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

ELECTRONIQUE ET LOISIRS magazine

ELECTRONIQUE

http://www.electroniquemagazine.com

n°137 HIVER 2016

Caméra Pi : la vue pour le RaspberryPi

Bras robotisé 5 mouvements

- Cours Arduino VII
- Breakout Board - III
- Sapin de Noël caméléon
- Demoboard PIC18F - II
- Imprimante 3D bicolore - III
- Programmez sous iPhone, iPad - III

N° 137 Décembre 2016

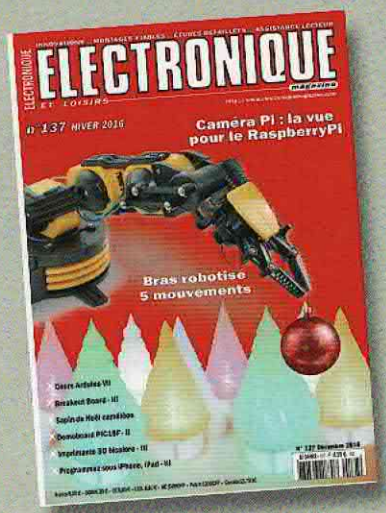
M 04662 - 137 - F: 8,30 € - RD

France 8,30 € - DOM 8,30 € - CE 8,80 € - LUX. 8,80 € - NC(S)990FP - Poly.fr 1300CFP - Canada 12,70 $C

Sommaire

ARTICLES

Numéro 137
Hiver 2016
100 pages



07 INFORMATIQUE

CAMERA PI : LE SENS DE LA VUE POUR LE RASPBERRYPI

Nous allons étudier la vision numérique en utilisant un RaspberryPI et un module caméra nommé « Caméra Pi ». Cette caméra peut filmer une vidéo à la norme 1080p30 soit une résolution de 1080 lignes verticales avec 30 images/s, et une résolution horizontale de 1920 points. Elle prend en charge le format vidéo HD 720p60 avec une résolution horizontale de 1280 points par 720 lignes verticales et 60 images/s ainsi que le format VGA .

17 MICROCONTRÔLEUR

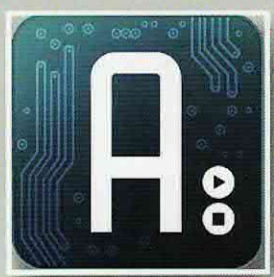
PLATINE DE DÉVELOPPEMENT POUR PIC18F8XXX - DEUXIÈME PARTIE

Nous analyserons, dans cette deuxième partie, le firmware de la platine de développement pour PIC-18F8XXX. Les routines, qui sont écrites en Assembleur pour PIC, sont optimisées pour une utilisation dans l'environnement de développement MPLAB de Microchip, mais aussi pour différents compilateurs tels que le PICBASIC et le CCS.

35 ARDUINO

COURS ARDUINO SEPTIÈME PARTIE

Nous allons apprendre, dans cette septième leçon du Cours Arduino, à utiliser une carte mémoire de type « SD Card ». Pour cela nous allons utiliser une carte électronique spécialement conçue pour être interfacée avec Arduino. Pour notre application, nous avons choisi un adaptateur pour carte SD déjà prêt à l'utilisation que vous trouverez facilement dans le commerce, il s'agit de l'adaptateur « SD Card shield V4.0 ».



48 DIDACTIQUE

UN SAPIN DE NOËL CAMÉLÉON

Il existe deux approches différentes pour réaliser un gadget de Noël qui prend la couleur du support sur lequel il repose. C'est l'occasion de voir la différence entre le développement d'un projet « amateur » et d'un produit « commercial ». En particulier, pour l'électronique, nous avons opposé une réalisation basée sur Arduino et une autre en partant de zéro basée sur un microcontrôleur spécifique.

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

L'EDITO HIVER 2016

Chères lectrices, chers lecteurs,

La Rédaction d'Électronique et Loisirs Magazine vous souhaite de Joyeuses Fêtes de Noël ainsi qu'une Bonne et Heureuse Année 2017.

En cette fin d'année nous proposons à nos fidèles lectrices et lecteurs un montage original pour cette période de fêtes : il s'agit du Sapin de Noël Caméléon qui prend la couleur de la surface sur laquelle il repose. L'intérêt de l'article réside dans le fait que nous proposons deux approches différentes, une orientée prototype l'autre orientée production de masse. En effet, en partant d'une même idée, nous réalisons deux montages très différents ayant la même fonction. Nous vous montrons ainsi comment passer d'un prototype à une production en grande série.

D'autre part le bras robotisé, bien qu'il ressemble à un jouet, intéressera aussi bien les débutants que les passionnés de robotique.

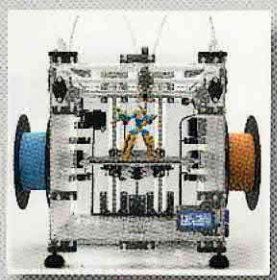
Nous continuons l'étude des cartes de prototypage BREAKOUT BOARD avec 4 nouvelles cartes couvrant la plupart des domaines de l'électronique, ainsi que l'étude des routines en Assembleur de la platine de développement pour PIC 18FXX.

Enfin nous donnons la vue au RaspberryPi avec la Caméra Pi qui pourra, à distance, filmer ce qui se passe.

La Rédaction

63 IMPRIMANTE 3D BICOLORE

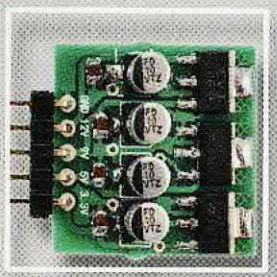
3DVERTEX : LA MÉCANIQUE (SUITE)



Dans le précédent numéro 136 d'Electronique et Loisirs Magazine, nous avons abordé la construction mécanique de la 3DVERTEX. Nous nous sommes arrêtés à l'assemblage de la tête d'impression (extrudeuse) qui, rappelons-le, est une étape longue et minutieuse. Nous allons maintenant, dans cette troisième partie, aborder le montage de la tête d'impression qui demande un travail très soigneux. Entrons dans le vif du sujet !

73 LABORATOIRE

CARTES DE PROTOTYPAGE « BREAKOUT BOARD » MULTIFONCTIONS - III



Nous enrichissons notre série de cartes de prototypage avec diverses fonctionnalités utiles comme la gestion d'un relais par des niveaux logiques, l'interfaçage d'un capteur optique et d'une alimentation linéaire multiple qui fournit 3 tensions stabilisées avec des valeurs standards : 9 VDC, 5 VDC et 3,3 VDC. Enfin nous vous proposons un convertisseur N/A géré par le BUS I2C, dont nous étudierons le fonctionnement à l'aide d'une carte Arduino Uno.

82 ROBOTIQUE

BRAS ROBOTISÉ 5 MOUVEMENTS

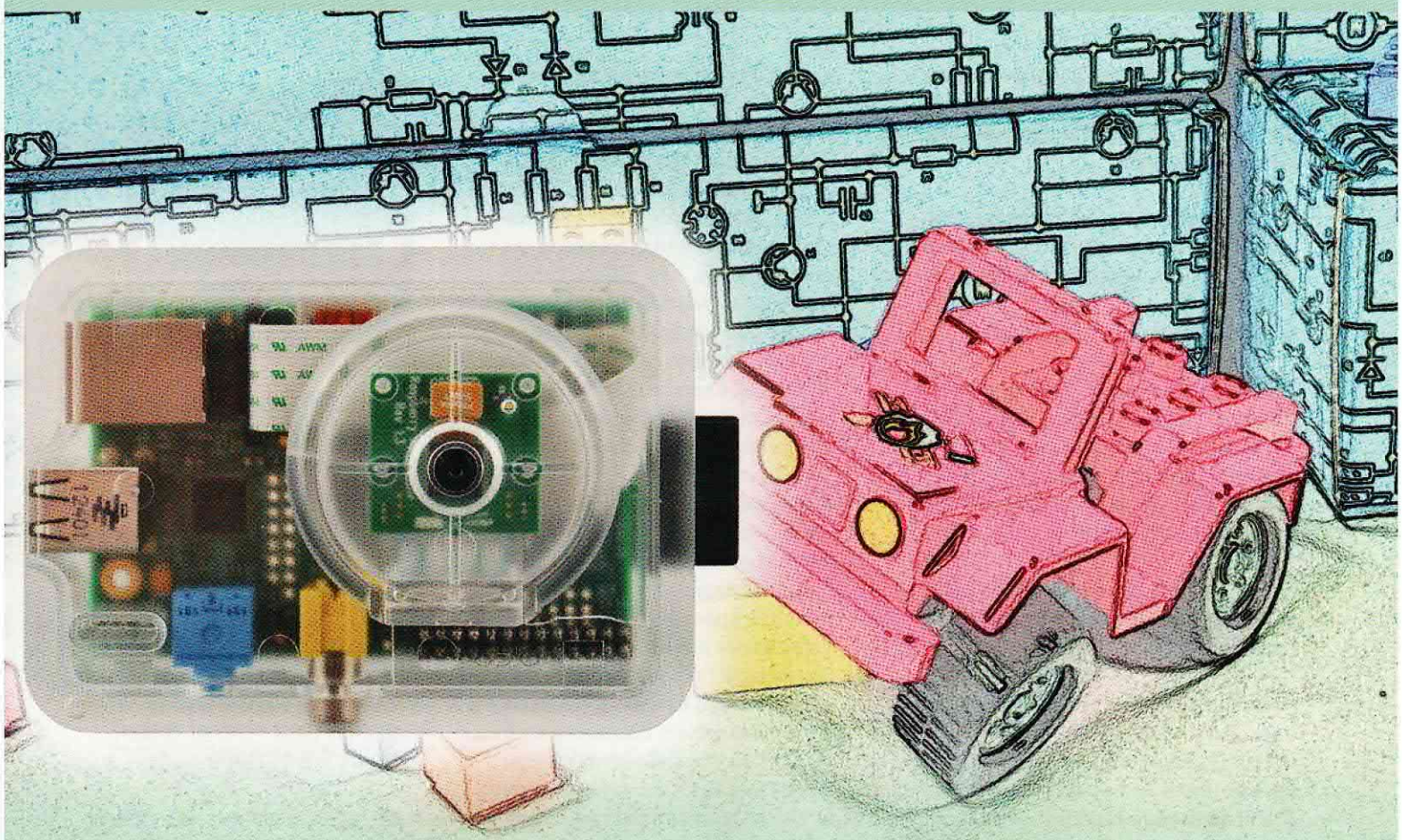
Parmi les nombreux robots présents sur le marché, nous avons testé un produit en kit qui intéressera certainement les lecteurs passionnés de robotique mais aussi les novices dans ce domaine. C'est un véritable « jouet » technologique. Il s'agit d'un bras robotisé, capable d'effectuer 5 mouvements et de soulever un poids de 100 g en le serrant entre ses « doigts » artificiels et, pas seulement dans le sens vertical. Il dispose au niveau de ses pinces d'une LED en lumière blanche qui éclaire la zone de travail.

87 IPHONE

COURS DE PROGRAMMATION SUR IPHONE : TROISIÈME PARTIE

Dans cette troisième leçon du Cours de programmation sur iPhone, nous allons étudier des patrons de conception aussi appelés modèles de conception (en Anglais « Design Patterns »), dont nous avons abordé le concept dans la première partie. Plus précisément, nous allons apprendre à utiliser deux d'entre eux : le « Model View Controller » (Modèle-Vue-Contrôleur) et le « Singleton ».

à l'adresse **www.electroniquemagazine.com** dans le sommaire de la revue 137 et à l'onglet « Télécharger »



CAMERA PI : LE SENS DE LA VUE POUR LE RASPBERRYPI

Nous commençons une série d'articles consacrés à l'étude de la vision numérique en utilisant un RaspberryPI et un module caméra nommé « Caméra Pi », dont nous vous présenterons également les fonctionnalités.

de Marco MAGAGNIN

Les robots dotés de capacités autonomes et d'organes sensoriels similaires à ceux de l'homme ont toujours poussé l'imagination des écrivains de romans de science-fiction. La riche production de livres et de films en est une démonstration pratique. Il suffit de se reporter au film « I robot » dans lequel les robots pouvaient même être dotés d'une conscience.

Comme toujours, à un moment donné dans l'histoire, la science traditionnelle a commencé à traiter le sujet avec comme argument fort le remplacement de la main d'œuvre humaine par des robots pour des tâches laborieuses.

Une fois la problématique prise en compte, les recherches ont été orientées vers des objectifs plus concrets pour aboutir à la construction de solutions technologiquement exploitables. À l'heure actuelle, de nombreuses applications trouvent leur utilisation dans les secteurs industriels et les services. Cela va du montage des voitures dans l'industrie jusqu'à la lecture des codes à barres dans les services.

Là encore la technologie a évolué, en passant du balayage laser d'une séquence de barres verticales blanches et noires jusqu'à l'interprétation d'une image dans laquelle se trouve un code tel que le « QR Code » (Quick Response Code) au moyen d'une caméra.

D'autres applications importantes se trouvent dans les entreprises de transformation et d'emballage alimentaire, dans l'agriculture, l'électronique, la santé, etc.

Dans cette série d'articles, nous vous présenterons des expériences qui vous permettront de vous familiariser avec cette technologie.

Depuis 3 ans, nous vous présentons des projets et des applications centrés autour du RaspberryPi, cependant ce projet implique l'intégration de différentes disciplines et technologies.

Un système de vision, en fait, se compose de plusieurs technologies intégrées les unes aux autres, comme par exemple l'optique, l'électronique, l'informatique, les systèmes de communication et la mécanique.

Pour commencer, nous allons vous exposer un bref aperçu du scénario associé à la vision par ordinateur.

Nous mettons de côté pour l'instant l'acquisition des images à des fins de transmission et/ou d'enregistrement, comme dans le cas d'une webcam, d'un caméscope ou d'un système de surveillance vidéo « passive ». Dans ces cas, les images capturées sont modifiées et mises en forme pour être transmises sur des canaux analogiques ou numériques et/ou pour être stockées sur un support électronique.

Les sujets que nous voulons aborder exigent, sous différentes formes, le traitement et l'interprétation du contenu des images afin d'en extraire des informations utiles pour piloter les processus de prises de décisions automatiques ou semi-automatiques.

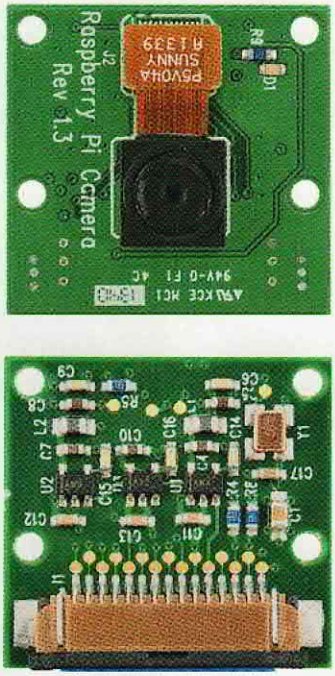
Voici les grandes lignes de la problématique, des objectifs et des questions que pose la vision numérique, par ordre de complexité :

- **Détection** : traitement de l'image afin de vérifier si des changements ont eu lieu entre les trames successives, dans la totalité ou une partie de l'image. Cette technologie est largement utilisée dans les systèmes de surveillance vidéo, dans lesquels une ou plusieurs caméras couvrent des zones sensibles afin de détecter une intrusion, et éventuellement l'enregistrer pour toute la durée de l'événement. La visualisation de l'enregistrement permet de déterminer les méthodes utilisées ;
- **Reconnaissance** : le traitement de l'image a pour but de reconnaître des formes appartenant à des classes déterminées. Par exemple des objets de forme circulaire ou rectangulaire, des objets caractérisés par des contours fermés ou par des zones d'une certaine couleur peuvent, compte tenu de leur forme, appartenir à une classe d'objet particulière (un visage ou une partie de celui-ci comme les yeux ou le nez) ;
- **Identification** : ceci est un nouveau perfectionnement du traitement précédent, il permet « d'identifier » dans les images la présence d'une instance spécifique d'une forme appartenant à une classe connue. Revenons à l'exemple de la reconnaissance des formes, après avoir déterminé qu'une certaine forme au sein de l'image est un visage, le système tente de déterminer si le visage correspond à un visage connu, et si celui-ci correspond à une personne connue ou non ;
- **Reconstruction** : ce type de traitement d'image s'effectue à partir de deux ou plusieurs images du même objet pris sous des angles différents afin de générer un modèle 3D complet de l'objet. La principale difficulté de ce type de traitement est de trouver et de réunir les points communs de l'objet provenant des différentes images.

Nous verrons par la suite d'autres aspects qui affectent la vision numérique, et comment ceux-ci influent sur les conditions d'acquisition et les caractéristiques de l'application.

Parmi les conditions associées, citons les sources d'éclairage. Normalement, la prise de vue est éclairée par la lumière naturelle ou artificielle, c'est-à-dire dans les mêmes conditions que l'œil humain.

Des possibilités supplémentaires sont données par l'utilisation de caméras infrarouges, qui permettent de réaliser des systèmes de vision nocturne ou d'enregistrer un rayonnement infrarouge émis en fonction de la température présente dans la zone de vision.



Le module caméra pour le RaspberryPI (vue avant et arrière).

De cette façon, il est possible d'obtenir et de traiter des images numériques dans des parties du spectre lumineux non détectées par l'œil humain.

Quelle que soit la configuration du système de vison adopté, l'architecture intègrera les composants suivants :

- des caméras et des lentilles ;
- un système d'éclairage ;
- l'objet ou la scène à examiner ;
- un système d'acquisition et de traitement des images ;
- une interface Homme-Machine ;
- des interfaces avec l'environnement extérieur.

Le système se compose d'une caméra et de son optique de capture d'image. L'optique contient un capteur (CMOS ou CCD) qui est composé d'un ou plusieurs réseaux de photodétecteurs (caméra noir et blanc ou couleur) transformant la lumière qu'ils absorbent en une grandeur électrique qui est ensuite traitée par le système.

De cette façon, les images capturées peuvent être faites par un logiciel capable de « lire » la représentation des images et en extraire les informations nécessaires pour identifier ses caractéristiques (de l'image).

Cela permet de reconnaître les aspects d'une image (par exemple les contours, les arêtes, les formes) afin d'effectuer des comparaisons par le système. Sur la base des résultats obtenus, le système sera en mesure de prendre des décisions.

Pour les essais et les applications qui seront faites avec le RaspberryPi, nous utiliserons une caméra appelée « Caméra Pi » qui est spécialement conçue pour une utilisation avec le RaspberryPi.

La connexion s'effectuera directement sur le connecteur CSI (Caméra Serial Interface) et non pas sur un port USB comme dans le cas d'une webcam.

« Caméra Pi » est disponible en deux versions, l'une sensible à la lumière du jour et l'autre sans filtre infrarouge (IR). Cette dernière étant utilisée pour la vision nocturne et la capture d'images thermiques.

Les raisons qui nous ont fait choisir cette caméra par rapport à une webcam classique sont :

- une haute résolution de 2592 × 1944 pixels pour les images ;
- un format vidéo de « 1080p30 », c'est-à-dire 1080 lignes de définition verticale, la lettre « p » signifie balayage progressif (l'image n'est pas entrelacée), le chiffre 30 correspond à 30 images par secondes ;
- la disponibilité d'un logiciel de gestion, spécialement conçu pour la « Caméra Pi », capable de la contrôler comme pour une vraie caméra avec des réglages et des effets spéciaux ;
- ses dimensions qui sont seulement de 25 mm x 20 mm x 9 mm.

« Caméra Pi » a la capacité de filmer une vidéo à la norme 1080p30 soit une résolution de 1080 lignes verticales avec 30 images par seconde, et une résolution horizontale de 1920 points. Elle prend également en charge le format vidéo HD 720p60 avec une résolution horizontale de 1280 points par 720 lignes verticales et 60 images par seconde ainsi que le format VGA 640x480p60/90.

Installation de « Caméra Pi »

L'installation de base de « Caméra Pi » se compose de deux étapes, une première étape consistant à l'installation matérielle et une seconde étape correspondant à la configuration du logiciel pour un fonctionnement standard. Nous commençons par l'installation matérielle.

Avant de commencer l'installation, nous vous recommandons de manipuler avec prudence la « Caméra Pi ». Elle est particulièrement sensible aux décharges électrostatiques, donc avant de la retirer de son emballage antistatique, vous devez toucher un objet métallique ou un dispositif relié à la terre. Il existe aussi dans le commerce des gants antistatiques pour manipuler des composants sensibles aux décharges électrostatiques.

Ensuite vous devez la connecter au RaspberryPi. La caméra dispose d'un câble plat flexible de 15 contacts, il doit être relié au connecteur CSI (Camera Serial Interface) du RaspberryPi. Le connecteur CSI est disposé entre la prise réseau (LAN) et la prise HDMI.

Comme vous pouvez le voir sur le schéma de la figure 1, l'interface CSI comporte deux lignes de données, une ligne d'horloge, une interface de contrôle compatible avec le BUS I2C et deux lignes d'alimentation (3,3 V et la masse).

L'interface de communication CSI entre la caméra et le RaspberryPi est une interface série différentielle unidirectionnelle, avec des lignes de données et d'horloge, conforme au standard « MIPI Alliance » des interfaces « D-PHY ».

La figure 2 représente les connexions entre l'émetteur et le récepteur au standard CSI-2. En dehors de la « Caméra Pi », ce type de connexion est utilisé en interne pour les caméras des smartphones et de manière générale pour les périphériques mobiles. L'interface de contrôle du système est une interface bidirectionnelle basée sur le standard du BUS I2C.

Sortez la « Caméra Pi » de son emballage, en faisant attention aux décharges électrostatiques. Une fois extraite vous apercevrez un circuit imprimé d'environ

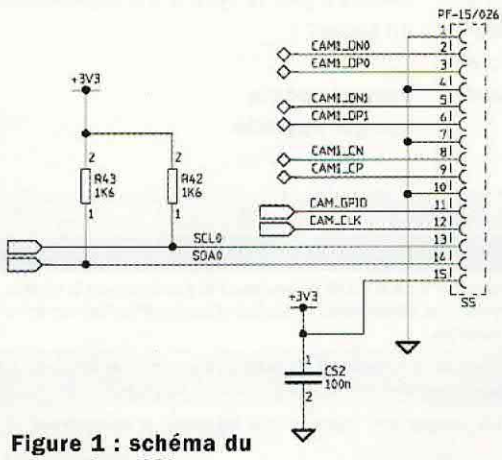


Figure 1 : schéma du connecteur CSI.

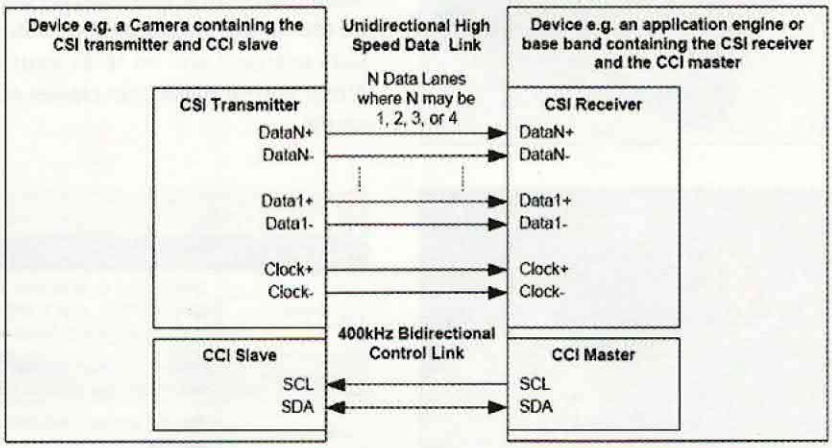


Figure 2 : connexions entre l'émetteur et le récepteur au standard CSI-2.

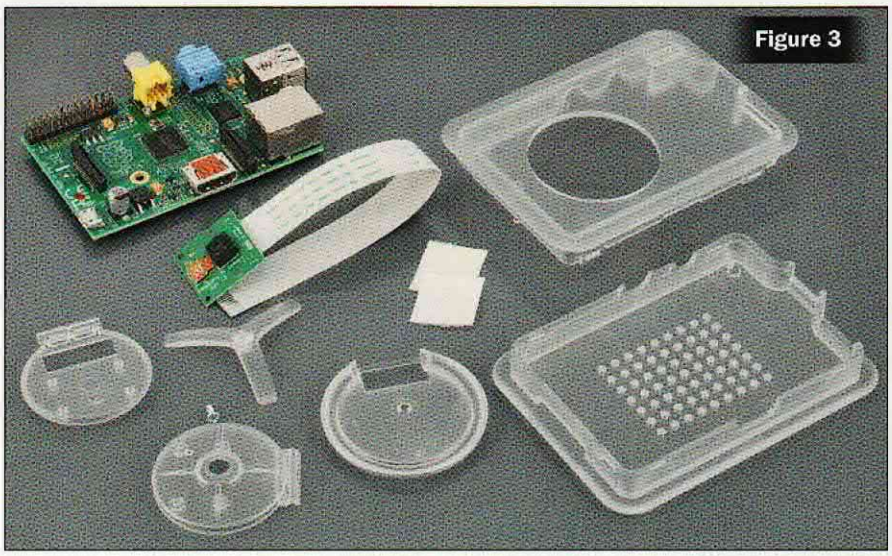


Figure 3

Figure 6

Figure 7

5 cm² sur lequel se trouve une minuscule caméra microscopique, derrière le circuit se trouve le câble plat à 15 conducteurs.

Il existe différents boîtiers disponibles dans le commerce pour le RaspberryPi et la « Caméra Pi », de façon à réaliser un système compact et adapté aux différents besoins. Celui que nous avons choisi pour nos tests donne à l'ensemble « RaspberryPi/Caméra Pi » une allure d'appareil photo.

Avant d'insérer la « Caméra Pi » dans le boîtier, vous devez enlever la protection en plastique placée sur l'objectif en tirant sur la languette.

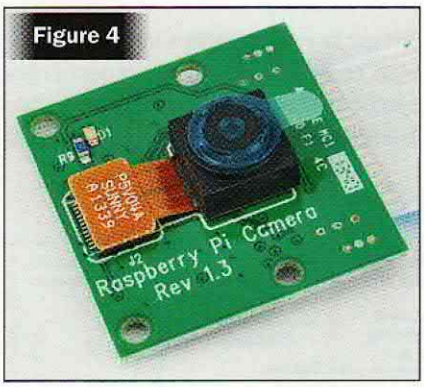


Figure 4

Figure 5

Si vous utilisez le boitier visible sur nos photos, vous devez d'abord faire passer le câble plat dans toutes les encoches prévues à cet effet, avant de l'insérer dans le connecteur CSI-2.

Pour relier physiquement la « Caméra Pi » au RaspberryPi, vous devez simplement insérer le câble plat dans le connecteur CSI-2 situé entre le connecteur Ethernet et le connecteur HDMI.

Lors de l'insertion du connecteur, vérifiez que les broches argentées se situent face au connecteur HDMI et donc, que le côté bleu du câble plat est positionné face au connecteur Ethernet comme le montre la figure 6.

Pour ce type d'expérience, il est préférable d'utiliser le RaspberryPi comme un ordinateur classique avec moniteur (par exemple sur un téléviseur doté d'une entrée HDMI), un clavier et une souris.

N'oubliez pas de connecter le RaspberryPi à Internet. Pour les premières configurations, vous pouvez vous connecter à distance en utilisant le protocole SSH (Secure Shell).

Eventuellement, reportez-vous aux anciens numéros d'Electronique et Loisirs Magazine à partir du 123 et suivants.

Après avoir vérifié les connexions et retiré la protection de l'objectif, vous pouvez alimenter le RaspberryPi et attendre le démarrage (boot). Commencez la configuration, ouvrez une fenêtre de terminal et connectez-vous avec l'utilisateur « root ». À l'invite, mettez à jour le système d'exploitation en tapant :

```
apt-get update
apt-get upgrade
```

Tableau 1

Options de prévisualisation	
--preview, -p	Dimensions de la fenêtre "Preview" <'x,y,w,h'>. Les paramètres x et y positionnent la fenêtre dans la vidéo, w et h déterminent les dimensions. La fenêtre d'aperçu (Preview) recouvre toujours les autres fenêtres ouvertes.
--fullscreen, -f	Force la fenêtre "Preview" à occuper tout l'écran. Dans cette configuration, les proportions de l'image sont conservées avec l'insertion d'éventuelles barres sur les côtés.
--nopreview, -n,	Pas de fenêtre "Preview". Les images sont quand même capturées et enregistrées, si nécessaire.
--opacity, -op	Définit la transparence de la fenêtre "Preview". "0" = complètement transparente (invisible), 255 = recouvre entièrement les graphiques sous-jacents.

À la fin, ouvrez l'outil de configuration de Raspbian en tapant :

raspi-config

Faites défiler les éléments du menu jusqu'à ce que vous mettiez en surbrillance « Enable Camera » (voir la figure 8).

Appuyez sur « Enter » (ou sur « Entrée » selon votre clavier) et à la page suivante sélectionnez « <Enable> » à l'aide de la touche « Tab » (généralement à gauche du clavier au-dessus de la touche « Maj »).

Retournez au menu principal et sélectionnez « <Finish> » (voir la figure 10). Il vous sera demandé si vous voulez redémarrer le RaspberryPi.

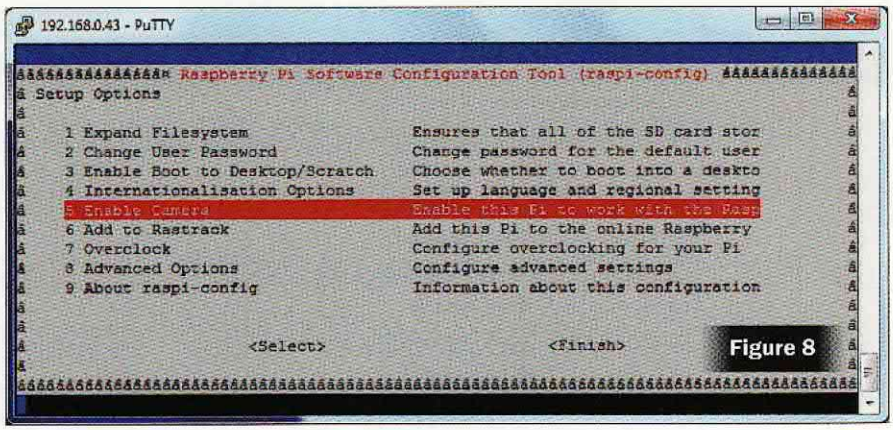


Figure 8

Choisissez « <Yes> » (voir la figure 11) et attendez le redémarrage.

À ce stade, nous pouvons faire quelques expériences avec la « Caméra Pi » en utilisant les commandes « natives » sous forme de lignes de commande.

Les commandes sont « **raspistill** » et « **raspistillyuv** » pour la capture et le stockage des images et « **raspivid** » pour la capture et le stockage de séquences vidéo. À chaque commande, nous avons un ensemble de paramètres qui peuvent être divisés en deux catégories.

Tableau 2

Contrôle général de la caméra		
–sharpness, -sh	Règle la définition (netteté) de l'image. Les valeurs vont de -100 à 100. Par défaut = 0.	
–contrast, -co	Règle le niveau de contraste. Les valeurs vont de -100 à 100. Par défaut = 0.	
–brightness, -br	Règle le niveau de luminosité de l'image. Les valeurs vont de 0 (blanc) à 100 (noir). Valeur par défaut = 50.	
–saturation, -sa	Règle le niveau de saturation. Les valeurs vont de -100 à 100. Par défaut = 0.	
–ISO, -ISO	Règle l'échelle de sensibilité ISO à utiliser lors des opérations de capture. Les valeurs vont de 100 à 800.	
–vstab, -vs	Active la stabilisation vidéo, fonctionne uniquement en mode vidéo.	
–ev, -ev	Règle le niveau de compensation d'exposition en unités EV (déclenchement d'ouverture du diaphragme). Les valeurs vont de -10 à 10.	
–exposure, -ex	Définit le type d'exposition, les options sont :	
	off	
	auto	calcul automatique de l'exposition
	night	mode pour les prises de vues nocturnes
	nightpreview	mode "preview " (aperçu) pour les prises de vues nocturnes
	backlight	mode optimal pour les prises de vues éclairées par l'arrière
	spotlight	mode pour un éclairage au moyen d'un spot
	sports	mode pour les prises de vues sportives et les plans rapides
	snow	mode optimal pour photographier des paysages enneigés
	beach	mode optimal pour photographier des paysages comportant des plages
	verylong	mode adapté pour de longues expositions
	fixedfps	règle la fréquence des images (FPS)
	antishake	mode anti-vibration
	fireworks	mode adapté pour photographier des feux d'artifices
	Certains des paramètres décrits peuvent être utilisés ou non en fonction d'autres paramètres de la caméra.	
–awb, -awb	Règle le mode de la balance automatique des blancs (AWB - balance des blancs automatique) :	
	off	désactive la fonction de balance
	auto	mode de balance automatique (par défaut)
	sun	balance en plein soleil
	cloud	balance avec ciel nuageux
	tungsten	balance avec un éclairage au moyen de lampes à tungstène
	fluorescent	balance avec un éclairage au moyen de lampes fluorescentes
	incandescent	balance avec un éclairage au moyen de lampes à incandescence
	flash	balance avec un éclairage au flash
	horizon	mode Horizon (par exemple, pour photographier des couchers de soleil)

Tableau 2 suite		
Contrôle général de la caméra		
--imxfx, -ifx	Réglage des effets à appliquer aux images :	
	none	aucun effet appliqué
	negative	produit une image en négatif
	solarise	polarise l'image
	whiteboard	effet de voile blanc
	blackboard	effet de voile noir
	sketch	effet d'esquisse
	denoise	élimine le bruit de l'image
	emboss	effet estampé
	oilpaint	effet peinture à l'huile
	hatch	effet dessin avec tirets
	gpen	effet dessin au crayon
	pastel	effet pastel
	watercolour	effet aquarelle
	film	effet de film avec du grain
	blur	effet de flou
	saturation	image avec des couleurs saturées
	colourswap	en cours d'implémentation
	washedout	en cours d'implémentation
	posterise	en cours d'implémentation
	colourpoint	en cours d'implémentation
	colourbalance	en cours d'implémentation
	cartoon	en cours d'implémentation
--awb, -awb	Règle le mode de la balance automatique des blancs (AWB - balance des blancs automatique) :	
	off	désactive la fonction de balance
	auto	mode de balance automatique (par défaut)
	sun	balance en plein soleil
	cloud	balance avec ciel nuageux
	tungsten	balance avec un éclairage au moyen de lampes à tungstène
	fluorescent	balance avec un éclairage au moyen de lampes fluorescentes
	incandescent	balance avec un éclairage au moyen de lampes à incandescence
	flash	balance avec un éclairage au flash
	horizon	mode Horizon (par exemple, pour photographier des couchers de soleil)
--metering, -mm	Réglages de la mise au point, indique la méthode de calcul de la mise au point pour le "Preview" et la capture d'image :	
	average	moyenne sur l'ensemble du champ de vision
	spot	mise au point sur un endroit précis
	backlit	considère l'image éclairée par derrière
	matrix	mesure matricielle (Matrix Metering), tient compte de l'ensemble de l'image pour évaluer la quantité de lumière disponible
--rotation, -rot	Règle la rotation de l'image (0-359). Définit la rotation en degrés qui doit être imposée à l'image. En théorie, il est possible de régler n'importe quel angle de rotation. Dans la pratique, les limitations matérielles du RaspberryPi ne permettent que d'utiliser les valeurs en degrés suivantes : 0, 90, 180 et 270	
--hflip, -hf	Définit le retournement horizontal de l'image. Elle est capturée et affichée dans le "Preview" puis stockée dans un fichier, inversée horizontalement.	
--vflip, -vf	Définit le retournement vertical de l'image. Elle est capturée et affichée dans le "Preview" puis stockée dans un fichier, inversée verticalement.	
--roi, -roi	Définit la zone d'intérêt pour le capteur de la caméra (ROI - Region Of Interest). Permet de spécifier une zone rectangulaire du champ de vision pour être utilisée comme source pour les fonctions de "Preview", de capture et de stockage. Les paramètres requis sont les positions x et y du coin supérieur gauche de la zone d'intérêt, la largeur et la hauteur de la zone. Toutes les valeurs sont en coordonnées normalisées, comprises entre 0.0 à 1. Pour décrire une zone qui commence au centre du rectangle et l'étirer d'un quart vers la droite et vers le bas, vous devez spécifier les valeurs : -roi 0.5,0.5,0.25,0.25	

La première catégorie, commune à toutes les commandes, permet de configurer la caméra. Par exemple, il est possible de régler la saturation, le contraste et le type d'exposition.

La deuxième catégorie permet de configurer le fonctionnement des commandes, telles que l'intervalle entre les prises de vues, la durée d'une vidéo, et des indications sur la mémorisation des résultats obtenus. Dans le **Tableau 1**, vous trouverez les paramètres de configuration de la caméra qui sont valables pour toutes les commandes.

Le **Tableau 2** contient les paramètres utilisés pour la capture de simples images, puis les paramètres avec les commandes « **raspistill** » et « **raspistillyuv** ».

Le **Tableau 3** liste les paramètres utilisés avec la commande « **raspivid** » pour capturer des séquences vidéo. Examinons quelques exemples pratiques d'utilisation de ces commandes.

Tout d'abord nous devons choisir si nous allons utiliser ou non la fonction « Preview », qui nous permet de visualiser un aperçu des images traitées par les commandes.

Si nous nous servons de la fonction « Preview », nous devons utiliser le RaspberryPi comme un PC, c'est-à-dire connecté à un écran, un clavier et une souris, et le faire fonctionner directement à partir de l'interface graphique du système d'exploitation (voir la figure 12).

Sinon, nous pouvons le faire fonctionner en mode terminal distant en utilisant le **protocole SSH** avec les logiciels « **Putty** » et « **WinSCP** ».

Voyons quelques exemples de commandes pour capturer des images et enregistrer des vidéos.

Prendre des photos avec « Caméra Pi »

La commande pour réaliser des photos est :

```
raspistill
```

suivie des paramètres et des configurations que vous trouverez dans les Tableaux 1 et 2.

La combinaison de ces paramètres permet d'utiliser la « Caméra Pi » comme un véritable appareil photo de bon niveau. Vous pouvez définir les valeurs d'exposition, de mise au point et appliquer une quantité considérable d'effets.

Vous pouvez choisir de visualiser l'image et/ou de l'enregistrer dans un fichier. La commande la plus simple qui prend une photo et la stocke dans un fichier au format JPEG est :

```
raspistill -o photo001.jpg
```

Pour enregistrer une image après 2 secondes de « Preview » (l'intervalle est spécifié en millisecondes), la commande est :

```
raspistill -t 2000 -o photo001.jpg
```

Pour enregistrer une image de dimension et de résolution réduites, la commande est :

```
raspistill -t 2000 -o image.jpg -w 640
-h 480
```

Pour enregistrer une image de qualité inférieure afin d'économiser de l'espace, la commande est :

```
raspistill -t 2000 -o image.jpg -q 5
```

Tableau 3

Options de la caméra - commande "raspistill"	
--width, -w	définit la largeur de l'image <size>
--height, -h	définit la hauteur de l'image <size>
--quality, -q	définit la qualité JPEG <0 à 100> La valeur 100 représente l'image non compressée (qualité maximale), la valeur 75 peut être considérée comme un bon compromis entre la qualité et le taux compression.
--raw, -r	ajoute des métadonnées au jpeg (raw Bayer)
--output -o	nom du fichier où est mémorisée l'image ou la vidéo <filename>. Si vous ne spécifiez pas de fichiers, le résultat de la capture n'est pas enregistré. Si le nom du fichier correspond à "-", tout le flux de sortie est envoyé à "stdout".
--verbose, -v	lors de l'exécution de la commande, des messages de débogage et des informations sont générés.
--timeout, -t	intervalle avant l'acquisition de l'image. La commande est exécutée pour la durée spécifiée avant d'effectuer l'opération de capture si un fichier de sortie a été spécifié. Si l'intervalle n'est pas spécifié, il est imposé à 5 secondes.
--timelapse, -tl	mode Timelapse (film accéléré). La valeur spécifie l'intervalle en millisecondes entre les acquisitions des images individuelles. Pour maintenir les images dans l'ordre, vous pouvez spécifier le facteur "%04d" à l'endroit où vous souhaitez insérer le compteur d'images. Par exemple, en combinant les deux paramètres suivants : -t 30000 -tl 2000 -o image%04d.jpg vous obtenez une acquisition d'image toutes les 2 secondes pendant une durée totale de 30 secondes. Les images seront nommées : image0001.jpg, image0002.jpg...image0015.jpg. Le facteur "%04d" génère un numéro à 4 chiffres avec des zéros à gauche. Le chiffre "04" correspond au nombre de chiffres qui caractérisent le compteur, par exemple "%08d" correspond à un compteur à 8 chiffres.
--thumb, -th	définit les caractéristiques des images miniatures (thumbnail) incluses dans le fichier JPEG (x:y:quality). Si non spécifié, la valeur par défaut impose une dimension de 64 x 48 avec une qualité JPEG de 35.
--demo, -d	effectue une démonstration pour le nombre de millisecondes indiquées. Cette option présente une série de fonctions de la caméra sans effectuer d'acquisition. La démonstration se termine à la fin de la période indiquée quelle que soit la phase dans laquelle elle se trouve.
--encoding, -e	codage du fichier de sortie. Les formats gérés sont le "jpg", "bmp", "gif" et "png". Les formats d'images non accélérées (gif, png, bmp) nécessitent plus de temps pour être sauvegardés par rapport au format jpg, qui lui, profite de l'accélération matérielle. Le format est prioritaire sur un éventuel suffixe spécifié pour le fichier de sortie.
--exif, -x	liste des balises EXIF (Exchangeable image file format) à inclure dans le fichier de sortie JPEG. Le format de chaque balise (tag) est de type <clé = valeur>. Ce paramètre permet d'insérer jusqu'à 32 tags EXIF aux métadonnées de l'image. Une partie de ces données est automatiquement gérée par les commandes de CameraPi. Les choix pris avec ce paramètre prévalent sur ceux automatiques.
--fullpreview, -fp	mode "Full Preview". Ce paramètre règle la fenêtre « Preview » à la même résolution que l'acquisition. Dans ce mode, le nombre maximal d'images est de 15 par seconde et la fenêtre de « Preview » montre le même champ de vision que l'image en acquisition.

Pour positionner la fenêtre « Preview » sur l'écran aux coordonnées (100, 100) avec des dimensions de 300 par 200 pixels, la commande est :

raspistill -t 2000 -o image.jpg –p 100,100,300,200

Pour désactiver la fenêtre « **Preview** », tapez :

raspistill -t 2000 -o image.jpg –n

Pour activer l'effet « **estampé** » sur l'image capturée, tapez :

raspistill -t 2000 -o image.jpg -ifx emboss

Pour capturer des images à intervalles réguliers, une toutes les 10 secondes pendant 10 minutes (600 000 ms) et les stocker dans des fichiers avec des noms par ordre croissant du type photo_numero_1.jpg, photo_numero_2.jpg, etc., tapez la commande :

raspistill -t 600000 -tl 10000 –o photo _ numero _%d.jpg

Expérimentez différents cas.

Enregistrer une vidéo avec « Caméra Pi »

Les paramètres de réglage des images et le « **Preview** » sont pratiquement identiques à ceux des images, certains paramètres sont disponibles uniquement pour la gestion de séquences vidéos. La taille par défaut des séquences vidéos est de 1920 x 1080 pixels.

Pour enregistrer une vidéo de 5 secondes avec la taille par défaut (1080p30), tapez la commande :

raspivid -t 5000 -o video.h264

Pour enregistrer une vidéo de 5 secondes avec un débit par défaut de 3,5MBits/s, tapez :

raspivid -t 5000 -o video.h264 -b 3500000

Pour enregistrer une vidéo de 5 secondes avec 5 images par seconde (5fps), tapez :

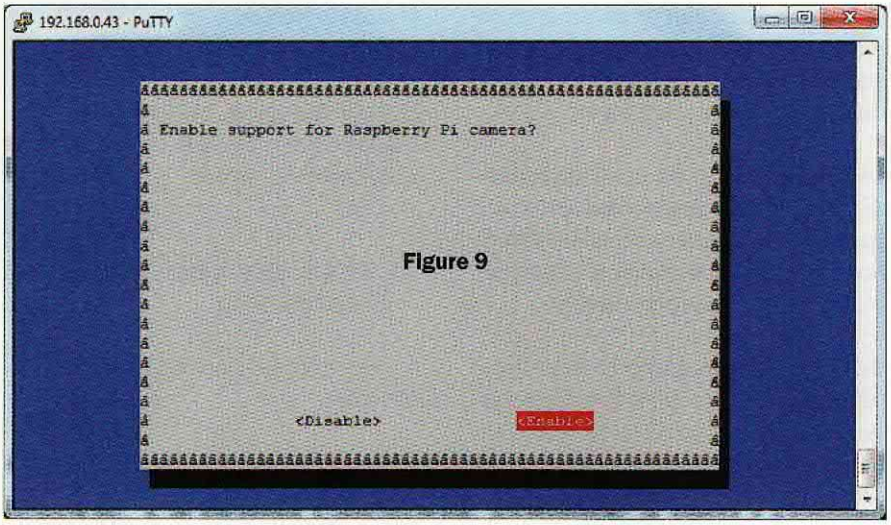


Figure 9

Tableau 4

Options de la caméra - commande "raspivid"	
--width, -w	définit la largeur de l'image <size>. Valeur en pixel entre 64 et 1920.
--height, -h	définit la hauteur de l'image <size>. Valeur en pixel entre 64 et 1080.
--bitrate, -b	Règle la vitesse de transmission du transfert en bits par seconde. 10MBits/s sont exprimés de la manière suivante : "-b 10000000". Pour un flux de haute qualité H264 de 1080p, un débit de 15 Mbits/s ou plus est recommandé.
--output, -o	nom du fichier où est mémorisée l'image ou la vidéo <filename>. Si vous ne spécifiez pas de fichiers, le résultat de la capture n'est pas enregistré. Si le nom du fichier correspond à "-", tout le flux de sortie est envoyé à "stdout".
--verbose, -v	lors de l'exécution de la commande, des messages de débogage et des informations sont générés.
--timeout, -t	intervalle avant l'acquisition de l'image. La commande est exécutée pour la durée spécifiée avant d'effectuer l'opération de capture si un fichier de sortie a été spécifié. Si l'intervalle n'est pas spécifié, il est imposé à 5 secondes. La valeur "0" configure la commande pour fonctionner en continu, jusqu'à ce qu'elle soit terminée avec la commande CTRL-C.
--demo, -d	effectue une démonstration pour le nombre de millisecondes indiquées. Cette option présente une série de fonctions de la caméra sans effectuer d'acquisition. La démonstration se termine à la fin de la période indiquée, quelle que soit la phase dans laquelle elle se trouve.
--framerate, -fps	configure le nombre d'images par seconde pour enregistrer le fichier de sortie. Le nombre d'images minimum par seconde autorisé est de 2 et le maximum est de 30.
--penc, -e	montre l'image dans le « Preview » après l'opération de codage. Visualise la fenêtre de « Preview » après la compression, cela permet de mettre en évidence tous les artefacts dans la même fenêtre. Dans les conditions normales, la fenêtre « Preview » affiche les images prises par la caméra avant compression. Il n'est pas certain que cette fonction soit maintenue dans les futures versions.

