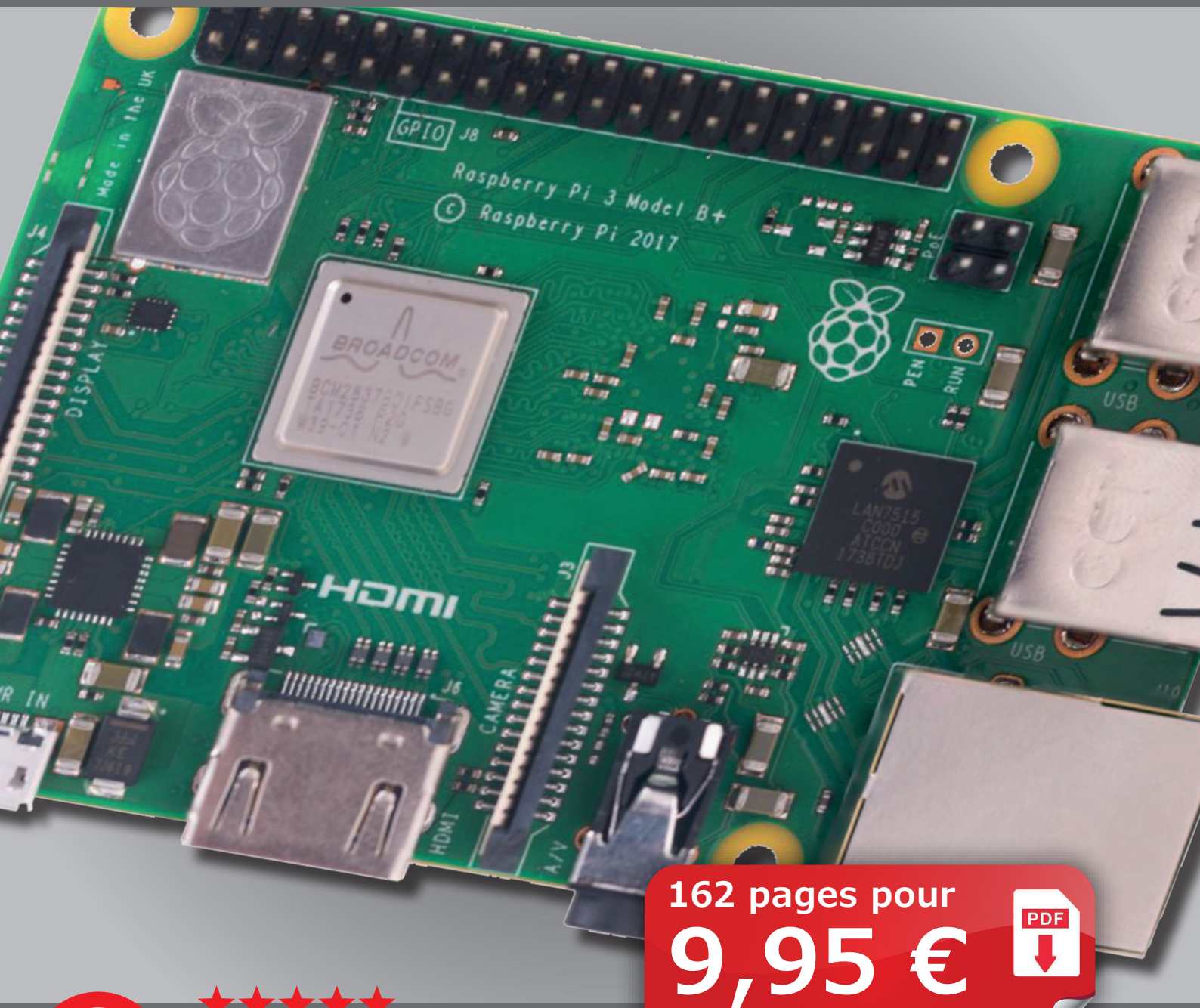
BEST-OF SERIES

COMPILATION 2 RASPBERRY PI

30 ARTICLES PARUS DANS LE MAGAZINE ELEKTOR



162 pages pour

9,95 €

PDF

7,50 € pour membres

elektor SELECT

DÉCOUVRIR CRÉER PARTAGER

ER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PAR
ARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER
R • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCO
ÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PA
PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER • DÉCOUVRIR • CRÉER • PARTAGER

canne télémétrique vocale

Raspberry Pi + ultrasons + synthèse vocale

= aide pour les malvoyants

Bera Somnath (Inde)

Ce télémètre vocal mesure et annonce la distance qui le sépare des objets. Il peut aussi déclencher une alarme sonore ou activer un vibreur lorsque la distance mesurée est inférieure à une certaine limite programmable. Vous avez un RPi et un capteur de distance à ultrasons ? Laissez-vous guider, et peut-être inspirer, car ce montage expérimental illustre parfaitement le potentiel d'une technologie embarquée, grand public, comme le RPi.

Un brin de compétences en design industriel devrait permettre de transformer ce projet en canne blanche électronique pour aveugles et malvoyants. Les alarmes se déclenchent en fonction du « niveau de danger » défini par l'utilisateur, et sont actionnées par des relais commandés via des E/S à usage général. Les sorties à relais peuvent activer des dispositifs vibreurs afin de fournir un retour tactile à l'utilisateur.

Pourquoi RPi ?

Windows est préféré pour les jeux, Linux pour les réseaux. De la même façon, le Raspberry Pi s'avère un des meilleurs choix pour les systèmes automatisés de petite taille où la consommation doit être aussi faible que possible. Une carte Arduino peut suffire dans certains cas, mais puisqu'il s'agit d'un simple microprocesseur, cette carte est plus adaptée à un rôle d'« esclave » que de « maître ». De son côté, le RPi peut simultanément exécuter des fonctions, surveiller et administrer des processus, le tout avec rapidité.

Matériel

Vous identifierez sans peine les principaux composants de la carte (**fig. 1**) : un capteur de distance à ultrasons HC-SR04 à 4 broches sur K1, un Raspberry Pi modèle A ou B avec le système d'exploitation Raspbian Wheezy, un écran LCD optionnel compatible HD 44780 de 2 x 16 caractères, quelques transistors, un convertisseur I²C-parallèle MC23008 de Microchip (IC1), et deux transistors FET pour la commande des relais RE1 et RE2, dont les contacts sont reliés aux borniers K2 et K3.

Le circuit est alimenté en +5 V par le connecteur micro-USB PWR IN du RPi. La plupart des signaux échangés entre le RPi et le circuit passent par les broches GPIO, et toutes les commandes utilisateur par le poussoir S1.

Notez que les transistors FET et le poussoir sont alimentés par le rail +3,3 V du RPi.

Pour la fonction sonar, le RPi envoie les impulsions par la ligne TRIG (GPIO25)

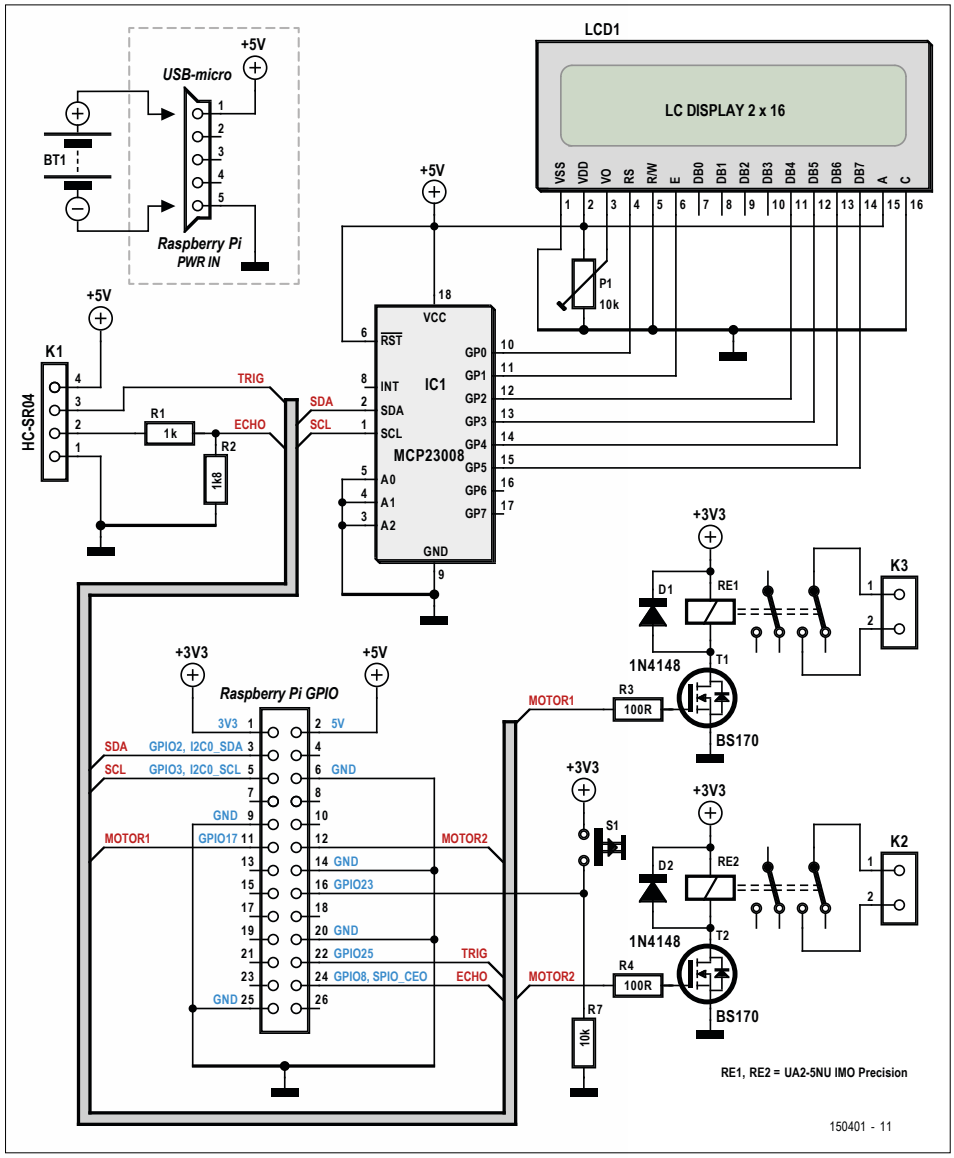


Figure 1. Le Raspberry Pi est si puissant et possède tellement de lignes GPIO qu'il n'a fallu que quelques composants supplémentaires pour concevoir la canne parlante. Le LCD est optionnel. Le menu et les valeurs affichées peuvent être utiles à la personne qui accompagne ou assiste l'utilisateur, et ils peuvent servir à paramétrer le dispositif.

Caractéristiques

- Canne télémétrique pour aveugles et malvoyants
- Alarme par buzzer/vibreur pour distance ≤ 1 m et > 2 m
- Transducteur à ultrasons avec résolution de 0,3 cm
- Distances lues par synthèse vocale, avec écart minimal de lecture réglable par logiciel (delta type : 2 à 5 cm)
- Commandé par Raspberry Pi
- Affichage des mesures sur LCD (optionnel)
- Code RPi téléchargeable gratuitement

Liste des composants

Résistances
R1 = 1 kΩ
R2 = 1,8 kΩ
R3, R4 = 100 Ω
R7 = 10 kΩ
P1 = ajustable de 10 kΩ, horizontal

Semi-conducteurs
IC1 = MCP23008-E/P, Microchip
(réf. 1439387, Newark/Farnell)
D1, D2 = 1N4148
T1, T2 = BS170

Divers
K1 = embase femelle, 4 voies, 90 °
K2, K3 = bornier pour CI à 2 voies,
pas de 5,58 mm
S1 = bouton à effleurement, 6 x 6 mm
RE1, RE2 = relais, DPDT, 5 V,
modèle UA2-5NU de IMO Precision Controls,
(réf. 1094048, Newark/Farnell)
LCD1 = LCD, alphanumérique, 2 x 16 car.
(réf. 120061-74, www.elektor.fr)
Capteur de distance à ultrasons
(réf. 140194-91, www.elektor.fr)
Carte de prototypage ELPB-NG
(réf. 150180-1, www.elektor.fr)

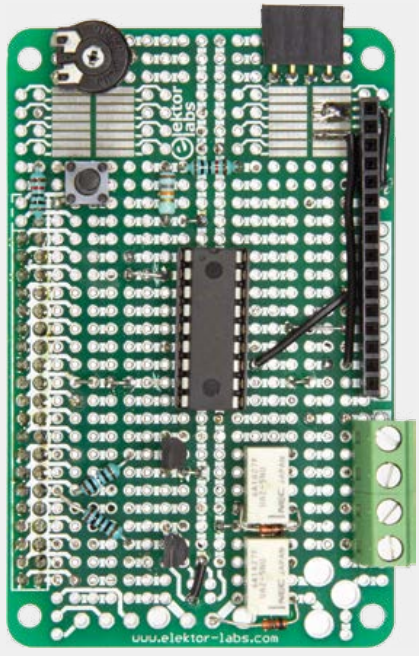


Figure 2. La carte de prototypage Elektor ELPB-NG utilisée pour assembler le projet.

et écoute la réponse sur la ligne ECHO (GPIO8/SPIO_CEO). Le temps écoulé entre une salve émise et celle reçue sert à calculer la distance entre le transducteur à ultrasons et l'objet réfléchissant ; la vitesse du son est considérée comme constante. Le signal a une fréquence d'environ 40 kHz. La proximité d'une personne équipée d'un module similaire (ou d'une voiture avec un radar de stationnement !) peut provoquer des interférences. Je travaille à une solution. La portée du module HC-SR14 est d'environ 2 à 5 cm, pour une précision d'environ 0,3 cm et un angle d'ouverture de 15 °.

Un petit haut-parleur est relié à la sortie audio du RPi, mais on peut aussi utiliser un casque ou des oreillettes. L'écran LCD est optionnel.

Logiciel

Nous vous conseillons d'avoir le code RPi du projet sous les yeux pour poursuivre la lecture de l'article (code disponible en [1]). Le **listage 1** est la routine qui traite l'écho des ultrasons.

Synthèse de la parole

Vous aurez à installer un synthétiseur vocal si vous souhaitez adapter le projet à votre langue et/ou l'utiliser avec une voix particulière. On trouve de nombreux logiciels de synthèse de la parole sur l'internet, le plus polyvalent étant à mes yeux *espeak*, gratuit qui plus est. Citons également Festival, Pico et Cepstral [2]. espeak utilise une méthode de synthèse vocale différente de celle des autres synthétiseurs texte-parole *open source* et possède un son unique. Le rendu n'est peut-être pas très naturel ou « lisse », mais selon moi les voix ont une élocution suffisamment distincte pour être facilement compréhensibles. Pour l'installer :

```
$> sudo apt-get update
$> sudo apt-get upgrade
$> sudo apt-get install alsa-utils
```

Ouvrez ensuite le fichier */etc/modules* avec nano, et ajoutez la ligne suivante :

```
$> sudo nano /etc/modules
snd_bcm2835
```

Enregistrez le fichier (Ctrl+o), quittez nano (Ctrl+x), puis redémarrez le RPi :

```
$> sudo reboot
```

Installez les programmes suivants :

```
$> sudo apt-get install mplayer
$> sudo apt-get install espeak
$> sudo apt-get install espeak-gui
```

espeak est maintenant installé. Pour consulter son manuel :

`$> man espeak` ou `$> espeak –h`

La sortie audio du RPi n'est pas assez puissante pour attaquer autre chose qu'un casque. Afin de tester espeak, il faut donc relier un amplificateur à la prise audio du RPi. Après avoir lancé *espeak-gui*, on peut attribuer différents critères à la voix du synthétiseur, dont la hauteur et le volume. Voici quelques exemples :

```
$> espeak “Bonjour, bienvenue chez
   Elektor” // voix masculine
$> espeak -vfr+f3 «Bonjour,
   bienvenue chez Elektor” // voix
   féminine
$> espeak -vfr+f3 -k9 -s150 -a200
   «Bienvenue chez Elektor” // voix
   aiguë, volume max., féminine
```

Ce projet a également besoin du module Python *rpi.gpio* [3]. Téléchargez la dernière version et installez-la.

Une canne, pas un perroquet

Le Raspberry Pi y laisserait tout son jus de framboise à force de répéter encore et toujours les mêmes valeurs. Le programme contient donc un tableau dans lequel sont stockées les vingt dernières distances mesurées. Lorsque la différence entre la dernière distance et la valeur en cours est supérieure à la différence par défaut (`sv` dans le code, 2 cm ici), le RPi énonce la distance mesurée, sinon il reste muet. La constante `sv` est déclarée au début du code, vous pouvez la modifier en fonction de vos besoins.

1 2 3, test micro

En utilisant un amplificateur relié à la prise audio ou un casque, lancez le programme Python avec les droits d'administrateur :

```
$> sudo python ultra3.py
```

La première initialisation prend un certain temps. Le programme énonce d'abord le message de bienvenue, puis entre dans une boucle de mesure des distances. Les broches GPIO 25 et 8 sont reliées à deux relais de 5 V dont le rôle est d'émettre chacun un signal lorsqu'un objet est

détecté : un pour une distance > 200 cm (pas de danger), un autre pour une distance < 100 cm (danger). Pour aider les aveugles, les relais peuvent être connectés à deux petits vibreurs récupérés dans des vieux téléphones portables.

Enfin il faut modifier le fichier */etc/rc.local* pour que le programme se lance au démarrage du RPi :

```
$> sudo nano /etc/rc.local
```

Ajoutez à la fin du fichier, avant « exit », la ligne suivante :

```
sudo python /home/pi/ultra4.py &
```

Le code associé à la canne parlante est intéressant à bien des égards. Je vous invite à l'étudier, l'expérimenter, le compléter, l'optimiser ou l'améliorer. Et bien sûr à partager vos résultats sur www.elektormagazine.fr/labs.

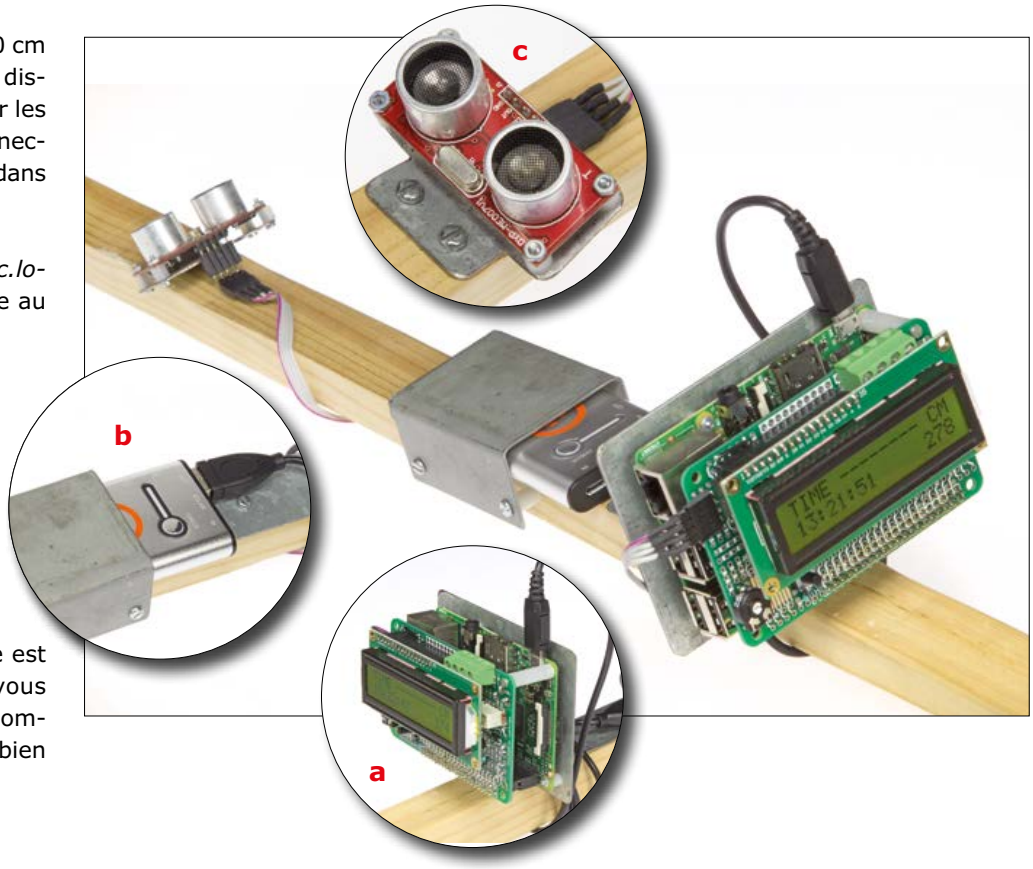


Figure 3. Montage expérimental sur un bâton de bois. **a.** Le RPi, la carte d'extension et le LCD solidement fixés en un seul bloc à l'extrémité de la canne. **b.** La batterie insérée dans un support métallique fixé à peu près au milieu de la canne. Ici nous avons utilisé un modèle Li-ion à recharge solaire. **c.** Le sonar positionné entre le quart et le tiers de la canne. Sa position a été déterminée expérimentalement. Elle devrait dépendre des souhaits et capacités physiques de l'utilisateur, ainsi que de l'angle d'ouverture du module.

Construction

Puisque vous êtes en train de lire Elektor, il y a fort à parier que l'assemblage des composants électroniques vous posera moins de problèmes que le montage mécanique, sans parler d'éventuelles tentatives de « design industriel ou ergonomique ». Quoi qu'il en soit, le soudage est ici inévitable. Plutôt que de dessiner un circuit imprimé pour ce projet, le labo d'Elektor a utilisé sa carte de prototypage ELPB-NG, disponible dans l'e-choppe [4]. La carte assemblée se fixe sur le RPi. Voilà pour l'électronique. Reportez-vous à la **liste des composants** et à la **figure 2**, elles disent et montrent l'essentiel.

