Milk, Famous,	70	Voyelle nasale

NB : les exemples de mots sont en Anglais car le circuit SpeakJet prononce en Anglais.

égale à la **distance par rapport à l'axe horizontal** et la **luminosité** correspond à la **distance le long de l'axe noir/blanc** (vertical).

Si l'espace des couleurs est représenté par des disques de luminosité différente, alors la teinte et la saturation sont équivalentes aux coordonnées polaires d'un point quelconque du plan. En utilisant l'espace des couleurs « **HSL** », il est possible d'effectuer une classification identique en terme de

perception en introduisant des valeurs de teinte pour définir la couleur, des valeurs de saturation pour définir si la couleur est « vive » ou « pâle », des valeurs de luminosité pour définir l'aspect clair ou foncé de la couleur.

Par exemple, la couleur rouge est caractérisée par une teinte comprise entre 0° et 12°, par une saturation supérieure à 0,5 et une luminosité comprise entre 0,6 et 0,8. Pour être en mesure de réaliser une telle classification, il est nécessaire de convertir les niveaux « **RVB** » acquis dans l'espace « **HSL** ».

Cette conversion peut être effectuée à l'aide de plusieurs algorithmes. Dans le Listing 1 nous montrons un de ces algorithmes en langage C que nous avons inséré dans notre programme.

Le circuit « SpeakJet »

Après avoir étudié le capteur de couleur et les principes de classification des couleurs, étudions maintenant le 2ème élément indispensable de notre projet, à savoir le **synthétiseur vocal**.

Le circuit intégré « **SpeakJet** » (www.magnevation.com/SpeakJet.swf) est un synthétiseur vocal et de sons complexes contenu dans une seule puce.

Il utilise la technologie « **MSA** » (Mathematical Sound Architecture ou Architecture Mathématique des Sons) pour contrôler ses 5 canaux internes utilisés pour générer un vocabulaire virtuellement illimité de synthèse vocale et de sons complexes.

Le « **SpeakJet** », dont le brochage est visible en figure 9, est préconfiguré avec **72 éléments vocaux** ou « **allophones** », **43 effets sonores** et **12 fréquences** (tonalités) **DTMF**.

Grâce à la sélection de ces composants « MSA » en combinaison avec d'autres paramètres, tels que le contrôle du « pitch » (variation de tonalité), du débit (« rate »), de l'infléchissement (variation forcée de la fréquence) et du volume, le circuit peut reproduire des phrases au vocabulaire illimité ainsi que des effets sonores avec des centaines de variations possibles.

Il ne s'agit pas de formes d'ondes enregistrées ou de fragments de sons, mais un son réel synthétisé à partir de formules mathématiques.

Le circuit « **SpeakJet** » peut être contrôlé par l'intermédiaire de signaux logiques appliqués sur une seule de ses 8 lignes d'entrées, soit par des données transmises à partir d'un CPU, soit par un microcontrôleur fonctionnant de manière autonome.

Les autres caractéristiques du circuit sont la présence d'un « **buffer** » (étage tampon) d'entrée de **64 octets**, d'une **EEPROM** entièrement utilisable, de **3 sorties programmables** et la possibilité d'accéder directement aux 5 canaux internes de la synthèse sonore. L'alimentation du circuit est comprise entre 2 VDC et 5,5 VDC.

La broche « **RCX** » est une entrée série pour des périphériques externes communiquant avec le « **SpeakJet** ». Le niveau du signal est **compatible TTL** et la configuration série par défaut est : **9600 bauds**, **8 bits de données**, **pas de parité**, **1 bit d'arrêt**.

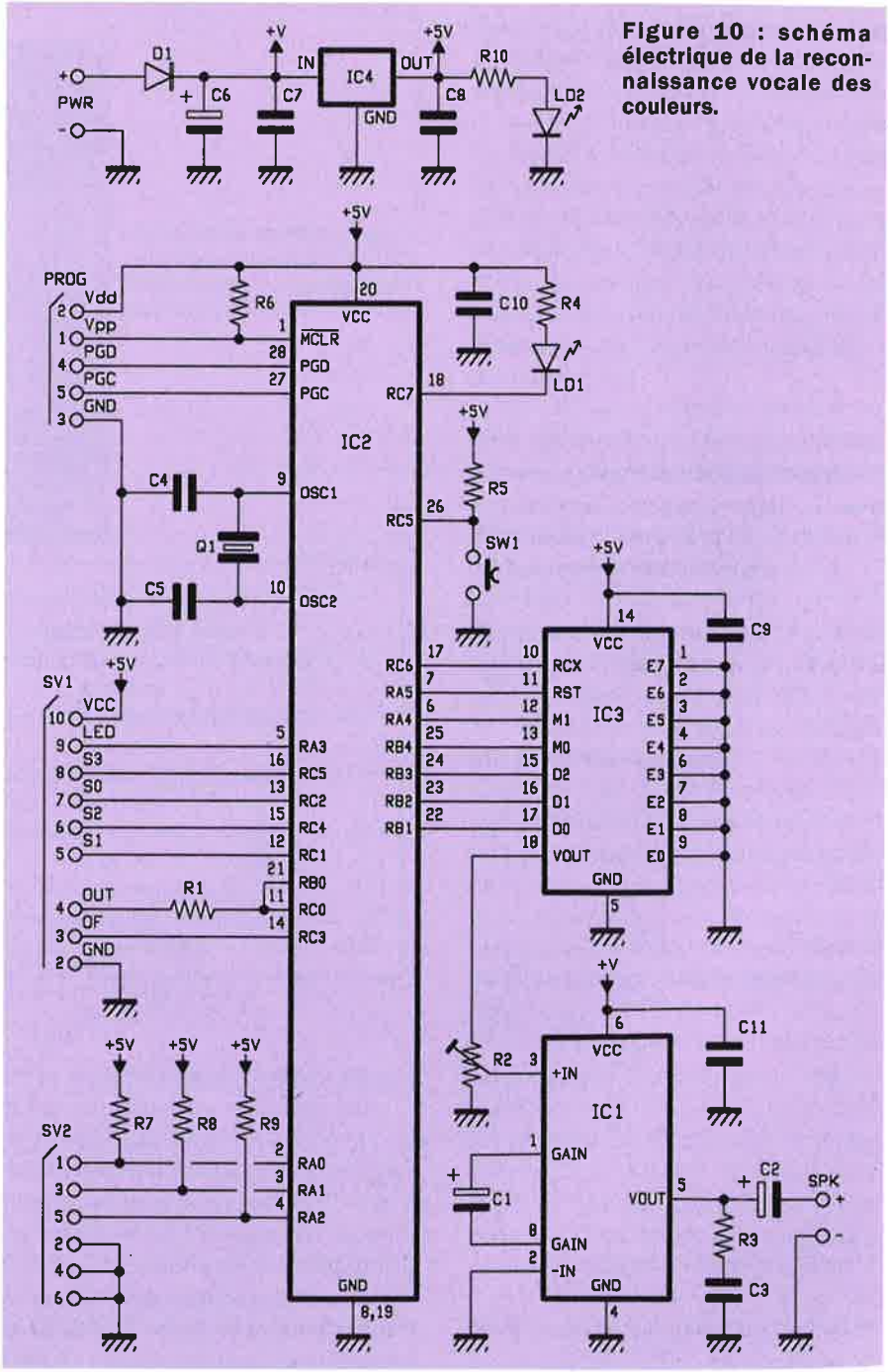


Figure 10 : schéma électrique de la reconnaissance vocale des couleurs.

Il est possible de modifier la vitesse de transmission à tout moment en basculant le circuit dans le mode de configuration (le réglage s'effectue après la réinitialisation de la broche M1 à niveau logique bas).

La broche **M0**, s'active après la réinitialisation, et permet de sélectionner le mode de démonstration du circuit.

En mode de fonctionnement normal la broche **M0** doit être placée à un **niveau logique bas** et la broche **M1 à un niveau logique haut** à l'aide d'une résistance de pull-up.

Les 3 broches de sortie **D0**, **D1** et **D2** indiquent respectivement que le « **SpeakJet** » est prêt à accepter des commandes, qu'il reproduit un son et que son buffer est rempli à moitié.

La broche « **Master Reset RST** » permet de réinitialiser le circuit dans la condition de « power-up », en fonctionnement normal elle doit être maintenue à un **niveau logique haut**.

La broche « **Vout** » est la sortie analogique qui, pour piloter un haut-parleur, doit être amplifiée.

En ce qui concerne le jeu de commandes MSA et les codes des allophones, **reportez-vous aux tableaux 1**, **2** et **3** qui en montrent une liste partielle.

L'utilisation du circuit est relativement simple. Il suffit d'envoyer en **mode série** (avec le format : 9600, N, 8,1) **sur la broche RCX la séquence des allophones** que nous voulons reproduire.

Ci-après nous reportons à titre d'exemple quelques allophones et codes de contrôle utilisés pour la prononciation de certains mots en Anglais.

```
...
hello = \HE \FAST \EHLE \LO \OWWW
mail= \MM \EYIY \LO
dog = \DO \OH \OH \OG
memory = \MM \EH \MM \FAST \AXRR
\IY
goodbye=\Slow \GO \UW \OD \FAST \
P4 \SOFT \BO \OHIH
welcome = \WW \EHLE \KE \UX \MM
...
```

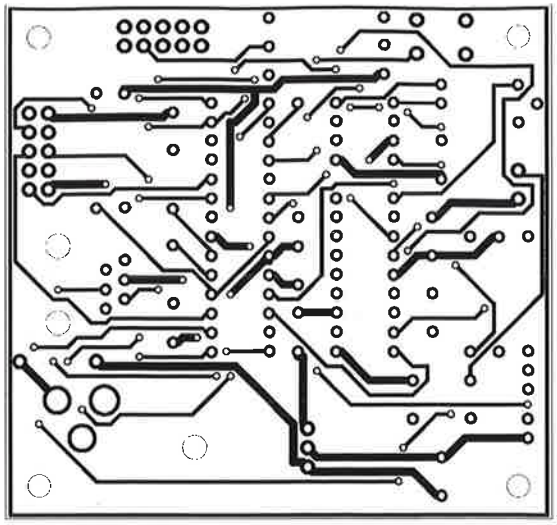


Figure 11 : circuit imprimé à l'échelle 1 : 1 côté soudures de la reconnaissance vocale des couleurs.

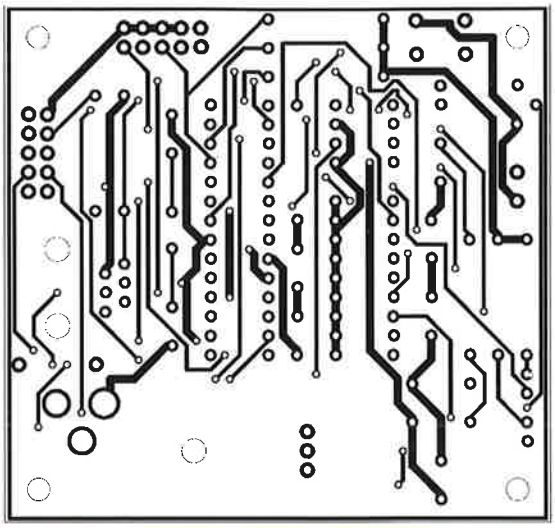


Figure 12 : circuit imprimé à l'échelle 1 : 1 côté composants de la reconnaissance vocale des couleurs.

Liste des composants du ET771

R1..... 470 Ω
R2..... 10 kΩ trimmer multitour
R3..... 10 Ω
R4..... 470 Ω
R5..... 10 kΩ
R6..... 10 kΩ
R7..... 10 kΩ
R8..... 10 kΩ
R9..... 10 kΩ
R10.... 470 Ω

C1..... 10 µF/100 V électrolytique
C2..... 220 µF/25 V électrolytique
C3..... 100 nF multicouche
C4..... 15 pF céramique
C5..... 15 pF céramique
C6..... 220 µF/25 V électrolytique
C7..... 100 nF multicouche
C8..... 100 nF multicouche
C9..... 100 nF multicouche
C10.... 100 nF multicouche
C11.... 100 nF multicouche
IC1..... LM386N
IC2..... PIC16F876A

Le schéma électrique

Le schéma du circuit de reconnaissance vocale des couleurs est relativement simple. Il est basé sur un microcontrôleur **PIC16F876** (IC2) qui dispose de 22 broches d'E/S configurables en entrées ou en sorties, et dont une grande partie est utilisée pour communiquer avec le module « **TCS3200-DB** » (RA3, RB0 et RC0 à RC5) ainsi qu'avec le circuit « **SpeakJet** » (RA4, RA5, RB1 à RB4 et RC6).

Pour le **comptage des impulsions**, le microcontrôleur dispose d'un compteur matériel (**TIMER1**) sur la broche **RC0** directement reliée à la sortie du module « **TCS3200-DB** ». Il dispose également d'une **interface série RS232** utilisée

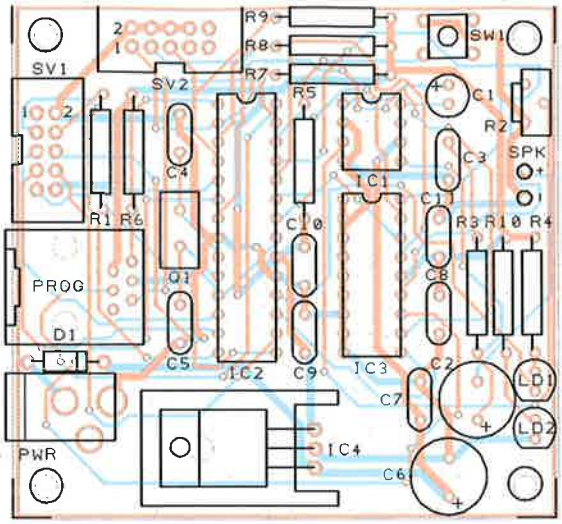


Figure 13 : schéma d'implantation des composants de la reconnaissance vocale des couleurs.

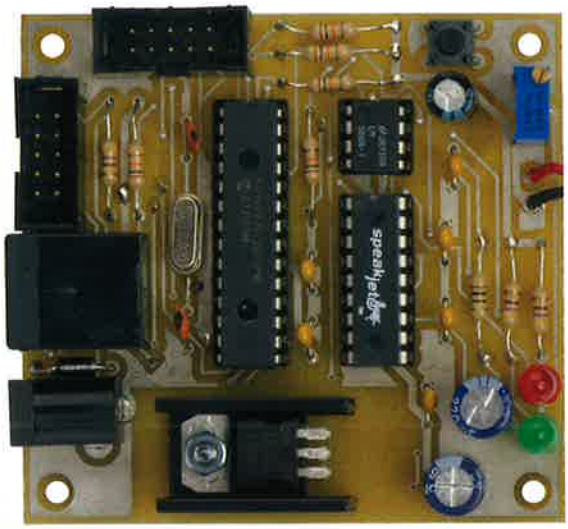


Figure 14 : photo de l'un de nos prototypes de la reconnaissance vocale des couleurs.

IC3..... SpeakJet
IC4..... 7805
D1..... 1N4007
LD1.... LED 5 mm rouge
LD2.... LED 3 mm verte
Q1..... quartz 20 MHz
SW1... poussoir

Divers :
Capteur TCS3200-DB *voir texte
Fiche d'alimentation pour ci
Dissipateur ML26
Vis 10 mm 3 MA
Ecrou 3 MA
Connecteur RJ11 pour ci
Support pour circuit intégré 2 x 14 broches
Support pour circuit intégré 2 x 9 broches
Support pour circuit intégré 2 x 4 broches
Embase mâle HE10 droite pour ci x 2
Connecteur femelle HE10
Haut-parleur 0,5W/8 Ω
Câble nappe 10 fils

pour communiquer avec le circuit « **SpeakJet** ». La tension d'alimentation est stabilisée à **5 VDC** grâce au régulateur **IC4** (7805), ce qui permet d'alimenter tous les circuits intégrés numériques.

En ce qui concerne le circuit **LM386** qui **amplifie le signal de sortie** du « **SpeakJet** », il est alimenté directement à partir de la tension d'entrée « **+ V** » afin de piloter correctement le haut-parleur.

Nous avons prévu des entrées supplémentaires accessibles via le connecteur « **SV2** » pour la connexion de photodiodes ou pour tout autre dispositif de détection de luminosité. Le connecteur « **PROG** » permet la **programmation de microcontrôleur en circuit** et aussi le débogage du programme à l'aide d'un débogueur approprié.

Le bouton poussoir « **SW1** » permet d'indiquer des événements au microcontrôleur, tels que la demande de reconnaissance d'une couleur. La LED « **LD1** », qui est gérée par la broche **RC7**, permet de signaler à l'utilisateur plusieurs événements, par exemple qu'une erreur s'est produite. La LED « **LD2** » indique que le circuit est **sous tension**.

Réalisation pratique

Comme vous pouvez le constater, le montage n'est pas très complexe à réaliser.

En fait il y a seulement quelques composants traversant soudés sur un circuit imprimé double face que vous pouvez fabriquer chez vous. N'oubliez pas de souder les « **vias** » aux endroits prévus afin que les pistes côté soudures et côté composants communiquent.

Vous pouvez télécharger les typons des circuits imprimés à partir de notre site web **www.electroniquemagazine.com** dans le sommaire détaillé de la revue **133** à l'onglet « **Télécharger** ».

Commencez par souder les vias, ensuite comme d'habitude soudez les composants ayant un profil bas tels que les résistances, les condensateurs non polarisés, le poussoir.

Continuez ensuite à souder les 3 supports des circuits intégrés, les condensateurs électrolytiques (respectez la polarité, le « - » est indiqué sur le boitier, la patte la plus longue étant le « + »), les connecteurs HE10 et la prise d'alimentation.

Terminez par le montage du régulateur de tension 7805, vous devez plier les pattes à 90 ° (attention à ne pas forcer) et fixer l'ensemble radiateur et régulateur sur le circuit imprimé. Aidez-vous des illustrations présentes dans l'encadré intitulé « Plan de montage ».

Reliez enfin le module « **TCS3200-DB** » au connecteur « J2 » au moyen d'un

Listing 2

Vous trouverez ci-dessous le code source complet du programme écrit en langage C ANSI. Le principe de fonctionnement est très simple. Le microcontrôleur attend une pression sur le bouton poussoir, puis entre dans le processus d'acquisition d'une couleur au format « RVB » du capteur. Ensuite il convertit le format « RVB » en format « HSL », et analyse le résultat de la conversion pour choisir l'ensemble des commandes à envoyer au synthétiseur vocal afin que celui-ci prononce la couleur identifiée.

```
/*****************************************************/
/*                                                   */
/*        TAOS INTERFACE                             */
/*                                                   */
/* Auteur          :       Fabio Riscica             */
/* Date            :       09/05/2015                */
/* Version         :       1.0                       */
/* Compilateur     :       CCS C COMPILER            */
/*                                                   */
/* Révision        :                                 */
/* 1.0  09/05/07           Version initiale.         */
/*                                                   */
/* (I) = input = entrée                              */
/* (O) = output = sortie                             */
/* TAOS = module TCS3200-DB                          */
/* Speakjet = circuit de synthèse vocale             */
/*****************************************************/

#include <16F876.h>
#fuses HS,NOWDT,PUT,NOPROTECT,BROWNOUT,NOLVP
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,BITS=8,STOP=1,xmit=PIN_C6)

#define AI0  PIN_A0          /* entrée externe 1 (I)                                  */
#define AI1  PIN_A1          /* entrée externe 2 (I)                                  */
#define AI2  PIN_A2          /* entrée externe 3 (I)                                  */
#define LED  PIN_A3          /* TAOS LED active à un niveau bas (O)                   */
#define M1   PIN_A4          /* Sélection du mode M1 du Speakjet (O)                  */
#define RST  PIN_A5          /* RESET du Speakjet active à un niveau bas (O)          */
#define FIN  PIN_B0          /* TAOS sortie en fréquence (I)                          */
#define D0   PIN_B1          /* Speakjet prêt (I)                                     */
#define D1   PIN_B2          /* Speakjet parle (I)                                    */
#define D2   PIN_B3          /* Buffer du Speakjet à moitié plein (I)                 */
#define M0   PIN_B4          /* Sélection du mode M0 du Speakjet (O)                  */
#define PUSH PIN_B5          /* Poussoir actif à un niveau bas (I)                    */
#define FOUT PIN_C0          /* TAOS sortie en fréquence (I)                          */
#define S1   PIN_C1          /* TAOS échelle de la fréquence (O)                      */
#define S0   PIN_C2          /* TAOS échelle de la fréquence (O)                      */
#define OE   PIN_C3          /* TAOS activation de la sortie fréquence active à un niveau bas (O) */
#define S2   PIN_C4          /* TAOS sélection du groupe de photodiode (O)            */
#define S3   PIN_C5          /* TAOS sélection du groupe de photodiode (O)            */
#define LED1 PIN_C7          /* LD1 active à un niveau bas (O)                        */

float hue,saturation,lightness;

#define MAX1(a,b) (((a) > (b)) ? (a) : (b))
#define MAX2(a,b) (((a) > (b)) ? (a) : (b))
#define MIN1(a,b) (((a) < (b)) ? (a) : (b))
#define MIN2(a,b) (((a) < (b)) ? (a) : (b))

void RGBtoHSL(int r,int g,int b)
{
 float fmax,fmin;
 float var_R,var_G,var_B;
 extern float hue,saturation,lightness;
```

```
var_R = r/255.0;
var_G = g/255.0;
var_B = b/255.0;

fmax = MAX1(MAX2(var_R,var_G),var_B);
fmin = MIN1(MIN2(var_R,var_G),var_B);

lightness = fmax;

if (fmax > 0)
        saturation = (fmax - fmin)/fmax;
else
        saturation = 0;

if (saturation == 0)
        hue = 0;
else
 {
  if (fmax == var_R)
    hue  = (var_G - var_B)/(fmax - fmin);
  else if (fmax == var_G)
    hue  = 2 + (var_B - var_R)/(fmax - fmin);
  else hue  = 4 + (var_R - var_G)/(fmax - fmin);
    hue = hue/6;
    if (hue < 0) hue += 1;
    hue*=360;
 }
}

void main()
{
 long red_value,green_value,blue_value;
 int red,green,blue;
 char buffer[16];

 extern float hue,saturation,lightness;

output_high(RST);
 output_low(M0);
 output_high(M1);
 output_high(OE);
 output_high(LED);
 output_high(S0);                    /* TAOS échelle de la fréquence à 100 % */
 output_high(S1);
 output_low(S2);
 output_low(S3);
 output_high(LED1);

setup_timer_1(T1_EXTERNAL_SYNC|T1_DIV_BY_1);

 while(1)
 {
  while(input(PUSH));                /* Poussoir pressé */
  buffer[0]=195;                     /* prononciation de l'expression en Anglais "color check" */
  buffer[1]=8;
  buffer[2]=134;
  buffer[3]=146;
  buffer[4]=7;
  buffer[5]=151;
  buffer[6]=5;
  buffer[7]=182;
  buffer[8]=131;
  buffer[9]=131;
```

```
buffer[10]=196;
buffer[11]=0;
printf("%s",buffer);
while(input(D1));
output_low(LED1);
output_low(OE);                    /* active le capteur */
output_low(LED);                   /* allume les 2 LED haute luminosité */
delay_ms(500);
output_low(S2);                    /* sélectionne le groupe de photodiodes rouges */
output_low(S3);
delay_ms(10);
set_timer1(0);
delay_ms(1000);
red_value=get_timer1();            /* lit le groupe de photodiodes rouges */
output_high(S2);                   /* sélectionne le groupe de photodiodes vertes */
output_high(S3);
delay_ms(10);
set_timer1(0);
delay_ms(1000);
green_value=get_timer1();          /* lit le groupe de photodiodes vertes */
output_low(S2);                    /* sélectionne le groupe de photodiodes bleues */
output_high(S3);
delay_ms(10);
set_timer1(0);
delay_ms(1000);
blue_value=get_timer1();           /* lit le groupe de photodiodes bleues */
output_high(LED);

red = red_value*255.0/31261;
green = green_value*255.0/46385;
blue = blue_value*255.0/63866;

RGBtoHSL(red,green,blue);          /* convertit le format RVB en format HSL */

output_high(LED1);

if (lightness >= 0.90)             /* Si la couleur est blanche*/
{                                  /* Commandes de l'allophone */
  buffer[0]=185;                   /* prononciation de l'expression en Anglais "white" */
  buffer[1]=155;
  buffer[2]=191;
  buffer[3]=0;
  printf("%s",buffer);
  while(input(D1));
}
else if (hue<30)                   /* Si la couleur est rouge */
{                                  /* Commandes de l'allophone */
  buffer[0]=148;                   /* prononciation de l'expression en Anglais "red" */
  buffer[1]=8;
  buffer[2]=134;
  buffer[3]=131;
  buffer[4]=176;
  buffer[5]=0;
  printf("%s",buffer);
  while(input(D1));
}
else if (hue<73)                   /* Si la couleur est jaune */

{                                  /* Commandes de l'allophone */
  buffer[0]=128;                   /* prononciation de l'expression en Anglais "yellow" */
  buffer[1]=131;
  buffer[2]=146;
  buffer[3]=137;
```

```
      buffer[4]=0;
     printf("%s",buffer);
     while(input(D1));
   }
  else if (hue<160)                    /* Si la couleur est verte */
   {                                   /* Commandes de l'allophone */
     buffer[0]=8;                      /* prononciation de l'expression en Anglais "green" */
     buffer[1]=179;
     buffer[2]=7;
     buffer[3]=148;
     buffer[4]=8;
     buffer[5]=128;
     buffer[6]=141;
 buffer[7]=0;
     printf("%s",buffer);
     while(input(D1));
   }
  else if (hue<252)                    /* Si la couleur est bleue */
   {                                   /* Commandes de l'allophone */
     buffer[0]=171;                    /* prononciation de l'expression en Anglais "blue" */
     buffer[1]=7;
     buffer[2]=146;
     buffer[3]=162;
     buffer[4]=0;
     printf("%s",buffer);
     while(input(D1));
   }
  else if (hue<360)                     /* Si la couleur est pourpre */
   {                                   /* Commandes de l'allophone */
     buffer[0]=199;                    /* prononciation de l'expression en Anglais "purple" */
     buffer[1]=7;
     buffer[2]=151;
     buffer[3]=199;
     buffer[4]=8;
     buffer[5]=146;
     buffer[6]=0;
     printf("%s",buffer);
     while(input(D1));
    }
  }
}
```

câble plat comportant 10 fils et de longueur appropriée. Reportez-vous à la figure 3 pour le brochage du module.

Vérifiez si tous les composants sont à leur place avec les bonnes valeurs (notamment pour les résistances). **Mettez le circuit sous tension sans insérer les 3 circuits intégrés** et **sans relier le module** « **TCS3200-DB** ».

À l'aide d'un **multimètre vérifiez directement sur les supports des circuits**, si vous avez + 5 V entre les broches 20 et 8, 19 de IC2 et les broches 14 et 5 de IC3. Vérifiez aussi la tension entre les broches 6 et 4 de IC1, vous devez avoir aux environs de 8,4 V, ce qui correspond à la tension « + V ».

Le montage doit être alimenté par un bloc secteur ou une alimentation de 9 VDC stabilisée si possible.

Mettez le montage hors tension, attendez quelques instants afin que les condensateurs se déchargent. Ensuite, à l'aide du détrompeur, insérez dans le bon sens le microcontrôleur, le « **SpeakJet** » et le **LM386**.

Programmez le PIC avec le fichier « **.hex** » « **MF771.hex** » à l'aide d'un programmateur en circuit, en le reliant au connecteur « PROG », sinon vous pouvez le faire avec un programmateur normal mais il faudra alors à chaque fois enlever et remettre le PIC dans son support si vous devez apporter des modifications au programme.

Le programme « **MF771.hex** » est **disponible avec les typons des circuits imprimés** (voir plus haut).

Le programme et l'utilisation pratique

Le programme de gestion du microcontrôleur a été écrit en **langage C ANSI** à l'aide du compilateur « **PIC-C** » correspondant à un produit de milieu de gamme de la marque « **CCS** ».

Vous pouvez télécharger gratuitement une **version complète limitée à 45 jours d'utilisation** à l'adresse suivante :

http://www.ccsinfo.com/ccsfree-demo.php.

Quelques appareils de reconnaissance vocale des couleurs dans le commerce

Le marché offre différents modèles d'appareils de reconnaissance vocale des couleurs. Ils peuvent être achetés dans les magasins ou sur les sites Internet recommandés par les différentes associations de personnes malvoyantes.

Voici quelques modèles :

- le « **ColorTest 2000** » (en figure A) peut identifier plus de 1700 couleurs différentes. Son prix est aux alentours de 799 € (www.marland.eu/fr/) ;
- le « **Colorino** » (en figure B) peut différencier plus de 150 couleurs. Son prix est de 175 € (www.ceciaa.com);
- le « **Color Teller** » (en figure C) peut détecter 42 couleurs. Son prix est d'environ 110 € (www.brytech.com/colorteller/index.htm).


Fig. A


Fig. B

Fig. C

Les opérations qu'effectue le programme sont essentiellement au nombre de 3.

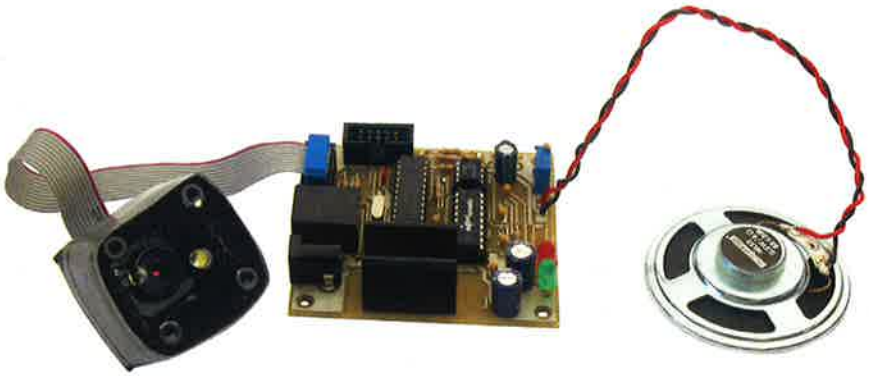
Tout d'abord, le programme scrute si le bouton est pressé, si c'est le cas il active le module « **TCS3200-DB** » et compte les impulsions générées correspondant aux 3 groupes différents de photodiodes.

Ces fréquences sont mémorisées et le format « **RVB** » est converti au format « **HSL** » (Teinte, Saturation, Luminosité) selon l'algorithme du Listing 1.

Ensuite, le format « **HSL** » est analysé et les commandes correspondantes sont sélectionnées, afin de piloter correctement le circuit « **SpeakJet** » pour qu'il puisse prononcer la couleur « observée ».

Dans le programme nous avons prévu la gestion d'un nombre limité de couleurs de base, la luminosité n'est pas prise en compte et la procédure d'étalonnage du blanc n'est pas présente.

Le **programme identifie correctement 6 couleurs**, à savoir : le blanc (white), le rouge (red), le jaune (yellow), le vert (green), le bleu (blue), le violet (purple). La normalisation des niveaux « **RVB** » a été effectuée d'une manière empirique par des mesures en laboratoire, en correspondance avec les valeurs standards des couleurs rouge, verte et bleue.



Après le montage du circuit et la programmation du microcontrôleur, nous pouvons procéder au test de notre dispositif de détection des couleurs.

Pour effectuer une 1ère vérification, il est souhaitable de **disposer de feuilles comportant des couleurs de base rouge**, **verte** et **bleue**. Les feuilles peuvent être réalisées à l'aide d'une imprimante couleur.

Le **capteur de couleur doit être correctement protégé de la lumière ambiante le long de ses bords afin d'éviter des perturbations**.

Vous pouvez remarquer que nous avons enveloppé avec du carton et de l'adhésif noir tous les côtés du capteur.

Placez le module sur une feuille et appuyez sur le poussoir. L'expression « **color check** » doit être **prononcée** et vous remarquerez l'**activation des 2 LED haute luminosité**. Après environ 3 secondes, l'appareil **annoncera le nom de la couleur détectée**.

À ce stade, vous pouvez également vérifier des objets d'autres couleurs, assurez-vous que les couleurs sont correctement détectées.

Comme vous pouvez le remarquer dans le programme, la reconnaissance d'une couleur se fait simplement à l'aide d'une série d'instructions « IF » et « ELSE – IF » qui analysent les différentes valeurs des variables de la luminosité et de la teinte.

Il ne sera pas difficile d'ajouter d'autres instructions « IF » dans le programme (attention la mémoire est limitée) et de nouvelles commandes pour le « **SpeakJet** » afin que l'appareil détecte des couleurs supplémentaires.

Vous remarquerez que la technique de la synthèse vocale a des limites en termes de qualité de reproduction, étant donné que la prononciation s'avère être « métallique ».

Conclusion

Nous devons remercier la Doctoresse Monica Cusenza pour son étude sur la représentation des couleurs et la mise en œuvre d'un dispositif de détection dans le laboratoire d'instrumentation biomédicale de l'université de Trieste.

Nous tenons à remercier en particulier le Professeur Agostino Accardo pour les explications apportées au cours de ce projet.

L'appareil décrit dans cet article ne rivalise pas en termes de nombre de couleurs détectées par rapport aux appareils du commerce, mais il a pour mérite d'offrir une approche didactique du problème posé.

Cependant, avec une modification du programme et/ou l'utilisation d'un microcontrôleur plus performant, le nombre de couleurs détectées peut approcher la centaine.

Le module « **TCS3200-DB** » est disponible chez **Parallax**, le « **SpeakJet** » est disponible chez **Magnevation** ou **Lextronic**.

Les typons des circuits imprimés ainsi que le programme **MF771.hex** sont téléchargeables gratuitement sur notre site à l'adresse suivante :

www.electroniquemagazine.com

dans le sommaire détaillé de la revue numéro **133** section « **Télécharger** ». ■

3DRAG - 13

Comment fabriquer un boîtier avec la 3DRAG

de SIMONE MAJOCCHI

L'imprimante 3DRAG est bien adaptée à la fabrication d'objets utiles. Nous vous proposons de réaliser un boitier qui accueillera le montage d'impression autonome décrit dans le précédent numéro 132 d'Électronique et Loisirs Magazine.

Dans le précédent article, nous avons décrit un **circuit électronique pouvant se connecter à la 3DRAG, afin de gérer de manière autonome** (sans PC) les **fichiers d'impressions** au format **G-Code**, **directement** à partir d'une **carte mémoire SD**.

Le montage était doté d'un afficheur LCD, d'un encodeur rotatif et d'un bouton, afin de surveiller l'impression en cours. Le circuit a été réalisé avec l'idée de l'intégrer dans un boitier, mais bien évidemment il n'existe pas sur le marché un boitier déjà percé et usiné aux endroits convenables. Dans le cas d'un boitier standard nous devons donc effectuer le perçage et la découpe de l'afficheur.

C'est ainsi que nous devions procéder jusqu'à il y a quelques années et nos lecteurs passionnés d'électronique savent que la partie mécanique d'une réalisation électronique est la partie la plus laborieuse.

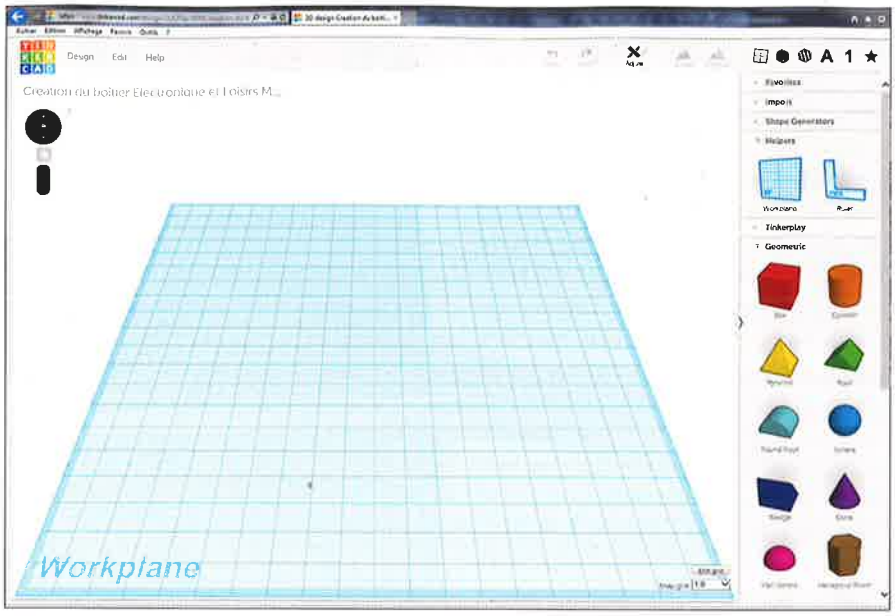


Figure 1 : ici l'interface lors du démarrage d'un nouveau projet. Cliquez sur « Create new design » (Créer un nouveau projet) après vous être enregistré sur le site.

D'autant plus lorsqu'il faut travailler sur un boitier en aluminium. Nous avons donc pensé à fabriquer un boitier sur mesure, grâce à la 3DRAG, pour notre montage, celui-ci ne nécessitant qu'un boitier plastique.

Comme la 3DRAG peut imprimer des objets de taille moyenne (20 cm), elle peut également « imprimer » (fabriquer) des boitiers sur mesure pour chacun des projets que nous voulons réaliser (en restant dans la limite de travail de l'imprimante).

Des sites web tels que « Thingiverse » contiennent énormément de modèles et des pièces de toutes sortes, mais il est presque certain que vous ne trouverez pas le modèle correspondant exactement à votre projet. Cela a été le cas pour nous.

La solution passe alors par la conception à l'aide d'un logiciel de CAO, cela est trop compliqué pour un simple boîtier. Cependant avec l'aide d'outils adéquats nous pouvons concevoir des objets relativement complexes, même pour les débutants.

Les outils du commerce

Pour aborder le problème de la conception, il est nécessaire d'utiliser plusieurs instruments et, plus précisément, nous devons utiliser un gabarit, un mètre, un crayon, du papier et bien sûr une application de modélisation sous forme de solide.