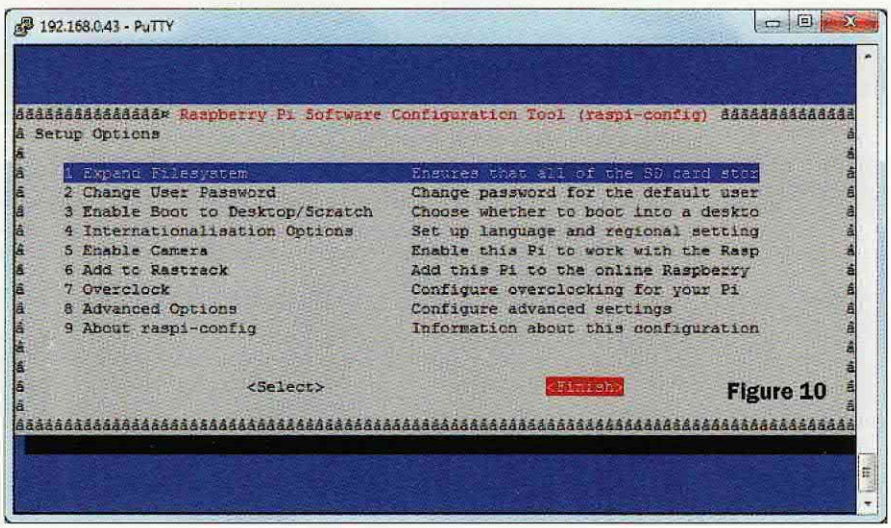


Figure 10

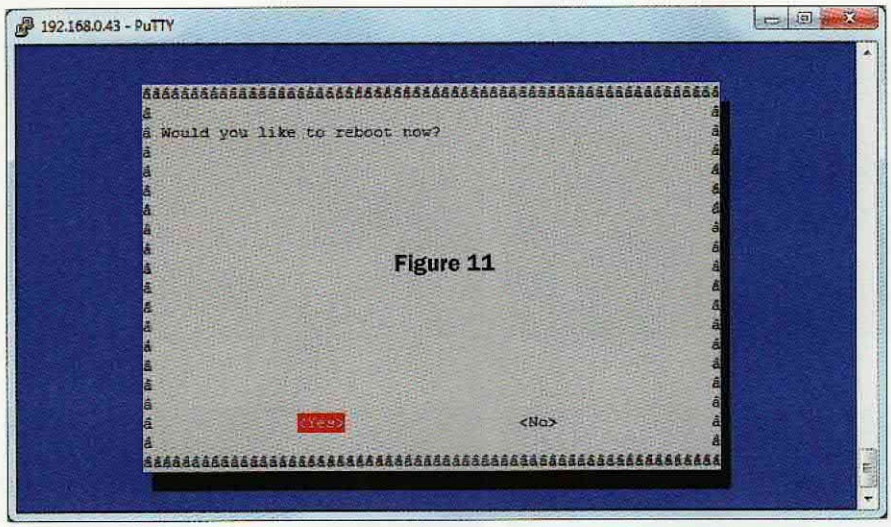


Figure 11

```
raspivid -t 5000 -o video.h264 -f 5
```

Pour enregistrer une vidéo de 5 s dans un fichier, tapez la commande :

```
raspivid -t 5000 -o - > mon_file.h264
```

Utilisez « Caméra Pi » à partir d'un programme en Python

Il est évident que les commandes à l'aide d'une fenêtre de terminal ne représentent pas la seule possibilité d'utiliser la « Caméra Pi ».

Pratiquement tout ce qui peut être exécuté à l'aide de lignes de commandes peut également être géré par un programme.

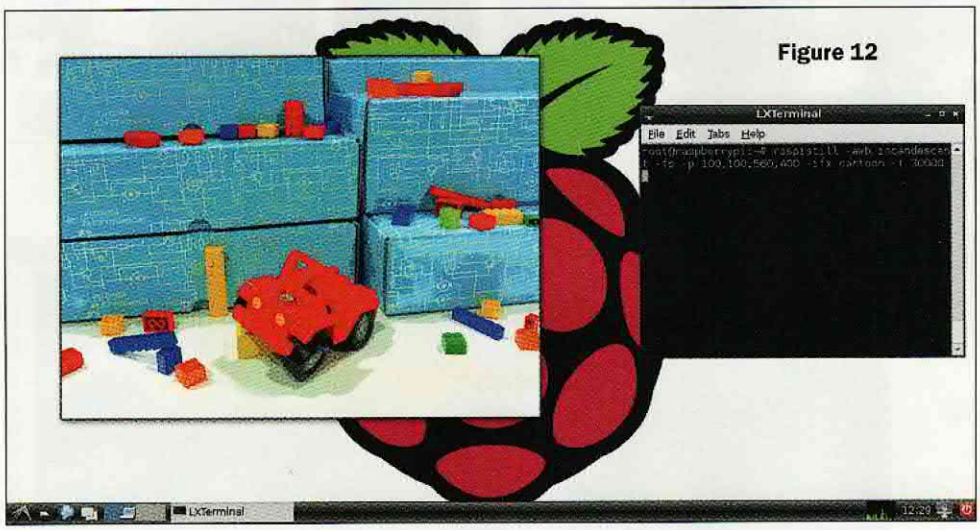


Figure 12

Par exemple, un programme peut contrôler la caméra via une interface Web ou automatiser la capture et le stockage d'images ou de vidéos. Mais pas seulement, les images peuvent être acheminées vers un autre programme qui peut être un logiciel de traitement d'images.

Pour gérer la « Caméra Pi » à l'aide d'un programme en Python, nous avons utilisé un programme de gestion créé par Dave Jones.

Figure 13 : normal.

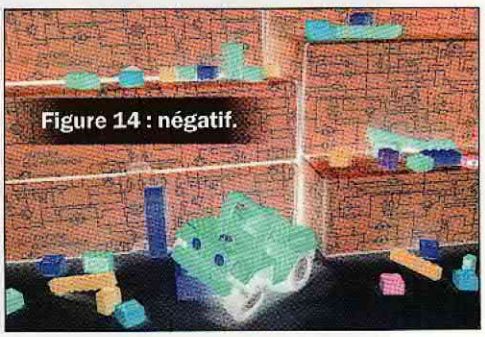


Figure 14 : négatif.

Ce programme est contenu dans la distribution Raspbian, nous pouvons donc l'installer avec les commandes familières :

```
apt-get update
apt-get install python-picamera
```

Le paquet fournit un certain nombre de fonctions que nous étudierons dans les prochains articles.

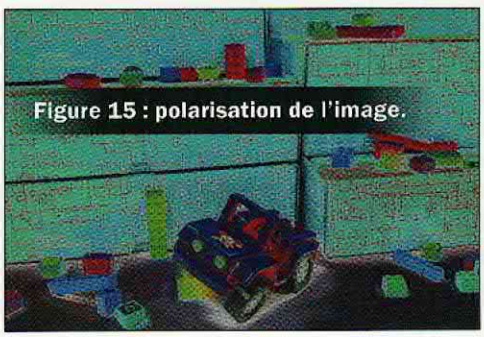


Figure 15 : polarisation de l'image.

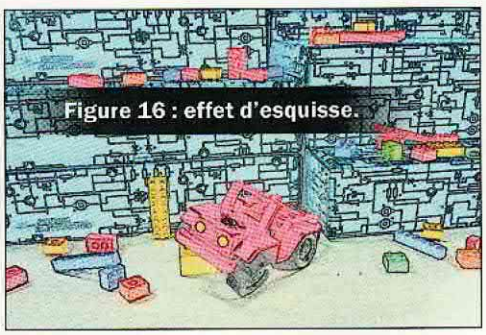


Figure 16 : effet d'esquisse.

Pour le moment, contentons-nous de répéter à l'aide d'un programme les configurations que nous avons effectuées à partir des lignes de commandes.

Le programme du Listing 1 permet de prendre une photo après un retard de 2 secondes, et de l'enregistrer dans un fichier.

Nous avons utilisé la méthode « **camera.start_preview()** » pour commencer la phase de « **Preview** » et, au bout de deux secondes avec la méthode « **camera.capture** » l'acquisition de l'image et son enregistrement dans un fichier.

Le programme du **Listing 2** permet de capturer une vidéo d'une durée de 60 secondes au format 640 x 480 pixels.

Il existe de nombreuses autres fonctions, par exemple l'enregistrement d'une vidéo sur plusieurs fichiers, la capture d'images lors d'un enregistrement vidéo, l'enregistrement en continu, etc.

Toutes les options présentées dans cet article seront essentielles pour les prochains articles, dans lesquels nous présenterons des systèmes simples de traitement d'images. Dans le traitement d'images et de vidéos, il est très important de gérer les paramètres d'acquisition, tels que l'éclairage, la mise au point, le format, la résolution, la balance, la couleur ou le noir et blanc (ou les deux).

Toutes les fonctions sont implémentées dans le logiciel de gestion « **Caméra Pi** », nous verrons comment les mettre à profit. Pour l'instant amusez-vous en expérimentant les différentes options disponibles. Suite à de nombreux courriers de nos lecteurs, dans le prochain numéro nous ferons un tour d'horizon sur l'utilisation du RaspberryPi 2.

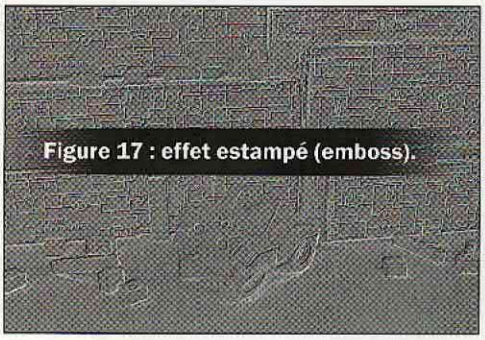


Figure 17 : effet estampé (emboss).

Figure 18 : effet peinture à l'huile (oilpaint).

Figure 19 : effet dessin avec tirets (hatch).

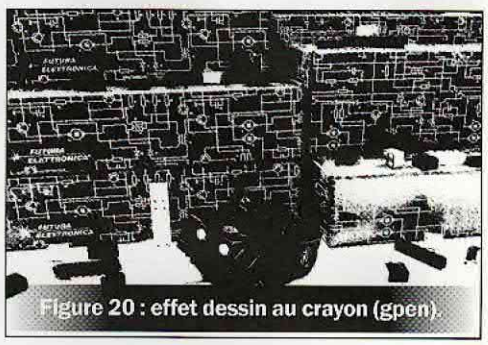


Figure 20 : effet dessin au crayon (gpen).

Listing 1

```
import time
import picamera

with picamera.PiCamera() as camera:
    camera.resolution = (1024, 768)
    camera.start_preview()
    camera.image_effect = none

# Retard d'acquisition
    time.sleep(10)

#camera.capture('photo.jpg')
    camera.image_effect = 'sketch'
    time.sleep(10)
    camera.image_effect = 'emboss'
    time.sleep(10)
    camera.image_effect = 'negative'

    time.sleep(10)
```

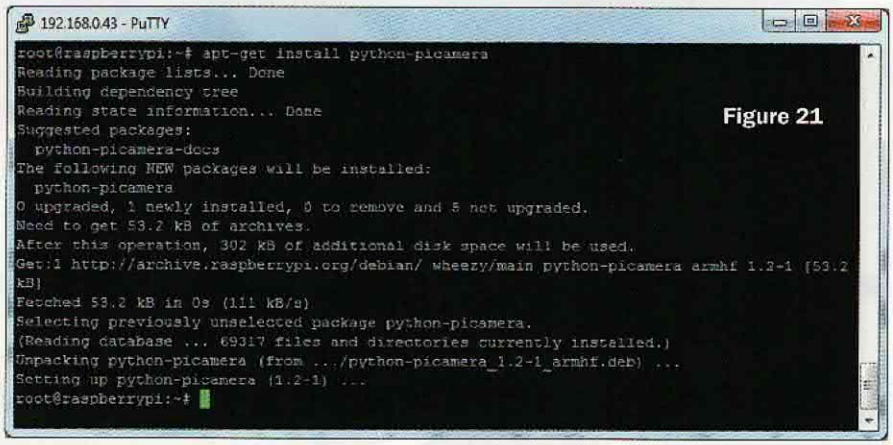


Figure 21

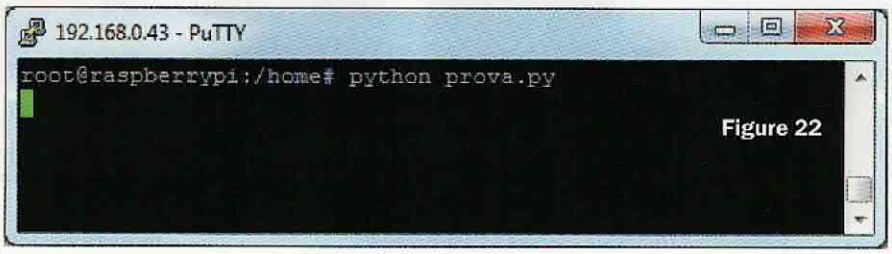


Figure 22

Listing 2

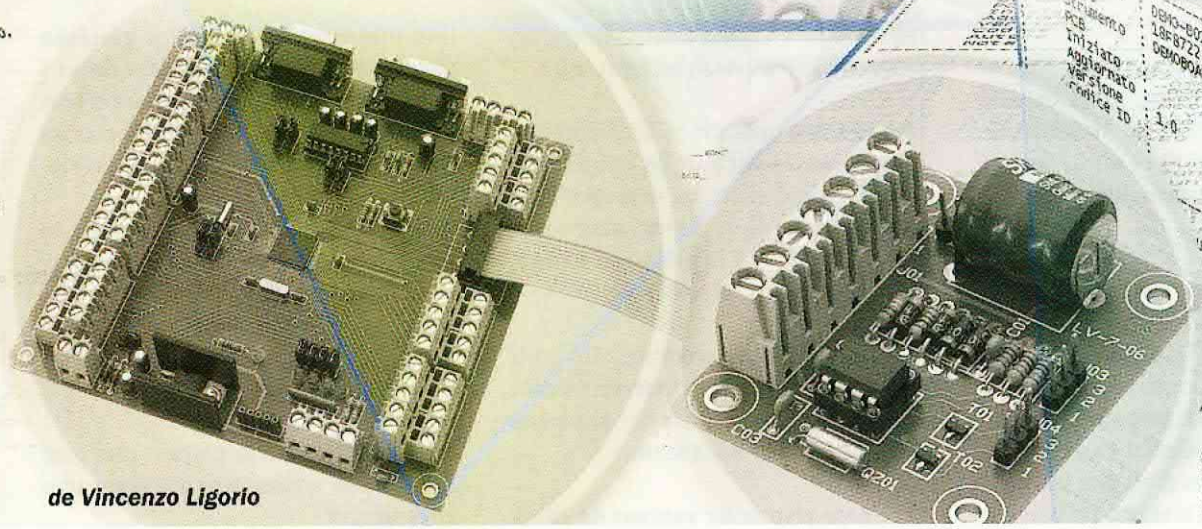
```
import picamera

with picamera.PiCamera() as camera:
    camera.resolution = (640, 480)
    camera.start_preview()
    camera.start_recording('video.h264')
    camera.wait_recording(60)
    camera.stop_recording()
    camera.stop_preview()
```

DEMOBOARD - PIC18F

Platine de développement pour PIC18F8XXX

Deuxième partie

Nous analyserons, dans cette deuxième partie, le firmware de la platine de développement pour PIC18F8XXX publiée dans le précédent numéro 136 d'Electronique et Loisirs Magazine. Les routines, qui sont écrites en Assembleur pour PIC, sont optimisées pour une utilisation dans l'environnement de développement MPLAB de Microchip, mais aussi pour différents compilateurs tels que le PICBASIC et le CCS. Vous pourrez inclure ces routines dans des portions dédiées au code ASM. Nous vous proposerons aussi une routine de débogage via le port série à l'aide de l'émulateur de terminal réalisé en Visual Basic.



de Vincenzo Ligorio

Dans l'article précédent, nous avons abordé l'aspect matériel de cette platine de développement pour PIC18F8XXX qui se compose d'une carte principale (carte mère) basée sur un microcontrôleur PIC18F8723, et de 4 périphériques : un module RTC, un clavier, une interface pour afficheur LCD alphanumérique et un circuit de gestion de l'alimentation.

Comme nous l'avons mentionné en introduction, les routines du firmware (c'est-à-dire une partie du programme du PIC) sont écrites en Assembleur et optimisées. Le logiciel de communication avec la carte est quant à lui écrit en Visual Basic. Il peut être utilisé pour communiquer avec des prototypes à base de microcontrôleurs de la famille PIC18F.

Dans notre carte de développement, nous avons intégré un microcontrôleur haut de gamme de la famille PIC 18. Par conséquent, toute d'application conçue pour fonctionner sur cette carte peut être facilement adaptée pour fonctionner sur des plateformes utilisant un microcontrôleur différent,

à condition d'adapter les différentes routines aux lignes d'entrées/sorties de celui-ci.

Le PIC18F8723 dispose de 10 ports d'entrées/sorties (PORTA à PORTJ). Les PIC18F de taille inférieure sont compatibles avec les instructions (routines) mais disposent d'un nombre de « PORT » inférieur et aussi d'une quantité moindre de mémoire FLASH. Il faudra donc adapter les programmes selon le microcontrôleur utilisé. Nous allons étudier maintenant la partie firmware de cette carte de développement.

Diagramme de flux (flow chart) des librairies

Un **diagramme de flux** ou **organigramme** ou encore « **flow chart** » est un **graphe représentant les flux des données** circulant lors de l'**exécution d'un programme** voire de plusieurs processus à l'intérieur dudit programme. En règle général, une platine de développement dispose d'un microcontrôleur non programmé dans lequel il faut « introduire » un programme exécutant une tâche souhaitée.

Par conséquent, pour développer et tester une application, nous devons écrire un programme en Assembleur, mais aussi en BASIC et en C, qui utilise une ou plusieurs routines prêtes à l'emploi.

Ces routines en assembleur sont mises à disposition par le fabricant du microcontrôleur (Microchip dans notre cas). Elles sont contenues dans les fichiers « **DEMO-01.asm** » et « **DEMO-01.inc** », respectivement l'archive des routines et le fichier d'en-tête (header) contenant les déclarations de tous les registres et de toutes les variables dont le premier fichier a besoin.

Si nous voulions décrire le fichier « DEMO-01.asm » avec un organigramme, nous obtiendrions le diagramme de flux de la figure 1.

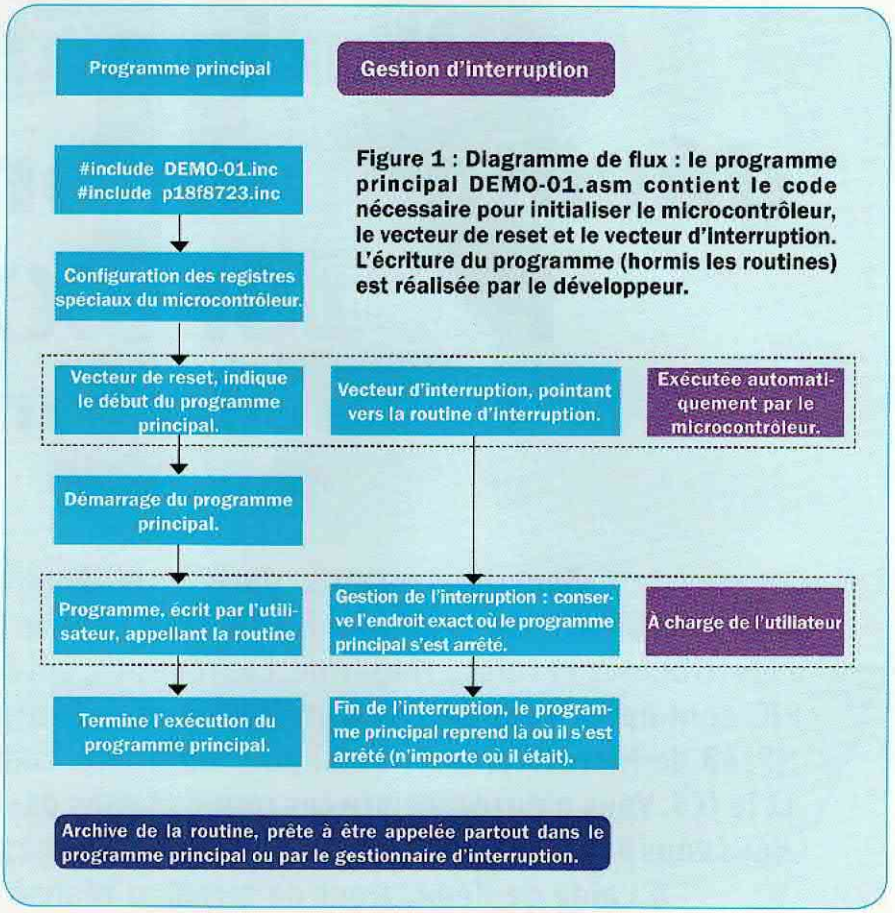


Figure 1 : Diagramme de flux : le programme principal DEMO-01.asm contient le code nécessaire pour initialiser le microcontrôleur, le vecteur de reset et le vecteur d'interruption. L'écriture du programme (hormis les routines) est réalisée par le développeur.

En pratique, le programme charge le fichier « header » spécifique à la platine de développement (DEMO-01.inc) et un fichier spécifique au microcontrôleur (PIC18F8723.inc) qui configure les registres du PIC (oscillateur, watchdog, programmation basse tension, etc.).

Une fois ces opérations préliminaires effectuées, le vecteur de réinitialisation (l'endroit exact où le programme commence réellement) amène le microcontrôleur à la première instruction du programme principal.

À ce stade, le programme réel écrit par l'utilisateur commence, en faisant usage de **routines** qui sont appelées lorsque cela est nécessaire.

Au même niveau du **vecteur de réinitialisation** nous pouvons trouver le **vecteur d'interruption** (IRQ, Interrupt ReQuest), c'est-à-dire l'emplacement de la mémoire où le programme « saute immédiatement » dès qu'une interruption a lieu.

Mais qu'est-ce qu'une **interruption** ? Il s'agit d'un **événement qui suspend immédiatement l'exécution du programme principal et qui exécute une partie du code avec la plus haute priorité**.

Une interruption peut être générée par le déclenchement d'un « timer » (minuterie), par le changement d'état d'une entrée, par la réception d'un caractère spécifique sur le port série.

Une interruption est utile lorsque le programme doit effectuer des opérations particulières dans un temps très court, ou de manière aléatoire. Une interruption a la priorité sur l'exécution du programme principal.

Cependant, dans la plupart des cas, il n'est pas nécessaire d'utiliser une interruption.

Dans notre code, nous avons prévu 2 vecteurs « **IRQ** » (2 interruptions) qui dans la pratique ne sont pas utilisés. Il appartiendra au programmeur de décider s'il veut écrire un programme en exécutant une instruction après l'autre ou utiliser des interruptions.

Pour conclure sur l'interruption, nous pouvons la considérer comme un événement spécial, qui lorsqu'il se produit, arrête immédiatement l'exécution du

programme principal et lance un programme secondaire qui exécute des procédures particulières.

Une fois l'interruption terminée, elle indique de nouveau au microcontrôleur de reprendre le cheminement du programme principal à l'endroit où il a été arrêté.

Analysons la structure de base de la librairie, en abordant les différentes routines. Comme il y a beaucoup de routines, nous ne pouvons pas les étudier toutes en détail par manque de place.

Nous nous limiterons à étudier les plus importantes et celles qui sont généralement les plus utilisées. Les 2 fichiers « DEMO-01.asm » et « DEMO-01.inc » sont largement commentés, vous pouvez les télécharger sur notre site dans le sommaire détaillé de la revue 137 à l'onglet « Télécharger ».

Nous commençons la description de la routine d'initialisation et du démarrage du microcontrôleur. Nous avons dû adapter la routine originelle de Microchip selon les besoins de notre projet et du microcontrôleur utilisé.

Le début du programme

La première des choses à faire, lorsque vous écrivez un programme en Assembleur, BASIC ou en C qui utilise des routines, est d'inclure au début du listing le fichier « header » nécessaire au compilateur pour créer le code hexadécimal qui sera chargé dans le PIC.

Reportez-vous au Listing 1, les 2 instructions pour le compilateur sont :

```
#include DEMO-01.inc
#include p18f8723
```

Elles communiquent au compilateur que les deux fichiers déclarés avec « **#include** » contiennent tous les « **alias** » (noms équivalents) des **registres utilisés** par les différentes routines.

Supposons, par exemple, qu'une routine utilise la variable « **MAVAR1** » pour effectuer des opérations.

Cette variable doit être déclarée (par exemple dans un fichier header ou au début du programme) et affectée à un emplacement de la mémoire au format hexadécimal, étant donné que le compilateur ne peut pas connaître la signification de la variable « **MAVAR1** ».

En la déclarant, par exemple, soit dans le programme soit dans un fichier header (par exemple « my_vars.inc ») selon la syntaxe suivante :

```
MAVAR1 EQU H'010'; variable
MAVAR2 EQU H'011'; variable
```

le compilateur, lors de la compilation du programme, sait qu'à chaque répétition de alias « **MAVAR1** » il doit faire référence à l'emplacement de la RAM à l'adresse « 0x010 ».

Généralement, immédiatement après les lignes « **#include** », se trouvent les lignes des définitions du mode de fonctionnement du matériel (hardware). Le microcontrôleur a des caractéristiques particulières qui souvent doivent être configurées uniquement pendant la phase de programmation.

Par exemple le type d'oscillateur (quartz XT ou HS, oscillateur interne, oscillateur RC externe, etc.), l'activation ou non du « watchdog » (c'est un compteur, qui une fois démarré, doit être remis périodiquement à 0). Lorsque le programme bloque quelque part, le « watchdog » réinitialise le microcontrôleur pour le faire redémarrer.

Ces définitions seront toujours identifiées par l'instruction « **__CONFIG** » suivie de « **_CONFIG<byte><registre/valeur>**.

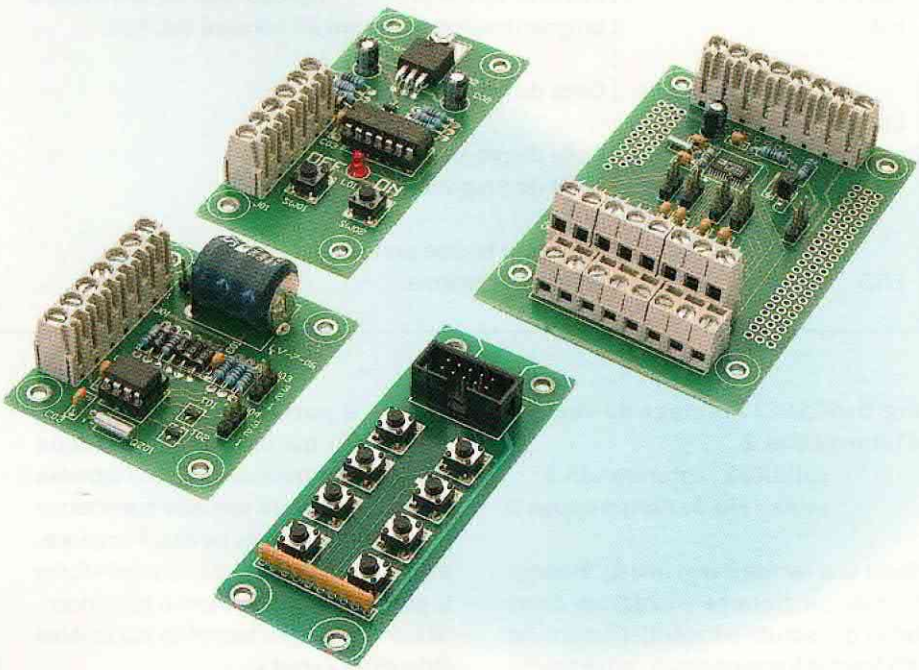
Par exemple les lignes suivantes :

```
_CONFIG _CONFIG2H,
        _WDT_OFF_2H
```

indiquent au compilateur que le « **watchdog** » est désactivé.

Immédiatement après la configuration des registres spéciaux, nous trouvons les **pointeurs des vecteurs de reset** et **d'interruption**, identifiés par les instructions :

```
org 0x0000 ; Pointeur du vecteur de reset
        goto start ; Va au label start
org 0x0008 ; Pointeur du vecteur d'interruption 1
        call IRQ1 ; Interruption 1
        retfie ; Fin de l'interruption 1
```



Listing 1

```
;-----------------------------------------------------------------------------------------------
; Nom du fichier  : DEMO-01.asm
; Micro           : 18F8723
; Dispositif      : DEMOBOARD
; Version         : 1.0
; Auteur : Electronique et Loisirs Magazine
;-----------------------------------------------------------------------------------------------
;*** Initialisation de la structure du microcontrôleur ***

#includeDEMO-01.inc
#includep18f8723.inc

__CONFIG  _CONFIG1H,  _OSC_HS_1H
__CONFIG  _CONFIG2L,  _BOREN_SBORDIS_2L & _BORV_43_2L & _PWRT_ON_2L
__CONFIG  _CONFIG2H,  _WDT_OFF_2H
__CONFIG  _CONFIG3L,  _MODE_MC_3L
;__CONFIG  _CONFIG4L,  _LVP_OFF_4L & _DEBUG_OFF_4L & _XINST_OFF_4L
__CONFIG  _CONFIG4L,  0x8B
__CONFIG  _CONFIG5H,  0xC0
__CONFIG  _CONFIG6H,  0xE0

;*** Vecteur de reset et d'interruption ***

org 0x0000              ; Pointeur du vecteur de réinitialisation.
        goto start      ; Point au label start (début).
org 0x0008              ; Pointeur vers le premier vecteur d'interruption.
        nop             ; Les instructions doivent être écrites ici.
        retfie          ; Retour d'interruption.
org 0x0018              ; Pointeur vers le deuxième vecteur d'interruption.
        nop             ; Les instructions doivent être écrites ici.
        retfie          ; Retour d'interruption.


;***  Démarrage du code écrit par le programmeur  *****

org 0x0300              ; Pointeur vers le premier emplacement de la mémoire
Init                    ; programme. Init se trouve à l'adresse 0x0300.
        ;....           ;
        nop             ; Code du programmeur.
Start                   ;
        nop             ; Code du programmeur.
        nop             ; Code du programmeur.
        ;...            ;
        goto start      ; Répète la boucle principale
END                     ; Fin du programme.
```

```
org 0x0018  ; Pointeur du vecteur d'interruption 2
        call IRQ2 ; Interruption 2
        retfie ; Fin de l'interruption 2
```

Après une remise à zéro (reset), le compteur du programme (« program counter ») qui scrute séquentiellement les différentes instructions du programme, débute à partir de l'emplacement « 0x0000 » qui contient l'instruction indiquant au microcontrôleur l'adresse exacte à partir de laquelle commence le programme. Dans ce cas, l'emplacement « 0 » du « program counter » force le programme à exécuter un saut inconditionnel à la ligne identifiée par le label (étiquette) « start ».

Lors de la réinitialisation, les deux événements d'interruption sont immédiatement sautés par le « **program counter** » pour aller aux emplacements « **0x0008** » et « **0x0018** » qui à leur tour appellent les routines de gestion des événements et y mettront fin avec l'instruction « retfie », laquelle indique au microcontrôleur que la gestion de l'interruption est terminée.

Listing 2

```
*** Appels des routines d'initialisation ***

        call inizPORTS          ; Initialisation des PORT
        call inizREGISTERS      ; Initialisation des registres
        call inizADinter        ; Initialisation des convertisseurs AD internes.
        call inizUART1          ; Initialisation de l'UART1
        call inizDISPLAY        ; Initialisation de l'afficheur

;************
```

Listing 3

```
;*** Routine d'initialisation des différents PORT  ***

inizPORTS                       ; Nom de la routine
                                ; MSB       LSB
                                ; PORTA x = 00000000
                                ; Direction du port : 0 = output (sortie), 1 = input (entrée)
        movlw   b'00000000'     ; Le port A est initialisé à 00.
        movwf   PORTA           ;
        movlw   b'11000111'     ; Direction des broches comme spécifiées (dépend de l'hardware)
        movwf   TRISA           ;
        ...
        ...
        ...
        movwf   PORTJ           ;
        movlw   b'00000000'     ; le PORTJ à toutes ses broches configurées en sortie
        movwf   TRISJ           ;
        return                  ; Fin de la routine d'initialisation des PORT

;************
```

Listing 4

```
;*** Routine d'initialisation des différents registres ***

inizREGISTERS           ; Nom de la routine.
        movlw   0xFF    ; xxxxx111 Aucun comparateur analogique sur le PORTF
        movwf   CMCON   ;
        movlw   0x0F    ; xxxx1111 Pas d'entrées analogiques
        movwf   ADCON1;
        return          ; Fin de la routine d'initialisation des registres

;************
```

À partir de là commence le programme réel écrit par le développeur (programmeur). Après la déclaration des vecteurs de reset, généralement nous trouvons les routines d'initialisation des registres, des périphériques, des variables, etc. Immédiatement après, le programme principal est exécuté de manière cyclique.

Ici intervient les 5 routines qui appellent les instructions visibles dans le Listing 2. Elles doivent être **insérées immédiatement après le label « Init »** (voir le Listing 1), avant le commencement de la boucle principale identifiée par le label « start ». La 1ère routine « **inizPORTS** » (Listing 3) initialise tous les PORT du microcontrôleur.

En pratique, la routine configure tous les registres « TRIS » et les différents PORT. La direction de chaque broche est configurée par le registre « TRIS » (en entrée ou en sortie) à l'aide d'un simple bit. Dans le cas où le bit est configuré en sortie, il est nécessaire d'indiquer le niveau logique que doit prendre le PORT au démarrage.

Voici un exemple de déclarations :

```
movlw   b'00000000'
movwf   PORTA
```

initialise tous les bits du port A, tandis que l'instructions suivante :

```
movlw   b'11000111'
movwf   TRISA
```

indique au microcontrôleur que les broches 0, 1, 2, 6 et 7 du port A sont définies en entrées tandis que les broches 3, 4 et 5 sont définies en sorties.

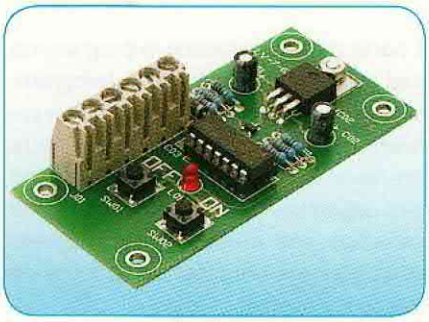
La deuxième routine d'initialisation « **inizREGISTERS** » (Listing 4) permet de configurer les registres particuliers des différents périphériques.

Dans l'exemple du Listing 4, il est défini que les comparateurs analogiques du « PORTF » soient désactivés car il n'est pas prévu d'utiliser des entrées analogiques.

Ici, il est possible d'insérer d'autres « initialisations » comme par exemple l'initialisation d'un « TIMER », d'un compteur, de différents périphériques, tout cela en fonction de ce que le programmeur a besoin pour son projet.

En fait, la routine fixe une ligne directrice, mais ne prend pas en compte toutes les initialisations que le PIC18F8723 peut accepter.

La routine suivante, appelée « **inizADinter** » (Listing 5) est utilisée pour initialiser le convertisseur A/D interne. Là encore, il s'agit d'une ligne directrice pour 95 % des cas.



Listing 5

```
; *** Routine d'initialisation du convertisseur A/D interne ***
; *** 3 entrées sont utilisées ***

inizADinter
        movlw   0x01        ; A/D interne
        movwf   ADCON0      ; canal 0  activé
        movlw   0x0C        ;
        movwf   ADCON1      ; Rif=+5V/GND - 3  canaux analogiques,
        movlw   0x3E        ;
        movwf   ADCON2      ; résolution 8 Bits - Tacq = 20Tad (temps d'acquisition) - Clock/64.
        return              ; Fin de la routine

;************
```

Listing 6

```
;*** Routine d'initialisation de l'UART1 ***

inizUART1                   ; Quart 10 MHz, 8 bits pas de parité
                            ; 2 bits de STOP, pas de contrôle de flux
                            ; baud rate 9600 avec Qx= 10 MHz
        movlw   0x00        ;
        movwf   BAUDCON1    ;
        movlw   0x40        ; ( BRGH=1 ) 64=40
        movwf   SPBRG1      ;
        movlw   0x04        ; Asynchrone, Hi-speed, 8 bits tx, tx-désactivé
        movwf   TXSTA1      ;
        movlw   0x00        ; pas d'UART pour le reset
        movwf   RCSTA1      ;
        nop                 ;
        movlw   0x80c       ; Bits du port pour l'UART, 8 bit rx,
                            ; rx-désactivé, pas d'erreur.
        movwf   RCSTA1      ;
        bsf     TXSTA1,TXEN ; TX activé.
        bsf     RCSTA1,CREN ; RX activé.
        return              ; Fin de la routine

;************
```

Listing 7

```
;*** Routine d'initialisation de l'afficheur ***

inizDISPLAY                     ; Nom de la routine.
        call    msec100         ; Retard de 100 msec.
        call    msec100         ; Retard de 100 msec.
        bcf     PORTD,0         ; une commande doit être envoyée
        movlw   0x01            ; 01 = Clear
        movwf   PORTE           ; sur le bus de l'afficheur
        bcf     PORTD,1         ; bit d'impulsion basse
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; bit d'impulsion haute
        call    msec10          ; la commande a besoin de 1,64 ms
        bcf     PORTD,0         ; une commande doit être envoyée
        movlw   0x3C            ; 3C = 8bit - 2/4 lignes - générateur de caractères
        movwf   PORTE           ; sur le bus de l'afficheur
        bcf     PORTD,1         ; bit d'impulsion basse
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; bit d'impulsion haute
        call    msec10          ; la commande a besoin de 40 ms
        bcf     PORTD,0         ; une commande doit être envoyée
        movlw   0x0C            ; 0C= afficheur activé – pas de curseur
        movwf   PORTE           ; sur le bus de l'afficheur
        bcf     PORTD,1         ; bit d'impulsion basse
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; bit d'impulsion haute
        call    msec10          ; la commande a besoin de 40 ms
        bcf     PORTD,0         ; une commande doit être envoyée
        movlw   0x06            ; 06= pas de décalage
        movwf   PORTE           ; sur le bus de l'afficheur
        bcf     PORTD,1         ; bit d'impulsion basse
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; bit d'impulsion haute
        call    msec10          ; la commande a besoin de 40 ms
        return                  ; Fin  de la routine.

;************
```

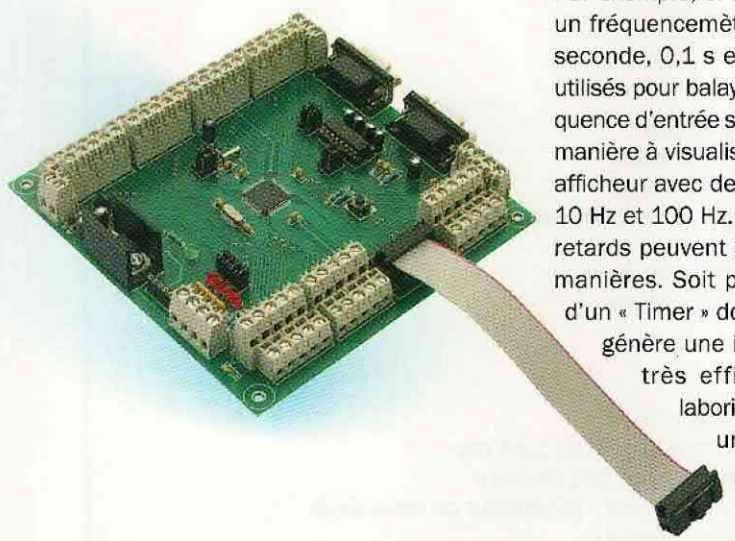
Pour les autres cas, vous devrez configurer les différents bits de chaque registre en fonction de vos besoins. Ici, nous avons activé le canal 1 du convertisseur A/D interne, configuré pour fonctionner avec seulement 3 canaux analogiques, avec une tension de référence interne, avec une résolution de 8 bits et avec une fréquence d'horloge égale à 1/64 de la fréquence du système.

Notez que le convertisseur A/D est configuré au moyen des registres ADCON0, ADCON1 et ADCON2.

Ensuite dans le programme est appelée la routine d'initialisation de l'UART « **inizUART1** » (Listing 6) qui, comme vous pouvez le voir, est le premier des 2 UART disponibles dans le PIC. Par rapport au Listing 5 de configuration du convertisseur A/D, ici il y a plus d'opérations à exécuter car il y a plus de registres impliqués. La **vitesse de transmission** (baud rate) est définie par la configuration des **registres** « **BAUDCON1** » et « **SPBRG1** » qui sont programmés avec des valeurs hexadécimales qui dépendent de l'horloge du système.

Ces valeurs sont calculées grâce à des formules disponibles dans la documentation technique du microcontrôleur.

L'activation des lignes de transmission et de réception s'effectue à l'aide des **registres** « **TXSTA1** » et « **RCSTA1** ». Dans notre exemple, l'UART est configuré pour fonctionner à 9600, N, 8, 1, sans contrôle de flux, à partir de la fréquence de base de 10 MHz. Pour régler la vitesse en fonction de vos besoins, vous devez paramétrer les registres comme décrit ci-dessus.



La dernière routine d'initialisation « **inizDISPLAY** » (Listing 7) est utilisée pour initialiser un afficheur ayant un contrôleur de type Hitachi HD44780 ou équivalent.

Le listing est largement commenté pour la compréhension de la routine. Bien évidemment selon votre application il sera nécessaire de la modifier en fonction de vos besoins.

Les diverses routines

Nous savons que chaque routine que nous allons décrire ci-après utilise beaucoup de variables de « support » (toutes définies dans le fichier header « DEMO-01.inc » sous la forme de déclarations de registres et de variables) dont certaines sont utilisées comme une « entrée » en fonction de la routine et d'autres comme une « sortie ».

Pour simplifier à l'aide d'un exemple, considérons une routine qui effectue la multiplication entre A et B, en utilisant deux registres (variables) dans lesquels sont écrites les valeurs de A et B, et un troisième registre dans lequel vous pouvez lire le résultat.

Routine de temporisation et de retard

Largement présentes dans tous les programmes, les routines de temporisation et de retard sont utilisées pour gérer et générer des événements à des cadences précises.

Par exemple, si nous voulons réaliser un fréquencemètre, des retards d'une seconde, 0,1 s et 0,01 s peuvent être utilisés pour balayer l'intervalle de la fréquence d'entrée sur un calibre donné, de manière à visualiser la fréquence sur un afficheur avec des résolutions de 1 Hz, 10 Hz et 100 Hz. En règle générale, les retards peuvent être générés de deux manières. Soit par la programmation d'un « Timer » dont la fin du comptage génère une interruption (méthode très efficace, mais un peu laborieuse), soit en générant une « boucle récursive » qui, une fois initiée, effectue un certain nombre d'opérations et de répétitions calculées, en tenant compte soit du nombre d'opérations exécutées soit des « cycles machine » nécessaires pour effectuer chaque opération ou instruction.

Le listing 8A montre le code d'une routine qui génère un retard de 100 microsecondes à partir d'une fréquence d'horloge de 10 MHz. Elle est appelée à partir du programme principal avec l'instruction :

```
call      microsec100
```

ou avec une instruction similaire si vous utilisez un environnement de développement différent (l'**appel de chaque sous-routine est effectué** à l'aide de l'**instruction « call »**). Immédiatement après l'appel, est chargé dans le registre « WAITR1 » la valeur hexadécimale 0x32, qui correspond à 50 en décimal avec une durée de 0,8 µs. Le cycle est ensuite exécuté 49 fois (de 50 à 1) :

```
nop                    ; (0,4 µs)
nop                    ; (0,4 µs)
decfsz WAITR1,1        ; (0,4 µs)
goto microsec100A      ; (0,8 µs)
```

et dure 2 microsecondes à chaque fois, pour un total de 98 microsecondes. Ensuite 2 instructions sont exécutées :

```
decfsz   WAITR1,1
```

lorsque le registre est égal à 0 (0,4 µs) et :

```
return
```

exécutée en 0,8 µs.

Le temps nécessaire pour exécuter la routine en partant de l'appel jusqu'à la sortie correspond par conséquent à :

- 0,8 microsecondes pour l'initialisation ;
- 98 microsecondes pour l'exécution de la boucle de comptage ;
- 0,4 microsecondes pour identifier que le registre « WAITR1 » R1 est égal à zéro ;
- 0,8 microsecondes pour revenir au point du programme où l'appel de la sous-routine a été effectué, soit un total de 100 µs.

En utilisant de manière appropriée un ou plusieurs registres « WAITR », initialisé(s) avec des valeurs hexadécimales calculées, il est possible de générer des routines de temporisation qui exécutent un ou plusieurs cycles de décrémentation récursive.

Par exemple le listing 8B montre comment générer un retard de 1 ms en utilisant les registres « WAITR1 » et « WAITR2 », dans lesquels le premier exécute 5 cycles de 0,2 milliseconde soit un total de 1 ms et le second 99 cycles de 2 microsecondes (0,2 ms en fait).

Chaque fois que le programme effectue 99 cycles de 2 microsecondes, il appelle (pour une durée calculée de 1 µs) la routine de comptage des cycles de 0,2 ms, après quoi la routine est terminée.

Si vous désirez faire un peu d'arithmétique, vous pouvez ajouter toutes les durées nécessaires à l'exécution de chaque instruction, vous vérifierez que la routine est exécutée exactement en 1 ms.

Gardez à l'esprit que, pour des temporisations longues, vous devez utiliser plusieurs registres de comptage.

La routine « sec1 » (non représentée dans ces pages, mais qui figure dans le programme complet téléchargeable sur notre site), a besoin de 3 registres « WAITR1 », « WAITR2 » et « WAITR3 ».

Si vous travaillez à une fréquence de 10 MHz, toutes les routines de temporisation décrites dans cet article peuvent être appelées et exécutées dans un laps de temps correct sans aucune modification.

Listing 8A

```
;*** Routine de retard ***

; Les routines sont calculées pour une fréquence d'horloge de 10 MHz, les registres WAITR1, WAITR2 et WAITR3 sont utilisés.
; Certaines routines sont assez courtes, par conséquent des instructions NOP ont été ajoutées pour calibrer la
; la durée avant la fin de l'exécution.
; Cela permet de remplacer les différentes instructions NOP si nécessaire, avec des "goto", "bsf","bcf",
; tout en conservant la durée de la routine.
; Temps d'exécution des instructions (en microsecondes).
; bsf  = 0,4      bcf   = 0,4
; call = 0,8      return = 0,8
; movlw = 0,4     movwf  = 0,4
; nop  = 0,4      decfsz = 0,4
; goto = 0,8
; Toutes les routines ont un seul niveau de profondeur pour éviter qu'un trop grand nombre d'appels sature la pile du registre.

;*** Routine de 100 microsecondes, utilise le registre WAITR1.

microsec100                     ; Nom de la routine.
        movlw   0x32            ;
        movwf   WAITR1          ; WAITR1 compteur de 50 cycles de 0,2 microsecondes
microsec100A                    ; Point d'entrée pour la répétition du cycle.
        nop                     ; 0,4 microsec.
        nop                     ; 0,4 microsec.
        decfsz  WAITR1,1        ; 0,4 microsec.
        goto    microsec100A    ; 0,8 microsec.
                                ; 2,0 microsec au total
        return                  ; 49 cycles 'C' = 98 microsecondes.
;************************
```

Listing 8B

```
;*** Routine de 1 milliseconde, utilise les registres WAITR1 et WAITR2.

msec1                           ; Nom de la routine.
        movlw  0x05             ;
        movwf  WAITR1           ; WAITR1 compteur de 5 cycles de 0,2 millisecondes
msec1A                          ; premier point d'entrée
        movlw  0x63             ;
        movwf  WAITR2           ; WAITR2 compteur de 99 cycles de 2 microsecondes
msec1B                          ; second point d'entrée.
        nop                     ; 0,4 microseconde
        nop                     ; 0,4 microseconde
        decfsz WAITR2,1         ; 0,4 microseconde
        goto msec1B             ; 0,8 microseconde
                                ; 2,0 microsecondes au total
                                ; 49 cycles 'C' = 98 microsecondes.
        nop                     ; 0,4 microseconde
        decfsz WAITR1,1         ; + 0,4 microseconde
        goto msec1A             ; + 0,8 microseconde
                                ; + 0,8 microseconde
                                ; = 1  milliseconde
        return                  ; Fin de la routine.

;************************
```

Cependant, les routines peuvent être adaptées facilement à vos besoins.