Le labo d'Elektor a utilisé les matériaux à sa disposition pour construire la canne et ainsi prouver la faisabilité du projet. Vous pouvez voir le résultat sur la **figure 3**, et chaque élément séparé sur les figures **3a** (RPi, carte d'extension, LCD, pas de haut-parleur), **3b** (batterie) et **3c** (télémètre à ultrasons).

On peut rendre la canne encore plus pratique en plaçant les vibreurs de façon à ce que l'utilisateur puisse associer leur vibration à la distance mesurée et au niveau de risque : ≤ 1 m (danger) ou > 2 m (danger relativement faible). On peut fixer le RPi et la carte ELPB NG un peu plus bas que la poignée, et arrimer

Lancement automatique du script

Voici la recette :

- Copier le dossier *rpi_distance_meter* dans le répertoire *home/pi* du Raspberry Pi en utilisant une clé USB ou un programme comme WinSCP.
- Lancer le terminal et entrer `cd rpi_distance_meter`.
- Rendre le script de lancement exécutable avec la commande `sudo chmod 755 launcher.sh`.
- Revenir au répertoire d'accueil (*/home*) en tapant `cd`.
- Créer un dossier *logs* avec la commande `mkdir logs`.
- Lancer *crontab* avec `sudo crontab -e`. Crontab est un processus qui s'exécute en arrière-plan et permet d'ordonnancer des tâches.
- Dans la fenêtre *crontab*, entrez `@reboot sh /home/pi/rpi_distance_meter/launcher.sh >/home/pi/logs/cronlog 2>&1`. Le script sera lancé au démarrage.
- Redémarrez le RPi avec `sudo reboot` pour vérifier que le script se lance bien. Si ce n'est pas le cas, cherchez les erreurs dans le fichier *log* du dossier *logs*.

Le script se lance automatiquement au démarrage du RPi. Il émet un message de bienvenue, puis commence son cycle de mesure et d'annonce des distances. Le RPi ne parle que si la distance mesurée dépasse la distance précédente d'une valeur prédéfinie. Pour l'éteindre, appuyez sur le bouton. Le RPi annonce qu'il va se mettre hors tension.

Listage 1. Extrait du code RPi pour la canne parlante. Le code complet est en [1].

```
#print "Ultrasonic Measurement"
GPIO.setup(GPIO_TRIGGER,GPIO.OUT)  # Trigger
GPIO.setup(GPIO_ECHO,GPIO.IN)      # Echo

while shutdown == 0:
    GPIO.output(GPIO_TRIGGER, False)
    time.sleep(0.1)
# Send 10us pulse to trigger
    GPIO.output(GPIO_TRIGGER, True)
    time.sleep(0.00001)

    GPIO.output(GPIO_TRIGGER, False)
    start = time.time()
    while GPIO.input(GPIO_ECHO)==0:
        start = time.time()
    while GPIO.input(GPIO_ECHO)==1:
        stop = time.time()
    elapsed = stop-start
#multiplied speed of sound (cm/s)
    distance = elapsed * 34300 / 2
    lcd.setCursor(0,0)
    lcd.message("TIME -------- CM")
    lcd.setCursor(0,1)
    buf = "%.0f" % distance
    if distance>99:
        lcd.message(datetime.now().strftime('%H:%M:%S     ')+buf)
    if distance<=99:
        if distance>9:
            lcd.message(datetime.now().strftime('%H:%M:%S      ')+buf)
        if distance<=9:
            lcd.message(datetime.now().strftime('%H:%M:%S       ')+buf)

    print " Distance : %.1f cm" % distance

    if distance<=200 and distance>100:
        GPIO.output(motor1,1)
        GPIO.output(motor2,0)

    elif distance<=100:
        GPIO.output(motor1,0)
        GPIO.output(motor2,1)

    else:
        GPIO.output(motor1, 0)
        GPIO.output(motor2, 0)

    # if the difference between the previous distance and the current
    # distance is greater than a set value
    if abs(prevDistance-distance)>=sv:
        exitCode = subprocess.call(["espeak","-ven+f3","-a200",
        "-s120","%.0f" % distance])

    prevDistance = distance
```

la batterie sous les cartes (plomb-acide, Li-ion, NiCd). Cette position répond à la plupart des usages. La sonde sonar doit être placée un peu en dessous du milieu de la canne pour que les objets situés au ras du sol, ceux susceptibles de faire trébucher l'utilisateur, soient détectés correctement.

La longueur de la canne est critique et devrait être adaptée aux aptitudes physiques et à la taille de l'utilisateur. Comme je l'ai dit plus haut, la synthèse vocale déclenchée par une mesure dépend de la valeur préréglée `sv` (*set value*). Si la personne utilise un casque dont le son ne peut pas être réglé, le volume peut être ajusté dans le programme `ultra3.py` en modifiant l'argument `-a` de la commande espeak ; la valeur par défaut est `-a200`, c'est-à-dire le volume maximal. ⏮

(150401 – version française : Hervé Moreau)

Liens

[1] Logiciel du projet : www.elektormagazine.fr/150401

[2] Synthétiseur vocal Cepstral : www.cepstral.com/raspberrypi

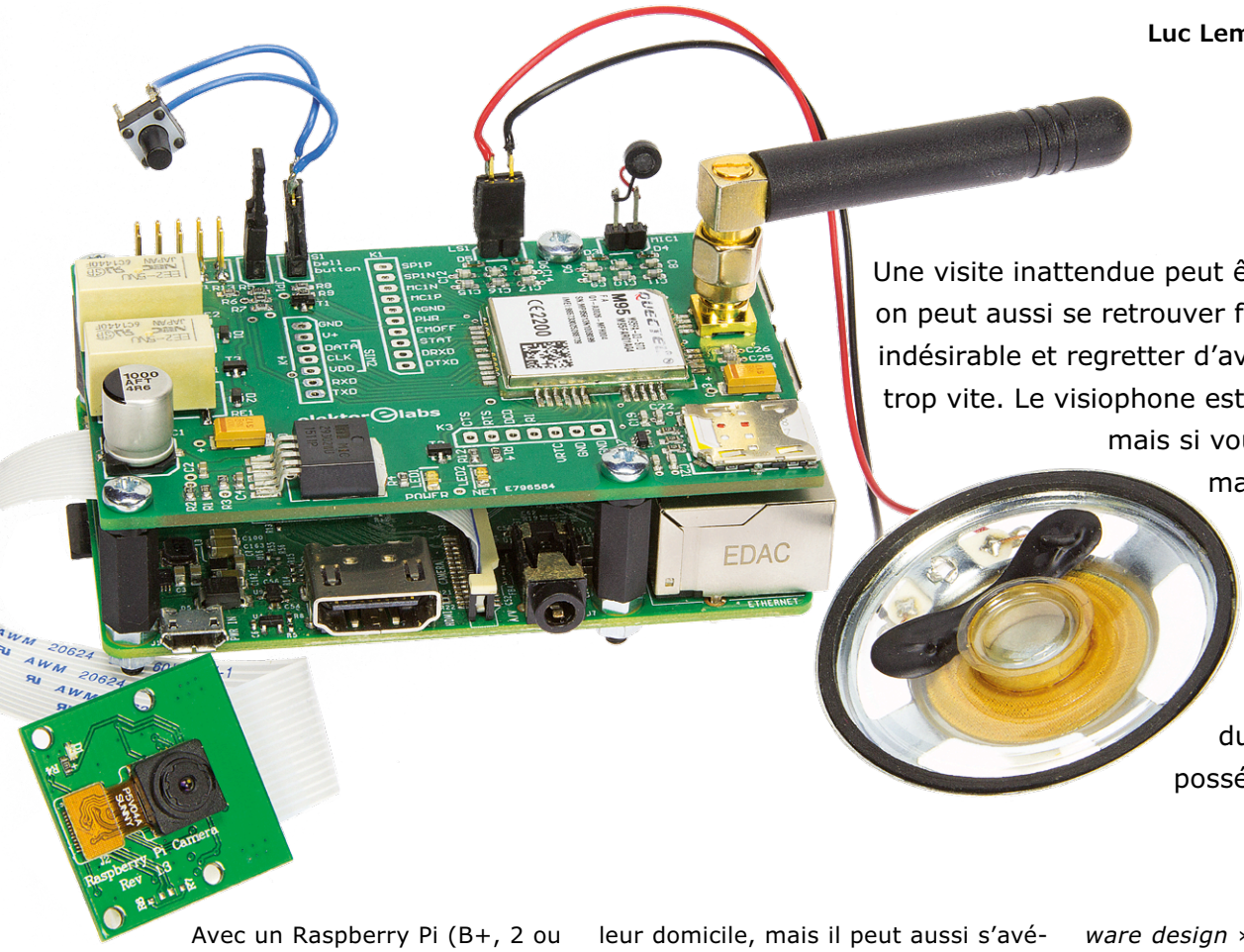
[3] Module RPi-GPIO : https://sourceforge.net/projects/raspberry-gpio-python/

[4] Carte de prototypage ELPB-NG (*Elektor Labs Prototyping Board Next Generation*), réf. 150180-1 : www.elektor.fr

judas connecté avec Raspberry Pi

communiquer avec un visiteur via son ordiphone

Luc Lemmens (labo d'Elektor)



Une visite inattendue peut être agréable, mais on peut aussi se retrouver face à une personne indésirable et regretter d'avoir ouvert la porte trop vite. Le visiophone est une solution, mais si vous n'êtes pas à la maison... Avec notre projet, vous pourrez communiquer avec votre visiteur, et même lui ouvrir la porte à distance ; du moins, si vous possédez un ordiphone.

Avec un Raspberry Pi (B+, 2 ou 3, peu importe), un module caméra RPi, et la carte *HAT* GSM pour RPi conçue par notre labo, nous réalisons un interphone qui prend une photo du visiteur dès qu'il sonne. La photo est envoyée dans un MMS vers votre ordiphone, ce qui vous permet de voir qui est à la porte. Vous pouvez ensuite appeler votre « portier connecté » : le module GSM prendra automatiquement l'appel, et vous communiquerez avec le visiteur par microphone et haut-parleur interposés. Nous avons aussi prévu la possibilité d'actionner une gâche électrique, suite à l'envoi d'un SMS à l'interphone. Le SMS contient un mot de passe, et le numéro de l'appareil qui a envoyé le message est aussi contrôlé, pour plus de sécurité.

Le système a été avant tout pensé et conçu pour les personnes absentes de leur domicile, mais il peut aussi s'avérer utile dans des situations où on n'est pas en mesure d'aller voir qui a sonné. Et pourquoi pas pour « filtrer » vos visiteurs ? Il y a des jours où on n'a pas envie de recevoir n'importe qui...

Cette idée nous est venue après avoir fini le projet de carte de liaison GSM [1] qui repose sur le module GSM M95 de Quectel. Nous avions les bases nécessaires pour notre interphone, mais il semblait trop difficile d'en faire un ensemble compact. Nous avons donc préféré réaliser une carte *HAT* (*Hardware Attached on Top*) pour RPi avec le matériel additionnel nécessaire à l'interphone ; examinons-en maintenant le schéma.

Schéma

L'essentiel du schéma provient de la documentation de Quectel, « *M95 hardware design* » [2]. Le cœur du circuit (**fig. 1**) est constitué du module GSM M95 (MOD1) avec son antenne, et d'un connecteur pour carte SIM (SIM1). Les résistances R15 à R17, la barrette de diodes D7, et les condensateurs C19 à C22, constituent un dispositif antiparasites et de protection contre les décharges électrostatiques, pour l'interface de la carte SIM. Les condensateurs C6 à C17 et les diodes D3 à D6 remplissent la même fonction pour l'entrée du microphone à électret et la sortie de l'amplificateur audio du M95.

L'UART du M95 est connecté au RPi via K5, la communication se fait avec des commandes *AT*. Le diviseur de tension constitué par R6 et R7 adapte le niveau du signal *TxD* aux 3,3 V de l'entrée du RPi. Nous aurions pu alimenter la carte

GSM en 3,3 V, pour éviter la différence de niveaux logiques, mais cela provoque un problème inattendu : bien que le M95 soit spécifié pour une tension d'alimentation de 3,3 à 4,6 V, il envoie une alerte en continu via son UART dès que l'on approche à moins de 100 mV du minimum ou du maximum. Outre le fait que cette alerte gêne le décodage d'autres messages transmis par le M95 au RPi, le M95 peut aussi s'éteindre soudainement. Quectel n'explique pas les raisons de ce comportement, nous avons donc choisi d'alimenter le circuit en 4,4 V via un MIC29302 (IC1), même si cette tension n'est pas usuelle.

LED2 clignote dès que le circuit est alimenté. Le nom de cette sortie, « *net-light* », pourrait laisser croire que le module s'est connecté à un réseau mobile, mais ce n'est pas le cas. Le M95 se met en fait en mode *stand-by*, et une impulsion à l'entrée « *pwrkey* » l'allume ou l'éteint.

Le bouton-poussoir S1 est un bouton de sonnette ordinaire. Il commande le relais

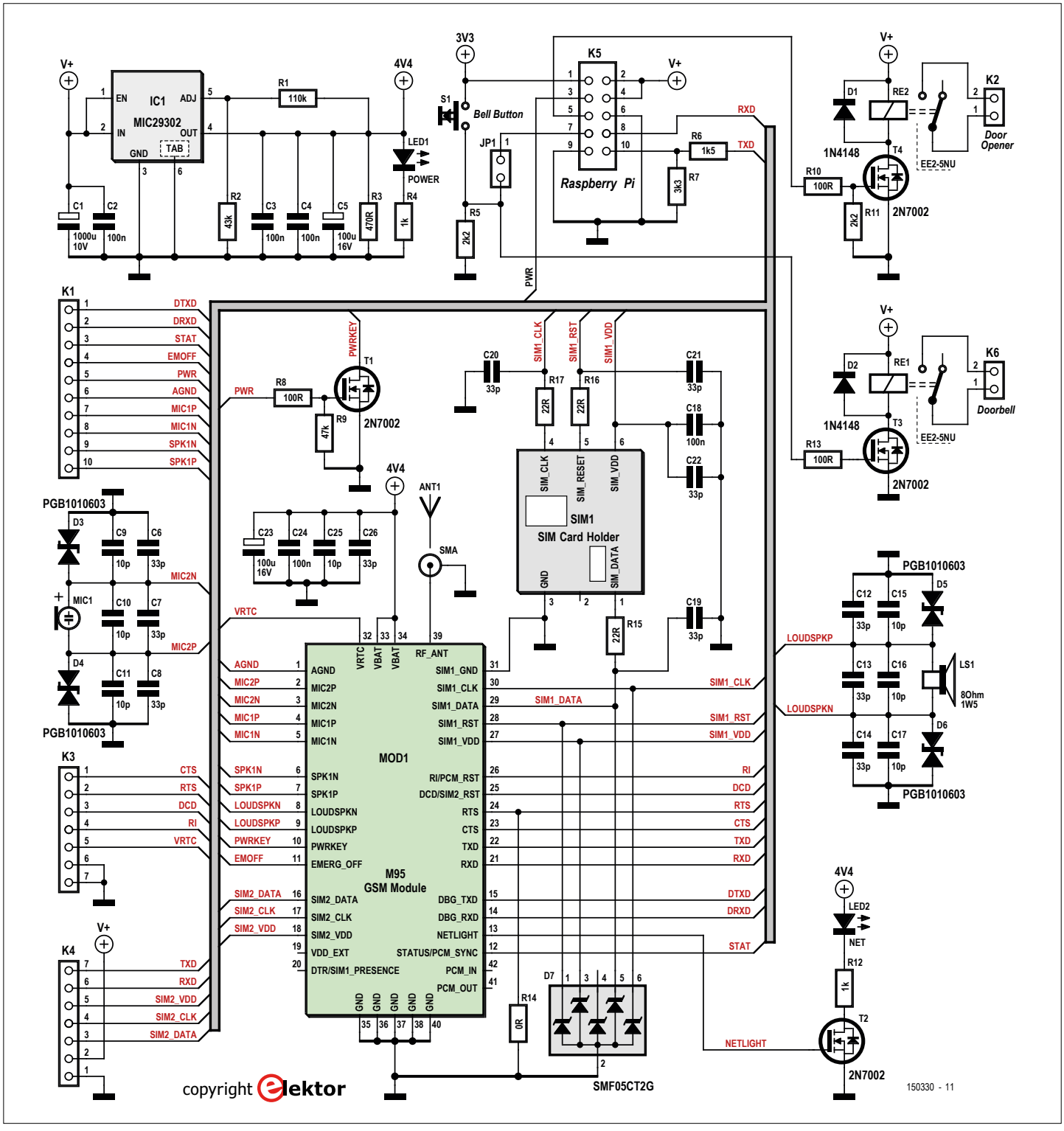


Figure 1. Les composants les plus importants sont le module GSM M95 et le connecteur pour carte micro-SIM.

RE1, qui peut être utilisé pour alimenter (en courant alternatif) une sonnette existante. Le cavalier JP1, que l'on peut remplacer par un interrupteur marche/arrêt, détermine si le signal de sonnerie est envoyé vers le RPi, auquel cas le processus d'envoi d'une photo démarre. L'envoi d'un MMS n'est pas gratuit, et si on est à la maison, on peut en principe se contenter de la sonnette.

RE2 permet la connexion d'une gâche électrique, qui sera actionnée par le RPi via T4 lors de la réception d'un SMS contenant un mot de passe déterminé, et si le numéro de l'appareil envoyant le message est reconnu.

Nous ne sommes pas tout à fait convaincus de l'utilité de cette dernière option : il faudrait de toute manière pas mal de matériel pour sécuriser le relais, et empêcher qu'un malfrat ne parvienne à l'actionner ou à le contourner, et à ouvrir la porte. Il appartient à l'utilisateur de choisir quelles mesures de sécurité il appliquera ou non.

Liste des composants

Résistances :

(CMS 0603, 1% / 0,1 W)
R1 = 110 kΩ
R2 = 43 kΩ
R3 = 470 Ω
R4, R12 = 1 kΩ
R5, R11 = 2,2 kΩ
R6 = 1,5 kΩ
R7 = 3,3 kΩ
R8, R10, R13 = 100 Ω
R9 = 47 kΩ
R14 = 0 Ω
R15, R16, R17 = 22 Ω

Condensateurs :

C1 = 1000 µF / 10 V, CMS radial
C2, C3, C4, C18, C24 = 100 nF, X7R, CMS 0805
C5, C23 = 100 µF / 16 V, CMS 2312
C6, C7, C8, C12, C13, C14, C19, C20, C21, C22, C26 = 33 pF, X7R, CMS 0805
C9, C10, C11, C15, C16, C17, C25 = 10 pF, X7R, CMS 0805

Semi-conducteurs :

D1, D2 = 1N4148, SOD-323
D3, D4, D5, D6 = PGB1010603MR, CMS 0603
D7 = SMF05CT2G, SC-88
LED1, LED2 = LED verte, 50 mcd, 2,1 V @ 20 mA, CMS 0805
T1, T2, T3, T4 = 2N7002, SOT-23
IC1 = MIC29302WU TR, TO-263

Divers :

Mod1 = module GSM Quectel M95
SIM1 = connecteur micro-SIM (Molex 78723-1001)
ANT1 = connecteur SMA pour circuit imprimé (Molex 73391-0070)
+ antenne hélicoïdale (RF Solutions ANT-GHEL2R-SMA)
K1 = barrette femelle à 10 broches, pas de 2,54 mm (en option)
K2, K6 = barrette femelle à 2 broches, pas de 2,54 mm (en option)
K3, K4 = barrette femelle à 7 broches, pas de 2,54 mm (en option)
K5 = barrette femelle à 2×5 broches à longues pattes (Samtec ESQ-105-14-G-D)
S1 = barrette mâle à 2 broches, pas de 2,54 mm, pour bouton de sonnette
JP1 = barrette mâle à 2 broches, pas de 2,54 mm, avec cavalier
LS1 = barrette mâle à 2 broches, pas de 2,54 mm, pour le haut-parleur (8 Ω / 1,5 W)
Mic1 = barrette mâle à 2 broches, pas de 2,54 mm, pour le microphone à électret (ABM-715-RC)
Re1, Re2 = relais CMS, DPDT, 2 A, 5 V_{DC} (Kemet EE2-5NU)
Raspberry Pi B+, 2 B ou 3 B
Module caméra pour Raspberry Pi (rev 1.3)
Circuit imprimé réf. 150400-1 ou circuit imprimé avec composants CMS montés réf. 150400-71 (www.elektor.fr)

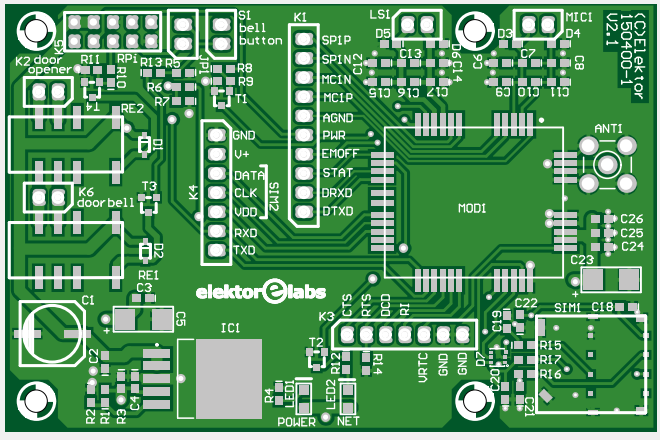


Figure 2. Le circuit imprimé de la carte *HAT* pour RPi comporte essentiellement des CMS.