Le crayon et le papier vous permettront de noter les mesures effectuées avec le mètre et le gabarit, et ensuite de faire une première esquisse pour réaliser le modèle dans une application CAO. Dessiner les détails et réfléchir aux solutions techniques est une étape très importante pour mettre en place la structure géométrique que vous devrez dessiner sur votre ordinateur.

Un élément supplémentaire à prendre en compte lors de la conception de la pièce est la capacité d'impression de la 3DRAG pour l'impression des parties inclinées. Vous devrez avoir recourt à des supports. Ceux-ci augmenteront le temps d'impression et seront fastidieux à enlever une fois l'objet imprimé.

L'idéal est de concevoir l'objet sans trop créer d'éléments critiques supplémentaires que sont les supports, nous étudierons plus tard quelques solutions à ce problème.

Quelle application ?

Avec le développement de l'impression 3D et le nombre croissant d'imprimantes 3D sur le marché, la conception et la modélisation des solides sont devenues des questions d'actualités.

De plus en plus d'utilisateurs, dont le seul but est de pouvoir imprimer sur leur imprimante 3D personnelle, recherchent des solutions professionnelles adaptées à la création de fichiers.

Il n'est pas nécessaire alors de disposer de bibliothèques de pièces ou de méthodes dans les différents domaines telles que la conception, l'architecture et la mécanique.

Il suffit simplement d'utiliser une solution qui gère différentes unités de mesure tels que les millimètres ou les pouces (pour ceux qui utilisent le système impérial de mesure), qui permet d'obtenir des primitives qui peuvent être assemblées sous forme de somme ou de soustraction logiques, et qui est dotée d'instruments de positionnement et de centrage, sans avoir des menus complexes souvent difficiles à interpréter.

Après avoir essayé plusieurs solutions, nous avons constaté que même avec des limites et des défauts, l'application web « **Tinkercad** » pourrait être la meilleure solution pour ceux qui n'ont jamais abordé la question de la conception de modèles tridimensionnels.

Étant donné que c'est une application web, elle fonctionne avec un navigateur qui prend en charge la technologie « WebGL », comme par exemple Google Chrome ou Mozilla Firefox.

L'application ne nécessite pas l'ajout de programme supplémentaire à télécharger et à installer. Son fonctionnement s'effectue donc uniquement en ligne, et tous vos travaux sont stockés directement sur le serveur de « **Tinkercad** ».

Une fois votre compte créé vous avez toujours à disposition vos travaux et l'application, quel que soit le type d'ordinateur que vous utilisez. La seule condition est d'être connecté à Internet avec un navigateur compatible.

Le choix de cette solution a été encouragé par le fait que l'application web a été rachetée par un acteur historique du monde de la CAO (Autodesk), ainsi le service perdurera dans le temps.

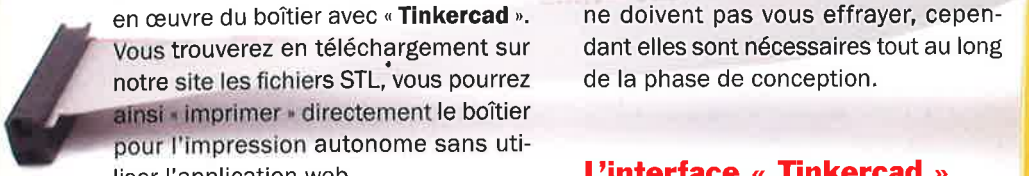
Notre boîtier

Tout d'abord, nous pouvons vous dire que l'ensemble du boîtier, les supports d'impression pour la 3DRAG, les caches du bouton de réinitialisation et du contacteur rotatif ont été générés avec « **Tinkercad** » sans trucs ou astuces particulières.

Pour des raisons de place dans la revue, nous allons nous concentrer uniquement sur une partie de la mise en œuvre du boîtier avec « **Tinkercad** ». Vous trouverez en téléchargement sur notre site les fichiers STL, vous pourrez ainsi « imprimer » directement le boîtier pour l'impression autonome sans utiliser l'application web.

Pour les plus courageux, nous abordons la réalisation du boîtier. Souvenez-vous qu'il doit y avoir une fenêtre pour l'afficheur LCD, un trou pour l'encodeur et une solution mécanique qui nous permette d'appuyer sur le bouton de réinitialisation qui se trouve sur le circuit imprimé. Celui-ci mesure 160 mm de largeur par 62 mm d'hauteur, et a une épaisseur de 35 mm avec l'afficheur LCD.

Avec ces mesures de base, nous pouvons commencer à imaginer ce que pourraient être les dimensions du boîtier. Il faut ajouter 2 mm aux dimensions du circuit imprimé avec 2 mm en plus pour l'épaisseur des côtés. Ensuite il faut tenir compte d'une hauteur de 10 mm pour les fixations du circuit imprimé plus une épaisseur de 2 mm pour le fond du boîtier.



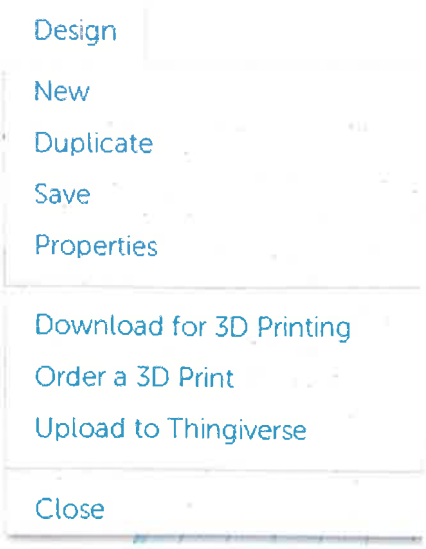


Figure 2 : ici les différentes fonctions du menu « Design ».

L'écran LCD mesure 45 mm par 90 mm, le câble et la carte SD ont besoin d'une encoche d'au moins 25 mm. Le trou pour l'encodeur rotatif doit être de 15 mm pour accueillir l'axe de 10 mm de Ø. Le bouton de réinitialisation est positionné à environ 30 mm en dessous du niveau de l'écran LCD et cela doit être pris en compte pour le système d'activation du bouton. Toutes ces mesures ne doivent pas vous effrayer, cependant elles sont nécessaires tout au long de la phase de conception.

L'interface « Tinkercad »

Avant de commencer le projet, il est nécessaire de se familiariser avec l'interface de l'application web « **Tinkercad** ». Pour cela ouvrez votre navigateur Google Chrome ou Mozilla Firefox et entrez l'adresse suivante :

https://www.tinkercad.com

Pour ouvrir la page principale vous **devez d'abord créer un compte qui est gratuit**, ensuite vous pourrez accéder à votre espace personnel. « Tinkercad » propose un certain nombre de tutoriels (malheureusement en Anglais) pour apprendre à utiliser les principales fonctionnalités de l'application.

En figure 1, vous pouvez voir l'interface lors du démarrage d'un nouveau projet lorsque vous cliquez sur « **Create new design** » (créer un nouveau projet) dans votre espace personnel.

Avant de continuer vous devez effectuer quelques modifications dans votre navigateur. Voici quelques astuces pour activer « WebGL » dans Chrome, Firefox et Safari.

- **Chrome**

Entrez « about:flags » dans la barre d'adresse. Recherchez WebGL puis activez l'option.

- **Firefox**

Entrez « about:config » dans la barre d'adresse. Recherchez « webgl.force-enabled » et changez la valeur à « true » en cliquant sur la ligne.

Recherchez « webgl.disabled », si la valeur existe changez la valeur à « false ».

- **Safari**

Ouvrez le menu « Safari » et sélectionnez « Préférences ». Cliquez sur l'onglet « Avancés ». Cochez le champ « Afficher le menu Développement dans la barre des menus ». Ouvrez le menu « Développement » et cliquez sur « Activer WebGL ».

Notez qu'avec **Internet Explorer 11**, tout fonctionne correctement sans modifications.

En haut à gauche se trouve le bouton « **Tinkercad**» qui est un raccourci pour les paramètres de votre compte.

Cliquez sur « **New** », une nouvelle fenêtre apparaît avec 3 boutons en haut à droite. En fait ce sont 3 menus : « **Design** », « Edit » et « Help » à partir desquels vous pouvez accéder aux différentes fonctions pour la conception de votre projet.

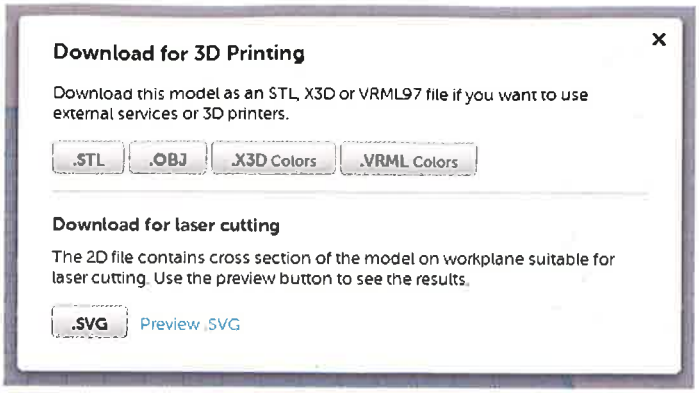


Figure 3 : ici les différents boutons de gestion du modèle et de téléchargement des fichiers dans différents formats.

Figure 4 : vous pouvez commander une impression.

La première chose à souligner est que l'**application se préoccupe d'enregistrer votre travail sans que vous ayez à le faire**. Périodiquement et lorsque vous réalisez certaines opérations, vous activez une sauvegarde automatique et votre travail est toujours enregistré.

Un autre élément important concerne le bouton « **Undo** » situé en haut vers la droite et qui vous permet de revenir en arrière après une fausse manipulation ou si vous n'êtes pas sûr de l'opération en cours. Quand vous appuyez une ou plusieurs fois sur le bouton « **Undo** », un autre bouton dont la fonction est opposée apparaît. Il s'agit bouton « **Redo** » qui permet de refaire les étapes que vous venez d'annuler dans l'ordre inverse.

Un dernier élément important concerne la nature des objets dans le programme. « Tinkercad » garde en mémoire l'historique et la structure des primitives utilisées. Il est donc toujours possible de désassembler les éléments et de les regrouper dans un ou plusieurs groupes (boutons « Group » et « Ungroup »).

Dans le menu « **Design** », que vous pouvez voir en figure 2, il existe 3 groupes de fonctions. Dans le premier groupe, le bouton « **New** » permet d'abandonner le projet en cours et d'en créer un nouveau. Le bouton « **Duplicate** » copie le projet en cours et affiche la copie pour d'éventuelles modifications.

Le bouton « **Save** » force l'enregistrement du projet en cours et le bouton « **Properties** » permet de modifier le nom du projet et d'accéder aux paramètres de publication et des droits d'auteur du projet.

Si les propriétés définies dans « **Public access settings** » sont de type « **Private** », le travail est visible uniquement par vous. Si dans « **Visibility** » vous choisissez « **Public** », le projet devient visible pour tous les autres utilisateurs, ils peuvent télécharger une copie de votre projet et le modifier pour leurs propres besoins.

Le second groupe contient les fonctions pour la gestion du modèle et pour le téléchargement des fichiers dans différents formats appropriés, pour un traitement ultérieur et pour l'impression, comme indiqué en figure 3.

Vous pouvez commander une impression réalisée avec des machines professionnelles et qui est proposée par des partenaires du site « Tinkercad » (voir la figure 4).

Vous pouvez charger (uploader) un projet provenant de votre compte « Thingiverse » (www.thingiverse.com), si vous en possédez un.

Le troisième groupe permet de fermer l'éditeur d'objet et de revenir à votre page d'accueil. Vous y trouverez des images miniatures représentant tous vos projets (voir la figure 5).

A partir des miniatures, vous pouvez afficher une prévisualisation statique ou une vue en 3D interactive en mode édition d'objet. Si vous avez accès à des projets d'autres personnes, vous aurez les mêmes fonctions, mais les modifications ne seront possibles qu'à partir d'une copie du projet original.

À côté des boutons « **Undo** » et « **Redo** » du menu (à haut à droite), vous pouvez voir les boutons « **Adjust** », « **Group** » et « **Ungroup** ». Encore une fois, les icônes deviennent actives seulement lorsque vous êtes dans une situation qui l'exige. Lorsque les icones sont actives, elles prennent une couleur noire foncée, et lorsqu'elles sont désactivées elles apparaissent transparentes (ou grisées).

Lorsque vous cliquez sur « **Adjust** », 2 sous menus apparaissent : « **Align** » et « **Mirror** ». Ces 2 sous menus activent 2 fonctions (alignement et miroir) qui permettent soit d'aligner, soit d'effectuer une vue « miroir » de l'objet. Par exemple, l'image reflétée par un miroir plan est une image symétrique de l'objet par rapport au plan du miroir.

En continuant l'exploration de l'interface, vous trouverez une série d'icônes positionnées sur la barre latérale avec une palette d'objets. Ce sont des raccourcis des fonctions.

Les premières étapes de l'édition

Lorsque vous choisissez la forme qui vous convient sur la palette de droite, **cliquez dessus et faites-la glisser** sur la surface de travail. La forme géométrique sélectionnée apparaît automatiquement sur le plan de travail, ses dimensions sont de 20 mm par 20 mm et sa hauteur dépend de sa forme. Les lettres et les nombres sont par contre plus petits.

En figure 6 vous pouvez voir un cube, créé par défaut à partir de l'onglet « **Geometric** » puis en sélectionnant « **Box** ». En cliquant sur le petit carré blanc situé en bas, deux ancrages

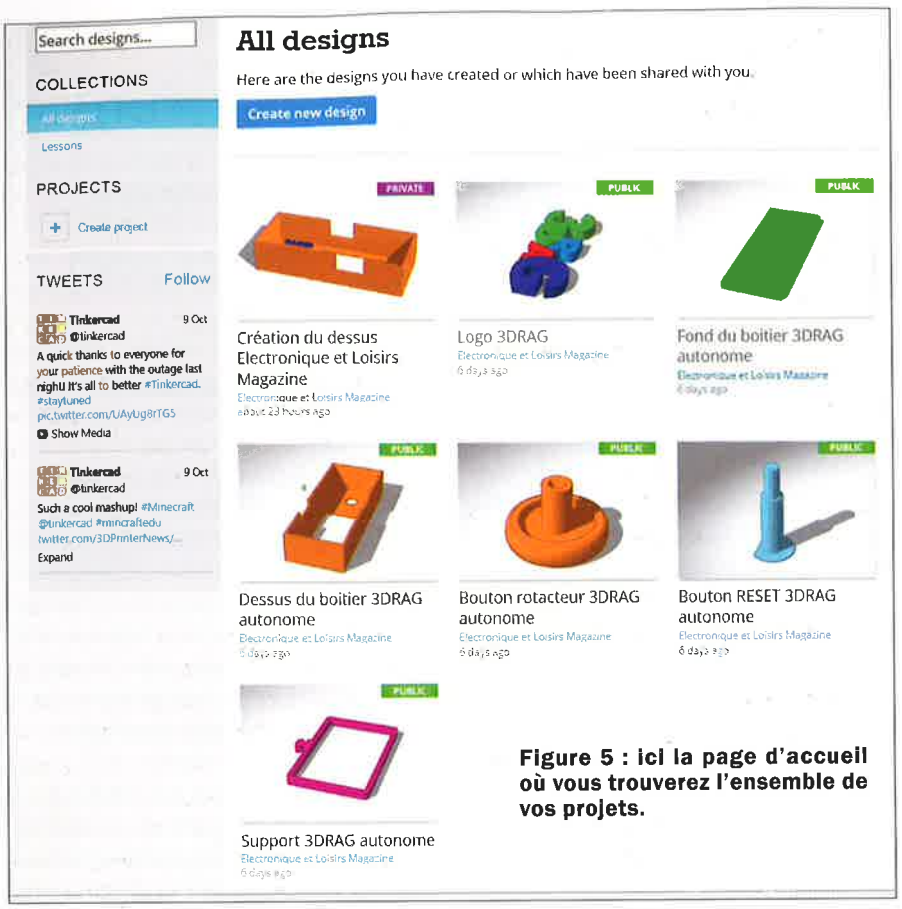


Figure 5 : ici la page d'accueil où vous trouverez l'ensemble de vos projets.

(cotations) apparaissent et indiquent les dimensions du cube. Vous remarquerez aussi que le petit carré blanc devient rouge, cela confirme la fonction d'affichage des dimensions.

Si vous sélectionnez le petit carré rouge avec la souris, en faisant un « glissé » tout en maintenant le bouton gauche de la souris appuyé, vous modifiez les dimensions de l'objet.

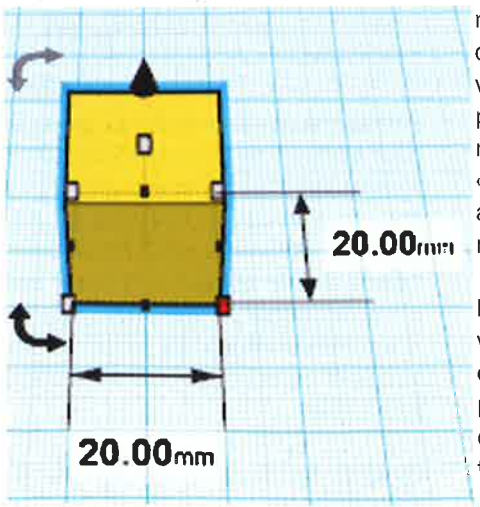


Figure 6 : ici le cube créé par défaut à partir de l'onglet « Geometric » en sélectionnant « Box ».

Vous pouvez répéter l'opération sur les différents petits carrés blancs de l'objet pour obtenir les dimensions désirées.

Notez que les carrés blancs agissent sur 2 dimensions, les carrés noirs sur une seule dimension (un seul côté dans le cas du cube).

Les **flèches permettent la rotation de l'objet dans tous les sens** selon un cercle gradué. Si vous cliquez uniquement sur l'objet et non sur les petits carrés (noirs ou blancs) ou les flèches, vous déplacez simplement l'objet sur la plan de travail. Dans ce cas, le déplacement est visualisé et les distances « parcourues » par les deux axes sont affichées, elles indiquent le déplacement par rapport au point de départ.

Notez que le quadrillage du plan de travail représente une grille dont les grands carreaux font 10 mm, cela permet de positionner les objets, de connaître la distance qui sépare 2 objets ou la distance de déplacement d'un objet.

La figure 7 montre comment modifier les réglages de la grille. Cliquez sur le bouton « **Edit grid** » en bas à droite, la fenêtre de propriétés de la grille s'ouvre (Grid properties). Le champ « **Units** » permet de choisir l'unité de mesure en millimètre ou en Inches (Pouces). **1 Inch vaut 25,4 mm**. Les champs « **Width** » et « **Height** » permettent de modifier les dimensions de l'espace de travail.

Par défaut nous avons 200 mm par 200 mm, ce qui correspond à la surface d'impression maximale de la 3DRAG. Vous pouvez aussi sélectionner un autre espace de travail pour un autre type d'imprimante. Pour cela utilisez la liste déroulante « **Use a Preset** ». Lorsque vos réglages sont terminés, vous devez cliquer sur le bouton bleu « Update Grid » pour que les nouveaux paramètres de la grille soient enregistrés.

Bien que l'application Tinkercad semble simple à première vue, elle offre des fonctions cachées qui simplifient certaines opérations. En haut à gauche vous pouvez voir un cercle contenant 4 flèches et le symbole d'une maison. Les flèches permettent une rotation de l'objet.

En cliquant sur le symbole de la maison, vous revenez à l'affichage de départ. En dessous du cercle contenant les 4 flèches, vous apercevez une petite icône représentant une vue 3D. Elle permet d'ajuster la vue de votre objet à votre espace de travail. Les 2 petits boutons en dessous « + » et « - » permettent d'effectuer un « zoom+ » ou un « zoom- ».

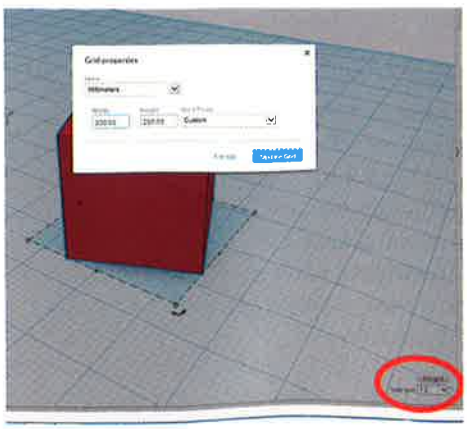


Figure 7 : cliquez sur le bouton « Edit grid » (en bas à droite entouré de rouge), les différents réglages de la grille apparaissent dans la fenêtre « Grid properties ».

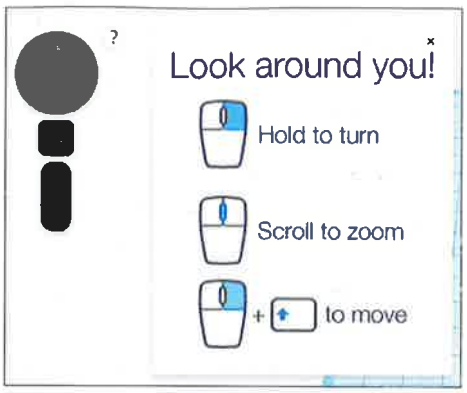


Figure 8 : ici les différents contrôles que vous pouvez obtenir grâce aux boutons de la souris.

En figure 8, vous pouvez voir les différents contrôles qu'il est possible d'obtenir grâce aux boutons de la souris. Pour afficher ces contrôles, cliquez sur le point d'interrogation (?) situé en haut à gauche à côté du cercle contenant les 4 flèches et le symbole d'une maison. Vous voyez apparaître un encadré dénommé « Look around you ! » (Regardez autour de vous !) qui fournit 3 fonctions.

- « **Hold to run** » : maintenez appuyé le bouton droit de la souris et manœuvrez-la. Le plan de travail se déplace selon les 3 axes (dans tous les sens) ;
- « **Scroll to zoom** » : en faisant tourner la roulette de la souris vers l'avant, vous agrandissez (zoom+) l'image et donc les détails. Si vous allez vers l'arrière, vous réduisez la taille de l'image (zoom-) ;
- « **To move** » : en maintenant appuyé le bouton droit de la souris et la touche « **SHIFT** » (flèche vers le haut) du clavier qui se situe en dessous de la touche « MAJ », vous déplacez l'ensemble « objet + espace de travail » dans une direction donnée.

Lorsque vous faites un « zoom + », vous pouvez bouger l'objet d'une unité de carreau dans les 4 directions grâce aux 4 flèches du clavier. Cela permet de positionner avec précision l'objet. Si au cours d'une opération vous lâchez la touche « **SHIFT** », l'action en cours continue jusqu'à obtenir l'effet prévu.

Lorsque vous redimensionnez l'objet en cliquant sur un des carrés blancs tout en maintenant la touche « **SHIFT** » enfoncée, le redimensionnement a lieu selon les 3 axes, alors que sans la touche « **SHIFT** » enfoncée, il ne s'effectue que sur 2 axes.

Cependant, le redimensionnement proportionnel peut provoquer la perte de l'accrochage de l'objet selon l'axe X ou Y ou sur l'un des 2 côtés. La grille s'accroche toujours du côté où vous utilisez les petits carreaux.

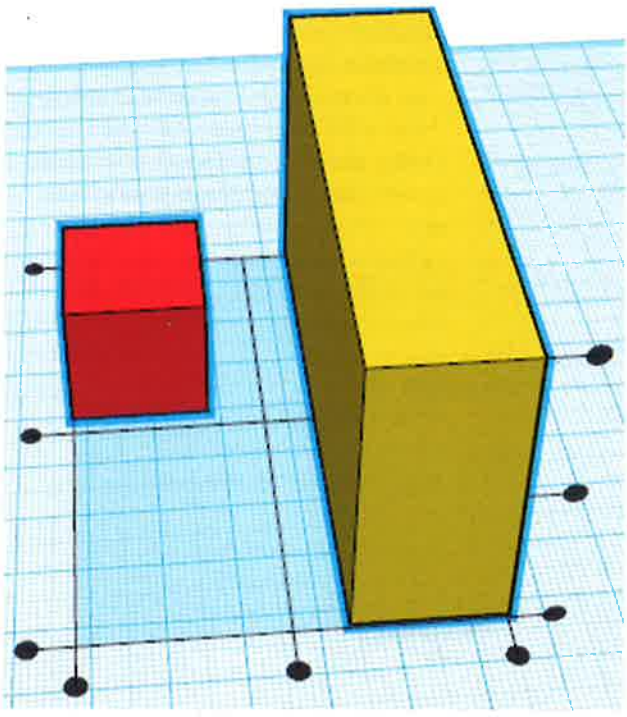


Figure 9 : autour des deux solides sélectionnés apparaît une série de points d'accrochage.

Si un côté a une valeur décimale mais que vous voulez l'accrocher à la grille, vous devez agir sur les autres petits carreaux pour le faire correspondre au pas de la grille.

Alignement et centrage

Travailler avec une grille permet d'avoir la précision requise pour la modélisation d'un solide (figure de base : box, cylinder, sphere, letters, etc.) qui est basée sur l'addition et la soustraction des différentes parties. Mais lorsque les dimensions ne coïncident plus avec l'accrochage de la grille, vous pouvez utiliser l'outil d'alignement qui permet de faire correspondre les bords du solide avec la grille , ou de centrer exactement le solide selon un ou plusieurs axes.

Pour que la fonction soit active, vous devez sélectionner au moins 2 solides. Pour sélectionner le premier solide cliquez dessus, puis appuyez et maintenez la touche « **SHIFT**» enfoncée et sélectionnez l'(les) autre(s) solide(s) en cliquant dessus. Une fois les solides sélectionnés, cliquez sur l'icône « **Adjust** » en haut à droite, puis cliquez sur « **Align** ».

Autour des deux (ou plus) solides sélectionnés, apparaît une **série de points d'accrochage,** comme illustré dans les figure 9. Les points d'accrochage sur le côté gauche et l'avant de la figure correspondent à l'alignement sur le plan de travail, tandis que les 3 points d'accrochage à droite visuellement sur le solide jaune dans le sens vertical correspondent à l'alignement sur la base (du solide jaune) centré verticalement ou vers le haut.

En cliquant sur chaque point noir (points d'accrochages), vous obtenez une prévisualisation de l'alignement.

Lorsque des solides sont déjà alignés dans une situation spécifique (par exemple alignés selon l'axe des X), le (les) point(s) d'accrochage(s) sont de couleur grise, pas noire foncée. Lorsque deux solides sont alignés sur un côté ou sur l'autre, il est certain que les faces du côté correspondant sont parfaitement alignées, cela est vital pour rendre les opérations de soustraction efficaces.

Somme et soustraction

Les dernières fonctions de cette présentation assez complète de Tinkercad sont les fonctions « **group/ungroup** » (grouper/dégrouper) et « **color** », c'est-à-dire la couleur des solides. Lorsque deux solides à grouper ont une couleur différente (vous comprendrez la raison de cette spécification), ceux-ci sont fusionnés en un seul objet (composé de plusieurs solides de base) plus complexe (voir les figures 10 et 11).

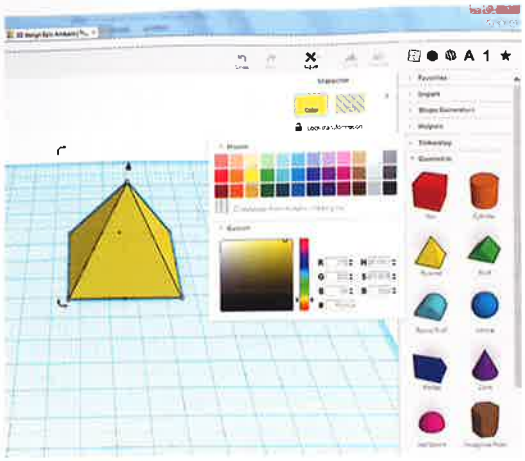


Figure 12 : ici vous pouvez voir le panneau « Inspector » comprenant les différents réglages de la couleur d'un solide. À côté du rectangle jaune, en haut du panneau, se trouve le bouton activant la fonction « Hole ».

positionnez un solide sur le plan de travail (voir la figure 12). Il vous permet de définir la couleur du solide, soit à l'aide d'une palette de **couleurs standard** « **Presets** », soit à l'aide de **couleurs personnalisées** « **Custom** ».

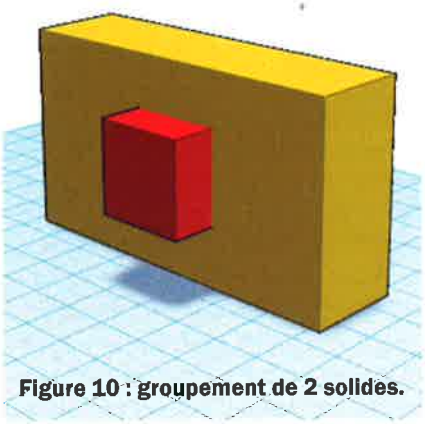


Figure 10 : groupement de 2 solides.

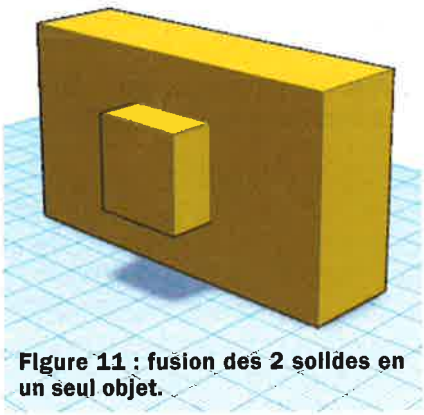


Figure 11 : fusion des 2 solides en un seul objet.

Ce n'est pas simplement une fusion de la surface extérieure mais de l'ensemble, c'est-à-dire de l'intérieur des solides.

Par exemple, si une sphère est creuse à l'intérieur et que nous ajoutons un cylindre, celui-ci va remplir partiellement la sphère creuse lorsqu'il est fusionné avec la fonction « **Group** ». La couleur unique prise par les solides groupés confirme la transformation en un seul objet.

Le panneau « **Inspector** » apparaît (en haut à droite) chaque fois que vous

En cliquant sur « **Hole** » à côté de la couleur jaune de la figure 12, des hachures apparaissent et représentent le solide « **vide** ». Si nous appliquons la fonction « **Hole** » à un objet, même si celui-ci provient de solides, il ne se passe rien.

Par contre, **si cet objet est groupé** (alors qu'il apparaît avec des hachures) avec un autre, **il va soustraire son volume dans l'autre solide** en créant ainsi un vide. Après quoi la partie qui n'a pas d'intersection avec l'autre solide disparaît, les opérations ultérieures n'ont plus d'effets. Ceci est illustré dans les figures 13 et 14.

Avec ce système d'addition et de soustraction de formes effectué de manière progressive sur un même solide, il est possible de construire des formes très complexes avec précision.

D'un point de vue design, vous devez imaginer comment obtenir une forme

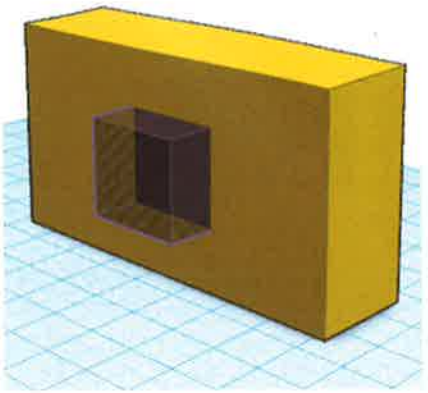


Figure 13 : ici vous pouvez voir la soustraction du solide hachuré.

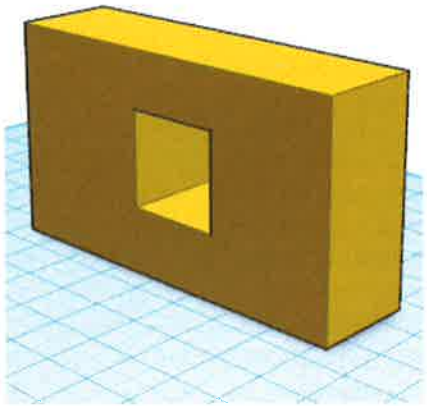


Figure 14 : l'objet principal est maintenant creux au centre après la soustraction du solide hachuré.

spécifique en ajoutant et/ou en soustrayant des solides à d'autres, sans oublier qu'un solide obtenu par soustraction peut à son tour être transformé en un espace vide (« Hole ») et être soustrait à un autre solide. Le raisonnement peut sembler compliqué, mais devient très clair après quelques exercices.

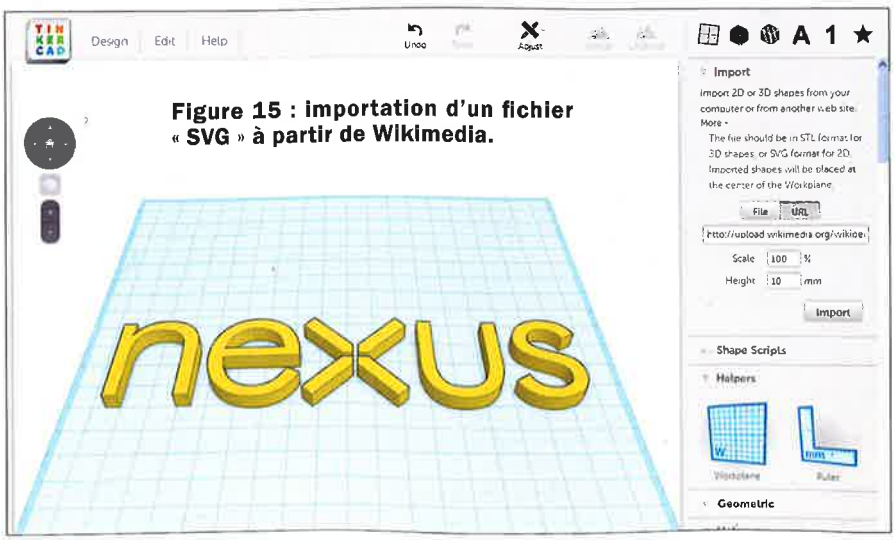


Figure 15 : importation d'un fichier « SVG » à partir de Wikimedia.

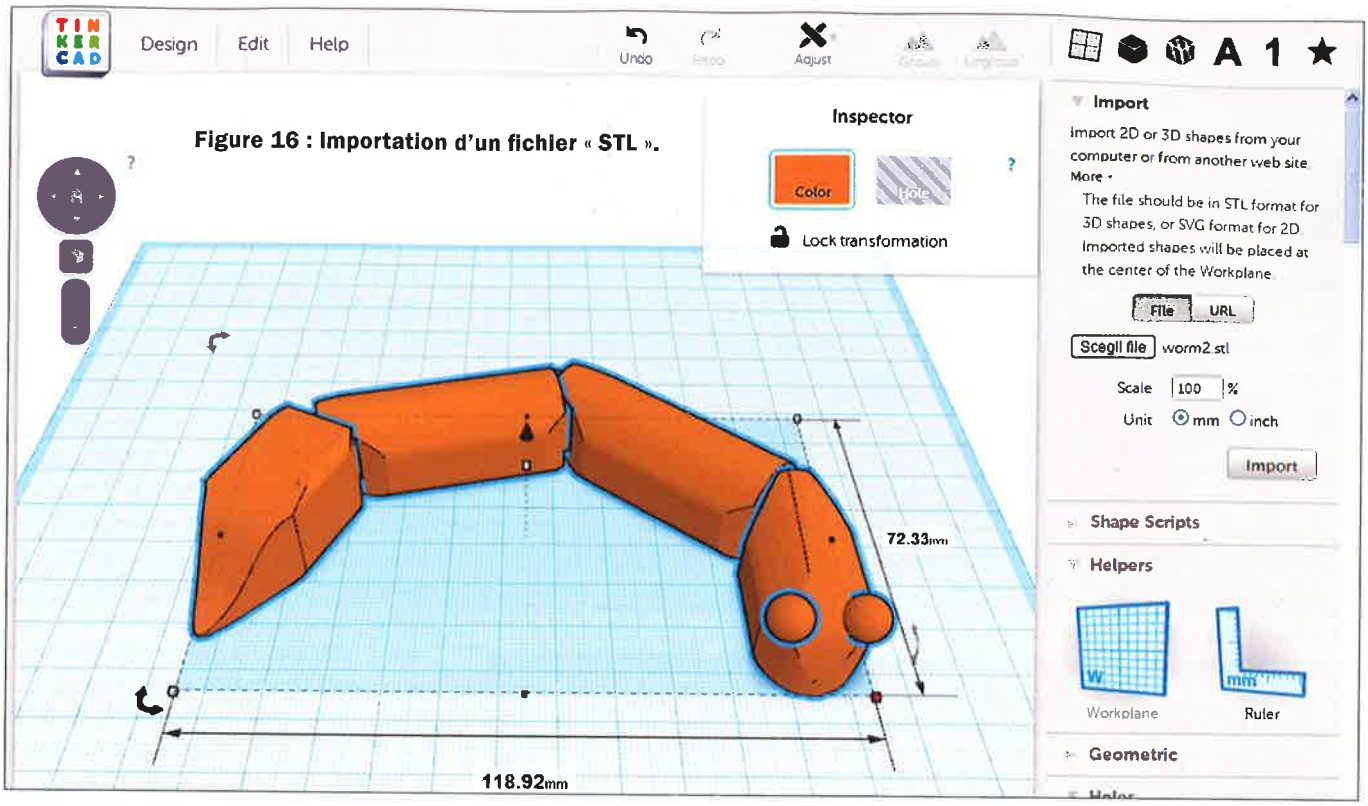


Figure 16 : Importation d'un fichier « STL ».

Importation 2D et 3D

Nous concluons avec une fonctionnalité qui a été ajoutée récemment dans l'application « Tinkercad », c'est la possibilité d'**importer des fichiers en deux et en trois dimensions**. Respectivement ce sont des fichiers de type « **SVG** » et « **STL** ». Un fichier de type « SVG » est un fichier vectoriel bidimensionnel, il est importé comme une structure extrudée de quelques millimètres, mais d'une hauteur uniforme.

Ce type de fichier est utile pour créer des formes complexes comme du texte, des logos, qui peuvent être incluses dans d'autres objets. Son épaisseur peut être modifiée avec les points d'accrochage comme pour tous les autres objets.