Par exemple, la routine de « 100 microsecondes » peut être modifiée pour une temporisation de 200 microsecondes, il suffit d'initialiser « WAITR1 » à 99 (0x63), afin d'obtenir 99 cycles de comptage pour un total de 198 microsecondes.

Routines pour les opérations mathématiques

Effectuer des opérations mathématiques avec des microcontrôleurs

Listing 9

```
; A * B = C
; 147.188 * 72.236 = 10.632.272.368
; 0x023ef4 * 0x011a2c = 0x0279bb99F0
movlw   0x2b        ; écrit le signe
movwf   MATH0       ; de A.
movlw   0x02        ; écrit MSD
movwf   MATH1       ; de A.
movlw   0x3e        ; écrit MSD-1
movwf   MATH2       ; de A.
movlw   0xf4        ; écrit LSD
movwf   MATH3       ; de A.
movlw   0x2b        ; écrit le signe
movwf   MATH4       ; de B.
movlw   0x01        ; écrit MSD
movwf   MATH5       ; de B.
movlw   0x1a        ; écrit MSD-1
movwf   MATH6       ; de B.
movlw   0x2c        ; écrit LSD
movwf   MATH7       ; de B.
call    multiplicationE ; multiplication
movf    MATH8,W     ; prélève le signe
movwf   RESSIGNE    ; résultat
movf    MATH9,W     ; prélève 6^ byte
movwf   RES6        ; (MSD) résultat
movf    MATHA,W     ; prélève 5^ byte
movwf   RES5        ; résultat
movf    MATHB,W     ; prélève 4^ byte
movwf   RES4        ; résultat
movf    MATHC,W     ; prélève 3^ byte
movwf   RES3        ; résultat
movf    MATHD,W     ; prélève 2^ byte
movwf   RES2        ; résultat
movf    MATHE,W     ; prélève 1^ byte
movwf   RES1        ; (LSD) résultat
```

Listing 10

```
;*** Routine de visualisation sur un afficheur ***

; Utilise le registre RA,
; le registre d'index 2 (FSR) pour lire la RAM (080..0CF),
; de la routine msec1 et microsec100.

Display                         ; lancement de la routine
        bcf     PORTD,0         ; l'afficheur reçoit une commande
        movlw   0x02            ; envoie de la commande
        movwf   PORTE           ; sur le bus de l'afficheur
        bcf     PORTD,1         ; active l'écriture
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; désactive l'écriture
        call    msec1           ; le contrôleur est de type 4 x 20
        call    msec1           ; attend
        lfsr    2,0x080         ; le registre d'index 2 pointe vers le 1er caractère
        movlw   0x14            ; doit envoyer les 20 premiers caractères ( à la 1ère ligne)
        movwf   RA              ; la séquence des lignes est  1 - 3 - 2 - 4.
        bsf     PORTD,0         ; l'afficheur reçoit une donnée.
display1                        ; écriture sur la ligne 1
        movf    POSTINC2,0      ; lit le registre d'index 2 et l'incrémente
        movwf   PORTE           ; écrit les données sur le PORTE.
        bcf     PORTD,1         ; active l'écriture
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; désactive l'écriture
        call    microsec100     ; attend 100 µs
        decfsz  RA,1            ; décrémente RA, si RA vaut 0 terminé
        goto    display1        ; écrit le caractère suivant sur la 1ère ligne
        lfsr    2,0x0A8         ; Fin du registre d'index 2 vers le 1er caractère
        movlw   0x14            ; envoi de 20 caractères à la 3ème ligne.
        movwf   RA              ; rafraîchit le registre RA
        bsf     PORTD,0         ; l'afficheur reçoit une donnée

display2                        ; continue à écrire
        ;...                    ; les 3 lignes de l'afficheur
```

Tableau 1 - Fonctions mathématiques

Appel de la routine	Bit /Signe	Fonction	Registre 1ère variable (A)	Registre 2ème variable (B)	Registre résultat (C)
soustractionE sommeE	24 2B(+) 2D(-)	A ± B = C, somme/soustraction A=<3 octets + signe> B=<3 octets + signe> C=<4 octets + signe>	MATH0[signe] MATH1[MSD] MATH2[] MATH3[LSD]	MATH4[signe] MATH5[MSD] MATH6[] MATH7[LSD]	MATH8[signe], MATH9[MSD] ... MATHC[LSD)
mutiplicationE	24 2B(+) 2D(-)	A * B = C, mutiplication A=<3 octets + signe> B=<3 octets + signe> C=<4 octets + signe>	MATH0[signe] MATH1[MSD] MATH2[] MATH3[LSD]	MATH4[signe] MATH5[MSD] MATH6[] MATH7[LSD]	MATH8[signe], MATH9[MSD] ... MATHE[LSD)
division128E ... division2E	48 2B(+) 2D(-)	A / 2^n = C, division d'un nombre de 6 octets par un nombre d'une puissance de 2. A=<6 octets + signe>	MATH8[signe], MATH9[MSD] ... MATHE[LSD)	Non applicable	MATH8[signe], MATH9[MSD] ... MATHE[LSD)
divisionE	48 2B(+) 2D(-)	A / B = C, division d'un entier A=<6 octets + signe> B=<3 octets + signe> C=<3 octets + signe> /0 ou overflow (OF): MATHF=01	MATH8[signe], MATH9[MSD] ... MATHE[LSD	MATH0[signe] MATH1[MSD] MATH2[] MATH3[LSD]	MATH4[signe] MATH5[MSD] MATH6[] MATH7[LSD] MATHF[erreur]
comparaisonE	24 2B(+) 2D(-)	A <=> B, comparaison A=<3 octets + signe> B=<3 octets + signe>	MATH0[signe] MATH1[MSD] MATH2[] MATH3[LSD]	MATH4[signe] MATH5[MSD] MATH6[] MATH7[LSD]	MATH8 00: A=B 01: A<B 02: A>B
binbcdE	24	A(Bin) conv C(BCD), conversion binaire vers BCD. A=<3 octets>, C=<8 octets>	MATH0[MSD] MATH1[] MATH2[LSD]	Variable de service MATH2... MATH5	MATH8[MSD] ... MATHF[LSD]
bcdbinE	8digit	A(BCD) conv C(bin), conversion BCD vers binaire A=<8 octets>, C=<3 octets>	MATH8[MSD] ... MATHF[LSD]	Variable de service MATH3... MATH5	MATH0[MSD] MATH1[] MATH2[LSD]
mutiplication	16 2B(+) 2D(-)	A ± B = C, somme/soustraction A=<2 octets + signe> B=<2 octets + signe> C=<4 octets + signe>	MATH0[signe] MATH1[MSD] MATH2[LSD]	MATH3[signe] MATH4[MSD] MATH5[LSD]	MATH6[signe] MATH7[MSD] ... MATHA[LSD]
division128 ... division2	32 2B(+) 2D(-)	A / 2^n = C, division d'un nombre de 4 octets par un nombre d'une puissance de 2. A=<4 octets + signe> C=<4 octets + signe>	MATH6[signe] MATH7[MSD] ... MATHA[LSD]	Non applicable	MATH6[signe] MATH7[MSD] ... MATHA[LSD]
division625 division125	24 2B(+) 2D(-)	A / nnn = C, division d'un nombre de 3 octets par 625 ou 125 A=<3 octets + signe> C=<2 octets + signe>	MATH6[signe] MATH8[MSD] MATH9[] MATHA[LSD]	Non applicable	MATH0[signe] MATH1[MSD] MATH2[LSD]
soustraction somme	16 2B(+) 2D(-)	A ± B = C, somme/soustraction A=<2 octets + signe> B=<2 octets + signe> C=<3 octets + signe>	MATH0[signe] MATH1[MSD] MATH2[LSD]	MATH3[signe] MATH4[MSD] MATH5[LSD]	MATH6[signe] MATH7[MSD] MATH8[] MATH9[LSD]
comparaison	16 2B(+) 2D(-)	A <=> B, comparaison A=<2 octets + signe> B=<2 octets + signe>	MATH0[signe] MATH1[MSD] MATH2[LSD]	MATH3[signe] MATH4[MSD] MATH5[LSD]	MATHA 00: A=B 01: A<B 02: A>B
binbcd	16	A(Bin) conv C(BCD), conversion binaire vers BCD. A=<2 octets>, C=<5 octets>	MATH0[MSD] MATH1[LSD]	Variable de service MATH2... MATH5	MATH8[MSD] ... MATHC[LSD]
bcdbin	5 digits	A(BCD) conv C(bin), conversion BCD vers binaire A=<5 octets>, C=<2 octets>	MATH8[MSD] ... MATHC[LSD]	Variable de service MATH2... MATH5	MATH0[MSD] MATH1[LSD]

non dotés d'une unité arithmétique et logique (ALU) est toujours assez complexe.

Une addition de 2 octets, qui est une opération en elle-même banale, nécessite 2 octets en entrée et 2 octets en sortie pour le résultat, qui en fait correspond à la somme de deux nombres pouvant chacun prendre une valeur maximale de 255 (en binaire « 11111111 »), soit au maximum 510 (en binaire « 111111110 »).

Les choses se compliquent rapidement lorsque vous devez effectuer une multiplication, une division ou des conversions de format (par exemple du format binaire vers le BCD ou vice-versa).

Ces opérations sont très fréquentes. Au fil du temps, de nombreux développeurs ont écrit et optimisé des routines mathématiques de divers types, routines que vous trouverez dans notre fichier « DEMO-01.asm ».

Ce que nous pouvons dire en premier pour les routines mathématiques, c'est que plus vous désirez une « **résolution** » importante du résultat, c'est-à-dire le nombre d'octets du « **word** » (mot) résultat, plus grand est le temps utilisé pour calculer le résultat.

Pour cette raison, nous avons divisé les routines mathématiques en deux groupes, l'une qui calcule le résultat en 16 bits (deux octets) et l'autre en 24 bits, avec pour les 2 groupes le signe positif ou négatif.

Il n'est pas prévu des calculs en virgule flottante, car ils nécessiteraient tellement de cycles machine qu'ils ralentiraient le système à un niveau inacceptable. Par conséquent, les résultats ne sont que des entiers.

Dans le cas de calculs sur 16 bits, chaque résultat est formaté en 3 octets, un octet pour le signe et deux autres octets pour le résultat (l'octet haut correspond aux bits 15 à 8, l'octet bas aux bits 7 à 0).

Pour les calculs sur 24 bits chaque résultat est constitué de 4 octets. Un octet pour le signe et 3 autres octets pour le résultat (l'octet haut correspond aux bits 24 à 16, l'octet moyen aux bits 15 à 8 et enfin l'octet bas aux bits 7 à 0).

Toutes les routines mathématiques s'appuient sur 16 registres spécifiques : MATH0, MATH1, ..., MATHF.

En appelant n'importe laquelle de ces routines, nous sommes certains de ne pas modifier par inadvertance l'un ou l'autre des emplacements mémoire (registres), puisque les données en entrée et en sortie sont transférées uniquement avec les registres spécifiés.

Sachez que pour décrire toutes ces routines complexes et les commenter, il faudrait beaucoup plus de pages.

Au lieu de décrire des centaines de lignes de code en Assembleur, nous avons préféré résumer le tout dans un tableau (voir le Tableau 1) qui explique comment les utiliser et les appeler.

En principe, chaque fonction mathématique nécessite une phase préparatoire précédant l'appel de la sous-routine.

Nous allons prendre un exemple pratique en supposant que nous souhaitons effectuer la multiplication de 2 nombres de 3 octets. La fonction à appeler est « mutiplicationE », qui effectue l'opération A * B = C.

Dans le Tableau 1, notez que les octets de la multiplication se trouvent dans les registres MATH0 à MATH3, que le multiplicateur se trouve dans les registres MATH4 à MATH7 et que le résultat se situe dans les registres MATH8 à MATHE.

Par conséquent, avant d'appeler la fonction, nous devons écrire le multiplicateur et la multiplication dans les différents registres.

Listing 11

```
;*** Routine copiedisplay  ***

; Utilisation du :  registre RA comme compteur de caractères
;                   registre d'index 2 pour écrire sur l'afficheur
;                   pointeur de table qui pointe vers ce qui est déjà écrit
;                   (sauf les octets HH qui sont mis à 1)

copiadisplay                    ; Routine de copie de la RAM
        clrf    TBLPTRU         ; ceux écrits sont à 1Fxxx.
        bsf     TBLPTRU,0       ;
        lfsr    2,0x080         ; le registre d'index 2 pointe vers le 1er caractère
        movlw   0x50            ; RA=50h (doit écrire 80 caractères)
        movwf   RA              ;
copiadisplay1                   ;
        tblrd*+                 ; lit le caractère
        movf  TABLAT,0          ; le copie dans W puis le déplace
        movwf  POSTINC2         ; dans le buffer de l'afficheur après avoir été incrémenté
        decfsz RA,1             ; répétitions, si 80 caractères sont copiés, alors fin
        goto copiadisplay1      ; sinon on continue
        return                  ; Fin de la routine
```

Listing 12

```
;*** Routine curseurdisplay ***

; Le registre RA contient la position où nous voulons que le curseur se trouve
; Display 4 x 20
;       RA = position  00..13 1ère ligne
;       RA = position  40..53 2ème ligne
;       RA = position 14..27  3ème ligne
;       RA = position  54..67 4ème ligne

cursordisplay                   ; Nom de la routine
        bcf     PORTD,0         ; une commande doit être envoyée
        bsf     RA,7            ; à l'adresse + position
        movff   RA,PORTE        ; sur le bus de l'afficheur
        bcf     PORTD,1         ; bit d'impulsion basse
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; bit d'impulsion haute
        call    msec1           ; la commande a besoin de 40 µs
        bcf     PORTD,0         ; une commande doit être envoyée
        movlw   0x0D            ; 0D = curseur clignotant
        movwf   PORTE           ; sur le bus de l'afficheur
        bcf     PORTD,1         ; bit d'impulsion basse
        nop                     ; actif
        nop                     ; actif
        nop                     ; actif
        bsf     PORTD,1         ; bit d'impulsion haute,
        call    msec1           ; la commande a besoin de 40 µs
        return                  ; fin de la routine.
```

Le prototype de la fonction pourrait ressembler à celle visible dans le listing 9, dans lequel l'écriture des deux facteurs et la lecture du résultat correspondent tout simplement à des transferts de registres passant par l'accumulateur (registre W) du microcontrôleur.

Notez que l'addition, la soustraction, la multiplication, la division par un nombre « puissance de 2 », la division par un nombre entier, la comparaison, la conversion binaire vers BCD et vice versa, sont les opérations mathématiques les plus courantes utilisées avec les PIC, qui sont des microcontrôleurs non prévus pour exécuter des fonctions mathématiques complexes.

Dans le cas où les routines ne conviendraient pas pour votre application, vous devrez obligatoirement opter pour des microcontrôleurs de type DSP.

Un DSP est un microprocesseur optimisé pour exécuter de nombreuses opérations mathématiques complexes comme par exemple une analyse de Fourier et cela le plus rapidement possible.

Routine pour la gestion d'un afficheur alphanumérique

Un afficheur LCD alphanumérique est un composant électronique très utilisé avec les microcontrôleurs PIC, cela est dû à la structure particulière du contrôleur **HD44780** qui se **connecte facilement à l'un des ports du PIC**.

De nombreux compilateurs tels que le PICBASIC disposent d'instructions spécifiques pour gérer un afficheur LCD avec une seule de code.

En règle générale, l'afficheur est géré en « accès direct », c'est-à-dire que les données et les commandes sont envoyées directement à l'afficheur.

Cependant dans notre sous-routine (dont une partie du code source est visible dans le listing 10) le fonctionnement est un peu différent. Le microcontrôleur n'écrit pas directement sur l'afficheur.

En fait la zone d'adresse de « 0x80 » à « 0xCF » de la RAM (soit un total de 80 caractères qui sont utilisés pour gérer l'afficheur de 4 lignes de 20 caractères) est une copie virtuelle de ce que nous voulons afficher sur l'écran.

Après avoir écrit dans la RAM les caractères à afficher, la routine de visualisation est appelée et prend en charge le transfert des caractères en utilisant les différentes commandes et en respectant les temporisations (timing) nécessaires.

Lorsque vous utilisez cette routine, vous devez vérifier que les instructions « pointent » toujours sur le bon PORT, celui connecté à l'afficheur. Nous avons utilisé le « PORTD » pour les commandes et le « PORTE » pour les données. Rien ne vous empêche d'avoir un câblage différent pour votre afficheur, la routine fonctionnera à condition qu'elle soit correctement adaptée.

En plus de la routine d'affichage, nous avons prévu deux autres routines, une pour copier dans la RAM n'importe quel contenu écrit dans le programme (voir la routine « copiedisplay » du listing 11), et

Listing 13

```
;*** Routine vérifiant l'arrivée d'un caractère via l'UART ***
; Utilisation des registres RA et RB
; RA : 0 = pas arrivé  1= arrivé
; RB : caractère

UARTcar                     ; Début de la routine
        clrf    RA          ; RA = 0 par défaut
        btfsc   RCSTA1,1    ; Teste si dépassement (overrun)
        goto    UARTcarerror ; Va à la gestion de l'erreur
        btfss   PIR1, RCIF  ; Teste si un caractère arrive
        return              ; S'il n'arrive, pas fin de la routine
        movff   RCREG1,RB   ; Lit le caractère de l'UART et le copie dans RB
        btfsc   RCSTA1,2    ; Teste si la réception du mot est incomplète (Framing error), structure de l'octet correcte ?
        goto    UARTcarerror ; Framing error : saut
                            ; un caractère correct est arrivé
        bsf     RA,0        ; =1, flag du caractère reçu
        return              ;
UARTcarerror                ; Erreur dans la réception de l'UART
        movlw   0x00        ; pas UART pour le reset
        movwf   RCSTA1      ;
        nop                 ;
        movlw   0x80        ; Reset de RX UART.
        movwf   RCSTA1      ; Bit PORTE pour l'UART, 8 bits,
                            ; rx désactivé, pas d'erreur
        bsf     RCSTA1,CREN ; RX activé
        return              ; Fin  de la routine.
```

Listing 14

```
;*** Routine qui vérifie l'arrivée ***
; d'un caractère via l'UART à l'adresse (0x80) ***
; Utilisation des registres RA et RB
; RA : 0 = pas arrivé  1= arrivé

indUART                     ; Début de la routine
        clrf    RA          ; RA = 0 par défaut
        btfsc   RCSTA1,1    ; Teste si dépassement (overrun)
        goto    indUARTerror ; Va à la gestion de l'erreur
        btfss   PIR1,RCIF   ; Teste si un caractère arrive
        return              ; S'il n'arrive pas, fin de la routine
        movff   RCREG1,RB   ; Lit le caractère de l'UART et le copie dans RB
        btfsc   RCSTA1,2    ; Teste si la réception du mot est incomplète (Framing error)
                            ; structure de l'octet correcte ?
        goto    indUARTerror ; Framing error : saut
                            ; un caractère correct est arrivé
        movlw   0x80        ;
        cpfseq  RB          ; compare avec 80 le caractère reçu
        return              ; Si non, retourne avec le flag = 0
        bsf     RA,0        ; Si  flag STX reçu à 1
        return              ; Fin de la routine.
indUARTerror                ; Erreur dans la réception de l'UART
        movlw   0x00        ; pas UART pour le reset
        movwf   RCSTA1      ;
        nop                 ;
        movlw   0x80        ; Reset de RX UART.
        movwf   RCSTA1      ; Bit du port pour l'UART, 8 bits,
                            ; rx désactivé, pas d'erreur
        bsf     RCSTA1,CREN ; RX activé
        return              ; Fin de la routine.
```

une autre qui permet de faire clignoter le curseur à un emplacement souhaité (voir la routine « curseurdisplay » du listing 12).

Comme pour les routines mathématiques, celles gérant l'afficheur nécessitent une phase préparatoire avant d'effectuer les appels.

Dans la pratique, il est nécessaire de définir les différents registres avec les codes hexadécimaux correspondants aux caractères à afficher ou à la commande que l'afficheur doit exécuter.

Comme chaque routine fonctionne selon ses propres registres, reportez-vous aux commentaires des Listings 10, 11 et 12.

Routine pour la gestion de l'UART

La gestion de l'UART en Assembleur est quelque chose d'assez complexe. En fait, généralement il est possible de travailler simultanément avec les deux interruptions ou par interrogation séquentielle (polling).

Les registres de contrôle et de gestion sont nombreux et le risque de faire des erreurs est très élevé.

Afin de faciliter l'écriture du firmware par le concepteur, nous avons réalisé des routines optimisées et testées, pouvant être appelées à tout moment.

Elles permettent d'effectuer toutes les fonctions normales d'un UART, c'est à dire la réception et la transmission d'un caractère, la réception et la transmission d'une chaîne de caractères et enfin la réception d'une chaîne reconnaissant un caractère prédéfini.

La première de ces routines est visible dans le listing 13, il s'agit de la routine vérifiant l'arrivée d'un caractère via l'UART. Lorsqu'elle est appelée, elle vérifie que l'UART a bien reçu un caractère.

La réponse passe par les registres RA et RB, dont le premier contient l'accusé de réception (RA = 1) et le second, le caractère.

Le principe de fonctionnement est simple, le programme principal appelle la sous-routine avec l'instruction « UARTcar », à la fin de la routine, le contenu du registre RA est vérifié. Si RA=1 alors l'UART a reçu un caractère qui est prélevé dans le registre RB.

La seconde routine est « indUART » (Listing 14), elle permet de vérifier que l'UART a reçu une chaîne de caractères qui contient, comme 1er caractère, une valeur par défaut (dans notre cas, cette valeur est 0x80).

Cette routine est utilisée lors du débogage série grâce à notre programme de monitoring.

L'envoi par le PC et la réception de la part de l'application de ce caractère permet l'exécution de certaines instructions et fonctions de contrôle.

Nous avons évoqué précédemment qu'une chaîne entière peut être transmise, cela est possible grâce à la routine « txUART » dont le code source est visible dans le listing 15.

Le principe de fonctionnement est simple, le programme principal écrit la chaîne de caractères à envoyer à l'UART dans les emplacements de la mémoire RAM identifiés par les alias par le registre « CARTX ».

Si au contraire, vous devez envoyer un seul caractère, vous pouvez utiliser la routine « txcarUART » visible dans le listing 16 (c'est la routine la plus simple pour la gestion de l'UART).

Le fonctionnement est simple, vous écrivez le caractère à envoyer dans le registre RA et la routine est appelée.

La procédure se complique considérablement lorsqu'au lieu de recevoir une chaîne de caractères, ceux-ci arrivent à des moments différents dans des chaînes de longueurs différentes. Cela génère parfois des erreurs.

La routine « rxUART » (voir le listing 17) s'occupe de recevoir une chaîne de caractères et de la placer, caractère par caractère, dans le tampon de réception situé dans la RAM à l'adresse « 0x100 (RXBUF00) » jusqu'à l'adresse « 0x1FF (RXBUFFF) ».

Comme pour la transmission d'une chaîne de caractères, la longueur de la chaîne est limitée à 256 caractères pour la réception.

En sortie la routine « rxUART » place dans la mémoire tampon de la RAM, à partir de l'adresse « RXBUF00 » jusqu'à « RXBUFFF », la chaîne reçue et, grâce au registre « CARRX », communique le nombre de caractères reçus.

Les 5 routines sont immédiatement utilisables, sans modification, avec la famille PIC18F.

Notez que le PIC18F8723 comporte deux UART matérielles (hardware), vous devez modifier le « 1 » en « 2 » si vous utilisez le 2ème UART (rxUART2, txUART2 ...).

Routine de gestion de l'horloge

Une routine que nous trouvons particulièrement efficace de l'énergie consommée.

Listing 15

```
;*** routine de transmission d'une chaîne via l'UART ***
; la chaîne à transmettre est dans TXBUF00 à TXBUFFF
; utilise le registre RA, CARTX
; RA = utilisé (caractère en transit)
; CARTX = nombre de caractères à envoyer
; utilise également le registre d'index 2
; à la fin de la chaîne ajoute CR

txUART                          ; Début de la routine
        lfsr    2, TXBUF00      ; le registre d'index 2 pointe
                                ; au 1er emplacement du buffer.
txuart1                         ;
        movff   POSTINC2,RA     ; déplace le caractère à transférer dans RA
        call    txcarUART       ; envoi
        decfsz  CARTX,1         ; décrémente le nombre de caractères à envoyer
        goto    txuart1         ; répète le cycle de transmission
        movlw   0x0D            ; transmet le caractère de fin de chaîne <CR>.
        movwf   RA              ;
        call    txcarUART       ; envoi
        movlw   0x03            ; transmet le caractère de fin de chaîne <ETX>
        movwf   RA              ; le cas échéant
        call    txcarUART       ; envoi
        return                  ; fin de la routine.
```

Listing 16

```
;*** routine de transmission d'un caractère via l'UART ***
; le caractère à transmettre est dans RA

txcarUART                       ; Début de la routine
        btfss   TXSTA1,TRMT     ; teste si TX est libre ; 0=occupé 1=libre.
        goto    txcarUART       ; attends que l'UART soit libre
        movff   RA,TXREG1       ; transmet le caractère
        return                  ; fin de la routine
```

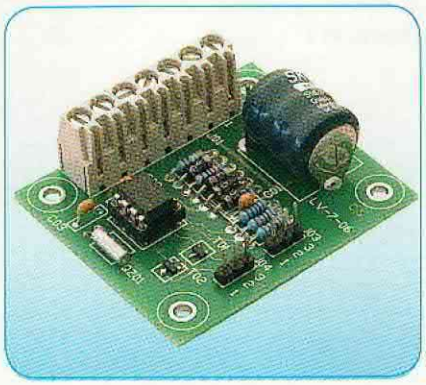
Listing 17

```
;*** Routine de réception d'une chaîne via l'UART avec un délai d'attente ***
; le buffer de réception est dans le banc 1 : 0x100 (RXBUF00) .. 0x1FF
; la chaîne finit avec un <CR>
; utilise le registre RA et le registre d'index 2
; utilise les registres WAITR1, WAITR2 et WAITR3 pour le délai d'attente (timeout)
; le nombre de caractères est dans CARRX

rxUART                          ; Début de la routine
        lfsr    2,RXBUF00       ; le registre pointe au 1er caractère à recevoir.
        clrf    CARRX           ; R3 est le registre « RX OK » de la chaîne entière.
rxuart00                        ;
        movlw   0x10            ; règle le timeout à environ 10h = 5 secondes
        movwf   WAITR1          ; 64h = 30 secondes
        movlw   0xFF            ;
        movwf   WAITR2          ;
        movwf   WAITR3          ;
rxuart0                         ; RA = caractère rx.
        btfsc   RCSTA1,1        ; teste si dépassement (overrun)
        goto    rxcarerror      ; overrunn, alors gestion de l'erreur
        btfss   PIR1,RCIF       ; teste si un caractère arrive
        goto    timeoutgest     ; s'il n'arrive pas, reste en attente
        movff   RCREG1,RA       ; lit le caractère de l'UART et le place dans RA
```

```
        btfsc   RCSTA1,2        ; teste si framing error, structure de l'octet correcte ?
        goto    rxcarerror      ; framing error : saut
    goto        rxuartok        ; OK, saut à la fin de la routine
timeoutgest                     ; routine de gestion du timeout
        decfsz  WAITR1,1        ;
        goto    rxuart0         ;
        movlw   0x64            ; règle le timeout à 30 secondes
        movwf   WAITR1          ;
        decfsz  WAITR2,1        ;
        goto    rxuart0         ;
        movlw   0xFF            ;
        movwf   WAITR2          ;
        decfsz  WAITR3,1        ;
        goto rxuart0            ;
rxcarerror                      ; s'il y a eu une erreur
        movlw   0x00            ; pas UART pour le reset
        movwf   RCSTA1          ;
        nop                     ;
        movlw   0x80            ; RX
        movwf   RCSTA1          ; bits du PORT de l'UART, 8 bits, rx désactivé,
                                ; pas d'erreur
        bsf     RCSTA1,CREN     ; RX activé
        clrf    CARRX           ; CARRX= 0 si erreur de RX, ou pour la sortie du timeout
        return                  ; fin de la routine
rxuartok                        ; un caractère arrive
        movff   RA,POSTINC2     ; dans le buffer de réception
                                ; incrémente le pointeur du buffer
        infsnz  CARRX           ; incrémente le compteur des caractères reçus
        goto    rxcarerror      ; Si > 256  trop de caractères = erreur
        movlw   0x0D            ;
        cpfseq  RA              ; compare avec 0D <CR>
        goto    rxuart00        ; retourne à la réception
        return                  ; fin de la routine
```

Tableau 2 - Routine pour la gestion de l'horloge

Appel de la routine	Fonction	Registres/variables utilisées (non compris celles de service)
inizCLK	Initialisation du module d'horloge.	CLKWB
lecturetime	Lit la date et l'heure dans le module d'horloge, et les transfert dans la RAM du microcontrôleur.	CLOCK0 [an] CLOCK1 [mois] CLOCK2 [jour] CLOCK3 [heure] CLOCK4 [minute] CLOCK5 [seconde]
ecrituretime	Transfert de la RAM du microcontrôleur au module d'horloge la date et l'heure.	CLOCK0 [an] CLOCK1 [mois] CLOCK2 [jour] CLOCK3 [heure] CLOCK4 [minute] CLOCK5 [seconde]
lecturealarme	Lit à partir du module d'horloge les informations d'alarme et les transfert dans la RAM du microcontrôleur.	CLKSO [heure] CLKSM [minute]
ecrituredalarme	Transfert de la RAM du microcontrôleur au module d'horloge les informations d'alarme.	CLKSO [heure] CLKSM [minute]
alarmeON	Active l'alarme.	
alarmeOFF	Désactive l'alarme.	



Toutes les données relatives à l'horloge sont prises en charge par les registres CLKWB, CLKRB, CLOCK0 à CLOCK5, CLKSM et CLKSO.

Compte tenu de leur complexité (ils utilisent des structures particulières et de nombreuses routines spécifiques que nous n'avons pas étudiées ici), nous avons préféré faire un résumé dans un tableau comme pour les opérations mathématiques. Le Tableau 2 résume la façon dont ces routines doivent être utilisées, en se référant à la liste des détails.

Routine pour le débogage série

Comme vous l'avez certainement vérifié, le fichier « DEMO-01.asm » contient un grand nombre de routines de service que nous n'avons pas reporté dans ces pages.

Ces routines, si elles sont utilisées de manière appropriée, vous permettent d'effectuer le débogage en temps réel d'un programme qui est en cours d'exécution. En fait elles effectuent des requêtes spécifiques à partir de l'ordinateur, comme par exemple :

- « voir » le contenu de la RAM (appel d'un registre qui correspond aux emplacements de la mémoire vive) ;
- « voir » le contenu de l'EEPROM ;
- « écrire » directement dans la RAM à tous les emplacements ;
- « écrire » dans l'EEPROM des données.

Toutes les routines peuvent être compilées et intégrées dans le firmware de l'application ou, plus simplement, vous pouvez les « copier/coller » en les prenant directement du programme de démonstration, et en les plaçant dans votre programme.

Conclusion

Dans cette deuxième partie, nous avons présenté les routines les plus couramment utilisées avec les microcontrôleurs PIC de chez Microchip. Dans la prochaine et dernière partie de cette série d'articles, nous allons explorer quelques-unes des routines de débogage via le port série, ainsi que le programme que nous avons développé en Visual Basic.

ÉMETTEURS & RÉCEPTEURS 1,2 & 2,4 GHz

EMETTEURS 1,2 & 2,4 GHz
20 mW et 1000 mW

Alimentation :13,6 VDC. 4 fréquences en **2.4 GHz :2,4 - 2,427 - 2,454 - 2,481 GHz** ou 8 fréquences en **1.2 GHz 20 mW: 1,112 - 1,139 - 1,193 - 1,220 - 1,247 - 1,264 - 1,300 GHz** ou 4 fréquences en **1.2 GHz 1 W: 1,120 - 1,150 - 1,180 - 1,255 GHz**. Sélection des fréquences : dip-switch. Stéréo : audio 1 et 2 (6,5 et 6,0 MHz). Livré sans alimentation ni antenne.

TX2.4G..... Emetteur 2,4 GHz 4 c monté 20 mW ..... 39,20 €
TX1.2G..... Emetteur 1,2 GHz 20 mW monté 8 canaux ..... 48,20 €
TX1.2G-1-... Emetteur 1,2 GHz monté 1 W 4 canaux ..... 79,30 €

RÉCEPTEURS 4 CANAUX
1,2 & 2,4 GHz

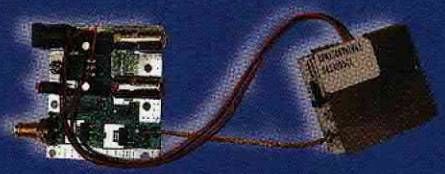
Alimentation : 13,6VDC. 4 fréquences en **2.4 GHz :2,4 - 2,427 - 2,454 - 2,481 GHz** ou 8 fréquences en **1.2 GHz : *1,112 - 1,139 - 1,193 - 1,220 - 1,247 - 1,264 - 1,300 GHz***. Sélection des fréquences : dip-switch pour le 1,2 GHz et par poussoir pour les versions 2,4 GHz. Stéréo : audio 1 et 2 (6,5 et 6,0 MHz). Fonction scanner pour la version 1.2 GHz. Livré sans alimentation ni antenne.

RX2.4G.....Récepteur monté 2.4 GHz 4 canaux..... 39,00 €
RX1.2G.....Récepteur monté 1.2 GHz 8 canaux ..... 48,20 €

VERSION 256 CANAUX

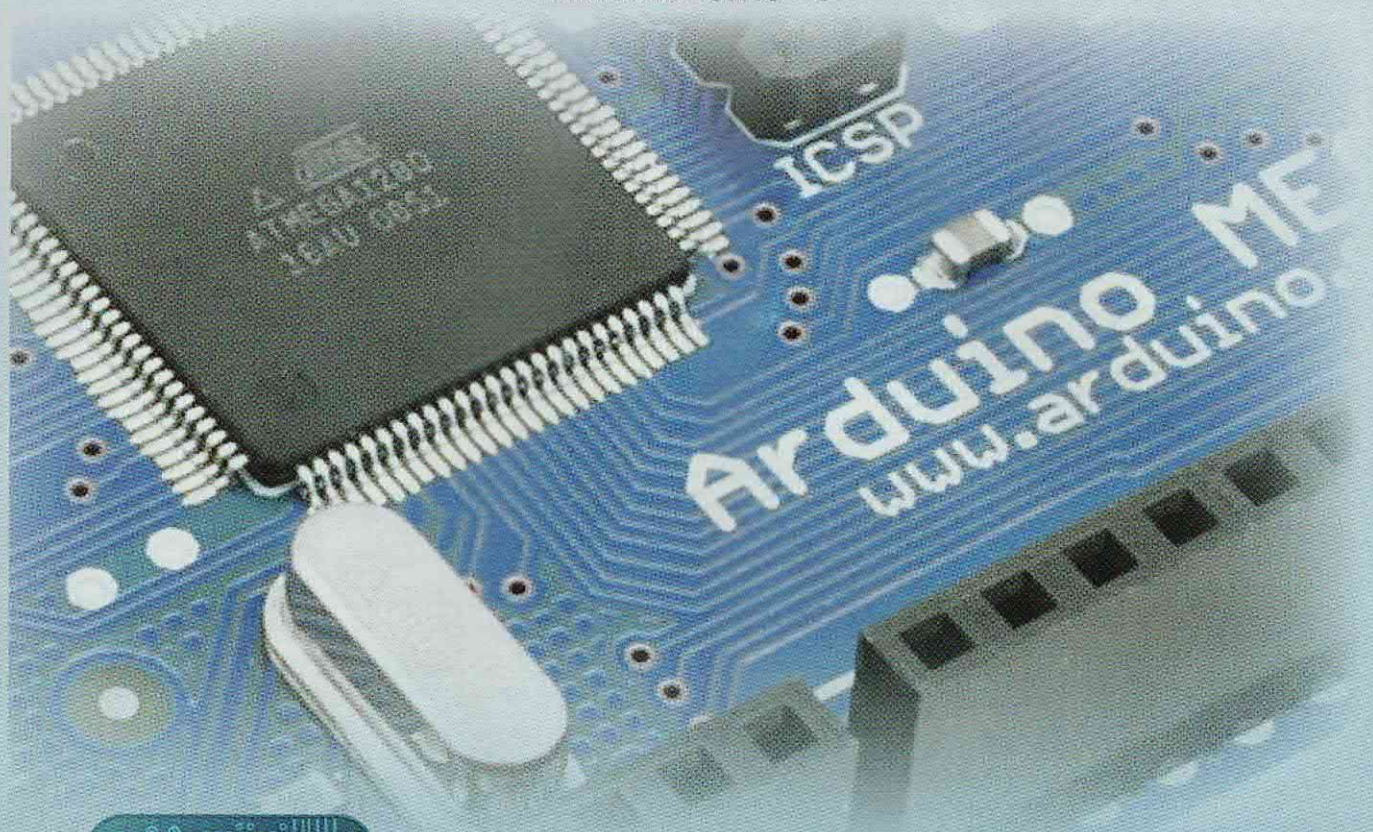
Extensions pour émetteurs **TX2.4G / TX1.2G** et récepteurs **RX2.4G / RX1.2G** : permettent d'atteindre **256** canaux.
Le pas est de 1 MHz. Sélection des canaux par dip-switch.
Fréquences de départ : **2,3** pour les versions **TX2,4G / REX2.4G** et **1,2** pour les **TX 1,2G / RX 1,2G**. Extension vendue sans émetteur ni récepteur.

TEX1.2..... Kit extension 1,2 à 1,456 GHz .....19,90 €
TEX2.3..... Kit extension 2,3 à 2,556 GHz.....19,90 €
REX1.2..... Kit extension 1,2 à 1,456 GHz .....19,90 €
REX2.3..... Kit extension 2,3 à 2,556 GHz .....19,90 €



COMELEC CD 908 - 13720 BELCODENE Tél.: 04 42 70 63 90 Fax: 04 42 70 63 95 www.comelec.fr
Photos non contractuelles. Publicité valable pour les mois de parution. Prix exprimés en euros TTC. Sauf erreurs typographiques ou omissions

Cours ARDUINO - 7



Cours ARDUINO

septième partie

de Mirco SEGATELLO

Nous allons apprendre, dans cette septième leçon du Cours Arduino, à interfacer une carte mémoire de type « SD Card ». Pour cela nous allons utiliser une carte électronique spécialement conçue pour être interfacée avec Arduino.

De nos jours les cartes mémoire sont largement utilisées dans les appareils photo, les smartphones, les GPS pour stocker des données. Le type de carte mémoire « SD Card » est devenu le plus utilisé en peu de temps, même dans les équipements électroniques pour hobbyistes.

La raison est très simple. Ce type de carte mémoire est peu couteux, dispose d'une grande capacité de stockage, est facile à gérer et conserve les données sans alimentation (une sorte de EEPROM).

En résumé, elles sont devenues le complément indispensable de la mémoire de base (souvent petite) des microcontrôleurs, ou de tout appareil où les applications exigent un espace de stockage important sans dépenser beaucoup d'argent.

Par exemple, vous pouvez trouver une carte mémoire SD de 8 Go pour une dizaine d'euros. En considérant qu'un bon microcontrôleur dispose d'environ 512 Ko de mémoire non volatile, il semble évident qu'une carte mémoire permet d'améliorer considérablement le stockage de données.

La principale application d'une carte SD concerne essentiellement les enregistreurs de données (data-logger), ou des dispositifs qui, à intervalles réguliers, stockent sur un support non volatile une série de données. Par exemple, les différentes positions relevées par les récepteurs GPS.

Une carte SD peut être facilement gérée en lecture et en écriture à partir d'un PC à l'aide d'un lecteur de carte peu coûteux, les ordinateurs portables récents disposent de lecteurs de cartes SD intégrés.

Dans ce cours nous allons vous apprendre à utiliser une carte SD avec Arduino, mais avant d'entrer dans les détails faisons un bref rappel sur ce type de mémoire.

Une carte SD (« SD » voulant dire « Secure Digital ») est une carte mémoire amovible de stockage de données numériques créée en janvier 2000 par une alliance formée entre Panasonic, SanDisk et Toshiba.

Les cartes SD sont utilisées pour le stockage de fichiers dans les appareils photo numériques, les systèmes de navigation GPS, les consoles de jeux vidéo, les smartphones, ou encore les systèmes embarqués.

Depuis 2010, les cartes SD sont devenues le standard de stockage, à la suite de l'abandon progressif des autres formats, notamment du Memory Stick (Sony) ou encore la « **MultiMediaCard** » (MMC) qui a été l'une des premières cartes mémoires standard. De nos jours, on parle de carte « **SDHC** » pour des capacités comprises entre 4 et 32 Go, et de carte « **SDXC** » jusqu'à 2 To.

Le format physique d'une carte SD (24 m × 32 mm × 2,1mm) est proche de celui d'une carte MMC. Une carte SD dispose d'un microswitch sur le côté permettant de **protéger les données en écriture** (un peu comme les languettes des cassettes audio qui interdisaient l'enregistrement lorsqu'elles étaient retirées).

Les cartes SD utilisent de la **mémoire flash NAND**, c'est une technologie de stockage à base de semi-conducteurs **qui ne requiert pas d'alimentation électrique pour conserver les données**. Elle est donc dite « **non volatile** », car contrairement à une mémoire vive (RAM) les données ne s'effacent pas lorsque l'alimentation électrique de la mémoire est coupée.

La mémoire Flash NAND a été conçue par Toshiba à la fin des années 80 et a été utilisée dans des clés USB, les cartes mémoire et dans des appareils électroniques tels que téléphones et appareils photos avant de faire son apparition dans les disques durs de stockage nommés « SSD (Solid State Storage) ».

La mémoire Flash NAND est en fait une mémoire de type EEPROM dans laquelle chaque cellule stocke un bit (mémoire de type Single Level Cell utilisée pour des cartes hautes performances) ou 2 bits (mémoire Multi Level Cell) voire 3 bits (mémoire Triple Layer Cell) afin de privilégier l'espace de stockage au détriment des performances.

L'indice de débit en écriture des cartes SD standard était à l'origine exprimé en multiples d'un débit minimal, « x1 » équivalant à 150 ko/s (idem que pour un CD).

Les débits en lecture sont en général sensiblement plus élevés que ceux en écriture.

En 2006, il a été décidé de ne plus spécifier la vitesse d'écriture maximale supportée par les cartes SD en multiples, mais d'utiliser le terme de « **classe** » pour indiquer la vitesse d'écriture minimale garantie par ces mêmes cartes, le numéro de classe figurant encerclé par un « **C** ».

Classe	**2**	**4**	**6**	**10**
Débit en écriture (Mo/s)	2	4	6	10
Taux	x13	x26	x40	x66

Le **Tableau 1** résume les différentes caractéristiques des **3 principaux formats** de cartes utilisées : **SD standard**, **mini-SD** et **micro-SD**. Les 2 formats mini-SD et micro-SD peuvent être utilisés sur un lecteur de carte SD standard en utilisant un adaptateur.

Comme pour une carte « MultiMediaCard », une carte SD utilise des contacts de surface au lieu de connecteurs enfichables mâles/femelles. Cela procure une robustesse, mais ce type de connexion est moins appropriée dans des environnements difficiles, notamment en présence de vibrations.

Une carte SD d'une capacité supérieure à 2 Go a une vitesse minimale de lecture/écriture de 2,2 Mo/s. Ce type de carte est appelé « **SDHC** » (Secure Digital High Capacity) ou « SD 2.0 », et n'est pas compatible avec les anciens lecteurs de carte SD.

Une carte « SDHC » (qui a été créée pour accélérer la lecture de contenus multimédias) est classée en plusieurs classes (voir le Tableau 1) en fonction de son taux de transfert ou « SD Speed Class Ratings » qui a été défini par la « SD Association ».

Les cartes SD ayant des capacités supérieures à 32 Go sont appelées « **SDXC** » (Secure Digital eXtended Capacity) ou « SD 3.0 ».

Tableau 1 : caractéristiques des différentes cartes mémoire.

Type	Dimensions	Système de fichiers	Vitesse	Capacité maximale
MMC	30 x 23 x 1,4 mm	FAT16	2 MB/s	512 Mo
RS-MMC	18 x 24 x 1,4 mm			
MMC mobile	14 x 11 x 1.1mm			
SD	32 x 24 x 2,1 mm	FAT16 FAT32	2 MB/s	2 Go
Mini-SD	21,5 x 20 x 1,4 mm			
Micro-SD (Trans Flash Card)	11 x 15 x 1 mm			
SDHC	32 x 24 x 2,1 mm	FAT32	Classe 2 - 2 MB/s Classe 4 - 4 MB/s Classe 6 - 6 MB/s Classe 10 - 10 MB/s	32 Go
Mini-SDHC	21.5 x 20 x 1.4 mm			
Micro-SDHC	11 x 15 x 1 mm			
SDXC	32 x 24 x 2,1 mm	FAT64	100 MB/s	2 To

Figure 1 : les différents formats d'une carte SD.

Figure 2 : Exemple d'une carte Mini-SD avec son adaptateur.

La capacité maximale théorique de la norme « SDXC » est de 2048 Go (2 To) avec une vitesse de 104 Mo/s.

Le système de fichier utilisé pour ce type de carte est appelé « ExFAT » (FAT64).

Les différents lecteurs de cartes SD ne sont pas compatibles avec les versions ultérieures, mais seulement avec les versions précédentes.

Par exemple un lecteur de carte « SDXC » sera compatible avec des cartes « SDXC », « SDHC » et « SD », tandis qu'un lecteur de carte SD sera seulement compatible avec des cartes SD.

Tableau 2 : brochage d'une SD-Card (dans la colonne Type, S=alimentation; I=entrée; O=sortie avec push-pull).

Mode SD				Mode SPI		
Broche	Nom	Type	Description	Nom	Type	Description
1	CD/DAT3	I,O	Détection de la carte / Ligne de données 3 (Bit 3)	CS	I	Sélection de la carte (actif bas)
2	CMD	I,O	Commande / Réponse	DataIn	I	Hôte de la carte commandes et données
3	VSS1	S	Masse (GND)	VSS1	S	Masse (GND)
4	VDD	S	Positif alimentation	VDD	S	Positif alimentation
5	CLK	I	Horloge	CLK	I	Horloge
6	VSS2	S	Masse (GND)	VSS2	S	Masse (GND)
7	DAT0	I,O	Ligne de données 0 (Bit 0)	DataOut	O	Hôte de la carte commandes et status
8	DAT1	I,O	Ligne de données 1 (Bit 1)	RSV	-	Réservé
9	DAT2	I,O	Ligne de données 2 (Bit 2)	RSV	-	Réservé

Tableau 3 : brochage d'une MM Card (Type : S=alimentation; I=entrée; O=sortie).

Mode Multi Media Card				Mode SPI		
Broche	Name	Type	Description	Name	Type	Description
1	RSV	NC	Non connectée	CS	I	Sélection de la carte (actif bas)
2	CMD	I,O,PP,OD	Commande / Réponse	DataIn	I	Hôte de la carte commandes et données
3	VSS1	S	Masse (GND)	VSS1	S	Masse (GND)
4	VDD	S	Positif alimentation	VDD	S	Positif alimentation
5	CLK	I	Horloge	CLK	I	Horloge
6	VSS2	S	Masse (GND)	VSS2	S	Masse (GND)
7	DAT0	I,O,PP	Donnée 0	DataOut	O	Hôte de la carte commandes et status

Les périphériques qui ne sont pas compatibles avec la norme « SDHC » sont incapables de reconnaître (et donc lire et écrire) des cartes « SDHC ».

D'autre part, une carte de 4 Go qui n'est pas certifiée au standard « SDHC » ne supportera pas la norme « SD2.0/SDHC ».