Circuit imprimé

Le circuit imprimé est en **figure 2**. Comme le module GSM n'est disponible qu'en version CMS, nous avons décidé de n'utiliser que des composants CMS, à l'exception des connecteurs – dont celui de l'antenne. Cela risque de compliquer la tâche des amateurs les moins expérimentés, mais qu'ils se rassurent ! Elektor propose le circuit imprimé avec tous les CMS montés (150400-71). Les électroniciens chevronnés se procureront le circuit imprimé nu, ou le réaliseront eux-mêmes ; le dessin est disponible en [4]. Certains connecteurs sont optionnels, c'est mentionné dans la liste des composants. Attention : il faut insérer K5 côté pistes du circuit imprimé et ensuite souder ses pattes côté composants (**fig. 3**). Ce connecteur sera ensuite enfiché dans le connecteur GPIO du RPi. Avant de souder, vérifiez s'il faut ajuster la hauteur du connecteur par rapport au RPi. Il faut que la carte *HAT* reste éloignée de la prise USB du RPi.

Préparation du RPi

Avec la version actuelle de *Raspbian*, appelée *Jessie*, il est relativement simple d'adapter le RPi à notre montage. Il faut d'abord télécharger l'image de la version la plus récente du système d'exploitation sur le site du RPi [3] (version 4.4 du noyau au moment de la rédaction de cet article). Copiez le fichier image (extension IMG) sur une carte SD, à l'aide de *Win32 DiskImager* ou d'un programme similaire. Ensuite démarrez le RPi avec cette carte SD, ouvrez une fenêtre de terminal et modifiez la configuration après saisie de la ligne suivante :

```
sudo raspi-config
```

On sélectionne l'option « *6. Enable Camera* » pour connecter la caméra RPi. Ensuite on passe à « *9. Advanced Options* », puis « *A7 Serial* », et « *No* », pour mettre hors service l'accès au *login-shell* (première instruction à s'exécuter lors d'une ouverture de session) via le port série. On termine avec « *Finish* », et on redémarre le RPi.

N. B. : avec d'anciennes versions de *Raspbian*, il faut installer la caméra manuellement, et mettre la console série hors service dans les fichiers de démarrage. Il y a de nombreux exemples et des explications sur l'internet.

Il faut cependant effectuer une modifica-

tion importante dans le script en *Python* du portier connecté : à la ligne 45, il faut supprimer le caractère « # » avant « *port.open* » ! Il y a un certain temps que notre prototype est prêt, et lorsque nous avons refait toute l'installation à partir de la dernière version de *Raspbian* en vue de cet article, le script nous a renvoyé le message d'erreur « *Port already open* » à la ligne 45. Il semble qu'avec la dernière version du logiciel la commande « *open* » soit superflue.

Connexion du matériel

Le Raspberry Pi est fin prêt, mais il reste encore pas mal de liaisons à établir avant de pouvoir passer à la phase de test.
Le plus simple est de commencer par la caméra RPi, la limande (*Flex Flat Cable – FFC*) peut passer sous la carte *HAT*.
Notre carte s'adapte parfaitement au RPi. Il faut à nouveau vérifier que les entretoises sont de longueur suffisante, et qu'il n'y a aucun contact entre le dessous de notre circuit imprimé – surtout les soudures du connecteur d'antenne – et les prises USB. On peut ensuite assembler les deux cartes avec boulons et écrous (**fig.4**).
Connectez ensuite un bouton-poussoir à S1 pour faire office de bouton de sonnette, et placez le cavalier sur JP1. Raccordez le microphone à MIC1, le haut-parleur à LS1, et l'antenne à ANT1. Il nous faut encore insérer une carte micro-SIM dans SIM1, mais ne courez pas tout de suite en acheter une...

Carte SIM et MMS

Le module GSM M95 ne fonctionnera qu'avec une carte SIM sur laquelle il y a suffisamment de crédit (ou un abonnement, bien sûr). Nous avons utilisé deux cartes SIM prépayées différentes, et un avertissement s'impose : comparez bien avant votre achat ! Les MMS sont apparemment peu utilisés, et les opérateurs ne mentionnent donc pas clairement les tarifs de ce service. D'après nos recherches, le coût d'un MMS est fixe chez la plupart des opérateurs, pour autant que l'annexe, une photo au format JPEG dans notre cas, ne dépasse pas 300 Ko. La taille de ce fichier est également importante dans notre application, puisqu'elle conditionnera le temps de transmission du message. Si vous mettez trop de temps à répondre au coup de sonnette, il y a des chances que votre visiteur n'attende pas ! Au labo, il fallait

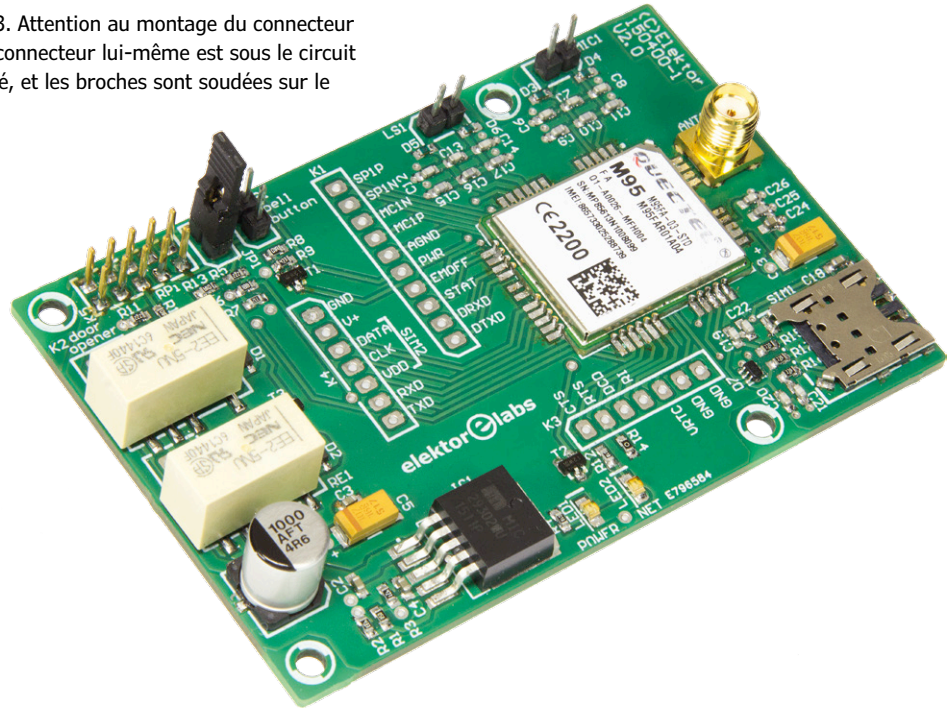


Figure 3. Attention au montage du connecteur K5 ; le connecteur lui-même est sous le circuit imprimé, et les broches sont soudées sur le dessus.

entre 20 et 30 s pour recevoir le message sur l'ordiphone, auxquelles il faut encore ajouter le temps de réaction. Conclusion : le système n'aura guère d'utilité avec des visiteurs pressés...

Script en Python et test initial

Après avoir configuré le RPi et installé les bibliothèques *Python*, nous sommes prêts à exécuter le script de gestion de l'interphone. Ce script en *Python* est téléchargeable en [4] : « *MMSautoSend.py* » ; il faut aussi le reconfigurer avant de pouvoir passer aux tests. Il est préférable de connecter un moniteur, un clavier, et éventuellement une souris pendant cette phase, cela facilitera le travail ; nous n'aurons plus besoin de ces accessoires par la suite.
Au début du script, quelques constantes sont définies, il faut les modifier.

- « *ThisNumber* » : numéro d'appel de la carte SIM du M95.
- « *ThatNumber* » : numéro d'appel de l'ordiphone vers lequel le MMS est envoyé.

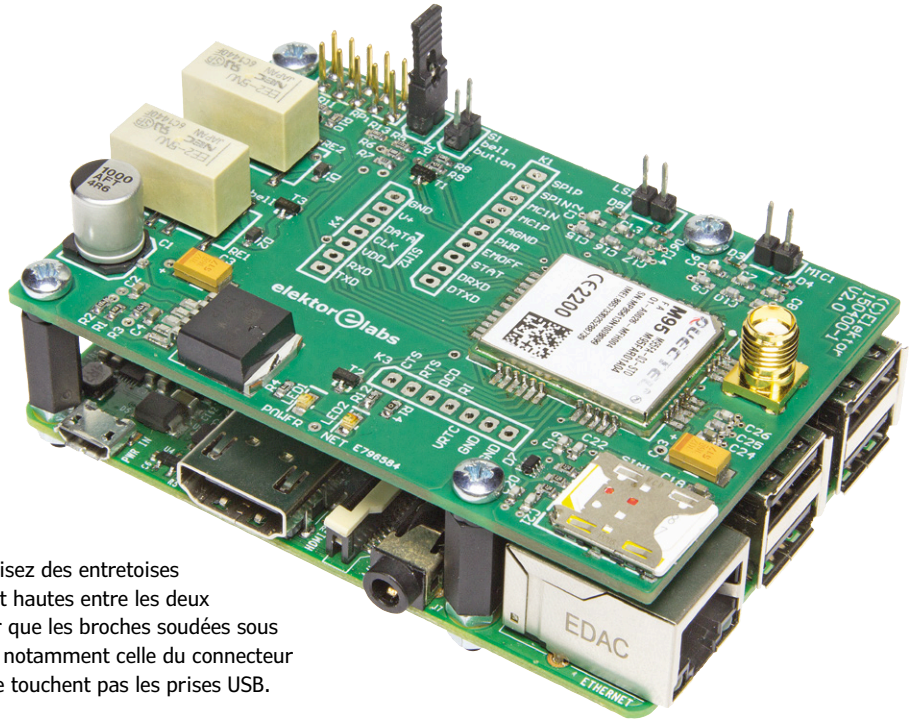
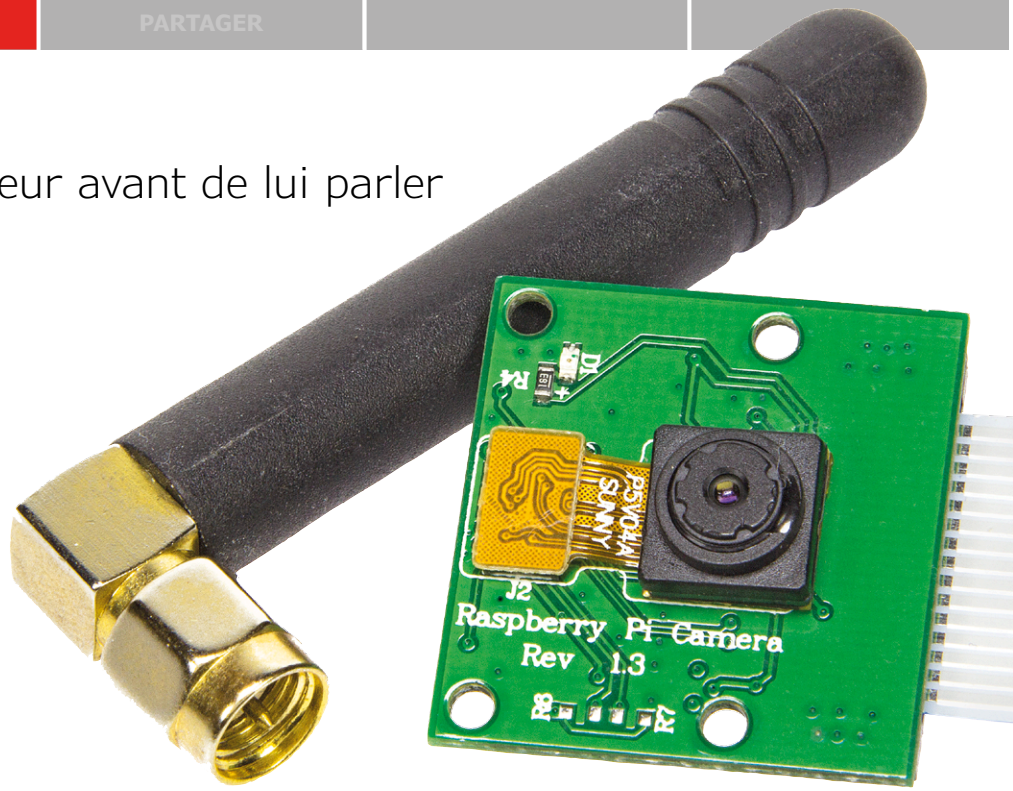


Figure 4. Utilisez des entretoises suffisamment hautes entre les deux circuits, pour que les broches soudées sous la carte *HAT*, notamment celle du connecteur d'antenne, ne touchent pas les prises USB.

le son et l'image, identifiez votre visiteur avant de lui parler



Données relatives à l'opérateur de la carte SIM du M95 pour l'emploi du service MMS :

- *APN* : *Access Point Name*
- *MMSC* : *Multimedia Messaging Service Centre*
- *MMSproxy* : *Multimedia Messaging Service proxy server*
- *MMSport* : *Multimedia Messaging Service port number*

Vous trouverez ces données sur le site de l'opérateur. Dans le script originel, nous avons repris les données des deux opérateurs néerlandais avec les cartes SIM desquels nous avons effectué les tests au labo.

Nous sommes maintenant fin prêts pour le test initial. Ouvrez une fenêtre de terminal RPi, et tapez la commande `sudo su` ; c'est nécessaire pour obtenir les droits d'accès à la bibliothèque Python du bus GPIO, autrement dit pour pouvoir gérer les entrées/sorties du RPi. Nous démarrons ensuite le script avec

```
python MMSautoSend.py.
```

Le script est simple à comprendre. Le RPi envoie des commandes *AT* au module GSM, suivies à chaque fois de « *time.sleep* », pour laisser le temps au M95 d'exécuter les commandes. Il y a d'abord quelques configurations, dont les paramètres pour la caméra, la sensibilité du microphone et le volume du haut-parleur ; vous pourrez les modifier plus tard en fonction de l'environnement de l'interphone.
Nous avons désactivé la saisie du code PIN des cartes SIM utilisées, pour plus de facilité ; il suffit d'insérer les cartes dans un téléphone mobile et de supprimer la protection par code PIN. Si vous souhaitez conserver le code PIN, il y a dans le script un bloc de quatre lignes (en commentaires) qui permettent d'entrer le code de la carte utilisée. Il faudra aussi supprimer le « *#* » devant « *port.write* » et « *time.sleep* ».

Ensuite le script entre dans une boucle sans fin, que l'on interrompra par une pression sur les touches « CTRL+C » pour quitter le script. En fonctionnement normal, le RPi vérifie en permanence si la sonnette est actionnée, ou si un SMS est reçu pour l'ouverture de la porte.
Lors de la réception d'un SMS, le script vérifie d'abord le numéro appelant ; si ce numéro est correct (« *ThatNumber* »), le contenu est ensuite comparé au mot de passe qui a été entré comme constante « *Very_Secret* ». Si c'est le cas, le relais RE2 est excité durant une seconde ; cette durée peut être modifiée dans le script si nécessaire.

Mais n'oublions pas qu'une autre action conditionne la réception du SMS : il faut qu'un visiteur ait sonné à la porte ! Le RPi prend alors une photo, prépare un MMS, et l'envoie à votre ordiphone. En sus de la photo, un petit texte est aussi transmis, avec entre autres le numéro d'appel de l'interphone dans un lien hypertexte. À vous de décider si vous voulez parler au visiteur, en utilisant le lien hypertexte ; le M95 répondra automatiquement à l'appel, et vous mettra en communication avec ce visiteur. Dès que la conversation est terminée, le M95 « replace le combiné sur son support », et se met en attente d'un SMS ou d'une nouvelle pression sur le bouton de la sonnette.

Montage

Les aspects mécaniques du montage final constituent sans doute le plus grand défi de ce projet. L'idéal – et sans doute le plus sûr – est de ne mettre à l'extérieur que le microphone, le haut-parleur et la caméra ; le reste de l'électronique sera placé à l'abri à l'intérieur. La caméra peut aussi effectuer ses prises de vue à travers une fenêtre ou un vasistas.
Comme nous l'avons mentionné, quelques paramètres de la caméra peuvent être modifiés dans le script. L'orientation et la luminosité seront adaptées en fonction de l'environnement. La résolution sera la plus basse possible, afin de minimiser la taille du fichier JPEG – et donc le temps de transmission. Pendant les essais, mettez l'envoi de MMS hors service dans le script, vous économiserez de l'argent ; vous pouvez juger de la qualité de la photo en ouvrant le fichier « test1.jpg » stocké dans le RPi.

(150400 – version française : Jean-Louis Mehren)

Liens

[1] www.elektormagazine.fr/labs/gsm-breakout-board-150330
[2] www.quectel.com/UploadImage/Downlad/M95_Hardware_Design_V1.3.pdf
[3] www.raspberrypi.org/downloads/raspbian/
[4] www.elektor.fr/150400

compteur de visites pour vos pages web

avec écran LCD sans fil

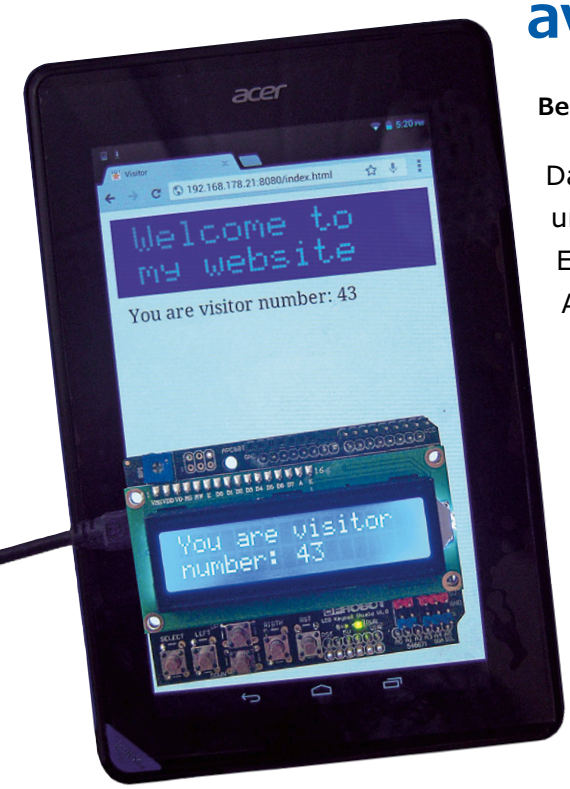


Figure 1. Tablette, carte XinoRF et *shield* LCD.

Bert van Dam (Pays-Bas)

Dans cet article, nous vous montrons comment créer une page web sur un serveur Raspberry Pi et afficher le nombre de visites de cette page. En outre ces informations s'afficheront aussi sur l'écran LCD d'une carte Arduino sans fil.

Le matériel nécessaire pour ce projet se résume à une carte XinoRF (Arduino Uno avec module radio intégré) et un *shield* LCD. Comme serveur web, nous nous servirons d'un nano-ordinateur Raspberry Pi avec le module *Slice of Radio*. La carte XinoRF et le module Slice of Radio font partie du kit RasWIK (*Wireless Inventors Kit for Raspberry Pi*) disponible dans l'e-choppe d'Elektor [1]. Il faut que le Raspberry Pi soit connecté à votre routeur par câble Ethernet ou Wi-Fi. N'oubliez pas que l'adresse IP est celle de votre Raspberry Pi.

Ce projet a été testé avec la carte SD du livre « Raspberry Pi - 45 applications utiles pour l'électronicien » [2] mais fonctionne très bien avec la carte SD du kit RasWIK. Les autres cartes SD ne disposent pas de réglages pour les liaisons radio et/ou les serveurs Python, il vaut mieux ne pas s'en servir.

Dans cet article, nous supposons que vous savez vous servir du Raspberry Pi, installer des logiciels et saisir des commandes. Si ce n'est pas le cas, nous vous suggérons de lire au préalable le livre mentionné ci-avant.

Le projet

Le Raspberry Pi est doté d'un serveur internet Python simple. La page d'accueil de ce serveur (*index.html*) affiche d'abord une illustration et lance ensuite automatiquement un programme Python sur votre Raspberry Pi. Le résultat des instructions de ce programme s'affiche sur la page *index.html* dans un cadre *iframe* (voir **listage 1**). Il s'agit d'un cadre qui se trouve à l'intérieur d'une page web. Vous pouvez ainsi afficher une page web à l'intérieur d'une autre. Dans notre projet, un programme écrit en Python crée cette seconde page web.

L'interface CGI (*Common Gateway Interface*) permet à la page d'accueil de lancer un programme sur le Raspberry Pi. Pour vous protéger contre tout usage abusif, le projet ne fonctionne qu'avec les programmes qui se trouvent dans un dossier spécial nommé *cgi-bin*. En outre, le propriétaire du Raspberry Pi doit donner au préalable à ces programmes des droits d'exécution. Ainsi des personnes non autorisées ne pourront pas démarrer ces programmes.

La commande HTML *viewport* au début du **listage 1** ne vous est peut-être pas familière. Elle sert à adapter la page aux formats des tablettes et ordiphones (voir **figure 1**). La largeur de l'illustration affichée sur la page est de 286 pixels, nous réglons donc *viewport* sur une valeur légèrement supérieure, soit 300 pixels, pour avoir une petite bande blanche à droite de l'illustration. Vous auriez pu paramétrer *viewport* exactement sur la même largeur, mais l'utilisateur risquerait alors de se demander si une partie de la page n'est pas mangée parce qu'il n'y a pas de bordure.