Lors de l'importation, il peut y avoir quelques problèmes de compatibilité si vous n'utilisez pas le format « SVG » pur.

Pour les tests, nous nous sommes servis d'un fichier « SVG » à partir du web à l'adresse suivante :
http://upload.wikimedia.org/wikipedia/commons/e/e3/Google_Nexus.svg

et nous l'avons importé, nous obtenons le résultat de la figure 15.

L'importation de fichiers au format « **STL** » est plutôt intéressante pour retoucher et/ou modifier un modèle existant. Avec le système d'addition/soustraction, il est possible d'importer des objets réalisés avec d'autres applications ou téléchargés à partir du web. Un modèle importé au format « **STL** » est géré comme un objet complexe et ne se décompose pas en primitives, cependant vous pouvez ajouter et soustraire des parties comme pour un solide natif de « **Tinkercad** ».

Voici en figure 16, un exemple d'importation de fichier de type « **STL** ».

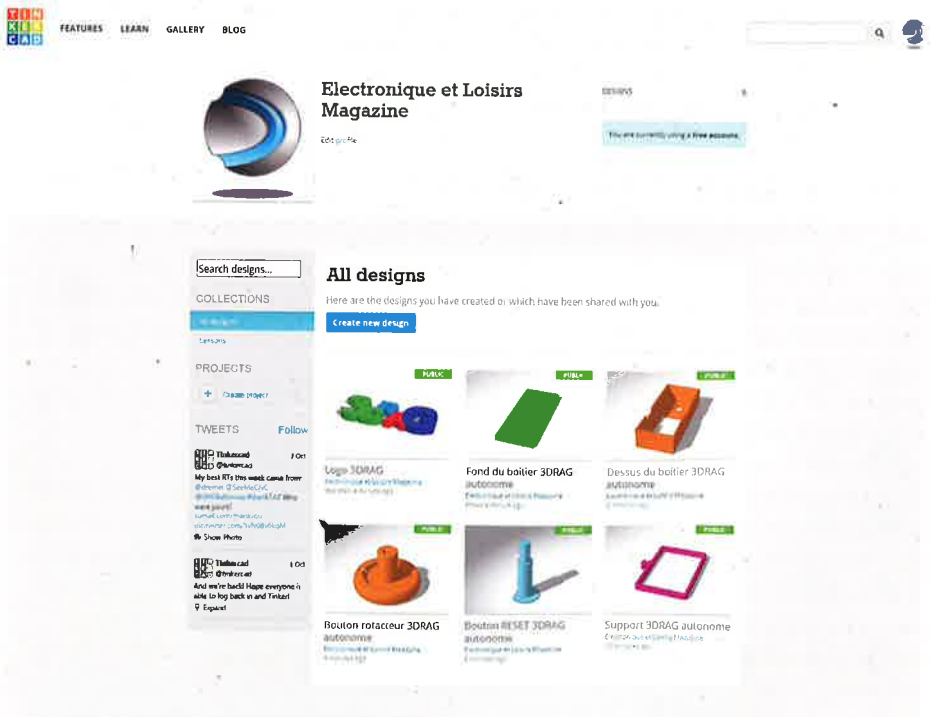


Figure 17 : les différentes pièces du boîtier mises dans le domaine « PUBLIC » de Tinkercad, afin que vous puissiez les copier et les utiliser.

Et maintenant, si nous commencions à construire notre boîtier !

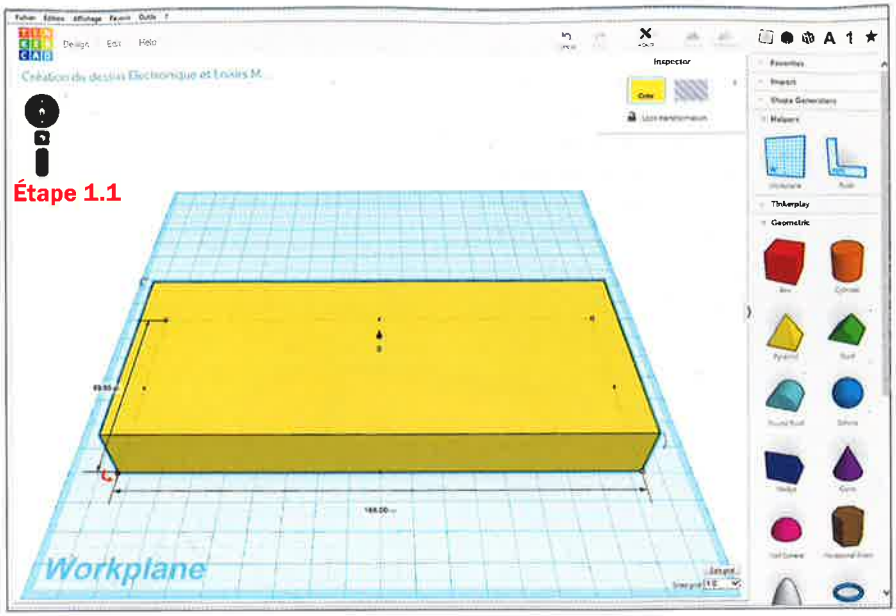
Maintenant que vous avez acquis les fondamentaux de l'application « Tinkercad », vous pouvez vous aventurer à réaliser le boîtier qui protègera le circuit imprimé du montage d'impression autonome. Afin de ne pas occuper trop de pages dans la revue, nous allons seulement vous expliquer la construction de la partie supérieure du boîtier. Il est intéressant de noter que pour fixer les deux parties, nous avons décidé de créer un joint entre le couvercle et la partie inférieure. Les vis de fixation du circuit imprimé seront vissées sur 4 entretoises de 6 mm de Ø avec des trous de 3 mm de Ø. Une armature interne rendra le boîtier plus rigide.

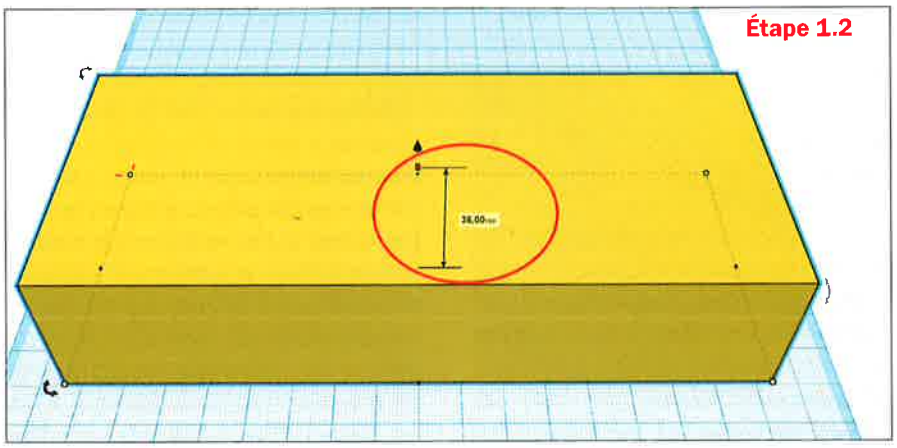
Les parois du couvercle ont une épaisseur de 2 mm et, par conséquent, la découpe à effectuer dans les parois de la base pour créer l'emboîtement doit avoir une hauteur de 3 mm et une épaisseur de 2 mm.

À l'épaisseur de la base, nous devons ajouter 8 mm afin de placer les entretoises des vis, soit un total de 13 mm d'épaisseur pour la base. De cette manière, la matière imprimée appuie sur les entretoises qui seront alignées avec la partie supérieure de la base. Cela simplifie l'usinage de la découpe rectangulaire à travers laquelle nous insérerons et retirerons la carte mémoire SD.

Le bouton rotatif et le poussoir de réinitialisation sont relativement simples à réaliser. Pour maintenir le boitier fixé à l'imprimante, nous avons utilisé un système de fixation avec des profilés en aluminium. Lorsque vous aurez imprimé le boîtier, vous en apprécierez la commodité.

Pour ceux qui veulent mieux comprendre comment nous avons fabriqué les différentes pièces du boîtier, **nous mettons à disposition les modèles dans « Tinkercad »**. Vous pourrez ainsi réutiliser, dissocier ou décomposer les différents éléments pour comprendre les différentes étapes. En figure 17, vous pouvez voir les différents objets que vous devez rechercher dans « Tinkercad ».





Voici la liste :

- Fond du boîtier 3DRAG autonome ;
- Dessus du boîtier 3DRAG autonome ;
- Bouton rotatif 3DRAG autonome ;
- Bouton RESET 3DRAG autonome ;
- Support 3DRAG autonome ;
- Logo 3DRAG (supplément n'intervient pas dans cet article).

Étape 1

Cliquez sur « **New Design** » dans votre espace personnel (que vous aurez créé au préalable), vous obtenez une zone de travail (Workplane) vide. Nous voulons que le boîtier ait un espace de 2 mm tout autour et que les côtés fassent 2 mm d'épaisseur. Cela nous conduit donc à une largeur de 168 mm et une profondeur de 69 mm, alors qu'une hauteur de 36 mm nous permet de positionner l'afficheur LCD à environ 1 mm de la face supérieure. Cliquez sur l'onglet « Geometric » à droite puis sélectionnez le cube (« Box »). Maintenez le bouton gauche de la souris enfoncé et faites glisser le cube vers le centre de la zone de travail.

À l'aide du pointeur de la souris étirez les points d'accrochage noirs (les petits carrés noirs) sur les deux côtés de la base afin d'obtenir les dimensions désirées. Vous pouvez aussi étirer à partir d'un carré blanc, dans ce cas le solide s'agrandit selon les 2 axes (X, Y). Cliquez sur le carré blanc situé sur le dessus au milieu (juste en dessous du triangle noir), maintenez le bouton gauche de la souris enfoncé et déplacez le solide vers le haut pour atteindre 36 mm (voir le cercle rouge sur l'image ci-contre). La cotation s'affiche à côté du pointeur de la souris. Allez dans le panneau « Inspector » et sélectionnez la

couleur jaune dans « Presets » (2ème ligne et 3ème carreau en partant de la gauche). Cliquez sur la couleur jaune, vous voyez le solide changer de couleur. Vérifiez que les mesures sont correctes, pour cela cliquez sur le petit carré blanc à gauche à la base du solide, les cotations s'affichent. De même pour le carré blanc du milieu en dessous du triangle noir. Bougez la roulette de la souris, vous voyez l'objet s'agrandir et se réduire (« zoom+ » et « zoom- »).

Étape 2

Pour créer le vide à l'intérieur du solide précédent (le solide jaune), nous devons préparer un second parallélépipède dont les mesures sont calculées en soustrayant l'épaisseur des parois latérales du boîtier, c'est-à-dire 2 mm de chaque côté, soit 4 mm à chaque mesure. Ainsi, nous obtenons les dimensions suivantes : 164 x 65 mm.

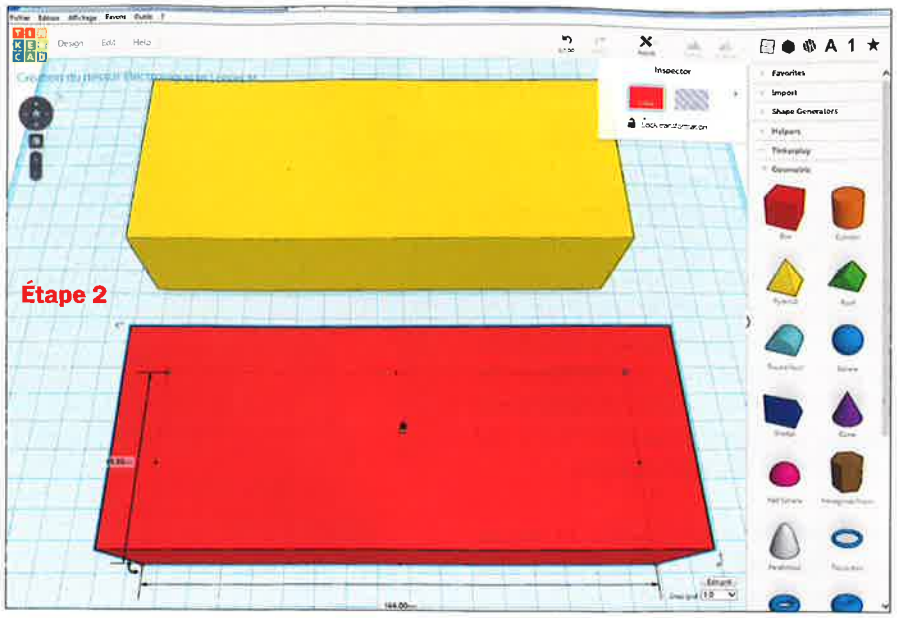
La hauteur du parallélépipède n'est cependant pas un problème, à condition qu'elle soit au moins égale ou supérieure à 36 mm. Au moment de la fusion, nous soulèverons le parallélépipède en le faisant sortir par le haut, cela ne posera pas de problème.

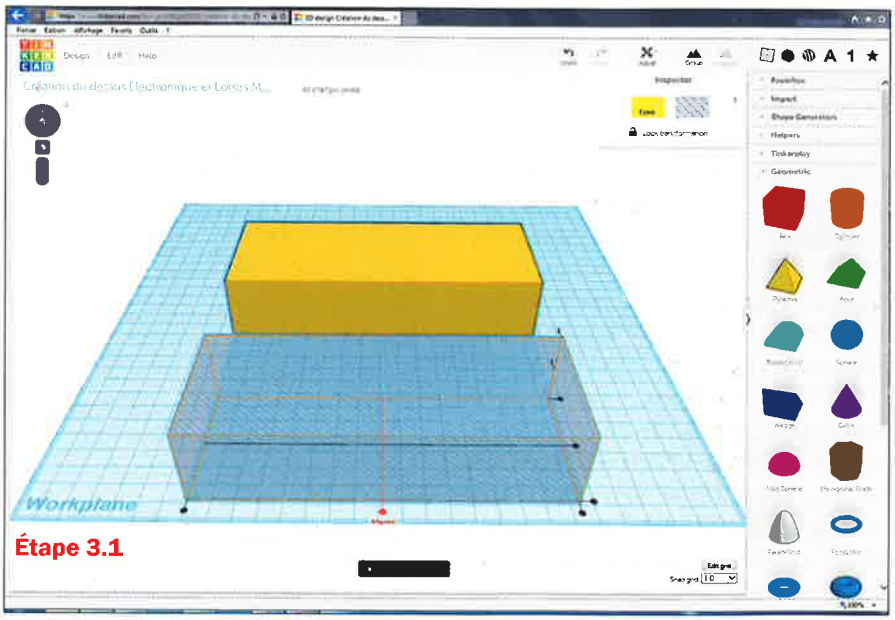
Notez que l'application « Tinkercad » propose un plan de travail dont les dimensions sont supérieures à celles de la 3DRAG, nous sommes ainsi limités à une surface d'impression de 200 mm par 200 mm.

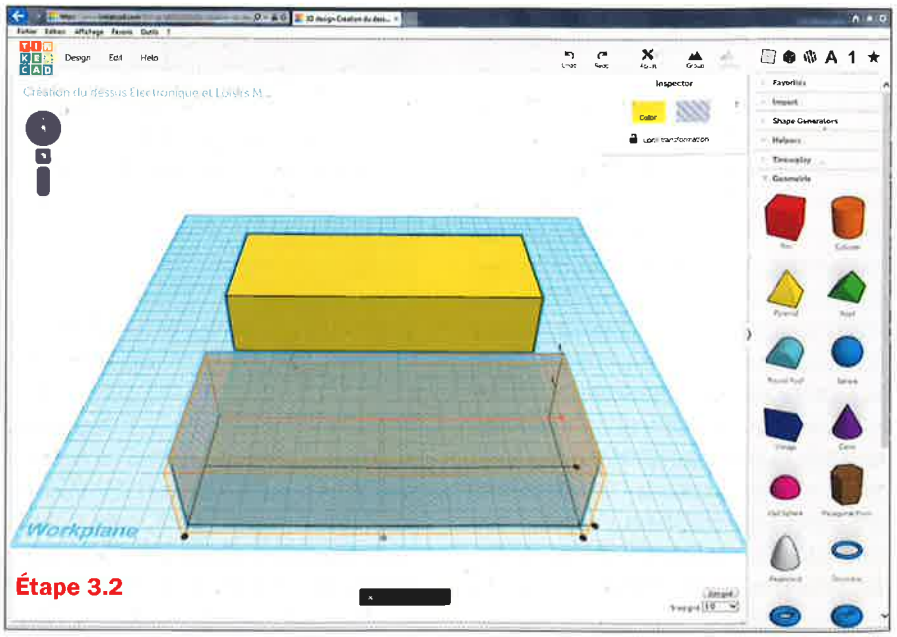
Déplacez les 2 solides pour les adapter à la surface de travail et vérifiez de nouveau les mesures, pour le parallélépipède rouge nous avons 164 mm par 65 mm, comme vous pouvez le voir sur la figure ci-contre. Ne vous préoccupez pas de l'alignement à ce stade, vous pouvez cliquer sur « **Design → Properties** » pour renommer comme vous le souhaitez votre projet.

Étape 3

Pour adapter les 2 solides (ou parallélépipèdes) correctement, nous utilisons l'outil d'alignement. Vous devez d'abord sélectionner le plus petit solide (en rouge) et cliquer sur « **Hole** » (icône hachurée) dans le panneau « **Inspector** ».







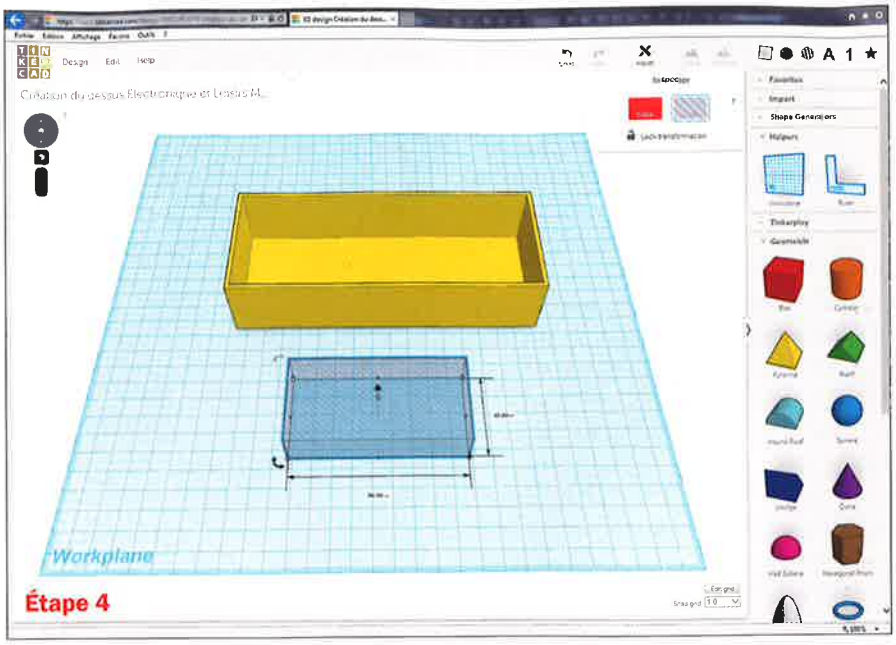
Le solide devient alors transparent et de couleur grise. Sélectionnez les 2 solides, puis cliquez sur « **Adjust** → **Align** » pour afficher les points d'alignement.

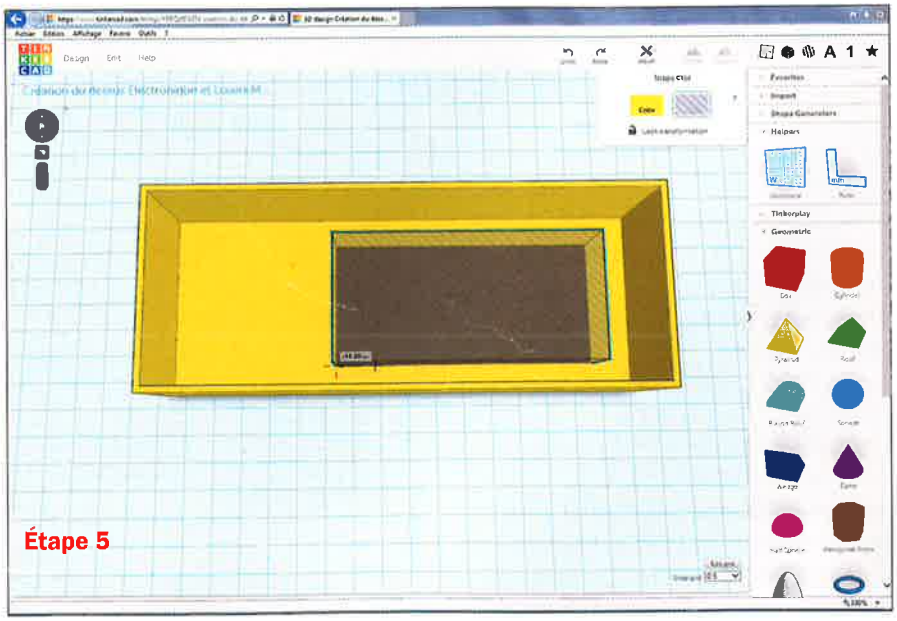
Sélectionnez les points d'alignements correspondant au centrage, celui du milieu en rouge, puis cliquez dessus, le mot « Aligned » apparaît. Faites de même sur celui de droite au fond en rouge. Par contre, lorsque vous cliquez dessus le solide grisé se déplace à l'intérieur du solide jaune. Ils sont donc alignés selon les axes X et Y (voir les figures de la page précédente).

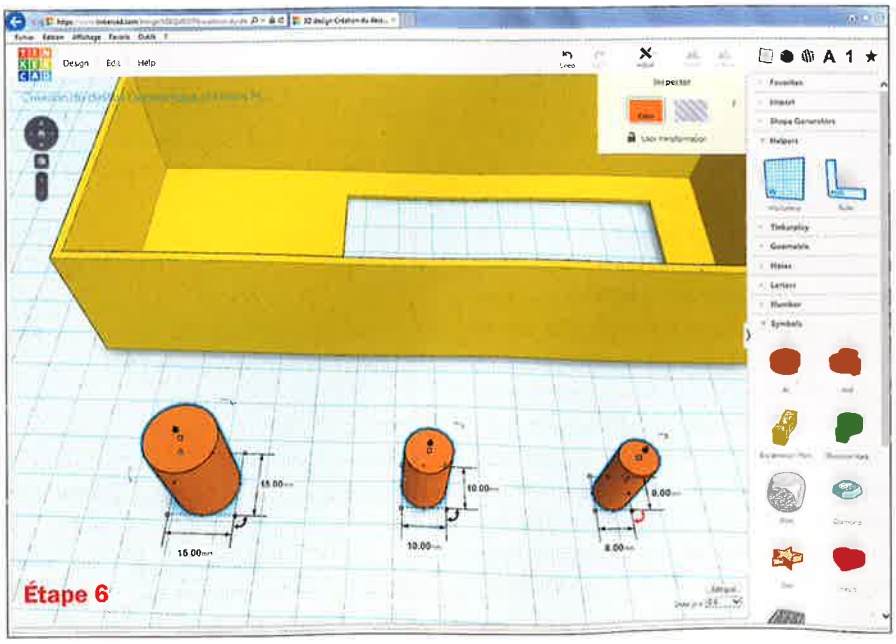
Pour réaliser la face avant du boîtier nous utilisons le parallélépipède le plus petit afin d'obtenir une épaisseur de 1,6 mm par rapport au plan. Pour cela, nous devons d'abord sélectionner 0,1 en bas à droite du menu déroulant « **Snap grid** ». Ensuite sélectionnions le parallélépipède interne en utilisant le point d'accrochage en forme de cône qui se situe au centre de la surface supérieure. Faisons glisser la souris afin de lire la valeur 1,6 mm. Ramenons « **Snap grid** » à 1,0 mm, et sélectionnons les deux objets (qui sont alignés) et cliquons sur « **Group** ».

Étape 4

Faites glisser le couvercle créé à l'étape précédente dans la partie supérieure de l'espace de travail et procédez à la création du parallélépipède « négatif » afin de réaliser la découpe pour l'écran LCD. Les dimensions de l'afficheur sont de 42 mm x 86 mm. Si vous préparez un parallélépipède de 45 mm x 90 mm et que vous le positionnez de manière appropriée, vous pourrez voir correctement les informations affichées par l'écran LCD. Les quelques millimètres supplémentaires du parallélépipède formeront un cadre autour de l'afficheur. Pour réaliser la découpe de l'écran LCD, utilisez la procédure vue précédemment. Faites glisser le cube (« Box ») sur le plan de travail, puis à l'aide des points d'accrochage donnez-lui les dimensions adéquates (45 x 90 mm). Ensuite assignez au solide l'attribut « Hole » dans le panneau « Inspector ». Par soustraction vous obtenez la découpe de l'afficheur. Notez que dans ce cas la hauteur n'a pas d'importance lors de la fusion, car le solide jaune a un côté « ouvert ».







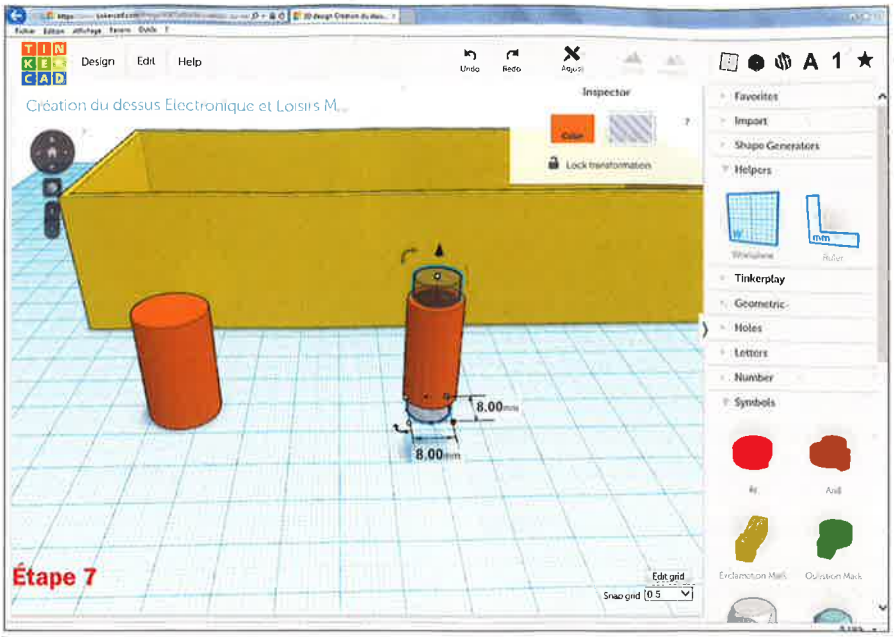
Étape 5

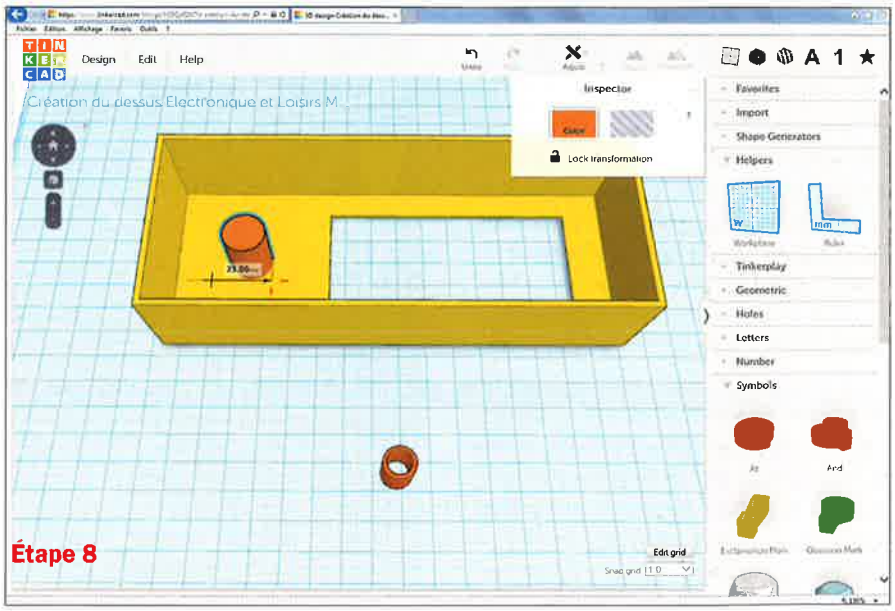
Dans cette étape, vous allez apprendre à utiliser la cotation du système et le verrouillage du déplacement. L'objectif est de positionner avec précision la découpe de l'afficheur dans le solide jaune. Des mesures de l'étape précédente, vous en déduisez que la découpe doit être centrée verticalement, et située à environ 12 mm du bord de la base. De la base à l'extérieur du boîtier, il y a environ 3,5 mm, en positionnant la découpe à 14 mm par rapport à l'extérieur du boîtier et en tenant compte de l'épaisseur des parois de 2 mm, nous obtenons environ ½ mm de tolérance. Pour positionner la découpe vous devez utiliser correctement l'outil d'alignement, pour cela vous devez centrer la découpe et le couvercle sur l'axe Y. Utilisez les points d'accrochage qui se trouvent à droite du solide. La découpe est alors centrée verticalement. Maintenant sélectionnez le solide grisé, commencez à le déplacer vers la gauche et appuyez en même temps sur la touche « **Shift** » pour déplacer le solide selon un seul axe. Arrêtez lorsque vous lisez exactement un déplacement de -14 mm (voir l'image ci-contre). Sélectionnez les deux solides et cliquez sur « **Group** ».

Étape 6

Maintenant vous devez percer deux trous, un pour l'axe du commutateur et l'autre pour le bouton de réinitialisation. Celui du commutateur a un Ø de 15 mm, tandis que pour le RESET il faut créer un cylindre creux ayant 2 diamètres différents et qui viendra appuyer contre l'extrémité du poussoir. Dans l'image ci-contre vous apercevez les 3 cylindres que vous devez créer durant cette étape. La hauteur pour celui de 15 mm n'a pas d'importance car c'est un trou, tandis que pour le cylindre de 10 mm il faut une hauteur de 18 mm. La partie du cylindre de 8 mm de Ø devra avoir au minimum une hauteur de 18 mm.

Pour bien comprendre le cheminement de la construction, nous avons considéré que le passage du bouton de réinitialisation de la face avant sera aligné sur le bouton poussoir, celui-ci ayant une hauteur de 5 mm seulement





(il faut tenir compte du fait que le bouton est soudé directement sur le circuit et donc qu'il se trouve beaucoup plus bas que la face de boîtier). En fait la distance entre le bouton et la face avant est d'environ 30 mm, il nous faut donc coller une pièce qui actionnera le poussoir lors de la pression. Nous devons créer un cylindre de 35 mm de haut qui servira de piston afin qu'il puisse actionner le poussoir.

Cependant le piston devra rester aligné pendant toute sa course, nous devons donc créer un cylindre creux qui guidera le piston pendant le mouvement. Pour éviter que le piston sorte de son logement, son Ø inférieur doit être de 7 mm et son Ø supérieur de 4 mm.

Étape 7

Pour obtenir le résultat de la figure ci-contre, vous devez attribuer la fonction « Hole » au cylindre de 8 mm, puis l'aligner au centre par rapport aux axes X et Y avec le cylindre de 10 mm, et enfin les regrouper. Vous obtenez comme résultat un cylindre creux dont la paroi est positionnée à 1 mm de la partie interne du couvercle du boîtier. Cependant il manque encore le cylindre avec lequel vous devez faire le trou dans le panneau frontal. Celui-ci doit être d'au moins 5 mm de sorte que le piston de 4 mm passe du premier coup. Si cela n'est pas le cas lorsque l'impression est terminée, vous pouvez ajuster le trou à l'aide d'une

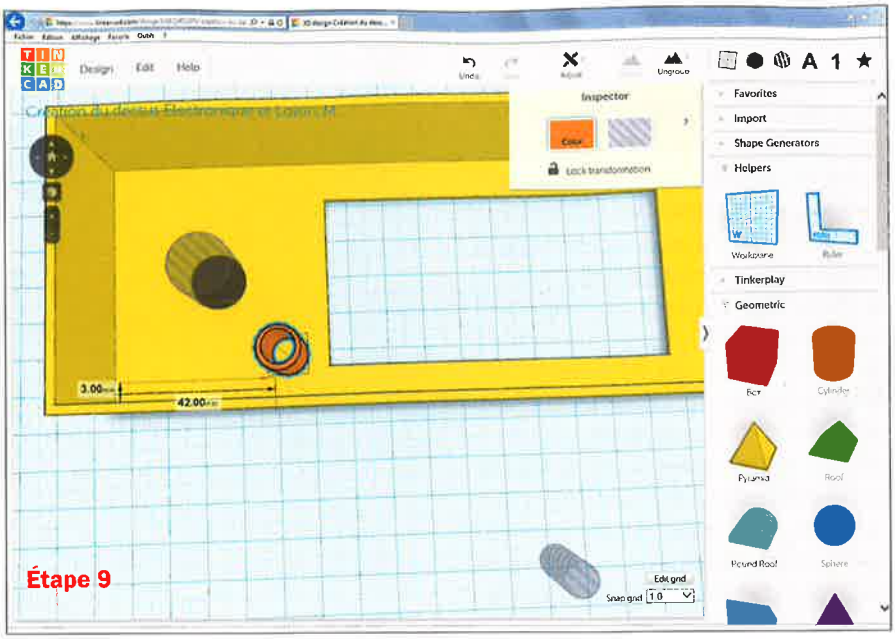
petite lime ronde pour éliminer l'excès typique de matériau dû à la couche d'impression initiale. Malheureusement, il n'est pas possible de grouper un solide avec un autre vide (« Hole ») car le volume deviendrait négatif, il ne reste que la partie « pleine ». Pour créer le cylindre de guidage et le trou vous devez les séparer. Positionnez le 1er puis alignez le 2ème.

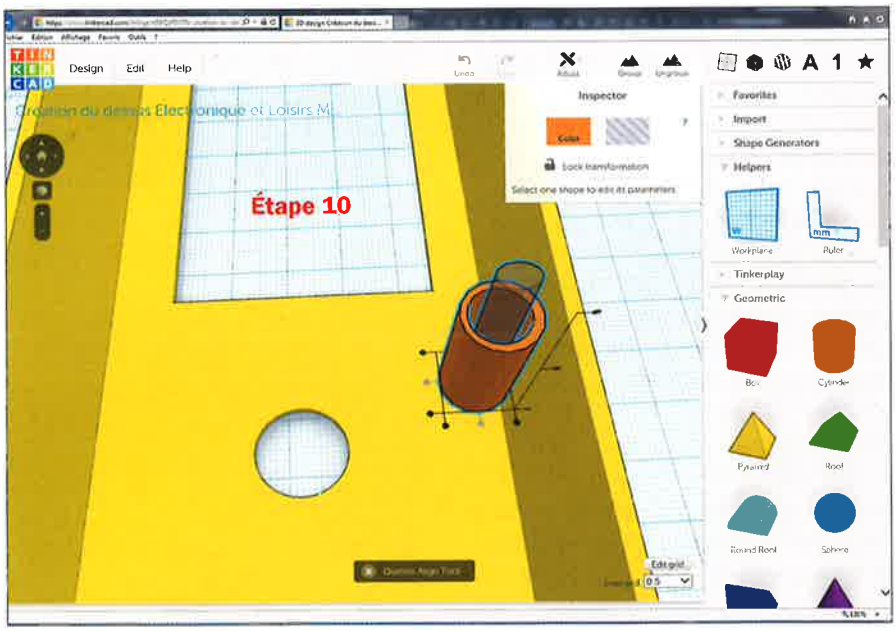
Étape 8

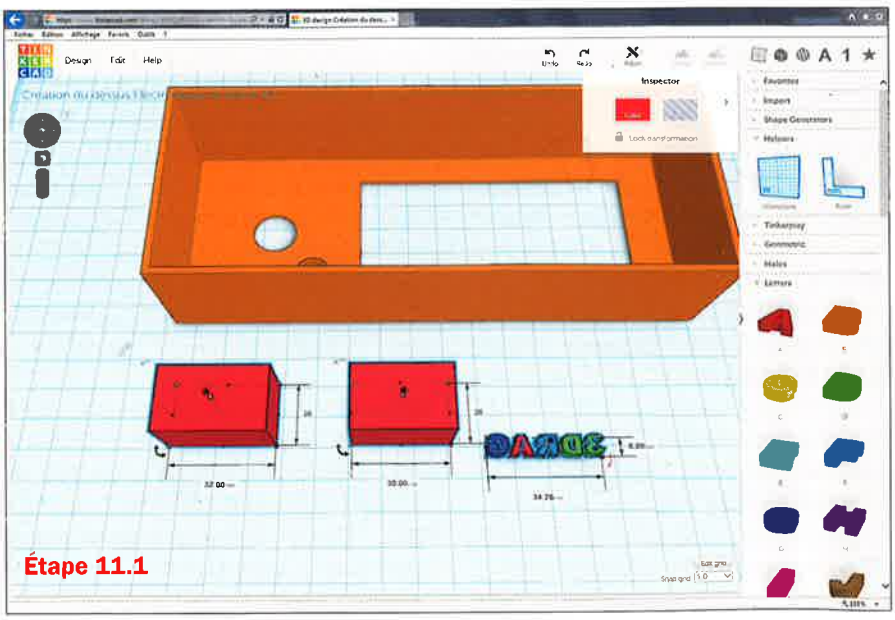
Le trou de l'encodeur rotatif doit également être positionné correctement, même si le bouton couvrira le trou laissant passer l'axe et cachera les éventuels défauts d'alignement. Encore une fois, nous avons pris nos mesures et nous savons que l'axe de l'encodeur est centré verticalement, il se situe à environ 26 mm du bord du circuit imprimé. Ajoutons 2 mm pour le bord, 2 mm pour l'épaisseur de la paroi et soustrayons 7,5 mm car les mesures sont prises par rapport au bord du cylindre et non par rapport au centre. Nous obtenons 22,5 mm que nous arrondissons à 23 mm, cela correspond à la distance du bord du cylindre par rapport à la paroi gauche. Comme nous l'avons fait pour la découpe de l'afficheur LCD, nous sélectionnons le couvercle et le cylindre, à l'aide de la fonction « **Align** » nous positions le cylindre à gauche et nous l'alignons verticalement par rapport au centre. Ensuite nous sélectionnons uniquement le cylindre et en utilisant la touche « **SHIFT** » nous bloquons le déplacement selon l'axe des X. Nous faisons alors glisser vers la droite jusqu'à lire 23 mm sur la cotation qui apparaît lors du déplacement. Maintenant, vous pouvez affecter la fonction « **Hole** » au cylindre et, si vous le souhaitez, faire un « **Group** » avec le couvercle.