L'analyse des caractéristiques électriques d'une carte SD nous montre qu'elle dispose de **9 broches**, et des modes de **communication compatibles avec une interface de 4 bits en parallèle** (mode SD) et une **interface de type série « SPI »** (Serial Peripheral Interface) à 2 lignes de données et une ligne d'horloge (mode SPI). La tension d'alimentation typique est de **3,3 V** (les valeurs autorisées vont de 2,7 V à 3,6 V).

Les niveaux des signaux de communication devront être du même ordre.

Si le microcontrôleur qui est connecté à une carte SD fonctionne à 3,3 V, les signaux peuvent être appliqués directement. Si les signaux ont une tension de 5 V, il est nécessaire de prévoir une adaptation des niveaux des tensions en observant les consignes décrites ci-après.

Les signaux de sortie de la carte SD (3,3 V) peuvent être appliqués directement sur les entrées d'un microcontrôleur alimenté en 5 V, car il reconnaîtra correctement les niveaux logiques.

Une tension de 3,3 V sur l'entrée du microcontrôleur sera interprétée comme un niveau logique « 1 » (haut). Par contre les signaux de sortie du microcontrôleur (5 V) allant vers la carte SD devront être atténués à 3,3 V.

Tableau 4 : connexions entre une carte SD et Arduino.

Broches SD	Fonction	Broches Arduino	Fonction
1	CS	10	SS
2	DI	11	MOSI (Serial Data OUT)
3	VSS1	GND	GND
4	VDD	+3,3 V	+3,3 V
5	SCLK	13	Clock
6	VSS2	GND	GND
7	DO	12	MISO (Serial Data IN)
8	RSV	Non utilisée	
9	RSV	Non utilisée	

Pour ce faire, il existe deux solutions. Soit utiliser un diviseur de tension afin de réduire le 5 V à 3,3 V ou utiliser des **convertisseurs de niveaux de 5 V vers 3,3 V**, comme par exemple les circuits intégrés suivants : 74HC4050N, 74AHC125N, 74LCX245.

La différence essentielle réside dans le fait qu'un diviseur résistif, tout en étant simple et économique, crée des capacités parasites qui peuvent perturber les signaux à haute fréquence.

En utilisant un circuit translateur de niveau (plus cher), celui-ci permettra d'atteindre une plus grande vitesse de communication.

L'interfaçage entre une carte SD et un microcontrôleur s'effectue via le bus série en mode SPI, par l'intermédiaire de **2 lignes de données**, une **ligne d'horloge** et une ligne pour le « **Chip Select** ».

Arduino et la carte SD

Eh bien, après ce mini tutoriel sur les cartes mémoire SD, nous entrons maintenant dans le vif du sujet, à savoir comment utiliser une carte SD avec une carte Arduino. Afin de faciliter l'utilisation d'une carte SD avec Arduino, il existe dans le commerce différentes cartes adaptatrices pour carte SD ou Micro-SD.

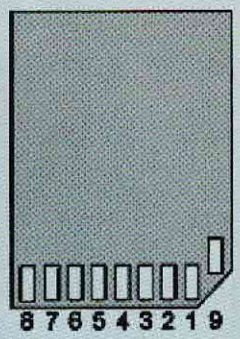


Figure 3 : brochage d'une carte SD.

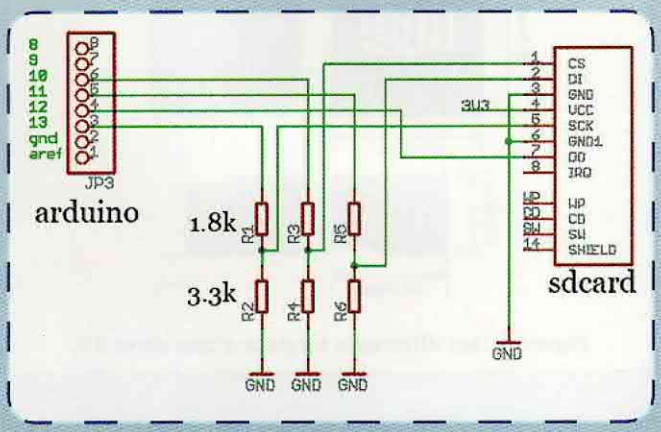


Figure 4 : câblage d'une carte SD avec Arduino.

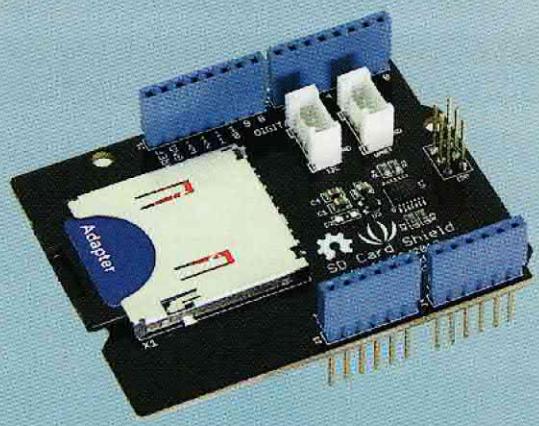


Figure 5 : un adaptateur V4.0 pour carte SD.

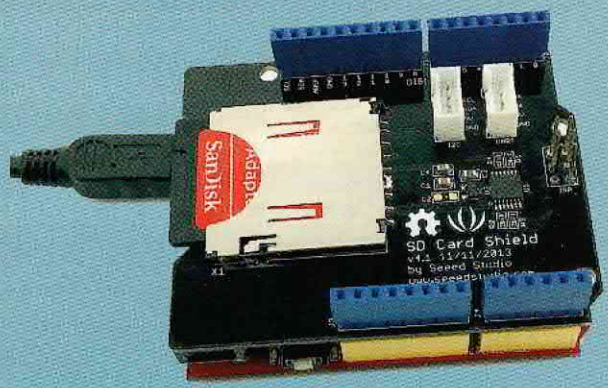


Figure 6 : l'adaptateur pour carte SD insérer sur une carte Arduino.

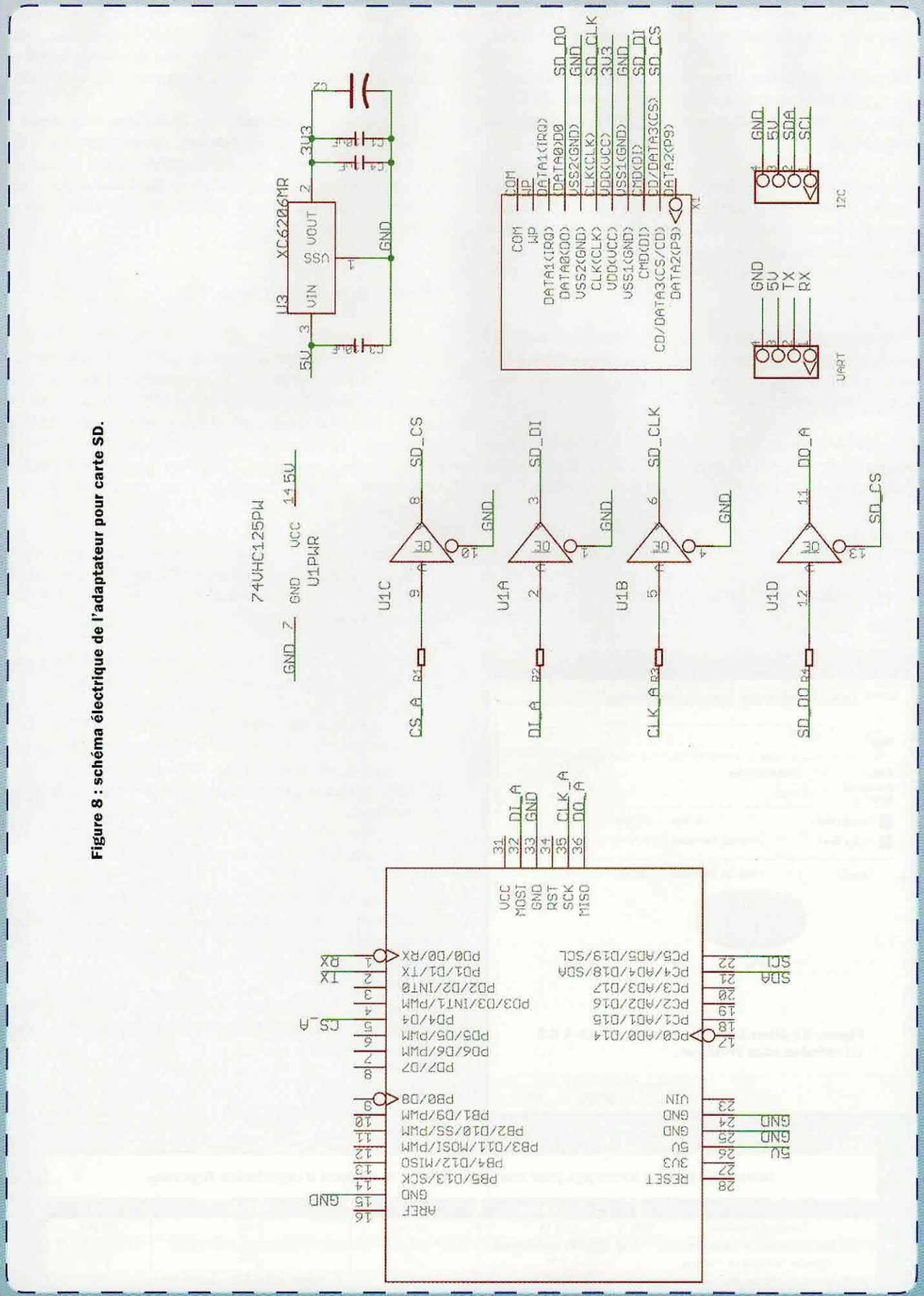


Figure 8 : schéma électrique de l'adaptateur pour carte SD.

Ces adaptateurs sont proposés soit avec des translateurs de niveaux de types résistifs soit de types à logique intégrée.

Pour notre application, nous avons utilisé un adaptateur pour carte SD déjà prêt à l'utilisation que vous trouverez facilement dans le commerce, il s'agit de l'adaptateur « **SD Card shield V4.0** ».

Cet adaptateur est très simple à installer ,il est d'un prix abordable. Il permet le câblage rapide d'une carte SD avec une carte Arduino. Il autorise la lecture/écriture une carte SD via la librairie intégrée à Arduino. Il prend en charge les cartes de type SD, SDHC et Micro-SD. Il occupe le port SPI d'Arduino.

En comparaison avec les versions précédentes, il combine un connecteur SD standard et un connecteur Micro-SD. Il est possible d'empiler d'autres cartes fonctionnant avec des broches inutilisées.

En **fonction de la librairie utilisée**, les cartes de **différents types seront prises en charge** : MMC, SD ou SDHC. A titre d'exemple, nous avons utilisé une carte SD de 4 GB.

Si vous branchez une telle carte directement sur un PC via un lecteur de carte externe (ou intégré dans la plupart des portables), vous verrez apparaître les propriétés du périphérique (représenté sous Windows comme un disque amovible).

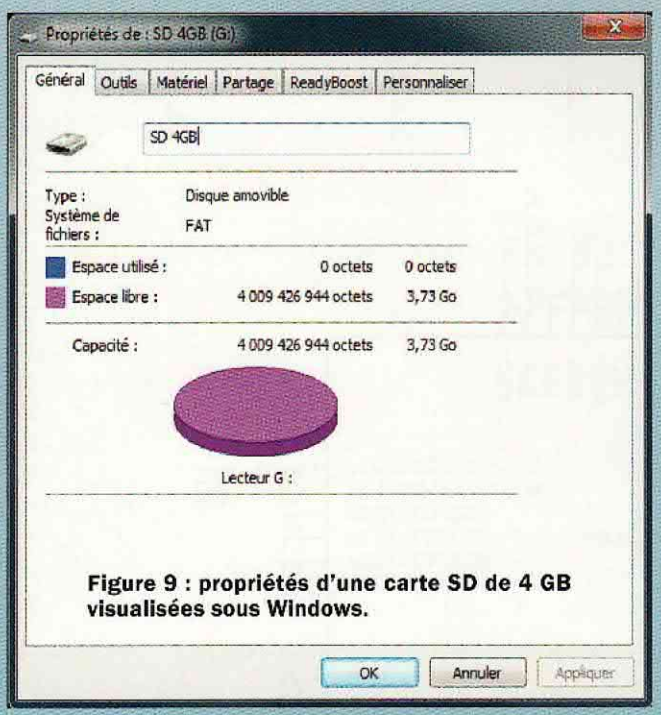


Figure 9 : propriétés d'une carte SD de 4 GB visualisées sous Windows.

Vous devez voir sur votre écran une fenêtre correspondant à celle de la figure 9. En plus de l'espace disponible, vous pouvez voir le format de fichiers utilisés, dans notre exemple il s'agit du format « **FAT** » qui correspondant à une **FAT16**.

Pour information « **FAT** » veut dire « **File Allocation Table** » ou **table d'allocation des fichiers.** Vous devez connaître le type de « FAT » avec laquelle votre carte est formatée, cela est très important car **certaines librairies Arduino ne supportent pas** le standard « **FAT32** » et donc il faut vérifier avant toute utilisation.

Formatage de la carte SD

La « File Allocation Table » (FAT) ou table d'allocation des fichiers est un système de fichiers pour différents systèmes d'exploitation tels que le système MS-DOS et Microsoft Windows jusqu'à la version « Windows ME ». Windows NT et les versions ultérieures ont introduit le nouveau système de fichiers « **NTFS** » pour « New Technology File System ». Ils ont maintenu la compatibilité avec le système de fichier « FAT », ainsi qu'avec d'autres systèmes d'exploitation (Unix, Linux, Mac OS).

Le système « FAT » est relativement simple et est pris en charge par de nombreux systèmes d'exploitation. Ces caractéristiques le rendent approprié par exemple pour des clés USB et des cartes mémoire.

Le système de fichiers FAT est composé de plusieurs sections dont les plus importantes sont les suivantes :

1. le « **secteur de boot** » contenant le « **BPB** » (Bios Parameter Block). C'est le premier secteur de la partition, il contient des informations essentielles ;
2. les « **tables d'allocation** » ou « **FAT** ». C'est une carte du disque ;
3. le « **répertoire racine** » ou « **Root directory** ». C'est une liste des fichiers présents à la racine du disque.

La totalité de la mémoire est divisée en zones appelées « **clusters** ». Il existe plusieurs versions (FAT12, FAT16, FAT32, exFAT) de ce système de fichiers, basé sur le nombre de bits alloués à chaque cluster du disque.

Un **cluster** est un **groupe de secteurs.** Il sert d'unité d'allocation aux fichiers. **Chaque cluster stocke donc les données d'un fichier.**

Pour un fichier de 9 000 octets par exemple, sur un disque utilisant des clusters de 8 192 octets (soit 16 secteurs de

Tableau 5 : type de formatage pour une carte SD pour le système d'exploitation Windows.

	ExFat (FAT64)	FAT32	FAT16	FAT12
Capacité maximale	512 To	32 Go	2 Go	16 Mo
Nombre maximum de fichiers	2 796 202 (par dossier)	4 194 304	65 536	
Nombre maximal de clusters	2^{64}	2^{32}	2^{16}	2^{12}
Longueur maximale des noms des fichiers	255	255	Standard - 8.3 étendu jusqu'à 255	254

512 octets), 2 clusters sont utilisés dont le dernier n'utilise que 808 octets (9 000 - 8 192).

Une **table d'allocation est une carte où chaque nombre représente un cluster**. Cette table est un tableau de nombres, indexé par un numéro de cluster.

Quand un fichier est enregistré dans la mémoire, il va occuper un certain nombre (entier) de clusters. Des clusters de grande taille permettent l'accès aux fichiers avec une plus grande vitesse, mais par contre imposent une faible efficacité dans l'utilisation des bits de mémoire (fragmentation).

Par exemple, un simple fichier texte dans lequel est écrit un seul caractère occupe un espace d'un octet, mais l'espace disque occupé sera égal à la taille d'un cluster.

Le système de fichiers FAT est structuré en 4 sections :

1. la première section, qui est une zone réservée, correspond au **secteur de démarrage** ou secteur de boot. Elle contient le code du boot-loader qui permet le démarrage du système d'exploitation ;
2. la seconde section correspond à la **table d'allocation des fichiers** (FAT) et contient le mappage ou cartographie de toutes les données du disque. Pour des raisons de sécurité, la FAT existe en 2 exemplaires ;
3. le « **répertoire racine** » ou « **Root directory** » stocke les dossiers et les fichiers présents dans le répertoire racine (appelé « root »). Il existe seulement pour les FAT12 et FAT16. Dans la **FAT32 le stockage a lieu directement dans la zone des données** ;
4. la **zone des données** correspond à l'emplacement où l'ensemble des fichiers est stocké, elle occupe la majeure partie de la partition.

Tableau 6 - Structure d'un système de fichiers (File System).

Zone réservée			FAT		Root directory (FAT12/16 uniquement)	Zone des données
Secteur d'amorçage	Informations FS (FAT32 uniquement)	Réservé (facultatif)	FAT N°1	FAT N°2		

La **FAT répertorie les zones du disque disponibles et celles déjà utilisées par les fichiers et répertoires**. La différence entre les différentes tables d'allocations FAT12, FAT16, FAT32, ExFAT réside précisément dans le nombre de bits attribués pour chaque cluster du disque. Nous avons donc :

- **FAT12** : maximum de 2^{12}, soit **4 096 clusters** de taille fixe choisie au départ entre 512 octets et 4 Ko. Ce système a été principalement utilisé pour les disquettes ;
- **FAT16** : maximum de 2^{16}, soit **65 536 clusters** de taille fixe choisie au départ entre 2 Ko et 64 Ko ;
- **FAT32** : apparu avec Windows 95, elle supporte un maximum de $\mathbf{2^{28}}$ (268 millions) de **clusters** de taille variable comprise entre 4 Ko et 32 Ko. **La taille des fichiers ne peut dépasser 4 Go**, c'est-à-dire que vous ne pouvez pas copier ou déplacer un fichier ayant une taille supérieure. Il est nécessaire alors de **passer en système de fichiers NTFS**.

La première version de la FAT était la « FAT12 », un système de fichiers pour disquettes, dont les adresses des clusters étaient sur 12 bits. Elle était capable de gérer des disques durs jusqu'à 16 Mo.

En 1987 est apparu le format FAT16 avec des clusters jusqu'à 64 ko et un adressage sur 16 bits. Il était alors possible théoriquement de créer une partition de 2 Go sur un disque dur, mais à l'époque il n'existait pas des disques d'une telle taille.

Pour surmonter les limitations imposées par les caractéristiques d'une FAT16, Microsoft a créé une nouvelle FAT appelée « FAT32 », caractérisée par un adressage des clusters sur 32 bits, bien qu'en réalité l'adressage s'effectue sur 28 bits.

Avant de commencer à utiliser une carte SD avec Arduino, il est nécessaire de se procurer les librairies qui permettent la mise en œuvre du protocole de communication. La page web de référence Arduino est la suivante :

www.arduino.cc/playground/Learning/SDMMC.

Il existe différentes versions de librairies pouvant gérer la FAT16 ou la FAT32. Certaines permettent la gestion du bus SPI à grande vitesse, il est alors nécessairement d'utiliser l'adaptateur pour carte SD car les translateurs de niveaux doivent être des circuits rapides.

Chaque librairie utilise également ses propres lignes de commande pour la carte SD. La librairie que nous allons décrire est appelée « **SDuFAT** ». Elle a été développée par David Cuartielles et est disponible sur notre site www.electroniquemagazine.com dans le sommaire détaillé de la revue 137 à l'onglet « Télécharger ». Le fichier se nomme « **SDuFAT.zip** ».

La librairie « SDuFAT » est compatible avec une carte SD standard formatée en FAT16 avec un protocole de communication SPI. Elle permet des opérations de lecture et d'écriture sur des fichiers existants.

Pour l'utiliser, décompressez le fichier et copiez l'intégralité du dossier « SDuFAT » dans le dossier « libraries » de l'IDE Arduino. En général vous trouverez le dossier « libraries » de l'IDE sur votre ordinateur selon le chemin suivant (cela peut varier selon le PC) :

C:\Program Files (x86)\Arduino\libraries

Figure 10 : l'IDE Arduino avec la librairie « SD_uFAT_basic » chargée.

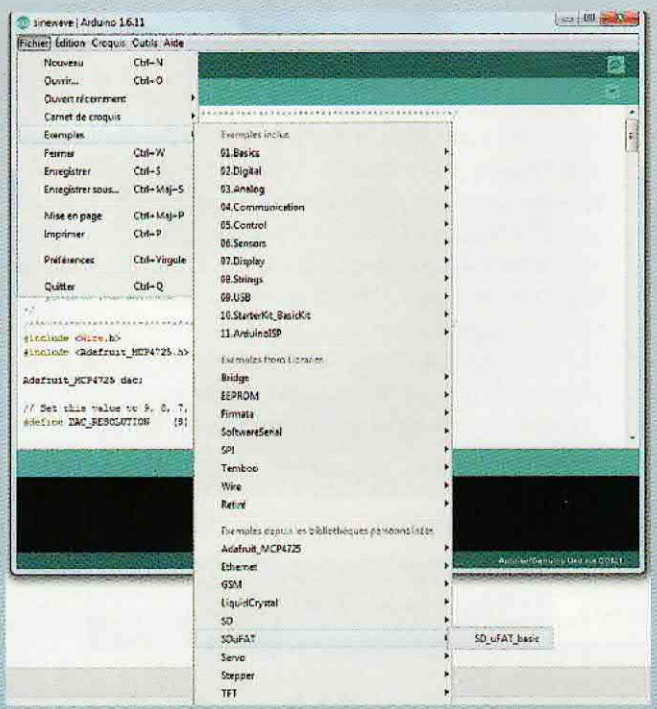


Figure 11 : formatage de la carte SD en cours.

Lorsque vous lancez l'IDE Arduino, vous devez retrouver dans les exemples la librairie « **SD_uFAT_basic** » comme vous pouvez le voir en figure 10.

Prenez la carte SD, insérez-la dans le lecteur de votre PC et assurez-vous qu'elle soit formatée en mode FAT.

Normalement une carte neuve est déjà formatée, mais vous pouvez facilement la reformater au format FAT et en une seule partition.

Créez sur la carte SD un fichier texte que vous nommez « hola.txt ». Ecrivez un certain nombre de caractères dans ce fichier, par exemple **20 caractères** de type « **espace** ».

Attention, utilisez un éditeur de texte qui n'ajoute pas de mise en forme, tel que le « Bloc-notes » de Windows. Copiez sur la carte SD le fichier de test « **hola.txt** » présent dans le dossier « libraries\SDuFAT » de l'IDE (C:\Program Files (x86)\Arduino\libraries\SDuFAT).

Vous devez faire ces opérations car la librairie ne peut pas créer un fichier, elle peut simplement écrire ou lire à l'intérieur des caractères que vous avez déjà saisis.

Insérez la carte SD dans la carte adaptateur, puis à son tour insérez l'adaptateur sur la carte Arduino. Connectez Arduino via le port USB à votre PC.

Redémarrez l'IDE Arduino et chargez l'exemple « SDuFAT ». Maintenant, comme d'habitude, téléverser le sketch Arduino en allant dans le menu « **Croquis → Téléverser** ». Dans le menu « **Outils → Moniteur série** » définissez une vitesse de transmission de 19 200 bps.

Des indications rapportées sur le sketch, vous pouvez en déduire les commandes séries permettant de gérer la carte SD. Pour vérifier que la liaison fonctionne, tapez le caractère « **H** » (en majuscule) et cliquez sur « **SEND** ». Arduino répond avec la liste des commandes (voir la figure 13). Envoyez le caractère « **L** » pour connaitre l'état de la carte SD.

Maintenant, envoyez la commande « **D** » qui permet d'effacer le contenu du fichier et insérer automatiquement le caractère spécial « **ETX** » (End of Text, 0x03 en ASCII) à la fin du texte. Relisez l'état du fichier avec la commande « **L** ».

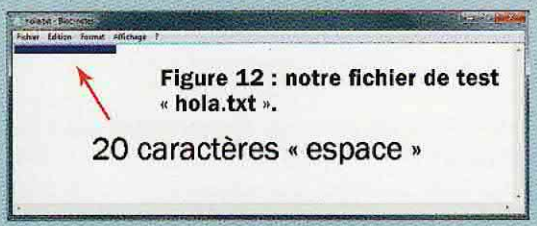


Figure 12 : notre fichier de test « hola.txt ».

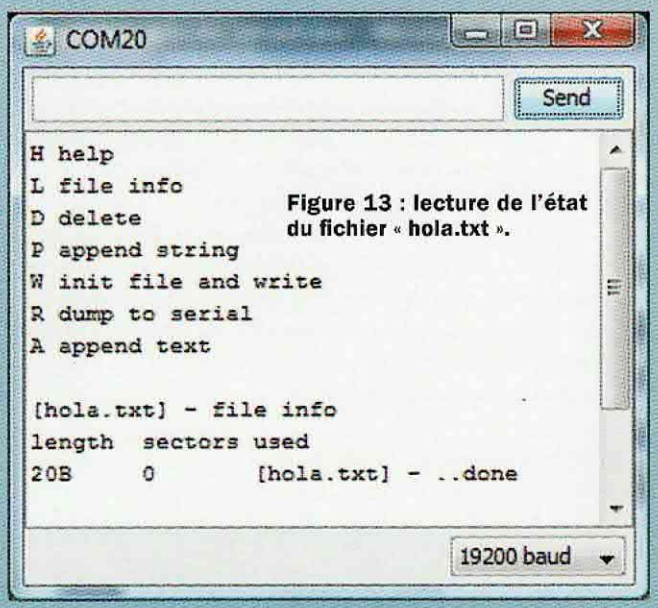


Figure 13 : lecture de l'état du fichier « hola.txt ».

Le paramètre « **length** » (longueur) reportera le nombre de caractères que vous avez initialement ajouté. Il représente aussi le nombre maximum de caractères que vous pourrez écrire.

Le paramètre « **sector** » (secteurs) vous indique le nombre de secteurs occupés par le fichier, et le paramètre « **used** » (utilisés) indique le nombre de caractères utilisés, dans ce cas il est égal à 0.

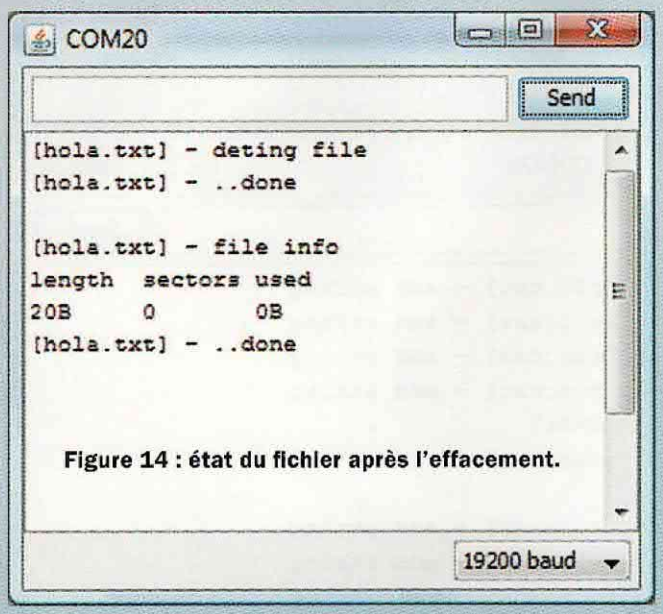


Figure 14 : état du fichier après l'effacement.

Envoyez la commande « **P** » qui permet d'écrire le texte « **hola caracola** ». La confirmation de l'écriture sera effective avec la mention « Done ». Les caractères « **\n** » indiquent l'inclusion d'une fin de ligne.

Répétez la commande « **L** » pour vérifier l'état du fichier, et envoyez la commande « **R** » pour lire le texte écrit dans le fichier (voir la figure 15). Comme vous pouvez le voir maintenant, le paramètre « **used** » (utilisé) vaut **15**, ce qui indique l'écriture de 15 caractères.

Vous pouvez retirer la carte SD de Arduino (sans forcément débrancher quoi que ce soit) et l'insérer dans votre lecteur de carte pour lire le fichier sur votre PC.

Listing 1

```
case 'S':
    Sample=analogRead(0);
    result = SD.println("hola.txt", itoa(Sample, buf, 10));
    result = SD.println("hola.txt", "\r\n");
    Serial.print("AN0=");
    Serial.println(Sample);   // Report sur le Moniteur de série
    break;
```

Vous remarquerez dans le fichier la phrase « hola caracola » suivie d'une série de caractères spéciaux ajoutés lors de la suppression du fichier avec la commande « D ».

En comptant les caractères « Null » (un avant et un après le texte), vous trouverez au total les **20 caractères** que vous avez écrits au début.

Une seconde écriture sur le même fichier impliquera la saturation en raison du dépassement du nombre maximal de caractères. Maintenant, nous allons voir toutes les commandes basiques mises en œuvre dans cette librairie.

- **ls(filename);** : indique la taille maximale, le nombre de secteurs et le nombre de caractères utilisés ;
- **del(filename);** : supprime le fichier et insère le caractère NULL (0x00 en ASCII) au début de chaque secteur ;
- **print(filename, string);** : ajoute une chaîne à la fin du fichier ;
- **println(filename, string);** : ajoute une chaîne et un « EOL » (End Of Line) à la fin du texte ;
- **cat(filename);** : lit le contenu du fichier ou l'envoie sur le port série ;
- **append(filename);** : attend l'arrivée du texte à partir du port série et l'ajoute à la fin du fichier.

Le sketch comprend la gestion de ces commandes en envoyant des caractères sur le port série à l'aide de l'outil « Moniteur série ». Plus précisément, les fonctions des commandes sont les suivantes :

- « **H** » écrit la liste des commandes ;
- « **L** » écrit la liste des informations du fichier (hola.txt) ;
- « **D** » supprime le fichier (hola.txt) ;
- « **P** » écrit dans le fichier « hola.txt » le texte « hola caracola » ;
- « **W** » initialise le fichier, y compris le texte reçu à partir du port série ;
- « **R** » lit le contenu du fichier ou l'envoie au port série ;
- « **A** » ajoute au fichier le texte reçu par le port série.

Étant donné que l'une des principales applications des cartes SD dans les circuits électroniques est l'enregistrement de données à intervalles de temps réguliers (data-logger), nous proposons une extension de l'exemple du sketch en ajoutant la commande « **S** ».

Avec cette commande « **S** », il est possible d'**acquérir la valeur de la broche analogique AN0** et de l'**enregistrer sur la carte SD**.

Le nouveau sketch s'appelle « **SD_01.pde** » (vous pouvez le télécharger sur notre site). Dans le **Listing 1**, nous rapportons la partie intéressante du nouveau sketch de commande.

En plus de la ligne d'acquisition de l'entrée analogique, nous utilisons la fonction « **ITOA** » pour **convertir la valeur numérique en une chaîne**, il est nécessaire d'avoir la compatibilité avec la fonction « **println** » de la librairie « **SDuFAT** ».

a deuxième ligne de « println », c'est-à-dire les caractères « **"\r\n"** », permettent de commencer sur une nouvelle ligne, puis d'écrire toutes les valeurs sous forme de colonne au lieu de les écrire sur une même ligne.

Après l'acquisition de quelques valeurs, il sera possible de vérifier l'enregistrement correct des données en lisant la carte SD sur le PC (figure 19).

Il existe une version simplifiée de la librairie « SDuFAT », optimisée pour réduire l'espace mémoire occupé du microcontrôleur et augmenter la vitesse d'écriture.

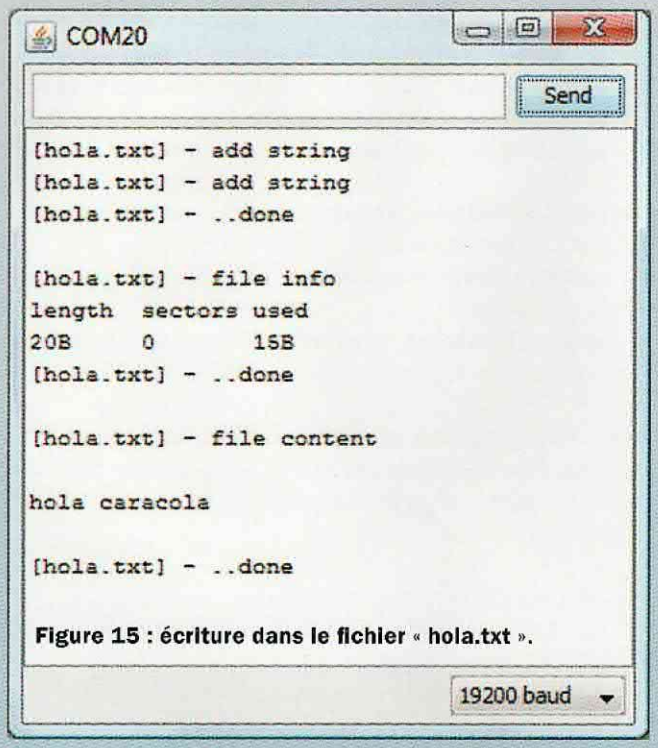


Figure 15 : écriture dans le fichier « hola.txt ».

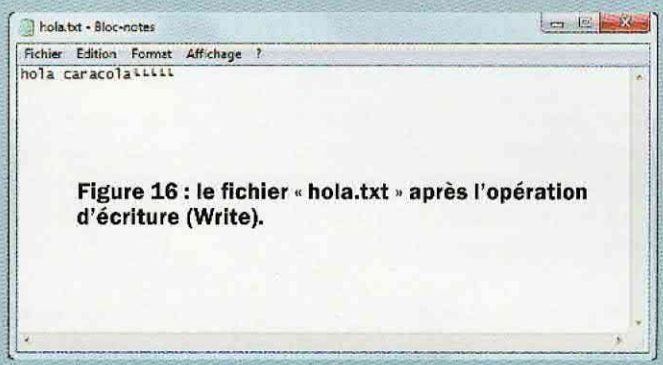


Figure 16 : le fichier « hola.txt » après l'opération d'écriture (Write).

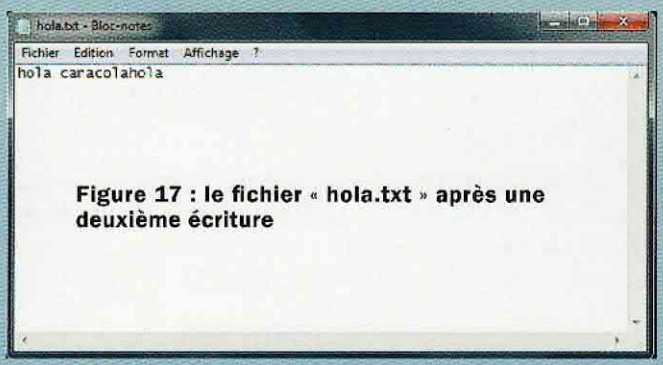


Figure 17 : le fichier « hola.txt » après une deuxième écriture

Elle se nomme « **FileLogger** », elle est téléchargeable à cette adresse : http://code.google.com/p/arduino-filelogger/ (cliquez ensuite sur « Downloads »). Le fichier correspondant s'appelle « FileLogger.V0.X.zip » où X représente la version de la librairie.

Avec cette librairie, vous pouvez écrire sur une carte SD formatée en « FAT16 », avec l'avantage de ne pas nécessairement prédéfinir le texte à l'intérieur. Il suffit qu'au moins un caractère soit présent. Nous avons testé avec succès les version 0.1 et 0.5, mais nous avons eu des problèmes avec la version 0.6.

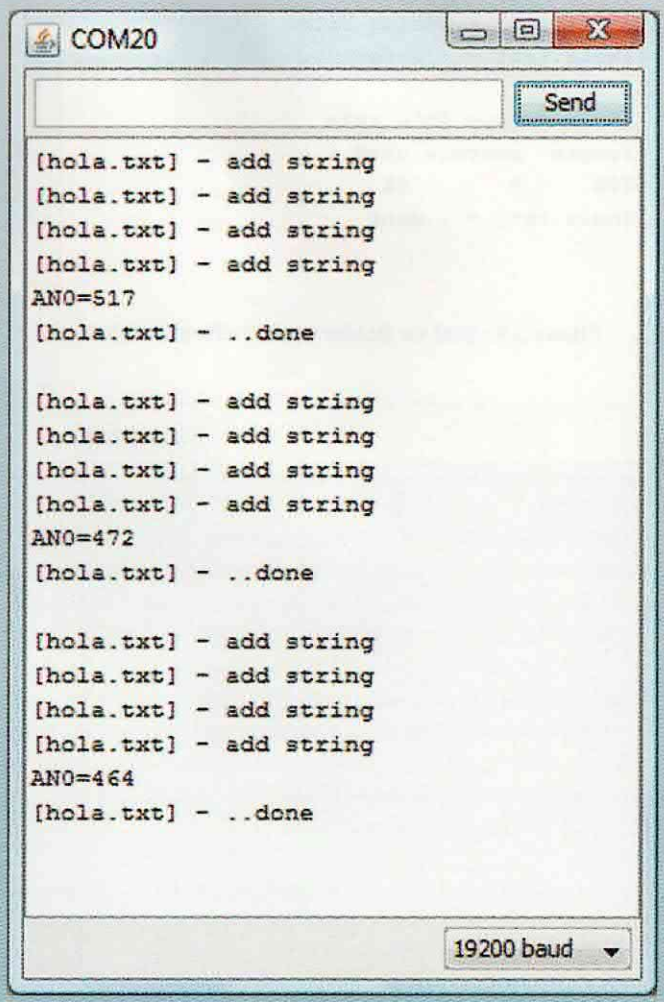


Figure 18 : écriture des données avec l'exemple SD_01.pde.

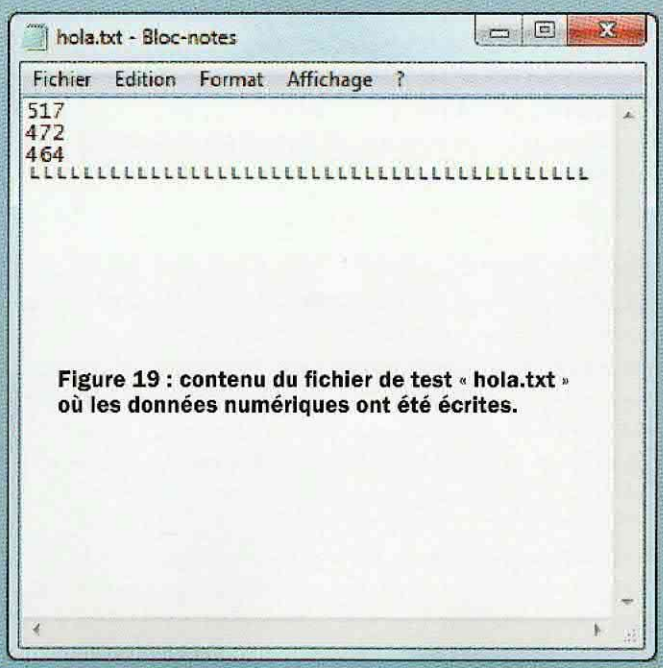


Figure 19 : contenu du fichier de test « hola.txt » où les données numériques ont été écrites.

Après avoir décompressé le fichier zippé, copiez le dossier « FileLogger » directement dans le dossier « libraries » de l'IDE Arduino (le chemin complet doit être comme celui-ci : C:\Program Files (x86)\Arduino\libraries\FileLogger\).

Le fichier d'exemple s'appelle « **FileLoggerDemo** », comme d'habitude vous devez le téléverser dans la carte Arduino.

Listing 2

```
case 'S':
      Sample = analogRead(0);
      strcat(logStr,"\nVal= ");
      itoa(Sample,buffer,10);
      strcat(logStr,buffer);
      Serial.print (logStr);     // retour sur Moniteur série

// conversion de la chaîne de caractères en un tableau
d'octets

 unsigned int length = (strlen(logStr)+1);
      byte bufSD[length];
      int i;
      for(i=0; i<length;i++)
      {
      bufSD[i] = logStr[i];
      }
       if( FileLogger::append("data.log", bufSD, 10) != 0) {
        Serial.println("Error write SD!");
      }
```

Assurez-vous que la carte SD soit formatée en « **FAT16** », créez sur la carte un fichier nommé « **file.log** » et écrivez quelques caractères. Insérez la carte SD dans l'adaptateur et exécutez le sketch. Vous devez également ouvrir le Moniteur série (réglé sur 9600 bauds) et envoyer la commande d'écriture « **W** ».

Il sera écrit dans la carte SD le message déclaré par la ligne « **#define MESSAGE;** ». Si l'opération s'est bien déroulée, vous verrez apparaître sur le Moniteur série la chaîne « **OK** ».

Nous avons également créé un exemple minimaliste (**SD_03.pde**) dans lequel, en plus d'écrire un texte, est écrit la valeur acquise sur l'entrée analogique AN0. La partie du code qui effectue cette étape est représentée dans le **Listing 2**.

Comme vous pouvez le voir, les instructions sont très simples et comportent essentiellement l'acquisition des données de l'entrée analogique 0 ainsi que la création de la chaîne « **logStr** » qui contient la valeur acquise et qui sera affichée dans le Moniteur de série.

La chaîne de caractères est ensuite convertie en un tableau d'octets selon les besoins de l'instruction « **FileLogger::append** » qui gère l'écriture sur la carte SD.

Une autre librairie très avancée, avec une grande documentation relative à la partie matérielle (en Anglais), est disponible à l'adresse suivante : https://github.com/greiman/SdFat. Il s'agit de la librairie « **SdFat** ».

Elle utilise le bus SPI à une vitesse de 8 MHz, le translateur de niveau doit être un circuit intégré. Elle a été testée avec succès sur notre adaptateur. Elle permet de gérer l'écriture, la lecture, la création et la suppression de fichiers et gère également les sous-répertoires. Elle fonctionne avec des cartes SD et SDHC formatées en « FAT16 » ou « FAT32 ».

Décompressez le fichier « **SdFat-master.zip** » et copiez uniquement le dossier « **SdFat** » dans le dossier « libraries »

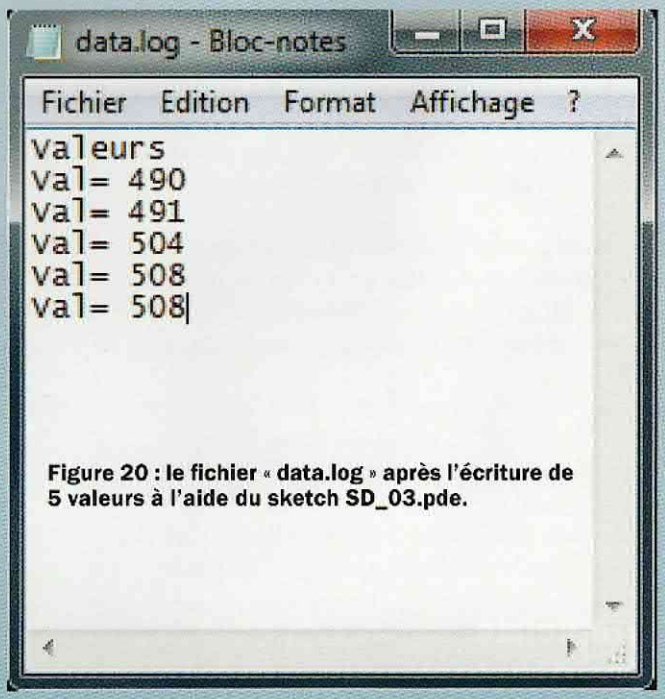


Figure 20 : le fichier « data.log » après l'écriture de 5 valeurs à l'aide du sketch SD_03.pde.

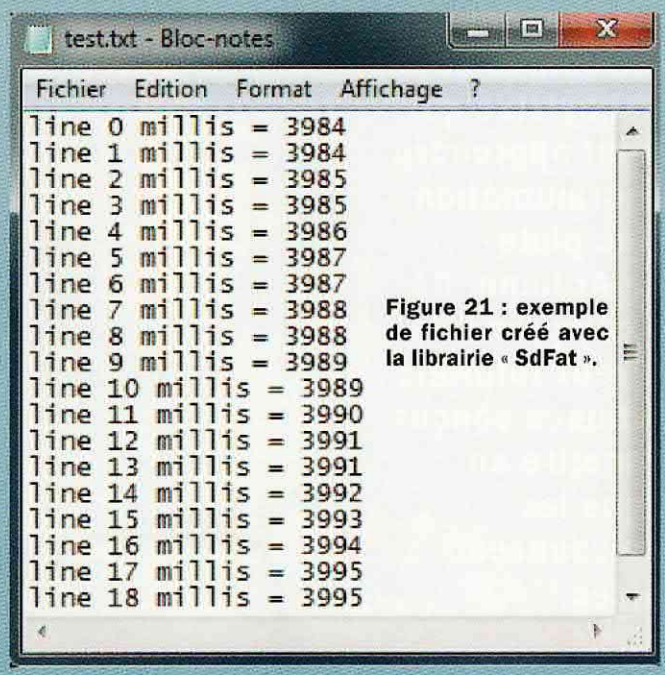


Figure 21 : exemple de fichier créé avec la librairie « SdFat ».

de l'IDE Arduino, de sorte que vous aurez aussi plusieurs fichiers d'exemples.

Téléversez dans Arduino l'exemple « ReadWrite ». Créez un fichier « test.txt » et lancez le sketch. Si tout se passe bien le fichier doit contenir la chaîne de caractères suivante « testing 1, 2, 3. ».

Pour tester les fonctionnalités de cette librairie, nous avons écrit un petit sketch (**SD_02.pde**) qui permet d'ajouter au fichier « **test.txt** » 10 valeurs acquises à partir de l'entrée « Analog0 » à des intervalles d'une seconde.

Comme indiqué dans la documentation, il est possible de diminuer la vitesse du BUS à 4 MHz au lieu de 8 MHz, afin d'assurer la compatibilité avec des translateurs de niveaux à résistances.

Cette librairie gère une carte SD en FAT16 avec les fonctions de lecture, écriture, création et suppression de fichiers dans un répertoire principal avec des noms au format 8.3 (8 caractères pour le nom et 3 pour l'extension).

Elle a été conçue pour une utilisation en haut débit du bus SPI avec des translateurs de niveaux à circuits intégrés. Attention cependant, **l'utilisation d'une telle librairie demande des notions avancées en langage C**.

Afin de vérifier la compatibilité, nous avons également testé une librairie FAT16 créée par Ryan Owens et disponible à l'adresse : http://www.roland-riegel.de/sd-reader/index.html.

Vous devez télécharger la version 2010-10-10, c'est celle qui semble fonctionner le mieux.

Le fichier se nomme « **sd-reader_source_20101010.zip** ». Cette librairie permet l'écriture et la lecture d'un fichier existant sur une carte de type MMC, SD et SDHC formatée en « FAT16 » et en « FAT32 » (« FAT32 » uniquement avec ATmega328). Le principe de fonctionnement reste similaire aux autres librairies décrites précédemment.

La librairie « SDFAT » disponible à l'adresse www.sparkfun.com/tutorial/microSD_Shield/SdFat.zip, dont le nom du fichier est « **SdFat.zip** », correspond à la version « **Sparkfun** » de la librairie SDFATLIB dont elle hérite de toutes les fonctionnalités.

Tableau 7 : mémoire utilisée pour chaque librairie.