Le programme *visitor.py* lancé par la page *index.html* va d'abord essayer d'ouvrir le fichier où est conservé le nombre de visiteurs. Ce fichier se trouve sur le serveur Raspberry Pi. Si ce fichier n'existe pas encore, c'est qu'il n'y a pas encore eu de visite et le compteur est mis à zéro. Si le fichier existe bien, la valeur qui s'y trouve est lue. Ensuite, la valeur est incrémentée d'un et enregistrée dans le fichier. Le texte « You are visitor number: » est ajouté devant le chiffre, cette chaîne est envoyée via la connexion sans fil à l'Arduino. Enfin, le programme crée une page HTML qui contient cette chaîne et la renvoie à la page *index.html*, où elle s'affichera dans l'*iframe*. Le **listage 2** ne contient que la dernière partie de ce programme, vous trouverez le code source complet dans les fichiers à télécharger [3].
Vous voyez que le programme « imprime » une page HTML. En réalité, tout ce qu'un programme CGI délivre n'est pas envoyé à l'écran (ou une imprimante), mais est transmis au navigateur sous la forme d'un fichier. Il affiche le fichier sous forme de page, dans ce cas au milieu de la page *index.html* dans le cadre *iframe*.

Listage 1.

```
<HTML>
<HEAD>
  <TITLE>Visitor</TITLE>
  <META NAME="viewport" content="width=300" CONTENT="initial-scale=1">
</HEAD>
<BODY>

<IMG SRC="welcome.jpg" WIDTH="286" HEIGHT="70"
ALIGN="BOTTOM" BORDER="0" NATURALSIZEFLAG="0">

<iframe name="myframe" src="cgi-bin/visitor.py" height="50"
      width="100%" frameborder="0"></iframe>

</BODY>
</HTML>
```

Listage 2.

```
# show the HTML page with the
      number of visits
print "Content-Type: text/html"
print """
<HEAD>
<TITLE>Server Counter</TITLE>
</HEAD>
<html>
<body>
%s
</body>
</html>
""" % comment
```

Le programme *visitor.ino* tourne en permanence sur la carte Arduino. Ce n'est donc pas la page *index.html* qui le lance. Ce programme initialise l'écran LCD et attend ensuite les données envoyées par radio. Ces données sont affichées sur l'écran LCD et réparties entre les lignes. Lorsque le Raspberry Pi envoie le signe tilde (~), l'écran LCD est effacé. Nous savons ainsi que le message est mis à jour. Le code source de ce programme se trouve aussi dans les fichiers à télécharger [3].

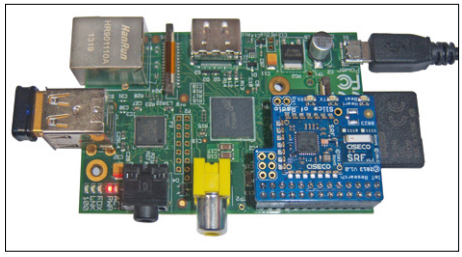


Figure 2. Raspberry Pi et module *Slice of Radio*.

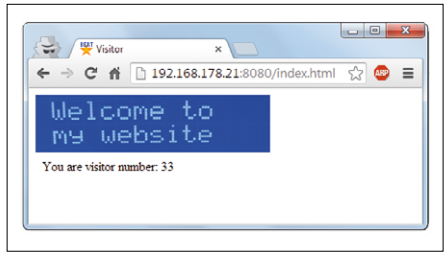


Figure 3. Page web *index.html* affichée sur un ordinateur portable.

Comment procéder ?

Voici un aperçu des étapes à suivre :

1. Transférez le programme *visitor.ino* depuis votre PC sur la carte XinoRF, à l'aide de votre IDE Arduino.
2. Veillez à ce que la distance entre l'Arduino et le Raspberry Pi soit au moins de 50 cm pour la connexion sans fil. Il n'y a en effet aucun câble entre l'Arduino et le Raspberry Pi !
3. Créez un dossier pour ce projet sur votre Raspberry Pi et appelez-le *server*. Dans ce dossier, créez un sous-dossier appelé « cgi-bin ». Attention : vous *devez* utiliser la désignation *cgi-bin* (sans majuscule et avec un trait d'union), sans quoi vous ne pourrez ouvrir aucun programme à distance.
4. Copiez dans le dossier *server* les fichiers suivants : *index.html*, favicon.ico et welcome.jpg (fichiers disponibles en [3]).
5. Copiez le fichier *visitor.py* dans le dossier *cgi-bin*. Attention : n'ouvrez pas ce fichier sur votre ordinateur, pas même pour le lire. Si vous le faites quand même, il sera sauvegardé **et** modifié, il ne pourra plus remplir sa fonction (le serveur affichera une fenêtre pour vous dire qu'il n'a pas trouvé le fichier alors qu'on sait qu'il est bien là).
6. Donnez au fichier les droits d'exécution en tapant l'instruction suivante dans le dossier *cgi-bin* :
   `chmod u+x visitor.py`
   Si vous oubliez, vous verrez la page *index.html*, mais pas le compteur. Dans la fenêtre où tourne le serveur, vous verrez un message d'erreur.
7. Retournez dans le dossier *server* et lancez le serveur en tapant :
   `python -m CGIHTTPServer 8080`
   Il faut le faire depuis le dossier *server*, sans quoi la page *index.html* n'apparaîtra pas.
8. Allez ensuite dans votre navigateur internet sur votre ordinateur ou votre tablette à l'adresse qui suit (n'oubliez pas de remplacer mon numéro IP par celui de votre RPi). Si vous ne vous souvenez plus de l'adresse IP de votre Raspberry Pi, allez sur votre routeur et ouvrez la page des utilisateurs. Vous y trouverez votre Raspberry Pi et son adresse IP.
   `http://192.168.178.21:8080/index.html`
   Vous voyez maintenant une page comme celle de la **figure 3**. Chaque fois que la page est mise à jour, le compteur s'incrémente d'une unité (voir la figure 1).

(150551 – version française : Eric Dusart)

Liens

[1] www.elektor.fr/raswik
[2] www.elektor.fr/rpi
[3] www.elektormagazine.fr/150551

L'auteur

Bert van Dam écrit des livres, des cours et des articles consacrés aux microcontrôleurs, à l'électronique, aux ordinateurs, à l'intelligence artificielle et à la programmation.

Kodi : lecteur multimédia

pour PC, Raspberry Pi, tablette et bien d'autres

Harry Baggen (labo d'Elektor)

PC, ordinateurs portables, ordiphones, tablettes…, aujourd'hui ils sont tous équipés d'un logiciel pour regarder des photos et des vidéos, et écouter des fichiers audio. Mais en général cet outil n'est pas universel ni assez souple : certains formats ne sont pas pris en charge, des paramètres manquent ou encore il n'est pas possible de lire un fichier en ligne. Quelle est la meilleure alternative ?

Il existe beaucoup de lecteurs alternatifs pour différentes plateformes, mais en termes de polyvalence et d'extensibilité, un logiciel sort du lot : Kodi, peut-être plus célèbre sous son ancien nom XBMC. Il s'agit d'un lecteur multimédia à code source ouvert, développé par la Fondation XBMC (un groupe de programmeurs/utilisateurs enthousiastes). Kodi/XBMC est un lecteur qui a été conçu en 2004 pour la première Xbox, d'où son nom *Xbox Media Center* (abrégé en XBMC). Les versions ultérieures ont été portées sur d'autres systèmes d'exploitation, notamment Windows, Linux, iOS et Android. Il y a également eu plusieurs versions autonomes pour des téléviseurs interactifs, des décodeurs et des lecteurs multimédias. Désormais Kodi prend en charge plus de 75 langues.

Un des points forts de Kodi/XBMC est la possibilité d'ajouter des modules complémentaires (*add-ons*) qui permettent de multiplier les fonctions ou d'avoir accès à des sources de médias externes via l'internet. Outre les innombrables additifs officiels, il en existe de nombreux autres qui permettent de consulter toutes sortes de sources audio et vidéo illégales. La Fondation XBMC prend expressément ses distances par rapport à ces sources.

Dans cet article, je vais vous présenter les nombreuses possibilités de Kodi et des appareils sur lesquels vous pouvez l'utiliser. Si de nombreux collègues et amis semblent connaître le nom de Kodi, ils ne savent pas comment s'en servir. En fait, c'est assez facile et vous trouverez sur YouTube des centaines de vidéos pédagogiques pour vous aider à vous y mettre.

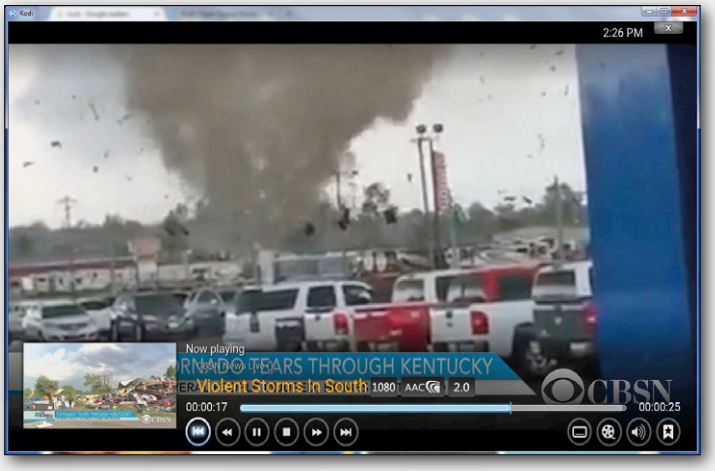
Abondance de versions

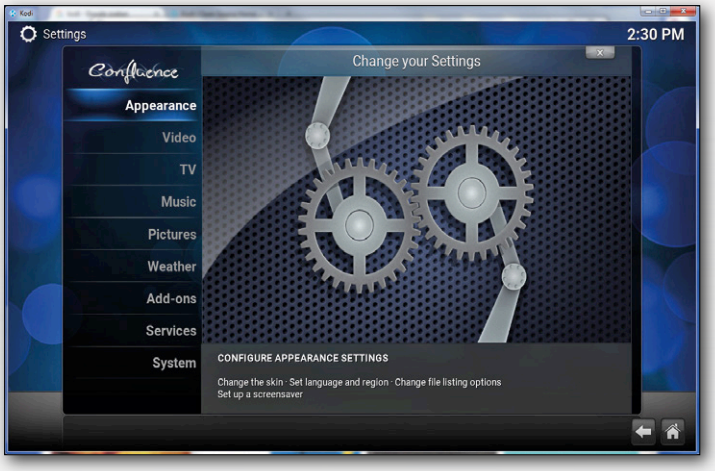
Sur le site officiel de Kodi [1], la page *Downloads* présente toutes les versions actuellement disponibles pour Windows, Linux, Mac OS X, iOS, Android et une série de plateformes matérielles spécifiques comme Raspberry Pi et Amazon Fire TV. Pour la plupart des systèmes, l'utilisateur n'a qu'à télécharger un fichier d'installation et à l'exécuter sur son ordinateur. Pour les appareils Android, Kodi est également disponible dans le Google Play Store. Aujourd'hui, un nombre croissant de lecteurs multimédias qui tournent sous Android arrivent sur le marché, Kodi y est de plus en plus présent.

Le nano-ordinateur Raspberry Pi est une plateforme matérielle très populaire, qui permet de construire soi-même et à moindre coût un lecteur multimédia. Un grand nombre d'entre eux sont sans doute achetés dans ce but. Dès la première version du RPi, on trouvait des versions de Kodi spécialement adaptées à cet effet. OpenELEC (*Open Embedded Linux Entertainment Center* [2]) est l'une des plus populaires. C'est ce qu'on appelle un *Just Enough Operating System* (JeOS) ; vous n'avez donc pas à installer Linux ou un autre système d'exploitation, tout le nécessaire est déjà intégré dans le logiciel. OpenELEC est disponible pour Raspberry Pi A, B, 2 et 3, mais aussi pour plusieurs autres plateformes matérielles comme les PC équipés d'un processeur x86 (idéal pour un PC serveur multimédia, pas besoin de Windows) et des « boîtes » multimédias équipées du processeur Freescale IMX6 (Cubox).

Bien qu'il ne soit pas difficile d'installer le logiciel OpenELEC sur un RPi, il fonctionne un peu différemment de la version Kodi ordinaire. Il faut télécharger une image du disque (une copie exacte du contenu de la mémoire) qui doit ensuite être transférée sur une clé USB ou une carte (micro-)SD. Elle ne peut pas être copiée en utilisant l'Explorateur de Windows ou un autre programme du même genre. Sur une page du site d'OpenELEC [3], vous trouverez

Kodi et RPi, un mariage idéal pour donner naissance à un lecteur multimédia autonome





comment faire sous Linux, Windows et Mac OS X. La plupart des lecteurs d'Elektor utilisent Windows, auquel cas il faut avoir recours à un petit programme, Win32Diskimager [4], pour transférer fidèlement l'image sur une clé USB ou une carte SD. Il suffit ensuite de connecter la clé ou d'enficher la carte dans le RPI, et de le démarrer.

Utilisation

Au début, habituez-vous à l'interface de Kodi et aux nombreux réglages possibles. Le mieux c'est de d'abord jeter un œil au guide de démarrage rapide (*Quick Start Guide*) du wiki Kodi [5] pour découvrir l'interface et ses réglages. Sous *System/Settings*, il y a un grand nombre d'onglets avec des possibilités de réglage ; vous pouvez y choisir votre niveau entre *Basic*, *Standard*, *Advanced* et *Expert* (commencez avec *Standard*, tous les paramètres importants vont s'afficher). La plupart des réglages doivent rester tels quels, sauf peut-être la langue de l'interface sous *Appearance/International* et la langue de sous-titrage des vidéos sous *Videos/Subtitles*. Dans *System/Audio output*, vous pouvez sélectionner le format de sortie audio. Dans un RPi, cela dépend si la sortie HDMI est connectée à un récepteur *surround*, ou directement à un téléviseur.

En plus de lire des vidéos et des photos de votre collection personnelle, il est bien sûr intéressant d'activer un certain nombre d'additifs qui permettent d'accéder à une variété de sources multimédias sur l'internet, depuis les émissions que vous avez ratées jusqu'aux programmes de cuisine et les informations de divers pays. En sélectionnant *Videos/Add-ons*, et ensuite *Get more...* (il en va de même pour *Pictures* et *Music*), vous voyez une liste des additifs présents par défaut dans Kodi. Ici, vous pouvez installer les additifs que vous souhaitez. Ensuite, ils seront visibles dans *Videos/Add-ons*. La liste peut être étendue à de nombreux autres additifs. Un aperçu de tous les additifs officiels de Kodi est disponible en [6]. Ce faisant, attention à la version de Kodi que vous utilisez. Chaque version de Kodi a son propre nom sous lequel sont regroupés les additifs. La version la plus récente est la 16, baptisée « Jarvis ». L'installation d'additifs est décrite dans le wiki Kodi [7], mais il est conseillé de consulter YouTube sur ce sujet.

Tous les additifs ne sont pas approuvés par les développeurs de Kodi, mais il y a en circulation beaucoup d'additifs non officiels intéressants (à ne pas confondre avec les additifs illégaux). Citons deux sources qui ont réuni un grand nombre d'additifs et les ont assemblés dans des paquets zip bien pratiques ; l'utilisateur a ainsi accès en une fois à un grand nombre d'additifs : Super Repo [8] a réuni plus de 2 000 additifs, et TVAddons [9] en compte 1 200. Outre ces deux sources, il existe beaucoup de modules complémentaires illégaux, faites donc attention à ce que vous installez !

(160031 version française : Eric Dusart)

Liens

[1] https://kodi.tv/
[2] http://openelec.tv/
[3] http://wiki.openelec.tv/index.php/HOW-TO:Installing_OpenELEC/Creating_The_Install_Key#tab=DiskImage
[4] https://sourceforge.net/projects/win32diskimager/
[5] http://kodi.wiki/view/Quick_start_guide
[6] http://kodi.wiki/view/Category:All_add-ons
[7] http://kodi.wiki/view/Add-ons
[8] https://superrepo.org/
[9] https://www.tvaddons.ag/

Swiss Pi

couteau suisse pour Raspberry Pi

Ilse Joostens & **Peter S'heeren** (Belgique)

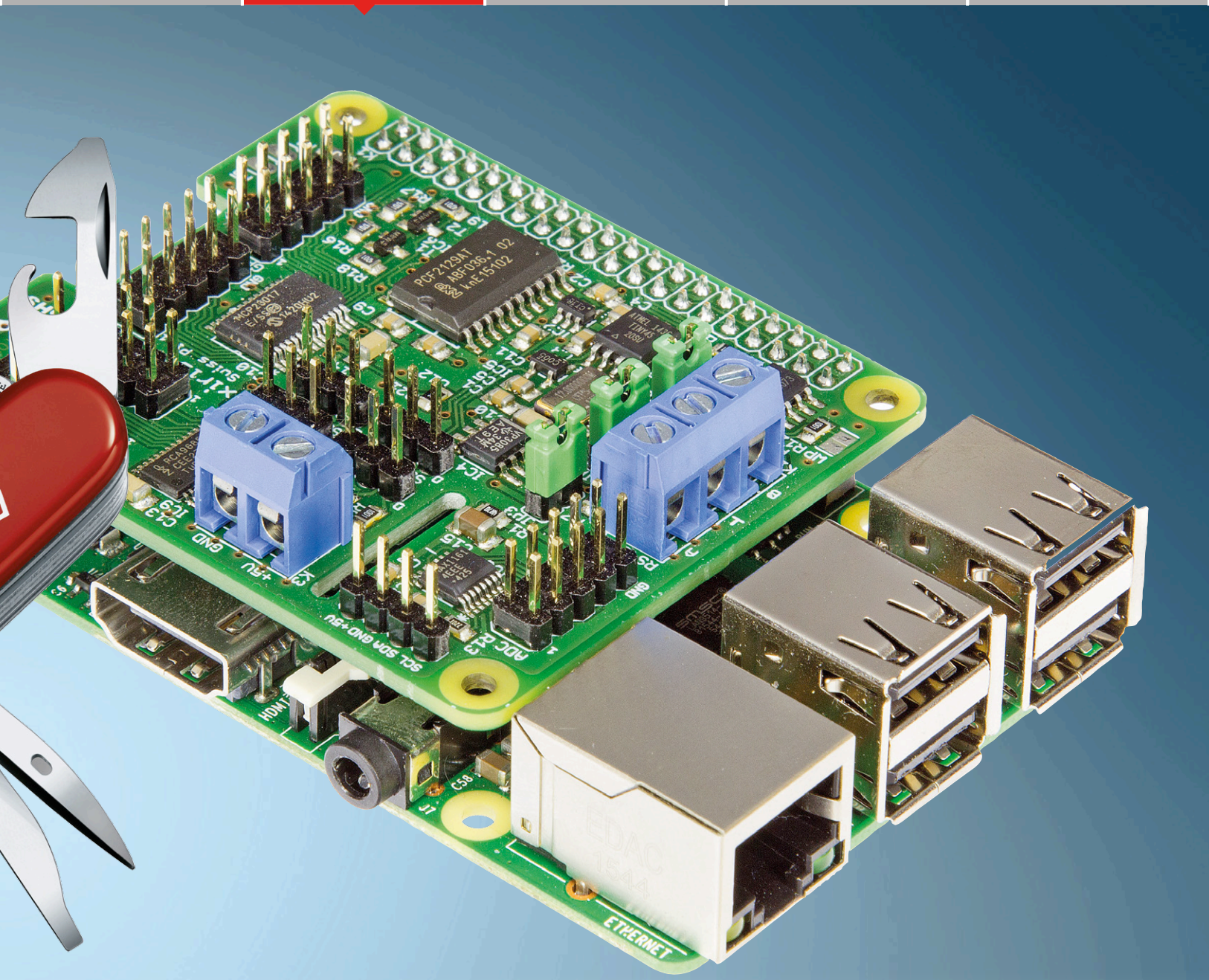
L'ordinateur mono-carte Raspberry Pi est apparu au printemps 2012. Quatre ans plus tard, il est toujours très populaire. La fondation Raspberry Pi ne s'est pas endormie sur ses lauriers ; fin février 2016, elle a présenté au grand public la version 3. Si RPi rencontre un tel succès, c'est sûrement grâce à son connecteur GPIO à 40 contacts qui permet de raccorder facilement des extensions.

Comparé à celui de l'Arduino, le connecteur GPIO (E/S d'usage général) de Raspberry Pi présente certaines limitations. Ainsi, les lignes de GPIO, directement branchées à l'unité centrale, sont très vulnérables et côté signaux MLI, une seule broche est disponible. Il lui manque aussi des entrées analogiques (CA/N). Le HAT (*Hardware Attached on Top*, matériel en impériale) Swiss Pi pallie ces manques et ajoute même des fonctions comme une horloge en temps réel et une interface RS485.

Le logiciel correspondant est disponible gratuitement sur [1], il fonctionne aussi bien sous Linux que Windows. Plusieurs clients peuvent utiliser des composants de Swiss Pi en même temps via un programme de serveur. On peut aussi s'adresser au serveur en différents langages tels que Python et PHP. C'est pourquoi on peut envisager de commander la carte avec une interface web via l'inter-

Caractéristiques

- 16 lignes d'E/S à usage général avec résistance commutable, niveau 5 V
- 16 canaux MLI à 12 bits, réglables de 24 à 1 526 Hz, niveau 5 V
- 4 connecteurs pour servomoteurs (répartis sur les canaux MLI 0 à 3) + connecteur d'alimentation
- 8 entrées CA/N à 12 bits, plage de 0 à 4,096 V
- interface RS-485 semi-duplex selon norme TIA/EIA-485A avec protection contre les décharges électrostatiques
- horloge en temps réel précise avec pile de secours, précision de 3 ppm
- connecteur d'extension I^2C, niveau 5 V
- ID EEPROM



net. Enfin, une interface utilisateur de type graphique, bien ordonnée, facilite l'utilisation des fonctions de la carte, d'où une grande simplicité pour les essais par exemple.

Le matériel

En vrai canif suisse, la carte offre une panoplie d'outils pratiques, le schéma de la **figure 1** vous en convaincra. Passons-les donc en revue.

Extension d'E/S

Pour commander les 16 lignes de GPIO, nous avons choisi une extension d'E/S MCP2307 dotée de l'I²C. On peut configurer chacune de ces lignes en entrée à haute impédance, entrée à résistance de polarisation ou sortie. C'est une combinaison très répandue avec le RPi, si bien que de nombreux exemples de programmes et des informations sous forme de tutoriels sont disponibles.