Étape 9

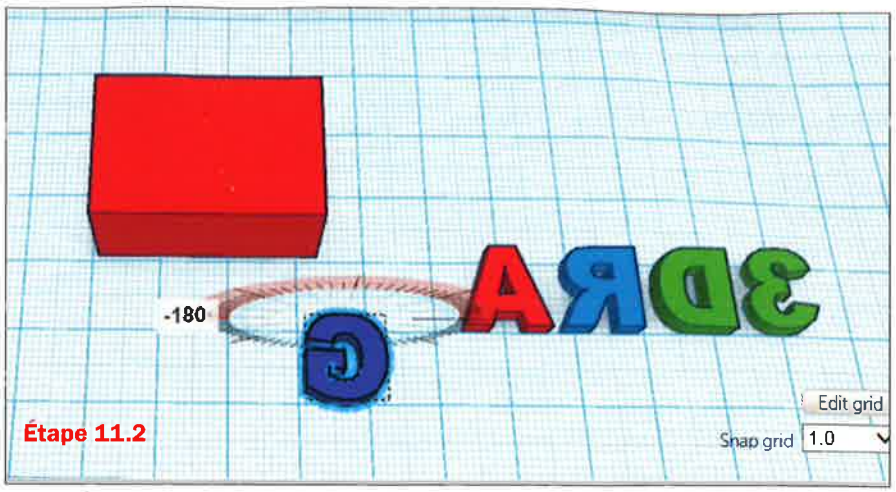
Le positionnement du cylindre qui abrite le piston du bouton de réinitialisation nécessite un peu d'attention. Cependant l'application « Tinkercad » est en mesure de faciliter également les opérations de précision. Le système est basé sur l'alignement, et les renseignements sur le déplacement de l'objet sont calculés par rapport au point de départ du déplacement.







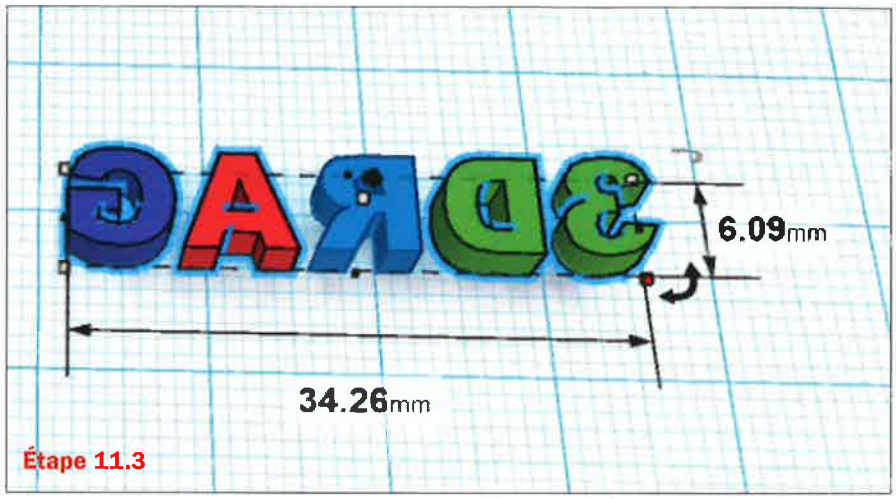
Si tout en déplaçant l'objet vous lâchez malheureusement le bouton de la souris, le point de départ a été modifié et se situe désormais à l'endroit où vous avez lâché accidentellement l'objet, vous devez donc recommencer. La fonction « **Undo** » dans ce cas est très utile. Sur l'image ci-contre, vous voyez que le cylindre creux est positionné à 42 mm du bord de la paroi gauche du boîtier et à 3 mm de la base. Dans la pratique il reste 1 mm d'espace entre la paroi et le cylindre. Notez que nous avons laissé le trou dans sa position précédente. Amenons le cylindre dans la partie inférieure gauche du boîtier et utilisons la fonction « **Align** » pour le positionner correctement dans l'angle, ensuite déplaçons le cylindre jusqu'à lire les coordonnées indiquées (42 mm, 3 mm).



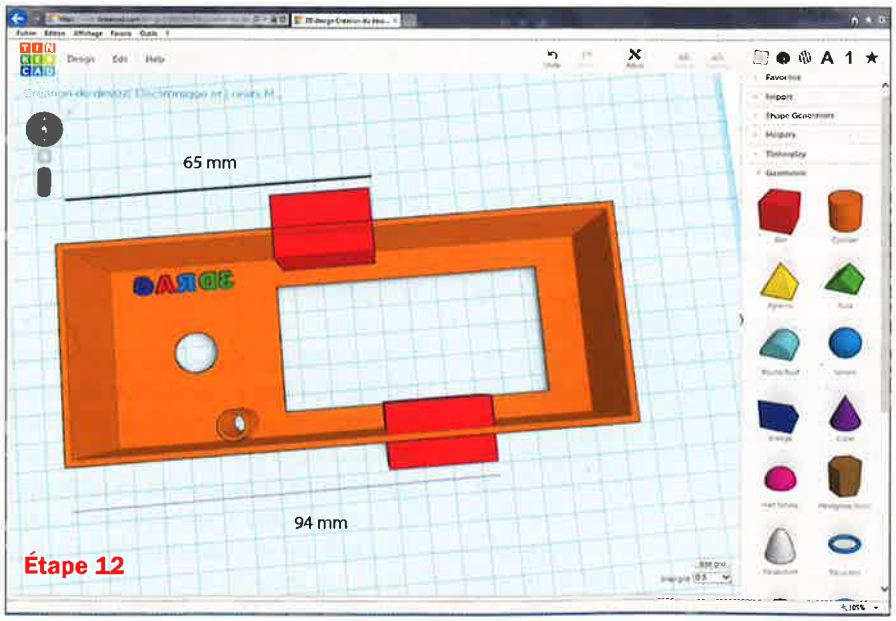
Étape 11.2

Étape 10

Nous devons réaliser le trou de 5 mm dans la face avant, sachant que le cylindre est déjà positionné de manière correcte. Il reste donc à aligner le « trou » (cylindre grisé) avec le centre du cylindre orange servant de guide. Si vous avez essayé de cliquer sur les points ronds de la fonction « **Align** », vous aurez remarqué qu'il y a un objet qui reste immobile parmi les deux à aligner. Il peut arriver de déplacer le mauvais solide. Le rectangle qui apparaît entre les deux objets sélectionnés pour l'alignement montre l'endroit où les objets sont pris en charge. Dans le cas de l'alignement cela se situe sur les côtés extérieurs, tandis que pour le centrage, les deux objets sont déplacés, à moins que l'un des 2 solides ne soit pas à l'intérieur de la zone de l'autre. Dans notre cas, nous déplaçons donc le « trou » en premier à l'intérieur du cylindre et seulement après nous activons la fonction « **Align** ». De cette façon le cylindre de guidage ne se déplace pas et le « trou » (cylindre grisé) peut être centré sans problème. Une fois le centrage effectué, vous sélectionnez le « trou » (cylindre grisé), le cylindre orange (le guide) et le couvercle, ensuite vous cliquez sur « **Group** ».



Étape 11.3



Étape 12

Étape 11

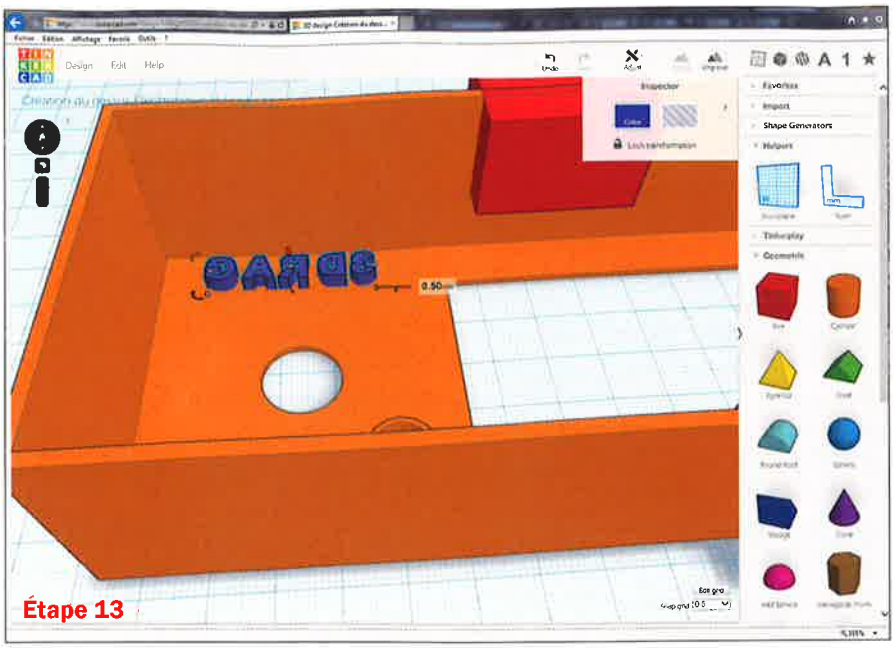
Les trous de la face avant sont maintenant tous réalisés, nous devons préparer les deux encoches qui correspondent à l'insertion de la carte SD et au passage du câble plat provenant de la carte contrôleur de la 3DRAG. Pour simplifier le montage et obtenir un bon résultat, nous allons créer des trous au bord de la paroi.

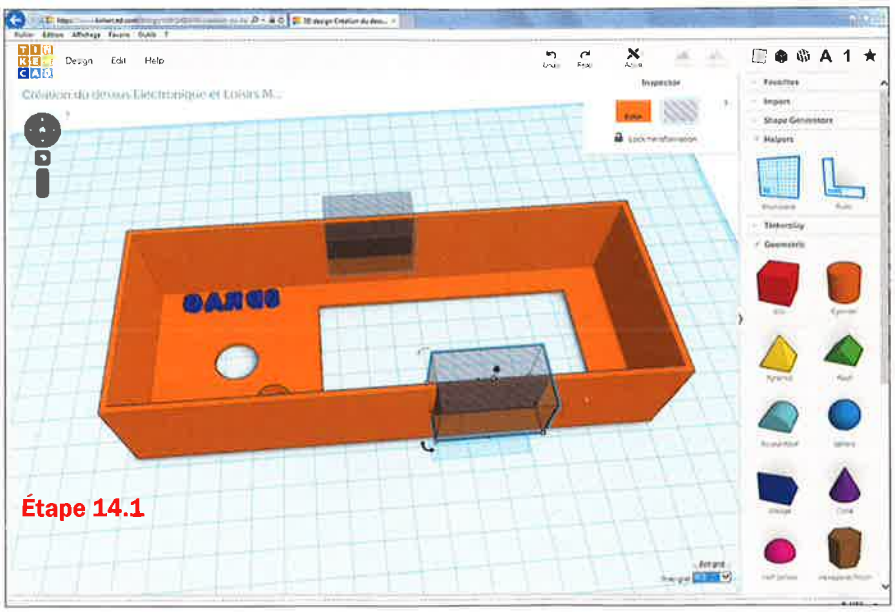
La carte mémoire a besoin d'au moins 25 mm, tandis que le câble plat de 20 conducteurs a une largeur de 26 mm. Ajoutons un peu de marge sur les deux côtés, nous obtenons ainsi 32 mm pour le câble plat et 30 mm pour la carte SD.

L'épaisseur de l'encoche de 32 mm (câble) doit être de 9 mm, tandis que celle de 30 mm (carte SD) doit avoir une épaisseur de 4 mm. Pendant que nous y sommes, nous créons aussi un logo « 3DRAG » que nous positionnons sur la face avant au-dessus du bouton de l'encodeur.

Dans le menu « Geometric » cliquez sur le cube rouge (Box) puis faites le glisser sur le plan de travail. Ensuite attribuez-lui les dimensions 32 mm par 20 mm. Faites de même avec le 2ème cube, cependant les dimensions sont de 30 mm par 20 mm. Pour créer le logo, allez dans l'onglet « Number » à droite, cliquez sur le chiffre « **3** » et faites le glisser sur le plan de travail. Ensuite allez dans l'onglet « Letters » et faites de même pour les lettres. Il faut sélectionner et faire glisser les lettres : « **D** », « **R** », « **A** », « **G** ». Cependant vous remarquerez que le chiffre et toutes les lettres doivent être « inversés » (effet miroir) pour que le logo soit imprimé dans le bon sens sur la face avant du boîtier. Pour cela il faut effectuer une rotation de 180° pour tous les caractères selon l'axe Y.

Lorsque vous sélectionnez à l'aide de la souris un caractère, vous devez apercevoir une flèche noire foncée et une autre flèche grisée. Cliquez sur cette dernière qui se situe en haut à gauche de la lettre « **G** » par exemple (voir l'image de la lettre « G » ci-avant) et effectuez une rotation vers la gauche jusqu'à ce que la valeur de l'angle atteigne « **-180°** ». Faites de même pour tous les autres caractères. Ensuite à l'aide de la touche « **SHIFT** » sélectionnez tous les caractères et alignez-les. Ajustez les dimensions de l'ensemble pour obtenir un logo de 34 mm par 6 mm. Notez que dans notre exemple les dimensions du logo sont de 34,26 mm par 6,09 mm, cela n'a pas d'importance mais nous voulons souligner le fait qu'il est parfois difficile d'obtenir des cotations exactes à « main levée » avec la souris. Notez aussi qu'au lieu d'effectuer un effet miroir sur chaque caractère, il est possible de positionner tous les caractères sur le plan de travail, de les aligner, de les sélectionner avec la touche « **SHIFT** » et enfin d'appliquer l'effet miroir sur l'ensemble selon l'axe Y.


Étape 13


Étape 14.1

Étape 12

Pour positionner correctement les deux encoches à leur place, utilisons les mesures du circuit imprimé. L'encoche pour le câble est pratiquement alignée sur le côté gauche de l'afficheur LCD, tandis que l'encoche de la carte mémoire SD est positionnée vers le milieu. La découpe de 30 mm doit être à 65 mm du côté gauche, tandis que celle de la carte mémoire doit être à 94 mm du bord. Alignez-les sur le côté gauche ainsi que sur le bord supérieur, de sorte que les découpes créent des espaces en forme de fentes par rapport aux arêtes du boîtier. L'alignement et les dimensions par rapport à l'axe des Y de ces 2 parallélépipèdes n'ont pas d'importance car ils seront soustraits des parois.

Dans cette étape, positionnons aussi le logo vers le bord supérieur. Laissez environ 3 mm entre le logo et l'intérieur du couvercle, en se souvenant que le bouton a un Ø de 38 mm et que sa distance par rapport au centre du trou est de 19 mm. Si le texte est trop petit, il sera caché par le bouton.

Etape 13

Il est temps de s'occuper du logo « **3DRAG** » que nous avons créé à l'étape 11. Le logo a été « inversé » (effet miroir) pour qu'il puisse apparaître dans le bon sens lors de l'impression.

L'idée de base est que nous devons créer quelque chose de différent mais toujours facile à imprimer. Il faut que le logo apparaisse sous la forme d'une inscription sur la face avant sans traverser celle-ci (sinon certaines parties des lettres ne pourraient pas exister, par exemple l'intérieur du « D » et du « A »). Si nous les avions « creusées » (imprimées) de l'extérieur vers l'intérieur du couvercle, l'imprimante aurait dû « fermer » les caractères par-dessus, vu que la partie frontale extérieure est en contact avec le plateau d'impression. Nous avons plutôt soulevé de 2 couches l'impression du logo, afin que la transparence des couches permette de le voir (par transparence).

En faisant le calcul, si nous soulevons de 0,5 mm le logo, nous obtenons l'impression de deux couches complètes avant de commencer l'extrusion du logo. Pour rehausser le logo, modifions le pas de la grille à 0,5 (« Snap grid »), puis avec le point en forme de cône rouge juste au-dessus de la lettre « R » (voir l'image ci-contre) déplacez le logo de 0,5 mm vers le haut. Vérifiez également que le logo est centré horizontalement par rapport au centre du trou de l'encodeur.

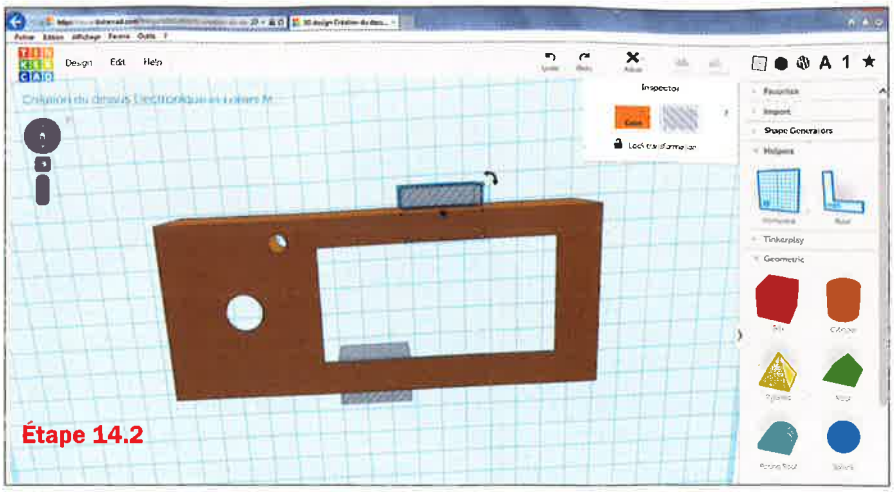
Étape 14

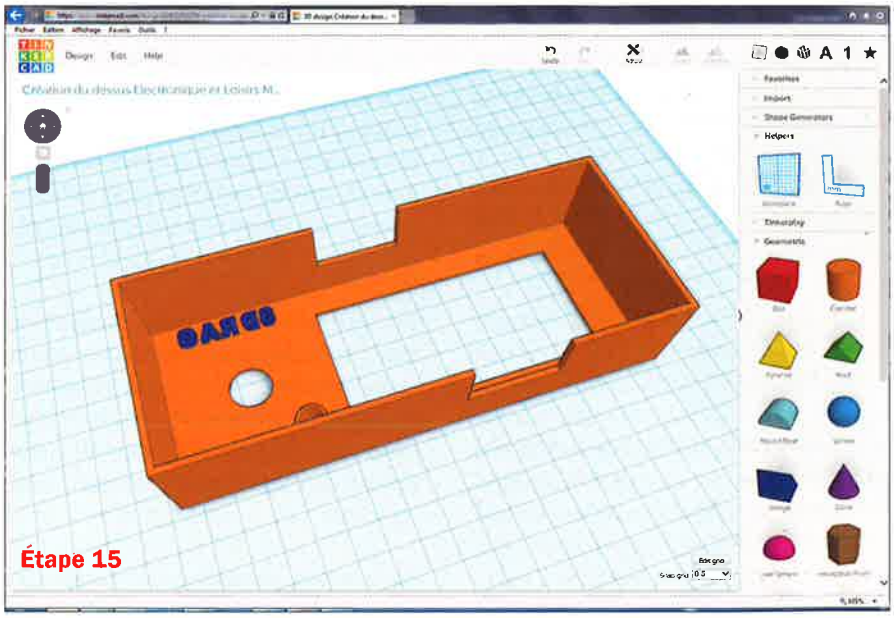
Nous sommes presque arrivés à la fin et nous devons incorporer les derniers solides « creux » (vides) dans la structure du couvercle. Cliquez sur chacun des solides et assignez la fonction « **Hole** » à chacun d'eux dans le panneau « **Inspector** ». Sélectionnez ensuite le couvercle et les 2 solides grisés, appliquez la fonction « **Group** » à l'ensemble. Avec la souris, vous pouvez examiner le résultat final et observer comment l'alignement des côtés des deux solides avec la fonction « **Align** » permet d'avoir toujours des découpes parfaites entre une partie pleine et vide, ce qui est toujours le cas avec les applications CAO. Pour visualiser la face avant du boîtier, il suffit de déplacer le plan de travail (voir la 2ème image ci-contre).

Étape 15

Le travail est terminé et la face avant repose sur le « sol » (plan de travail). Cela nous permet de réduire de façon importante les difficultés d'impression, car il n'y a pas d'inclinaison particulière ou des supports. La seule difficulté réside dans le fait que la surface est assez imposante et que l'objet sera soumis à des contraintes lors de la séparation du plateau (risque de casse). Cliquez sur l'onglet « Design » (en haut à gauche) puis choisissez « Download for 3D printing » pour afficher la fenêtre qui vous montre les différents formats.

Cliquez sur « .STL » et enregistrez le fichier sur votre ordinateur. Préparez ensuite l'imprimante pour ce nouveau travail. Vous pouvez télécharger à partir de notre site web ou sur www.tinkercad.com les différentes pièces constituant le boîtier comme indiqué plus haut dans l'article.





Pour effectuer une impression sans problème, nous vous suggérons d'im-

Image 1. : les différentes pièces constituant le boîtier.

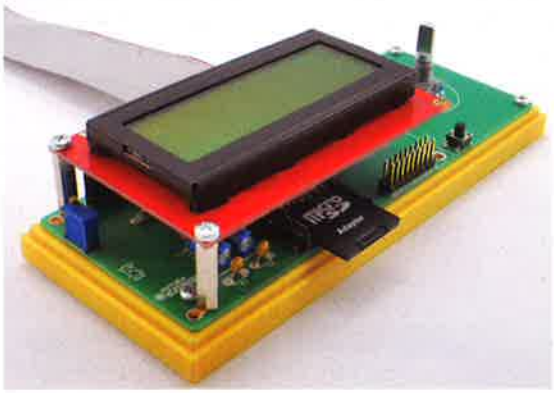
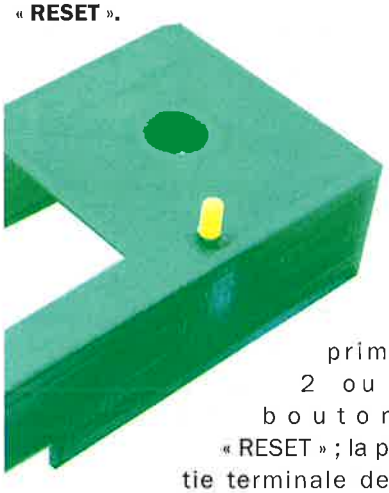


Image 2. : remarquez les quelques millimètres d'espacement entre le circuit imprimé et le fond du boîtier.

Image 3. : ici le piston du bouton « RESET ».



primer 2 ou 3 boutons « RESET » ; la partie terminale de 4 mm est trop petite pour pouvoir être imprimée toute seule sans déformation, si l'impression est effectuée sur deux ou trois boutons à chaque couche, le PLA a le temps de se refroidir et de maintenir la forme.

Assemblons la carte dans le boîtier

Avec la réalisation de la carte électronique dans le précédent numéro, la modélisation du boîtier grâce à l'application « Tinkercad » et l'impression des différentes pièces, nous avons tout ce dont nous avons besoin pour rendre la 3DRAG autonome. Pour compléter l'assemblage, nous utilisons seulement 4 vis autotaraudeuses de 3,5 mm avec un filetage compris entre 6 mm et 8 mm. Dans l'image « 1. », vous pouvez voir le fond du boîtier, le piston du bouton « RESET » et les 2 cadres de fixations en jaune.

Image 4. : ici le bouton de l'encodeur correctement inséré.

Nous avons imprimé en rouge le bouton du sélecteur et en vert la partie supérieure du boîtier (couvercle) comprenant les différentes découpes et les divers trous. La première étape consiste à fixer le circuit imprimé sur la partie correspondante au fond du boîtier, en utilisant les 4 vis autotaraudeuses. Celles-ci sont suffisamment longues pour maintenir le circuit imprimé, mais elles ne traversent pas le fond.

Du côté de l'encodeur, les trous sont en correspondance avec les angles du circuit imprimé tandis que l'autre côté, les trous des entretoises supportant l'afficheur LCD sont positionnés à l'intérieur par rapport à ceux du circuit imprimé.

Comme vous pouvez le remarquer sur l'image « 2. », il y a quelques millimètres d'espacement entre le circuit imprimé et le fond du boîtier, tandis que l'afficheur arrive à fleur du couvercle.

Sur l'image nous avons déjà connecté le câble plat, alors qu'en réalité ce sera la dernière chose à faire avant d'emboîter le fond avec le couvercle.

L'étape suivante concerne le bouton de réinitialisation et le couvercle. Sachant que l'impression génère sur la 1ère couche un écrasement, vous devrez peut-être ajuster avec une petite lime ronde le trou. Cependant le piston peut avoir des bavures qui doivent être éliminées avec une lime à ongle. Prenez le couvercle et le piston et essayez de faire coulisser ce dernier sans frottements dans son logement. Sur l'image « 3. » vous apercevez le piston sortir de son trou.

Mettez de côté le piston du bouton de réinitialisation, passez maintenant à l'adaptation du bouton de l'encodeur. C'est une opération potentiellement délicate, mais inévitable. Selon les paramètres de « slicing » (tranchage) et de vitesse, vous pourriez avoir un trou trop grand ou trop petit.

Dans le modèle que nous avons imprimé, le trou du bouton où l'axe de l'encodeur s'emboîte était plus grand de quelques dixièmes de millimètres par rapport à la taille nominale, cependant il y a de grandes chances que l'axe de l'encodeur n'entre pas dans le trou prévu.

La solution à ce problème, si l'axe n'entre pas dans le logement du bouton,

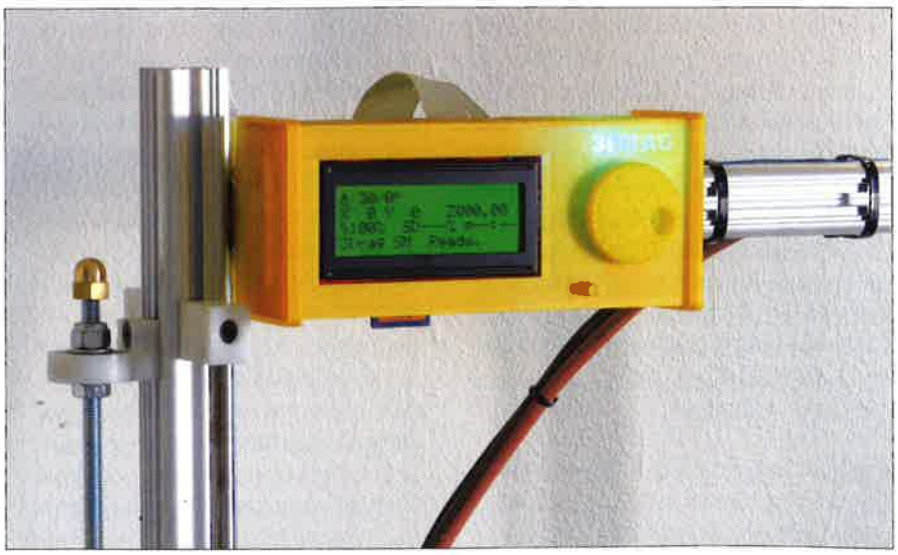


Image 5. : l'emplacement idéal pour fixer le boîtier se situe dans le coin supérieur gauche du côté opposé à celui de la bobine de PLA.

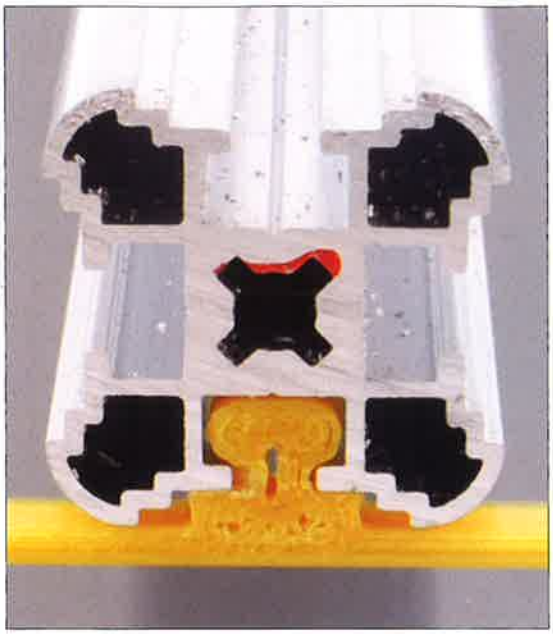


Image 6. : voici en détail la fixation des cadres sur le profilé en aluminium.

est de chauffer légèrement l'axe avec un briquet puis de le pousser dans le logement du bouton avec un mouvement de rotation de manière à agrandir le trou. Soyez prudent de ne pas pousser trop loin l'axe car il peut être difficile de retirer le bouton en cas de démontage.

Avancez de quelques millimètres à chaque fois, chauffez de nouveau légèrement l'axe de l'encodeur et répétez la procédure jusqu'à ce que vous obteniez le résultat de l'image « 4. ». Le bouton ne doit pas avoir de jeu par rapport à l'axe car nous devons pouvoir le presser pour effectuer un clic. Le bouton ne doit pas non plus être trop enfoncé sur l'axe, car il toucherait le couvercle lors du clic et le « contact mécanique » de l'encodeur ne s'enclencherait pas.

L'étape suivante concerne la fixation du câble plat sur la structure de la 3DRAG : un côté va sur la carte contrôleur et l'autre vers l'emplacement prévu dans le boîtier.

L'endroit idéal, où nous avons fixé notre boîtier, se situe dans le coin supérieur gauche du côté opposé à celui de la bobine de PLA, comme le montre l'image « 5. ».

Avec environ un mètre de câble plat, vous pouvez connecter le boitier à la carte contrôleur tout en le dissimulant derrière le profilé en aluminium. Fixez le câble en disposant des colliers en nylon le long du profilé

La position du boîtier étant déterminée, nous pouvons fixer les deux cadres de fixation dans le profilé en aluminium. Si nécessaire utilisez une lime pour ajuster les cadres au profilé. Soyez minutieux, ne forcez pas car les cadres sont fragiles.

Pour assembler chaque cadre, vous devez l'insérer horizontalement puis le tourner à 90 ° sans forcer. Si après la rotation initiale le cadre ne s'insère pas, utilisez une lime à ongle pour l'ajuster. Sur l'image « 6. » vous pouvez voir en détail la fixation des cadres sur le profilé en aluminium.

Avec les deux cadres fixés dans les profilés en aluminium, nous pouvons enfin fermer le boîtier. Après avoir inséré le câble plat de manière correcte dans la carte contrôleur de votre imprimante, vous devez insérer le piston du bouton de réinitialisation, puis enclencher le couvercle sur le fond du boîtier en exerçant une légère pression.

Si les deux parties ont du mal à rester emboîtées, essayez avec les angles d'un même côté. Eventuellement donnez un coup de lime sur les arêtes du fond et à l'intérieur des angles du couvercle (là où les points du fond et du couvercle s'emboîtent).

Rappelez-vous que le câble plat doit sortir de son encoche, pliez-le de manière appropriée vers la bonne direction.

Une fois le boîtier fermé, vous pouvez le fixer aux 2 cadres de maintien sans exercer une trop forte pression. Si vous rencontrez des difficultés, vérifiez qu'il n'y a pas de bavures à l'intérieur des cadres. Vous devriez obtenir le résultat de l'image « 7. » lorsque tout est en place.

Conclusion

À travers cet article, nous vous avons expliqué les bases de l'utilisation de l'application « Tinkercad » qui est en fait un logiciel de modélisation 3D sur Internet, très facile d'emploi. Il vous permet de concevoir très rapidement des modèles à partir de formes géométriques simples. Il offre aussi la possibilité d'importer/d'exporter des fichiers STL et SVG.

Maintenant vous disposez des connaissances de base pour créer n'importe quel objet avec Tinkercad, et notamment des boîtiers sur mesure qui protègeront vos montages électroniques.

Pour plus de renseignements sur les possibilités de la 3DRAG, consultez notre site à l'adresse suivante : **www.3dprint.electroniquemagazine.com** ■

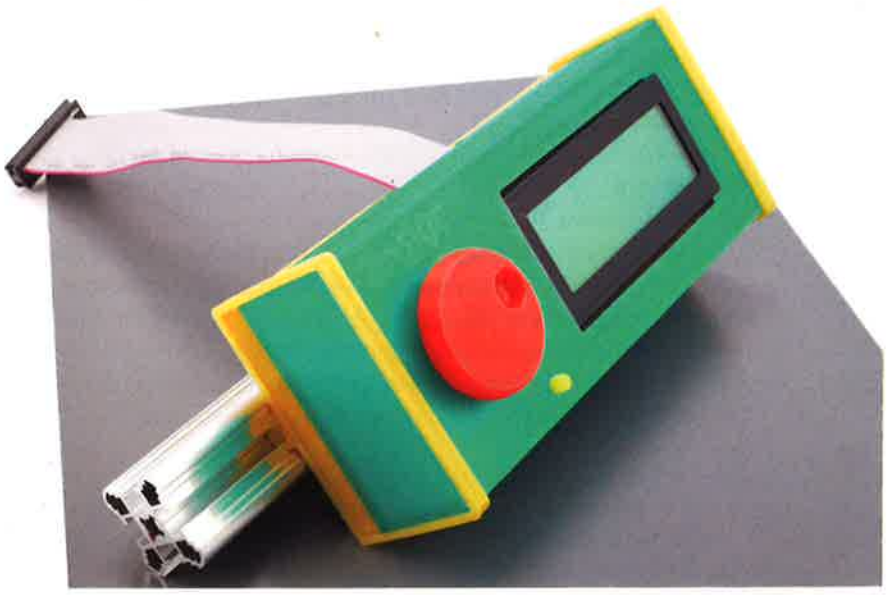


Image 7. : vue du boîtier une fois le couvercle et le fond emboîtés. Remarquez les 2 cadres jaunes servant de support à l'ensemble.

MK172

DÉCOREZ VOTRE SAPIN AVEC UNE ÉTOILE SPECIALE !

de Paolo GASPARI

Cette étoile originale munie de LED est appropriée pour décorer votre sapin de Noël et attirer l'attention de vos invités. Le montage est de type analogique, sans programme et sans microcontrôleur. Il dispose d'un microphone qui capte la musique ou la parole, et allume un nombre plus ou moins grand de LED en fonction du niveau sonore ambiant, un peu comme un VU-mètre.

Comme chaque année la fête de Noël arrive et avec elle les vacances tant attendues, en particulier par les enfants qui les associent aux cadeaux. Elles sont aussi une opportunité pour construire des décorations traditionnelles, coutume à laquelle même les adultes n'échappent pas.

Nous vous proposons une décoration simple à réaliser avec une touche technologique qui s'apparente à un jeu de lumière ... quelque chose ressemblant à une étoile à cinq branches.

Une étoile un peu spéciale, car son illumination n'est pas due à une séquence fixe ou un programme fonctionnant en boucle ! Elle se comporte un peu comme un VU-mètre, en ce sens qu'**elle est dotée de LED de différentes couleurs qui sont allumées en fonction des voix ambiantes ou des sons produits dans la pièce**. Ceci est possible grâce à l'utilisation d'un microphone intégré. L'étoile peut être combinée avec un haut-parleur (ou une enceinte acoustique) permettant ainsi d'allumer les LED en fonction de la musique en cours de diffusion.

L'étoile est réglée de telle manière qu'au repos, donc sans fond sonore, toutes les LED sont éteintes. Au fur et à mesure que le niveau sonore augmente, les LED situées vers l'intérieur de l'étoile commencent à s'allumer et ensuite celles disposées vers l'extérieur, comme si chaque rangée constituait une barre d'un VU-mètre.

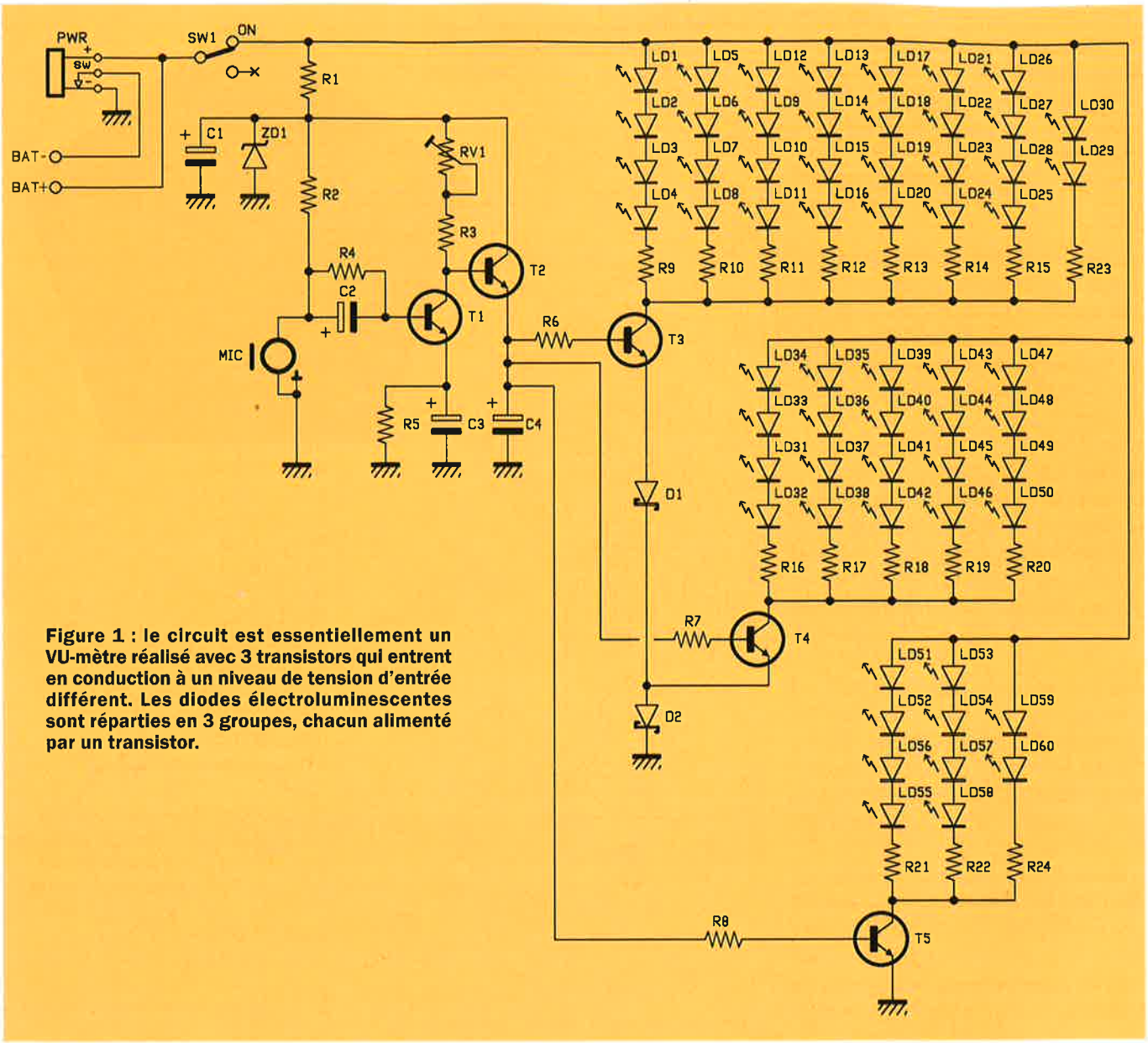


Figure 1 : le circuit est essentiellement un VU-mètre réalisé avec 3 transistors qui entrent en conduction à un niveau de tension d'entrée différent. Les diodes électroluminescentes sont réparties en 3 groupes, chacun alimenté par un transistor.