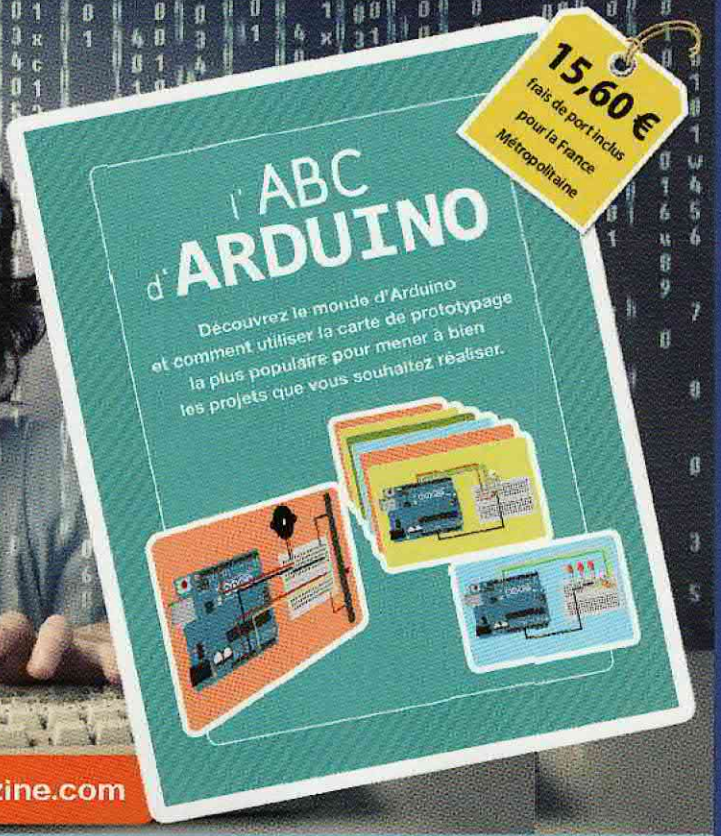
Librairie	Taille en octets du sketch
SDuFAT	8340 à 30720
FileLogger	6020 à 30720
SDFATLIB	11706 à 30720

Vous devez garder à l'esprit que plus une librairie est complexe et riche en fonctionnalités, plus elle aura besoin de ressources en termes de mémoire. C'est pour cette raison que nous vous recommandons d'utiliser la librairie qui dispose du minimum de fonctions requises pour votre application.

A titre d'exemple, dans le Tableau 7, nous indiquons l'occupation de la mémoire pour chaque librairie dans le cas de l'exemple base.

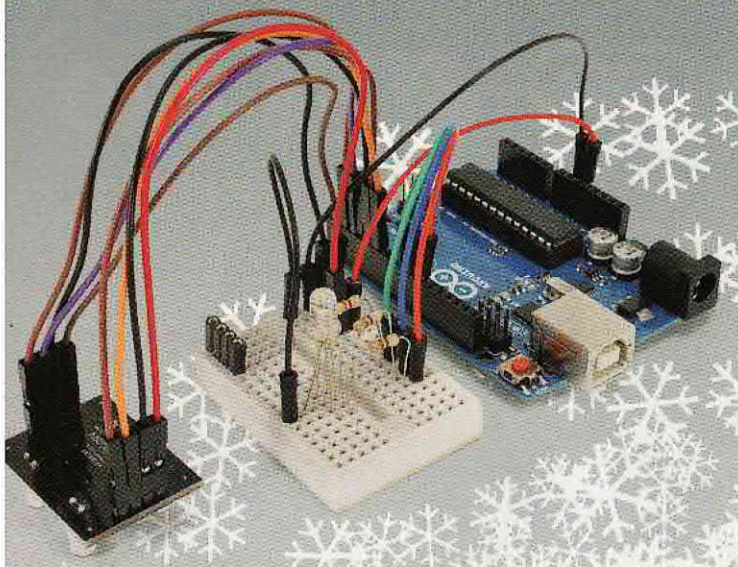
Nous arrivons au terme de cette 7ème leçon sur Arduino, vous pouvez télécharger sur notre site www.electroniquemagazine.com dans le sommaire détaillé de la revue 137 à l'onglet « Télécharger » **toutes les librairies et programmes** relatifs à cette partie du cours Arduino.

Un livre pensé pour les débutants qui abordent pour la première fois l'électronique et pour les hobbyistes qui désirent apprendre la programmation sur une plate-forme Arduino. Ce livre comprend de nombreux tutoriels didactiques conçus pour mettre en pratique les connaissances acquises.



Achetez en ligne : www.electroniquemagazine.com

ET 1170



Un sapin de Noël caméléon

de Marco MAGAGNIN

Il existe deux approches différentes pour réaliser un gadget de Noël qui prend la couleur du support sur lequel il repose. C'est l'occasion de voir la différence entre le développement d'un projet « amateur » et d'un produit « commercial ».

Souvent, il nous arrive à la Rédaction de débattre de manière spontanée sur la façon de traiter les projets qui sont ensuite présentés dans la revue. L'un des principaux problèmes est l'approche à utiliser pour un projet donné.

Est-il préférable de proposer un montage avec des composants discrets, ou utiliser des composants de plus en plus intégrés comme par exemple un microcontrôleur ?

De toute évidence, les solutions sont souvent liées à la formation et l'expérience des concepteurs du projet.

C'est l'une des raisons pour lesquelles les discussions deviennent « chaudes » lors des réunions... La situation décrite s'est présentée à plusieurs reprises lors de la décision de présenter ce projet comme un gadget pour Noël. Il a été convenu de vous proposer dans cet article une lampe à LED capable de prendre la couleur de la surface sur laquelle elle repose, de plus elle peut être insérée dans un sapin de Noël creux.

À ce stade, une série de questions inévitables se sont posées : quelles lampes ? comment reconnaître les couleurs ? comment reproduire le montage ?

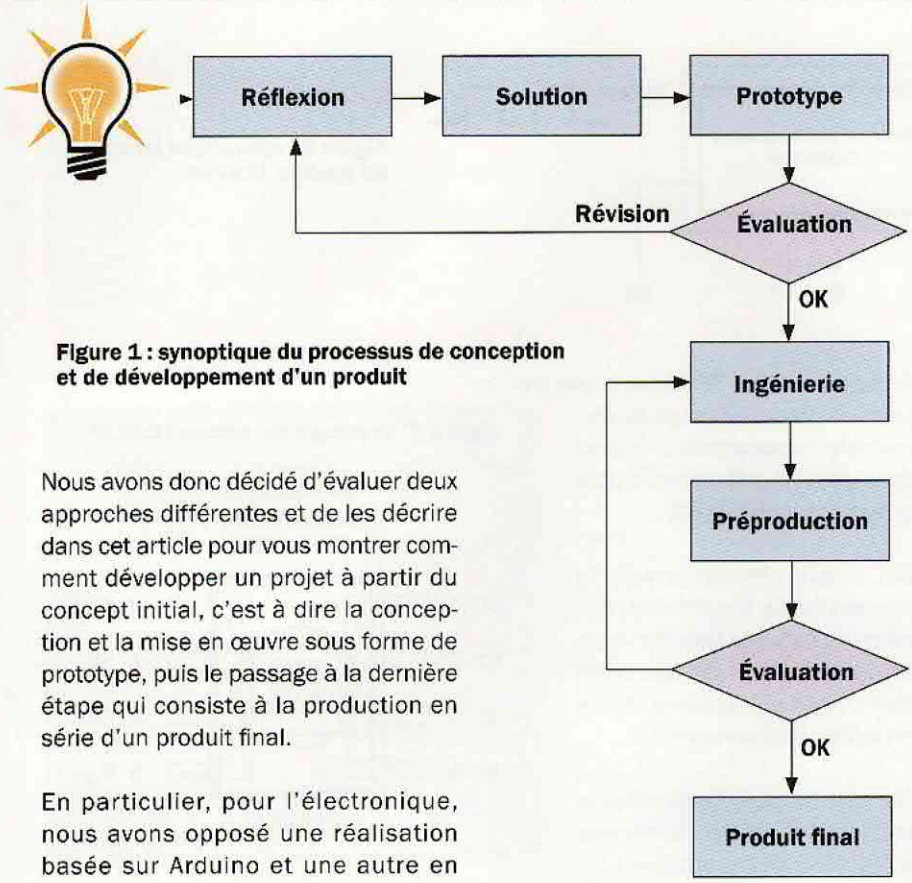


Figure 1 : synoptique du processus de conception et de développement d'un produit

Nous avons donc décidé d'évaluer deux approches différentes et de les décrire dans cet article pour vous montrer comment développer un projet à partir du concept initial, c'est à dire la conception et la mise en œuvre sous forme de prototype, puis le passage à la dernière étape qui consiste à la production en série d'un produit final.

En particulier, pour l'électronique, nous avons opposé une réalisation basée sur Arduino et une autre en partant de zéro basée sur un microcontrôleur spécifique.

La première réalisation est idéale pour le développement d'un prototype, cependant elle présente un coût et un encombrement inadaptés à une production de masse. La carte à microcontrôleur spécifique étant dans ce cas la meilleure solution.

L'approche du projet

Le synoptique du processus de conception et de développement d'un produit est représenté en figure 1.

Nous partons de l'hypothèse, dans notre cas, de réaliser notre lampe de Noël qui soit en mesure de prendre la même couleur que la surface sur laquelle elle est posée. La première étape qui conduit, à partir d'une idée simple, à l'hypothèse de ce projet est la définition de sa fonctionnalité, le choix des composants ainsi que la conception du design.

Le processus inclut une première phase dédiée au choix des composants, la réalisation de l'électronique ainsi que l'aspect esthétique (design) du corps externe de la lampe.

Sur la base des premiers choix, nous avons réalisé un premier prototype afin d'évaluer et d'améliorer le processus de fabrication jusqu'à obtenir une version adaptée à une production en série.

La contrainte de ce projet a été de réaliser différents prototypes de la lampe à l'aide d'une imprimante 3D, de manière à obtenir un prototype très similaire en apparence au produit final. La forme initiale sera celle d'un arbre en 3 dimensions rappelant la forme d'un sapin de Noël.

Maintenant, étudions le fonctionnement de la lampe qui consiste à détecter la couleur de la surface sur laquelle elle repose et à émettre une lumière correspondant à la couleur de la surface.

Cela introduit une première contrainte qui est que le corps de la lampe doit être constitué d'un matériau translucide avec une couleur qui n'interfère pas avec la teinte de la lumière émise, afin de ne pas fausser les couleurs. Dans la pratique, le blanc est obligatoire.

Pour simplifier le développement du projet, nous devons le diviser en différentes parties ayant chacune une seule fonction, nous les appellerons « modules ».

Nous avons donc un « module capteur » qui permet de reconnaître la couleur de la surface sur laquelle la lampe est posée, un « module de traitement » capable de comprendre et d'interpréter le signal de sortie du capteur afin de fournir « quelque chose » capable de piloter une source lumineuse « programmable ». La dernière partie étant un « module lumière » capable de reproduire une teinte lumineuse en fonction d'un signal d'entrée prédéfini.

Les fonctions individuelles des « modules » ou « étages » doivent être définies une fois pour chacun d'eux, à ce stade nous devons choisir la technologie à adopter et donc l'ingénierie qui en découle.

En ce qui concerne le module de reconnaissance de couleur, nous disposons de plusieurs solutions. Nous pouvons utiliser un appareil photo et ensuite traiter les images acquises pour extrapoler la couleur prédominante, cependant cette solution nécessite un matériel doté d'une puissance de traitement assez importante.

Nous pouvons aussi envisager la construction d'un capteur à l'aide de composants discrets, des capteurs de lumière avec des diodes multicolores reliées à une logique de traitement et de commande, une solution déjà plus appropriée mais encore trop compliquée à mettre en œuvre.

Une troisième solution consiste à chercher sur le marché un produit existant remplissant les fonctions dont nous avons besoin. Ce sont les mêmes critères que nous adoptons quand nous réalisons des projets sous GNU/Linux, à savoir la recherche de « briques » (blocs de construction) disponibles et fonctionnelles afin de réaliser un projet en les ajoutant les unes avec les autres. Cette recherche sera fructueuse lorsque nous aurons décomposé le projet en différents « modules » (parties).

Nous avons donc trouvé le circuit intégré « **TCS230** », il s'agit d'un capteur doté d'une reconnaissance des couleurs. Il est fabriqué par la société **Shenzhen LC Technology Co** (www.chinalctech.com).

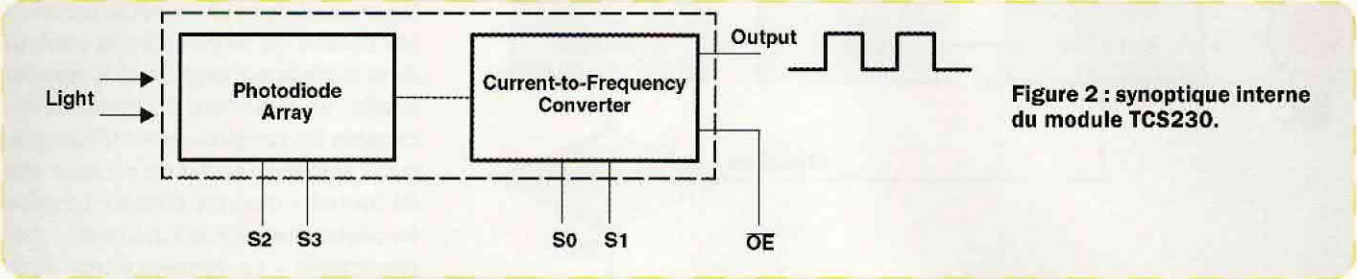


Figure 2 : synoptique interne du module TCS230.

Pour reproduire les couleurs détectées par le capteur, le choix est très simple.

Nous utilisons une LED RVB aussi bien au stade de prototype qu'au stade de produit final, étant donné que le pilotage de LED RVB s'applique à une large gamme de produits disponibles sur le marché avec différentes formes, tailles, puissances et prix.

Maintenant abordons la définition de la fonction principale du « module de traitement » qui consiste à produire pour chaque couleur un **signal PWM proportionnel à l'intensité de la composante de la couleur** correspondante **détectée** par le capteur.

Il nous reste plus qu'à choisir le type de « module de traitement » à utiliser pour une fonction unique. Probablement que le bon choix est d'utiliser un microcontrôleur avec un nombre suffisant d'entrées/sorties analogiques et digitales.

Les performances du microcontrôleur pour une telle application ne sont pas très importantes, car une lampe est un objet statique.

Une fois placée sur une surface, il ne faut plus la déplacer jusqu'à ce qu'on veuille qu'elle change de couleur.

Pour être clair, notre application ne nécessite pas une solution en temps réel basée sur un FPGA avec une fréquence d'échantillonnage élevée, un simple microcontrôleur économique est plus que suffisant pour les besoins de ce projet.

Pour le premier prototype, le choix est facile. Il suffit d'utiliser une carte **Arduino Uno**. Sa vocation est précisément de créer rapidement et facilement des prototypes fonctionnels.

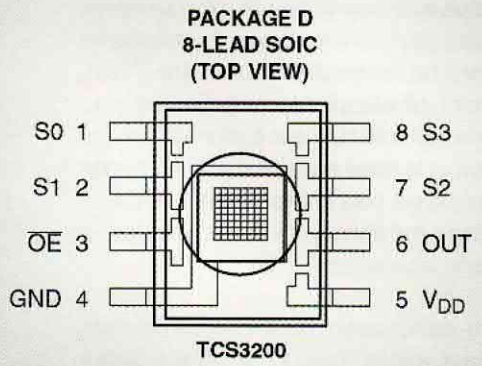
L'IDE et le langage de développement Arduino ont été développés de manière à être accessible à tous. Nous avons tout le nécessaire pour entreprendre la réalisation de notre premier prototype.

Pour la production en série, nous devrons remplacer la carte Arduino Uno et probablement la LED RVB, par une solution adaptée à une production de masse.

Nous étudierons cela en temps voulu. Arduino Uno a fait l'objet de nombreux articles dans notre revue et d'un livre « **L'ABC d'Arduino** ». Reportez-vous aux revues **131** à **136** d'**Electronique et Loisirs Magazine** ainsi qu'au livre « **L'ABC d'Arduino** » disponible à la vente sur notre site **www.electroniquemagazine.com**.

Nous allons dans cet article nous concentrer sur les caractéristiques du module TCS230 et l'utilisation de la LED RVB.

Figure 3 : brochage du module TCS230



Le module TCS230

Le module TCS230 est produit par la société Shenzhen LC Technology Co basée en Chine, dans la région de Shenzhen. Ce module contient un **capteur optique** (photodiode) et un **convertisseur programmable** qui permet de transformer la **lumière détectée en une fréquence proportionnelle à l'intensité lumineuse reçue**.

Le circuit intégré est en technologie CMOS. Il comprend une matrice 8 x 8 photodiodes et un convertisseur « **courant/fréquence** » selon le synoptique de la figure 2.

Dans le capteur, il existe **3 filtres de couleur R, V** et **B**. La mesure des composantes de la couleur se compose

Tableau 1 : signification des broches du module TCS230.

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
GND	4		Power supply ground. All voltages are referenced to GND.
$\overline{OE}$	3	I	Enable for f_o (active low).
OUT	6	O	Output frequency (f_o).
S0, S1	1, 2	I	Output frequency scaling selection inputs.
S2, S3	7, 8	I	Photodiode type selection inputs.
V_{DD}	5		Supply voltage

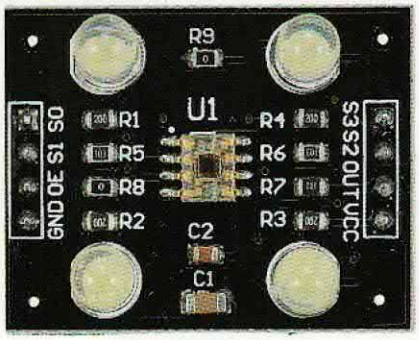


Figure 4 : photo du module TCS230, vous pouvez voir sur les côtés du circuit les broches correspondant au Tableau 1.

d'une **première mesure de l'intensité de la lumière totale reçue** (sans les filtres) et ensuite d'une **seconde mesure relative à l'intensité de la couleur** en appliquant un filtre à la fois.

En figure 3, vous pouvez voir le brochage du module TCS230, tandis que le Tableau 1 définit la signification de chaque broche. En figure 4, vous pouvez voir sur les côtés du circuit imprimé du module TCS230 les connexions correspondantes des broches décrites dans le Tableau 1.

Les **diodes présentes sur le module** sont alimentées en parallèle et servent à **éclairer la surface à analyser** par le capteur.

Le module peut être alimenté avec une tension comprise entre 2,7 VDC et 5,5 VDC. Il est donc compatible avec les niveaux des tensions d'un microcontrôleur alimenté en 3,3 VDC ou 5 VDC.

Le convertisseur du TCS230 lit les photodiodes qui constituent la matrice. En fonction de la configuration des broches de contrôle S2 et S3 (voir le Tableau 2), la matrice de diodes filtre la lumière rouge, verte, bleue ou blanche et génère en sortie sur la broche « OUT » un signal carré avec un rapport cyclique de 50 % et une fréquence directement proportionnelle à l'intensité de la lumière perçue (irradiante).

Le fonctionnement du capteur exploite le fait que les **différentes nuances de couleurs peuvent être attribuées à une combinaison des 3 couleurs de base, rouge, verte** et **bleue** dans des proportions appropriées (voir la figure 5).

S2	S3	PHOTODIODE TYPE
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Tableau 2 : filtrage de la lumière en fonction de la configuration des broches de contrôle S2 et S3.

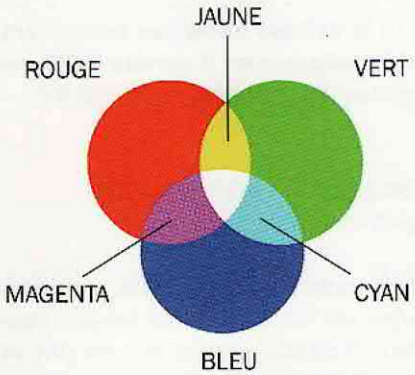


Figure 5 : combinaison des 3 couleurs de base formant les différentes nuances de couleurs.

Lorsque le filtre est réglé pour une couleur prédominante spécifique, par exemple la couleur bleue, le TCS230 mesure uniquement l'intensité de la couleur considérée, à l'exclusion des autres. De cette façon, il est possible de déterminer la couleur de la composante bleue.

La même procédure permet de déterminer l'intensité des autres composantes rouge et verte. L'amplitude du spectre des fréquences émises peut être mise à l'échelle à l'aide d'un facteur spécifique exprimé dans le Tableau 3.

En configurant correctement les **niveaux** sur les broches **S0** et **S1**, il est possible d'**adapter l'amplitude** à la capacité de traitement du microcontrôleur utilisé.

S0	S1	OUTPUT FREQUENCY SCALING (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

Tableau 3 : mise à l'échelle de l'amplitude du spectre des fréquences émises en fonction du facteur spécifique.

Les entrées et les sorties peuvent être interfacées à un microcontrôleur pour la configuration et la lecture des valeurs de la fréquence, ce que nous ferons avec notre carte Arduino Uno.

La LED RVB

Pour **reproduire les teintes des couleurs perçues** par le capteur, nous utilisons une **LED RVB** dans notre prototype. Il s'agit d'un dispositif optoélectronique **constitué de 3 LED** qui émettent respectivement une lumière **rouge, bleue** et **verte** (voir la figure 6).

Il existe deux types de LED RVB, celles à **anode commune** et celles à **cathode commune**. Dans le premier cas les 3 anodes, c'est-à-dire les bornes positives, sont reliées entre elles et forment le « commun » (anode commune). Les 3 LED sont commandées individuellement à l'aide de leur cathode.

Dans les LED à cathode commune, les 3 LED ont leur borne négative reliées ensemble, tandis que la commande de chaque LED s'effectue par l'anode. Cette dernière configuration est plus appropriée pour une utilisation avec une carte Arduino Uno.

En fait, pour **piloter** les LED, nous utilisons **3 sorties analogiques** (DAC) de la carte Arduino Uno, capables de convertir des valeurs numériques comprises entre 0 et 255 en valeurs de tensions proportionnelles comprises entre 0 et + 5 V. Chaque nuance de couleur, dans la représentation RVB avec une précision de 8 bits, est représentée par un triplet de valeurs comprises dans l'intervalle [0 – 255] qui « traduit » (reflète) l'intensité de chaque composante de la couleur.

Par exemple, le rouge intense est représenté par le triplet (255, 0, 0). À l'adresse web suivante : **http://www.toutes-les-couleurs.com/code-couleur-rvb.php**, vous trouverez des informations sur les teintes de base et les correspondances avec les valeurs RVB.

Il convient de souligner une particularité relative aux nuances de gris. Ceux d'entre vous qui prennent des photos, savent que le traitement des tons

de gris sont exprimés par des valeurs RVB égales, par exemple (150, 150, 150). Il n'est pas possible de reproduire avec notre LED des nuances de gris. Nous veillerons à ce que dans ces conditions, la LED émette une lumière blanche. La LED RVB que nous utilisons pour notre prototype est référencée **540R2GBC-CC**, cependant vous pouvez utiliser un autre modèle de LED RVB.

La stratégie de traitement

La stratégie de traitement du microcontrôleur du projet est visible sur le diagramme de la figure 7. Après l'initialisation du microcontrôleur, le programme démarre une boucle principale de traitement qui exécute à l'infini les opérations décrites ci-après.

Dans un **premier temps**, nous vérifions **si la lampe est posée sur une surface ou maintenue surélevée** en cours de déplacement d'un endroit à un autre. Dans ce cas, nous suspendons la fonction de « camouflage » pour aller vers le prochain processus de traitement. Si nous détectons que **la lampe est placée sur une surface**, il faut agir sur le capteur de manière à ce qu'il **reconnaisse la teinte de la couleur** de la surface sur laquelle la lampe est posée.

Pour obtenir ce résultat, nous effectuons une série de lectures de la fréquence de sortie, qui, comme nous l'avons vu, est proportionnelle à l'intensité de la composante lumineuse, en modifiant de temps à autre la configuration des filtres.

La **première lecture est effectuée en l'absence de filtres**, de façon à obtenir la lumière totale réfléchie par la surface. Ensuite nous effectuons des **lectures successives** en insérant **à chaque fois un filtre différent**, respectivement le filtre pour le **rouge**, pour le **vert** et pour le **bleu**.

Après chaque lecture, nous sommes en mesure de déterminer la façon dont la composante de la couleur sélectionnée est présente dans la teinte globale, en terme de pourcentage.

À la fin des lectures successives, nous prenons la valeur la plus grande et nous lui assignons le niveau « 255 », puis nous relions les deux autres proportionnellement, avec quelques ajustements que nous avons trouvés expérimentalement sur la base du comportement du capteur, comme décrit dans la documentation technique.

Les **3 valeurs obtenues** permettent de **configurer** les **3 sorties PWM** qui pilotent les **3 anodes** de la LED RVB.

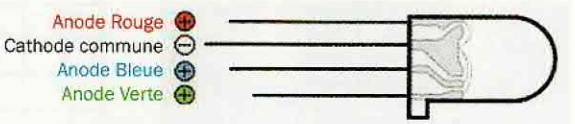


Figure 6 : une LED RVB est constituée de 3 LED qui émettent respectivement une lumière rouge, bleue et verte.

Schéma électrique du prototype

Sur le schéma de la figure 8, nous pouvons voir les connexions entre le capteur TCS230, la carte Arduino Uno et la LED RVB. L'assemblage du prototype avec une plaque d'essai est visible en figure 9.

Les broches S0 et S1, qui sont utilisées pour configurer la fréquence de la tension de sortie du capteur TCS230, sont reliées aux broches 2 et 3 de la carte Arduino. Les broches **S2** et **S3**, qui permettent de **définir le filtre à utiliser** pour la lecture de l'intensité lumineuse, sont reliées respectivement aux sorties digitales 6 et 5 de la carte Arduino.

La **sortie du capteur TCS230 est reliée** à l'entrée **4** d'Arduino. Enfin, les **3 anodes** de la LED RVB sont connectées aux **sorties PWM 9** (rouge),

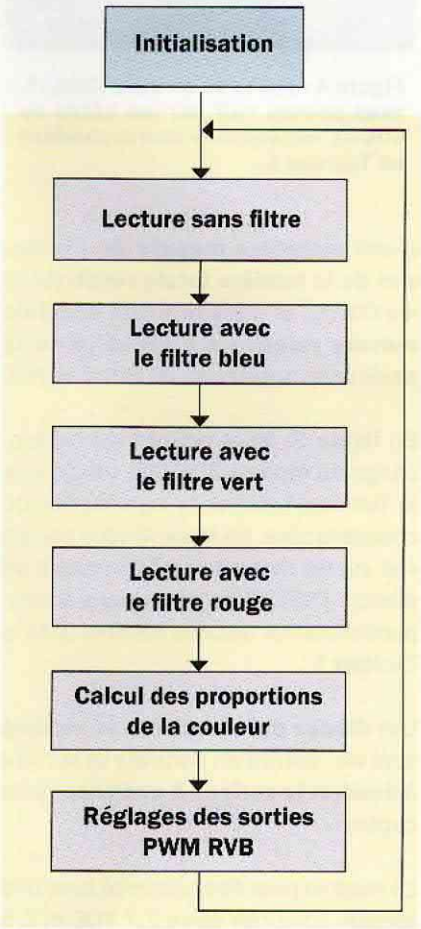


Figure 7 : diagramme de la stratégie de traitement du microcontrôleur.

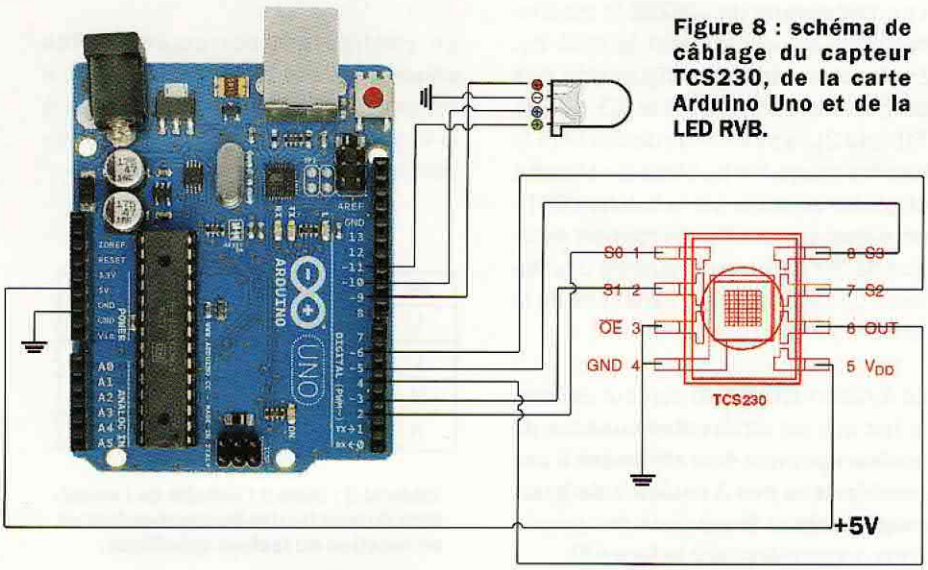


Figure 8 : schéma de câblage du capteur TCS230, de la carte Arduino Uno et de la LED RVB.

10 (verte) et **11** (bleue) de la carte Arduino. Les sorties PWM sont identifiées par un « ~ » sur la sérigraphie de la carte Arduino. Nous avons utilisé les 3 sorties précitées, car **elles sont caractérisées par une même fréquence du signal PWM.**

Si nous avions choisi d'autres broches, nous aurions pu avoir un fonctionnement anormal car les broches 5 et 6 ont une fréquence de sortie de 976 Hz tandis que les broches 3, 9, 10 et 11 ont une fréquence de sortie de 488 Hz.

Pour éviter de mauvaises surprises et des dysfonctionnements qui sont parfois difficiles à identifier lors de la réalisation d'un projet, il est conseillé de lire la documentation technique des composants.

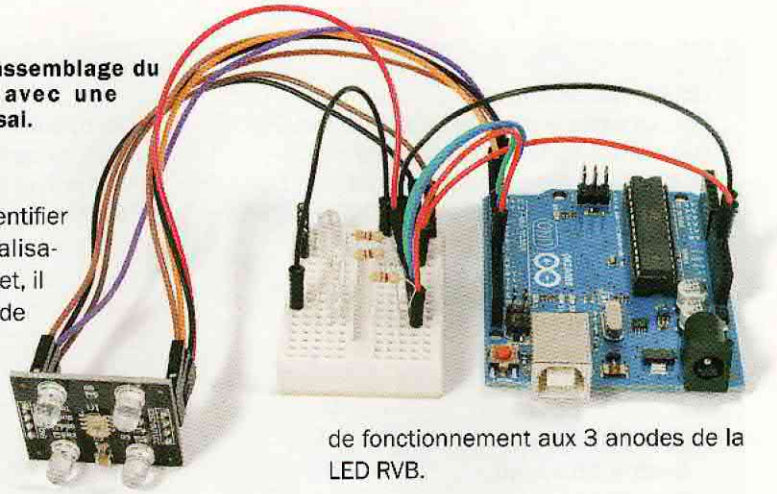


Figure 9 : assemblage du prototype avec une plaque d'essai.

Dans notre cas, nous avons relié les 3 sorties PWM ayant une même fréquence de fonctionnement aux 3 anodes de la LED RVB.

Les **3 résistances** de 470 Ω **limitent le courant** dans chaque LED à 10 mA.

Listing 1

```
/*
Sketch lampe de Noël
*/
int S0 = 2;                          // broche de contrôle de la fréquence de sortie
int S1 = 3;                          // broche de contrôle de la fréquence de sortie
int S2 = 6;                          // broche de contrôle du filtre de couleur
int S3 = 5;                          // broche de contrôle du filtre de couleur
int taosOutPin = 4;                  // broche de sortie de la fréquence
int LED = 13;
int R = 9;                           // PWM couleur rouge
int G = 10;                          // PWM couleur verte
int B = 11;                          // PWM couleur bleue
double Coeff = 0;
int Ro = 0;
int Go = 0;
int Bo = 0;

void setup() {
 TCS230setup();
 Serial.begin(115200);
 Serial.print("\n\n\nready\n\n\n");
 pinMode(R,OUTPUT);
 pinMode(G,OUTPUT);
 pinMode(B,OUTPUT);
 delay(100);
}
void loop() {
 Serial.print(detectColor(taosOutPin));      // configuration du port série pour débogage
 Serial.print("\n\n\n");
 delay(1000);
}
int detectColor(int taosOutPin){
                                             // contrôle de la présence d'une surface
 double isPresentTolerance = 5;
 double isPresent = colorRead(taosOutPin,0,0)/colorRead(taosOutPin,0,1);
 Serial.print("isPresent:");
 Serial.println(isPresent,2);
 Serial.print("valeur isPresentTolerance:");
 Serial.println(isPresentTolerance,2);
 if(isPresent < isPresentTolerance){
 Serial.println("pas de surface détectée");
 return 0;
```

```
}
  double red,blue,green;
  double white = colorRead(taosOutPin,0,1);        // lecture de l'intensité lumineuse sans filtre
  red = white/colorRead(taosOutPin,1,1);           // lecture de l'intensité de la composante rouge
  blue = white/colorRead(taosOutPin,2,1);          // lecture de l'intensité de la composante bleue
  green = white/colorRead(taosOutPin,3,1);         // lecture de l'intensité de la composante verte
Serial.print("red");
Serial.println(red);
Serial.print("blue");
Serial.println(blue);
Serial.print("green");
Serial.println(green);

if(red > blue && red > green){
    Coeff = 255 / red;
    Ro = 255;
    Go = green * Coeff;
    Bo = blue * Coeff;
    LED_RGB(Ro, Go, Bo);
    return 1;
  }
  if(blue > green && blue > red){
    Coeff = 255 / blue;
    Bo = 255;
    Go = green * Coeff;
    Ro = red * Coeff;
    LED_RGB(Ro, Go, Bo);
   return 2;
  }
  if(green > blue && green > red){
    Coeff = 255 / green;
    Go = 255;
    Ro = red * Coeff;
    Bo = blue * Coeff;
    LED_RGB(Ro, Go, Bo);
    return 3;
  }
}

double colorRead(int taosOutPin, int color, boolean LEDstate){
                                                   //assurez-vous que la broche est configurée en entrée
    pinMode(taosOutPin, INPUT);
    taosMode(1);
  int sensorDelay = 3;
                                                   // définit les broches pour sélectionner les filtres
  if(color == 0){                                  // blanc sans filtre
    digitalWrite(S3, LOW);                         //S3
    digitalWrite(S2, HIGH);                        //S2
  // Serial.print(" w");
  }else if(color == 1){                            // active le filtre rouge
    digitalWrite(S3, LOW);                         //S3
    digitalWrite(S2, LOW);                         //S2
  // Serial.print(" r");
  }else if(color == 2){                            // active le filtre bleu
    digitalWrite(S3, HIGH);                        //S3
    digitalWrite(S2, LOW);                         //S2
  // Serial.print(" b");
  }else if(color == 3){                            // active le filtre vert
    digitalWrite(S3, HIGH);                        //S3
    digitalWrite(S2, HIGH);                        //S2
  // Serial.print(" g");
  }
  double readPulse;
  if(LEDstate == 0){
   taosMode(0);
```

```
//      digitalWrite(LED, LOW);
  }
  if(LEDstate == 1){
  taosMode(1);
//      digitalWrite(LED, HIGH);
  }
  delay(sensorDelay);
  readPulse = pulseIn(taosOutPin, LOW, 80000);
                                        // gestion de la temporisation de la lecture
  if(readPulse < .1){
    readPulse = 80000;
  }
                                        // éteint les LED du capteur
  taosMode(0);
  return readPulse;
}
                                        // gestion de l'émission des fréquences.
taosMode(0);

                                        // capteur au repos
void taosMode(int mode){
  if(mode == 0){
  //power OFF
    digitalWrite(LED, LOW);
    digitalWrite(S0, LOW);  //S0
    digitalWrite(S1, LOW);  //S1
  // Serial.println("mOFFm");
  }else if(mode == 1){
                                        // fréquence 1:1
    digitalWrite(S0, HIGH);  //S0
    digitalWrite(S1, HIGH);  //S1
  // Serial.println("m1:1m");
  }else if(mode == 2){
                                        // fréquence 1:5
    digitalWrite(S0, HIGH); //S0
    digitalWrite(S1, LOW); //S1
  //Serial.println("m1:5m");
  }else if(mode == 3){
                                        // fréquence 1:50
    digitalWrite(S0, LOW); //S0
    digitalWrite(S1, HIGH); //S1
  //Serial.println("m1:50m");
  }
  return;
}
void LED_RGB(int Rx, int Gx, int Bx){
  analogWrite(R, Rx);
  analogWrite(G, Gx);
  analogWrite(B, Bx);
  return;
}
void TCS230setup(){
                                        // initialise la gestion des fréquences des broches
  pinMode(LED,OUTPUT);
                                        // broches de sélection des filtres de couleurs
  pinMode(S2,OUTPUT);                   // S2
  pinMode(S3,OUTPUT);                   // S3
                                        // broche de sortie conversion lumière/fréquence
  pinMode(taosOutPin, INPUT);
                                        // sélection du mode de la fréquence de sortie
  pinMode(S0,OUTPUT);                   // S0 pinB
  pinMode(S1,OUTPUT);                   // S1 pinA
  return;
}
```

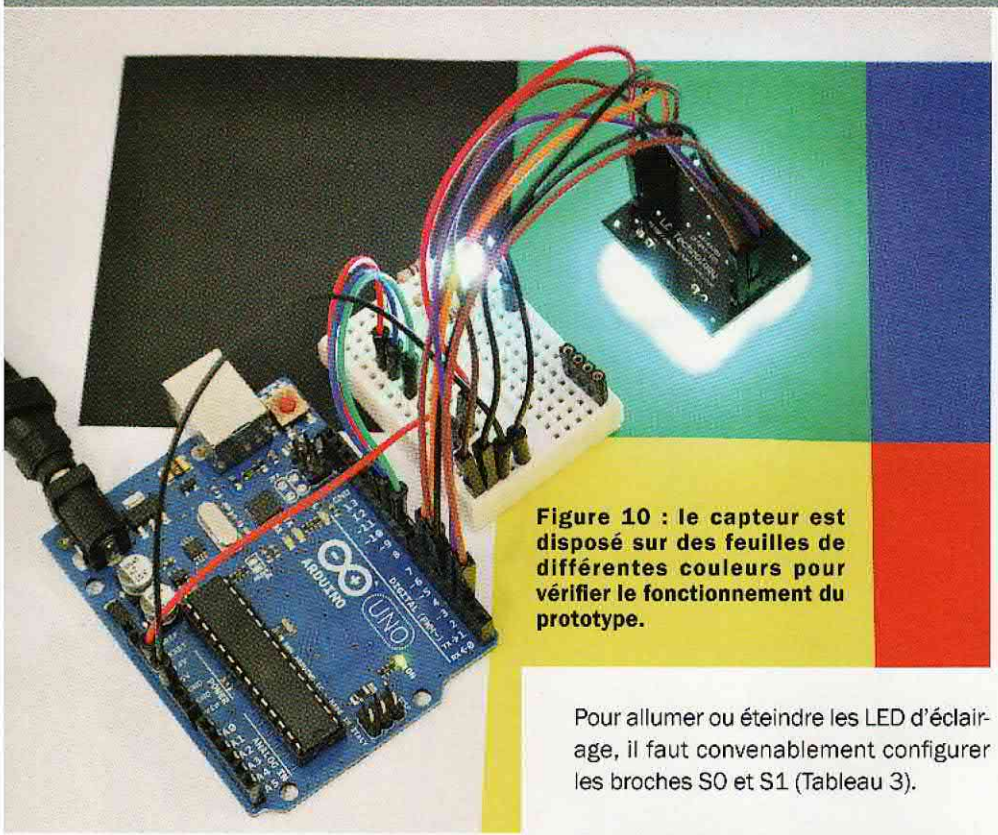


Figure 10 : le capteur est disposé sur des feuilles de différentes couleurs pour vérifier le fonctionnement du prototype.

Le sketch (programme) pour la carte Arduino Uno

Le **sketch** est visible dans le **Listing 1**, il permet de gérer la logique de la lampe, la reconnaissance de la couleur de la surface et la reproduction grâce à la LED RVB de la teinte de la couleur détectée.

Lorsque nous travaillons au **stade du prototypage** avec un outil simple comme Arduino, nous pouvons **introduire dans le code un ensemble d'instructions utiles pour le débogage.** Nous éliminerons ces instructions par la suite dans le processus de fabrication du produit final.

Dans notre cas, nous utilisons le **Moniteur Série** de l'**IDE Arduino** afin de mettre en évidence les valeurs des lectures et les résultats des calculs effectués.

La constante « **isPresentTolerance** » permet de définir une **valeur de seuil** pour vérifier si la lampe est posée ou non sur une surface.

Nous avons effectué une lecture avec les LED d'éclairage du capteur allumées et une autre lecture avec les LED éteintes.

Pour allumer ou éteindre les LED d'éclairage, il faut convenablement configurer les broches S0 et S1 (Tableau 3).

Dans le cas de la lampe reposant sur une surface, nous aurons deux valeurs très différentes pour ces 2 lectures, tandis que si la lampe est « suspendue », la lumière ambiante atteindra le capteur et donc les 2 lectures auront des valeurs proches.

Si la lampe repose sur une surface, nous mesurons l'intensité totale de la lumière réfléchie et l'intensité relative des différentes composantes de la couleur de la surface en appliquant le filtre approprié pour chaque lecture et en configurant correctement les broches S2 et S3 du capteur TCS230 (Tableau 2). L'intensité relative de chaque composante est calculée en fonction de l'intensité de la couleur globale sans aucun filtre.

Pour lire la fréquence de sortie correspondante à chaque application d'un filtre, nous utilisons l'instruction « **pulseIn** » qui lit la longueur d'une impulsion sur une broche donnée. L'instruction peut être configurée pour lire la durée d'une impulsion ayant un niveau bas (LOW) ou un niveau haut (HIGH).

Dans notre cas, nous avons un niveau bas (LOW). L'instruction attend que le niveau de la broche prenne un niveau bas (LOW) pour commencer la mesure du temps jusqu'à ce que le niveau de la broche prenne une valeur haute (HIGH). La mesure est alors interrompue.

Le résultat de la lecture est exprimé en microsecondes. Le format de l'instruction est le suivant :

pulseIn(pin, value, timeout)

« **pin** » correspond à la broche où la lecture de l'impulsion « **int** » s'effectue, « **value** » correspond au niveau de l'impulsion à lire haut (HIGH) ou bas (LOW). « **timeout** » (facultatif) correspond au temps en **microsecondes** pendant lequel l'impulsion doit être détectée. La valeur par défaut correspond à une seconde (unsigned long).

Si pendant le « timeout » aucune impulsion n'est détectée, l'instruction renvoie la valeur « 0 ». L'instruction « pulseIn » fonctionne pour des impulsions d'une durée comprise entre 10 microsecondes et 3 minutes, bien que pour de longues impulsions des erreurs de lectures peuvent se produire.

La logique de reconnaissance de la couleur nécessite une première lecture en l'absence de filtres, à condition que les broches S2 et S3 soient respectivement configurées à un niveau haut et bas comme indiqué dans le Tableau 2.

La valeur est lue sur la broche « **OUT** » et, dans l'ordre, les filtres correspondants sont appliqués pour les couleurs de base et les valeurs d'intensités relatives par rapport à la valeur d'intensité globale (Valeur de l'intensité totale/Valeur de l'intensité de la couleur) sont lues.

Pour pouvoir effectuer une comparaison nous sélectionnons l'intensité relative la plus grande, ensuite la couleur prédominante est calculée et la valeur correspondante est assignée au pilotage du PWM comme égale à 255.

Les valeurs de pilotage du PWM pour les autres couleurs sont calculées en fonction de la valeur lue pour la couleur en question, et la valeur de la couleur prédominante par rapport à 255.

Enfin, les valeurs PWM calculées sont utilisées pour piloter les broches respectives reliées aux bornes de la LED RVB.

Pour exécuter le sketch, ouvrez l'IDE Arduino puis recopiez le code dans l'éditeur de l'IDE.

Ensuite connectez la carte Arduino Uno au PC avec un câble USB, configurez correctement le port COM, compilez et téléchargez le sketch dans le microcontrôleur. Ouvrez le Moniteur Série de l'IDE, placez le capteur sur des objets de différentes couleurs ou des feuilles de papier de couleurs différentes et vérifiez le fonctionnement de l'ensemble (voir la figure 10).

Maintenant, nous allons voir si notre imprimante 3D a fini de réaliser le sapin de Noël qui constitue l'enveloppe de notre lampe. Assemblons l'ensemble et allons présenter notre prototype à des professionnels pour recueillir leurs commentaires et critiques.

Ingénierie du produit

Une fois que le prototype a été approuvé (validé), nous devons modifier les spécifications ainsi qu'adapter le design pour produire une version commerciale.

Nous ne pouvons pas garder le câblage en fils volants avec la carte Arduino Uno qui est de plus trop lourde et coûteuse pour un produit commercial. Nous devons donc faire des choix dans le but de satisfaire les exigences suivantes :

- remplacer la carte Arduino par un microcontrôleur moins cher et de dimensions réduites. En effet la carte Arduino Uno est surdimensionnée pour une utilisation en production, de plus un blindage serait nécessaire ce qui augmenterait les coûts du produit final avec en plus une solidité mécanique réduite ;
- fabriquer un circuit imprimé qui soit adapté à l'agencement des composants, à savoir le capteur disposé vers le bas et la LED RVB vers le centre de la lampe. Le circuit imprimé doit en plus se loger facilement à l'intérieur de la lampe afin de réduire les pertes de temps à la fabrication de masse ;
- protéger le programme du microcontrôleur en lecture afin qu'il ne soit pas facile à copier pour un usage commercial (contrefaçon) ;
- concevoir l'ensemble pour qu'il soit facilement assemblé, emballé et expédié.

Le corps de la lampe (qui correspond au boîtier du produit) doit faire l'objet d'une étude mécanique avec le minimum de pièces à fabriquer. Avec une imprimante 3D, il est possible de réaliser des prototypes et des petites séries, mais pour une production de masse il est préférable d'utiliser des moules avec des systèmes d'injection de matière plastique.

Dans ce cas, la production doit prévoir de très grands volumes afin d'amortir le coût de la construction des moules et l'outillage des machines de production.

Quant au microcontrôleur, nous aurions pu en choisir un de la famille Atmel qui soit compatible avec le code écrit pour le prototype, puis compiler et télécharger le programme dans le microcontrôleur avec une technique ne nécessitant pas de « bootloader ».

Au lieu de cela, nous avons préféré une solution « minimaliste », mais qui est très souvent utilisée dans ce type de projets. Nous utilisons un microcontrôleur de chez Microchip, un **PIC16F1827** qui coute environ 1,5 € à l'unité.

Cependant nous devons réécrire le programme pour qu'il soit compatible avec le PIC, mais ce type de microcontrôleur peut se programmer en **BASIC** avec une grande facilité d'adaptation.

Les raisons pour lesquelles nous avons choisi un produit de chez Microchip et particulièrement ce microcontrôleur, sont les suivantes :

- faible coût (moins de 1 € par 100 pièces) ;
- il est disponible en boîtier DIP ou SMD (CMS) :
- il est équipé d'une horloge interne qui nous permet d'éviter l'utilisation d'un quartz et des composants externes supplémentaires, cela permet encore de réduire les coûts de fabrication ;
- faible consommation électrique ;
- il dispose de 15 broches d'entrées/sorties capables de piloter directement chacune une LED ;
- il dispose de 4 sorties PWM, ce qui est suffisant pour notre application ;
- il est compatible broche à broche avec des microcontrôleurs de la même famille ayant des fonctionnalités supplémentaires, ce qui permet de faire évoluer le produit dans le temps (par exemple rajouter une synthèse vocale indiquant la couleur de la surface sur laquelle la lampe est posée).