Contrôleur MLI

Le PCA9685 est un pilote de LED I²C à 16 canaux, c'est lui qui va produire les signaux MLI. Avec une fréquence MLI réglable entre 24 et 1 526 Hz et une précision de 12 bits, ce pilote de LED (RGB) est aussi doué pour la commande de moteurs à CC (avec pont MOSFET) et de servomoteurs de modélisme. Dans cette dernière application, les canaux 0 à 3 sont reliés à quatre connecteurs tripolaires pour mettre en œuvre directement des servomoteurs. Vu la gourmandise de certains servos, on ne les alimente pas depuis le RPi, mais par une source externe branchée sur K3. Si vous utilisez plus de quatre servos, à vous de prévoir des raccordements supplémentaires.
Combiné à un filtre actif à réseau RC (**fig. 2**), le PCA9685 pourra aussi servir de convertisseur N/A, afin de fournir des signaux de commande de 0 à 10 V pour des gradateurs, par exemple.

Convertisseur A/N

Le convertisseur analogique/numérique utilisé, un MAX11614EEE+, compte

8 canaux d'une précision de 12 bits et une référence de tension interne de 4,096 V. On peut câbler les canaux en symétrique aussi bien qu'en asymétrique. Le logiciel prévu pour le Swiss Pi n'utilise les canaux qu'en asymétrique. Les dimensions des cartes d'extension pour RPi sont standardisées aux spécifications HAT, il en résulte que la place y est limitée. Aussi, pas de diodes de protection installées aux entrées du CA/N. Donc, si vos capteurs sont alimentés par des sources

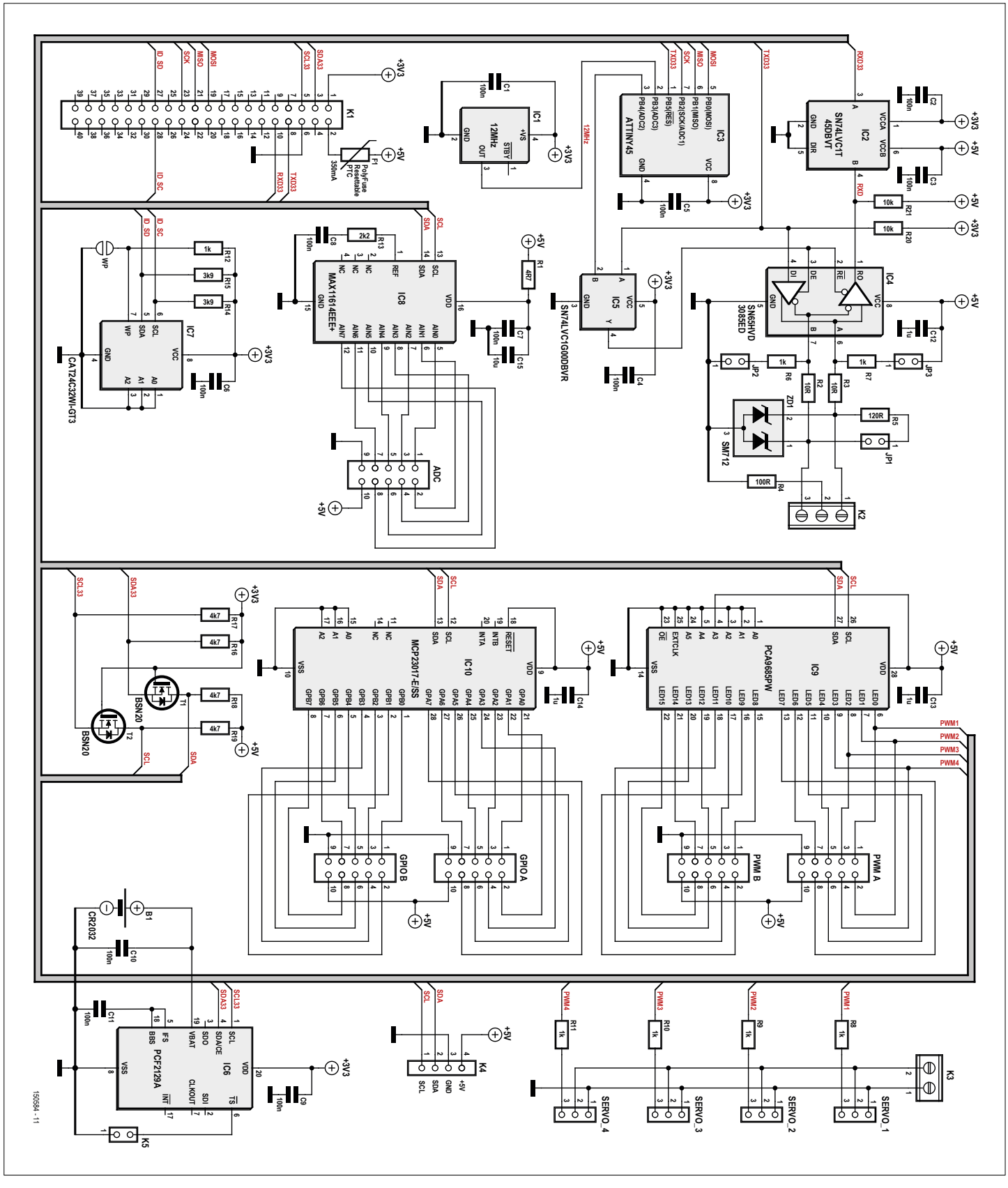


Figure 1. Le schéma de Swiss Pi.

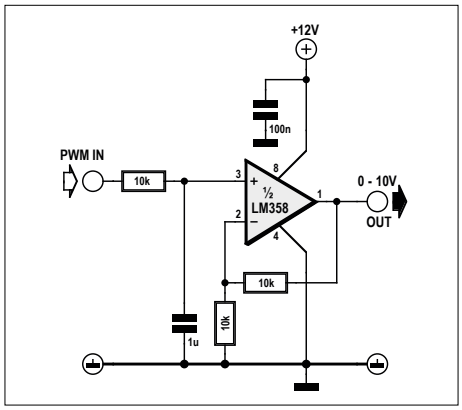


Figure 2. Comment fabriquer une belle tension analogique à partir de la MLI.

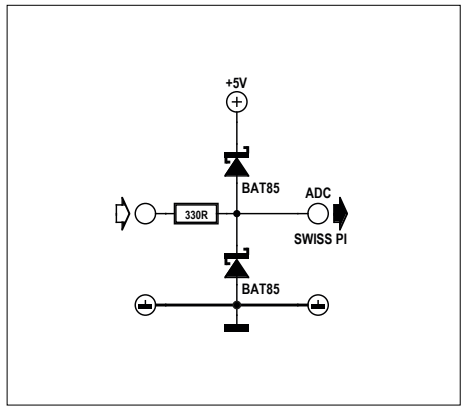


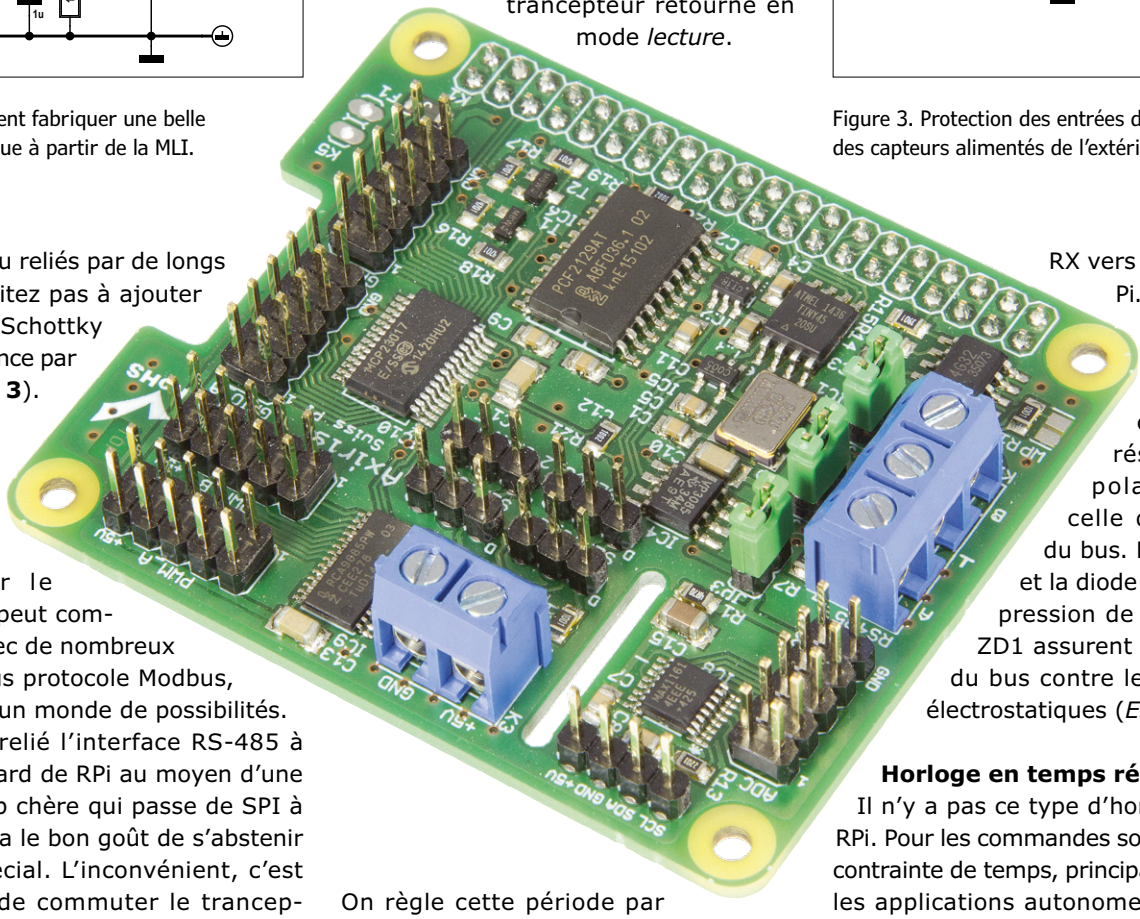
Figure 3. Protection des entrées du CA/N pour des capteurs alimentés de l'extérieur.

extérieures ou reliés par de longs câbles, n'hésitez pas à ajouter deux diodes Schottky et une résistance par sécurité (**fig. 3**).

Interface RS-485

Grâce à l'interface RS-485 sur le Swiss Pi, on peut communiquer avec de nombreux appareils sous protocole Modbus, ce qui ouvre un monde de possibilités. Nous avons relié l'interface RS-485 à l'UART standard de RPi au moyen d'une puce pas trop chère qui passe de SPI à RS-485. Elle a le bon goût de s'abstenir de pilote spécial. L'inconvénient, c'est la difficulté de commuter le trancepteur RS-485 de mode *émission* en mode *réception* (signal RE/DE) après l'envoi d'un mot. Le matériel du RPi ne dispose d'aucun moyen approprié et faire effectuer cette tâche par logiciel en utilisant une ligne GPIO pour la commutation ne serait pas fiable, vu que Linux n'est pas un SE en temps réel.

Alors, on y colle le petit microcontrôleur IC3, un ATtiny45, et une porte NON-ET (IC5). Quand l'interface sérielle est au repos et IC3 aussi inactif, les deux entrées de la porte logique sont hautes, donc la sortie basse met le trancepteur RS-485 en mode *lecture*. Dès que le Raspberry Pi commence à transmettre, l'entrée A de la porte est abaissée par le flanc descendant du signal TX (bit de départ). La sortie devient haute, ce qui met le trancepteur immédiatement en mode *émission*. Le signal TX arrive aussi à la broche 1 de l'ATtiny et quand il détecte ce flanc descendant, l'entrée B de la porte devient basse. De ce fait, le trancepteur reste en mode *émission*, même quand le signal TX redevient haut. Après un délai réglable, l'ATtiny remet l'entrée B de la porte au niveau haut. Il faut choisir cette période pour qu'elle se termine juste avant la fin du dernier bit d'arrêt, après quoi le trancepteur retourne en mode *lecture*.



On règle cette période par l'interface SPI du RPi en fonction du débit binaire, de la longueur de mot, des bits de parité et du nombre de bits d'arrêt. Nous ne pouvions pas utiliser l'I²C ici, parce que l'ATtiny n'a qu'une interface USI (*Universal Serial*) limitée, ce qui entraîne qu'une grande partie du protocole I²C (esclave) doit être réalisée en logiciel. Ce n'est évidemment pas aussi rapide qu'en matériel et il faudrait donc recourir à un allongement de la période d'horloge (*clock stretching*) pour que l'ATtiny puisse suivre le bus I²C. Une erreur matérielle dans le maître I²C du RPi empêche d'effectuer cet allongement, raison pour laquelle nous avons dû nous rabattre sur le SPI.

Le trancepteur RS-485 est ici alimenté en 5 V, la puce IC2 effectue l'adaptation nécessaire du niveau à 3,3 V pour la ligne RX vers le Raspberry Pi. Les cavaliers JP1, JP2 et JP3 permettent de configurer les résistances de polarisation et celle de bouclage du bus. Enfin, R2, R3 et la diode Transil (suppression de transitoires) ZD1 assurent la protection du bus contre les décharges électrostatiques (*ESD*).

Horloge en temps réel

Il n'y a pas ce type d'horloge dans le RPi. Pour les commandes soumises à une contrainte de temps, principalement dans les applications autonomes où aucune connexion internet n'est disponible (pour donner la date et l'heure), une horloge en temps réel est bien pratique. Nous avons choisi un PCF2129 à quartz intégré. Non seulement bon marché, cette puce est aussi très précise (à 3 ppm) et compensée en température. Une pile bouton CR2032 fournit une alimentation secourue d'une durée de vie de dix ans environ.

Avec K5, on peut accéder à un second horodateur. Toutes les informations à ce sujet se trouvent dans la fiche technique du PCF2129A.

Convertisseur de niveau I²C, ID EEPROM, *polyfuse*

À l'exception de l'horloge en temps réel, toutes les puces I²C sont alimentées en 5 V. Les lignes SDA et SCL demandent une adaptation du niveau, c'est le rôle

des deux MOSFET à canal N T1 et T2 avec les résistances R16 à R19.
Le fusible réarmable F1 limite la consommation totale à 350 mA. C'est suffisant pour le bon fonctionnement du RPi, même avec une alimentation de moindre puissance. Mais il faut tenir compte de cette limite lors du branchement d'extensions sur les connecteurs GPIO et MLI.

Il y a encore cette ID EEPROM IC7, nécessaire pour se conformer à la spécification HAT de RPi. L'idée est que le noyau Linux la lit au moment de l'amorçage et se sert de ces informations afin de configurer correctement pour le HAT (autoconfiguration / structure interne) les lignes GPIO du RPi. À notre connaissance, les explications sur le sujet sont très limitées.

Liste des composants

Résistances :
(0805, sauf mention contraire)
R1 = 4,7 Ω
R2, R3 = 10 Ω, résistante aux impulsions
R4 = 100 Ω ≥ ¼ W
R5 = 120 Ω ≥ ¼ W
R6 à R12 = 1 kΩ
R13 = 2,2 kΩ
R14, R15 = 3,9 kΩ
R16 à R19 = 4,7 kΩ
R20 à R21 = 10 kΩ

Condensateurs :
C1 à C11 = 100 nF, 0805
C12 à C14 = 1 µF, 1206
C15 = 10 µF, 1206

Semi-conducteurs :
T1, T2 = BSN20
IC1 = oscillateur à quartz CMS 12 MHZ, type LFSPXO018037
IC2 = SN74LVC1T45DBVR
IC3 = ATtiny45-20SU
IC4 = SN65HVD3085EDG4
IC5 = SN74LVC1G00DBVR
IC6 = PCF2129AT
IC7 = EEPROM 32 kbit, I²C, type BR24G32FJ-3GTE2
IC8 = MAX11614EEE+
IC9 = PCA9685PW,112
IC10 = MCP23017-E/SS
ZD1 = diode de suppression de transitoires, 12 V, type CDSOT23-SM712

Divers
F1 = fusible réarmable, 16 V, 0,35 A, type 1206L035/16YR
B1 = coupleur de pile bouton 2032
K1 = support 2×20 broches au pas de 2,54 mm
K2 = bornier à 3 vis pour bus RS-485 au pas de 5 mm
K3 = bornier à 2 vis au pas de 5 mm
K4 = embase à 1×4 picots au pas de 2,54 mm
K5 = embase à 1×2 picots au pas de 2,54 mm
ADC = embase à 2×5 picots au pas de 2,54 mm
PWM_A, PWM_B = embase à 2×5 picots au pas de 2,54 mm
GPIO_A, GPO_B = embase à 2×5 picots au pas de 2,54 mm
SERVO_1 à SERVO_4 = embase à 1×3 picots au pas de 2,54 mm
JP1 à JP3 = embase à 1×2 picots au pas de 2,54 mm

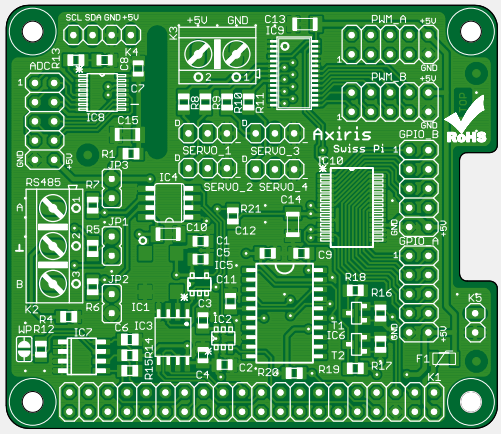
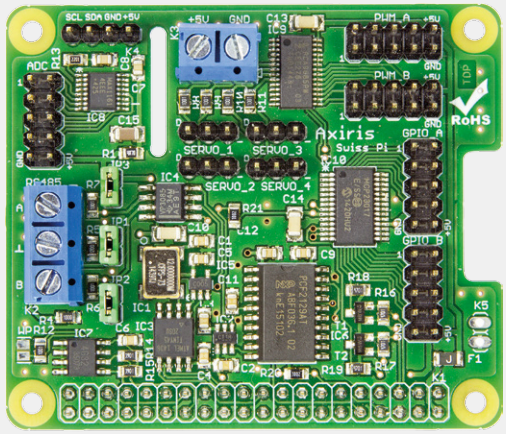


Figure 4. La face « composants » du circuit imprimé compact, exactement adapté au Raspberry Pi.



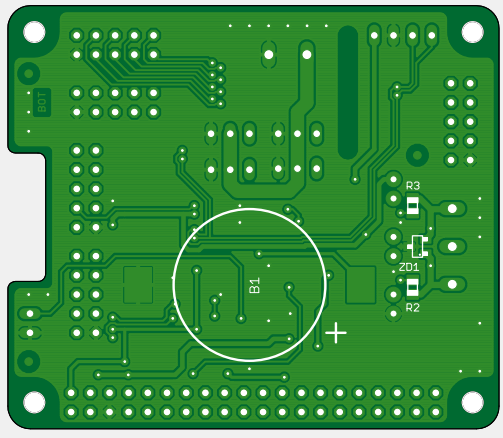


Figure 5. Le tracé des pistes du circuit imprimé.

Circuit imprimé

La **figure 4** montre le côté composants du circuit imprimé développé pour le Swiss Pi, l'envers est à la **figure 5**.

Le logiciel

Serveur Swiss

Le programme *Swiss Server* offre les fonctions de Swiss Pi sur des ports de réseau et des E/S normales suivant un protocole de communication spécial. Le serveur accepte des connexions clients multiples, si bien que plusieurs clients peuvent travailler simultanément avec le Swiss Pi, ce qui permet une grande souplesse.
Le protocole de communication repose sur l'échange de commandes et de réponses codées au format ASCII. Un client envoie une commande au serveur, le serveur retourne une réponse quand la commande a été exécutée. Grâce à cette syntaxe, on peut facilement piloter le serveur, directement ou avec un logiciel tiers.

Le serveur connaît un grand nombre de commandes pour travailler avec le Swiss Pi. Le protocole de communication est totalement asynchrone, si bien qu'un client n'a pas besoin d'attendre une réponse avant d'envoyer une nouvelle commande. Les clients peuvent envoyer plusieurs commandes à la suite, ce qui renforce grandement l'efficacité du système. Option : chaque commande peut comporter un numéro d'identification qui sera renvoyé dans la réponse.

Accessoirement, le serveur peut aussi échanger directement des octets de données entre l'interface RS-485 et un port réseau. Si vous désactivez cette option, les clients pourront utiliser des commandes pour envoyer des octets de données par l'interface RS-485.

Le serveur Swiss travaille tant sous Linux que sous Windows, mais pour le second cas, il faut une carte d'adaptation (AxiCat) entre le Swiss Pi et l'USB de l'ordinateur. Au sujet d'AxiCat, un prochain article donnera les informations nécessaires.

Commande de la liaison RS-485

Ce programme converse avec l'ATtiny afin de configurer les fonctions RS-485, au moyen d'un jeu de registres accessible par l'interface SPI. Si vous employez le serveur Swiss, vous n'aurez probablement que rarement besoin de ce programme,

puisque le serveur Swiss propose les fonctions RS-485 les plus couramment utilisées.

Il y a dans le programme une procédure pour déterminer les temps en mode *émission*. Elle est exécutable sur le Swiss Pi tout seul, sans aucun autre matériel spécifique.

I/O Card Explorer

La **figure 7** présente l'interface graphique interactive du programme *I/O Card Explorer* qui facilite le travail avec les puces I²C sur le Swiss Pi. Il y a différentes manières de communiquer avec le matériel. Si ce programme tourne sur RPi, il commande directement le Swiss Pi et il en a même l'exclusivité.