Le schéma électrique

Voyons en quoi consiste le projet, en jetant un coup d'œil au schéma électrique visible en figure 1.

Le montage est relativement simple avec **seulement 5 transistors**, en dehors de la grande quantité de LED nécessaires pour construire une animation lumineuse bien visible à une certaine distance.

Fondamentalement, le montage est un VU-mètre, c'est-à-dire un dispositif électronique indiquant (ou montrant) le niveau d'un signal électrique ou mieux, dans notre cas particulier, le niveau sonore ambiant dans une pièce.

Le transistor **T1 constitue l'interface d'entrée et reçoit le signal audio** provenant du microphone « **MIC** ». Il fonctionne comme un **amplificateur pour microphone** et élève la composante audio générée par la capsule microphonique, celle-ci captant les sons et les bruits environnants .

En sortie du collecteur du transistor T1 se trouve **la base d'un 2ème transistor** (**T2**) de type **NPN**, et dont la fonction est d'amplifier encore la composante audio.

T1 fonctionne en « émetteur commun » et T2 en « collecteur commun ». Ils procurent ainsi un **gain en tension d'environ 100 fois**, juste assez pour piloter les étages suivants constituant le VU-mètre.

Notez que le 1er amplificateur **T1 a un gain réglable**

[plan de MONTAGE]

Figure 2 : circuit imprimé à l'échelle 1 : 1 de l'étoile de Noël.

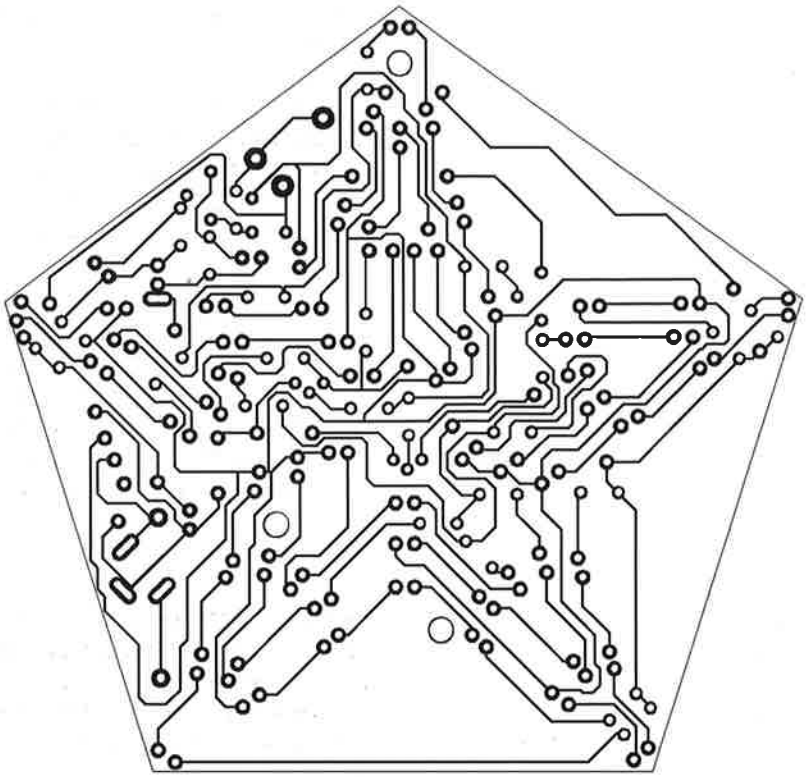
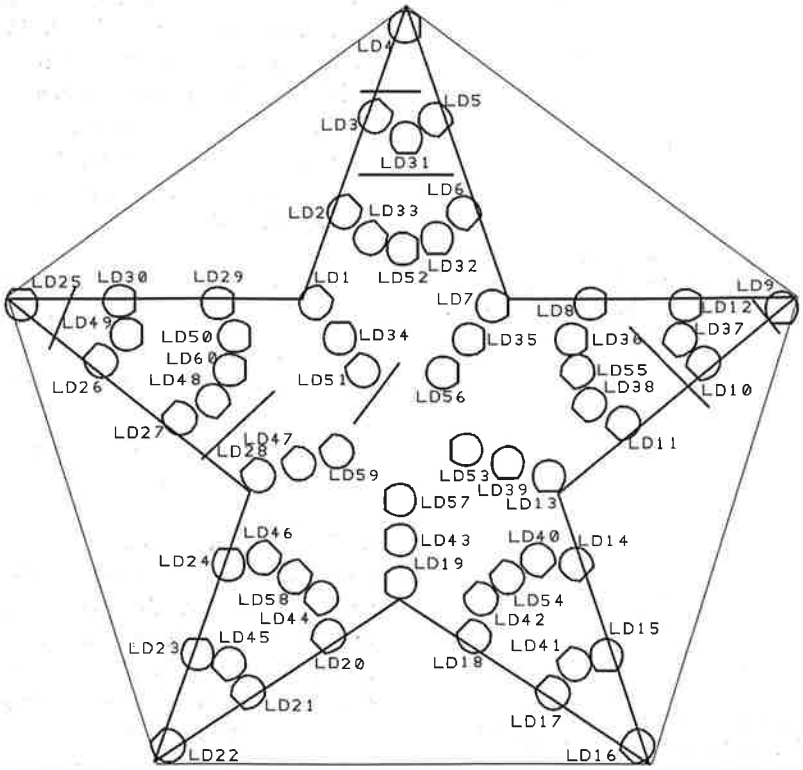


Figure 3 : schéma d'implantation des LED et des 7 ponts côté composants.



Liste des composants du MK172

R1..... 100 Ω
R2..... 4,7 kΩ
R3..... 68 kΩ
R4..... 4,7 MΩ
R5..... 10 kΩ
R6..... 4,7 kΩ
R7..... 4,7 kΩ
R8..... 4,7 kΩ
R9..... 2,2 kΩ
R10.... 2,2 kΩ
R11 ... 2,2 kΩ
R12 ... 2,2 kΩ
R13 ... 2,2 kΩ
R14.... 2,2 kΩ
R15 ... 2,2 kΩ
R16.... 1 kΩ
R17.... 1 kΩ
R18 ... 1 kΩ
R19 ... 1 kΩ
R20 ... 1 kΩ
R21.... 1 kΩ
R22 ... 1 kΩ
R23 ... 5,6 kΩ
R24.... 5,6 kΩ
RV1.... trimmer 220 kΩ mono-tour

C1..... 47 µF/16 V électrolytique
C2..... 1 µF/16 V électrolytique
C3..... 10 µF/16 V électrolytique
C4..... 4,7 µF/16 V électrolytique

D1..... BAT85
D2..... BAT85

ZD1.... diode zener 9,1 V/400 mW

T1 ..... BC547
T2 ..... BC547
T3 ..... BC547
T4 ..... BC547
T5 ..... BC547

SW1... interrupteur à glissière

MIC.... capsule microphonique

LD1 à LD30....LED 3 mm rouge
LD31 à LD50..LED 3 mm jaune
LD51 à LD60..LED 3 mm verte

Divers

Support de pile 6F22 pour ci
Fiche alimentation avec coupure droite

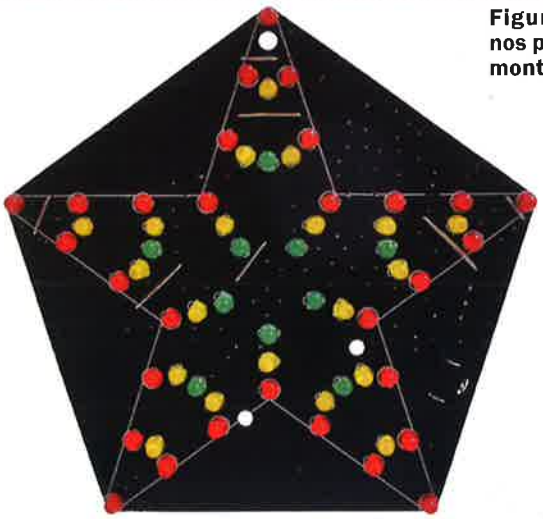


Figure 3a : photo de l'un de nos prototypes une fois les LED montées.

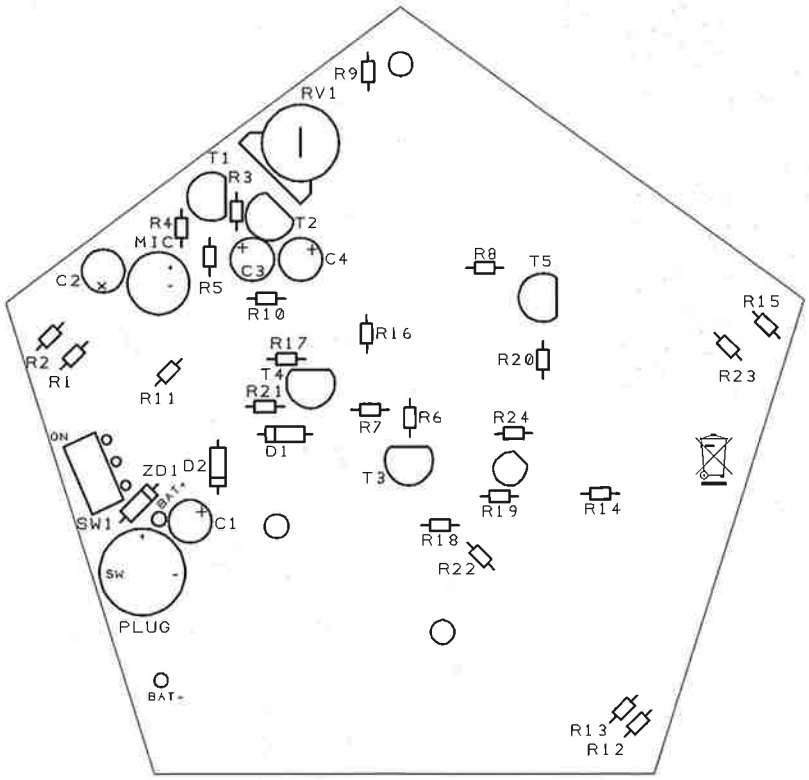


Figure 4a : photo de l'un de nos prototypes montrant les composants soudés du côté cuivre.

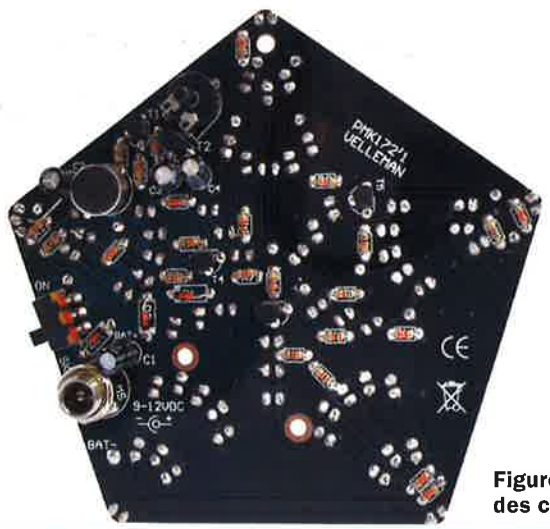


Figure 4 : schéma d'implantation des composants côté soudures.

afin d'adapter facilement la sensibilité (et donc le seuil d'allumage des LED) au niveau sonore ambiant. La **résistance de charge**, c'est-à-dire celle présente **sur le collecteur de T1**, est le potentiomètre **RV1**. Celui-ci permet de **modifier l'amplification** d'environ 100 fois jusqu'à 300 fois.

Le **signal** présent sur l'**émetteur** de **T2**, après avoir été filtré par le condensateur électrolytique C4, **pilote les 3 groupes de LED**, chacun étant alimenté par le **courant de collecteur** de l'un des 3 transistors NPN **T3**, **T4** et **T5**.

Les diodes électroluminescentes sont réparties en couleurs et **sont allumées en fonction du niveau de la tension présente aux bornes du condensateur C4**. Le 1er groupe est constitué par les LED vertes, le 2ème par les LED jaunes et enfin le 3ème groupe par les LED rouges.

Ce dernier groupe correspond à des niveaux sonores élevés. Pour obtenir un effet lumineux spécial, les couleurs sont mélangées de manière appropriée et divisées en lignes en forme d'étoile.

Le transistor **T3 pilote les LED rouges**, **T4 les LED jaunes** et **T5 les LED vertes**. Vous pouvez vérifier la polarisation de chaque transistor. **T5 entre en conduction avec le niveau de tension le plus faible** (le bruit ambiant présent dans la pièce), car son émetteur est directement relié à la masse (0 V). Pour le faire conduire, il suffit que sa tension base-émetteur V_{BE} **atteigne 0,7 V**.

Le transistor **T4 entre en conduction en second**, c'est-à-dire avec un niveau de tension aux bornes C4 supérieur à celui nécessaire pour allumer les LED vertes, mais inférieur à celui nécessaire pour faire conduire (on dit aussi polariser) le transistor T3.

En fait, l'**émetteur de T4** qui est un **NPN est relié à la masse à travers une diode Schottky polarisée en direct**. Cette diode introduit un seuil de conduction de **0,3 V**. Il en résulte que T4 entre en conduction lorsque l'émetteur de T2 restitue une tension dont l'amplitude est d'**au moins 1 V**.

Enfin, concernant le transistor **T3**, son émetteur est relié à la masse à travers les **2 diodes Schottky D1** et **D2**.

Il en résulte une chute de tension de 0,6 V aux bornes des diodes et donc pour polariser le transistor T3 il faudra une tension d'environ **1,3 V** sur l'émetteur de **T2** (V_{BE} de T3 + chute de tension de D1 et D2 = 0,7 + 0,6 = 1,3 V) pour allumer les LED rouges.

La **tension de polarisation de T3 est donc supérieure à celle de T4 et T5.**

L'ensemble du montage est prévu pour fonctionner avec une tension continue fournie par une de 9 V (type 6F22). L'interrupteur SW1 permet d'alimenter le montage lorsque cela est nécessaire, préservant ainsi la pile.

Vous pouvez aussi alimenter le montage à l'aide d'une alimentation de 12 V. Dans ce cas, la tension 12 V atteint directement les LED, tandis que l'étage d'amplification et la capsule microphonique sont alimentés par une tension de 9 V stabilisée grâce à la diode zener ZD1.

Si la pile est connectée au circuit, cela ne pose pas de problème car le micro interrupteur contenu dans la fiche d'alimentation isole la pile du montage.

Notez enfin la **configuration particulière des LED** qui sont, pour chaque transistor, **connectées en série et en parallèle.**

Ce type de branchement permet d'optimiser la consommation, et **réduit les pertes dans les résistances de chute**.

En fait, nous exploitons la tension d'alimentation pour alimenter un maximum de LED en série et chaque série de LED est connectée en parallèle avec l'autre.

Si nous avions opté pour une alimentation en parallèle, nous aurions dû mettre une résistance en série avec chaque LED.

Outre l'augmentation de la taille du circuit imprimé, cela aurait impliqué une augmentation de la consommation.

Réalisation pratique

Après vous avoir expliqué comment fonctionne ce gadget lumineux, nous allons étudier la manière de le construire. Tous les composants tiennent place sur un seul circuit imprimé simple face et donc relativement facile à graver.

Le typon du circuit est visible en figure 2 et téléchargeable sur notre site www.electroniquemagazine.com dans le sommaire détaillé du n°133 à l'onglet « Télécharger ».

Pour les moins courageux, sachez que ce gadget existe sous forme de kit complet dans la plupart des magasins d'électronique ou sur le web.

Le montage de ce gadget nécessite un peu d'attention et de compétences car la plupart des composants doivent être disposés et soudés du côté cuivre.

Seuls les LED et les 7 ponts (fabriqués à partir des pattes des résistances et des condensateurs) sont soudés côté composants.

Pour l'implantation des composants suivez attentivement les figures 3 et 4. Les composants doivent être insérés du côté de la couche de cuivre à leurs emplacements respectifs.

Attention, veillez à l'orientation des transistors, des **diodes** et des **condensateurs**.

N'enfoncez pas les composants jusqu'au bout, mais gardez une distance nécessaire pour la pointe du fer à souder. Evitez de garder le fer proche du corps des composants.

Ensuite coupez les pattes des composants afin qu'elles ne gênent pas pour l'insertion des LED du côté composants (voir la figure 4a).

Quant aux LED, vous devez les implanter comme illustré aux figures 3 et 3a, la **cathode est indiquée par le méplat** sur le boitier des LED (c'est aussi la patte la plus courte).

Pour l'alimentation par pile, prévoyez un support pour circuit imprimé pour une pile de type 6F22 de 9 V.

Le support doit être monté du côté des pistes de cuivre comme pour les composants. Comme il est en plastique, il n'y a donc pas de risque de court-circuit. Rappelez-vous que le **fil rouge** correspond à « **BAT +** » et le **noir** à « **BAT-** ».

Pour l'alimentation du montage avec une source de 12 VDC, vous devez insérer le connecteur de l'alimentation dans la fiche nommée « PLUG » du côté des pistes de cuivre (comme pour les composants).

Attention cette fiche dispose d'un coupe-circuit qui permet de laisser la pile dans le support. Faites attention au modèle.

Notez que **si vous ne montez pas cette fiche, vous devez court-circuiter les 2 pastilles correspondant aux contacts de la masse.**

En effet ce type de fiche contient un contact qui relie les 2 points lorsqu'aucune prise n'est insérée (ce qui permet de faire fonctionner le montage avec la pile, dès que la prise est insérée le contact est ouvert). Si vous ne montez pas la fiche « PLUG », il n'y a pas de contact.

Le montage fonctionnera sur pile uniquement si les 2 points de la masse sont reliés. Ces 2 points sont reliés à « BAT- » et la masse (« PWR - »).

Concernant le bloc d'alimentation secteur, il doit fournir une tension continue de **9 VDC** à **12 VDC** et un courant de **500 mA** environ.

Les typons des circuits imprimés ainsi que les fichiers GERBER sont téléchargeables gratuitement sur notre site : **www.electroniquemagazine.com** dans le sommaire détaillé de la revue numéro **133** section « **Télécharger** ».

Programmez avec Android

Sixième partie

de Andrea Chiapporl

Nous vous présentons dans cet article une nouvelle application qui peut gérer les appels GSM et SMS. Nous allons aborder la création d'un « service » Android et approfondir l'utilisation de certains capteurs des smartphones, comme l'accéléromètre ou la localisation GPS (sous réserve que le vôtre en soit équipé).

Maintenant, nous entrons dans la réalisation d'applications plus complexes sous Android et notamment , nous allons étudier la gestion directe des appels et des messages (SMS) d'un Smartphone, ainsi que la manière d'interagir avec les détecteurs de mouvements et la localisation GPS d'un smartphone, afin de piloter une carte « Arduino GSM » que nous présenterons dans le prochain numéro. En particulier, vous serez en mesure d'activer des dispositifs externes connectés à la carte grâce à un appel géré en interne par notre application, mais aussi d'envoyer diverses commandes par des SMS personnalisables.

Veuillez noter que cette application pourra être transposable à la carte d'extension GSM pour RaspberryPi moyennant des adaptations, la logique de fonctionnement globale restant la même.

Notre application dispose de plusieurs fonctionnalités. Par exemple, il permet de passer un appel à un numéro de téléphone prédéterminé après que le smartphone ait été secoué de manière répétée, ou dès que le smartphone se trouve à proximité d'une zone géographique préalablement déterminée et qui est réglable (voir la figure 1).

Cela nous donne l'occasion d'avoir un aperçu des possibilités des capteurs de mouvements et de la localisation GPS sous Android, mais aussi d'introduire un nouveau concept Android qui est le « **service** ». Ce dernier permet maintenir l'écran actif et l'application en cours de fonctionnement pendant tout le cycle de vie. Bien que cette application ait été conçue pour une approche didactique, elle permet la réalisation d'automatismes commandés à distance, comme une ouverture de portail ou un avertisseur de présence.

Les services sous Android

Bien que dans l'application décrite dans les précédents articles (carte LED RVB), nous avons toujours eu affaire à la classe « **Activity** » qui est dotée d'une interface graphique pour l'utilisateur et avec laquelle il est possible d'interagir activement, nous allons désormais la considérer comme

une sorte de code fonctionnant en arrière-plan (tâche de fond). Cela nous permet d'utiliser notre smartphone normalement, c'est-à-dire en restant en attente d'événements extérieurs (tels que les signaux provenant des capteurs de mouvement, la luminosité, le GPS ou autres) auxquels il va réagir en conséquence.

Pour ce faire nous allons utiliser précisément un « **service** » qui, sous Android, n'est autre qu'une classe (donc un fichier) c'est-à-dire une extension de la classe de base « **Service** ». Un « **service** » est donc une composante essentielle d'une application. Il est nécessaire lorsque l'application effectue des opérations ou des calculs en dehors de l'interaction utilisateur. Un « **service** » ne dispose pas d'interface graphique.

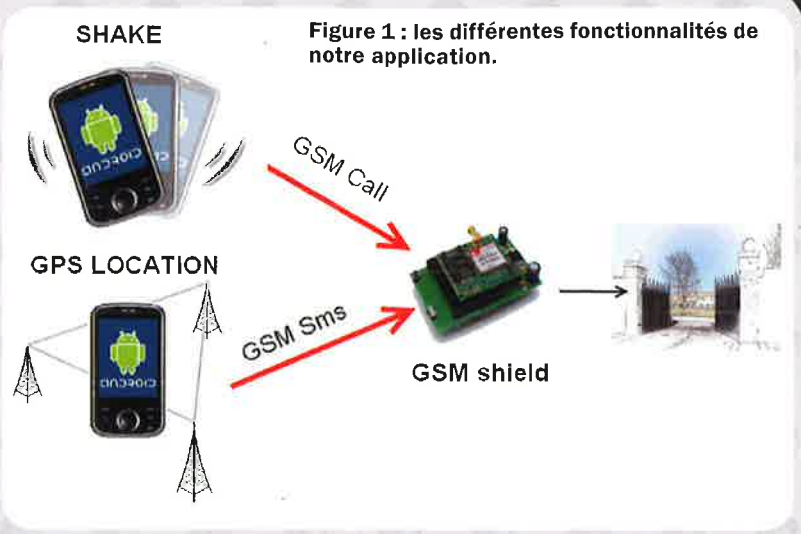


Figure 1 : les différentes fonctionnalités de notre application.

Par exemple le lecteur multimédia d'un smartphone permet d'écouter de la musique tout en surfant sur le web ou en lisant des mails. Cette fonctionnalité n'est possible qu'à l'aide de « **services** ». Il existe 2 types de services :

- « **LocalService** » : c'est un « **service** » qui s'exécute dans le même processus que l'application ;
- « **RemoteService** » : c'est un « **service** » qui s'exécute dans un processus indépendant de l'application.

Avant de passer à la création de notre premier **service**, nous allons créer un nouveau projet en suivant les mêmes étapes que dans les articles précédents, car la première classe avec laquelle nous allons lancer l'application sera toujours une classe « **Activity** » A travers cette dernière (dotée d'une interface graphique), nous pourrons lancer le « **service** » lui-même, l'arrêter et le configurer à notre convenance. Par exemple, nous pourrons ainsi modifier les numéros de téléphone, la sensibilité des capteurs et d'autres options que nous verrons plus tard.

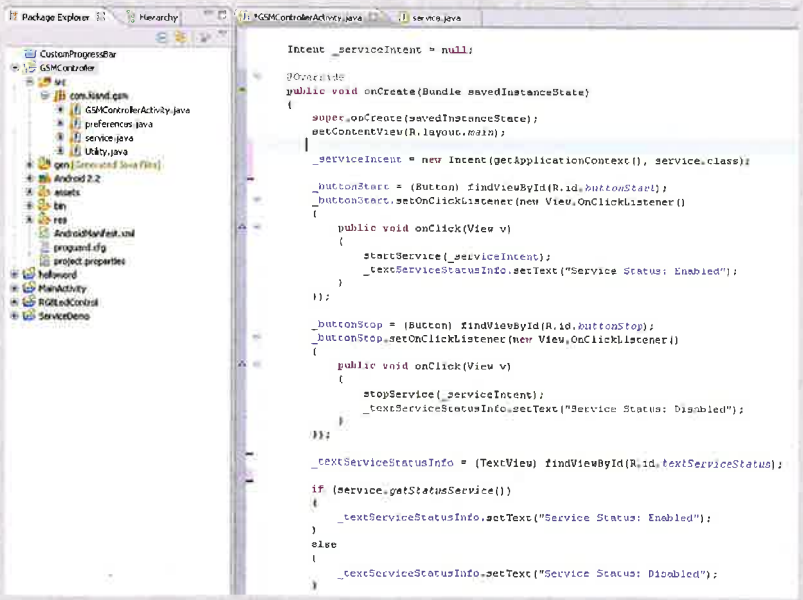
Donc, tout d'abord nous ajoutons **2 boutons** simples (« Start » et « Stop ») pour **démarrer** et **arrêter** le **service**, et un « **TextView** » qui indiquera l'état du **service** (voir la figure 4). Notez que si vous ne comprenez par les termes employés ci-avant et dans la suite de l'article reportez-vous au cours « **Programmez avec Android** » à partir du **numéro 128 et suivants d'Electronique et Loisirs Magazine**. Sans entrer dans les détails, nous allons associer à la pression de chaque bouton les fonctions du système avec lesquelles nous pouvons démarrer et arrêter le « **service** » de l'« **Activity** ».

Comme le montre la figure 2, pour démarrer le « **service** » il faut créer, dans la classe « **Activity** », un « **Intent** », avec lequel nous passerons le nom de notre classe « **GSMService** » (dont le nom est le même que celui du fichier « GSMService.java »).

L'« **Intent** » est ajouté dans le dossier « **src** ». Par la suite, nous appellerons la méthode « **startService()** » en tant que paramètre de l'« **Intent** » que nous venons de créer. De même, pour arrêter le « **service** » nous utiliserons la méthode « **stopService()** » toujours avec le même « **Intent** ».

Nous allons écrire maintenant le code de notre « **service** » c'est-à-dire le fichier « **GSMService.java** ». Pour faciliter la compréhension, nous allons ajouter un message « **Toast** » à l'entrée de la fonction qui gère le « **service** » pour signaler le cycle de vie du service. Un message « **Toast** » est un message d'information, d'**avertissement**. Il est **transitoire** car il ne nécessite aucune intervention de la part de l'utilisateur. Dans le cas où celui-ci serait en train d'effectuer une saisie dans un champ, la saisie continuera dans ce même champ durant tout le temps de l'affichage du message. Enfin, le **message disparait de lui-même**.

La figure 3 représente le squelette d'un service que nous allons compléter selon nos besoins.



```
Intent _serviceIntent = null;

@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    _serviceIntent = new Intent(getApplicationContext(), service.class);

    _buttonStart = (Button) findViewById(R.id.buttonStart);
    _buttonStart.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View v)
        {
            startService(_serviceIntent);
            _textServiceStatusInfo.setText("Service Status: Enabled");
        }
    });

    _buttonStop = (Button) findViewById(R.id.buttonStop);
    _buttonStop.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View v)
        {
            stopService(_serviceIntent);
            _textServiceStatusInfo.setText("Service Status: Disabled");
        }
    });

    _textServiceStatusInfo = (TextView) findViewById(R.id.textServiceStatus);

    if (service.getStatusService())
    {
        _textServiceStatusInfo.setText("Service Status: Enabled");
    }
    else
    {
        _textServiceStatusInfo.setText("Service Status: Disabled");
    }
```

Figure 2 : pour démarrer le « service », il faut créer dans la classe « Activity » un « Intent ».

GSMService.java

```
package com.kiand.gsm;

import android.app.Service;

public class GSMService extends Service
{
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }

    @Override
    public void onCreate()
    {
        Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();

    }

    @Override
    public void onStart(Intent intent, int startid)
    {
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    }

    @Override
    public void onDestroy()
    {
        Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();

    }
}
```

Figure 3 : représentation d'un squelette d'un service que nous compléterons.

Le démarrage du « **service** » s'effectue par le biais de la **méthode** « **startService()** », mais nous devons d'abord créer le « **service** » et ensuite le lancer. Ces deux phases sont réalisées respectivement à l'aide des méthodes « **onCreate()** » et « **onStart()** », dont nous insérons le code pour permettre l'exécution des actions qui fonctionneront en tâche de fond.

Nous allons aussi utiliser certaines classes spécialisées telles que la **classe** « **Listener** » qui permet de scruter des évènements. Pour savoir sous Android si un bouton est pressé, il faut utiliser la classe « **Listener** ». Celle-ci scrute en permanence les boutons et déclenche une méthode si l'un d'eux est pressé. Sur l'objet bouton que nous voulons scruter, nous utilisons la méthode « **setOnClickListener()** » en lui indiquant une instance de classe dans laquelle se trouvera une méthode qui doit être déclenchée en cas de pression sur ce bouton, nous reviendrons plus tard sur le sujet.

La méthode « **onDestroy()** » est appelée lorsque le **service** se termine, après l'appel de la méthode « **stopService()** » de l'« **Activity** », ou celui de la méthode « **stopSelf()** » interne à la classe « **Service** ». À ce point la classe « **Listener** » est aussi arrêtée.

Enfin, la méthode « **onBind()** » dans notre cas nous renvoie toujours une valeur nulle, car l'application ne gère pas les services de type « **Bounded** » dans lesquels une autre « **Activity** » peut interagir avec le « **service** » .

Listing 1

```
// GSMService.java
@Override
public void onCreate()
{
        Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();
        Log.d(TAG, "onCreate");
        // sensor
        _sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        Sensor s = _sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        _sensorManager.registerListener(_accelerometerSensorListener, s, SensorManager.SENSOR_DELAY_NORMAL);
}
@Override
public void onStart(Intent intent, int startid)
{
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        Log.d(TAG, "onStart");
        _statusService = true;
}
@Override
public void onDestroy()
{
        Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();
        Log.d(TAG, "onDestroy");

        _sensorManager.unregisterListener(_accelerometerSensorListener);
        _sensorManager = null;
        _statusService = false;
}
```

Listing 2

```
private  SensorEventListener _accelerometerSensorListener = new SensorEventListener()
{
        private int _count = 0;
        private long firstShake = 0;

        private int SHAKE_DURATION = 600;
        private int SHAKE_COUNT = 4;
        private int SHAKE_THRESHOLD = 2000;
        private int SHAKE_INTERVAL = 500;
        public void onAccuracyChanged(Sensor sensor, int accurancy)
        {
                //...
        }
        public void onSensorChanged(SensorEvent event)
        {
                float[] values = event.values;
                if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
                {
                        long curTime = System.currentTimeMillis();

// update every 100ms.
                        if ((curTime - lastUpdate) > 100)
                        {
                                long diffTime = (curTime - lastUpdate);
                                lastUpdate = curTime;

                                x = values[SensorManager.DATA_X];
                                y = values[SensorManager.DATA_Y];
                                z = values[SensorManager.DATA_Z];

                                NumberFormat numberFormat = new DecimalFormat("0.00");
                                String strX = numberFormat.format(x);
                                String strY = numberFormat.format(y);
                                String strZ = numberFormat.format(z);


                                float speed = Math.abs(x+y+z - last_x - last_y - last_z) / diffTime * 10000;

                                if (speed > (SHAKE_THRESHOLD) )
                                {

                                        Log.d(TAG, "_count:" + _count );
                                        if (_count == 0)
                                        {
                                                firstShake = curTime;
                                        }
                                        _count++;

                                        long durationStep = (curTime - firstShake);
                                        if(durationStep > SHAKE_INTERVAL)
                                        {
                                                _count = 0;
                                                Log.d(TAG, "durationStep:" + durationStep);
                                        }
```

```
                    if (_count >= SHAKE_COUNT)
                    {
                        _count = 0;
                        long duration = (curTime - firstShake);
                        if( duration <= SHAKE_DURATION )
                        {
                         Log.d(TAG, "SHAKE duration:" + duration + "count:"
            + _count );

                         _vibrate.vibrate(500);
                         callNumber("33671086810");
                        }
                    }
                }
                last_x = x;
                last_y = y;
                last_z = z;
            }
        }
    }
};
```

Nous pouvons maintenant ajouter à notre classe une méthode privée « **getStatusService()** » grâce à laquelle nous allons vérifier l'état du service (actif ou arrêté) au démarrage de l'application. Il nous suffit de définir notre propre variable interne à « **TRUE** » (vrai) lors de la phase de démarrage (« Start ») et à « **FALSE** » (faux) lors de la phase d'arrêt (« Destroy »).

Enfin, il convient de rappeler le « **service** » pour l'activer même si l'« **Activity** » principale de notre application a été fermée. Il continuera à fonctionner et nous pourrons l'arrêter en rouvrant notre application et en appuyant sur le bouton « **Stop** ». Pour vérifier l'état du « **service** », nous allons utiliser la zone de texte « **textServiceStatusInfo** ». Un premier « layout » (main.xml) de notre application est représenté en figure 4.

Avant de continuer, il peut être utile de donner quelques informations sur la gestion de la mémoire RAM dans un système Android. Lorsque nous fermons une quelconque application sous Android, celle-ci reste en mémoire jusqu'à ce qu'une autre application exige une quantité de mémoire supérieure à celle restant disponible (libre), ou alors après un certain lapse de temps depuis la dernière utilisation de l'application.

Un « **service** » en tâche de fond a une priorité inférieure à celle d'une « **Activity** » s'exécutant en premier plan, mais il a une priorité plus importante qu'une « **Activity** » qui reste en mémoire après avoir été fermée. De cette manière nous sommes sûrs que, sous réserve de conditions particulières, Android n'interrompra pas notre « **service** ».

Donc, comme nous l'avons fait dans les leçons précédentes, lorsque nous avons ajouté d'autres « **Activity** » dans le projet, nous devons informer le système de l'utilisation de la classe « **GSMService** » en ajoutant l'instruction suivante dans le fichier « **AndroidManifest.xml** » :

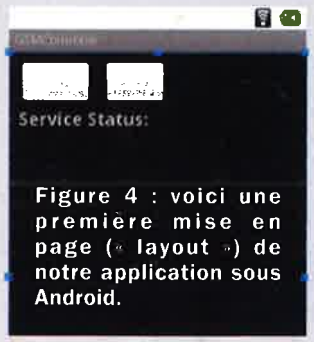


Figure 4 : voici une première mise en page (« layout ») de notre application sous Android.