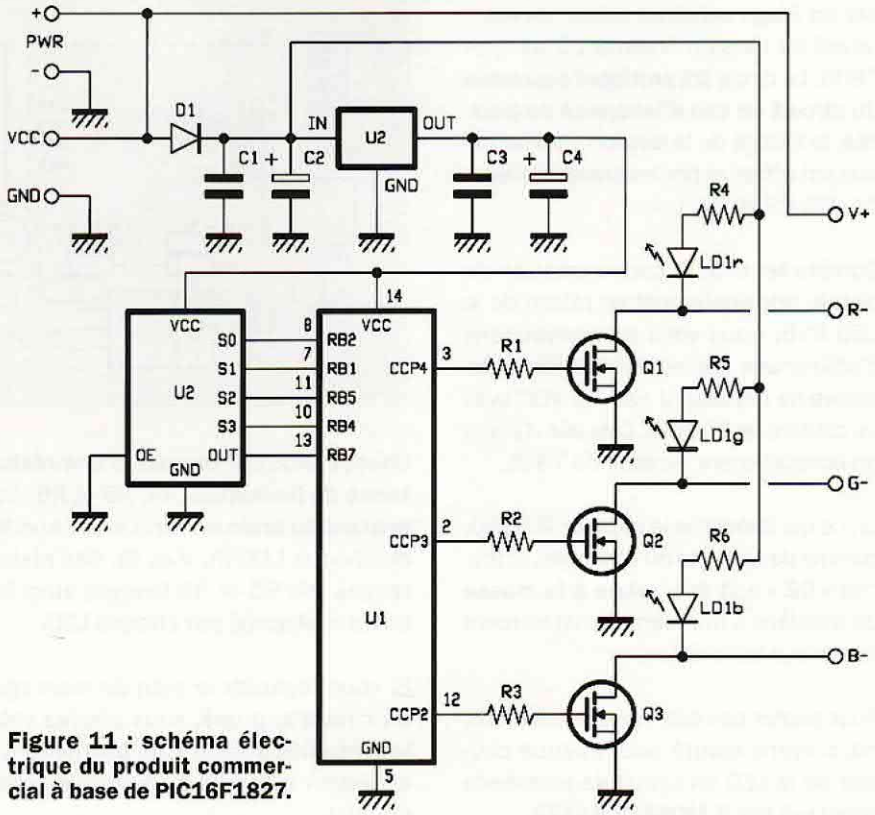


Figure 11 : schéma électrique du produit commercial à base de PIC16F1827.

Comme autre ingénierie du produit, nous avons remplacé la LED RVB par un modèle plus puissant et avec une meilleure gestion de la lumière produite. Cela a nécessité l'ajout pour chaque couleur d'un circuit d'alimentation à base de **transistors MOSFET**. Le résultat final est visible sur le schéma électrique en figure 12. Notez que la **LED RVB** est composée par les diodes **LD1r**, **LD1g**, et **LD1b**.

Vous pouvez voir, en se référant au **code source en BASIC** visible dans le **Listing 2**, les affectations des broches du microcontrôleur PIC16F1827. Sur le schéma électrique est indiqué à la fois le nom de la fonction de la broche et son numéro.

Comme vous pouvez le voir dans le Listing 2, les broches **RB1**, **RB2**, **RB4** et **RB5** sont **configurées en sorties et gèrent les entrées** du capteur **TCS230**.

La broche **RB7** est configurée en **entrée** et permet de **lire la durée d'impulsion** résultant de la mesure de l'intensité de la lumière. Pour les **sorties PWM**, nous avons utilisé les broches **CCP2**, **CCP3** et **CCP4** qui pilotent à travers les MOSFET les 3 LED LD1r, LD1g, et LD1b.

L'alimentation pour le microcontrôleur et pour le reste du circuit est fournie par un étage construit autour du régulateur de tension linéaire U2 de type 7805. La diode **D1 protège l'ensemble du circuit en cas d'inversion de polarité**, le filtrage de la tension d'alimentation est effectué par les condensateurs C1, C2, C3 et C4.

Compte tenu de la consommation du circuit, principalement en raison de la LED RVB, nous vous recommandons d'utiliser une alimentation externe stabilisée ne dépassant pas 7,5 VDC avec un courant de 500 mA. Cela afin d'éviter un échauffement excessif du 7805.

En ce qui concerne le capteur TCS230, comme dans le cas du prototype, la broche « **OE** » doit être **reliée à la masse** de manière à maintenir constamment le capteur en fonction.

Pour piloter correctement la LED RVB, nous avons ajouté pour chaque couleur de la LED un circuit de puissance constitué par **3 MOSFET BS170**.

Plan de montage

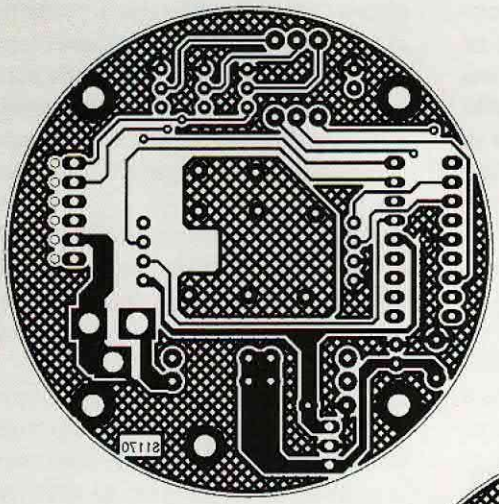


Figure A : circuit imprimé de la lampe à base de PIC16F1827 à l'échelle 1 : 1 côté soudures.

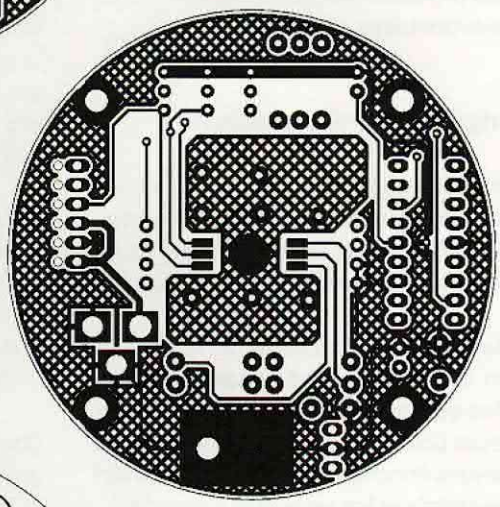


Figure B : circuit imprimé de la lampe à base de PIC16F1827 à l'échelle 1 : 1 côté composants.

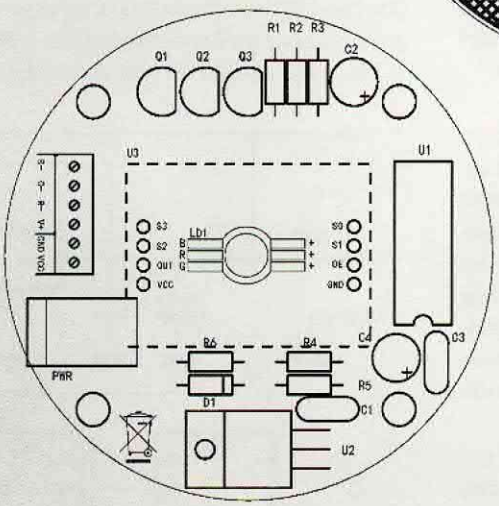


Figure C : Implantation des composants de la lampe à base de PIC16F1827.

Chaque MOSFET dispose d'une **résistance de limitation** (R4, R5 et R6) du **courant du drain** en série avec l'anode de chaque LED (R, V et B). Ces résistances R4, R5 et R6 limitent ainsi le courant absorbé par chaque LED.

Si vous regardez le plan de montage du circuit imprimé, vous pouvez voir les modifications qui permettent d'adapter la lampe à un produit commercial.

Nous avons prévu de monter le capteur en dessous de la lampe, face à la surface où la couleur doit être détectée.

Les autres composants sont montés vers le milieu, en particulier la LED RVB qui se trouve en position centrale de façon à répartir la lumière uniformément. Les circuits imprimés ainsi que la disposition des composants sont visibles dans l'encadré intitulé « **Plan de montage** ».

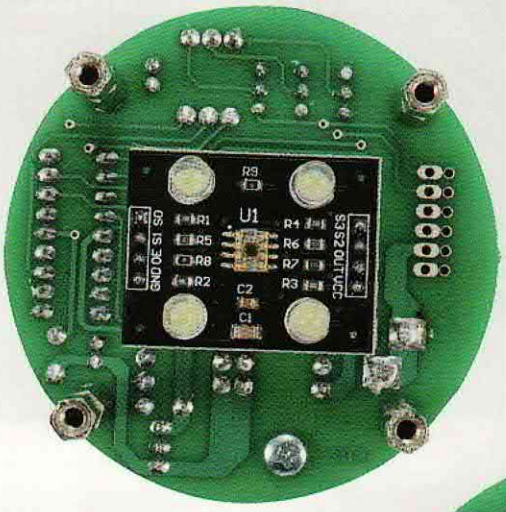


Figure E : photo de l'un de nos prototypes de la lampe à base de PIC16F1827 côté soudures. Vous pouvez voir le capteur TCS230 orienté vers la surface où la lampe repose.

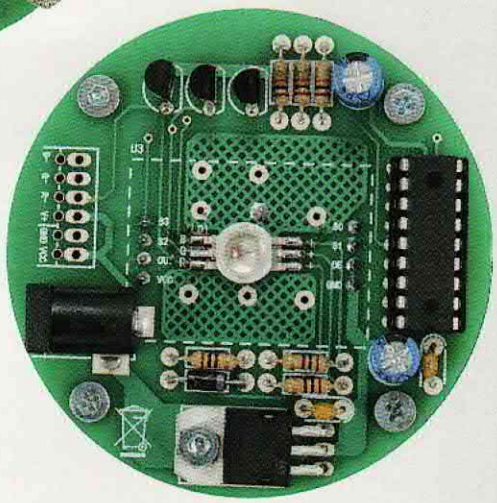


Figure D : photo de l'un de nos prototypes de la lampe à base de PIC16F1827 côté composants. Au centre se trouve la LED RVB.

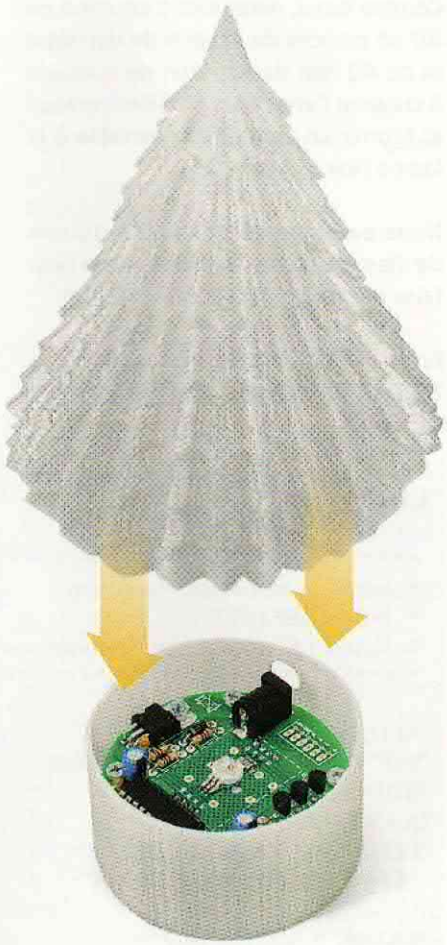


Figure 12 : la partie supérieure de la lampe est dissociée de la base.

Liste des composants de la lampe à base de PIC16F1827

R1.....1 kΩ
R2.....1 kΩ
R3.....1 kΩ
R4.....100 Ω
R5 .....100 Ω
R6 .....100 Ω
C1.....100 nF multicouche
C2.....100 µF/25 V électrolytique
C3.....100 nF multicouche
C4.....100 µF/25 V électrolytique
U1.....PIC16F1827-I/P
U2.....7805
U3.....TCS230
D1.....1N4007
Q1.....BS170
Q2.....BS170
Q3.....BS170
LD1....LEDRGB1W

Divers

Prise alimentation pour ci
Support 2 x 9 broches
Barrette femelle 4 pôles (x2)
Vis 3MA 6 mm (x4)
Entretoise F/F 23mm (x4)

Nous avons soudé sur la carte du TCS230 des barrettes femelles qui permettent de le relier (et de le maintenir) au circuit imprimé principal (où se trouve le PIC16F1827).

Il est évident que dans le cas d'une production de masse, le capteur serait soudé directement sur le circuit imprimé principal. Sur le circuit imprimé, 4 entretoises permettent de fixer l'ensemble de l'électronique sur la partie inférieure du boîtier de la lampe. Nous allons maintenant étudier le programme du PIC16F1827 qui permet de gérer la lampe.

Pour le programme du PIC, nous avons choisi d'utiliser le compilateur **PICBASIC PRO**.

Il s'agit d'un environnement de développement en BASIC dédié aux microcontrôleurs Microchip.

Vous pouvez télécharger gratuitement une version d'évaluation sur le site **http://melabs.com**.

Pour la programmation, nous utilisons l'outil de développement **MPLAB ICD 3**, cependant tout autre programmateur peut être utilisé à condition de configurer correctement les bits de programmation.

Au Listing 2, vous pouvez voir le programme en BASIC qui est conforme à la logique du diagramme de traitement du synoptique de la figure 7.

Les commentaires et la similitude avec le sketch Arduino ne laissent aucun doute sur l'interprétation.

Afin de rendre modulaire l'utilisation de la lampe, nous avons dissocié la partie supérieure de la base de la lampe. Vous pouvez donc changer facilement la forme de la partie supérieure de la

lampe par d'autres formes de boîtiers. Comme base, nous avons imprimé en 3D un cylindre de 70 mm de diamètre et de 40 mm de hauteur, de manière à contenir l'ensemble de l'électronique et fournir un support convenable à la lampe (voir la figure 12).

Nous avons percé un trou de 10 mm de diamètre sur le côté de la base pour faire passer la prise d'alimentation.

En figure 13, vous pouvez voir le résultat final de notre projet « un sapin de Noël caméléon » qui s'allume de la même couleur que la surface sur laquelle il repose.

Essayez de placer la lampe sur diverses surfaces ayant des couleurs différentes, vous verrez que la lampe s'adapte à son environnement comme un caméléon.

Nous vous souhaitons un bon amusement en cette période de Fêtes.

Figure 13 : l'arbre de Noël caméléon en version « commerciale ».

Listing 2

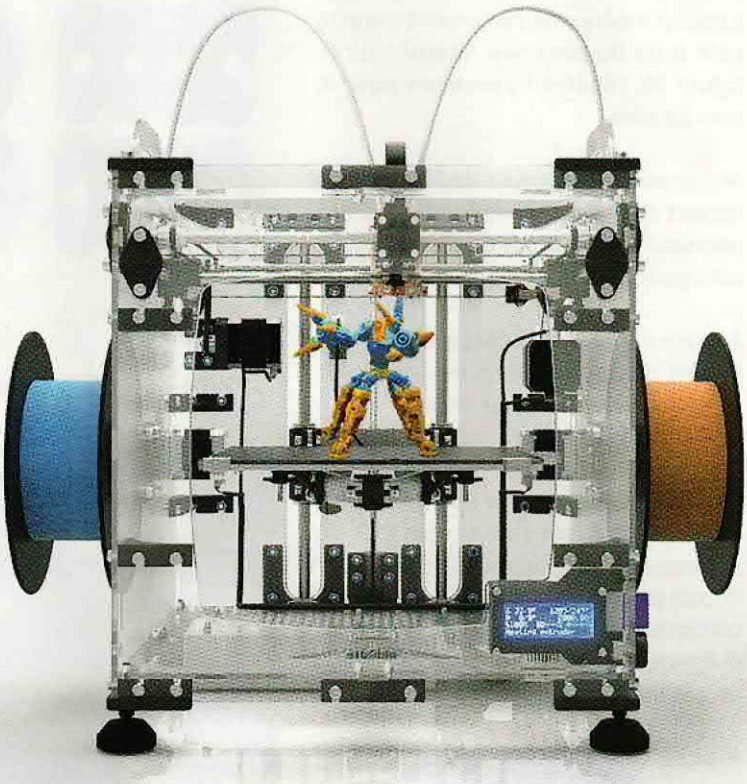
```
'*****************************************************************************
'* Nom   : Lampe cameleon.pbp                                                *
'* PIC   : 16F1827                                                           *
'* Notes : Firmware Lampe cameleon     Reproduit la couleur de la surface sur laquelle elle repose  *
'*'*************************************************************************** *

INCLUDE "modedefs.bas"
'@DEVICE INTRC_OSC_NOCLKOUT
'@DEVICE MCLR_OFF
DEFINE OSC 16
'DEFINE PULSIN_MAX 1350
'DEFINE PULSIN_MAX 1750

ADCON0 = %00000000
OPTION_REG.7=0
'CMCON0 =%00000111
ANSELA=%00000000
ANSELB=%00000000
'OSCCON=%01101010
OSCCON=%01111010
APFCON0.3=0                    'CCP2 sur broche RB6
'MDSRC=%00000011
'WPU=%00010000

'********************** configuration des ports du PIC ******************************

SYMBOL TX = PORTB.3            'broche 9 données série en sortie
SYMBOL S0 = PORTB.2            'broche 8
SYMBOL S1 = PORTB.1            'broche 7
SYMBOL S2 = PORTB.5            'broche 11
SYMBOL S3 = PORTB.4            'broche 10
SYMBOL OUTC = PORTB.7          'broche 13

Coeff VAR WORD

Ro          VAR WORD
Go          VAR WORD
Bo          VAR WORD

white       VAR WORD
white_off   VAR WORD
red         VAR WORD
blu         VAR WORD
green       VAR WORD
```

```
Tolerance       VAR WORD
mode            VAR BYTE

                'l'instruction Hpwm génère une modulation de largeur d'impulsion en utilisant les sorties PWM du PIC

DEFINE CCP3_REG PORTA                   'Hpwm 1 pin port
DEFINE CCP3_BIT 3                       'Hpwm 1 pin bit
DEFINE CCP4_REG PORTA                   'Hpwm 1 pin port
DEFINE CCP4_BIT 4                       'Hpwm 1 pin bit
DEFINE CCP2_REG PORTB                   'Hpwm 1 pin port
DEFINE CCP2_BIT 6                       'Hpwm 1 pin bit
INPUT OUTC
OUTPUT S0
OUTPUT S1
OUTPUT S2
OUTPUT S3

'********************* Déclaration des variables ***********************
CONTA VAR WORD                          'contient la longueur d'impulsion
ERRORE VAR BYTE                         '0: Pas d'erreur dans le signal - 1: signal non valide

Ro = 0
Go = 0
Bo = 0
CONTA=0
ERRORE=0

'******************* Initialisation du programme ************************
INIZIO:
CLEAR
SEROUT tx,T9600,[13,10,"SYSTEM STARTUP v1.0",13,10]

''********************** M A I N ***********************************
'Lecture de la longueur de l'impulsion"0" et successivement lecture des 3 bits reçus

MAIN:
  mode = 3
  GOSUB TAOS_MODE
CONTINUA:
                ' Lit le blanc
  LOW S3
  HIGH S2
  GOSUB LEGGI_COLORE
  SEROUT tx,T9600,[13,10,"W:",#CONTA," (",#S2,"-",#S3,")",13,10]
  white = CONTA  * 100
  SEROUT tx,T9600,[13,10,"Wx:",#white," (",#S2,"-",#S3,")",13,10]

                ' Lit le rouge
  LOW S3
  LOW S2
  GOSUB LEGGI_COLORE
  SEROUT tx,T9600,[13,10,"R:",#CONTA," (",#S2,"-",#S3,")",13,10]
  red = white / CONTA
  SEROUT tx,T9600,[13,10,"Rx:",#red," (",#S2,"-",#S3,")",13,10]

                ' Lit le vert
  HIGH S3
  HIGH S2
  GOSUB LEGGI_COLORE
  SEROUT tx,T9600,[13,10,"G:",#CONTA," (",#S2,"-",#S3,")",13,10]
  green = white / CONTA
  SEROUT tx,T9600,[13,10,"Gx:",#green," (",#S2,"-",#S3,")",13,10]

                ' Lit le bleu
  HIGH S3
```

```
  LOW S2
  GOSUB LEGGI_COLORE
  SEROUT tx,T9600,[13,10,"B:",#CONTA," (",#S2,"-",#S3,")",13,10]
  blu = white / CONTA
  SEROUT tx,T9600,[13,10,"Bx:",#blu," (",#S2,"-",#S3,")",13,10]
  IF red >= blu AND red >= green THEN
    Coeff = 25500 / red
    Ro = 255
    Go = green * Coeff / 100
    Bo = blu * Coeff / 100
    Bo=Bo * 3 / 4
    Go=Go * 3 / 4
  ENDIF

  IF blu >= red AND blu >= green THEN
    Coeff = 25500 / blu
    Bo = 255
    Go = green * Coeff / 100
    Ro = red * Coeff / 100
    Ro=Ro * 3 / 4
    Go=Go * 3 / 4
  ENDIF

  IF green >= blu AND green >= red THEN
    Coeff = 25500 / green
    Go = 255
    Ro = red * Coeff / 100
    Bo = blu * Coeff / 100
    Bo=Bo * 3 / 4
    Ro=Ro * 3 / 4
  ENDIF

  HPWM 2,b0,100
  HPWM 3,g0,100
  HPWM 4,r0,100

  SEROUT tx,T9600,[13,10,"RGB:",#Ro,"-",#Go,"-",#Bo,13,10]

  pause 500
GOTO CONTINUA

TAOS_MODE:
   SELECT CASE mode
      CASE  0
         LOW S0
         LOW S1
      CASE  1
         HIGH S0
         HIGH S1
      CASE  2
         HIGH S0
         LOW S1
      CASE  3
         LOW S0
         HIGH S1
   END SELECT
   pause 50
RETURN

LEGGI_COLORE:
  pause 50
  PULSIN OUTC,0,CONTA
RETURN
```

3DVERTEX - 3

Dans le précédent numéro 136 d'Electronique et Loisirs Magazine, nous avons abordé la construction mécanique de la 3DVERTEX. Nous nous sommes arrêtés à l'assemblage de la tête d'impression (extrudeuse) qui, rappelons-le, est une étape longue et minutieuse. Nous allons maintenant, dans cette troisième partie, aborder le montage de la tête d'impression qui demande un travail très soigneux. Entrons dans le vif du sujet !



3DVERTEX

L'IMPRIMANTE 3D BICOLORE

Troisième partie : La mécanique (suite)

de Boris LANDONI

Montage de la tête d'impression

À la fin du précédent article, nous avons terminé par l'assemblage de l'extrudeuse, maintenant il faut la monter sur la 3DVERTEX. La première des choses à faire est de disposer sur le plan de travail les éléments mécaniques dont nous avons besoin pour cette étape.

Le système de déplacement de la tête est constitué par deux tiges filetées (barres) orthogonales entraînées par des courroies au moyen d'engrenages. Cela permet le déplacement simultané de la tête dans les deux directions, et d'assurer un support mécanique de bonne qualité. Nous aurons besoin des pièces suivantes, aidez-vous de la figure 1 pour les visualiser :

- **8 fixations de roulement** (axial bearing clamp) ;
- **8 couvercles de roulements** (end cap) ;
- **8 roulements de Ø 8 mm** (sealed bearing);
- **16 écrous M5** ;
- **16 boulons M5 x 12 mm**.

Commencez par insérer un roulement de Ø 8 mm dans le coin droit du panneau frontal de l'imprimante (voir la figure 2).

Faites de même pour le coin gauche. Ensuite insérez un roulement dans le coin droit du panneau latéral (voir la figure 3), répétez l'opération pour le coin gauche.

Maintenant vous devez insérer un roulement de Ø 8 mm dans le coin droit du panneau arrière, faites de même pour le coin gauche (voir la figure 4).

Ensuite continuez en disposant un roulement de Ø 8 mm dans les coins droit et gauche du panneau latéral (voir la figure 5).

Maintenant vous devez insérer un roulement de Ø 8 mm dans le coin droit du panneau arrière, faites de même pour le coin gauche (voir la figure 4). Ensuite continuez en disposant un roulement de Ø 8 mm dans les coins droit et gauche du panneau latéral (voir la figure 5).

Placez une fixation (axial bearing clamp) sur le roulement et positionnez un écrou M5 dans le trou supérieur de la fixation (voir la figure 6).

Placez un couvercle de roulement (end cap) de l'autre côté et utilisez un boulon M5 x 12 mm pour serrer l'ensemble.

Disposez le couvercle de roulement de sorte que le centre du roulement ne soit pas obstrué. Le couvercle de roulement sera fermé ultérieurement (voir la figure 7).

Répétez le même processus pour le coin gauche du panneau frontal. Notez que le boulon M5 x 12 mm doit se situer dans le trou supérieur du couvercle.

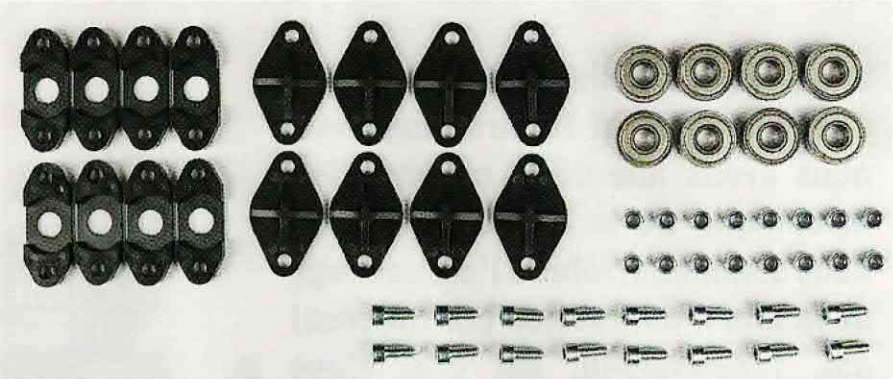


Figure 1 : les différentes pièces nécessaires pour le montage de l'extrudeuse.

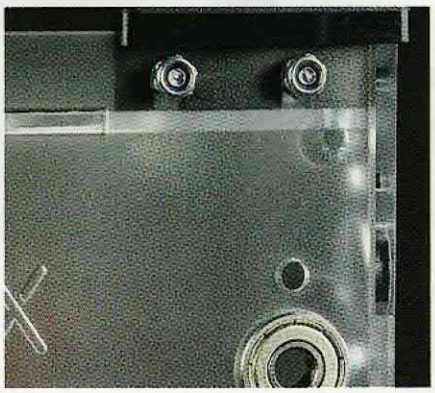


Figure 2 : insertion d'un roulement dans le coin droit du panneau frontal.

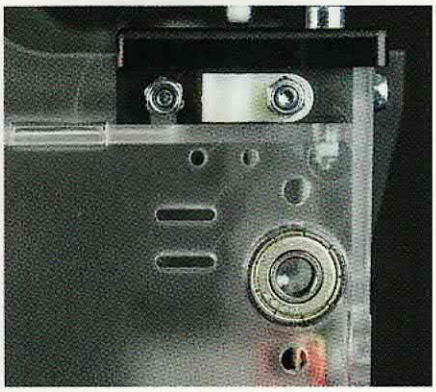


Figure 3 : insertion d'un roulement dans le coin droit du panneau latéral.

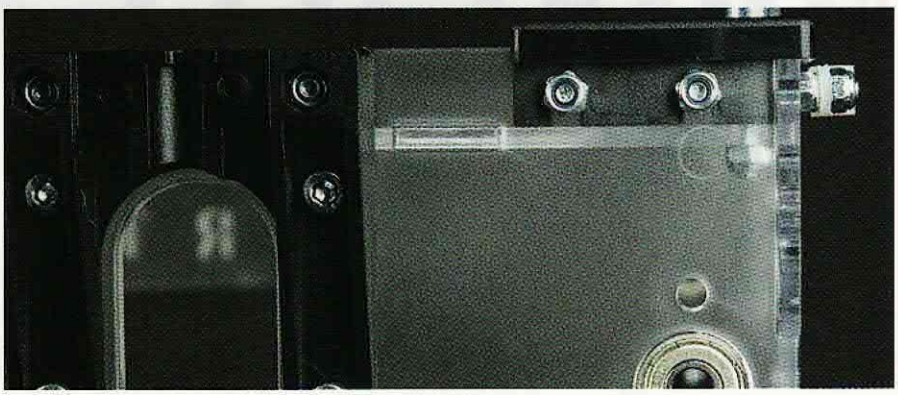


Figure 4 : insertion d'un roulement dans le coin droit du panneau arrière.

Figure 5 : insertion d'un roulement dans le coin droit du panneau latéral.

Figure 6 : positionnement des fixations des roulements.

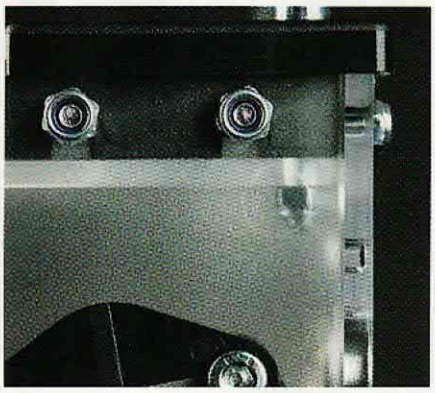


Figure 7 : disposition temporaire des couvercles des roulements.

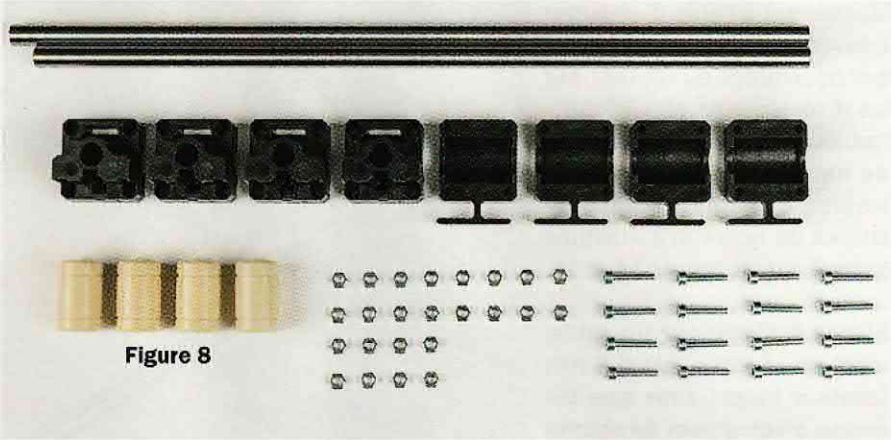


Figure 8

Répétez le même processus pour les coins gauche et droit du panneau latéral. Notez cependant que les boulons M5 x 12 mm doivent se situer dans les trous inférieurs des couvercles.

Répétez le même processus pour les coins gauche et droit du panneau arrière. Les boulons M5 x 12 mm doivent se situer dans les trous supérieurs des couvercles.

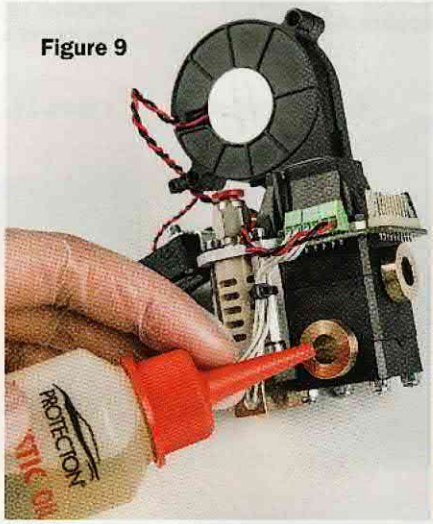


Figure 9

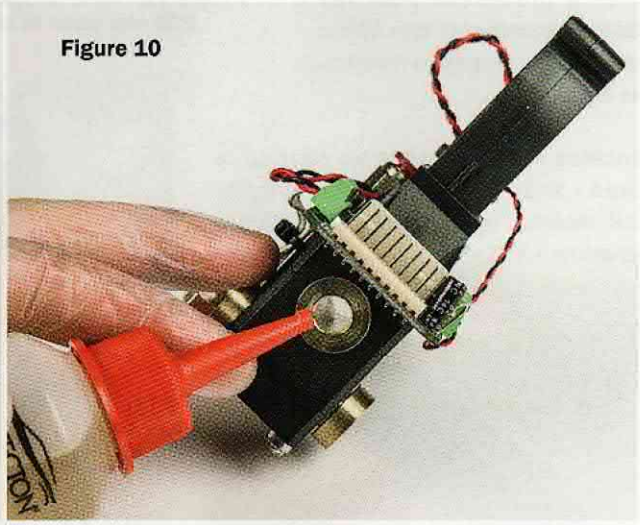


Figure 10

Répétez le même processus pour les coins gauche et droit de l'autre panneau latéral. Les boulons M5 x 12 mm doivent se situer dans les trous inférieurs des couvercles.

Normalement il doit vous rester des boulons et des écrous, mettez-les de côté et munissez-vous des pièces suivantes (reportez-vous à la figure 8) :

- **1 tige** de longueur **298 mm** et de Ø 8 mm ;
- **1 tige** de longueur **289 mm** et de Ø 8 mm ;
- **4 roulements** en « iglidur » de Ø 8 mm (couleur beige) ;
- **24 écrous M3** ;
- **16 boulons M3 x 16mm** ;
- **4 chariots** de type « **XY1** » ;
- **4 chariots** de type « **XY2** ».

Nous vous recommandons d'utiliser des gants pour la manipulation des tiges (barres), car elles peuvent **s'oxyder lorsqu'elles entrent en contact avec les mains**.

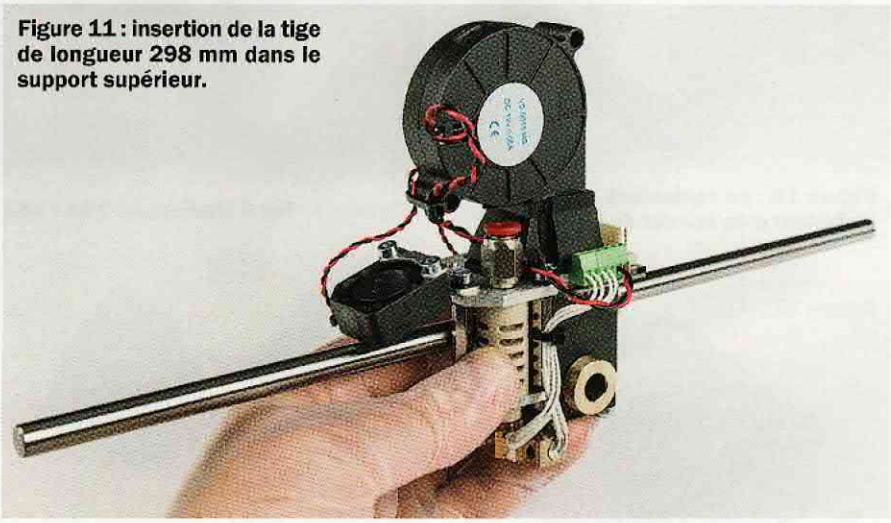


Figure 11 : insertion de la tige de longueur 298 mm dans le support supérieur.

Appliquez un peu d'huile sur les supports des tiges de la tête d'impression, aidez-vous des figures 9 et 10.

Ensuite insérez la tige de longueur 298 mm dans le support supérieur et vérifiez que la tête d'impression se déplace facilement le long de la tige (voir la figure 11).

Maintenant insérez la tige de longueur 289 mm dans le support inférieur et vérifiez que la tête d'impression se déplace sans frottements le long de la tige (voir la figure 12). Vous devez obtenir un ensemble identique à la figure 13. Faites attention de ne pas inverser les tiges, car elles ont des longueurs différentes.

Maintenant à l'aide d'un cutter enlevez les éventuelles bavures qui pourraient se trouver sur les 4 roulements en « iglidur » (couleur beige), **faites attention de ne pas les endommager**. Insérez 2 écrous M3 dans un chariot de type « XY2 » comme indiqué sur les figures 14 et 15.

Placez à l'intérieur un roulement en « iglidur » de Ø 8 mm (couleur beige) **sans que les écrous s'échappent du chariot** (voir la figure 16). Répétez le même processus pour les 3 autres roulements en « iglidur », vous devez obtenir le résultat de la figure 17.

Insérez 4 écrous dans un chariot de type « XY1 » comme indiqué en figure 18, répétez l'opération pour les 3 autres chariots « XY1 ».

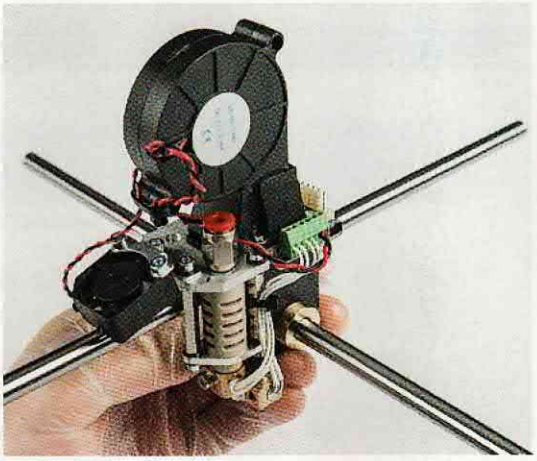


Figure 12 : insertion de la tige de longueur 289 mm dans le support inférieur.

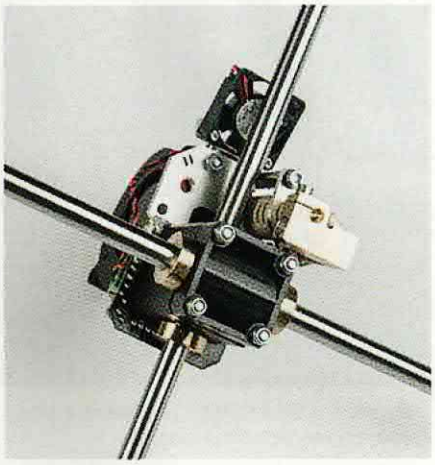


Figure 13 : ici l'ensemble tête d'impression avec les 2 tiges, faites attention de ne pas inverser les tiges.

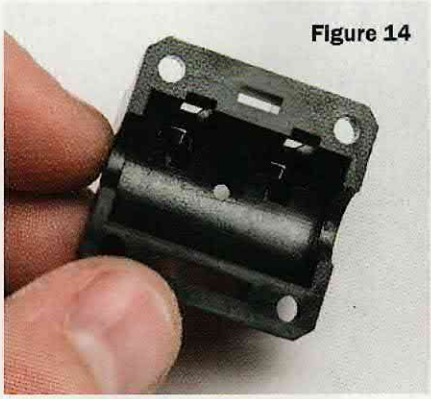


Figure 14

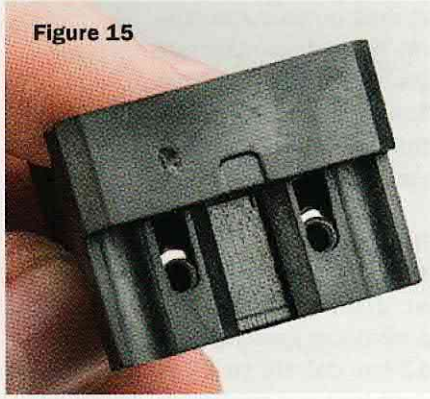


Figure 15

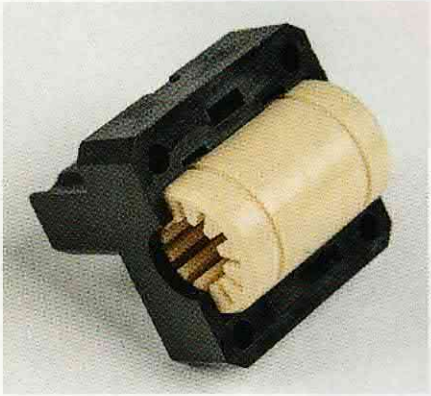


Figure 16 : un roulement « iglidur » à l'intérieur d'un chariot de type « XY2 ».

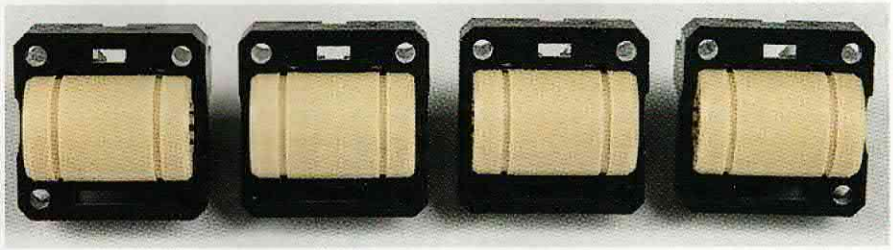
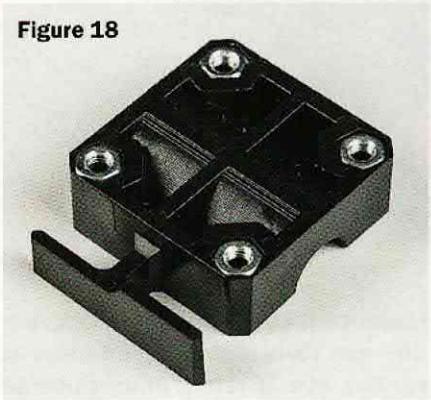


Figure 17 : les 4 chariots de type « XY2 » avec un roulement « iglidur » à l'intérieur.

Figure 18



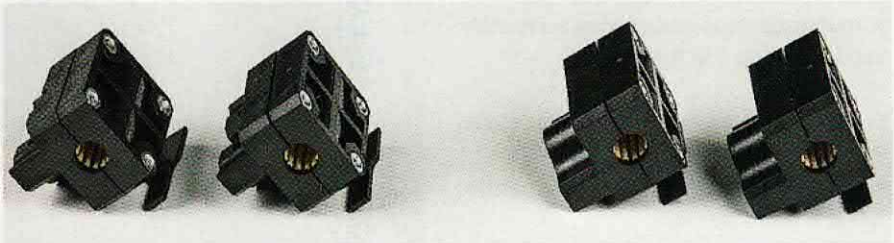


Figure 19a : ici les 2 ensembles avec 2 chariots retournés et 2 chariots non retournés.

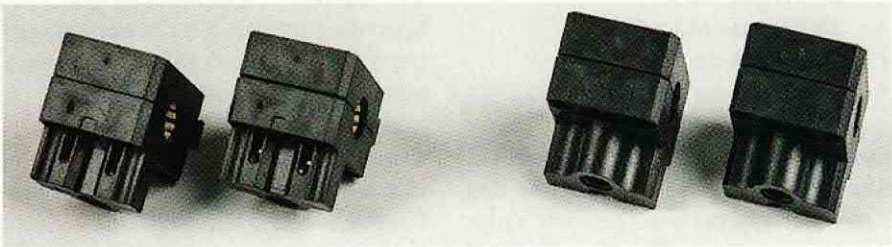


Figure 19b : vue sous un autre angle de la figure 19a.

Montez les chariots de type « XY2 » sur les chariots « XY1 », vérifiez en vous aidant des figures 19a et 19b que vous avez effectué le montage de 2 manières différentes.

Vous devez avoir 2 ensembles de pièces, un montage avec les chariots retournés (les uns par rapport aux autres) et un autre montage avec les chariots non retournés (montage normal).

Utilisez des boulons M3 x 16 mm pour fixer les ensembles constitués par les chariots (voir la figure 20).

Prenez un ensemble chariot et assurez-vous qu'il corresponde à celui de la figure 21.

Coupez la partie gauche de la patte du chariot comme illustré en figure 22, vous devez obtenir le résultat de la figure 23.

Attention avant de couper, vérifiez plusieurs fois qu'il s'agit de la bonne pièce en vous aidant des photos.

Vous devez avoir un ensemble composé de 4 chariots identiques à ceux représentés en figure 24.

Prenez la pièce dont vous venez de couper la patte et faites-la glisser sur la tige de longueur 298 mm et Ø 8 mm.

Vérifiez que vous la faites glisser sur la bonne extrémité (voir la figure 25).

Prenez l'autre pièce appartenant à l'ensemble chariot précédent, c'est-à-dire la pièce à côté de celle dont la patte est coupée en figure 24, et insérez-la à l'autre extrémité de la tige de longueur 298 mm et Ø 8 mm (voir la figure 26).

Vous devez obtenir un montage identique à celui de la figure 27.

À l'aide d'un mètre à ruban, mesurez la distance entre les 2 chariots que vous venez de monter sur la barre de 298 mm.

La distance entre les 2 chariots de bord à bord externes doit être de 33,6 cm (voir les figures 28a et 28b). Eventuellement déplacez les chariots le long de la tige pour obtenir la distance adéquate.

De la même manière, insérez les 2 autres chariots sur la tige de 289 mm. La distance entre les 2 chariots de bord à bord externes doit être de 32,8 cm (voir les figures 29a et 29b), éventuellement ajustez la distance entre les chariots pour qu'elle soit correcte.

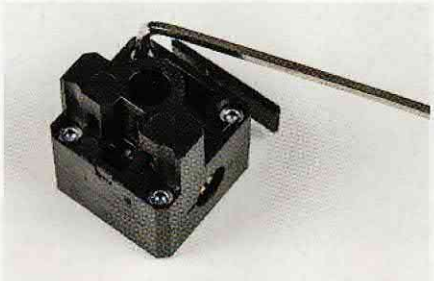


Figure 20 : fixation des chariots à l'aide de boulons M3 x 16 mm.

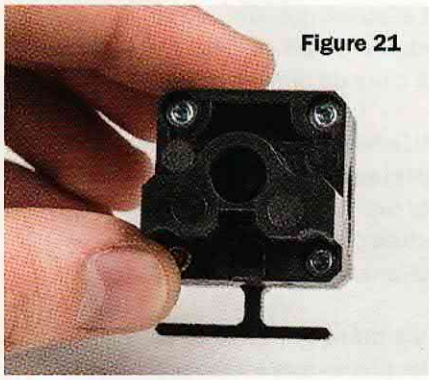


Figure 21

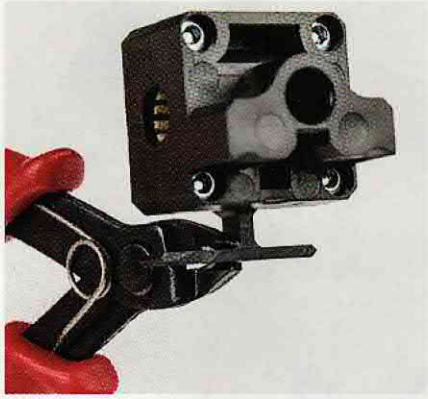


Figure 22 : la partie gauche de la patte du chariot doit être coupée comme illustré sur l'image.

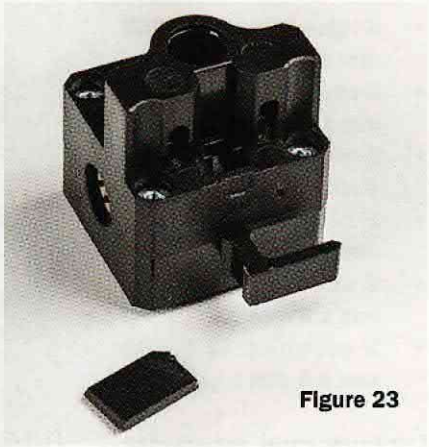


Figure 23

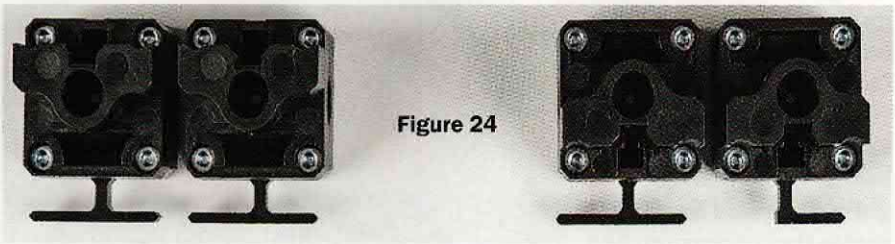


Figure 24

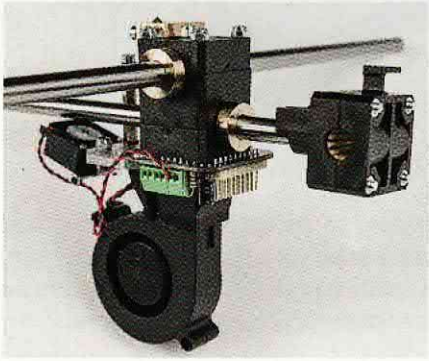


Figure 25 : insertion du chariot avec la patte coupée sur la tige de longueur 298 mm.