Quand c'est le serveur Swiss qui pilote le Swiss Pi, vous pouvez demander à *I/O Card Explorer* de communiquer avec le matériel via le serveur Swiss. Il est alors possible d'exécuter le programme sur un autre ordinateur et de le mettre en liaison avec le serveur Swiss via un réseau. Plusieurs instances du programme et d'autres clients peuvent communiquer simultanément avec le serveur Swiss, donc plusieurs utilisateurs peuvent travailler en parallèle avec le Swiss Pi.

Extensions

La mise en œuvre de Swiss Pi dans le monde réel demandera certainement des extensions spécifiques qu'il est utile de prévoir. Nous pensons entre autres à une carte à relais électromécaniques/statiques, une carte avec des entrées numériques isolées galvaniquement, un adaptateur pour boucle de courant de 4 à 20 mA, des pilotes de gradateur de 0 à 10 V, des commandes de moteur à CC avec pont en H, une carte de pilotes RVB pour ruban de LED, etc. Sur le connecteur I²C de Swiss Pi, on peut monter notre mini-hôte à 8 canaux de type 1-wire, pour ajouter des composants comme les capteurs de température DS18B20.
Enfin, il y a encore une variante de Swiss Pi pour BeagleBone : le Swiss Cape, comparable à Swiss Pi quant au matériel et qui utilise le même logiciel.
Les fichiers du projet (Eagle) et les fichiers Gerber sont disponibles sur [1]. Vous pouvez aussi vous procurer le module assemblé Swiss Pi à l'e-choppe [2].

(150584 – version française : Robert Grignard)

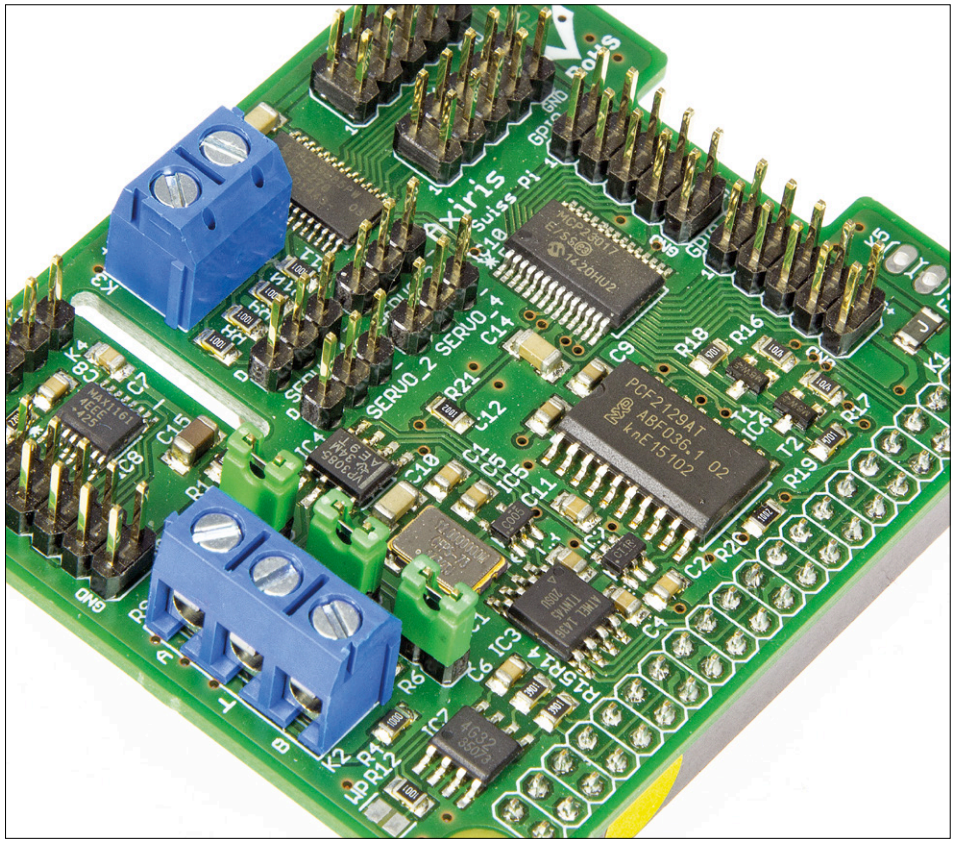


Figure 6. Le module assemblé de notre prototype.

Liens

[1] www.elektormagazine.fr/150584

[2] www.elektor.fr/swiss-pi

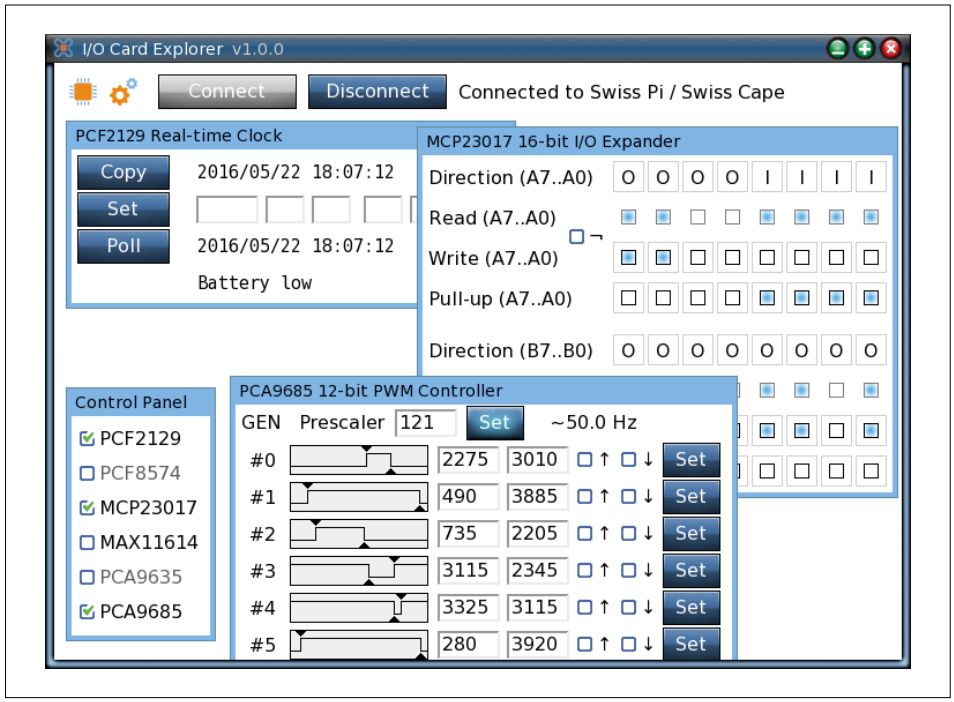


Figure 7. Un écran de l'appli « I/O Card Explorer » de ce projet.

nouvelles lames pour le Swiss Pi

exemples de programme

Peter S'heeren & **Ilse Joostens** (Belgique)

La carte d'extension Swiss Pi [1] dote le célèbre ordinateur mono-carte Raspberry Pi d'une foule de fonctions utiles. Après la présentation de ce couteau suisse pour RPi (Elektor, 09/2016), nous nous intéressons au serveur Swiss. C'est l'occasion de partager avec vous des exemples de programme écrits en Python et PHP.

C'est toute une série de fonctions que le Swiss Pi ajoute au Raspberry Pi : lignes GPIO, canaux MLI, commande de servo-moteurs, bus RS-485, canaux de CA/N, horloge en temps réel, etc. Pour que l'utilisateur puisse accéder facilement à ces fonctions et même les faire tourner en parallèle, une bonne complicité logicielle entre Swiss Pi et Raspberry Pi est indispensable. On peut comparer le serveur Swiss à un serveur http par ex., puisqu'il permet aux clients, comme à l'utilisateur et à d'autres programmes, de commander la carte avec des instructions. Plusieurs clients peuvent accéder simultanément à Swiss Pi.

Vous apprendrez ici à mettre le serveur en service, puis, comme client, à interagir avec lui pour piloter le Swiss Pi. Vous verrez aussi comment des programmes écrits en Python ou PHP peuvent communiquer avec le serveur pour commander le Swiss Pi.

Le logiciel

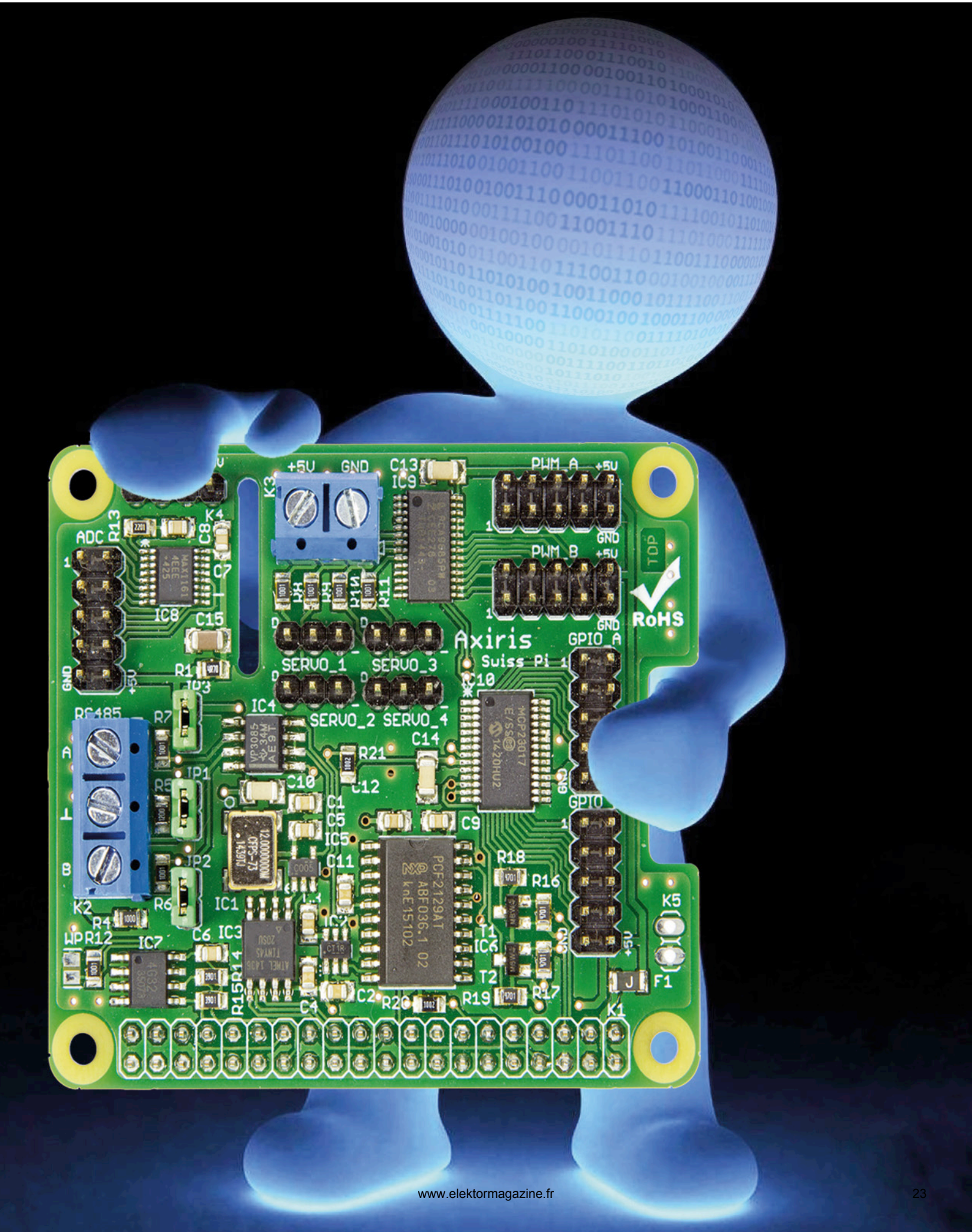
La page du produit de Swiss Pi [2] contient de la documentation et un lien vers la page du logiciel [3]. Cette page met à disposition le code source du serveur Swiss, des programmes en Python et PHP, et la documentation afférente.

Le serveur Swiss

Le programme *Swiss Server* offre les fonctions de Swiss Pi sur des ports de réseau et des E/S normales selon un protocole de communication qui repose sur l'échange de commandes et des réponses correspondantes. Un client envoie une tâche au serveur qui lui répond dès qu'elle a été exécutée. Un client peut envoyer un chapelet de tâches pour accélérer considérablement le déroulement.

Le protocole de communication est codé en ASCII. Sa syntaxe facilite l'usage tant manuel que programmé. Le protocole complet est décrit dans la documentation du logiciel [3].

L'hôte primaire du Swiss Pi est un RPi. Si vous combinez le Swiss Pi avec un AxiCat (**fig. 1**) [4], vous pourrez utiliser un PC sous Linux ou Windows comme hôte. Mais pour cet article-ci, l'hôte sera un Raspberry Pi.


+5V
GND
C13
IC9
K3
PWM_A
+5V
GND
PWM_B
+5V
GND
TOP
RoHS
ADC
R13
K4
C8
C7
IC8
C15
R1
D3
GND
+5V
Axiris
Suiss Pi
GPIO_A
SERVO_1
SERVO_3
SERVO_2
SERVO_4
C10
IC4
R7
RS485
A
B
IP1
R5
C14
R21
C12
C10
C1
C5
C9
GPIO
IC5
C11
R18
R16
IP2
R6
K2
IC1
R4
IC2
T1
IC6
T2
K5
WP
R12
IC7
C6
IC3
R14
R15
C4
C2
R20
R19
R17
J
F1
K1
MCP23017
E/SS
1420HU2
PCF2129AT
ABF036.1 02
knE15102

Swiss Pi offre une liaison directe entre son bus RS-485 et l'UART de l'hôte

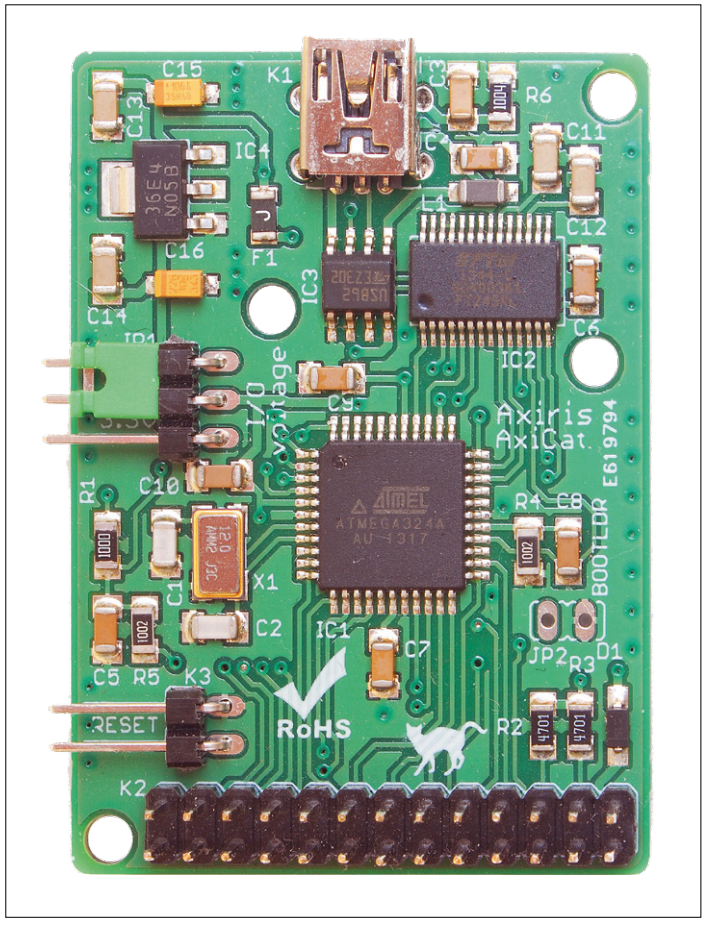


Figure 1. Besoin d'une interface I²C ou SPI pour votre PC sous Linux ou Windows ? Prenez une AxiCat !

Démarrage

Exemple de lancement du serveur avec un RPi comme hôte :

```
./swissserver -v -i2cdev /dev/i2c-1 -p 5003
```

```
./swissserver -v -bscdetect -spidev /dev/spidev0.0
   -serial /dev/ttyS0 -rs485 spi -crlf -p 5003 -stdio
   -sp 5004
```

Exemple de lancement du serveur avec AxiCat et Windows comme hôte :

```
swissserver.exe -v -console -axicat \\.\COM12 -rs485
   spi -crlf -p 5003 -stdio -sp 5004
```

Le paramètre `-v`, de *verbose*, signifie bavard. Le serveur montre alors les noms des interfaces matérielles utilisées et toute autre information opportune, toutes les instructions et réponses que le serveur traite. Pratique si vous travaillez en Python ou PHP pour voir si les commandes ont été correctement exécutées. Le paramètre `-console` est spécifique à Windows qui ouvre une console où le serveur peut montrer des informations.

Attention ! Les chemins de périphériques pour Linux tels que `/dev/ttyS0` sont normalement acceptés par les versions récentes de Raspbian et Raspberry Pi modèle 3. La présence et la dénomination de ces chemins ont déjà été modifiées plusieurs fois ces dernières années. Si certains chemins n'existent pas sur votre RPi, vérifiez les paramètres pour I²C, SPI et UART dans `raspi-config`. Vous trouverez sur l'internet plus d'information à ce sujet.

RS-485

Swiss Pi établit une liaison directe entre son bus RS-485 et l'UART de l'hôte. Le contrôleur RS-485 est relié au bus SPI de l'hôte. Pour accorder au serveur la commande totale sur le RS-485, passez-lui les paramètres suivants :

- `serial /dev/ttyS0`
  le serveur commande l'UART de l'hôte.
- `spidev /dev/spidev0.0`
  le serveur commande le bus SPI de l'hôte.
- `rs485 spi`
  le serveur commande le contrôleur RS-485 de Swiss Pi.

Il y a trois méthodes pour faire passer des données par le bus RS-485 :

- avec les instructions `serr` et `serw`, lancer le serveur par `-serial` ;
- par un port réseau (par ex. le port 5004), lancer le serveur avec `-sp` et `-serial`.
- par le UART de l'hôte (par ex. /dev/ttyS0).

Avec la méthode 3, le serveur ne fournit d'accès ni à l'UART de l'hôte, ni au bus RS-485. Il faut configurer convenablement le contrôleur RS-485 avec la commande `rste` de manière à ce que l'UART et le contrôleur respectent le même réglage sériel.

Interactif

Pour vous familiariser avec le serveur, nous conseillons d'envoyer d'abord des instructions de manière interactive. Si vous démarrez le serveur avec `-stdio`, vous pouvez saisir des instructions directement dans le *shell*. Si vous le démarrez avec `-p`, vous pouvez établir une liaison avec un programme de terminal (PuTTY, netcat, Hyperterminal) et transmettre les instructions manuellement. Par exemple, se relier à netcat sur Raspberry Pi : `$ nc localhost 5003`.
Vous pouvez alors saisir des instructions. Dans les exemples qui suivent, les instructions sont en noir et les réponses du serveur en bleu. L'instruction suivante demande la version du serveur :

```
ver
ver «1.0.1»
```

On peut si nécessaire adjoindre à chaque instruction un numéro d'identification qui sera répété dans la réponse. Par exemple

« attendre 100 ms » :

```
id 10 wait 100
id 10 wait ok
```

On peut aussi adjoindre à chaque instruction un préfixe pour qu'il n'y ait pas de réponse :

```
norsp wait 100
```

GPIO

Une ligne d'E/S d'usage général peut servir d'entrée. Par exemple « régler la broche 10 d'E/S comme entrée, activer la résistance de polarisation haute et lire l'état de l'entrée » :

```
iod 10 1
iod ok
iopu 10 1
iopu ok
ior 10
ior 10 1 0 1 1
```

La valeur soulignée dans la dernière réponse veut dire que l'entrée est à l'état haut.
Il est aussi possible d'utiliser une ligne d'E/S d'usage général comme sortie. Par exemple « régler la broche 11 d'E/S comme sortie, la mettre au niveau haut » :

```
iod 11 0
iod ok
iow 11 1
iow ok
```

CA/N

L'instruction `adcr` convertit les huit canaux du convertisseur A/N et renvoie les résultats dans la réponse.

```
adcr
adcr 3460 1843 0756 0233 0060 0000 0000 0000
```

Le serveur enregistre les résultats de la dernière conversion. Vous pouvez à tout moment les réclamer avec l'instruction `adcrc`. Dans l'exemple suivant, on lit les résultats enregistrés pour les canaux 2 à 4 du CA/N :

```
adcrc 2 3
adcrc 2 3 0756 0233 0060
```

MLI

Le contrôleur MLI possède un diviseur préalable pour régler la fréquence MLI et 16 canaux, chacun a quatre paramètres qui déterminent la position de départ et la période de l'onde carrée. La signification précise de ces paramètres est expliquée dans la documentation. Les instructions `ppr` et `ppw` permettent d'écrire et de lire dans le registre du diviseur préalable. Avec les instructions `pcr` et `pcw`, vous pouvez écrire et lire un ou plusieurs canaux MLI.

```
ppr
ppr 030
ppw 121
ppw ok
pcr 5
pcr 05 01 0 0000 1 0000
pcw 9 1 0 1250 0 1280
pcw ok
pcr 8 2
pcr 08 02 0 0000 1 0000 0 1250 0 1280
```

Ces instructions servent à lire le diviseur préalable, positionner le registre sur 121 (50 Hz), lire le canal 5 MLI, écrire le canal 9 MLI, lire les canaux 8 à 9 MLI. La documentation décrit plusieurs variantes de `pcr` et `pcw`.

Servo

Le serveur dispose d'une commande de servomoteurs embarquée pour 16 canaux indépendants. Les canaux 0 à 3 sont à disposition sur un connecteur distinct du Swiss Pi.