<service android:enabled="true" android name=".GSMService" />

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.kiand.gsm"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_GPS"/>
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".GSMControllerActivity"
            android:screenOrientation="portrait" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:enabled="true" android:name=".GSMService" />

    </application>
</manifest>
```

Figure 5 : le fichier « AndroidManifest.xml » au complet.

Les capteurs sous Android

Android, à travers la classe « SensorManager », nous permet d'accéder aux capteurs présents dans notre smartphone ou tablette. Cependant tous les appareils fonctionnant sous Android n'ont pas les mêmes caractéristiques et les mêmes capteurs, de sorte qu'il peut être utile d'obtenir une liste des capteurs présents grâce aux lignes de code suivantes :

```
SensorManager manager = (SensorManager) getSystemSer-
vice(SENSOR_SERVICE);
List<Sensor> list = manager.getSensorList(Sensor.TYPE_ALL);
```

Si vous connaissez déjà le type de capteur dont vous disposez, vous pouvez utiliser directement la méthode « getDefaultSensor() » en lui transmettant le type de capteur avec la ligne suivante :

```
Sensor s = _sensorManager.getDefaultSensor(Sensor.TYPE_
ACCELEROMETER);
```

Type	Capteur
TYPE_GYROSCOPE	Gyroscopique
TYPE_LIGHT	Capteur de lumière
TYPE_ACCELEROMETER	Accéléromètre
TYPE_MAGNETIC_FIELD	Capteur de champ magnétique
TYPE_PRESSURE	Capteur de pression
TYPE_ORIENTATION	Capteur d'orientation
TYPE_PROXIMITY	Capteur de proximité
TYPE_TEMPERATURE	Capteur de température

Méthode du capteur	Description
getMaximumRange()	rayon d'action du capteur
getname()	nom du capteur
getPower()	courant consommé du capteur en mA
getResolution()	résolution du capteur
getVendor()	fabricant du capteur
getVersion()	version du capteur

Les tableaux ci-dessus montrent une liste complète des capteurs et quelques méthodes de l'objet « Sensor » qui permettent d'obtenir des informations supplémentaires.

Cette ligne est fondamentale pour la bonne exécution du « **service** », sinon même si le projet est correctement compilé, nous obtiendrons une erreur de type « run-time » lors de l'exécution de l'application. Le fichier « **AndroidManifest.xml** » complet est visible en figure 5.

À ce stade, après avoir lancé notre « **service** » de l'« **Activity** » principale, nous pouvons fermer l'application (en utilisant le bouton physique de retour du smartphone) et utiliser normalement notre téléphone, sachant que pour arrêter le « **service** » nous devons relancer l'application et appuyer sur le bouton « Stop ». Lorsque nous ouvrons de nouveau l'application, la boîte de dialogue (c'est-à-dire la zone de texte « textServiceStatusInfo ») nous informe de l'état du « **service** ».

Maintenant nous disposons d'un « **service** » qui pour le moment ne fait rien. Pour vérifier sa présence dans le système, nous devons utiliser un **gestionnaire d'application** (« app manager ») que nous pouvons télécharger gratuitement sur le « market Android » (Google Play). Suivant le smartphone ou la tablette, le gestionnaire peut être déjà présent, il permet de surveiller tous les « services » actifs.

Gestion des capteurs sous Android

Nous devons maintenant « remplir » le « **service** » que nous avons créé, en le faisant réagir à des événements du système, tels que l'activation des capteurs de notre smartphone.

Pour savoir combien et quels capteurs sont présents dans notre smartphone, nous devons insérer les lignes suivantes, par exemple dans la méthode « **onCreate()** » de l'« **Activity** » principale :

```
SensorManager manager = (SensorManager) getSystem-
Service(SENSOR_SERVICE);
List<Sensor> list = manager.getSensorList(Sensor.
TYPE_ALL);
```

La classe « **SensorManager** » représente la gestion « globale » de tous les capteurs présents. A partir du gestionnaire d'objet, nous sommes en mesure d'obtenir un objet appartenant à la classe « **Sensor** » qui représente notre véritable capteur.

Listing 3

```
private void callNumber(String numberToCall)
{
        try
        {
                Intent callIntent = new Intent(Intent.ACTION_CALL);
                callIntent.setData(Uri.parse("tel:" + numberToCall));
                startActivity(callIntent);
                Log.v(TAG,"GSM Call Number:" + numberToCall);
        }
        catch (ActivityNotFoundException e)
        {
                Log.e("GSM Call Example", "Call failed", e);
        }
}

private void sendMessage(String phoneNumber, String message)
{
        SmsManager smsManager =  SmsManager.getDefault();
        smsManager.sendTextMessage(phoneNumber, null, message, null, null);
}
```

Listing 4

```
private PhoneStateListener _phoneListener = new PhoneStateListener()
{
        public void onCallStateChanged(int state, String incomingNumber)
        {
try
                {
                        switch (state)
                        {
                                case TelephonyManager.CALL_STATE_RINGING:
                                        Toast.makeText(GSMControllerActivity.this, "CALL_STATE_RINGING
                                FROM " + incomingNumber, Toast.LENGTH_SHORT).show();
                                break;

case TelephonyManager.CALL_STATE_OFFHOOK:
                                        Toast.makeText(GSMControllerActivity.this,
                                        "CALL_STATE_OFFHOOK", Toast.LENGTH_SHORT).show();
                                break;

case TelephonyManager.CALL_STATE_IDLE:
                                        Toast.makeText(GSMControllerActivity.this, "CALL_STATE_IDLE",
                                Toast.LENGTH_SHORT).show();
                                break;

                                default:
                                        Toast.makeText(GSMControllerActivity.this, "default",
                                Toast.LENGTH_SHORT).show();

Log.i("Default", "Unknown phone state=" + state);
                        }
                }
                catch (Exception e)
                {
                  Log.i("Exception", "PhoneStateListener() e = " + e);
                }
        }
};
```

La méthode « **getSensorList()** » du gestionnaire d'objet nous donne le capteur désiré.

Après avoir utilisé l'identifiant « **TYPE_ALL** » pour indiquer de quels types de capteurs nous disposons sur notre smartphone ou tablette, le gestionnaire renvoie une liste complète des capteurs présents. Mais si nous voulons trouver un type de capteur particulier, nous pouvons utiliser directement le type approprié, comme par exemple « **TYPE_PRESSURE** » dans le cas d'un capteur de pression.

Dans notre cas, nous utilisons la méthode « **getDefaultSensor()** » qui retourne directement l'objet unique « **Sensor** » correspondant. Dans l'encadré intitulé « **Les capteurs sous Android** » que vous trouverez ci-après dans ces pages, vous pourrez voir à quels types de capteurs vous aurez accès.

Dans notre application nous utilisons l'accéléromètre, nous accédons à notre objet « **Sensor** » à l'aide du type « **TYPE_ACCELEROMETER** ». De l'objet « **Sensor** » nous pouvons obtenir des informations intéressantes telles que la résolution, la consommation du capteur, la plage de fonctionnement et d'autres fonctionnalités.

Comme nous voulons récupérer toutes les données du capteur en mode asynchrone, sans utiliser la technique du « polling » (c'est une technique d'interrogation continuelle et séquentielle des périphériques pour vérifier s'ils ont des données à transférer), nous devons enregistrer un objet de la classe « **Listener** » (_accelerometerSensorListener) à l'aide de la méthode « **registerListener()** » de notre gestion « **SensorManager** », comme vous pouvez le voir dans le **Listing 1**.

De cette manière, nous indiquons au système Android d'intercepter toutes les données disponibles du capteur et de les déclarer à notre classe « **Listener** » que nous analysons brièvement. On notera également que le 3ème paramètre passé par la méthode « **registerListener()** » représente la fréquence d'échantillonnage du capteur.

Pour notre application, nous avons choisi une valeur normale (SENSOR_DELAY_NORMAL), mais dans certains cas

Listing 5

```
#include <GSM_Shield.h>
GSM gsm;

void setup()
{
 gsm.TurnOn(9600);
 gsm.InitParam(PARAM_SET_1);
 gsm.Echo(1);

 pinMode(13, OUTPUT);
 pinMode(11, OUTPUT);
}

void loop()
{

  int call;
  call=gsm.CallStatus();
  switch (call)
        {
   case CALL_NONE:
    Serial.println("no call");
    break;
   case CALL_INCOM_VOICE:
    delay(5000);
    gsm.PickUp();
    break;
   case CALL_ACTIVE_VOICE:
    delay(5000);
    gsm.HangUp();

      // do something
      digitalWrite(13, HIGH);
        break;
  }

  String sms = getSMS();

        if (sms == "cmd1")
        {
              digitalWrite(13, LOW);
        }

        if (sms == "cmd2")
        {
              digitalWrite(11, LOW);
        }

        delay(1000);
}

String getSMS()
 {
   char indexSms;
   indexSms =gsm.IsSMSPresent(type_sms);

   char numberCalling[15];
        char smsBody[122];

        String smsStr = "";

   if (indexSms!=0)
   {

   gsm.GetSMS(indexSms,numberCalling, smsBody,120);
        smsStr = smsBody;
        }
      return smsStr;
}
```

vous aurez besoin d'un échantillonnage plus important (SENSOR_DELAY_FAST) ou moins élevé (SENSOR_DELAY_GAME).

Comme le montre le listing 1, au moment de l'arrêt du « **service** » (méthode « onDestroy()) »), nous devons annuler l'enregistrement de la classe « **Listener** » afin d'éviter de laisser fonctionner le processeur même après la fermeture de l'application (cela permet d'économiser la batterie car actuellement, l'autonomie d'un smartphone n'est que d'une journée).

Nous allons maintenant écrire la classe « **SensorEventListener** » présente dans le listing 2, dans laquelle nous aurons accès aux données réelles lues par notre capteur.

En particulier, nous allons implémenter la méthode « **onAccuracyChanged()** » qui fait partie de l'interface, mais que nous allons laisser vide, ainsi que la méthode « **onSensorChanged()** » qui sera appelée par le système à chaque échantillonnage des données du capteur et dans laquelle nous prendrons les décisions.

Avec cette méthode, en raison de l'enregistrement précédent, nous avons aussi l'information (toujours passée par le système) de l'événement déclenché par le capteur (event) à partir de laquelle nous pouvons obtenir les données réelles (event.values) qui, dans notre cas, sont représentées par un tableau (array) de type « float » contenant la valeur de l'accélération sur les 3 axes (X, Y et Z), et également l'indication du capteur (event.sensor) qui a produit l'évènement.

L'objectif est de **composer un appel vers un numéro de téléphone à chaque fois que le smartphone est secoué à plusieurs reprises**. Nous devons vérifier la présence de fortes variations d'accélération dans les 3 axes et pendant une période plus ou moins courte. Ceci est simplement réalisé en mémorisant les dernières valeurs, puis en soustrayant les valeurs actuelles.

Listing 6

```
private LocationListener _locationListener = new LocationListener()
{
        public void onLocationChanged(Location location)
        {
                _vibrate.vibrate(300);

                if (location == null)
                {
                        return;
                }

                _lastLocation = location;
                _buttonStoreLocation.setEnabled(true);
                _buttonStoreLocation.setBackgroundColor(Color.GREEN);

                float speed = location.getSpeed();
                double altitude = location.getAltitude();
                double longitude = location.getLongitude();
                double latitude = location.getLatitude();

                float distanceMeters = (float)0;
                if (_storedLocation != null)
                {
                        distanceMeters = location.distanceTo(_storedLocation);
                        if(distanceMeters < _radiusGPS)
                        {
                                // do somethings
                        }
                }

                String dbgInfoString = "LAT:"+ latitude + "\n" + " LON:" +longitude + "\n" + " ALT:" +
altitude + "\n" + " SPD:" + speed + "\n" + "DST:" + distanceMeters;
                Log.v(TAG, dbgInfoString);
                Toast.makeText(GSMControllerActivity.this, dbgInfoString , Toast.LENGTH_LONG).show();

                NumberFormat numberFormat = new DecimalFormat("0.000");
                _textCurrentLocation.setText("LAT:"+ numberFormat.format(latitude) + "    LON:" +
        numberFormat.format(longitude));

        }

        public void onStatusChanged(String provider, int status, Bundle extras)
        {
                Toast.makeText(GSMControllerActivity.this, status , Toast.LENGTH_SHORT).show();
        }
};
```

Ensuite nous comptons le nombre de « secousses » pour atteindre un nombre limite (SHAKE_COUNT) au-delà duquel nous vérifions le temps écoulé depuis la 1ère « secousse » et qui ne doit pas dépasser une certaine valeur maximale (SHAKE_DURATION).

Avec cette dernière nous décidons si le smartphone a été suffisamment secoué ou pas. Pour gérer la variable « temps », nous avons utilisé la fonction du système appelée « **System.currentTimeMillis()** » et qui retourne les millisecondes écoulées depuis Janvier 1970.

Si tous les seuils sont dépassés, nous activons l'appel vers le numéro de téléphone désiré, en utilisant la fonction « **callNumber()** » que nous allons étudier maintenant.

Les fonctions GSM

Avec Android il est possible de gérer des appels GSM et des messages SMS au sein d'une seule application. Nous allons utiliser cette possibilité pour envoyer des commandes à une carte GSM (Arduino ou RaspberryPi). En fonction des messages reçus, la carte activera un dispositif externe ou

Passage des paramètres entre des « Activity » et des « services »

Il arrive souvent de devoir gérer une application avec plusieurs « **Activity** » et services, nous devons donc transférer les paramètres entre les différentes classes, non seulement entre une « **Activity** » et un ou plusieurs services, mais aussi entre plusieurs « **Activity** ». Il existe plusieurs façons plus ou moins élégantes, la plus complexe étant de créer une fenêtre d'options (classe « Preferences ») qui sont partagées dans toute l'application que nous avons analysée dans ces pages.

Une première façon est de fournir à l'« **Intent** » des informations supplémentaires et personnalisées, appelées « **extras** » qui seront lues par le service ou l'« **Activity** » du destinataire, après avoir été appelées par les méthodes « **startActivity()** » ou « **startService()** » de la classe principale. Pour écrire et lire ces informations « extra », nous utilisons deux méthodes de l'« **Intent** », respectivement « **putExtra()** » et « **getExtra()** ». L'exemple est donné en figure A.

ACTIVITY

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    _serviceIntent = new Intent(getApplicationContext(), GSMService.class);

    _buttonStart = (Button) findViewById(R.id.buttonStart);
    _buttonStart.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v)
        {
            _serviceIntent.putExtra("Latitude", _latitude );
            _serviceIntent.putExtra("Longitude", _longitude );
            startService(_serviceIntent);

            _textServiceStatusInfo.setText("Service Status: Enabled");
        }
    });
```

SERVICE

```
@Override
public void onStart(Intent intent, int startId)
{
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    Log.d(TAG, "onStart");

    _statusService = true;

    Bundle bundle = intent.getExtras();
    double lat    = bundle.getDouble("Latitude");
    double lon    = bundle.getDouble("Longitude");

    Log.d(TAG, "Received values:" + lat + " " + lon);

}
```

Les chaînes « **Latitude** » et « **Longitude** » sont les **labels**, tandis que les **valeurs** elles-mêmes (dans ce cas elles sont de type double ...) sont « **_latitude** » et « **_longitude** ». Dans l'exemple, le destinataire est le service qui a déjà l'« **Intent** » comme paramètre, mais si cela avait été une autre « **Activity** » elle aurait reçu l'« **Intent** » associé à travers la fonction « **getIntent()** ». En outre l'objet « **Bundle** », en plus de la méthode « **getDouble()** », offre toutes les autres méthodes pour renvoyer les types correspondants.

La deuxième façon est basée sur la classe « **SharedPreferences** » qui utilise le même mécanisme que la classe « **Preferences** », mais sans devoir créer les paramètres d'une interface graphique. Nous pouvons la comparer au **registre de Windows**, car elle est basée sur l'association de « **clés-valeurs** » qui restent persistantes dans le système même après la fermeture de l'application. Ces informations sont enregistrées dans un fichier dont le chemin est : « **data/data/** ». Les informations peuvent ainsi être partagées avec d'autres applications. Dans l'exemple de la figure B, les informations sont mémorisées au sein même de la classe « **Activity** » et peuvent être récupérées lors d'un prochain redémarrage. Rien ne nous empêche aussi de les utiliser dans une autre « **Activity** » ou dans d'autres « services ». La chaîne « **MYPREFS** » identifie l'association entre les deux sections et peut contenir du texte. La constante « **MODE_PRIVATE** » indique cependant que l'information n'est valable que pour notre application. Enfin la méthode « **getString()** » doit être passée comme paramètre, en plus de la clé, même la valeur par défaut si aucune touche correspondante n'est trouvée.

```
@Override
public void onDestroy()
{

    //...
    // save the preferences
    SharedPreferences prefs = getSharedPreferences(MYPREFS, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString("PHONE_NUMBER", newPhoneNumber);
    editor.putInt("RADIUS", newRadius);
    editor.commit();

}
```

```
@Override
public void onCreate()
{
    // load the preferences
    SharedPreferences prefs = getSharedPreferences(MYPREFS, Context.MODE_PRIVATE);
    String phonenumber = prefs.getString("PHONE_NUMBER", "No Phone");
    int radius         = prefs.getInt("RADIUS", 0);

    //...
}
```

répondra en envoyant un SMS contenant des informations particulières.

Pour gérer un appel sortant, nous avons créé la fonction « **callNumber()** » dans laquelle, comme vous pouvez le voir dans le Listing 3, nous utilisons un « **Intent** » du système (ACTION_CALL). Celui-ci, après avoir été configuré avec le numéro de téléphone désiré, est « passé » à la fonction « **startActivity()** » dans laquelle se trouve l'interface graphique habituelle que nous pouvons observer lors de la composition d'un numéro de téléphone sous Android.

Après l'appel (le périphérique distant sera paramétré, à travers Arduino, pour mettre fin à l'appel et effectuer l'action demandée), nous revenons à l'interface de notre « **Activity** ». Comme d'habitude, nous ne devons pas oublier d'insérer dans le fichier « **AndroidManifest.xml** » la ligne suivante :

```
<uses-permission android:name="android.permission.
CALL_PHONE"/>
```

En ce qui concerne l'envoi d'un message (SMS), rien ne nous interdit dans ce cas également, d'utiliser l'« **Activity** » du système avec laquelle nous envoyons habituellement des messages. Cependant il existe une autre possibilité permettant d'obtenir le même résultat en utilisant la classe « SmsManager » sans afficher une interface supplémentaire.

Cela est possible, toujours dans le Listing 3, grâce à la fonction « **sendMessage()** » dans laquelle la chaîne « message » représente le corps du message et « **phoneNumber()** » le numéro de téléphone du destinataire.

Dans notre application, nous associons à 3 boutons, l'envoi de messages particuliers configurables, que nous insérons avec la procédure désormais standard de l'« **Activity** » principale. Encore une fois, nous ne devons pas oublier d'inclure les permissions suivantes :

```
<uses-permission android:name="android.permission.
SEND_SMS" />
```

dans le fichier « **AndroidManifest.xml** » et d'ajouter l'import suivant :

```
import android.telephony.SmsManager;
```

De même pour la gestion des appels entrants, l'approche est asynchrone. Mais maintenant nous savons utiliser des classes « **Listener** » que nous avons vues dans le cas de l'accéléromètre, il ne devrait pas y avoir de problème. Nous allons utiliser la classe « **TelephonyManager** » et la ligne suivante :

```
import android.telephony.TelephonyManager;
```

ainsi que la permission suivante dans le fichier « **AndroidManifest.xml** » :

```
<uses-permission android:name="android.permission.
READ_PHONE_STATE"/>
```

À ce stade dans la méthode « **onCreate()** » de notre « **service** », à travers la classe « **TelephonyManager** », nous allons enregistrer la classe « **PhoneStateListener** » à l'aide des lignes de code suivantes :

```
TelephonyManager _telephonyManager=TelephonyMan-
ager) getSystemService(TELEPHONY_SERVICE);

_telephonyManager.listen(_phoneListener, PhoneState-
Listener.LISTEN_CALL_STATE);
```

Évidemment, nous devons d'abord implémenter la classe « **_phoneListener** » comme indiqué dans le Listing 4, où nous complétons la méthode « **onCallStateChanged()** » qui est appelée par le système d'exploitation chaque fois que nous modifions l'état d'un appel.

Sachez aussi que de manière automatique lors d'un appel entrant (incomingNumber), nous pouvons contrôler et exécuter des fonctions spécifiques. Pour l'instant, comme notre application est avant tout didactique, nous nous limitons simplement à visualiser des messages « **Toast** ».

Notez que les messages « **Toast** » sont autorisés par le « **service** », même s'ils ne représentent qu'une sorte d'interface graphique pour l'utilisateur. Cela signifie que même si notre « **Activity** » ne s'exécute pas en premier plan, mais que notre « **service** » est actif, lorsque nous recevons un appel, indépendamment de l'état de notre dispositif, nous visualisons sur l'écran une notification par le biais d'un message « **Toast** ».

Maintenant, nous allons voir ce qui se passe du côté de la carte Arduino qui reçoit les appels et active ses sorties où des dispositifs sont connectés. Le Listing 5 montre un exemple d'utilisation de la librairie « **GSM_Shield** » d'Arduino qui gère les appels et les SMS entrants. Périodiquement les appels entrants sont contrôlés ainsi que la présence de nouveaux SMS. Au moins deux sorties de la carte sont activées en fonction des messages reçus.

Utilisation du GPS

Notre application est maintenant fonctionnelle, mais nous pouvons encore l'améliorer en lui ajoutant des fonctionnalités supplémentaires, comme par exemple passer un appel ou envoyer un SMS à une carte GSM chaque fois qu'un smartphone ou une tablette se trouve dans une zone géographique déterminée. Nous allons utiliser le GPS qui n'est pas un capteur de la liste précédente, mais plutôt un module distinct.

Le mécanisme par lequel nous pouvons obtenir des données GPS (latitude et longitude) correspondant à la position du smartphone est cependant le même. Il existe un gestionnaire (LocationManager) dédié exclusivement à gérer la position et qui est utilisé pour enregistrer une classe « **Listener** » (LocationListener) de différentes manières. Celui-ci est représenté par les lignes suivantes qui doivent être écrites dans la méthode « onStart() » de notre « **service** » :

```
Location Manager _locationManager = (LocationManager)
this.getSystemService(LOCATION_SERVICE);
_locationManager.requestLocationUpdates(LocationMan-
ager.GPS_PROVIDER, 0, 0, _locationListener);
```

Il est possible de déterminer la position via Internet (NETWORK_PROVIDER) avec une précision moindre. Évidemment, il est nécessaire que le smartphone soit configuré pour utiliser une connexion Wi-Fi ou GPS.

Nous vérifions cela dans le code et éventuellement nous utilisons un « **Intent** » du système pour visualiser directement la configuration Android avec laquelle nous définirons les méthodes à utiliser, grâce simplement à ces 2 lignes de code :

```
Intent intent = new Intent(Settings.ACTION_LOCATION_
SOURCE_SETTINGS);
startActivity(intent);
```

Le résultat de l'opération est visible en figure 6. La classe « **Listener** » correspondante est décrite dans le Listing 6.

À chaque changement de position est appelé la méthode « onLocationChanged() » qui fournit l'objet « **Location** » contenant la position exprimée par la longitude et la latitude ainsi que le nombre de satellites détectés.

En plus de cela, à travers l'objet « **Location** » nous sommes en mesure de calculer la distance en mètres à partir d'un point déterminé (grâce à la méthode « distanceTo() ») que nous aurons préalablement fixé en appuyant sur le bouton « **_storedLocation** ».

À ce stade pour déclencher un appel ou envoyer un message particulier, il suffit de contrôler si cette distance devient inférieure à une certaine valeur. Pour gérer le GPS, il est nécessaire d'ajouter les lignes suivantes :

```
import android.location.LocationManager;
import android.location.LocationListener;
import android.location.Location;
```

Les permissions du fichier « **AndroidManifest.xml** » sont celles de la figure 5. En particulier « **ACCESS_FINE_LOCATION** » est utilisée si nous nous servons du GPS, alors que « **ACCESS_COARSE_LOCATION** » sera utilisée dans le cas du réseau. Nous pouvons cependant mettre les 2 permissions si nous envisageons de les gérer dans la même application.

Préférences Android

L'application est désormais terminée, mais pour la rendre plus polyvalente, nous allons insérer un certain nombre d'options permettant de gérer certains paramètres tels que la zone de la localisation, la nature des « secousses » qui provoqueront l'appel, le numéro du destinataire, le corps du message envoyé lors de la pression sur un bouton et plus encore. Android nous donne la possibilité, à l'aide d'outils, de créer facilement une fenêtre regroupant des options.

Cette fenêtre, ou plutôt cette « **Activity** », est appelée « **Preferences** » et est la même que celle utilisée par Android lorsque nous accédons aux paramètres du système (ou « Setting »). Elle est généralement utilisée pour gérer de grandes quantités d'options et offre l'avantage de partager des informations choisies par l'utilisateur avec d'autres « **Activity** » du projet.

De cette façon, les modifications que nous effectuons dans « **Preferences** » (lancée de l'« Activity » principale) sont

Listing 7

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
 <PreferenceCategory
  android:title="Gsm numbers information"
  android:summary="Destination phone number" >
 <EditTextPreference
   android:key="numberToCallAfterShaking"
   android:title="Phone Number to Call after shaking"
   android:summary="Called after shaking your device" />
 <EditTextPreference
   android:key="numberToCallInLocation"
   android:title="Phone Number to Call in location X"
   android:summary="Called when your device is in stored location" />
</PreferenceCategory>

<PreferenceCategory
  android:title="Accelerometer Settings"
  android:summary="Set the shaking sensibility" >

 <CheckBoxPreference
   android:key="chbShaking"
   android:title="Call Shaking Enable"
   android:summary="On/Off"
   android:defaultValue="true" />
```

Figure 6 : visualisation de la configuration Android.

```
<EditTextPreference
    android:key="shakeDuration"
    android:title="Shake Duration [ms]"
    android:summary="start call after this time"
    android:defaultValue="600"
   android:dependency="chbShaking" />

  <EditTextPreference
    android:key="shakeCount"
    android:title="Shake Count"
    android:summary="start call after count shake"
    android:defaultValue="4"
          android:dependency="chbShaking" />

 </PreferenceCategory>

 <PreferenceCategory
  android:title="Location Settings"
  android:summary="Set the calling in location" >

  <CheckBoxPreference
    android:key="chbLocation"
    android:title="Call Location Enable"
    android:summary="On/Off"
    android:defaultValue="true" />

  <ListPreference
    android:key="listProviderLocation"
    android:title="Location Provider"
    android:summary="Choice GPS or Network"
    android:entries="@array/listOptions"
    android:entryValues="@array/listValues" />

 </PreferenceCategory>

</PreferenceScreen>
```

également accessibles au sein de notre « **service** » où elles ont un effet.

Les modifications effectuées en interne dans « **Preferences** » sont automatiquement sauvegardées dans le système et nous pouvons les retrouver lors d'un prochain redémarrage de l'application sans aucune intervention de notre part.

Nous allons maintenant créer notre fenêtre « **Preferences** » qui représentera notre interface graphique. Pour cela nous devons créer un fichier de type « **XML** » (dans notre cas il se nomme « prefs.xml ») dans le dossier « **xml** » contenu dans le dossier système « **res** ».

Si celui-ci n'est pas déjà présent, nous pouvons le créer à partir du navigateur et actualiser le projet. Une partie de ce fichier est visible dans le listing 7.

Comme vous pouvez le voir, la philosophie est la même que celle du fichier « layout » (mise en page), mais cette fois il ne faut pas se préoccuper de la taille et de la position des objets graphiques. Ceux-ci sont principalement de 3 types :

- « **CheckBoxPreference** » utilisé pour les valeurs booléennes ;
- « **EditTextPreference** » utilisé pour les chaînes ;
- « **ListPreference** » utilisé pour sélectionner des valeurs d'une liste.

Chacun de ces objets est représenté par un **titre** (ou en-tête) principal (**android:title**), une **description** (résumé) qui apparaîtra sous le titre (**android:summary**) et une **clé** (**android:key**) qui doit être unique pour chaque contrôle, ainsi que d'autres champs optionnels tels que « **android:defaultValue** » pour définir la valeur par défaut et « android: dependency » qui sera imposée avec une clé de contrôle « **CheckBoxPreference** », indiquant que l'objet graphique sera activé ou non en fonction de la valeur de la « **checkbox** ».

Chacun de ces objets peut également être regroupé au sein d'une catégorie à l'aide d'une balise (tag) de type « **XML** » dénommée « **PreferenceCategory** », et qui dispose également d'un champ « **titre** » et d'un champ « **résumé** ».

Quant à l'objet « **ListPreference** », il mérite une discussion séparée car il aura un champ contenant des éléments de la liste (android:entries) et d'un autre champ (android:entryvalues) contenant les valeurs renvoyées en fonction de l'élément correspondant.

Nous choisissons ces valeurs à partir d'un fichier de type « **XML** » séparé (dans notre cas « array.xml ») qui contient ces tableaux, et que nous allons ajouter dans le dossier « **values** » contenu dans le dossier « **res** » comme le montre la figure 7.

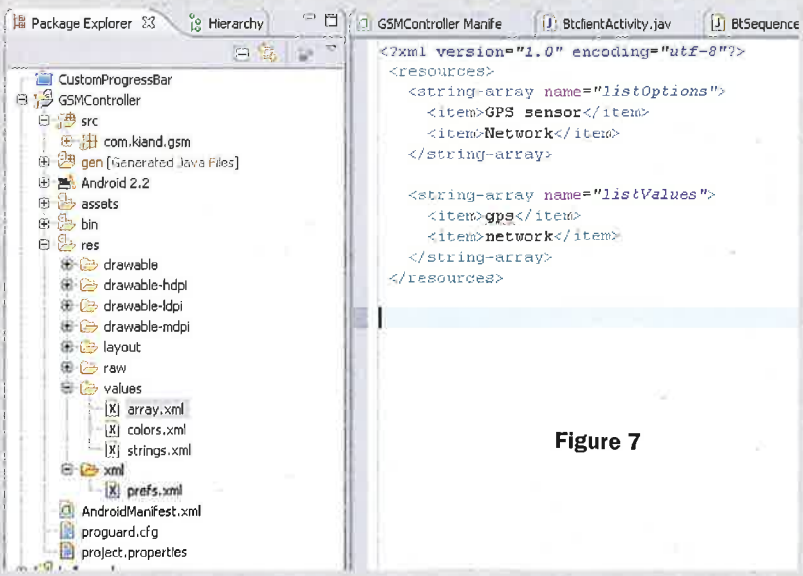


Figure 7

Avant d'entrer dans les détails, vous pouvez voir en figure 10 un aperçu de la représentation graphique.

Il est maintenant temps d'associer l'interface « **XML** » à une nouvelle « **Activity** », pour cela nous allons créer un nouveau fichier et une nouvelle classe qui s'appellera « **GSMPreferences** » et qui sera une classe de la catégorie supérieure « **PreferenceActivity** ».

Dans la méthode « **onCreate()** » de la figure 8, nous utilisons la fonction « **addPreferencesFromResources()** » où nous passons notre fichier « **XML** » créé précédemment (prefs.xml).

Notez que l'importation de « **PreferenceActivity** » est nécessaire pour la compilation de la classe.

Cette classe, comme vous pouvez le voir, est très courte, car elle ne contient que l'association avec le fichier « **XML** ». Mais toujours dans « **onCreate()** » nous ajoutons le code pour accéder aux objets graphiques de « **Preferences** » et pour contrôler la cohérence des valeurs saisies.

Au lieu d'utiliser la méthode classique « **findViewById()** » que nous avons utilisé avec l'« **Activity** », il suffit d'utiliser la méthode « findPreference() » qui fait « passer » la clé (android:key) de l'objet.

Pour obtenir le contrôle, nous pouvons aussi utiliser une classe « **Listener** » (setOnPreferenceChangeListener) que nous implémentons et qui sera appelée par le système dès que le contrôle sera réglé à une certaine valeur.

Maintenant nous devons juste créer un bouton ou un élément de menu (voir les articles précédents) à partir duquel nous allons lancer notre classe « **Preferences** ».

GSMController Manife | main.xml | GSMControllerActivit | GsmPreferences.java

```
package com.kiand.gsm;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class GSMPreferences extends PreferenceActivity
{
        @Override
        protected void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.prefs);
        }
}
```

Figure 8

```
        <activity
            android:label="@string/app_name"
            android:name=".GSMControllerActivity"
            android:screenOrientation="portrait" >

            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:enabled="true" android:name=".GSMService" />

        <activity
            android:name=".GSMPreferences"
            android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen"
            android:label="Settings">
        </activity>

    </application>
```

Figure 9

Figure 10

Listing 8

```
private void showUserSettings()
{
SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);

        String numberAfterShaking = sharedPrefs.getString("numberToCallAfterShaking", "111111");
        String numberAfterLocation = sharedPrefs.getString("numberToCallInLocation", "222222");
        boolean checkBoxShake = sharedPrefs.getBoolean("chbShaking", false);
        boolean checkBoxLocation = sharedPrefs.getBoolean("chbLocation", false);
        long shakeDuration = sharedPrefs.getLong("shakeDuration", 1000);
        int numberOfShakeCount = sharedPrefs.getInt("shakeCount",  4);
        int radius = sharedPrefs.getInt("radius", _defaultRadius);
}
```

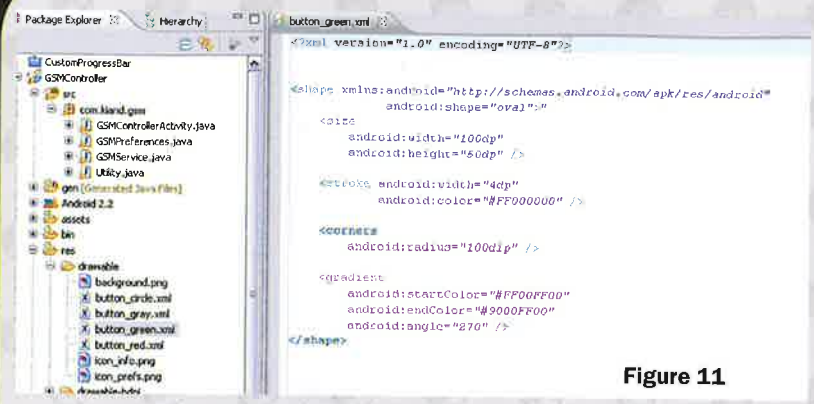


Figure 11

Pour cela nous allons utiliser de manière habituelle l'« **Activity** », en créant un nouvel « **Intent** » avec le nom de classe (GSMPreferences.class) et en utilisant la méthode « **StartActivity()** », comme dans les lignes suivantes :

```
Intent iPrefs = new Intent(this, GSMPreferences.class);
startActivity(iPrefs);
```

Comme c'est une « **Preferences** » d'une « **Activity** », nous ne devons pas oublier d'insérer dans le fichier « **AndroidManifest.xml** » les lignes de codes comme le montre la figure 9.

Figure 12

Enfin, nous pouvons également personnaliser l'apparence graphique de « **Preferences** », en utilisant le champ « android:theme » au sein de l'« **Activity** » comme par exemple :