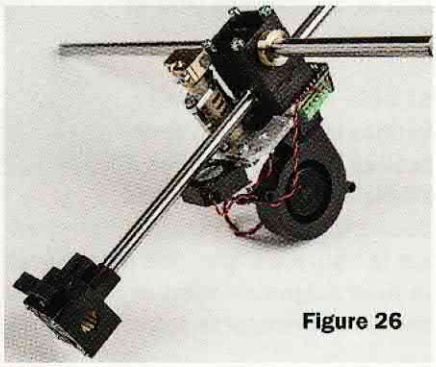


Figure 26

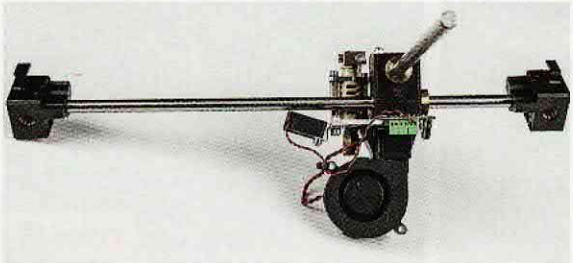


Figure 27 : les 2 chariots (dont l'un avec la patte coupée) insérés de chaque côté dans la tige de 298 mm.

Le montage final des 4 chariots doit correspondre à celui de la figure 30.

Vérifiez soigneusement les distances car la moindre erreur impliquerait beaucoup de travail à effectuer ultérieurement.

Maintenant procurez-vous les pièces suivantes visibles sur la figure 31 :

- **4 tiges** de longueur **350 mm** et de Ø 8 mm ;
- **8 poulies T2,5/19T** Ø 8 mm ;
- **8 vis de réglage M3 x 4 mm** ;
- **2** courroies **T2,5 180 mm x 6 mm** ;
- **2** courroies **T2,5 670 mm x 6 mm** ;
- **2** courroies **T2,5 685 mm x 6 mm** ;

Lorsque vous manipulez les tiges, munissez-vous de gants car les mains nues peuvent oxyder leur surface.

Vérifiez que toutes les poulies de type « T2,5/19T » de Ø 8 mm **glissent facilement sur les tiges** (barres) de 350 mm et de Ø 8 mm (voir la figure 32).

Si cela n'est pas le cas, **vérifiez la présence d'éventuelles bavures laissées à l'intérieur** des poulies. Vous pouvez utiliser une petite lime pour aléser le trou, mais attention manœuvrez la lime soigneusement et lentement sans endommager la poulie.

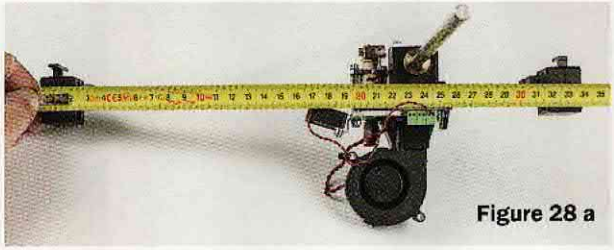


Figure 28 a

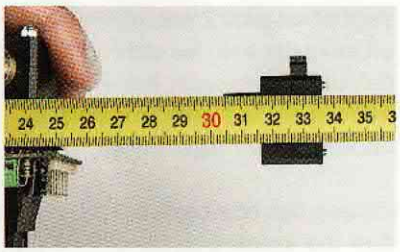


Figure 28b : la distance entre les 2 chariots de bord à bord externes sur la barre de 298 mm doit être de 33,6 cm.

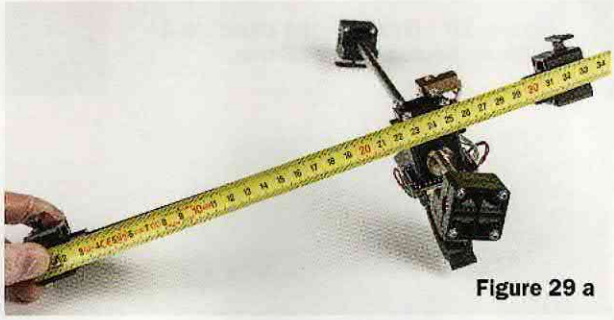


Figure 29 a

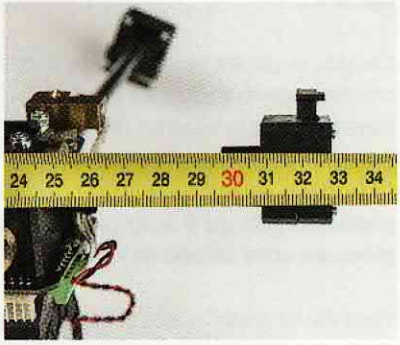


Figure 29b : la distance entre les 2 chariots de bord à bord externes sur la barre de 289 mm doit être de 32,8 cm.

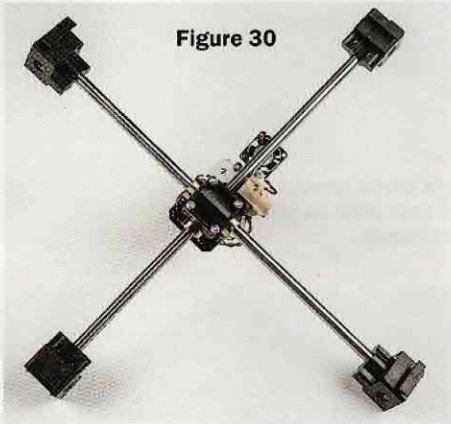


Figure 30

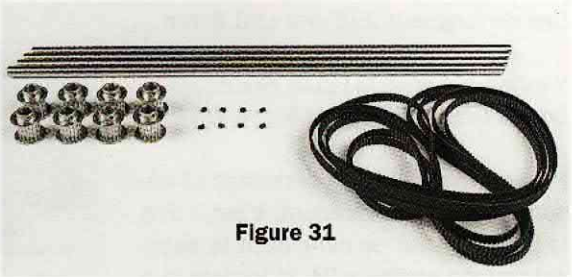


Figure 31

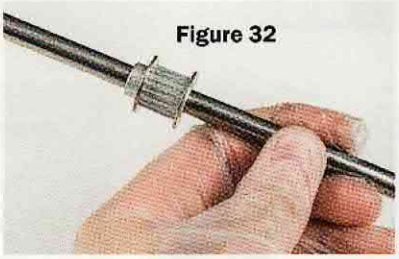


Figure 32

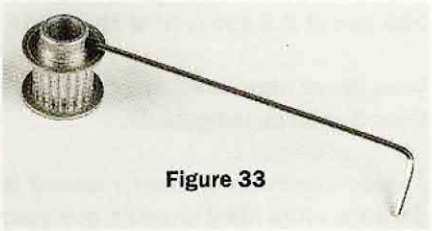


Figure 33

Vissez la vis de réglage M3 x 4 mm dans la poulie « T2,5/19T » de Ø 8 mm, vérifiez que la poulie peut toujours se déplacer facilement sur la barre.

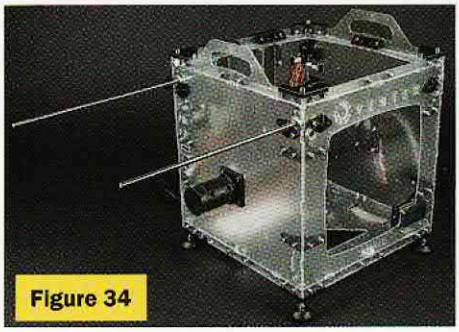


Figure 34

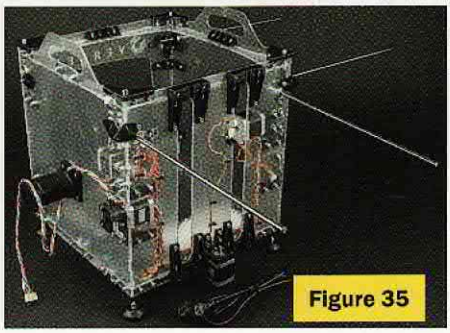


Figure 35

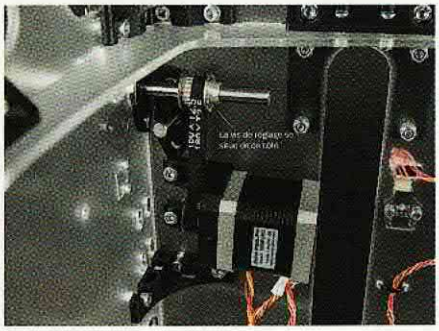


Figure 36 : la vis de réglage de la poulie doit se trouver vers le moteur, comme indiqué sur la photo.

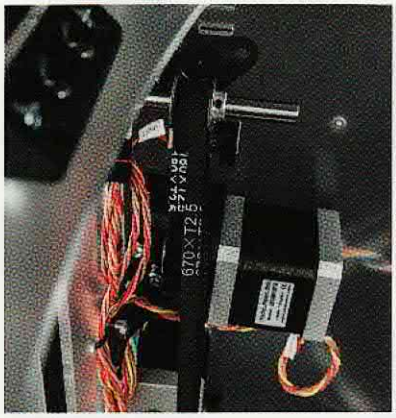


Figure 37 : la courroie de 670 mm se situe à droite de la courroie de 180 mm.

Figure 38 : la courroie de 685mm se positionne à droite de la courroie de 180 mm.

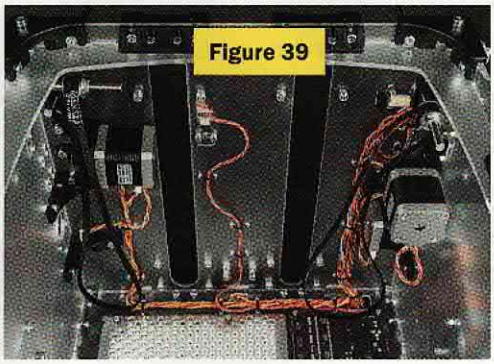


Figure 39

Figure 40

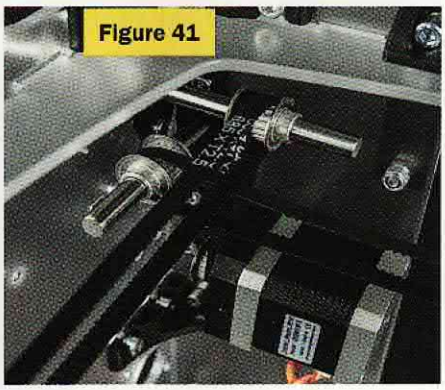


Figure 41

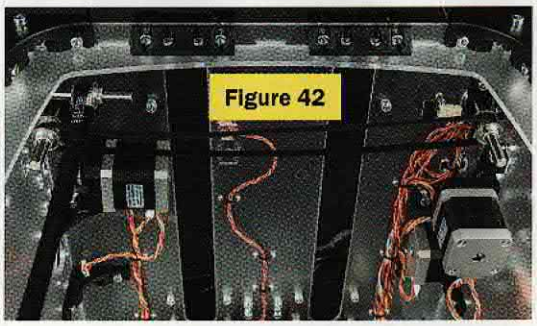


Figure 42

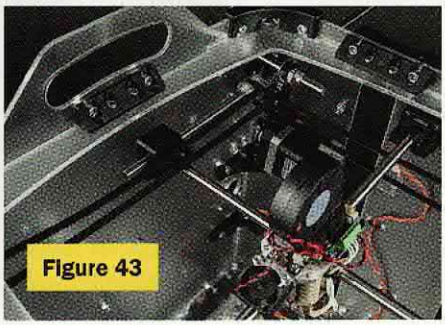


Figure 43

Effectuez la même opération pour les autres poulies « T2,5/19T » (8 au total).

Assurez-vous de ne pas abimer le filetage lorsque vous vissez la vis de réglage dans la poulie, serrez lentement sans forcer (voir la figure 33).

Ensuite, insérez sur le côté gauche de l'imprimante 2 tiges (barres) de 350 mm et de Ø 8 mm (voir la figure 34).

De même, insérez à partir de l'arrière de l'imprimante 2 tiges (barres) de 350 mm et de Ø 8 mm (voir la figure 35).

Faites glisser la poulie « T2,5/19T » de Ø 8 mm vers l'extrême gauche de la barre et placez une courroie « T2,5 180 mm x 6 mm » sur cette poulie ainsi que sur la poulie du moteur pas à pas.

Remarquez l'orientation de la vis de réglage de la poulie, elle doit se trouver vers le moteur (et non pas vers la paroi transparente) de la tige comme indiqué en figure 36.

Effectuez la même opération pour la barre arrière droite et vérifiez l'orientation de la vis de réglage de la poulie.

Insérez une courroie « T2,5 670 mm x 6 mm » à côté de la courroie « T2,5 180 mm x 6 mm » sur la poulie (que vous venez d'installer) de la barre située sur le panneau latéral gauche (lorsque vous regardez l'imprimante de face),

La courroie de 670 mm doit se trouver à droite de la courroie de 180 mm (voir la figure 37).

Ensuite insérez une courroie « T2,5 685 mm x 6 mm » sur la poulie que vous avez installée en premier (arrière gauche) à côté de la courroie « T2,5 180 mm x 6 mm ».

La courroie de 685mm doit se trouver à droite de la courroie de 180 mm (voir la figure 38). Vous devez obtenir un résultat semblable à celui de la figure 39.

Insérez une poulie « T2,5/19T » de Ø 8 mm sur la barre située à gauche et placez une courroie « T2,5 685 mm x 6 mm ».

Vous devez faire en sorte que cette courroie soit lâche (voir la figure 40).

La vis de réglage M3 x 4 mm doit se situer du côté le plus petit de la barre et non pas vers la paroi transparente de la figure 40.

Insérez une poulie « T2,5/19T » de Ø 8 mm sur la barre arrière gauche et placez une courroie « T2,5 670 mm x 6 mm » de façon qu'elle soit lâche sur cette poulie.

La vis de réglage M3 x 4 mm doit se situer du côté le plus petit de la barre (voir la figure 41). Vous devez obtenir un résultat semblable à celui de la figure 42.

Introduisez avec précaution dans l'imprimante, en **utilisant des gants**, l'ensemble tête d'impression avec les 2 chariots et les barres que vous avez monté précédemment, c'est-à-dire l'ensemble mécanique correspondant à la figure 27.

Faites glisser la barre arrière gauche à travers le chariot comme illustré en figure 43. L'orientation de la tête d'impression et l'orientation du chariot doivent correspondre à celles de la figure 43.

Ensuite faites glisser la barre arrière droite à travers l'autre chariot. L'orientation de la tête d'impression et l'orientation du chariot doivent correspondre à celles de la figure 44.

Vous devez obtenir un ensemble mécanique identique à celui de la figure 45, si cela n'est pas le cas vérifiez les étapes précédentes.

Ensuite vous devez introduire la barre latérale gauche située vers le fond de l'imprimante à travers le chariot dont la patte est coupée, si vous avez effectué correctement les étapes précédentes (voir la figure 46).

Ensuite faites glisser la barre latérale gauche située vers l'avant de l'imprimante à travers le chariot (comme illustré en figure 47).

Vérifiez encore une fois les orientations des chariots, notamment celle du chariot avec la patte coupée.

Insérez une poulie « T2,5/19T » de Ø 8 mm sur la tige latérale gauche située vers le fond de l'imprimante lorsque vous la regardez de face.

Positionnez la poulie vers la droite et placez une nouvelle courroie « T2,5 685 mm x 6 mm » (voir la figure 48).

La vis de réglage M3 x 4 mm doit se situer du côté du moteur droit lorsque vous la regardez de face (voir la figure 49).

Ensuite insérez une poulie « T2,5/19T » de Ø 8 mm sur la barre la plus proche de la face avant du côté droit (lorsque vous la regardez de face) et placez une courroie « T2,5 685 mm x 6 mm » de manière à ce qu'elle soit lâche.

Attention à l'orientation de la poulie, la vis de réglage M3 x 4 mm doit se situer vers l'intérieur de l'imprimante comme l'indique la figure 49.

Poussez la barre pour la faire passer dans le roulement situé de l'autre côté.

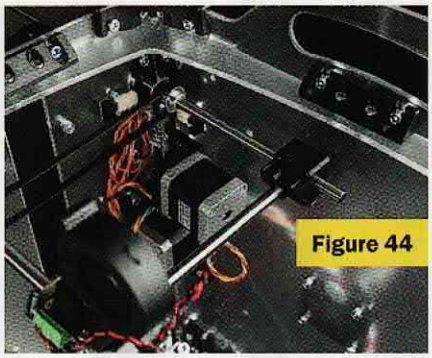


Figure 44

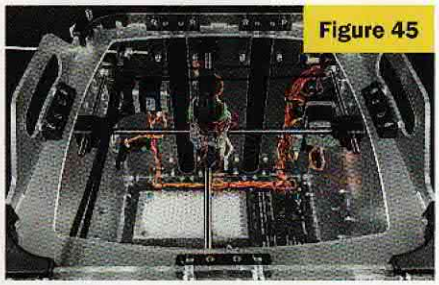


Figure 45

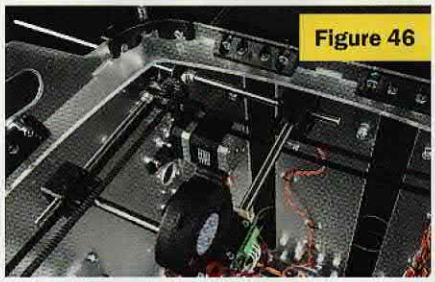


Figure 46

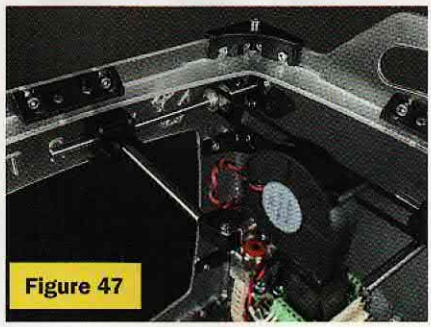


Figure 47

Figure 48

Figure 49 : la vis de réglage doit se situer du côté du moteur droit lorsque vous la regardez de face.

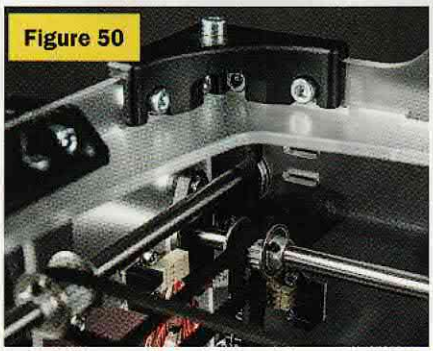


Figure 50

Poussez aussi la tige latérale gauche située vers le fond de l'imprimante pour qu'elle passe dans le roulement du côté opposé (voir la figure 50).

Maintenant, tournez les couvercles des roulements de cette barre de sorte qu'elle ne puisse plus coulisser (voir la figure 51).

Insérez une poulie « T2,5/19T » de Ø 8 mm sur la barre du fond à droite et placez une nouvelle courroie « T2,5 670 mm x 6 mm ».

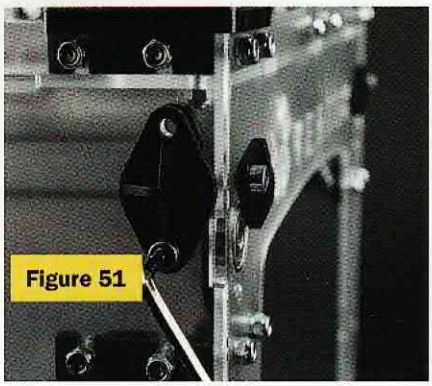
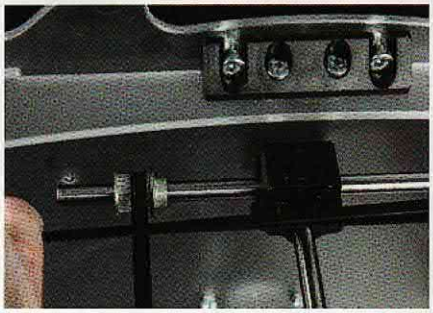


Figure 51

Figure 52

Figure 53 : la vis de réglage M3 x 4 mm doit se situer vers le chariot.



La vis de réglage M3 x 4 mm doit se situer du côté du chariot (voir la figure 52).

Ensuite insérez une poulie « T2,5/19T » de Ø 8 mm du côté gauche de la même barre arrière et placez une courroie « T2,5 670 mm x 6 mm » de façon à ce qu'elle soit lâche sur cette poulie. La vis de réglage M3 x 4 mm doit se situer vers le chariot (vers l'intérieur de l'imprimante) comme l'illustre la figure 53.

Enfin poussez la barre pour qu'elle atteigne les deux roulements situés de chaque côté. Tournez les couvercles afin que la barre ne puisse plus coulisser.

L'assemblage de l'imprimante doit ressembler à celui de la figure 54. Il ne vous reste plus qu'à visser les écrous de tous les couvercles.

Nous allons maintenant passez à l'étape du serrage des différentes courroies et poulies.

Serrage des courroies et poulies

Pour cette étape vous devez vous munir des pièces suivantes (voir la figure 55) :

- 4 pinces courroies ;
- 8 boulons M3 x 12 mm.

Disposez une pince courroie avec 2 boulons M3 x 12 mm du côté gauche du chariot « XY » de sorte que vous puissiez visser les deux écrous M3 (voir la figure 56).

Notez que l'encoche sur le dessus de la pince courroie doit correspondre avec l'encoche du chariot. Serrez les 2 boulons M3 x 12 mm et vérifiez que la courroie passe sous la pince courroie. Répétez la même opération pour les autres chariots.

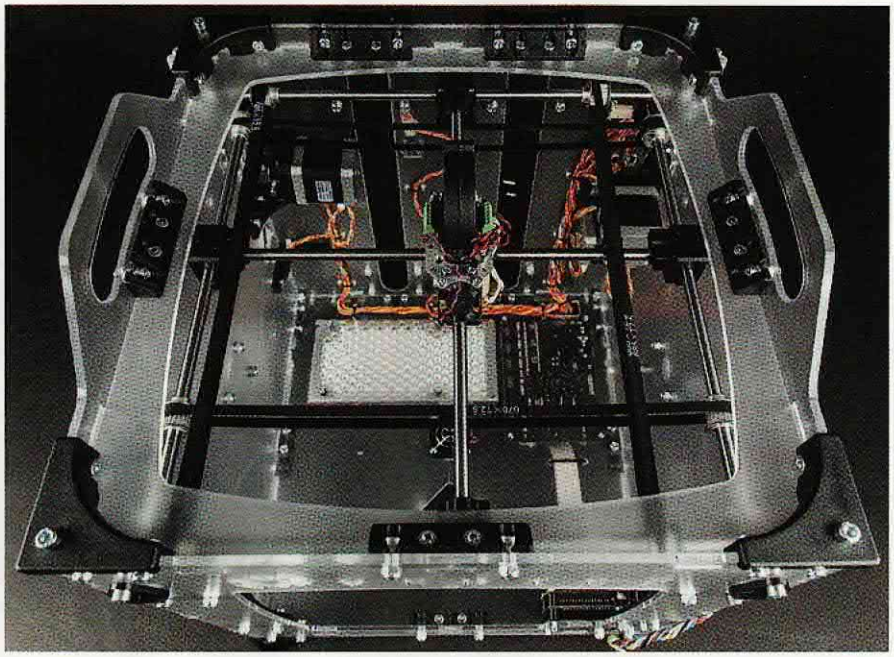


Figure 54 : l'assemblage de l'imprimante une fois terminé.

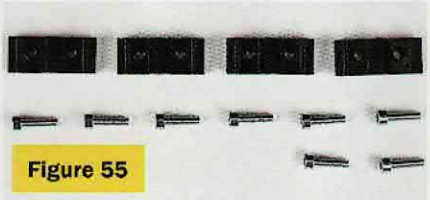


Figure 55

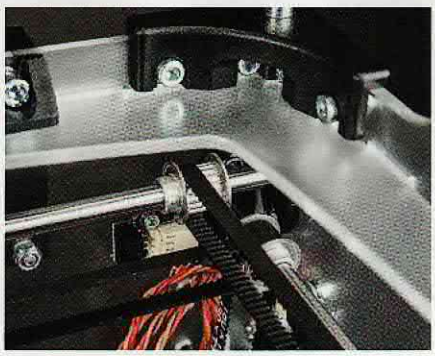


Figure 57 : les courroies doivent s'appuyer contre les côtés de chaque bride où se situent les vis de réglages.

Ensuite vérifiez que toutes les courroies s'appuient contre les côtés de chaque bride où se situent les vis de réglages pour toutes les poulies (voir la figure 57).

Déplacez toutes les poulies de façon à ce que les courroies se trouvent alignées sur chaque ensemble « poulie - chariot – poulie », à droite, sur la face avant, à gauche et à l'arrière.

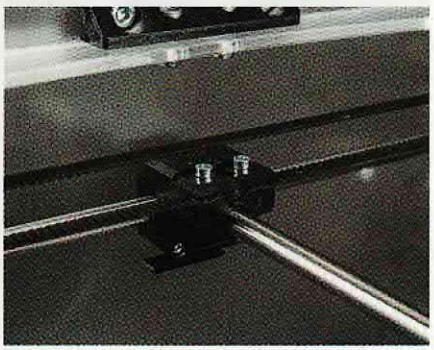


Figure 56 : disposition de la pince courroie du côté gauche du chariot.

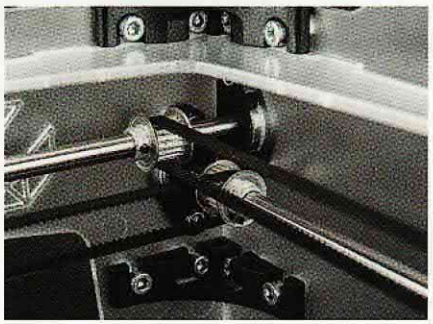


Figure 58 : ici le coin avant gauche.

Les 4 coins avec les poulies doivent se présenter comme l'exemple de la figure 58 qui correspond au coin avant gauche.

Vérifiez soigneusement les alignements des coins avant gauche, avant droit, arrière gauche et arrière droit, car les emplacements des poulies et des courroies déterminent la fluidité des mouvements.

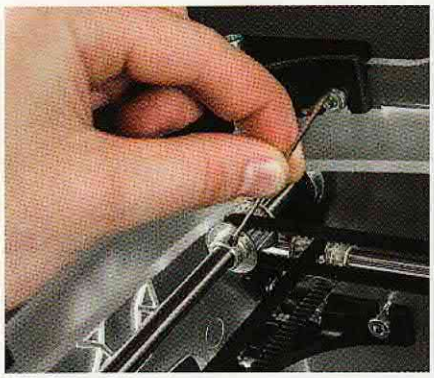


Figure 59 : serrage de la vis de réglage du coin avant gauche, répétez l'opération pour les autres coins.

Figure 61 : serrage de la vis de réglage du coin arrière droit. Répétez l'opération pour les autres coins.

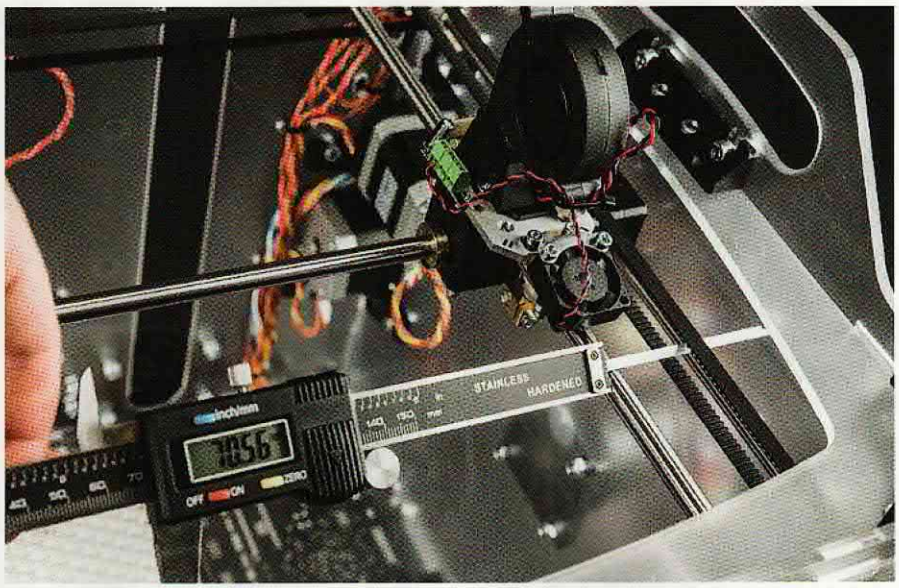


Figure 60 : à l'aide du pied à coulisse, vérifiez le parallélisme de la barre par rapport au panneau latéral droit.

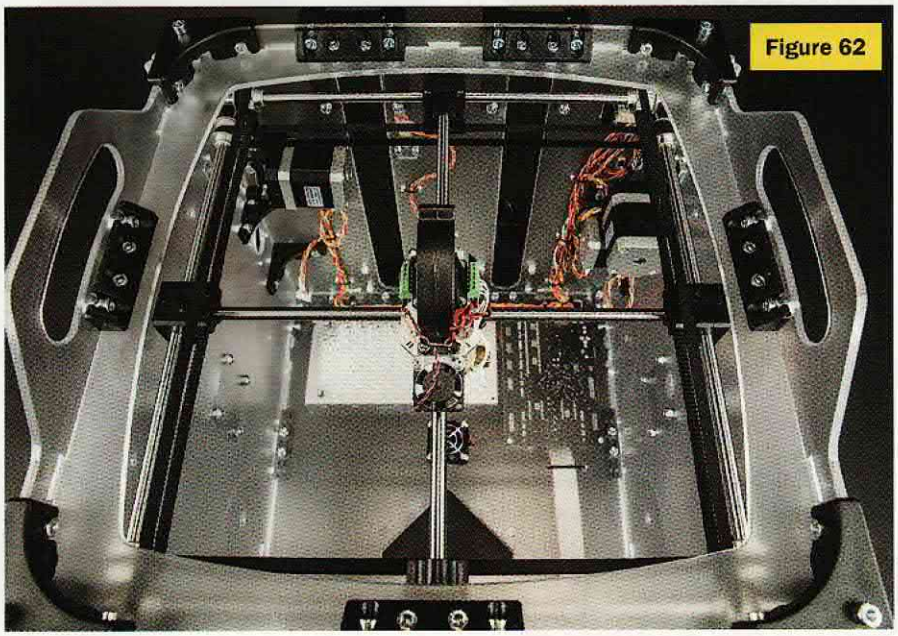


Figure 62

Maintenant vous devez vérifier que la barre est parallèle (c'est-à-dire qu'elle est bien d'équerre) par rapport au panneau arrière. Pour cela déplacez le chariot « XY », à l'aide d'un pied à coulisse mesurez la distance à chaque extrémité de la barre entre celle-ci et le panneau arrière. La mesure doit être identique.

Une fois le réglage effectué (barre parfaitement parallèle), serrez les vis de réglage M3 x 4 mm sur les 4 poulies (voir la figure 59). Si vous devez déplacer le chariot « XY » pour pouvoir atteindre la vis de réglage, assurez-vous que le parallélisme soit toujours maintenu.

Ensuite de la même manière, à l'aide du pied à coulisse, assurez-vous que l'autre barre soit parfaitement parallèle par rapport au panneau latéral droit (voir la figure 60). Ainsi les mouvements des roulements s'effectueront sans frottement.

Lorsque la barre est parallèle, serrez les vis de réglages sur les 4 poulies parallèles à cette barre (voir la figure 61). Si vous devez déplacer le chariot « XY » pour pouvoir atteindre la vis de réglage, assurez-vous que le parallélisme soit toujours maintenu.

L'assemblage de l'imprimante doit maintenant ressembler à celui de la figure 62. **La tête d'impression doit pouvoir se déplacer manuellement** (utilisez des gants).

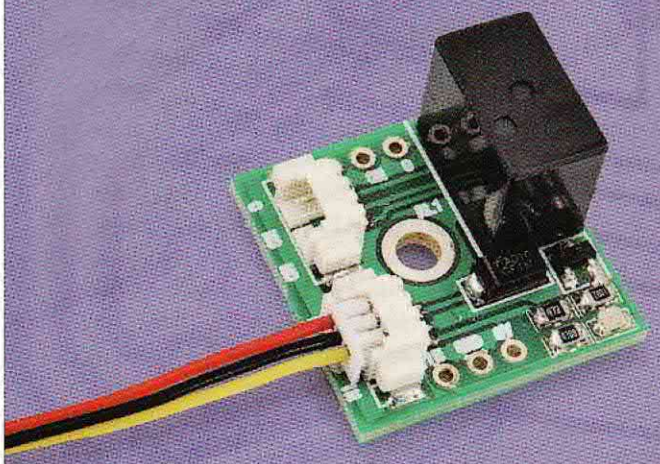
Il est normal que vous ressentiez une petite résistance. Cela est principalement dû à la tension de la courroie. Si vous pensez que la résistance est trop grande, assurez-vous que les barres du chariot « XY » soient parfaitement perpendiculaires l'une par rapport à l'autre.

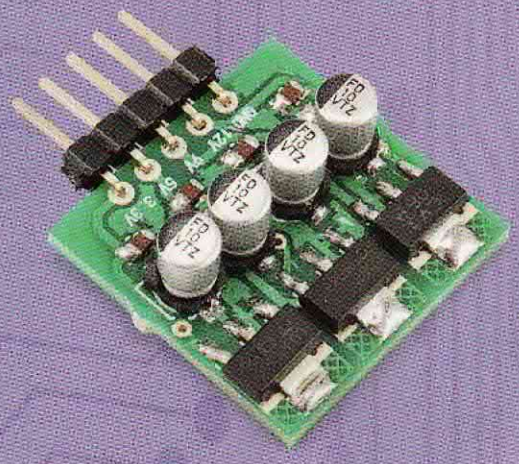
Si la résistance augmente lorsque vous approchez la tête d'impression de l'un des panneaux, il est possible que la distance entre les 2 chariots « XY » ne soit pas correcte. Essayez de résoudre le problème en les poussant ou en les tirant ensemble de manière délicate.

Enfin, fixez tous les boulons des supports des moteurs X et Y en vous assurant qu'une certaine tension soit présente sur les petites courroies.

Nous arrivons au terme de cette partie, nous n'avons pas encore terminé le montage mécanique car il nous manque l'assemblage de l'axe des Z ainsi que le plateau d'impression que nous aborderons dans le prochain numéro 138 d'Électronique et Loisirs Magazine. Pour l'instant vous avez beaucoup de travail !

BREAKOUT BOARD - 3





CARTES DE PROTOTYPAGE MULTIFONCTIONS

Troisième partie

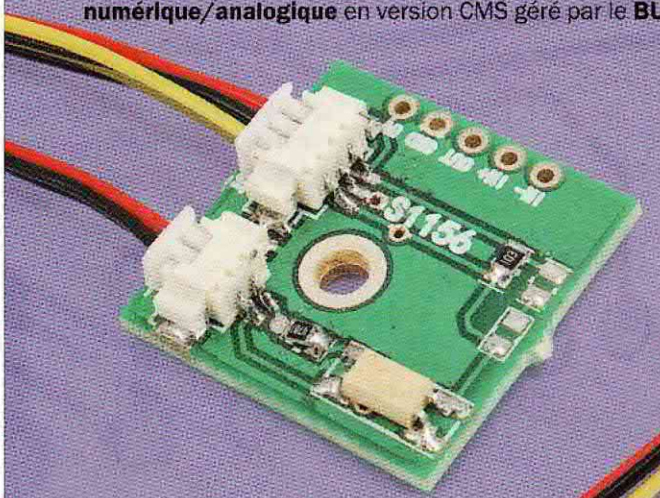
de Alessandro Sottocornola

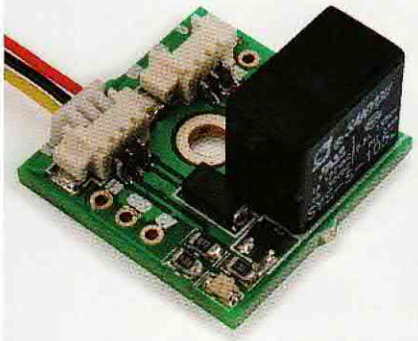
Nous décrivons dans cet article 4 nouvelles cartes de prototypage ou « breakout board » basées sur un relais, un optocoupleur, un régulateur de tension multiple, et pour terminer un convertisseur DAC que nous interfacerons avec une carte Arduino Uno.

Nous enrichissons notre série de cartes de prototypage et de tests avec diverses fonctionnalités utiles comme la **gestion d'un relais** par des niveaux logiques, l'interfaçage d'un **capteur optique** et d'une **alimentation linéaire** multiple qui fournit 3 tensions stabilisées avec des valeurs standards : 9 VDC, 5 VDC et 3,3 VDC.

Enfin nous vous proposons une carte dotée d'un **convertisseur numérique/analogique** en version CMS géré par le **BUS I2C**, dont nous étudierons le fonctionnement à l'aide d'une carte Arduino Uno.

Nous commençons par la description de la carte relais, qui peut être considérée comme un « actionneur générique » piloté par des niveaux logiques 0/5V (TTL). Grâce à cette carte il est possible de commander des charges électriques à basse ou moyenne tension tout en maintenant une isolation galvanique avec le circuit de commande.





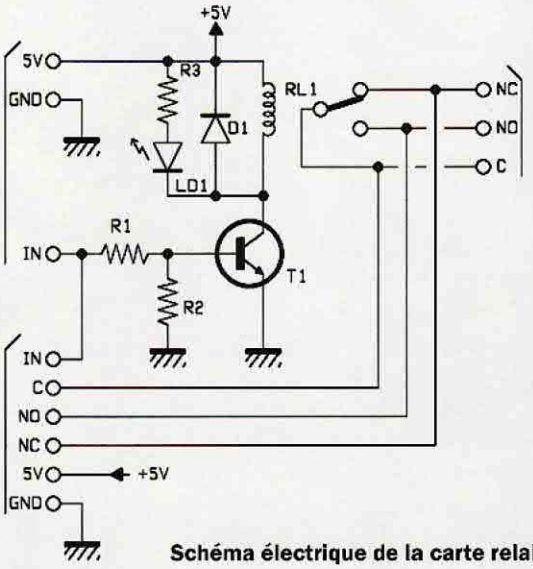


Schéma électrique de la carte relais

Par exemple des petits moteurs électriques, des sirènes ou encore des serrures, etc.

La carte relais

Cette carte contient un relais à **simple contact** dont la bobine fonctionne avec une tension de **5 V**. Elle permet de faire coller ou décoller les 3 contacts **C**, **NC** et **NO**. Chacun de ces contacts est reporté vers une borne de sortie ce qui permet d'insérer la carte dans un montage existant.

Par exemple, vous pouvez ajouter une sortie relais supplémentaire simplement en insérant la « breakout board », comme si elle était un composant enfichable.

Pour fonctionner, la carte nécessite une tension d'alimentation de **5 VDC** appliquée au point « **+ 5V** » (positif) et au point **GND** (masse). La **consommation** avec le relais enclenché est de **40 mA**. Sur l'entrée de commande « **IN** », une tension référencée à la masse (GND) doit être appliquée avec une valeur comprise entre **3 VDC** et **12 VDC**.

Cela signifie que nous pouvons commander la carte avec des niveaux logiques prélevés sur des circuits de type **TTL** fonctionnant à 3,3 VDC, mais aussi avec des circuits de type **CMOS** fonctionnant à 12 VDC.

Le courant absorbé par l'entrée de commande « IN » est de 0,6 mA à 3,3 VDC, 0,95 mA à 5 VDC et 2,5 mA à 12 VDC.

Cette valeur est faible, ce qui permet la commande de la carte par n'importe quel type de porte logique y compris une porte de type CMOS.

La tension appliquée sur l'entrée « IN » et la masse polarise directement et convenablement le **diviseur de tension** constitué par les résistances **R1** et **R2**.

La résistance R2 permet de faire en sorte que lorsque l'entrée « IN » est flottante (non connectée), la base du transistor T1 (BC817) ne soit pas polarisée par des perturbations environnantes.

Ainsi, lorsque la **base du transistor T1 est polarisée** correctement par un niveau logique appliqué sur l'entrée « IN », le transistor devient **conducteur** et donc son potentiel au niveau du collecteur tend vers 0 V (masse).

Il en découle qu'un **courant circule dans la bobine du relais** du « +5V » vers la masse, ce qui a pour effet d'établir le contact entre les points C et NO (au repos le contact est en position C - NC).

L'**enclenchement** du relais est signalé par la LED **LD1** munie en série d'une résistance R3 de limitation du courant. L'ensemble « **LED + R3** » est connecté en parallèle sur la bobine du relais RL1 de manière à ce que la LED s'allume lorsque la bobine est alimentée par la conduction de T1.

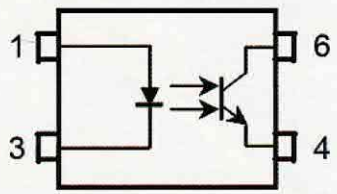
En parallèle sur la bobine se trouve également la **diode D1** de protection, dont la fonction est de **réduire les tensions inverses** provoquées par le **phénomène d'auto-induction qui s'oppose à la variation du courant dans le circuit** lorsque le transistor passe de l'état conducteur à l'état bloqué (lorsque « IN » est débranché ou remise à 0).

Sans la diode D1, lorsque **T1 change d'état** (état passant vers l'état bloqué, ce qui correspond à une interruption du courant dans le circuit), **la bobine génèrerait des surtensions inverses** et se mettrait à osciller jusqu'à ce que l'énergie stockée dans celle-ci disparaisse.

Le relais est un modèle miniature, son contact permet de commuter un courant de 1 A sous une tension alternative de 125 VAC (ou sous une tension continue de 60 VDC).

Si vous devez commander des charges plus importantes, vous devez utiliser un relais de puissance dont la bobine sera reliée au contact du relais miniature présent sur la « breakout board ».

Nous terminons la description de cette carte en remarquant que l'alimentation, l'entrée « IN » et les 3 contacts du relais sont reportés sur les côtés du circuit imprimé de la carte.



Connexions internes de l'optocoupleur TLP181. Cette intégration formée par une diode et un phototransistor ne comporte que 4 broches, contrairement à la plupart des optocouplers.

La carte optocoupleur

Cette carte est conçue pour équiper, d'une entrée opto-isolée, tout circuit fonctionnant à des niveaux logiques TTL, ou pour transmettre une condition logique entre deux circuits tout en les maintenant isolés galvaniquement.

Le cœur du circuit est l'**optocoupleur TLP181** composé d'une LED infrarouge connectée entre les broches 1 (anode) et 3 (cathode) et faisant face à un phototransistor NPN dont le collecteur est relié à la broche 6 et l'émetteur à la broche 4 (la base n'est pas accessible de l'extérieur).

Lorsqu'une **tension** d'une valeur suffisante est appliquée entre les **broches 1** et **3**, avec un potentiel supérieur sur la broche 1 (anode), la **photodiode à l'intérieur de l'optocoupleur devient conductrice** et s'allume.

Le **phototransistor**, qui était jusqu'alors bloqué, devient **conducteur** jusqu'à saturation. Le collecteur, polarisé par la tension d'alimentation de 5 V à travers la résistance R2 (10 kΩ), voit son potentiel passer de la condition de repos (+ 5 V) à la condition de fonctionnement (0 V).

L'alimentation de la carte optocoupleur s'effectue à l'aide d'une tension de **5 VDC stabilisée**, sa consommation de courant est très faible (1 mA maxi) de sorte que vous pouvez l'intégrer dans un montage existant ou l'utiliser dans des circuits fonctionnant sur piles ou avec de petits panneaux photovoltaïques.

L'entrée de cette carte s'effectue sur les contacts « **IN+** » et « **IN-** ». La tension appliquée entre ces 2 points peut être continue avec une valeur comprise entre **6 VDC** et **30 VDC**. Il est possible de relier aussi une **tension alternative sinusoïdale** dont l'amplitude est comprise entre **5 VAC** et **30 VAC**.

Dans le premier cas, le collecteur du phototransistor prend un niveau logique proche de 0 V pendant toute la durée de présence de la tension d'entrée.

Dans le second cas, le collecteur prend des niveaux logiques soit de 5 V soit de 0 V (signal rectangulaire) dont la fréquence est identique à celle de la sinusoïde appliquée sur les points « IN+ » et « IN- ».

La photodiode de l'optocoupleur conduit pour des courants assez faibles, de l'ordre du milliampère. Son **ratio de transfert de courant** (**CTR** ou **Current Transfer Ratio**), qui est le rapport entre le courant de sortie (le courant de collecteur du phototransistor) et le courant d'entrée, vaut environ **60 %** à saturation.

Le CRT peut atteindre jusqu'à 5 fois dans le meilleur des cas, avec un courant d'entrée de 5 mA et une tension collecteur-émetteur de 5 V, ce qui n'est pas notre cas.

Comme pour toutes les « breakout board » que nous vous proposons, tous les contacts (entrée, sortie, alimentation) sont reportés sur les côtés du circuit imprimé au pas de 2,54 mm afin d'accueillir une barrette mâle. De plus des connecteurs vous permettent de relier des fils volants afin de faciliter l'intégration de la carte.

La carte alimentation

Nous abordons maintenant la description de la carte alimentation contenant les régulateurs de tensions intégrés, grâce à laquelle, à partir d'une tension d'alimentation continue comprise entre 12 VDC et 15 VDC, nous obtenons **trois tensions** continues et stabilisées ayant les valeurs suivantes : **3,3 VDC**, **5 VDC** et **9 VDC**.

Cette carte a été conçue pour fournir les tensions les plus couramment utilisées. Le **3,3 V** correspond à l'alimentation des circuits logiques les plus modernes (ainsi que les microcontrôleurs récents), tandis que le **5 V** est adapté à l'alimentation des circuits logiques **TTL/CMOS** traditionnels.

La tension de **9 V** peut être utilisée pour alimenter des circuits audio, mais aussi des portes logiques CMS, des relais miniatures, etc.

Les 3 régulateurs de tension utilisés sont les suivants :

- **LM2940IMP-9.0/NOBP (U1)** ;
- **LD1117S50CTR (U2)** ;
- **LD1117S33CTR (U3)**.

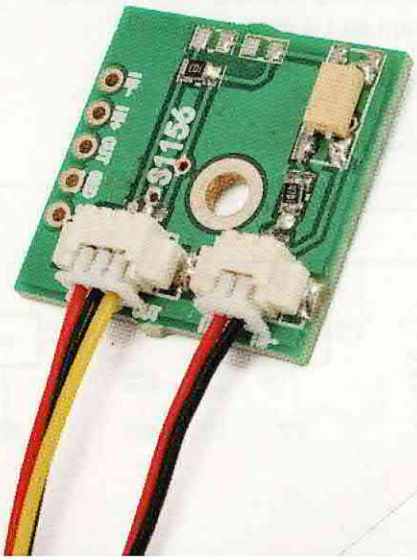
Ces trois régulateurs sont de **types linéaires** et **séries**, c'est-à-dire qu'ils maintiennent la tension de sortie constante même si la tension d'entrée varie dans des proportions importantes.

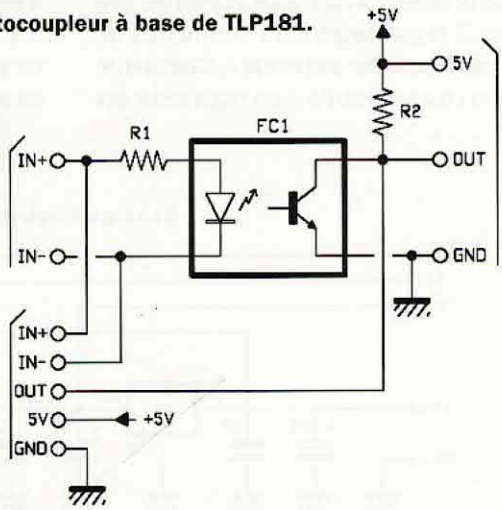
Ce mode de fonctionnement peut amener les régulateurs à **dissiper une puissance importante** qui peut les faire surchauffer. Afin de ne pas les endommager, nous vous rappelons que la puissance maximale dissipée est indiquée par le fabricant pour une température ambiante de fonctionnement de 25 °C.