On peut mettre en ou hors service la commande de servo par `svme` et `svmd`, configurer les canaux MLI en canaux de servo (`svcw`), activer des mouvements (`svmv`) et demander l'état (`svcr`).

Si vous travaillez avec des servomoteurs, vous devez régler la fréquence MLI sur 50 Hz :

```
ppw 121
ppw ok
```

Pour la configuration d'un canal MLI, `svcw` a besoin des informations suivantes :

- numéro de canal (0 à 15)
- largeur d'impulsion en 12 bits : la plus courte permise (0 à 4 095)
- largeur d'impulsion en 12 bits : la plus grande permise (0 à 4 095)
- type de commande de la position du moteur : par pas (0 à max.) ou largeur d'impulsion en 12 bits (0 à 4 095).

```
svcw 0 118 515 1000
svcw ok
```

Cette instruction définit que la commande de servo du canal MLI 0 peut opérer en largeur d'impulsion entre 118 et 515 et

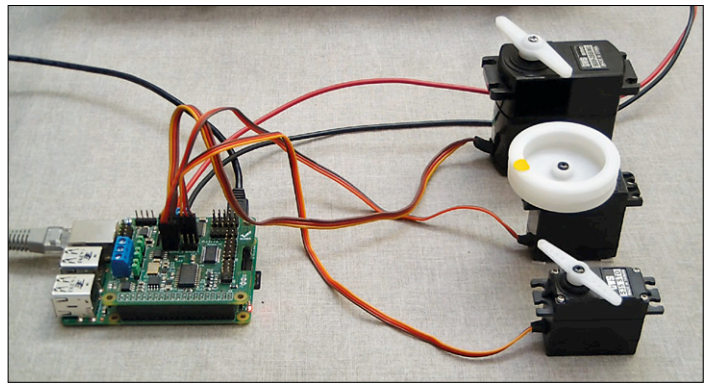


Figure 2. La commande de servomoteurs est un jeu d'enfant.

qu'une valeur de pas entre 0 et 1 000 sera donnée à svmv pour obtenir une largeur d'impulsion de 118 à 515.

```
svcw 1 122 520 0
svcw ok
```

Cette instruction définit que la commande de servo peut utiliser le canal MLI 1 avec des périodes MLI de 122 à 520 et qu'une largeur d'impulsion sera donnée à svmc.

Avant de démarrer un moteur, il faut activer la commande de servo :

```
svme
svme ok
```

Alors, pour changer la position du moteur :

```
svmv 0 500 0 0
svmv 00 ok
```

Cette instruction positionne le moteur au 500e pas, donc à mi-chemin entre 0 et 1 000. Le serveur convertit cette valeur de pas en largeur d'impulsion de 316, le milieu entre 118 et 515.

```
svmv 0 250 2000 0
svmv 0 500 2000 0
svmv 0 400 1000 0
```

Les instructions svmv sont exécutées l'une après l'autre. La durée totale est d'au moins 5 000 ms, vu qu'entre deux instructions, un retard peut survenir. Si vous souhaitez un mouvement fluide durant exactement 5 000 ms, utilisez le quatrième paramètre de svmv. Il indique que l'instruction svmv doit démarrer *x* millisecondes après l'instruction svmv précédente. Le serveur compense d'éventuels retards pour que l'exécution s'effectue toujours dans le temps prescrit.

```
svmv 0 250 2000 0
svmv 0 500 2000 2000
svmv 0 400 1000 2000
```

S'il n'y a eu aucune instruction svmv précédente pour le canal actif, le serveur prend svme comme point de départ. Veillez bien dans ce cas à ce que la largeur d'impulsion actuelle soit la bonne pour le domaine en question, sinon le serveur ne pourra pas faire exécuter le mouvement vers la nouvelle position. Cette façon de faire permet de synchroniser différents moteurs :

Pour travailler avec des servomoteurs, réglez la fréquence MLI à 50 Hz

```
svmv 1 400 0 0
svmv 01 ok
```

Cette instruction met le moteur dans la position qui correspond à une largeur d'impulsion de 400.

Le moteur se rend directement à la position indiquée ; on peut aussi fixer une période pour le mouvement :

```
svmv 0 500 2500 0
svcr 0
svcr 00 0118 0515 1000 0174 0141 1
svcr 0
svcr 00 0118 0515 1000 0257 0350 1
svmv 00 ok
svcr 0
svcr 00 0118 0515 1000 0316 0498 0
```

La première ligne fait exécuter le déplacement pendant une période de 2 500. Les instructions svcr révèlent la position actuelle du moteur. Le serveur n'envoie de réponse que quand le mouvement est terminé. L'exécution d'une svmv a lieu en arrière-plan, c'est donc une instruction asynchrone.

Vous pouvez envoyer plusieurs instructions successives au même canal :

```
svme
svme ok
svmv 0 250 2000 500
svmv 1 400 1550 500
```

Le serveur lance le mouvement des deux moteurs 500 ms après svme.

Horloge en temps réel

Le serveur dispose d'instructions pour lire l'horloge en temps réel (rtcr) et la régler (rtcw).

```
rtcr
rtcr 1 1
rtcw 2016 10 21 15 42 55
rtcw ok
rtcr
rtcr 0 1 2016 10 21 15 42 58
```

La première réponse signale que l'horloge en temps réel ne contient pas de temps valide et qu'il n'y a pas de pile branchée. La deuxième instruction met à jour et à l'heure. La troisième instruction renvoie la date et l'heure.

RS-485

Avec sercfg, il est possible de configurer l'UART de l'hôte et le contrôleur RS-485 d'un seul coup, à condition que le serveur

soit lancé par spi avec `-serial`, `-spidev` et `-rs485`.
Comme exemple, nous avons raccordé un compteur d'énergie SMD120C sur rail DIN au bus RS-485 et voici l'instruction qui configure la ligne sérielle sur 2 400 bauds, 8 bits de donnée, 1 bit d'arrêt, pas de parité et demande aussi au contrôleur RS-485 de configurer :

```
sercfg 2400 8 1 none 1
sercfg ok
```

Le SDM120C travaille avec le ModBus RTU. Pour demander la valeur de la tension :

```
serw 01h 04h 00h 00h 00h 02h 71h 0CBh
serw ok
```

Et pour lire la réponse :

```
serr
serr 001 004 004 067 107 153 154 117 231
```

La documentation du SDM120C indique que la tension est fournie dans un nombre à virgule flottante de 32 bits, ce sont ceux qui sont soulignés.

```
host = 'localhost'
port = 5003
client = textlinenetclient.
   TextLineNetClient(host,port)
```

Cette liaison est locale, le code Python doit être exécuté sur le RPi. Si vous voulez travailler avec le Swiss Pi depuis un autre ordinateur, vous devez donner l'adresse IP de RPi sous la forme `host = '192.168.1.115'`. Vous pouvez demander l'adresse IP de votre Raspberry Pi avec la commande `ifconfig`.
La variable `client` désigne la liaison. La classe dispose de fonctions pour envoyer des instructions et recevoir des réponses en lignes de texte.
L'écriture des instructions comporte deux étapes. La classe travaille avec un tampon d'écriture. La fonction `sendCmd` ajoute une instruction au tampon d'écriture. Pour transférer les instructions du tampon au serveur, on a la fonction `commit`. Autre solution : ajouter le paramètre `True` à `sendCmd`. On peut ainsi envoyer au serveur une kyrielle d'instructions d'un seul coup, c'est plus rapide que de les envoyer séparément.
La fonction `rcvRsp` lit les données du serveur et envoie une seule réponse par appel. Cette fonction bloque le serveur tant qu'elle n'a pas obtenu de réponse.
La fonction `tokenizeLine` découpe une ligne de texte en une série de symboles (*token*). Les programmes utilisent cette fonction pour simplifier l'analyse et préparer la réponse. Si vous avez lancé le serveur avec le paramètre `-v`, vous pouvez surveiller les réponses du serveur.

`sercfg` configure d'un coup l'UART de l'hôte et le contrôleur RS-485

Python
Les exemples de programme proposés sont écrits pour Python 3. Si vous utilisez Raspbian sur votre Raspberry Pi, Python 3 est déjà installé sur votre système. Parcourons-en les principaux aspects.
Pour exécuter un programme, vous lancez Python avec le nom de fichier du programme comme paramètre dans l'invite de commande :

Linux : `python3 swisspi_gpio_read_pin.py`

Windows : `python swisspi_gpio_read_pin.py`

Les programmes font usage du serveur Swiss pour communiquer avec Swiss Pi. Ainsi, vous pouvez faire tourner les programmes localement sur le Raspberry Pi ou sur un autre ordinateur.

Les programmes utilisent `class TextLineNetClient` dans *textlinenetclient.py*. Cela permet d'établir la liaison avec le serveur Swiss pour envoyer des tâches et traiter les réponses. Le nom de la classe indique qu'il s'agit d'un client de réseau qui échange des lignes de texte avec le serveur. La classe produit une exception du type `class TextLineNetClientError` quand survient une erreur telle qu'une rupture de liaison avec le serveur.
Le code suivant en Python établit une liaison avec le serveur Swiss :

Les nombres qui arrivent dans les réponses sont formatés selon un modèle réglable. Il y a différents formats afin d'obtenir une bonne lisibilité en mode interactif. Quand on travaille en Python, mieux vaut recevoir tous les nombres en valeurs décimales. Dans ce cas, les programmes envoient la commande :

```
client.sendCmd('norsp vfmts «*» dec 0 0 0 0 0 0')
```

Remarquez qu'avec `norsp`, le serveur n'envoie pas de réponse. Il n'est pas nécessaire non plus d'envoyer immédiatement après une instruction au serveur. Ce qui est important c'est que le serveur reçoive cette instruction en premier.

GPIO
Le code Python qui suit configure une ligne d'E/S d'usage général en entrée, active la résistance de polarisation haute et lit l'état de l'entrée. La variable `pin` sélectionne les broches de GPIO (0 à 15).

```
pin = 10
client.sendCmd('norsp iod %d 1' % pin)
```

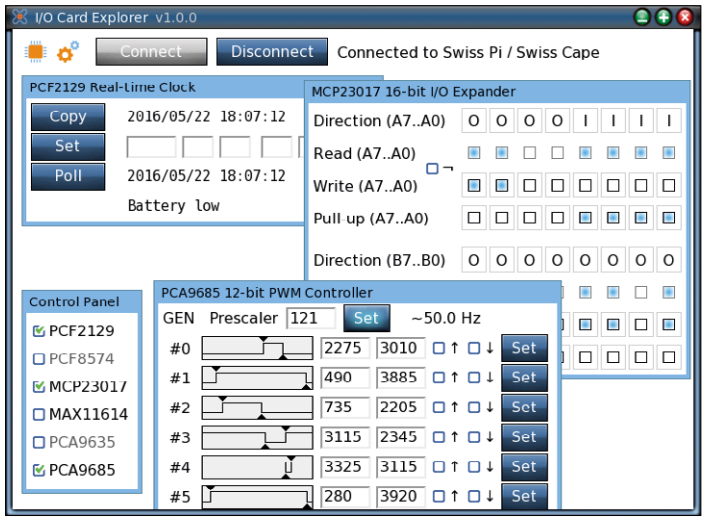


Figure 3. Avec *I/O Card Explorer*, la liaison avec le Swiss Pi est facile.

```
client.sendCmd('norsp iopu %d 1' % pin)
client.sendCmd('ior %d' % pin,True)
tokens = client.tokenizeLine(client.rcvRsp())
print("GPIO pin %d: input state %d" %
   (pin,int(tokens[2])))
```

Le code envoie trois instructions au serveur. La première met la broche GPIO 10 en entrée, la deuxième active la polarisation et la troisième `sendCmd` lit l'état, mais aussi transmet le tampon d'écriture au serveur à cause du paramètre `True`. Le serveur répond à la troisième instruction sous forme scindée, avec des symboles. Le troisième symbole (`tokens[2]`) contient l'état d'entrée de la broche de GPIO.

Configurer l'état de sortie d'une broche de GPIO se fait en sélectionnant aussi par `pin` la broche de GPIO (0 à 15), puis la variable `output` établit l'état (0 ou 1). Mettons la sortie de la broche 2 au niveau haut :

```
pin = 2
output = 1
client.sendCmd('norsp iow %d %d' % (pin,output),True)
```

ADC

L'instruction `adcr` convertit les huit canaux A/N et transmet les résultats dans la réponse. Les canaux sont numérotés de 0 à 7. Pour voir la valeur du canal 4 :

```
ch = 4
client.sendCmd('adcr',True)
tokens = client.tokenizeLine(client.rcvRsp())
print(«A/D channel %d: %d» % (ch,int(tokens[1 + ch])))
```

MLI

Le contrôleur MLI a un registre de division préalable pour régler la fréquence de la MLI. Pour lire le registre :

```
client.sendCmd('ppr',True)
tokens = client.tokenizeLine(client.rcvRsp())
print(«PWM prescale: %d» % int(tokens[1]))
```

Écrire dans ce registre se fait ainsi :

```
pre = 121  # 50 Hz
client.sendCmd('norsp ppw %d' % pre,True)
```

Chaque canal dispose de quatre paramètres pour fixer la position de départ et la période de l'onde carrée. Pour simplement régler la période :

```
ch = 4
on_period = 1250
client.sendCmd('norsp pcw %d 1 0 0 0 %d' %
   (ch,on_period),True)
```

Servo

Le code suivant configure deux canaux et tourne les moteurs dans une certaine position :

```
client.sendCmd('norsp svcw 0 118 515 1000')
client.sendCmd('norsp svcw 1 122 520 0')
client.sendCmd('norsp svme')
client.sendCmd('norsp svmv 0 250 0 0')
client.sendCmd('norsp svmv 1 400 0 0')
client.commit()
```

Horloge en temps réel

Le code qui suit règle l'horloge en accord avec le temps local de l'ordinateur ; il montre aussi l'usage de la fonction `commit`, qui équivaut à l'ensemble des paramètres `True` et `sendCmd`.

```
t = datetime.datetime.now()
client.sendCmd('norsp rtcw %d %d %d %d %d %d'
   % (t.year, t.month, t.day, t.hour, t.minute,
   t.second))
client.commit()
```

RS-485

L'exemple de programme *swisspi_rs485_sdm120c.py* lit la tension d'un compteur d'énergie SDM120C et l'imprime.

PHP

Les exemples de programme utilisent le serveur Swiss pour communiquer avec le Swiss Pi. Vous pouvez ainsi exécuter les programmes localement sur le Raspberry Pi mais aussi sur un autre ordinateur.

Les programmes se servent de `class SwissClient` dans *SwissClient.php*. Cette classe propose une série de fonctions qui exécutent les instructions du serveur. Les fonctions comprennent l'échange d'instructions et de réponses avec le serveur. L'avantage est que les fonctions sont faciles à utiliser. Toutefois il y a un inconvénient : l'impossibilité d'envoyer plusieurs instructions en même temps et d'opérer avec des réponses asynchrones. C'est logique, puisque le PHP s'utilise généralement

pour produire des pages internet et l'accès synchrone au Swiss Pi lui suffit.

Comment réaliser une liaison avec le serveur ? D'abord, il faut créer un objet `SwissClient`, puis appeler la fonction `connect`. Elle renvoie un `true` si la connexion a réussi, `false` si elle a échoué.

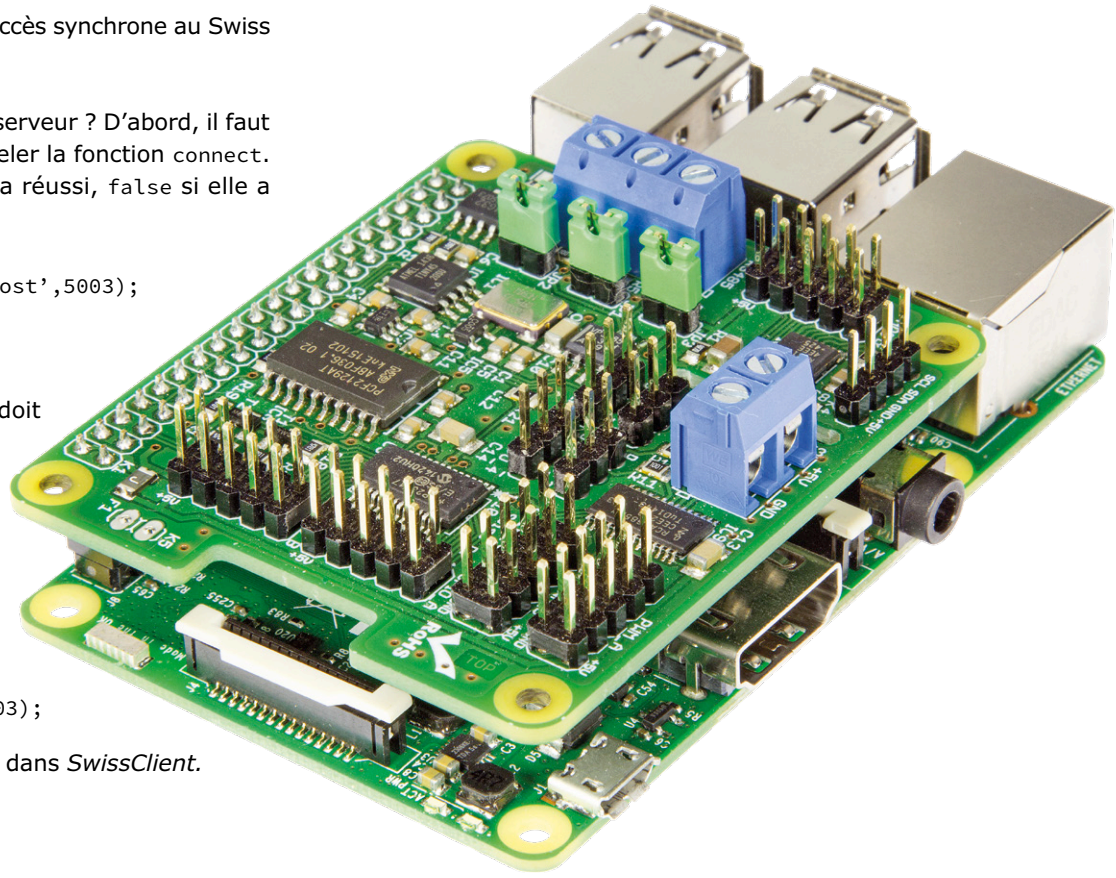
```
$client = new SwissClient('localhost',5003);
$ok = $client->connect();
if (!$ok) return false;
```

Cette liaison est locale, le code PHP doit être exécuté sur le Swiss Pi. Si vous voulez travailler depuis un autre ordinateur avec le Swiss Pi, vous devez donner l'adresse IP du Raspberry Pi. Voici comment ça marche :

```
$client = new

   SwissClient('192.168.1.115',5003);
```

Ce que font les fonctions est expliqué dans *SwissClient.php*. Suivent quelques exemples.



Avec *I/O Card Explorer*, observez l'effet des instructions sur le Swiss Pi.

Lire l'état de la broche 4 de GPIO et imprimer le résultat :

```
$res = $client->readIOPin(4);
var_dump($res);
```

Mettre la sortie broche 4 de GPIO au niveau haut, communiquer la réponse du serveur :

```
$client->writeIOPin(4,true,true);
```

Écrire vers deux canaux MLI :

```
$pwm_data = array
(
    array("always_on" => false, "on_pos" => 1200,
   "always_off" => false, "off_pos" => 2424),
    array("always_on" => true,  "on_pos" =>  444,
   "always_off" => true,  "off_pos" => 100)
);

$client->writePWM2Range(14,$pwm_data,true);
```

Et maintenant ?

Le logiciel contient de nombreux fichiers en Python et PHP que vous pouvez mettre à profit. Vous pouvez à tout moment établir une liaison avec le programme *I/O Card Explorer* [5] par le serveur Swiss (**figure 3**). Vous avez ainsi l'occasion, pendant le transfert interactif d'instructions ou l'exécution de codes Python ou PHP, d'observer l'activité de Swiss Pi et éventuellement d'intervenir.