```
android:theme="@android:style/Theme.Dialog"
```

et

```
android:theme="@android:style/Theme.Light.NoTitleBar.Fullscreen"
```

Les deux ont des effets différents comme vous pouvez le voir en figure 10.

Enfin, nous voyons la facilité d'accéder aux valeurs modifiées dans « **Preferences** », ou dans n'importe quelle « **Activity** » ou « **service** ». Il suffit de prendre un objet appartenant à la classe « **SharedPreferences** » et d'utiliser ces méthodes pour obtenir des variables de type entier, de type « string » ou « booléen » des contrôles associés à l'utilisation des touches que nous avons insérées dans le champ « **android:key** » du fichier « prefs.xml ».

Un exemple est donné dans le Listing 8, la fonction « **showUserSettings()** » peut être appelée dans chaque fichier du projet.

Des boutons personnalisés

Enfin, pour personnaliser davantage notre application, nous pouvons nous consacrer au style des boutons qui activent et arrêtent le « **service** ». Nous avons étudié dans les articles précédents comment changer la couleur d'une **barre de progression** (« SeekBar ») en ajoutant un fichier de type « **XML** » dans le dossier « drawable ».

Maintenant, si nous voulons donner une forme particulière à nos boutons, nous pouvons suivre la même procédure en modifiant la balise « **forme** » dans le nouveau fichier « **button_green.xml** » qui impose la valeur « ovale ». Nous pouvons changer également les valeurs « largeur » (width) et « hauteur » (height) de la balise « taille » afin d'obtenir carrément un rond. Vous pouvez trouver un exemple en figure 11.

Dans notre cas, nous avons créé 3 fichiers « **XML** » correspondant à de nouveaux boutons « **XML** » qui seront actifs lorsque le champ « android:background » sera paramétré par la balise « **Button** » de notre fichier « **Layout** ». De la même manière nous pouvons créer des effets d'ombre et de couleur sur la « SeekBar » pour simuler une pression d'une touche. Le résultat final est visible en figure 12.

Bien que l'accéléromètre et la localisation GPS fonctionnent au niveau du « **service** », nous avons ajouté les mêmes classes dans la classe principale « **Activity** » afin de faciliter le débogage. Le bouton « Store Location » permet de mémoriser la position actuelle qui deviendra le point de référence et qui déclenchera un appel vers Arduino chaque fois que nous nous trouverons dans la même zone.

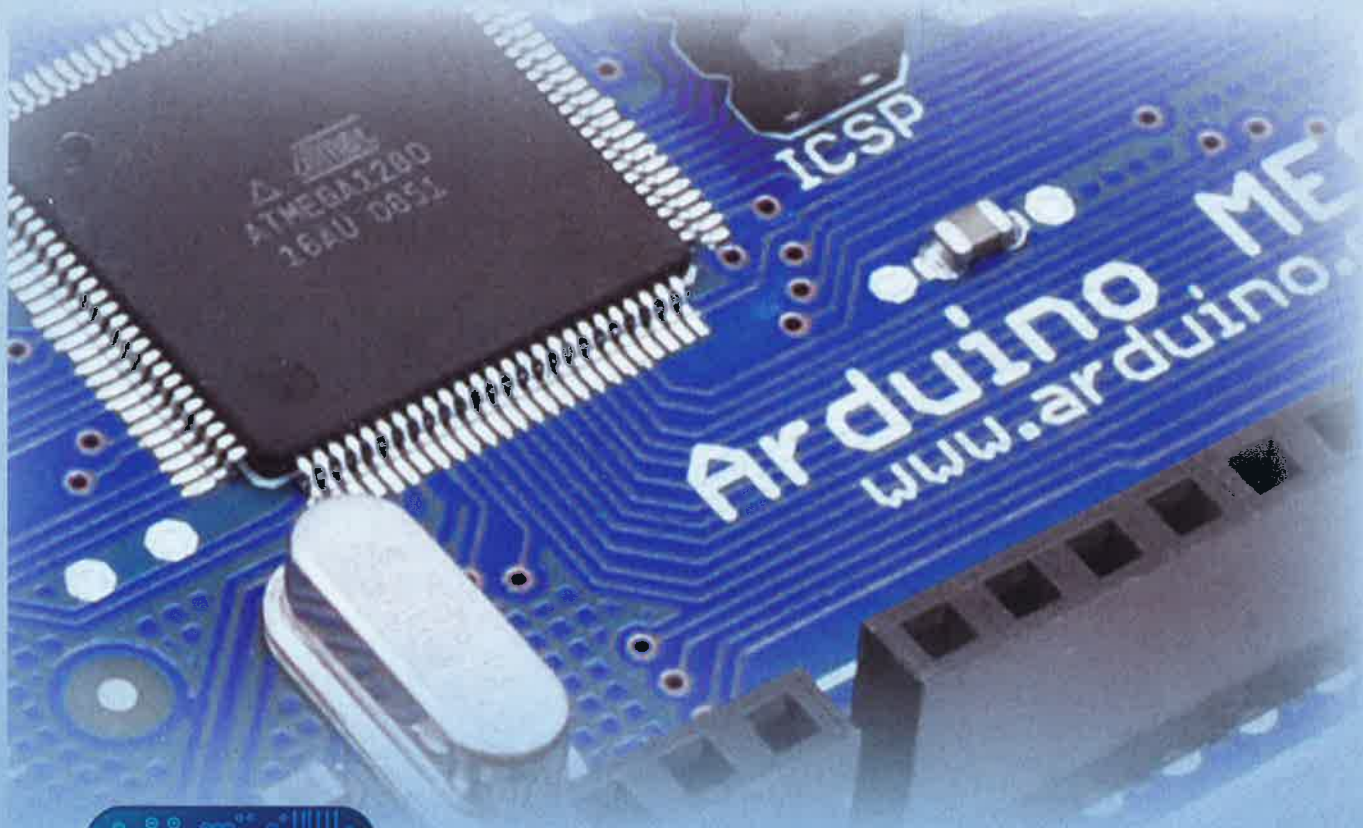
Le passage de cette donnée de l'« **Activity** » au « **service** » est réalisé à travers le mécanisme des méthodes « **putExtra()** » et « **getExtra()** » de l'« **Intent** » associé, comme indiqué dans l'encadré intitulé " **Passage des paramètres entre des « Activity »** et des « services » " (voir plus haut). De même, nous pourrions utiliser la classe « **SharedPreferences** » comme illustré dans le même encadré.

Conclusion

Maintenant, nous avons les outils nécessaires pour réaliser une application qui réponde à des événements externes détectés par les capteurs et qui communique ces informations vers des dispositifs distants. Tout cela est rendu encore plus confortable grâce à l'utilisation du « service » qui ne nous oblige pas à surveiller l'application.

Dans le prochain Cours Android, nous allons créer un « Widget » simple qui nous permettra de contrôler et de surveiller (feedback) les dispositifs distants connectés à la carte Arduino GSM directement à partir de l'écran du smartphone.

Cours ARDUINO - 3



Cours ARDUINO

Troisième partie

de Mirco SEGATELLO

Nous allons étudier, dans cette 3ème partie du Cours Arduino, la gestion des afficheurs LCD alphanumériques et graphiques, à l'aide de quelques exemples de « sketch » (programmes) valables pour les modèles d'afficheurs les plus courants dans le commerce.

Cette partie du Cours est consacrée aux **afficheurs LCD** aussi bien **alphanumériques** que **graphiques**. Nous allons faire un tour d'horizon des applications avec divers types d'afficheurs, toujours en vous proposant des exemples pratiques très simples qui concernent aussi bien la partie matérielle (hardware) que logicielle (software).

Les premiers afficheurs décrits seront de type alphanumérique, c'est-à-dire qu'ils affichent essentiellement du texte. Ils sont incontournables dans de nombreuses applications et très faciles à utiliser grâce aux librairies existantes.

Avant d'entrer dans les détails, faisons une brève introduction pour ce type d'afficheur, pas d'un point de vue fabrication mais plutôt de leur utilisation.

Commençons par dire que l'affichage du texte de l'écran LCD est basé sur un composant (puce) qui est en fait un **décodeur** spécialisé contenu dans l'afficheur lui-même.

Celui-ci lit les données entrantes arrivant sur l'afficheur et gère l'affichage des pixels sur l'écran de manière à obtenir les symboles ou lettres désirés. C'est précisément le **type de décodeur utilisé dans l'afficheur LCD qui déterminera son utilisation et ses fonctions, et non la taille ou le nombre de lignes**. Il est donc important, lorsque vous achetez un afficheur LCD, de **vérifier le type de décodeur** présent, car de lui va dépendre la gestion de l'afficheur au niveau du programme.

Au niveau matériel les choses sont plus compliquées, car même si chaque fabricant utilise le même circuit de décodage, les afficheurs ont un brochage différent les uns des autres, si bien que sans leur documentation technique ils sont inutilisables.

Dans ce **Cours Arduino**, nous allons utiliser uniquement des afficheurs LCD basés sur les contrôleurs **Hitachi** « **HD44780** » ou **Samsung** « **ST7066U** ».

Il s'agit d'**afficheurs LCD ayant une interface parallèle** et pour lesquels il existe beaucoup d'environnements de développements pour différents types de microcontrôleurs comprenant déjà les commandes de contrôle, cela évite de devoir élaborer une routine de contrôle spécifique.

L'interface matérielle est standard et comprend le brochage résumé dans le **Tableau 1**. En plus de ces lignes (broches), des contacts supplémentaires (BL+ e BL-) sont disponibles pour le **rétroéclairage** afin d'utiliser l'afficheur dans un endroit sombre.

Évidemment, les noms peuvent changer d'un fabricant à l'autre, ainsi que la dénomination des broches (pin) du connecteur de l'afficheur, mais la fonction reste la même.

Passons maintenant à un exemple pratique, en utilisant un afficheur basé sur le circuit Hitachi, dont le connecteur est situé en haut et le brochage est indiqué dans le **Tableau 2**.

Dans cet afficheur, le **rétroéclairage** (**backlight**) est activé en connectant simplement la broche « **BL-** » à la **masse** (GND) et la broche « **BL+** » vers le **+5 V** à travers une **résistance** de **20** Ω à **50** Ω en fonction de l'intensité lumineuse désirée du rétroéclairage.

À ce stade, nous sommes prêts à réaliser le câblage physique de l'afficheur avec une carte Arduino. Pour cela il existe plusieurs possibilités, par exemple l'utilisation d'une carte Arduino « Proto Shield » ou d'une plaque d'essai. L'important étant de respecter le schéma de câblage visible en figure 1.

Etudions un exemple en utilisant un afficheur LCD dont la référence est « **CDL4162** » et qui est disponible dans la plupart des magasins d'électronique ou sur le web.

Il s'agit d'un afficheur comprenant **2 lignes de 16 caractères chacune**. La correspondance entre les broches et les fonctions est donnée dans le **Tableau 3**.

Pour le rétroéclairage vous pouvez vous référer au paragraphe ci-dessus. Pour le câblage, vous devez vous référer au schéma électrique visible en figure 2.

Dans tous les cas, vous devez effectuer de la manière suivante les connexions entre la carte Arduino et l'afficheur LCD :

- broche « RS » du LCD avec la broche 12 d'Arduino ;
- broche « RW » du LCD à la masse (GND) ;
- broche « Enable » du LCD avec la broche 11 d'Arduino ;
- broche « D4 » du LCD avec la broche 5 d'Arduino ;
- broche « D5 » du LCD avec la broche 4 d'Arduino ;
- broche « D6 » du LCD avec la broche 3 d'Arduino ;
- broche « D7 » du LCD avec la broche 2 d'Arduino.

En plus de cela, vous devez connecter l'alimentation et le trimmer de réglage du contraste. Les broches D0, D1, D2, D3 peuvent être laissées libres (en l'air) ou connectées à la masse.

Tableau 1

+ 5V	Alimentation positive
GND	Masse
VLCD	Tension négative pour le réglage du contraste
Data	8 lignes de données
WR/RD	Ligne pour la lecture (1) / écriture (0) sur LCD
E	Commande d'activation de l'afficheur
RS	Ligne pour sélectionner des données ou des commandes

Tableau 2

Broche	Signal	Fonction
1	VSS	Alimentation (0 V)
2	VDD	Alimentation (5 V)
3	VO	Contraste (de 0 V à VDD)
4	RS	Si à 1 entrée d'une instruction, si à 0 entrée de données
5	R/W	Si à 1 lecture des données Si à 0 écriture des données
6	E	Signal d'activation
7	DB0	Lignes du bus de données
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	BL-	Alimentation négative du rétroéclairage LED (-)
16	BL+	Alimentation positive du rétroéclairage LED (+)

À ce point vous devez avoir votre afficheur LCD correctement câblé avec le microcontrôleur qui n'est pas encore programmé. Si vous allumez Arduino, vous ne remarquerez rien sur l'afficheur, peut-être des pixels partiellement allumés. À noter que le contraste n'est même pas réglé.

Pour être complet, nous allons décrire les fonctions de la **librairie** pour un afficheur LCD alphanumérique. Cette **librairie permet à une carte Arduino de contrôler un afficheur LCD alphanumérique standard** à cristaux liquides basé sur le circuit intégré **Hitachi HD44780** (ou compatible), ce qui est notre cas.

La librairie fonctionne aussi bien en **mode 4 bits** qu'en **mode 8 bits** (c'est-à-dire en utilisant 4 ou 8 broches de données D0.....D7 en plus des broches de contrôle RS, Enable et RW).

En **mode 4 bits, 6 broches de la carte Arduino suffisent pour contrôler un afficheur LCD**. Nous reportons ci-après toutes les fonctions disponibles lors de la programmation avec quelques exemples de programmes présents dans l'environnement de développement.

Tout d'abord commençons avec le constructeur de classe « **LiquidCrystal** ». Il permet de **créer une variable** de type « **LiquidCrystal** » en définissant les broches utilisées du LCD et son mode de fonctionnement. L'afficheur LCD peut être contrôlé avec 4 ou 8 lignes de données.

Tableau 3

Broche	Signal	Fonction
1	BL+	Broche d'alimentation du rétroéclairage LED (+)
2	BL-	Broche d'alimentation du rétroéclairage LED (-)
3	GND	Alimentation (0 V)
4	VDD	Alimentation (5 V)
5	VO	Contraste LCD
6	RS	Si à 1 entrée d'une instruction, si à 0 entrée de données
7	R/W	Si à 1 lecture des données Si à 0 écriture des données
8	E	Signal d'activation
9	DB0	Lignes du bus de données
10	DB1	
11	DB2	
12	DB3	
13	DB4	
14	DB5	
15	DB6	
16	DB7	

En **mode 4 bits** (**D4 à D7**), vous devez laisser les broches D0 à D3 non-connectées. La broche **RW** peut-être connectée à la **masse** (0 V) au lieu d'être connectée à une broche d'Arduino.

Syntaxe :

```
LiquidCrystal LCD(RS, E, D4, D5, D6, D7);
                    // mode 4 bits - RW non connectée

LiquidCrystal LCD(RS, RW, E, D4, D5, D6, D7);
                    // mode 4 bits - RW utilisée

LiquidCrystal LCD(RS, E, D0, D1, D2, D3, D4, D5, D6, D7);
                    // mode 8 bits - RW non connectée

LiquidCrystal LCD(RS, RW, E, D0, D1, D2, D3, D4, D5, D6, D7);
                    // mode 8 bits - RW utilisée
```

- **LCD** : le nom de la variable LiquidCrystal déclarée ;
- **RS** : le numéro de la broche d'Arduino où est connectée la broche RS du LCD ;
- **RW** : le numéro de la broche d'Arduino où est connectée la broche RW du LCD (option) ;
- **E** : le numéro de la broche d'Arduino où est connectée la broche E (Enable) du LCD
- **D0, D1, D2, D3, D4, D5, D6, D7** : les numéros des broches d'Arduino où sont connectées les broches de données du LCD. Les broches D0, D1, D2, D3 sont optionnelles.

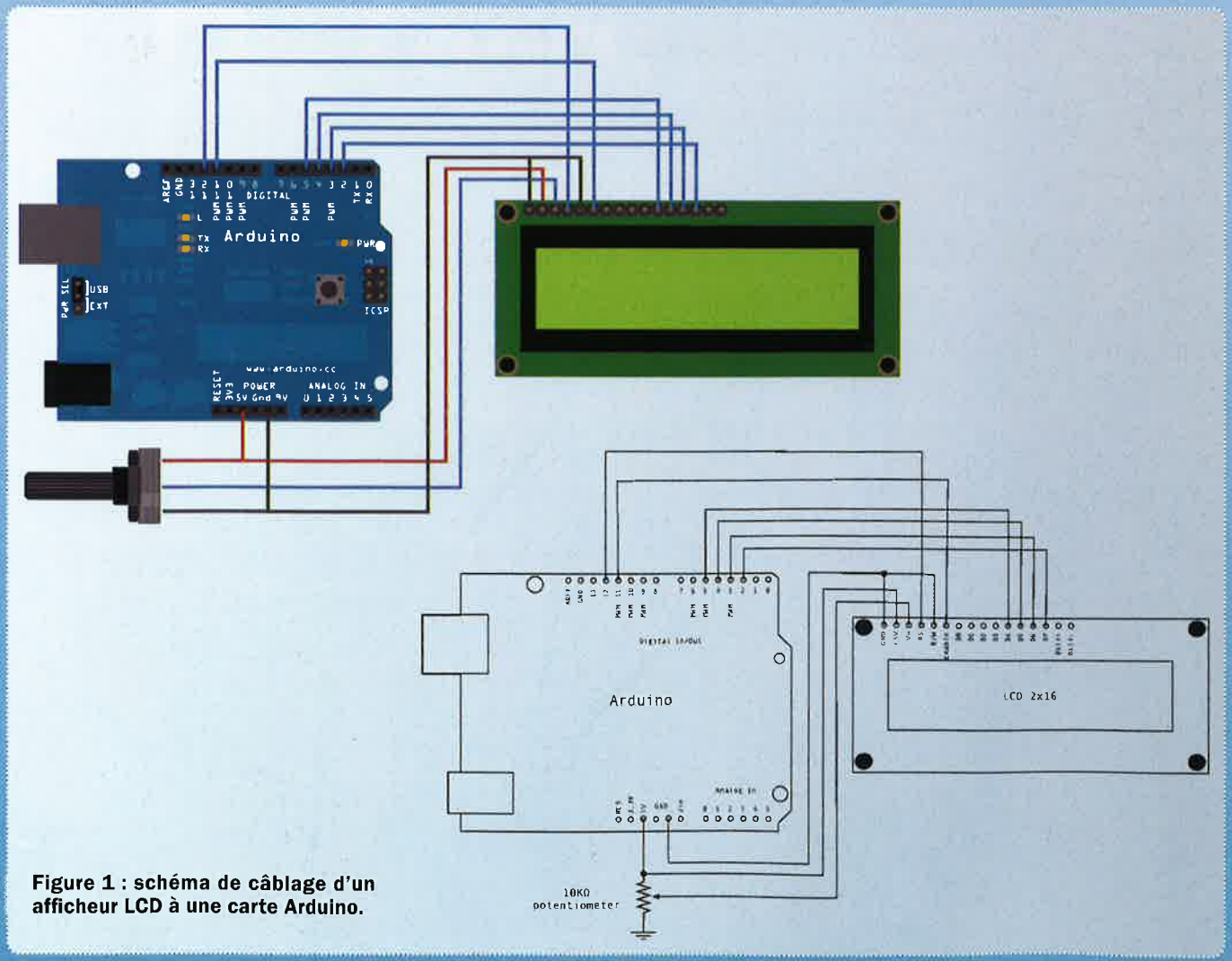


Figure 1 : schéma de câblage d'un afficheur LCD à une carte Arduino.

Fonctions d'initialisation :

begin() : spécifie les dimensions de l'afficheur LCD (nombre de colonnes et nombre de lignes).

Syntaxe : LCD.begin(colonnes, lignes)

Fonctions de gestion de l'écran :

clear() : efface l'écran et positionne le curseur dans le coin supérieur gauche.

Exemple :

```
#include <LiquidCrystal.h>

LiquidCrystal LCD(12, 11, 5, 4, 3, 2);
// déclare une variable LiquidCrystal appelée LCD
// mode 4 bits et RW pas utilisé

void setup()
{
 LCD.print("Bonjour");      // affiche le texte "Bonjour"
}
void loop() {}
```

Syntaxe : LCD.clear()

display() : rallume l'écran LCD après qu'il ait été éteint avec l'instruction noDisplay(). Le texte et le curseur s'affichent tels qu'ils étaient auparavant.

Syntaxe : LCD.Display()

Exemple :

```
LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup()  {    // début de la fonction setup()

LCD.begin(20,4)  // initialise le LCD avec 20 colonnes et 4 lignes
}
```

noDisplay() : éteint l'écran du LCD sans perdre le texte affiché.

Syntaxe : LCD.noDisplay()

Fonctions d'écriture :

write() : écrit un caractère sur l'afficheur LCD.

Syntaxe : LCD.write(data) où « data » représente un caractère.

print() : affiche du texte et des nombres sur le LCD.

Syntaxe :

LCD.print("texte") où « texte » est une chaîne de caractères.
LCD.print(data) où « data » est la variable à afficher (char, byte, int, long, float ou string).
LCD.print(data, BASE) où « BASE » est la base dans laquelle vous voulez afficher la variable : BIN pour binaire, DEC pour décimal, OCT pour octal et HEX pour hexadécimal.

Exemple :

```
#include <LiquidCrystal.h>

LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup()
{
 Serial.begin(9600);      // initialise une communication série à 9600 bps
}

void loop()
{
  if (Serial.available()) {          // si un caractère est disponible sur le port série
   LCD.write(Serial.read()); // écrit le caractère reçu sur le LCD
  }
}
```

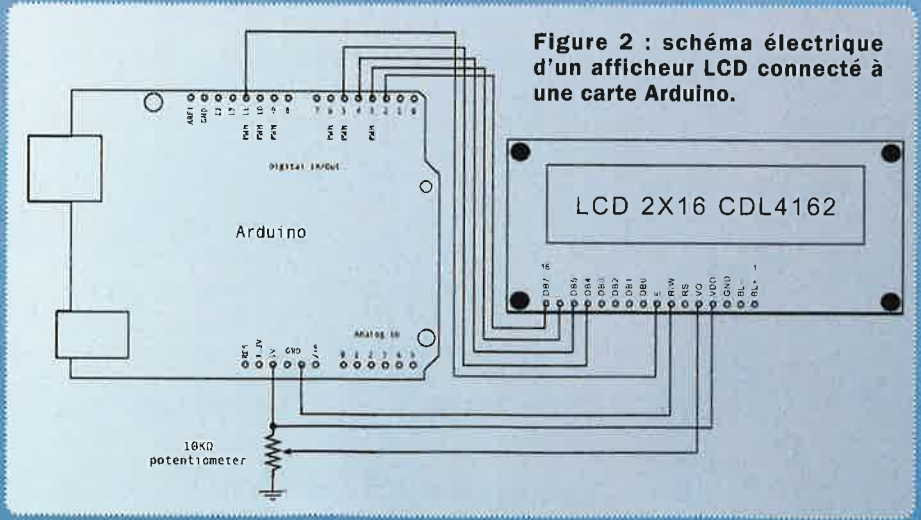


Figure 2 : schéma électrique d'un afficheur LCD connecté à une carte Arduino.

Fonctions de positionnement du curseur :

home() : positionne le curseur dans le coin gauche du LCD. Le texte est écrit à partir de cette position, pour effacer l'écran, utilisez plutôt l'instruction clear().

Syntaxe : LCD.home

setCursor() : positionne le curseur du LCD à la localisation voulue (colonne, ligne), c'est-à-dire définit la position à laquelle le texte est affiché.

Syntaxe : LCD.setCursor(colonne, ligne) où colonne représente la colonne dans

Exemple d'écriture :

```
#include <LiquidCrystal.h>

LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup()
{
 LCD.print("Bonjour");        // affiche le texte "Bonjour"
}
```

laquelle positionner le curseur (0 = 1ère colonne, 1 = 2ème colonne, etc.) et ligne représente la ligne sur laquelle positionner le curseur (0 = 1ère ligne, 1 = 2ème ligne, etc.).

Fonctions modifiant l'aspect du curseur :

cursor() : affiche le curseur sous la forme d'un « underscore » (tiret bas aussi appelé sous-tiret ou tiret du 8) sous l'emplacement actuel du curseur, là où sera écrit le prochain caractère.

Syntaxe : LCD.cursor()

noCursor() : masque le curseur

Syntaxe : LCD.noCursor()

Exemple de positionnement du curseur :

```
// --- Inclusion des librairies utilisées ---
#include <LiquidCrystal.h>   // inclusion de la librairie pour afficheur LCD

// --- déclaration des constantes ---

const int RS=12;
const int E=11;
const int D4=5;
const int D5=4;
const int D6=3;
const int D7=2;

// --- Initialisation des fonctionnalités utilisées ---

LiquidCrystal LCD(RS, E, D4, D5, D6, D7);   // initialisation LCD en mode 4 bits

void setup()  {          // début de la fonction setup()

LCD.begin(20,4);     // initialise un LCD de 20 colonnes et 4 lignes
delay(10);               // pause pour laisser un temps d'initialisation

// Test du LCD
LCD.print("LCD OK") ;   // affiche le message de test « LCD OK »
delay(2000);             // pause de 2 secondes

LCD.clear();             // efface l'écran et met le curseur en haut à gauche
delay(10);               // pause pour laisser le temps d'effacer l'écran
}                        // fin de la fonction setup()

void loop(){             // début de la fonction loop()

LCD.print("Electronique") ; // affiche le texte « Electronique»
delay(2000);             // pause de 2 secondes

LCD.setCursor(9, 1) ;    // positionne le curseur dans la 10ème colonne et sur la 2ème ligne
LCD.print("et") ;        // affiche le texte « et »
delay(2000);             // pause de 2 secondes

LCD.setCursor(4, 2) ;    // positionne le curseur dans la 5ème colonne  et sur la 3ème ligne
LCD.print("Loisirs") ;   // affiche le texte « Loisirs »
delay(2000);             // pause de 2 secondes

LCD.setCursor(12, 3) ;   // positionne le curseur dans la 13ème colonne  et sur la 4ème ligne (1ère=0)
LCD.print("Magazine") ;  // affiche le texte « Magazine »
delay(2000);             // pause de 2 secondes

LCD.clear();             // efface l'écran et met le curseur en haut à gauche
delay(10);               // pour laisser le temps d'effacer écran
}
```

blink() : active le clignotement du curseur à la position actuelle , si cette instruction est utilisée avec l'instruction cursor(), le résultat dépendra de l'afficheur.

Syntaxe : LCD.blink()

noBlink() : désactive le clignotement du curseur.

Syntaxe : LCD.noBlink()

Fonctions contrôlant le comportement du curseur :

autoscroll() : effectue un décalage automatique de 1 espace des caractères précédents déjà affichés à chaque nouveau caractère envoyé au LCD. Si le sens de défilement est de gauche à droite (par défaut), l'écran décale vers la gauche, et inversement si le sens de défilement est de droite à gauche, le décalage est vers la droite. Tout se passe comme si le curseur restait fixe, mais le texte se décale.

Syntaxe : LCD.autoscroll()

Exemple résumant l'aspect du curseur :

```
#include <LiquidCrystal.h>
LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup() {

 LCD.begin(16, 2); // initialise le nombre de colonnes et
```

```
de lignes du LCD
  LCD.print("Bonjour");  // affiche « Bonjour » à l'écran
}
void loop() {

  LCD.blink();      // Active le clignotement
  delay(3000);      // pause 3 s

  LCD.noBlink();    // Désactive le clignotement
  delay(3000);      // pause 3 s
}
```

noAutoscroll() : désactive le décalage automatique du LCD.

Syntaxe : LCD.noAutoscroll()

leftToRight() : initialise la direction d'écriture du texte sur le LCD de gauche à droite (valeur par défaut au démarrage).

Les caractères suivant cette instruction sont écrits sur l'afficheur de gauche à droite, les caractères précédents déjà affichés ne sont pas affectés.

Syntaxe : LCD.leftToRight()

rightToLeft() : initialise la direction d'écriture du texte sur le LCD de droite à gauche (la direction par défaut est de gauche à droite).

Exemple résumant le comportement du curseur :

```
#include <LiquidCrystal.h>

LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup() {

  LCD.begin(16, 2);          // initialise le nombre de colonnes et de lignes du LCD

void loop() {

LCD.setCursor(0, 0);         // positionne  le curseur à 0,0

  for (int thisChar = 0; thisChar < 10; thisChar++) {
                             // affiche les chiffres de 0 à 9
    LCD.print(thisChar);
    delay(500);              // pause
  }

LCD.setCursor(1,1);          // positionne le curseur en (1,1) (2ème colonne, 2ème ligne)

 LCD.autoscroll();           // initialise l'affichage autoscroll
 for (int thisChar = 0; thisChar < 10; thisChar++) {   // affiche les chiffres de 0 à 9
 LCd.print(thisChar);
    delay(500);
}
 LCD.noAutoscroll();         // désactive l'autoscroll
 LCD.clear();                // efface l'écran avant de recommencer
}
```

Les caractères suivant cette instruction sont écrits sur l'afficheur de droite à gauche, les caractères précédents déjà affichés ne sont pas affectés.

Syntaxe : LCD.rightToLeft()

Fonctions d'effets visuels :

scrollDisplayLeft() : décale le contenu de l'afficheur, texte et curseur, d'un espace vers la gauche.

Syntaxe : LCD.scrollDisplayLeft()

scrollDisplayRight() : décale le contenu de l'afficheur, texte et curseur, d'un espace vers la droite.

Syntaxe : LCD.scrollDisplayRight()

Fonction de création de caractère personnalisé :

createChar() : crée un caractère personnalisé, jusqu'à 8 caractères de 8 x 5 pixels sont supportés (ils sont numérotés de 0 à 7). L'apparence de chaque caractère personnalisé est définie par un tableau de 8 octets, un pour chaque ligne. Les 5 bits de poids faible de chaque octet définissent les

Exemple résumant le comportement du curseur :

```
#include <LiquidCrystal.h>

LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup() {
  LCD.begin(16, 2);  // initialise le nombre de colonnes et de lignes du LCD

void loop() {
   if (thisChar == 'e') {     // inverse la direction à la lettre 'e'
    LCD.rightToLeft();    // la lettre suivante va vers la droite
  }

 if (thisChar == 's') {     // inverse la direction à la lettre 's'

LCD.leftToRight();      // la lettre suivante va vers la gauche
  }
   if (thisChar > 'z') {           // réinitialisation à 'z':

   LCD.home();                   // revient en (0,0):

   thisChar = 'a';               // recommence à 'a'
  }
 LCD.print(thisChar, BYTE);  // affiche le caractère

   delay(1000);
}
```

pixels affichés dans chaque ligne. Pour afficher un caractère personnalisé sur l'écran, il faut utiliser l'instruction « write(num) » où « num » est le numéro du caractère (de 0 à 7).

Syntaxe : LCD. createChar()

Tout d'abord il faut déclarer un nouveau caractère :

```
byte nouvChar[8]={
  B10000,
  B01000,
  B00100,
  B00010,
  B00001,
  B00010,
  B00100,
  B00010
};
```

La 1ère ligne de code B10000 correspond à la 1ère ligne d'un caractère de l'afficheur, la 2ème ligne de code B01000 correspond à la 2ème ligne d'un caractère de l'afficheur et ainsi de suite. Si vous voulez allumer un pixel, il suffit de mettre un « 1 » et si vous voulez l'éteindre, il faut mettre un « 0 ». Vous pouvez créer 8 caractères différents comme cela avec la fonction « createChar() ». Vous devez donner un nom et un numéro à chaque nouveau caractère et sélectionner un caractère avec la fonction « SetCursor() ». Voici un exemple de code.

Des exemples de programmes pour afficheur LCD sont présents dans l'environnement Arduino. Pour cela lancez l'IDE Arduino, cliquez sur « **Fichier** » → « **Exemples** » → « **LiqudCrystal** ». Vous verrez apparaître une liste d'exemples de codes. Par exemple ouvrons le programme « HelloWorld » en cliquant dessus. Le sketch (programme) auquel nous faisons référence est visible dans le **Listing 1**.

Compilez et téléversez le sketch dans la carte Arduino, ensuite réglez le contraste de l'afficheur en ajustant le potentiomètre jusqu'à ce que vous voyiez apparaître le texte « hello, world! ».

Cet exemple est très important car il vous familiarise avec la manière d'afficher les variables. La variable « **millis ()** » contient le nombre de millisecondes écoulées depuis l'exécution du programme dans Arduino (ou depuis un reset). En la divisant par 1000 nous obtenons le nombre de secondes écoulées depuis le lancement du programme.

Notez que la variable est écrite dans la mémoire sous forme binaire, mais elle affichée sous la forme d'un nombre décimal grâce à une conversion automatique.

Exemple résumant les effets visuels :

```
#include <LiquidCrystal.h>

LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

void setup() {

  LCD.begin(16, 2);      // initialise le nombre de colonnes et de lignes du LCD

  LCD.print("Bonjour");  // affiche « Bonjour » à l'écran
 delay(200);

}

void loop() {

    for (int positionCounter = 0; positionCounter < 6; positionCounter++) {   // décale de 6 positions vers la gauche

   LCD.scrollDisplayLeft();         // décale d'1 position vers la gauche
    delay(150);
  }

  for (int positionCounter = 0; positionCounter < 5 ; positionCounter++) {    //décale de 5 positions vers la droite

  LCD.scrollDisplayRight();     // décale d'1 position vers la droite
       delay(150);
  }

   delay(1000);              // pause d'1 s avant reprise
}
```

Exemple pour 2 nouveaux caractères:

```
#include <LiquidCrystal.h>
LiquidCrystal LCD(12, 11, 5, 4, 3, 2);

//Déclaration des nouveaux caractères

byte nouvChar1[8] = {
    B00000,
    B10001,
    B00000,
    B00000,
    B10001,
    B01110,
    B00000,
};
byte nouvChar2[8]={
  B00000,
  B01010,
  B11111,
  B10101,
  B11111,
  B01010,
  B00100,
  B00000
};

//attribution des numéros des caractères

LCD.createChar(0, nouvChar1);
LCD.createChar(1, nouvChar2);

LCD.begin(16, 2);
    LCD.setCursor(4,0);     //choix de l'emplacement de
```

```
départ pour l'affichage
   LCD.print("Test LCD");
   LCD.setCursor(6,1);
   LCD.print(" LCD OK");

void loop() {
   lcd.display();
   lcd.setCursor(0,0);
   lcd.write(0);
   lcd.setCursor(1,0);
   lcd.write(1);
}
```

Un autre point important est de noter la présence de la ligne « #include <LiquidCrystal.h> » qui vous permet d'inclure la librairie et les commandes nécessaires pour gérer l'afficheur LCD.

De même, vous pouvez importer d'autres librairies, créées par des tiers, afin d'élargir les fonctions disponibles.

Vous pouvez tester tous les autres exemples de programmes liés à l'afficheur LCD. Vous remarquerez qu'il existe différentes manières d'écrire des caractères ou d'agir sur le curseur.

Pour plus de commodités, reportez-vous au **Tableau 4** qui résume les principales fonctions de gestion de l'afficheur que nous avons étudiées précédemment. Nous allons maintenant utiliser des **afficheurs LCD graphiques** (GLCD) qui sont certainement plus fascinants et plus souples que les alphanumériques.

La première chose à dire est que, comme pour les afficheurs alphanumériques, les afficheurs graphiques sont divisés en familles en fonction du décodeur utilisé.

Il existe sur le marché essentiellement 2 grandes familles ou catégories :

- la 1ère est basée sur le décodeur « **KS0107B** » qui se limite seulement à une résolution **128 x 64 pixels**. Ce

Listing 1

```
/*
 Librairie LiquidCrystal - Hello World

Démonstration de l'utilisation d'un afficheur LCD de 16 caractères et de 2 lignes. La librairie LiquidCrystal  fonctionne avec tous les afficheurs compatibles avec le décodeur Hitachi HD44780.

Ce sketch affiche "Hello World!" sur le LCD et indique le temps écoulé depuis la mise en service ou un reset.

Câblage du circuit :

 * Broche RS du LCD à la broche 12 d'Arduino
 * Broche Enable du LCD à la broche 11 d'Arduino
 * Broche D4 du LCD à la broche 5 d'Arduino
 * Broche D5 du LCD à la broche 4 d'Arduino
 * Broche D6 du LCD à la broche 3 d'Arduino
 * Broche D7 du LCD à la broche 2 d'Arduino
 * Broche R/W du LCD à la masse (GND)
 * Broche VSS du LCD à la masse (GND)
 * Broche VCC du LCD au + 5 V
 * Trimmer 10 kΩ
 * extrémités entre +5 V et GND
 * point milieu à la broche VO du LCD (broche 3)

 */
#include <LiquidCrystal.h>                  // inclus la librairie
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);      // initialise la librairie avec les numéros des broches de l'interface

void setup() {
   lcd.begin(16, 2);                        // initialise le nombre de colonnes et de lignes du LCD

  lcd.print("hello, world!");               // affiche le texte « hello, world! » sur le LCD.