Elle ne doit jamais être dépassée, cette puissance (Pd) se calcule grâce à la formule suivante :

$$Pd = (Vi - Vu) \times Iu$$

Schéma électrique de la carte optocoupleur à base de TLP181.





où « **Vi** » est la tension appliquée à l'entrée, « **Vu** » la tension de sortie et « **Iu** » le courant de sortie.

Examinons le schéma électrique de l'alimentation, nous reviendrons plus loin sur le principe de puissance dissipée (Pd).

Le circuit est alimenté typiquement par une **tension de 12 V** entre ses bornes « **12V** » et « **GND** » (respectivement le positif et le négatif).

Le **courant total consommé par** l'alimentation est **égal à la somme des courants** fournis sur chacune des 3 sorties.

Par exemple si sur le 3,3 V le courant fourni est de 200 mA, sur le 5 V de 400 mA et sur le 9 V de 400 mA, le courant total consommé est donc de 200 mA + 400 mA + 400 mA = 1 A.

La source d'alimentation d'entrée est filtrée par les condensateurs C1 et C2, ils réduisent les éventuels bruits provenant du secteur.

Les 3 régulateurs de tension disposent de 3 broches (une entrée, une sortie et une référence connectée à la masse).

Pour fonctionner, ils n'ont besoin que de très peu de composants externes, essentiellement des condensateurs de filtrage connectés sur l'entrée et la sortie afin d'assurer une bonne stabilité des tensions de sorties et filtrer les interférences.

Sur le schéma électrique de cette carte alimentation, vous constatez que les 3 régulateurs sont connectés en « **configuration parallèle** », c'est-à-dire que chaque entrée d'un régulateur est reliée en parallèle avec celles des 2 autres. Les 3 régulateurs sont chacun directement alimentés par la tension appliquée entre les points « 12V » et « GND ».

Le premier régulateur **U1** est un **LM2940IMP-9.0** de **Texas Instruments**. Il délivre une tension stabilisée de **9 VDC** avec un courant maximal de 1 A. Il supporte en entrée une tension comprise entre 6 V et 26 V.

U1 est un régulateur linéaire à **faible chute de tension** (« **low dropout** »), c'est à dire qu'il présente entre son entrée et sa sortie une faible chute de tension (donc un faible puissance dissipée), de l'ordre de **0,5 V** pour un courant de sortie de 1 A.

La **puissance dissipée du régulateur est ainsi limitée**, par exemple pour obtenir une tension de sortie de 9 V il suffit d'appliquer une tension de 9,5 V sur son entrée.

Avec un « **dropout** » (c'est-à-dire une chute de tension Vi - Vu) de seulement 0,5 V, même avec un courant de sortie de 1 A, le régulateur dissipe seulement une puissance de 0,5 W.

Compte tenu de la puissance maximale dissipée à 25 °C sans dissipateur (ce qui est notre cas car le régulateur est soudé directement sur le circuit imprimé) qui est de 2,2 W (selon le constructeur), cela signifie qu'en théorie pour un courant de sortie de 1 A le régulateur peut accepter une différence de potentiel de 2,2 V.

Il est possible d'alimenter le régulateur avec une tension de 11,2 V pour obtenir en sortie une tension de 9 V avec un courant de 1 A sans radiateur.

Étant donné que le « **dropout** » est très faible, il convient d'alimenter le régulateur avec une tension légèrement supérieure à celle de sortie pour utiliser le courant de sortie maximal, sinon vous devrez réduire celui-ci.

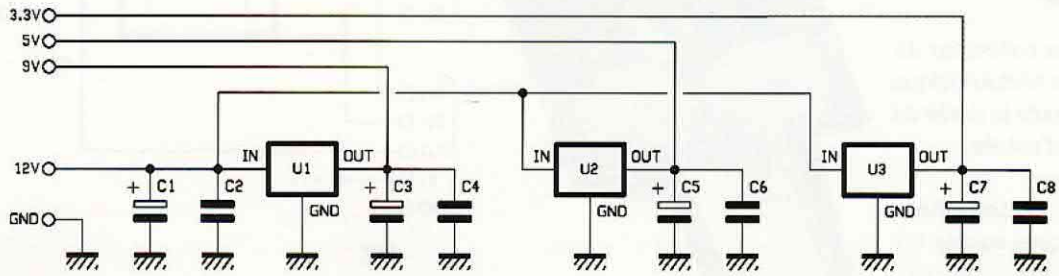
Le régulateur **LM2940IMP-9.0** est **protégé contre les courts-circuits en sortie**, il peut fournir un courant de crête de 1,9 A. Cependant pour un usage normal, il est conseillé de ne pas dépasser un courant maximal de 1 A. Au-delà, la tension de sortie diminue proportionnellement en fonction du courant en excès.

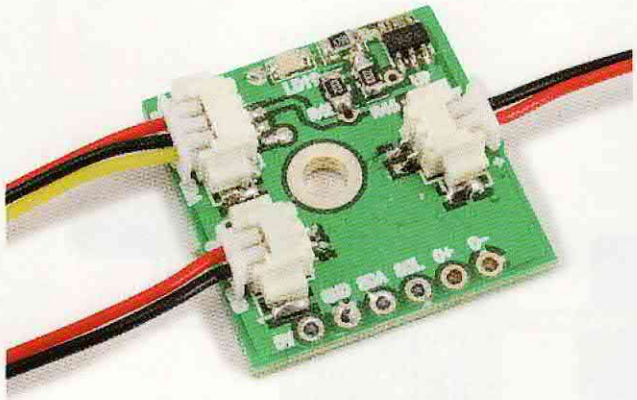
La tension de sortie de U1 est disponible entre les points « 9V » et « GND » (masse). De plus la tension de sortie est filtrée par les condensateurs C3 et C4.

Le deuxième régulateur est un modèle **LD1117S50** fabriqué par **STMicroelectronics**. Sa tension de sortie est de **5 VDC**, il est du même type que le précédent, c'est-à-dire à **faible chute de tension** ou « low dropout » (typiquement de l'ordre de 1 V).

Ce régulateur fournit donc une tension de sortie de 5 V pour un courant de sortie maximum de 800 mA. Sur le schéma électrique de la carte alimentation, la sortie 5 V de U2 est correctement filtrée par les condensateurs C5 et C6.

Schéma électrique de la carte alimentation.





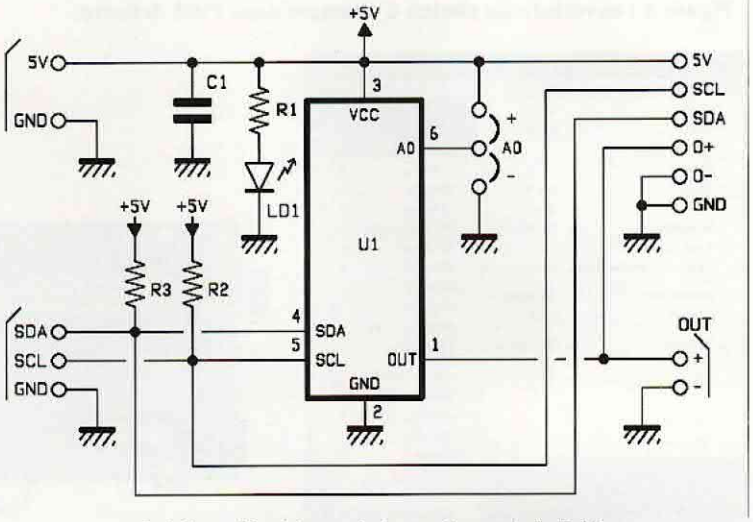


Schéma électrique de la carte module DAC.

En ce qui concerne la puissance maximale dissipée (Pd) de U2, elle dépend de la différence de potentiel entre son entrée et sa sortie.

Le régulateur U2 est alimenté par la tension d'entrée de la carte, qui vaut théoriquement 12 V. Si U2 doit fournir 800 mA en sortie alors sa puissance dissipée sera de :

Pd = 0,8 A * (12 - 5) V = 5,6 W

Ce régulateur avec un radiateur adéquat peut dissiper jusqu'à 12 W, mais sans dissipateur (ce qui est notre cas), à une température ambiante de 25 °C il ne peut dissiper que 1,13 W au maximum.

Ce qui signifie que dans notre configuration, il n'est pas possible d'obtenir le courant de sortie maximal de 800 mA.

En effet, étant donné que la puissance dissipée est donnée par le produit de la chute de tension (Vi - Vu) présente aux bornes du régulateur par le courant de sortie (Iu), nous pouvons déduire celui-ci à partir de la relation suivante :

I_u = Pd / (Vi - Vu)

Avec une tension d'entrée de 12 V, nous obtenons :

Iu = 1,13 W / 7 V = 161 mA

Par conséquent, le courant maximal de sortie du régulateur U2 est de 161 mA dans le cas d'une tension d'alimentation de 12 V. Si vous utilisez une **tension d'entrée de 10 V**, le courant de sortie de U2 peut atteindre **226 mA**.

Passons maintenant au dernier régulateur intégré U3, dont l'entrée est toujours alimentée par le 12V. En sortie il fournit une tension de **3,3 V** stabilisée et filtrée par les condensateurs C7 et C8.

Ce régulateur est toujours un **LD1117** de STMicroelectronics mais en **version 3,3 V**.

Il dispose des mêmes caractéristiques en termes de tension d'entrée maximale (15 V), de dissipation thermique (1,13 W sans radiateur), et de courant maximal de sortie. Ce dernier est théoriquement de 800 mA, mais comme pour le régulateur U2 nous devons tenir compte de la puissance maximale dissipée.

La différence de potentiel entre l'entrée et la sortie est de :

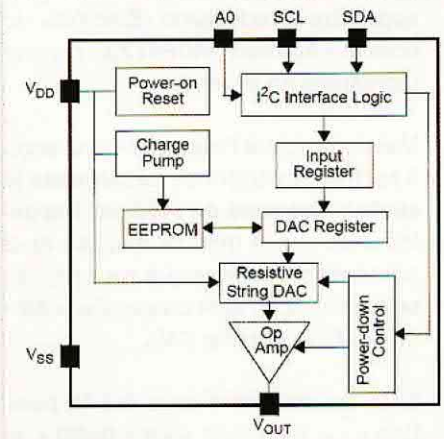
12 V - 3,3 V = 8,7 V

Le courant maximal en sortie sans radiateur sera de :

I_u = 1,13 W / 8,7 V ~ 130 mA

Avec une tension d'alimentation en entrée de seulement 10 V, U3 pourra fournir un courant de sortie d'environ 168 mA.

Nous concluons la description de cette carte d'alimentation en soulignant que, comme pour les autres « breakout board », toutes les broches (entrées, sorties et masse) sont reportées sur les bords du circuit imprimé afin de faciliter le câblage. Vous pouvez utiliser une barrette mâle au pas de 2,54 mm pour intégrer la carte dans un montage existant ou pour l'utiliser avec une plaque d'essai.



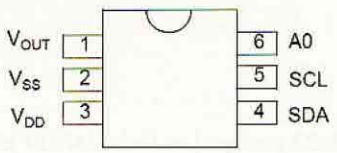
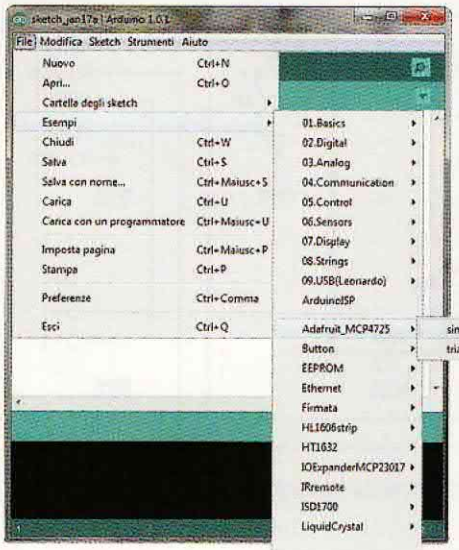


Schéma synoptique et brochage du convertisseur numérique/analogique Microchip MCP4725A0T-E. Comme vous pouvez le voir, il est encapsulé dans un boîtier CMS de type SOT-23 avec seulement 6 broches.

Le module DAC

Nous arrivons à la description de la dernière carte de prototypage pour cet article.

Figure 1 : ouverture du sketch d'exemple dans l'IDE Arduino.



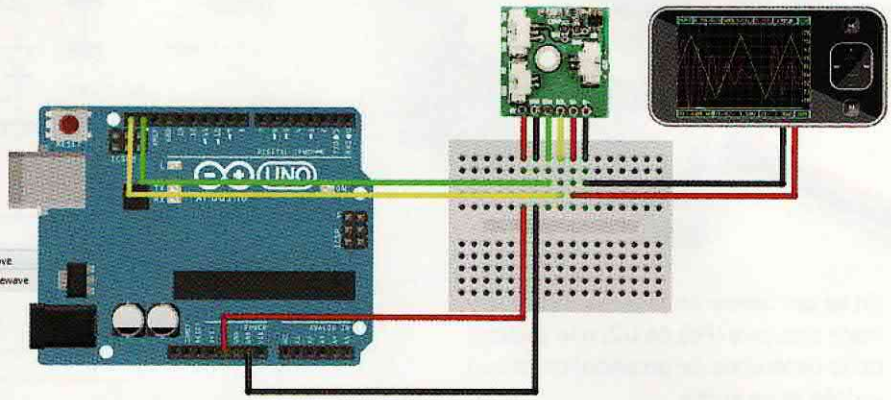


Figure 2 : câblage de la carte Arduino avec la « breakout board » contenant le DAC Microchip à l'aide d'une plaque d'essai à 400 contacts.

Il s'agit d'une carte dotée d'un DAC intégré ou **convertisseur numérique/analogique** dont la référence est **MCP4725A0T-E/CH**.

Ce composant est fabriqué par **Microchip**. Sa résolution est de 12 bits, il intègre une mémoire EEPROM et dispose d'un **bus I2C** dont l'**adresse peut être sélectionnée** à l'aide de la broche **A0** (6) en la plaçant à un niveau logique 1 ou 0.

Il est donc possible **de connecter sur un même bus I2C au maximum 2 DAC**. Les données de configuration et d'entrée du DAC peuvent être programmées dans l'EEPROM par l'utilisateur à l'aide du bus I2C.

Le **MCP4725A0T** s'alimente avec une tension unique (mono-tension) comprise entre **2,7 V** et **5,5 V**. Pour notre carte le circuit travaille avec une tension de **5 VDC** appliquée aux points « **5V** » et « **GND** ». La structure interne de l'amplificateur du DAC est de type « **rail-to-rail** », ce qui permet d'obtenir en sortie une tension variant de 0 V à 5 V.

La mémoire non volatile permet au DAC de maintenir les données entrantes en cas de coupure de courant, afin de présenter à nouveau la tension correspondante à la sortie analogique lorsque l'alimentation est rétablie.

Le convertisseur N/A peut fonctionner à **3 vitesses de conversion : Standard** (100 kbps), **Fast** (400 kbps) et **High-Speed** (3,4 Mbps).

Les connexions des données, d'alimentation et de sortie sont disponibles sur le côté du circuit imprimé (emplacement prévu pour une barrette au pas de 2,54 mm) ainsi que sous forme de connecteurs.

Cette carte DAC utilise un circuit intégré que nous n'avons pas choisi au hasard, il peut être **interfacé avec Arduino** et il est utilisable avec la **librairie Adafruit**.

Afin de gérer le module **DAC MCP4725A0T** avec **Arduino**, vous devez d'abord **copier la librairie dans le répertoire « \libraries »** de l'IDE Arduino.

Ensuite ouvrez l'IDE, et vous devez voir apparaître dans le menu « Exemples » le dossier « **Adafruit_MCP4725** » comme représenté en figure 1.

Maintenant que l'exemple est chargé, il est très important de **définir dans le sketch l'adresse du module**. Rappelez-vous que le module dispose de 2 adresses différentes que vous pouvez sélectionner à l'aide du cavalier « **A0** » présent sur la carte DAC.

Si le cavalier se trouve sur la position « - », l'adresse vaut « **0x60** », si

Listing 1

```
void setup(void) {
  Serial.begin(9600);
  Serial.println(«Hello!»);

  // For Adafruit MCP4725A1 the address is 0x62 (default) or 0x63 (ADDR pin
tied to VCC)
  // For MCP4725A0 the address is 0x60 or 0x61
  // For MCP4725A2 the address is 0x64 or 0x65
  dac.begin(0x60);

  Serial.println(«Generating a sine wave»);
}
```

Listing 2

```
#include <Wire.h>
#include <Adafruit_MCP4725.h>

Adafruit_MCP4725 dac;

void setup(void) {
 Serial.begin(9600);
 Serial.println(«Hello!»);

 // For Adafruit MCP4725A1 the address is 0x62 (default) or 0x63 (ADDR pin
tied to VCC)
 // For MCP4725A0 the address is 0x60 or 0x61
 // For MCP4725A2 the address is 0x64 or 0x65
 dac.begin(0x60);

 Serial.println(«Set Custom Value»);
}

void loop(void) {
        uint32_t value;
        value=2048;
        dac.setVoltage(value, false);  // from 0 to 4095
}
```

le cavalier se trouve sur la position « + », l'adresse vaut « **0x61** ». Dans notre cas, nous avons choisi d'utiliser l'adresse « **0x60** », nous devons donc rechercher dans le sketch la ligne « **dac.begin(0x62)** » afin de modifier la valeur « 0x62 » en « **0x60** », valeur qui correspond à celle de notre adresse comme vous pouvez le voir dans le Listing 1.

Maintenant, vous pouvez effectuer les connexions de la carte DAC comme le montre la figure 2. Comme vous pouvez le voir, nous avons connecté un oscilloscope afin de visualiser la forme de l'onde sinusoïdale que nous allons obtenir en exécutant le sketch Arduino.

Si vous souhaitez utiliser le module pour générer une tension unique (mono-tension), vous pouvez écrire un sketch d'exemple dans lequel, grâce à l'instruction « **dac.setVoltage(value, false);** », vous définissez la tension.

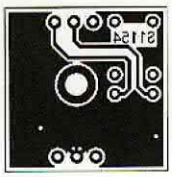
À cet égard, nous rappelons que la plage des valeurs est comprise entre 0 et 4095 (**4096 valeurs** soit **12 bits** ou **2^{12}**) qui correspondent respectivement à 0 V et 5 V. Si par exemple la valeur vaut 2048, la tension de sortie du DAC sera de 2,5 V.

Dans l'exemple du Listing 2, nous avons paramétré la variable « value » à 2048, ce qui correspond à une tension de sortie de 2,5 V.

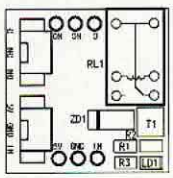
Réalisation pratique

Maintenant que nous avons décrit les schémas et expliqué le fonctionnement de ces 4 nouvelles cartes, nous

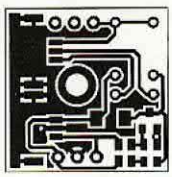
Plan de montage de la carte relais



Circuit imprimé à l'échelle 1 : 1 de la carte alimentation côté soudures.



Plan de câblage de la carte relais.



Circuit imprimé à l'échelle 1 : 1 de la carte alimentation côté composants.

Photo de l'un de nos prototypes de la carte relais.

Liste des composants de la carte relais

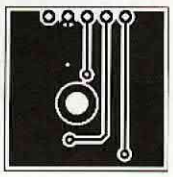
R1.....4,7 kΩ boîtier 0805
R2.....10 kΩ boîtier 0805
R3.....470 Ω boîtier 0805
D1.....GF1M VISHAY

LD1....LED rouge boîtier 0805
T1 .....BC817 boîtier SOT23
RL1....relais miniature 5 V 1 contact

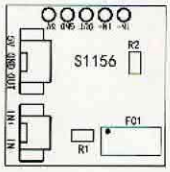
Divers
Barrette mâle 3 pôles (x2)

Connecteur JST 1,25mm 3 pôles pour ci (x2)
Connecteur JST 1,25mm 3 pôles fils volants (x2)

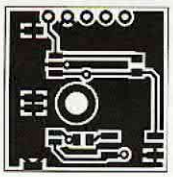
Plan de montage de la carte optocoupleur



Circuit imprimé à l'échelle 1 : 1 de la carte optocoupleur côté soudures.



Plan de câblage de la carte optocoupleur.



Circuit imprimé à l'échelle 1 : 1 de la carte optocoupleur côté composants.

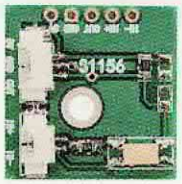
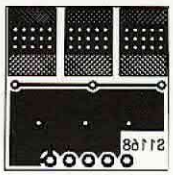


Photo de l'un de nos prototypes de la carte optocoupleur.

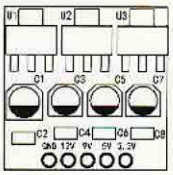
Liste des composants de la carte optocoupleur

R1.....4,7 kΩ boîtier 0805
R2.....10 kΩ boîtier 0805
C1.....100 nF céramique boîtier 0805
C2.....47 pF céramique boîtier 0805
LD1....LED verte boîtier 0805
FC1....TLP181
Divers
Barrette mâle 5 pôles coudée 90°
Connecteur JST 1,25mm 2 pôles pour circuit imprimé
Connecteur JST 1,25mm 3 pôles pour ci
Connecteur JST 1,25mm 2 pôles fils volants
Connecteur JST 1,25mm 3 pôles fils volants

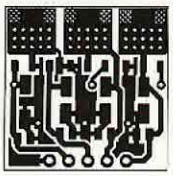
Plan de montage de la carte alimentation



Circuit imprimé à l'échelle 1 : 1 de la carte alimentation côté soudures.



Plan de câblage de la carte alimentation.



Circuit imprimé à l'échelle 1 : 1 de la carte alimentation côté composants.

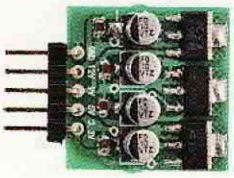


Photo de l'un de nos prototypes de la carte alimentation.

Liste des composants de la carte alimentation

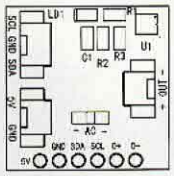
C1.....22 µF/16 V électrolytique (Ø 4mm)
C2.....100 nF céramique boîtier 0805
C3.....22 µF/16 V électrolytique (Ø 4mm)
C4.....100 nF céramique boîtier 0805
C5.....22 µF/16 V électrolytique (Ø 4mm)
C6.....100 nF céramique boîtier 0805
C7.....22 µF/16 V électrolytique (Ø 4mm)
C8.....100 nF céramique boîtier 0805

U1.....LM2940IMP-9.0/NOBP
U2.....LD1117S50CTR
U3.....LD1117S33CTR

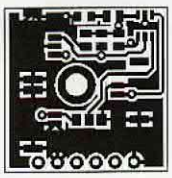
Divers
Barrette mâle 5 pôles coudée 90°

Plan de montage du module DAC

Circuit imprimé à l'échelle 1 : 1 du module DAC côté soudures.



Plan de câblage du module DAC.



Circuit imprimé à l'échelle 1 : 1 du module DAC côté composants.

Photo de l'un de nos prototypes du module DAC.

Liste des composants du module DAC

R1.....470 Ω boîtier 0805
R2.....10 kΩ boîtier 0805
R3.....10 kΩ boîtier 0805
C1.....100 nF céramique boîtier 0805

LD1....LED verte boîtier 0805
U1.....MCP4725A0T-E/CH

Divers

Barrette mâle 6 pôles coudée à 90°
Connecteur JST 1,25mm 2 pôles pour circuit imprimé (x2)

Connecteur JST 1,25mm 3 pôles pour circuit imprimé
Connecteur JST 1,25mm 2 pôles fils volants (x2)
Connecteur JST 1,25mm 3 pôles fils volants

allons étudier individuellement leur construction.

Vous devez dans un premier temps télécharger les typons des circuits imprimés sur notre site web : **www.electroniquemagazine.com** dans le **sommaire détaillé** de la revue **137** à l'onglet « **Télécharger** ». Vous y trouverez aussi la **librairie complète** « **Adafruit_MCP4725** ».

Rappelons que pour réaliser de bonnes soudures, il est conseillé d'utiliser un fer à souder de 25 W avec une pointe fine de 0,5 mm et de la soudure pour composants CMS. Il est préférable aussi de se munir d'une loupe, de préférence éclairée, afin de vérifier que les composants sont correctement centrés par rapport à leur emplacement et que de la soudure ne coule pas en dessous (risque de courts-circuits). Les câblages des composants des cartes sont visibles dans les plans de montages respectifs, aidez-vous des photos des prototypes.

Pour la carte relais vous devez prêter une attention particulière à l'orientation des transistors et des diodes. Le relais ne se positionne que dans un seul sens.

Pour la carte optocoupleur, il n'y a pas de précautions particulières sinon le respect de l'orientation du circuit intégré en boîtier TSSOP.

Quant à la carte alimentation, les 3 régulateurs de tension ont des boîtiers de type SOT-223, attention à ne pas les confondre entre eux, sinon l'alimentation risque de ne pas fonctionner correctement.

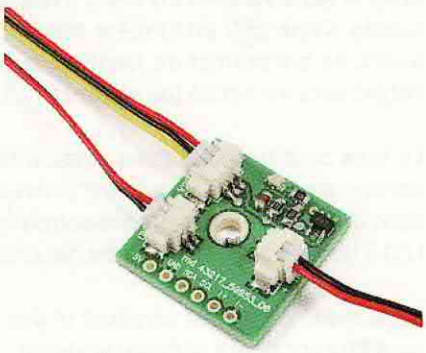
Enfin pour le module DAC (boîtier SOT-23), faites attention à son orientation.

La broche 1 est indiquée par le repère sur le boîtier et doit être positionnée vers le bord du circuit imprimé. Utilisez un peu de colle pour maintenir le circuit en position puis commencez par souder la broche 1 et continuez en diagonale du côté opposé, laissez refroidir le composant entre chaque soudure. Ne mettez pas trop de soudure, car elle risquerait de s'étendre en dessous du composant et provoquer des couts-circuits.

Terminez le montage du circuit en soudant les autres composants passifs. De façon générale, pour toutes les cartes les connecteurs JST sont facultatifs.

Si vous n'en trouvez pas dans le commerce, vous pouvez toujours souder délicatement des fils pour étendre les connexions. Les connecteurs JST sont disponibles sur le site de Farnell (http://fr.farnell.com).

Si vous devez insérer les « breakout board » dans des montages existants, utilisez des barrettes au pas de 2,54 mm droites ou coudées selon vos besoins.



Bras robotisé 5 mouvements

Parmi les nombreux robots présents sur le marché, nous avons trouvé et testé un produit en kit qui intéressera certainement les lecteurs passionnés de robotique mais aussi les novices dans ce domaine. C'est un véritable « jouet » technologique.

Nous avons trouvé sur le marché un produit sous forme de kit, certainement très intéressant sur le plan didactique et adapté pour ceux d'entre vous qui désirent aborder un domaine complexe, fascinant et futuriste qu'est la robotique.

Il s'agit d'un bras robotisé, capable d'effectuer 5 mouvements et de soulever un poids de 100 grammes, en le serrant entre ses « doigts » artificiels et, pas seulement dans le sens vertical. En effet, il dispose au niveau de ses pinces d'une LED en lumière blanche qui éclaire la zone avant, ce qui permet de localiser le point à partir duquel l'objet sera manipulé (ou attrapé).

Le bras peut travailler dans l'obscurité, car la LED utilisée comme projecteur a un angle d'émission très étroit et émet donc une lumière intense (en raison du fait que c'est aussi une LED à haut rendement). Le kit est disponible dans le commerce.

De la façon dont il est construit et des matériaux utilisés, il peut être considéré comme un « jouet » technologique, mais le définir comme un « jouet » est un peu simpliste.

En dehors de ses caractéristiques limitées par rapport à un robot industriel (poids qu'il peut soulever et finition du produit), qui sont proportionnelles à la cible et au prix du produit, ce bras robotisé est construit avec des mécanismes et une structure qui ressemblent à un robot utilisé dans les chaînes de productions industrielles.

Bien sûr, ici, ce bras est une version miniature et est construite avec des matériaux appropriés pour soulever ou déplacer des objets de faible masse. Mais le but de ce kit n'est pas d'assembler une voiture ou de souder des pièces mécaniques, il permet à des débutants dans le domaine de la robotique de se former à la réalisation de robots.

Pour un débutant, monter tout seul ce bras robotisé et le voir fonctionner pour découvrir ses possibilités, pourra le motiver à franchir le pas dans l'étude et la fabrication de robots beaucoup plus complexes. En effet, comme nous l'avons déjà expliqué, bien qu'il ressemble à un « jouet », ce bras est en mesure d'effectuer des mouvements articulés et complexes dans pratiquement toutes les positions.

Ses « doigts » (ou pinces) préhensiles (qui a la capacité de prendre, de saisir) permettent de saisir un objet dans n'importe quelle position.

Figure 1 : vous pouvez voir ici tous les éléments du kit qui sont regroupés et numérotés.

L'ensemble est similaire à celui de notre membre supérieur, la base correspond à l'épaule, le premier segment correspond à la zone humérale et sa jointure à l'articulation de l'épaule.

Le deuxième segment correspond à l'avant-bras, il se meut à l'aide d'un moteur qui permet la courbure de la partie correspondant à notre coude.

Enfin, la pince correspond à notre main et sa fixation au second segment est similaire à notre poignet. Le moteur permet d'effectuer des mouvements vers le haut et vers le bas (comme pour le poignet).

Les 5 mouvements sont :

- rotation sur le plan horizontal ;
- mouvement avant/arrière de la base du bras ;
- soulèvement/abaissement du second segment du bras ;
- montée/descente de la pince ;
- ouverture/fermeture de la pince.

Ces mouvements peuvent être exécutés en même temps, grâce à un boîtier de commande équipé de 5 interrupteurs séparés.

Pour effectuer ces 5 mouvements, le robot dispose de 5 moteurs électriques basse tension fonctionnant sur piles (ou accus).

Chaque moteur est équipé d'un réducteur qui permet d'effectuer les mouvements à une vitesse appropriée, et qui maintient la position du mouvement lorsque le moteur n'est plus alimenté.

De plus, chaque moteur comporte, sur l'engrenage de l'axe, un système d'embrayage qui empêche une surcharge lorsque le moteur n'arrive plus à fournir l'effort nécessaire au mouvement.

En résumé, comme nous l'avons évoqué précédemment, les solutions technologiques adoptées pour la conception de ce bras motorisé sont généralement présentes dans des robots de classe supérieure.

Pour apprendre et vous faire connaître ce produit, nous avons monté ce bras en suivant les instructions qui sont très détaillées et très claires dans le mode d'emploi du kit. Nous vous faisons part de notre expérience en vous expliquant quelques astuces.

Nous allons décrire les points principaux de la réalisation, et effectuer les tests. Tout d'abord voici les principales caractéristiques du bras robotisé.

- Force de levage : 100 g ;
- Alimentation : 4 piles ou accus R20 (non fournis) ;
- Hauteur maximale (bras déployé) : 38 cm ;

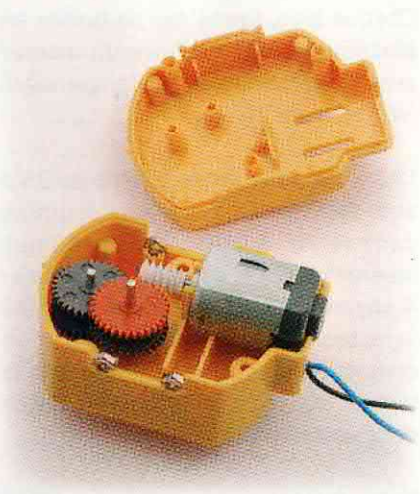


Figure 2 : vue du groupe moteur avec le réducteur, 4 engrenages et 2 arbres auxiliaires.

- Masse : 660 g ;
- Dimensions : 22,85 cm (L) x 16 cm (l) x 38 cm (H).

Tout d'abord nous commençons par déballer les différentes pièces constituant le kit, et nous les disposons sur notre plan de travail en les classant par catégorie (engrenages, moteurs, etc.) afin de rendre chaque pièce facilement identifiable (voir la figure 1).

Pour monter le bras robotisé, nous devons d'abord rassembler les 5 blocs moteurs avec leurs réducteurs.

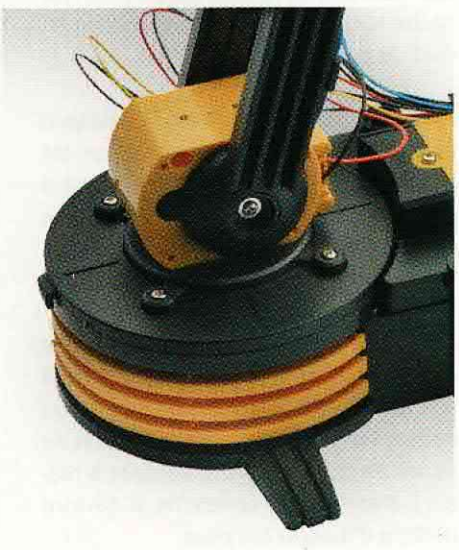


Figure 3 : ici vous pouvez voir la base ronde servant de support. Les anneaux jaunes contiennent le moteur qui fait tourner le bras, celui-ci étant soutenu par le plateau noir.

Chaque bloc, formé par un boîtier en plastique jaune, contient un moteur avec un certain nombre d'engrenages (voir la figure 2).

Chaque bloc se ferme à l'aide de 3 vis auto-taraudeuses qui sont fournies dans le kit, mais auparavant nous insérons les 2 axes et faisons glisser 4 engrenages. Ensuite nous insérons le moteur en plaçant la vis sans fin en contact avec le premier engrenage, le moteur doit être parfaitement inséré dans son emplacement.

Notez que les engrenages ne sont pas tous identiques, mais cela ne pose pas de problème particulier car ils se distinguent par leur couleur. Dans le manuel de montage, tous les endroits où doivent être disposés les engrenages de couleur différente sont clairement indiqués.

Une fois les moteurs montés, il est nécessaire de préparer la base. Celle-ci contient le compartiment à piles (4 de type R20) pour l'alimentation de la LED et des moteurs.

Dans le compartiment à piles, nous devons fixer les contacts à ressort et faire sortir les fils par le couvercle sur lequel nous fixons une carte regroupant les connexions avec le boîtier de commande, les connexions avec tous les moteurs, et la connexion à la LED haute luminosité qui se situe au sommet du bras.

Notez que les contacts des piles doivent être positionnés de telle sorte que les plus grands soient du côté opposé à celui de la base du bras, tandis que les deux autres contacts, avec un fil chacun, doivent être positionnés du côté opposé.

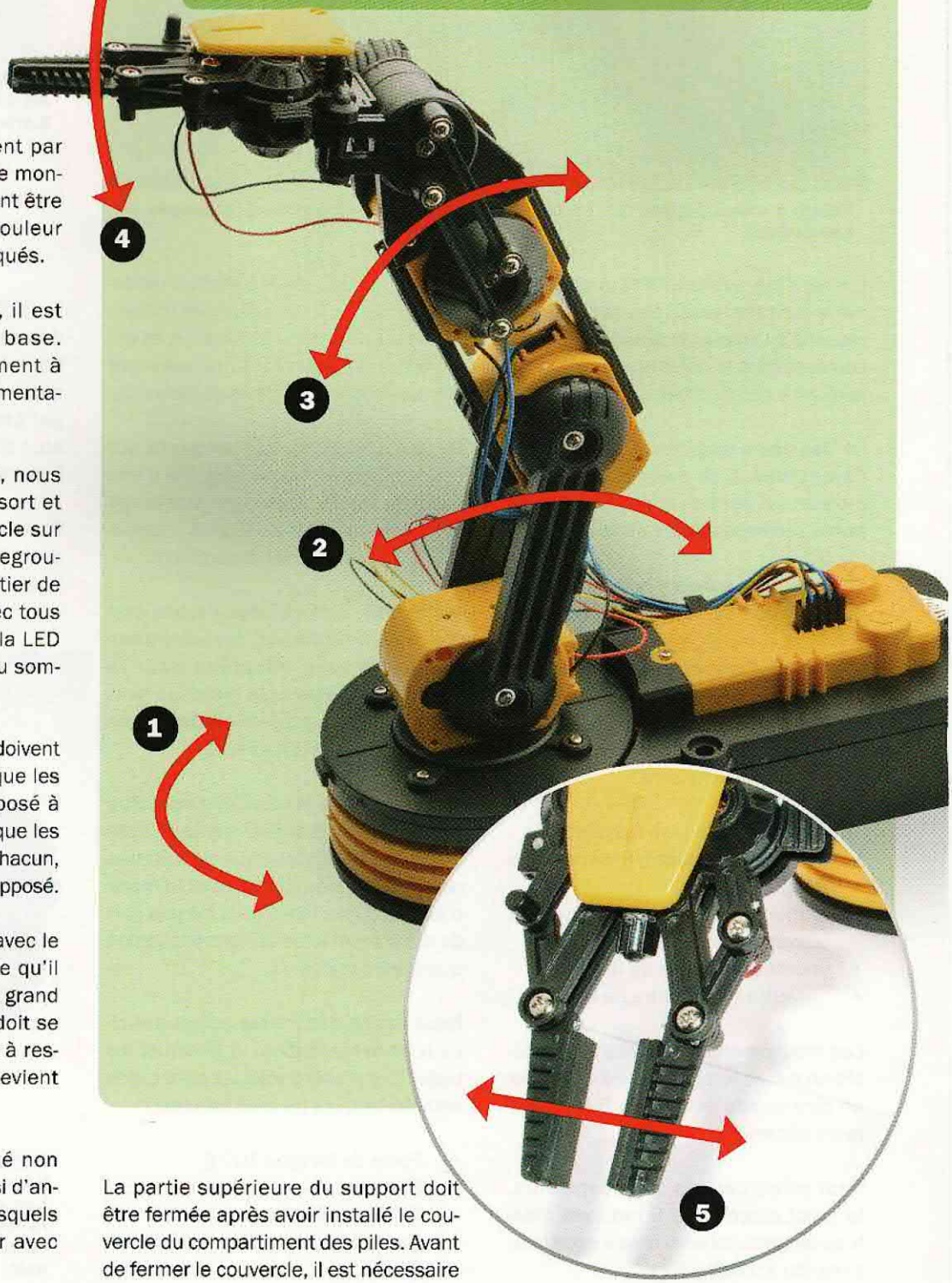
Parmi ces deux contacts, celui avec le ressort doit être inséré de sorte qu'il soit en face de la partie plate du grand contact tandis que le clip plat doit se situer en face du grand contact à ressort. Dans le cas contraire, il devient difficile d'insérer les piles.

Le support rotatif est constitué non seulement de la base, mais aussi d'anneaux à empiler à l'intérieur desquels est disposé un premier moteur avec son réducteur (voir la figure 3).

Les mouvements du bras

Grâce à la disposition judicieuse des 5 moteurs et à sa structure mécanique bien pensée, ce bras se comporte de manière simplifiée comme le bras d'un humain. Il peut effectuer différents mouvements qui le rendent très flexible. Il est adapté à la saisie et au positionnement d'objets dans un large champ d'applications.

1. rotation sur le plan horizontal de la base ;
2. mouvement avant/arrière de la base du bras ;
3. soulèvement/abaissement du segment supérieur du bras (« coude ») ;
4. montée/descente de la pince ;
5. ouverture/fermeture de la pince.



La partie supérieure du support doit être fermée après avoir installé le couvercle du compartiment des piles. Avant de fermer le couvercle, il est nécessaire

de raccorder au circuit imprimé le câble provenant du boîtier de commande.

Dans le boîtier de commande, des contacts métalliques doivent être assemblés par vissage sur le circuit imprimé (voir la figure 4).

À l'aide de leviers en plastique jaunes, les contacts sont maintenus pressés contre les pastilles de cuivre du circuit imprimé formant ainsi 5 interrupteurs.

Connectons le boîtier de commande et fermons le couvercle des piles. Fermons également le couvercle de la base rotative sur lequel il faut insérer le dernier engrenage du réducteur.

Maintenant nous devons assembler le bras en commençant par la pièce du bas selon les instructions. La partie inférieure du bras doit être disposée sur la base rotative, en l'encastrant dans l'engrenage du moteur de cette dernière.

Cette pièce contient ses propres engrenages, sur lesquels doit être calé l'engrenage du second moteur qui prend place dans le compartiment bas du bras. Ce moteur permettra l'inclinaison vers l'avant ou vers l'arrière du bras robotisé.

Ensuite nous devons assembler le deuxième segment du bras qui est composé de deux moteurs avec des réducteurs. L'ensemble est maintenu par deux lamelles externes vissées afin de réaliser un ensemble compact (figure 5).

Une fois l'assemblage terminé, l'engrenage d'un moteur entrera dans la roue dentée du 1er segment du bras, de manière à permettre la montée et la descente du 2ème segment (le moteur fait tourner l'engrenage sur lequel se trouve une pièce de la base).

L'autre engrenage s'enclenche dans celui de la partie haute de la pièce constituant la pince et la LED, de manière à permettre les mouvements de soulèvement et d'abaissement mais aussi un mouvement de flexion. Cette articulation rend encore plus « agile » le bras robotisé et permet l'exécution de mouvements précis similaires à ceux de la main. La pièce constituant la pince (que nous pourrions assimiler à la main) est assemblée avec le dernier moteur (voir la figure 6).

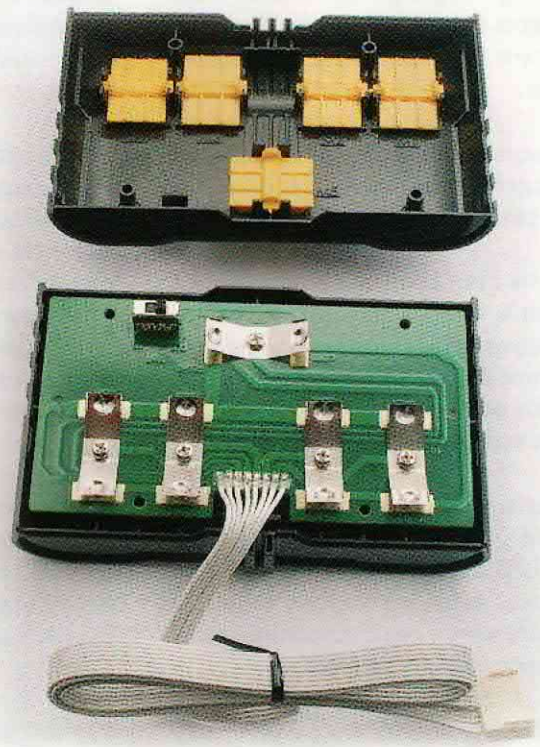
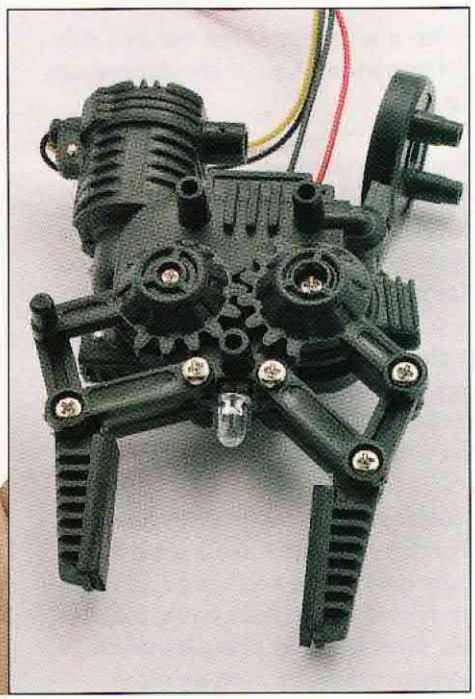


Figure 4 : ouvrez le boîtier de commande. À la base se trouve le circuit imprimé qui contient les contacts fixes. Les parties mobiles doivent être vissées de manière à recouvrir les parties fixes, mais sans les toucher.

Le réducteur doit être introduit dans le socle afin d'être fixé sur le sommet du 2ème segment du bras et inséré dans l'engrenage correspondant.

Figure 5 : assemblage de la partie supérieure du bras. Dans l'emplacement du moteur se trouve un engrenage couplé à la partie inférieure.

Figure 6 : ici la partie de la pince (« main ») contenant un moteur et des engrenages agissant sur les tiges de la pince.



Les engrenages permettant les mouvements de la pince doivent être insérés dans les emplacements correspondants.

Toujours au niveau de la pince, dans un logement approprié, est placée la LED blanche.

Les fils avec ceux du moteur doivent longer le 2ème segment du bras, puis le 1er segment jusqu'à atteindre le circuit imprimé où se situent les connexions. Les fils doivent être reliés aux connecteurs appropriés (voir la figure 7).

Notez que chaque connecteur femelle volant, qui correspond aux moteurs et à la LED, doivent être reliés aux bons endroits, sinon les boutons du boîtier de commande ne correspondront pas aux mouvements souhaités.

Les fils des deux moteurs du 2ème segment du bras doivent longer le 1er segment, puis doivent être correctement reliés au circuit imprimé.

Afin d'éviter que les différents fils finissent par entraver le mouvement du bras, nous devons les fixer à l'aide de clips fournis dans le kit.

Voilà, toutes les pièces sont assemblées, vérifiez si quelque chose ne manque pas. Si tout est correct, vous pouvez maintenant mettre à l'épreuve le bras robotisé.

Notez que l'assemblage complet doit correspondre à celui représenté en figure 8.

Pour faire fonctionner le robot, vous devez manœuvrer les commandes respectives, qui sont de simples commutateurs électromécaniques alimentant ou pas les moteurs.

Il s'agit de leviers qui n'ont pas de position stable, une fois relâchés ils retournent au repos et désactivent le circuit correspondant.

Testez toutes les commandes (marche/arrêt), il ne devrait pas y avoir de circuits qui restent allumés ou au repos en permanence. Vous pouvez laisser les piles dans le compartiment, la consommation est nulle au repos.

Le seul interrupteur à enclenchement est celui qui allume et éteint la LED blanche.

Tout d'abord, vous devez essayer de faire tourner le bras sur sa base. Pour cela manœuvrez le levier central, vers la droite pour faire pivoter la base vers la droite et vers la gauche pour faire pivoter la base vers la gauche.

Figure 7 : la partie arrière de la base contient le compartiment des piles, il est fermé par un couvercle où se loge le circuit imprimé sur lequel sont connectés les câbles provenant des moteurs et de la LED blanche.

Maintenant, abaissez l'intégralité du bras. Pour cela poussez vers l'arrière le 3ème levier en partant de la gauche, pour faire remonter le bras, poussez ce même levier vers l'avant.

Pour abaisser ou remonter le 2ème segment du bras, c'est-à-dire celui avec les deux moteurs, manœuvrez respectivement vers l'arrière ou vers l'avant le second levier de gauche.

Si vous voulez mouvoir la pince vers le bas ou vers le haut, agissez sur le levier de droite respectivement vers l'arrière ou vers l'avant.

Maintenant vous devez tester le mouvement de la pince. Pour fermer la pince, il suffit de pousser vers l'avant le 1er levier de gauche, et pour l'ouvrir faites basculer ce levier vers l'arrière.

Une fois que vous maîtrisez les mouvements de base, pourquoi ne pas essayer de faire quelque chose de plus complexe ?

Par exemple prendre un objet sur le plan de travail comme le fait un vrai robot industriel.

Donc, d'abord, manœuvrez le levier central vers la droite pour positionner le bras de sorte que la pince soit perpendiculaire à l'objet à prendre.

Ensuite abaissez le bras à l'aide du 3ème levier de gauche. Pour affiner le mouvement pliez le « coude », c'est-à-dire le bas du 2ème segment du bras, à l'aide du 2ème levier de gauche.

Agissez éventuellement sur la pince avec le 1er levier de gauche.

Figure 8 : le bras robotisé et le boîtier de commande entièrement assemblés.