}
void loop() {
lcd.setCursor(0, 1);                        // positionne le curseur à la colonne 0 et la ligne 1
                                            // (la ligne 1 est la 2ème ligne, car on compte à partie de 0)
lcd.print(millis() / 1000);                 // affiche les secondes écoulées depuis le RESET
}
```

Tableau 4 : les principales fonctions de la librairie LCD d'Arduino et leur description.

Fonctions	Description
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);	Initialise l'afficheur et déclare comme objet le terme "lcd"pour une utilisation dans le programme
lcd.begin(cols, rows);	Définit pour l'objet "lcd" le nombre de colonnes (cols) et le nombre de lignes (rows)
lcd.cursor();	Affiche le curseur sous la forme d'un « underscore » sur la position actuelle
lcd.noCursor();	Désactive l'affichage du curseur sur la position actuelle
lcd.blink();	Active le clignotement du curseur
lcd.noBlink();	Désactive le clignotement du curseur
lcd.display()	Allume l'afficheur
lcd.noDisplay();	Éteint l'afficheur
lcd.scrollDisplayRight();	Décale le contenu de l'afficheur d'un espace vers la droite
lcd.scrollDisplayLeft();	Décale le contenu de l'afficheur d'un espace vers la gauche

type d'afficheur est assez répandu et peu couteux, mais sa résolution n'est pas très élevée ;

- la 2ème catégorie est basée sur l'utilisation du circuit « **T6963C** » qui permet de gérer des afficheurs avec les résolutions suivantes : **128 x 128 pixels**, **240 x 128 pixels** et **240 x 64 pixels**.

Il existe d'autres modèles d'afficheurs graphiques dans le commerce, mais leurs librairies ne sont pas toujours disponibles et sont de ce fait difficiles à gérer.

Dans un premier temps, nous allons utiliser l'afficheur graphique monochrome « **LM12864MBC** » distribué par « **TOPWAY DISPALY** ». Les caractéristiques sont disponibles à l'adresse :

http://topwaydisplays.eu/FR/LM12864MBC-display-lcd-graphique-monochrome-avec-contrleur-128x64-points::F00285::P004627/.

Broche Arduino	Broche de l'afficheur	Nom	Fonction
GND	**1**	**VSS**	GND
+5V	**2**	**VDD**	+5V
Pin2 point mileur trimmer	**3**	**Vo**	Contraste du LCD
17 (analog3)	**4**	**RS**	Données/Instructions
16 (analog2)	**5**	**R/W**	Lecture/Ecriture
18 (analog4)	**6**	**E**	Enable
8	**7**	**DB0**	Données
9	**8**	**DB1**	Données
10	**9**	**DB2**	Données
11	**10**	**DB3**	Données
4	**11**	**DB4**	Données
5	**12**	**DB5**	Données
6	**13**	**DB6**	Données
7	**14**	**DB7**	Données
14 (analog0)	**15**	**CS1**	Chip select 1
15 (analog1)	**16**	**CS2**	Chip select 2
Reset (mettre à +5 V)	**17**	**RST**	Reset
Pin1 du trimmer Pin3 du trimmer à GND	**18**	**Vout**	Tension pour Vo
+ 5 V du rétroéclairage	**19**	**BLA**	Anode du rétroéclairage
GND	**20**	**BLK**	Cathode du rétroéclairage

Tableau 5 : connexions physiques entre l'afficheur LM12864MBC et la carte Arduino 2009.

Cet afficheur est basé sur un décodeur de type « **KS0107B** » avec une résolution de **128 x 64 pixels**. Comme nous l'avons déjà expliqué pour les afficheurs LCD alphanumériques, vous devez réaliser le câblage électrique et ensuite programmer correctement la carte Arduino. Les connexions sont résumées dans le Tableau 5.

Vu le caractère expérimental de l'application, dans ce cas aussi, nous utilisons une plaque d'essai sur laquelle nous effectuons les connexions avec des fils rigides.

Vous pouvez souder sur l'afficheur un connecteur femelle semblable à ceux de la carte Arduino, et à l'aide de fils rigides réaliser les connexions.

Il est également possible de souder les fils directement sur l'afficheur (nous vous le déconseillons), ou utiliser la carte « Proto Shield » pour effectuer toutes les connexions en soudant les fils.

Cependant, vous devez être très consciencieux et suivre scrupuleusement le Tableau 5, vérifiez plusieurs fois le travail effectué.

Le **trimmer de réglage du contraste**, d'une valeur comprise entre **10 kΩ** et **20 kΩ**, présente 3 broches.

Les 2 broches externes sont connectées l'une à la masse (GND) et l'autre à la broche 18 (Vout) de l'afficheur.

Le **point milieu** du trimmer est relié à la **broche 3** (Vo) de l'afficheur.

Vous devez insérer un condensateur de 100 nF entre le « + 5 V » et la masse (GND), aussi proche que possible de l'afficheur, afin de supprimer les perturbations électriques présentes dans les connexions.

Une fois le câblage effectué correctement, vous devez programmer la carte Arduino. Cependant, vous devez vous procurer une librairie spécifique, développée par des tiers, pour l'afficheur LCD graphique avec un décodeur de type K0108.

Vous pouvez télécharger la librairie « **KS0108 Graphics LCD library** » sur le site officiel Arduino en Anglais.

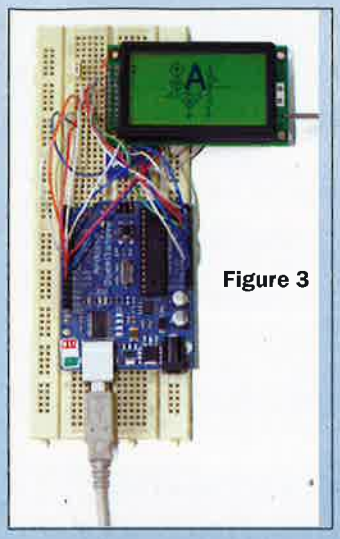


Figure 3

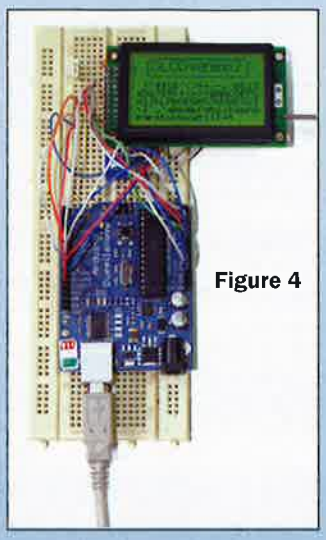


Figure 4

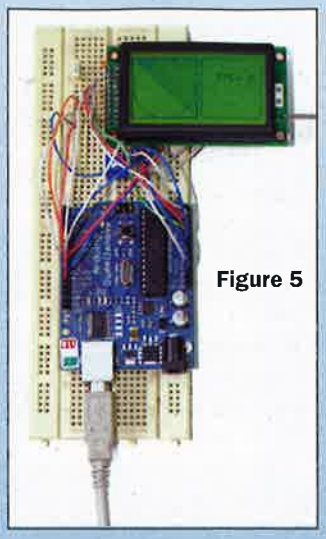


Figure 5

« Processing » → « glcdBitmap ») vous verrez 3 écrans avec dans le 1er une image, dans le 2ème des caractères affichés et dans le 3ème des lignes et des cercles.

Vous pouvez voir chacun de ces résultats sur les figures 3, 4 et 5.

Pour obtenir un bon affichage, vous aurez peut-être besoin d'ajuster le contraste grâce au trimmer. Le sketch décrit ici utilise la librairie GLCD pour le KS0108 permettant d'afficher une image bitmap contenant du texte et des graphiques.

Broche Arduino	Broche de l'afficheur	Nom	Fonction
GND	2	VSS	GND
+ 5 V	1	VDD	+ 5 V
Pin2 point mileu du trimmer	3	Vo	Contraste du LCD
17 (analog3)	16	D/I	Données/Instructions
16 (analog2)	15	R/W	Lecture/Ecriture
18 (analog4)	17	E	Enable
8	4	DB0	Données
9	5	DB1	Données
10	6	DB2	Données
11	7	DB3	Données
4	8	DB4	Données
5	9	DB5	Données
6	10	DB6	Données
7	11	DB7	Données
14 (analog0)	13	CS1	Chip select 1
15 (analog1)	12	CS2	Chip select 2
Reset (mettre à +5 V)	14	RST	Reset
Pin1 du trimmer / Pin3 du trimmer à GND	18	Vout	Tension pour Vo
+ 5 V du rétroéclairage	19	BLA	Anode du rétroéclairage
GND	20	BLK	Cathode du rétroéclairage

Tableau 9 : connexions physiques entre l'afficheur ADM12864H et la carte Arduino 2009.

Pour plus de simplicité nous vous proposons de la **télécharger directement** sur notre site **www.electroniquemagazine.com** dans le sommaire détaillé du numéro **133** à l'onglet « **Télécharger** ».

Nous avons enrichi la librairie avec d'autres exemples. Le dossier « **KS0108** » contient la librairie « **KS0108GLCD** » ainsi que des fichiers d'exemples. Il doit être copié dans le dossier « librairie » du dossier Arduino.

Ouvrez le sketch nommé « **GLCDexample.pde** » que vous trouverez dans le dossier « **GLCDexample** ». Pour cela, lancez l'IDE Arduino, cliquez sur « **Fichier** » → « **Exemples** » → « **KS0108** » → « **GLCDexample** » (voir la figure 6).

Une fois les différents sketchs chargés, pour celui nommé « **glcdBitmap** » (« Fichier » → « Exemples » → « KS0108 » →

Un autre exemple nommé dessine deux lignes croisées en plein écran, avec un cercle central.

La librairie est très complète et la liste des commandes ainsi que leurs descriptions se trouvent dans le **Tableau 6**.

Comme vous l'avez constaté, l'**affichage graphique requiert 8 lignes de données et 5 lignes** pour le **contrôle**. L'affectation de ces fonctions est définie dans le fichier « **ks0108_Arduino.h** », les lignes de code concernées sont visibles dans le **Listing 2**.

Vous pouvez modifier ces lignes de code pour définir différemment les broches utilisées pour la gestion de l'afficheur LCD graphique. Vous pouvez vous reporter aux notes explicatives sur le site officiel Arduino (malheureusement en Anglais).

L'utilisation d'un microcontrôleur à 28 broches permet la disponibilité de lignes supplémentaires pour des applications spécifiques.

Une autre possibilité est offerte par l'afficheur graphique « **ADM12864H** » (128x64) sur la base d'un circuit « KS0108 » qui a l'avantage de n'avoir que 20 broches et donc facilement utilisable avec une plaque d'essai.

Pour les connexions à une carte Arduino reportez-vous au **Tableau 9**.

Il existe une solution pour réduire les difficultés de câblage dues au grand nombre de lignes utilisées dans un afficheur graphique, c'est d'utiliser un **afficheur graphique de type série**. Il n'a besoin que de **4 lignes** (+ VCC, GND, TX, RX) qui sont contrôlées par des données envoyées en mode série.

Il existe aussi dans le commerce un **convertisseur série/parallèle pour afficheur GLCD** qui est en fait une interface et qui est appelé « **Graphic LCD serial backpack** ». Un exemple est le modèle « **LCD-09352** », parfaitement compatible avec l'afficheur graphique « **ADM12864H** » (voir la figure 6).

Tableau 6 : commandes de gestion des afficheurs graphiques pour les fonctions « Graphic Drawing Functions » (tracé des primitives graphiques) et « Font Functions » (affichage des caractères).

Commandes	Description
GLCD.Init(NON_INVERTED)	Initialise la librairie pour l'affichage normal ou inversé. Si normal, chaque pixel est affiché (apparaît sombre), si inversé le pixel est effacé.
GLCD.GotoXY(x,y)	Place le curseur à la position x,y. Si 0,0 cela correspond au coin supérieur gauche.
GLCD.ClearScreen()	Efface l'écran
Graphic Drawing Functions: si la couleur est WHITE efface tous les pixels. Si la couleur est BLACK active tous les pixels.	
GLCD.DrawCircle(x, y, radius, color)	Trace un cercle dont le centre est aux coordonnées x,y de rayon = radius et de couleur = color.
GLCD.DrawLine(x1,y1,x2,y2,color)	Trace une ligne des coordonnées x1,y1 aux coordonnées x2,y2 et de couleur = color.
GLCD.DrawVertLine(x, y, length, color)	Trace une ligne verticale aux coordonnées x,y, de longueur = length et de couleur = color.
GLCD.DrawHoriLine(x, y, length, color)	Trace une ligne horizontale aux coordonnées x,y, de longueur = length et de couleur = color.
GLCD.DrawRect(x, y, width, height, color)	Trace un rectangle aux coordonnées x,y de largeur = width, de hauteur = height et de couleur = color.
GLCD.DrawRoundRect(x, y, width, height, radius, color)	Trace un rectangle aux coins arrondis avec un angle = radius, aux coordonnées x,y de largeur = width, de hauteur = height et de couleur = color.
GLCD.FillRect(x, y, width, height, color)	Trace un rectangle plein aux coordonnées x,y de largeur = width, de hauteur = height et de couleur = color.
GLCD.InvertRect(x, y, width, height)	Inverse les pixels du rectangle spécifié
GLCD.SetInverted(invert)	Active le mode de dessin inversé
GLCD.SetDot(x, y, color)	Dessine un pixel aux coordonnées x, y.
GLCD.DrawBitmap(bitmap, x, y, color)	Dessine une image bitmap aux coordonnées x, y.
Font Functions	**Fonctions sur les polices de caractères**
GLCD.SelectFont(font, color)	Active la police des caractères "font" et de couleur = color.
GLCD.PutChar(character)	Ecrit un caractère
GLCD.Puts(string)	Ecrit une chaîne de caractères
GLCD.Puts_P(string)	Ecrit une chaîne de caractères contenue dans la mémoire programme.
GLCD.PrintNumber(number)	Ecrit la valeur décimale d'une variable numérique.
GLCD.CursorTo(x, y)	Définit les coordonnées de base pour les polices à largeur fixe.

Broches d'Arduino	Broches du LCD-09352	
Vin	**Vin**	Alimentation positive 6 V à 7 V
GND	**GND**	GND (masse)
TX	**RX**	Lignes de données

Tableau 7 : connexions physiques entre Arduino et une interface série LCD-09352 pour afficheur LCD graphique.

Dans ce cas, le câblage est très simplifié, car il suffit de 3 lignes décrites dans le **Tableau 7**. La carte supplémentaire doit être insérée sur le connecteur de l'afficheur. Dans notre cas, nous avons utilisé une barrette mâle sur l'afficheur et une barrette femelle sur le convertisseur.

Comme le convertisseur fonctionne avec une tension de 5 V fournie par son régulateur interne, vous pouvez l'alimenter avec une tension légèrement supérieure de 6 V à 7 V (recommandé). Le maximum autorisé étant 9 V. Voilà pourquoi la carte Arduino doit être alimentée par une source externe, car la tension provenant du port USB n'est pas suffisante.

Pour nos expériences, nous avons utilisé une simple alimentation non stabilisée de 0,5 A, fournissant une tension de sortie de 6 V. Même si cela peut sembler peu, en fait avec ce genre d'alimentation la sortie peut être facilement à 8 V voire 9 V.

La gestion de ce convertisseur série est simplement réalisée par l'**envoi de commandes de type séries** que la carte Arduino gère **via les deux lignes** (TX et RX), elles sont identiques à celles utilisées pour la communication avec un port USB (via le convertisseur FT232).

Dans le **Tableau 8**, vous pouvez voir un résumé des principales commandes envoyées à l'afficheur.

Elles doivent être conformes à la norme RS232, le réglage des paramètres de communication est : « **115200,N,8,1** » soit 115200 bauds (valeur par défaut mais modifiable), pas de bit de parité, 8 bits de données et 1 bit d'arrêt.

Figure 6 : ici le convertisseur « LCD-09352 » série/parallèle pour afficheur GLCD.

Tableau 8

Fonction	Octets (hexadécimale)	Note
Effacement	0x7C + 0x00	
Mode démo	0x7C + 0x04	
Mode inverse	0x7C + 0x12	
Réglage du rétroéclairage	0x7C + 0x02 + R	R comprise entre 0 et 100 Ω
Réglage/reset du pixel	0x7C + 0x10 + X + Y + P	X=coordonnée horizontale Y=coordonnée verticale P=0 reset P=1 réglage
Tracé de ligne	0x7C + 0x0C + X1 + Y1 + X2 + Y2 + P	X1,Y1=coordonnées de début X2,Y2=coordonnées de fin P=0 effacement P=1 Tracé
Tracé de cercle	0x7C + 0x03 + X1 + Y1 + R + P	X1,Y1=coordonnées du centre R=rayon P=0 effacement P=1 Tracé
Tracé de rectangle	0x7C + 0x0F + X1 + Y1 + X2 + Y2 + P	X1,Y1=coin supérieur gauche X2,Y2=coin inférieur droit P=0 effacement P=1 Tracé
Réglage du Baudrate	0x7C + 0x07 + N	N="1" = 4800bps N="2" = 9600bps N="3" = 19,200bps N="4" = 38,400bps N="5" = 57,600bps N="6" = 115,200bps
Initialisation de coordonnée	0x7C + 0x18 + X	X=nouvelle coordonnée X
Initialisation de coordonnée	0x7C + 0x19 + Y	Y=nouvelle coordonnée Y

Dans le **Listing 3** vous pouvez voir un programme utilisé pour tester l'afficheur, il permet de régler le **rétroéclairage à 0** (OFF), d'effacer l'écran, de **dessiner une ligne**, de **tracer un cercle** et enfin **afficher du texte**.

Comme vous pouvez le constater, l'affichage du texte est immédiat grâce au **générateur de caractères inclus dans le convertisseur « backpack »**. Il suffit d'**envoyer le code ASCII correspondant aux caractères** que vous désirez afficher sur l'écran.

Évidemment, nous n'avons pas besoin d'effectuer de conversion mais simplement d'utiliser l'instruction « **Serial. print** » suivie des caractères à afficher entre guillemets.

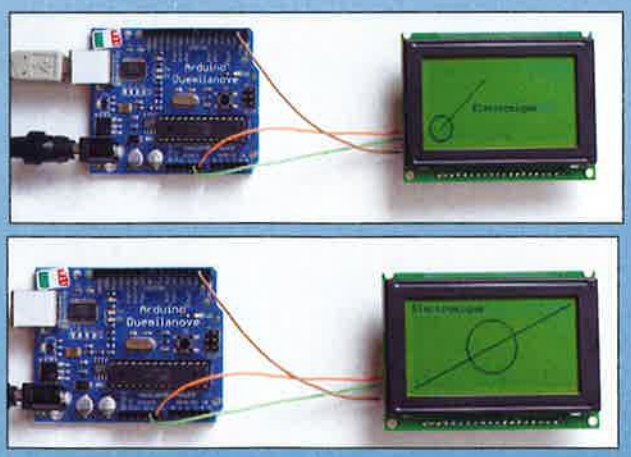
Une alternative à l'envoi des commandes qui vient d'être expliqué est décrite dans le **Listing 4**. Vous pouvez voir une partie du programme qui envoie les octets nécessaires un à la fois.

Listing 2

```
/*********************************************************/
/*  Configuration d'attribution des bits du LCD aux broches Arduino          */
/*********************************************************/
/* Broches d'Arduino utilisées pour les commandes
 * affectation par défaut, utilise les 5 premières  broches analogiques
 */

#define CSEL1           14          // CS1 Bit  // permutation de l'affectation des broches avec CSEL2, si gauche/
droite  l'image est inversée
#define CSEL2           15          // CS2 Bit
#define R_W             16          // R/W Bit
#define D_I             17          // D/I Bit
#define EN              18          // EN Bit

//#define RES           19          // Reset Bit  // enlever les slashs pour contrôler le reset du LCD sur cette broche

#define LCD_CMD_PORT    PORTC       // Commande Output Register broches 14-19

/* Broches d'Arduino utilisées pour les données du LCD */

#define dataPins8to11     // bits 0-3 assignés aux broches 8 à 11 d'Arduino, bits 4-7 assignés aux broches 4-7 d'Arduino

//#define dataPins14to17  //bits 0-3 assignés aux broches 14 à 17 d'Arduino, bits 4-7 assignés aux broches 4-7 d'Arduino

//#define dataPins0to3              // bits 0-3 assignés aux broches 0 à 3 d'Arduino, assignés aux broches 4-7 d'Arduino.

/*********************************************************/
/*    Fin de la configuration d'Arduino                                      */
/*********************************************************/
```

Listing 3

```
/*
  Display_02
  Exemple d'utilisation d'un afficheur graphique à commande série.
  Pour le câblage matériel :
  Graphic LCD Serial Backpack connecté au GLCD-ADM12864H
  Lignes utilisées : Vin, GND, TX.
 */

byte buf_erase[]  = {0x7C, 0x00};                  // Efface l'écran
byte buf_backl[]  = {0x7C, 0x02, 0};               // (Rétroéclairage 0 = 0 % 100 = 100 %)
byte buf_line[]   = {0x7C, 0x0C, 0, 0, 127, 63, 1}; // Ligne (X1,Y1) (X2,Y2) 1 = tracé   0 = effacement
byte buf_circle[] = {0x7C, 0x03, 63, 31, 20, 1};   // Cercle (X1,Y1) rayon   1 = tracé   0 = effacement

void setup()
{
  Serial.begin(115200);                            // Règle la vitesse de communication à 115200 bauds
}

void loop()                                        // Répète à l'infini
{
  Serial.write(buf_backl, 3);                      // Rétroéclairage à  0 %
  delay(100);
  Serial.write(buf_erase, 2);                      // Efface l'écran
  delay(1000);
  Serial.write(buf_line, 7);                       // Trace la ligne
  delay(1000);
  Serial.write(buf_circle, 6);                     // Trace le cercle
  delay(1000);
  Serial.print("Electronique");                    // Ecrit un texte
  delay(1000);
}
```



Il existe aussi une troisième alternative, qui est de construire une routine (sous-programme) qui doit être appelée à chaque fois que la fonction doit être exécutée. Cette solution qui est plus élégante est décrite dans le **Listing 5**.

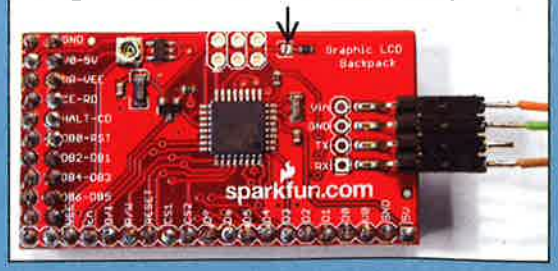
Notez que le convertisseur « **Graphic LCD serial backpack** » est également compatible avec un autre afficheur dénommé « **Graphic LCD 160x128 Huge** » et disposant de dimensions plus grandes et d'une résolution supérieure.

Au niveau du programme, la gestion est identique, bien évidemment la carte Arduino communique toujours avec le convertisseur « backpack », mais celui-ci gère l'afficheur de manière complètement autonome. Avec le rétroéclairage désactivé la visibilité est réduite, nous avons prévu un sketch spécial (display_05) qui fixe un rétroéclairage minimum. Le programme est disponible en téléchargement avec les autres, comme indiqué plus haut dans ces pages.

Le convertisseur « backpack » est normalement compatible avec un afficheur de 128 x 64 pixels, mais vous pouvez le régler pour fonctionner avec un afficheur de **128 x 160 pixels**. Pour cela **retirez simplement le cavalier** (jumper) présent sur le circuit imprimé (voir la figure 7).

Le cavalier est vraiment minuscule et est réalisé avec un pont de soudure qui doit être enlevé très soigneusement.

Figure 7 : le pont de soudure indiqué par la flèche doit être retiré soigneusement, pour pouvoir gérer des afficheurs de 128 x 160 pixels.



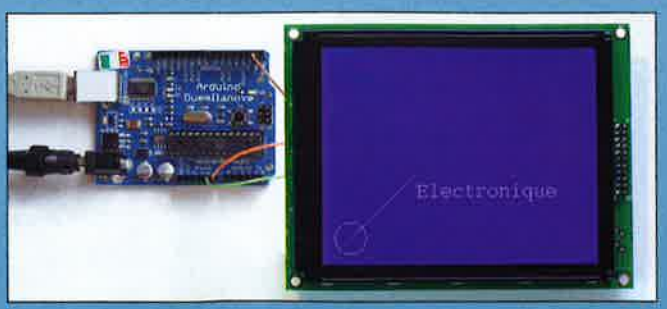
Listing 4

```
/*
Display_03
Exemple d'utilisation d'un afficheur graphique à commande série.
Utilisation des commandes en ligne
Pour le câblage matériel :
Graphic LCD Serial Backpack connecté au GLCD-ADM12864H
Lignes utilisées : Vin, GND, TX.
 */

void setup()
{
  Serial.begin(115200);                 // règle la vitesse de communication à 115200 bauds
}

void loop()                             // Répète à l'infini
{
  Serial.print(0x7C, BYTE);             // Efface l'écran
  Serial.print(0x00, BYTE);
  delay(100);
  Serial.print(0x7C, BYTE);             // Rétroéclairage à  0 %
  Serial.print(0x02, BYTE);
  Serial.print(0x00, BYTE);
  delay(100);
  Serial.print(0x7C, BYTE);             // Trace un cercle
  Serial.print(0x03, BYTE);             //
  Serial.print(100, BYTE);              // Coordonnée X
  Serial.print(50, BYTE);               // Coordonnée Y
  Serial.print(10, BYTE);               // Rayon
  Serial.print(1, BYTE);                // 1 = tracé    0 = effacement
  delay(1000);
  Serial.print(0x7C, BYTE);             // Règle la coordonnée X
  Serial.print(0x18, BYTE);             // Coordonnée X
  Serial.print(10, BYTE);               // Coordonnée X
  Serial.print(0x7C, BYTE);             // Règle la coordonnée Y
  Serial.print(0x19, BYTE);             // Coordonnée Y
  Serial.print(40, BYTE);               // Coordonnée Y
  Serial.print("Electronique");
  delay(1000);
}
```

Listing 5

```
 /*
Display_04
Exemple d'utilisation d'un afficheur graphique à commande série.
Utilisation d'une routine pour les commandes
Pour le câblage matériel :
 Graphic LCD Serial Backpack connecté au GLCD-ADM12864H
Lignes utilisées : Vin, GND, TX.
 */

void setup()
{
  Serial.begin(115200);                 // Règle la vitesse de communication à 115200 bauds
}

void loop()
{
  backlight(0);                         // Rétroéclairage à 0 %
```

```
 clearLCD();                                    // Efface l'écran
 cursorSet(40,30);                              // Définit la position du curseur pour le texte
 Serial.print("Electronique");                  // Ecrit un texte à la position indiquée précédemment
 line(10,10,50,50);                             // Trace une ligne
 circle(10,10,10);                              // Trace un cercle
 delay(2000);
}

// SubRoutine

 void clearLCD(){                               // Efface l'écran
 Serial.print(0x7C, BYTE);
 Serial.print(0x00, BYTE);
}

                                                // Gestion du rétroéclairage
 void backlight(byte light){
 Serial.print(0x7C, BYTE);
 Serial.print(0x02, BYTE);
 Serial.print(light);                           // définit à 0 pour désactiver le rétroéclairage
}

 // Trace une ligne
void line(byte x1, byte y1, byte x2, byte y2){
 byte buf_line[]  = {0x7C, 0x0C, 0, 0, 127, 63, 1};  // Ligne (X1,Y1)
 Serial.print(0x7C, BYTE);                      // Trace une ligne
 Serial.print(0x0C, BYTE);
 Serial.print(x1, BYTE);                        // Coordonnée X1
 Serial.print(y1, BYTE);                        // Coordonnée Y1
 Serial.print(x2, BYTE);                        // Coordonnée X2
 Serial.print(y2, BYTE);                        // Coordonnée Y2
 Serial.print(1, BYTE);                         // 1 = tracé   0 = effacement
}

 // Trace un cercle
void circle(byte xpos, byte ypos, byte radius){
 Serial.print(0x7C, BYTE);                      // Trace un cercle
 Serial.print(0x03, BYTE);                      //
 Serial.print(xpos, BYTE);                      // Coordonnée X
 Serial.print(ypos, BYTE);                      // Coordonnée Y
 Serial.print(radius, BYTE);                    // Rayon
 Serial.print(1, BYTE);                         // 1 = tracé   0 = effacement
}

// déplace le curseur à la position X,Y spécifiée
void cursorSet(byte xpos, byte ypos){
 Serial.print(0x7C, BYTE);                      // Règle la coordonnée X
 Serial.print(0x18, BYTE);                      // Coordonnée X
 Serial.print(xpos, BYTE);                      // Coordonnée X
 Serial.print(0x7C, BYTE);                      // Règle la coordonnée Y
 Serial.print(0x19, BYTE);                      // Coordonnée Y
 Serial.print(ypos, BYTE);                      // Coordonnée Y
}
```



Nous arrivons à la fin de cette 3ème partie du Cours Arduino, n'hésitez pas à faire les expériences et à modifier les sketchs pour comprendre le comportement d'un afficheur graphique.

Dans le prochain Cours Arduino, nous apprendrons à gérer des servocommandes de modélismes, des moteurs pas à pas et des moteurs à balais à l'aide de programmes contenant des routines simplifiées. ■

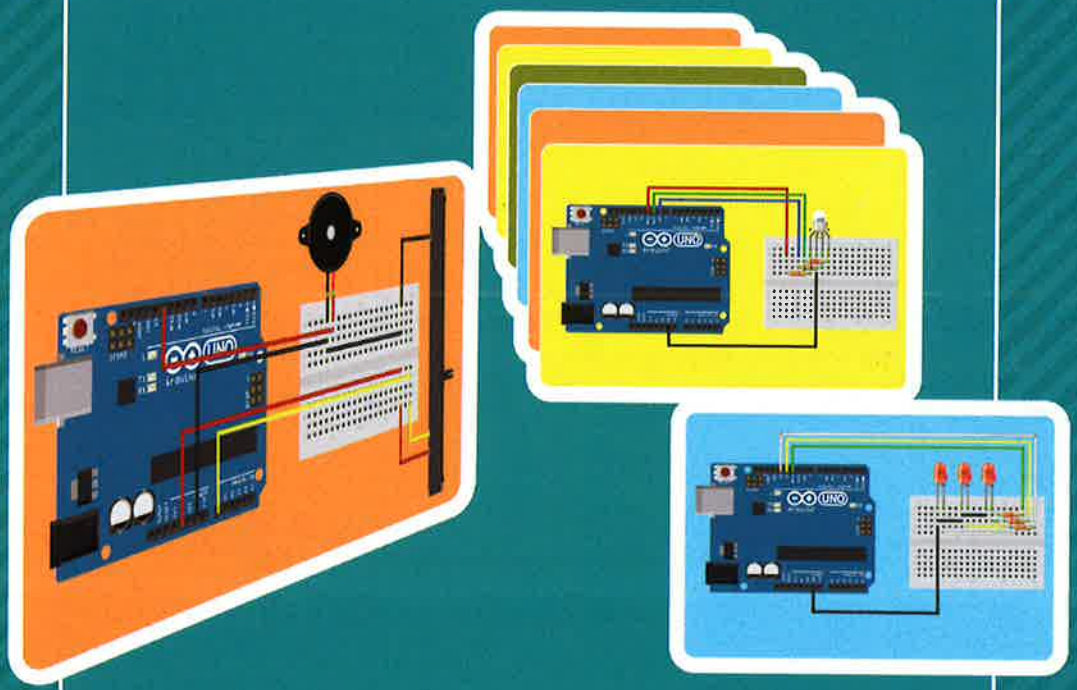
14,90 € frais de port compris pour la France Métropolitaine

17,90 € frais de port compris pour la CEE

les DOM-TOM et autres pays nous consulter pour un devis ou à calculer sur le site www.electroniquemagazine.com

l'ABC d'ARDUINO

Découvrez le monde d'Arduino et comment utiliser la carte de prototypage la plus populaire pour mener à bien les projets que vous souhaitez réaliser.



Aujourd'hui plus que jamais la possibilité de donner libre cours à la créativité dans des applications technologiques doit être envisagée en faisant appel à l'électronique numérique et la programmation. L'électronique programmable reste le seul moyen de réaliser des applications intéressantes telles que la robotique ou celles connectées Internet.

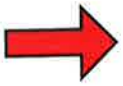
L'équipe qui a développé Arduino a trouvé un moyen de faire de l'électronique avec beaucoup d'idées mais peu d'outils, afin d'affronter la complexité de plus en plus grande des circuits électroniques. Il convient parfaitement à ceux qui sont réticents à étudier l'électronique et la programmation. Avec Arduino, vous pouvez compter sur une électronique facile à mettre en œuvre et programmable sans difficultés, grâce à un environnement de développement qui permet de passer de l'idée au projet très simplement, grâce au nombre grandissant de librairies disponibles ainsi qu'à une grande communauté.

Dans ce livre, vous découvrirez tout le potentiel de la carte de développement Arduino Uno avec des exemples pratiques, vous apprendrez à transformer vos idées en applications fonctionnant du premier coup et qui deviendront la base pour vos projets de plus en plus complexes.

JMJ EDITIONS - B.P. 20025 - 13720 LA BOUILLADISSE règlement par chèque à l'ordre de JMJ ÉDITIONS et/ou règlement par Paypal sur notre site : www.electroniquemagazine.com - Téléphone : 0820 820 534

ABONNEMENT

ABONNEZ-VOUS

OUI, Je m'abonne à ELECTRONIQUE ET LOISIRS magazine LE MENSUEL DE L'ÉLECTRONIQUE POUR TOUS **A PARTIR DU N°** 134 ou supérieur  N°

E0133

Ci-joint mon règlement de ..... € correspondant à un abonnement de 4 revues Annuel

Règlement CB directement sur le site **www.electroniquemagazine.com rubrique Abonnement**

Adresser mon abonnement à :

Nom _____ Prénom _____

Adresse _____

Code postal _____ Ville _____

Tél. _____ e-mail _____

*Date, le*_____

Signature obligatoire ▷

TARIFS FRANCE

❑ **4 numéros** **28€,00**

TARIFS CEE/EUROPE

❑ **4 numéros** **32€,00**

DOM-TOM/HORS CEE OU EUROPE :

NOUS CONSULTER SUR www.electroniquemagazine.com rubrique Abonnement

L'ASSURANCE de ne manquer aucun numéro en recevant votre revue directement dans votre boîte aux lettres près d'une semaine avant sa sortie en kiosques.

BÉNÉFICIER de 50% de remise** sur les CD-ROM des anciens numéros

POUR TOUT CHANGEMENT D'ADRESSE, N'OUBLIEZ PAS DE NOUS INDIQUER VOTRE NUMÉRO D'ABONNÉ (INSCRIT SUR L'EMBALLAGE)

Bulletin à retourner à : JMJ – Abo. ELM

B.P. 20025 - 13720 LA BOUILLADISSE – Tél. 0820 820 534 – Fax 0820 820 722

Directeur de Publication
Rédacteur en chef
Jean Marc MOSCATI

Direction - Administration
JMJ éditions
B.P. 20025
13720 LA BOUILLADISSE
Tél. : 0820 820 534

Secrétariat - Abonnements
Petites-annonces - Ventes
A la revue

Vente au numéro
A la revue

Publicité
A la revue

Maquette - Illustration
Composition - Photogravure
JMJ éditions sarl

Impression
Rotimpres

Distribution
Presstalis

Hot Line Technique
0820 820 534 *
du lundi au vendredi de 16 h à 18 h

Web
www.electroniquemagazine.com
E-mail
support@electroniquemagazine.com

* prix d'un appel local

JMJ éditions
Sarl au capital social de 7800 €
RCS MARSEILLE : 421 860 925
APE 221E
Commission paritaire : 1015T79056
ISSN : 1295-9693
Dépôt légal à parution

ELECTRONIQUE ET LOISIRS magazine LE MENSUEL DE L'ÉLECTRONIQUE POUR TOUS

EST RÉALISÉ EN COLLABORATION AVEC :

NUOVA ELETTRONICA Elettronica In

Testeur EOBD pour véhicules

réf : ET849..... livré en Kit ..... 81 €

Cette interface est en mesure de communiquer avec la prise EOBD (Enhanced On Board Diagnostic) de votre voiture. Elle permet de contrôler et de vérifier avec un PC les paramètres et l'état de fonctionnement des principaux composants gérés par la centrale électronique de votre auto. Elle pourra selon le type d'unité centrale de votre véhicule, le logiciel téléchargé et les restrictions imposées par les constructeurs, afficher les données concernant le nombre de tours moteur, la température de l'air et de l'eau, la quantité d'air aspirée, le fonctionnement de la vanne EGR, et beaucoup d'autres données spécifiques à chaque véhicule. Cette interface est le lien entre la centrale électronique de votre véhicule et un PC, dans lequel vous aurez à installer le logiciel « ScanMaster-ELM DEMO » (voir l'article consacré dans la revue) ou la version complète vendue séparément (voir ci-dessous).
Ce kit est capable de dialoguer avec toutes les voitures fabriquées à partir de 2001 pour les modèles essence et depuis 2003 pour les modèles diesel. Comme le système de diagnostic a évolué depuis des années, cette interface gère 5 protocoles de communications :
- SAE-J1850 PWM (General Motors) ;
- SAE-J1850 VPW (Ford) ;
- ISO-9141(Chrysler, véhicules européens et asiatiques) ;
-ISO-14230 (appelé Keyword Protocol 2000 ou KWP 2000, nouvelle version du ISO-9141 plus rapide) ;
-ISO-15765-4 CAN BUS (équipe déjà de nombreux véhicules, toutes marques confondues, standard sur tous les modèles à partir de 2008).
Le kit comprend tous les composants, le circuit imprimé percé et sérigraphié, le microcontrôleur ELM327, le boîtier et le câble DB9/EOBD. **Le logiciel Scan Master n'est pas compris dans le kit.**

ELM327 (pic seul programmé)

Microcontrôleur PIC seul, programmé en usine pour l'interface EOBD. Il communique avec les BUS CAN, SAE et K-Line. Ces 3 types de bus sont la norme pour les voitures ayant une prise OBD. Le microcontrôleur peut communiquer selon 12 protocoles différents, donc avec tous les véhicules à la norme « EURO3 » (2001 pour l'essence, 2003 pour le diesel). Boîtier Dual in line 28 broches.

réf : ELM327....PIC seul programmé.....44,50 €

Protocoles pris en charge :

SAE J1850 PWM (41.6 kbaud)
SAE J1850 VPW (10.4 kbaud)
ISO 9141-2 (5 baud init, 10.4 kbaud)
ISO 14230-4 KWP (5 baud init, 10.4 kbaud)
ISO 14230-4 KWP (fast init., 10.4 kbaud)
ISO 15765-4 CAN (11 bit ID, 500 kbaud)
ISO 15765-4 CAN (29 bit ID, 500 kbaud)
ISO 15765-4 CAN (11 bit ID, 250 kbaud)
ISO 15765-4 CAN (29 bit ID, 250 kbaud)
SAE J1939 CAN (29 bit ID, 125 kbaud)
USER1 CAN
USER2 CAN

Logiciel ScanMaster en version complète OEM pour testeur EOBD

ScanMaster ELM-OBD2/EOBD est un logiciel (version OEM) pour le diagnostic des véhicules. Il est développé spécifiquement pour le circuit ELM327. Le logiciel prend en charge les 10 fonctions du protocole SAE J1979 (OBD-II en mode diagnostic $01 - $0A), et tous les protocoles de communication. En combinaison avec l'interface basée sur le microcontrôleur ELM327, ce logiciel est la solution idéale pour les diagnostics OBD et OBD2 jusque-là réservés aux professionnels. Tous les véhicules équipés de la norme OBD2 sont automatiquement détectés et peuvent être diagnostiqués avec ce logiciel.
Le logiciel offre de nombreuses fonctions telles que :
* la mesure et l'affichage direct de la consommation de carburant ; * l'accélération du véhicule de 0 à 100 km/h ; * calculs complets et fonctions de test de la puissance et du couple du moteur (kW/CV et Nm) ; * lecture et effacement des codes d'erreurs ; * possibilité d'enregistrer un journal contenant les données lues de la centrale ;* possibilité du choix de la langue en Français ; * compatible avec Windows 2000, XP, 2003 Vista, 7 64 bits & 32 bits. **Le logiciel ScanMaster étant en version OEM, il ne peut être utilisé qu'avec le circuit ELM327 ou en combinaison avec notre interface.** **réf : *SCANMASTER* .... Logiciel...91 €**

Câble OBD2

d'une longueur de 1,8 m comprenant un connecteur mâle « J1962 » et un connecteur DB9 femelle. Le câble est de type « ULN2464 » blindé, idéal pour les signaux différentiels (PWM et CAN) et adapté pour une utilisation dans des environnements avec des niveaux élevés de bruits électromagnétiques. Ce câble est conçu pour les connecteurs de diagnostic Européens (qui sont souvent présents sous le tableau de bord). Le câble est conforme à la norme « RoHS » et ne contient pas de dispositif de conversion ou de circuits électroniques.

réf : J1962M-DB9F .....Câble.....16,60 €

Boîtier pour testeur EOBD

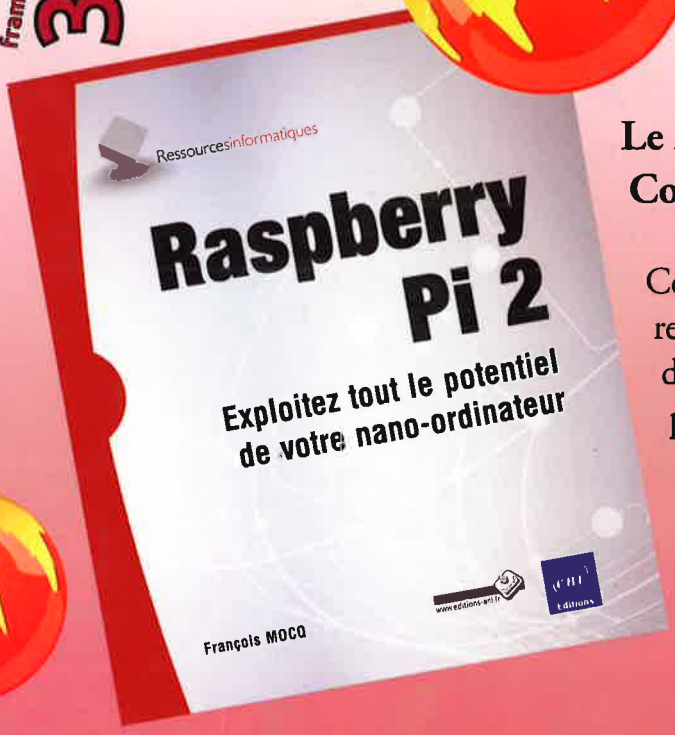
Boîtier en ABS noir pour le testeur EOBD.
Dimensions externes : 90 x 56 x 23 mm.
Dimensions internes : 86,31 x 52,56 x 17,66 mm.
Livré sans perçage

réf : SC-700..... Boîtier .....1,81 €

COMELEC
CD 908 - 13720 BELCODÈNE
Tél. : 04 42 70 63 90 Fax : 04 42 70 63 95
www.comelec.fr

Framboise 314

Le Raspberry Pi à la sauce française



Le livre de référence pour le Raspberry Pi 2
Comprend également les modèles A+ et B+

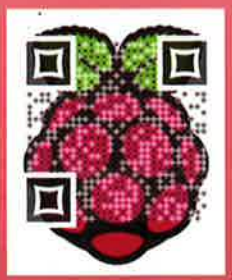
Ce livre fournira des bases solides pour explorer les ressources offertes par le Raspberry Pi 2 tant au point de vue du système d'exploitation que du développement et de l'interfaçage physique.

Aucun pré-requis en Linux, en programmation ou en électronique n'est nécessaire.

Mais aussi :

Le blog

L'actualité du Raspberry Pi
Des informations
Des tutoriels
Le Raspberry Pi dans la presse



Le forum

Une communauté
Des experts
De l'entraide

@framboise314

Framboise314

www.framboise314.fr