La prochaine fois, nous vous présenterons quelques extensions matérielles pratiques, parmi lesquelles une carte à relais à 8 canaux, une carte d'entrée numérique à 8 canaux, une commande de moteur à courant continu, une interface à boucle de courant et une petite carte qui convertit la MLI en tension entre 0 et 10 V. ⏮

(160237 – version française : Robert Grignard)

Liens

[1] www.elektormagazine.fr/150584

[2] www.axiris.eu/en/index.php/i-o-cards/swiss-pi

[3] www.axiris.eu/en/index.php/free-software/software-repository

[4] www.elektormagazine.fr/150585

[5] www.axiris.eu/en/index.php/free-software/i-o-card-explorer

[6] www.elektormagazine.fr/160237

PROJET DU LABO

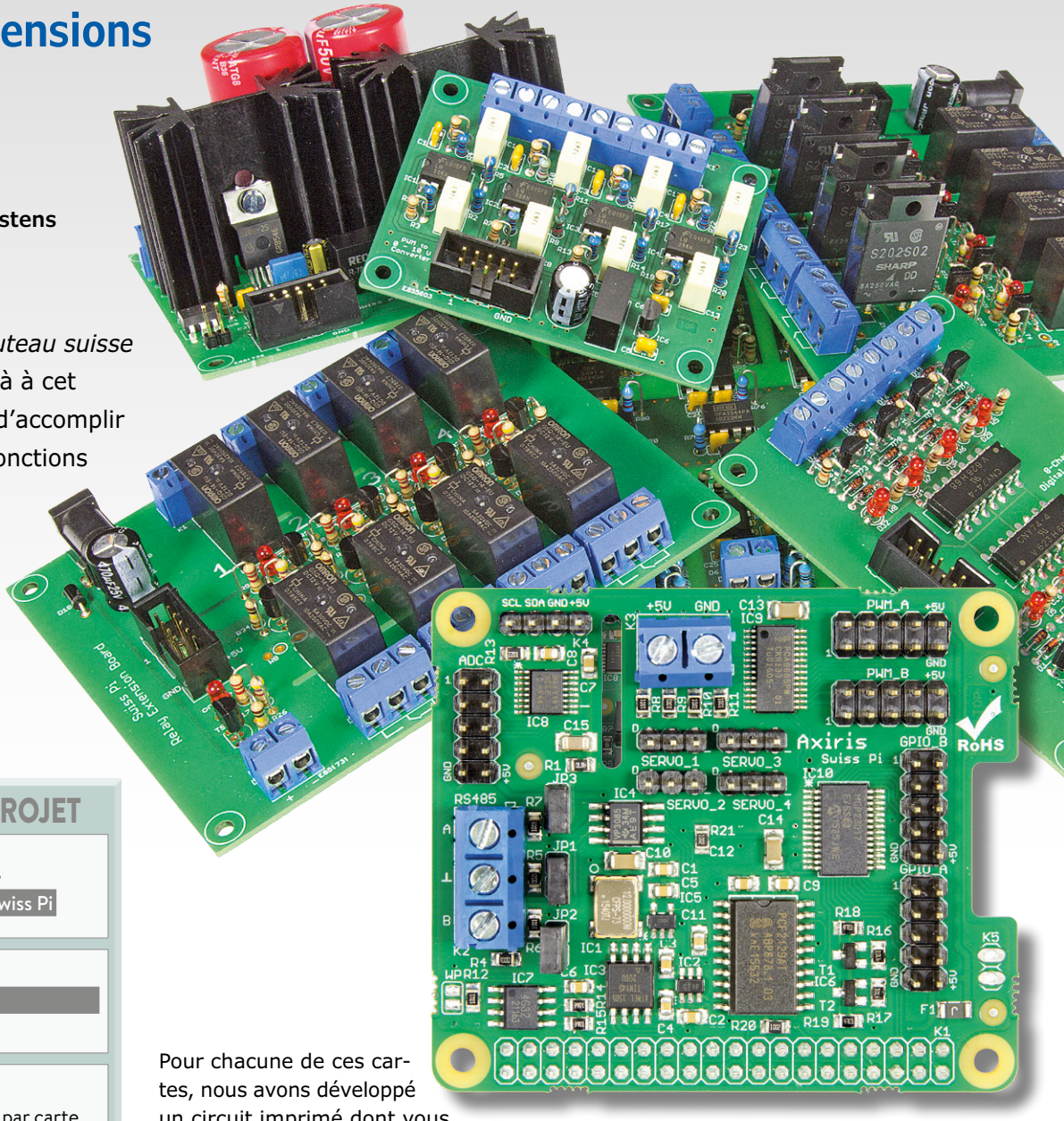
Swiss Pi

toujours plus polyvalent

grâce aux extensions matérielles

Peter S'heeren & **Ilse Joostens**
(Belgique)

La carte d'extension *couteau suisse pour RPi* [1] permet déjà à cet ordinateur mono-carte d'accomplir toute une panoplie de fonctions utiles. Continuons sur notre lancée avec une série de cartes d'extension matérielle pour différents domaines courants.



À PROPOS DU PROJET

Microcontrôleurs
Raspberry Pi | Swiss Pi

débutant
➡ connaisseur
expert

de 30 mn à 1 h30 par carte

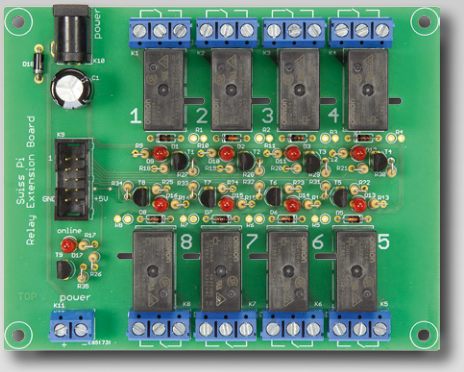
poste de soudage avec panne adéquate

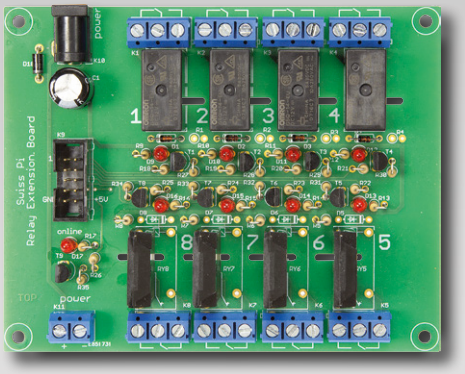
entre 25 € et 50 € par carte

Pour chacune de ces cartes, nous avons développé un circuit imprimé dont vous pouvez télécharger les fichiers Eagle et Gerber [2]. Ils vous permettront de faire réaliser ces circuits imprimés par le fabricant de votre choix. Autant que possible, nous avons privilégié les composants traversants aux CMS, question de vous simplifier la construction. Et si vous souhaitez créer vos propres cartes d'extension, vous trouverez ici l'inspiration nécessaire. Sans oublier que ces circuits ne sont pas forcément tributaires du Swiss Pi, ils accompagneront sans difficulté d'autres plateformes, un circuit à microcontrôleur de votre cru ou une carte Arduino.

Carte à relais à 8 canaux

La carte à huit relais (**fig. 1**) utilise des relais électromagnétiques de la série G5Q d'Omron. En lieu et place de ces





relais ordinaires, vous pouvez installer des modèles à semi-conducteurs S202S02 de Sharp. Ceux-ci ne sont plus fabriqués, mais ils sont encore largement disponibles sur eBay. Un autre type donnerait des complications, parce que celui de Sharp est de loin le seul relais statique qui possède la même empreinte que ceux de la série G5Q. L'entrée du S202S02 se fait sur une seule LED, elle a donc besoin d'une résistance série de limitation de courant. Malgré ses dimensions modestes, le relais G5Q répond à toutes les exigences en matière de rigidité diélectrique et de distances de sécurité pour supporter la tension du secteur. Les bornes du relais à semi-conducteur sont un peu plus serrées, aussi avons-nous pratiqué une saignée dans le circuit imprimé pour éviter les courants superficiels de fuite. Même si le relais G5Q est capable de commuter jusqu'à 10 A, la largeur des pistes n'a été dimensionnée que pour 5 A.

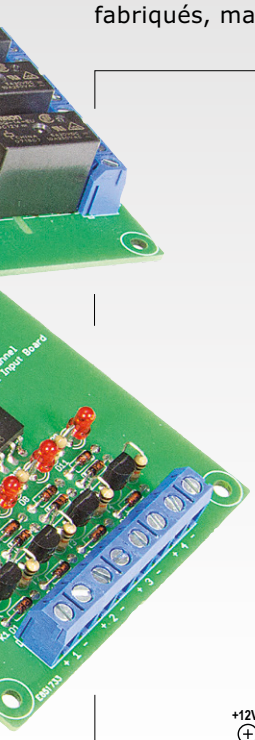


Figure 1. La carte à huit relais convient aussi bien pour les modèles électromagnétiques G5Q SPDT que pour les statiques S202S02.

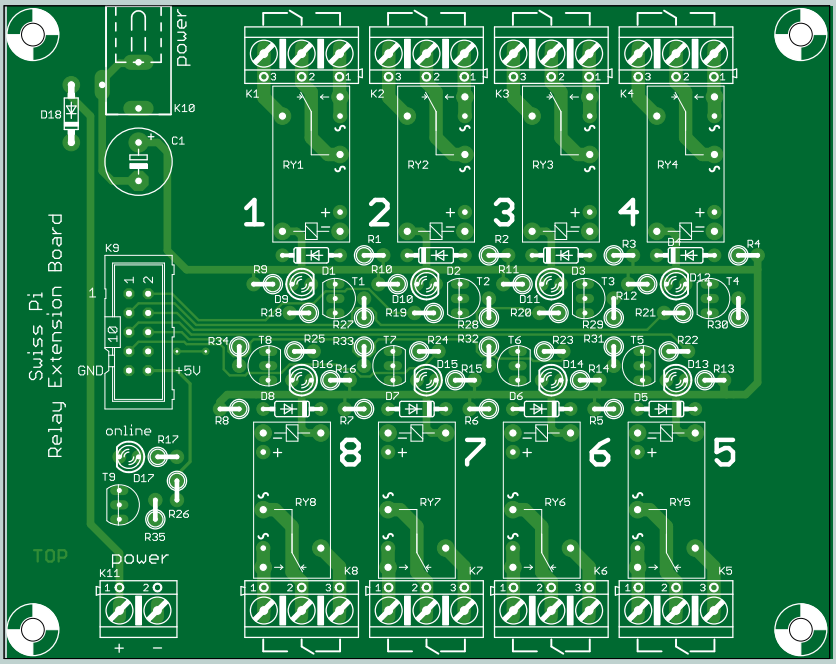
LISTE DES COMPOSANTS : CARTE À RELAIS

Résistances :
R1 à R8 = 560 Ω*
R9 à R17 = 2,2 kΩ
R18 à R26 = 4,7 kΩ
R27 à R35 = 100 kΩ

Condensateurs :
C1 = 100 µF/16 V

Semi-conducteurs :
D1 à D8 = 1N4148**
D9 à D17 = LED, 3 mm, rouge
D18 = 1N4007
T1 à T9 = BC547B

Divers :
RY1 à RY8 = G5Q-14-EU DC12V ou S202S02***
K1 à K8 = bornier à vis, à 3 pôles, au pas de 5 mm
K9 = embase à collerette à 10 broches
K10 = connecteur d'alimentation CC coaxial
K11 = bornier à vis, à 2 pôles, au pas de 5 mm



*à installer avec relais statique S202S02
**à installer avec relais G5Q
***à installer par canal au choix ; pour le relais G5Q, on peut choisir une autre tension si nécessaire, mais alors il faut adapter les valeurs des résistances R1 à R17 et du condensateur C1.

Ce sont des transistors NPN standard qui assurent la commande des relais. Un témoin à LED est aussi prévu. Une autre LED indique quand la carte et le Swiss Pi sont sous tension.

Si nécessaire, vous pouvez aussi installer des relais G5Q d'une autre tension, 24 V par exemple, auquel cas il faudra recalculer un certain nombre de résistances.

Huit entrées numériques protégées

Cette carte d'extension offre huit entrées à isolation galvanique par photocoupleur qui accepte toute tension continue dans

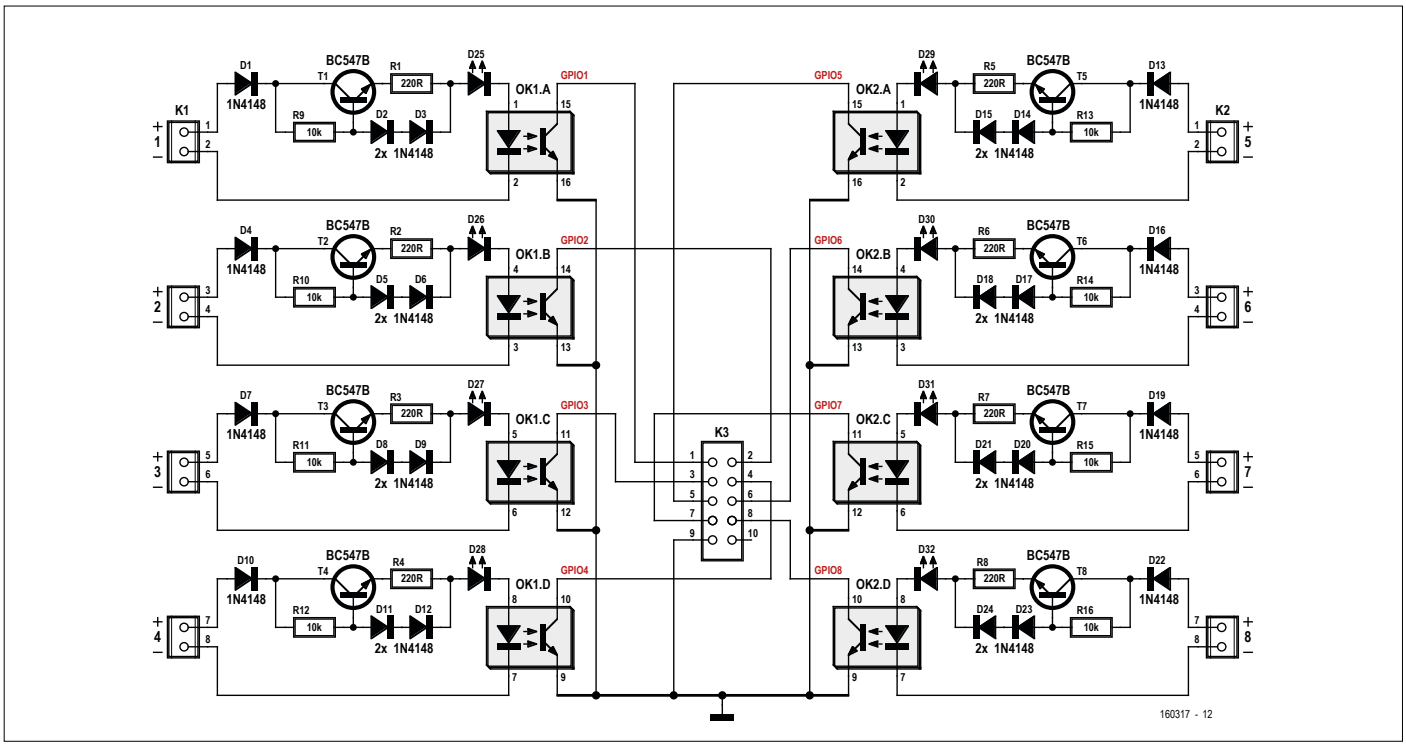


Figure 2. Les huit photocoupleurs assurent l'isolation galvanique des huit entrées numériques.

la plage de 5 à 45 V (cf. **fig. 2**). Des sources de courant par transistors BC547 alimentent les LED des photocoupleurs et les témoins à LED à débit relativement constant par rapport à la tension d'entrée. Le courant y est égal aux deux tensions de seuil des diodes 1N4148 moins la tension base/émetteur du transistor, le tout divisé par 220, la valeur des résistances R1 à R8. Dans la fourchette de tensions d'entrée de 5 à 45 V, il circule dans les diodes un courant entre quelques dizaines de microampères et quelques milliampères. La caractéristique de la 1N4148 montre que c'est dans ce domaine que la tension de seuil varie le plus en fonction du courant. En pratique, le débit produit par les sources de courant n'est pas constant sur toute la plage des tensions d'entrée : il peut aller de 3 à 6 mA.

Si l'on inversait la polarité sur la source de courant, la tension entre collecteur et émetteur du transistor serait négative. Le transistor continuerait à fonctionner comme un (piètre) NPN avec un gain très faible et une basse tension de claquage. Tant que la tension entre collecteur et émetteur reste inférieure à environ −6 V, le transistor reste en conduction et le courant n'est plus limité que par la résistance de 220 Ω et le courant de fuite

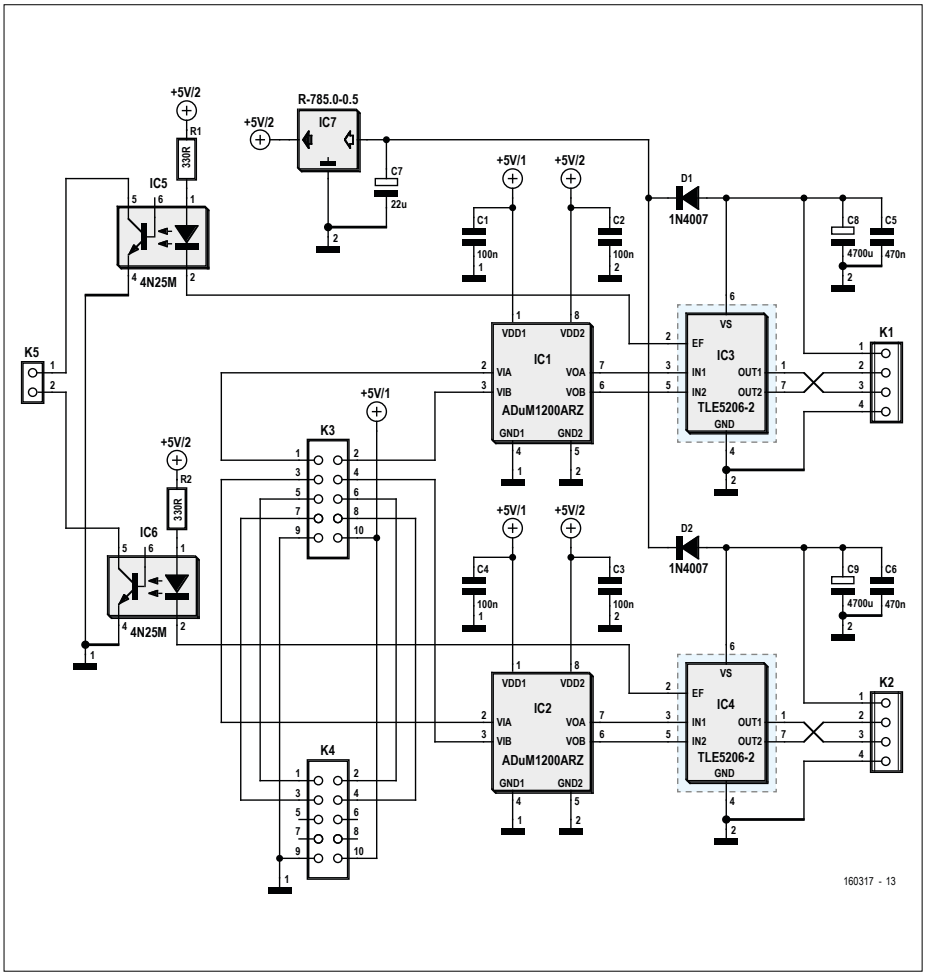


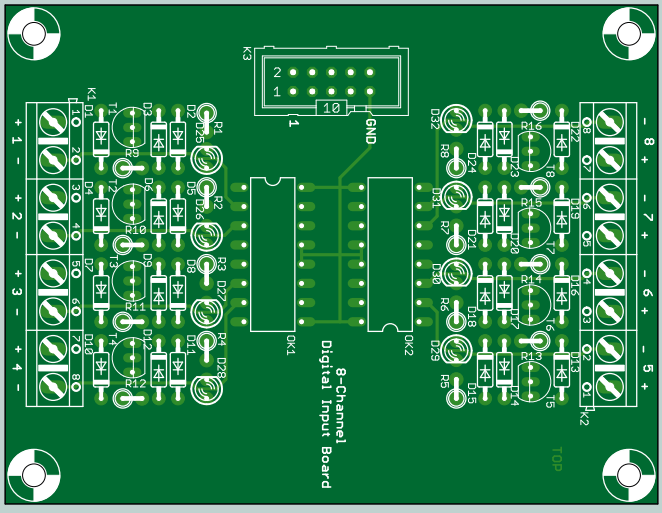
Figure 3. Les commandes de moteur à CC avec séparation galvanique peuvent se coupler par deux pour la commande de huit moteurs au maximum.

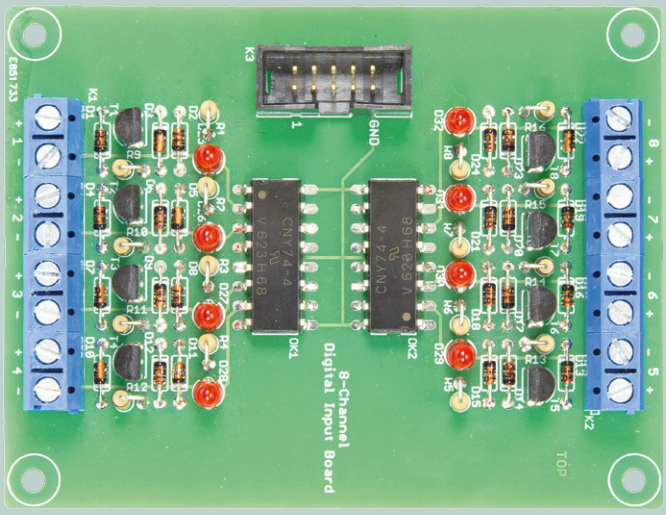
LISTE DES COMPOSANTS : 8 ENTRÉES NUMÉRIQUES

Résistances :
R1 à R8 = 220 Ω
R9 à R16 = 10 kΩ

Semi-conducteurs :
D1 à D24 = 1N4148
D25 à D32 = LED, 3 mm, rouge
T1 à T8 = BC547B
OK1, OK2 = CNY74-4

Divers :
K1, K2 = bornier à vis, à 8 pôles, au pas de 5 mm
K3 = embase à collerette à 10 broches





des LED du photocoupleur et du témoin. Pour éviter d'éventuels dégâts sur les LED, principalement à cause de tensions d'entrée plus élevées, il y a une diode 1N4148 supplémentaire contre l'inversion de polarité.

Comme LED témoins, vous pouvez choisir d'autres couleurs que le rouge du prototype. Il faut alors malgré tout tenir compte de variations dans la plage de tensions d'entrée, la tension minimale en particulier.

Côté Swiss Pi, les photocoupleurs présentent des sorties à collecteur ouvert et les E/S doivent donc être configurées en entrée avec résistance de polarisation. Une tension sur une entrée fait passer la ligne GPIO correspondante à 0 (logique inverse).

Commande de moteur à courant continu

Cette carte (**fig. 3**) peut piloter indépendamment deux moteurs à CC sous une tension de travail de 40 V maximum et jusqu'à 5 A. La cheville ouvrière de ce circuit est constituée de deux ponts en H de MOSFET TLE5206-2. Ce circuit intégré est particulièrement facile à commander, il est aussi protégé contre la surintensité et le court-circuit. Avec une $RDS_{(on)}$ de 0,2 Ω, la dissipation de puissance reste dans les normes. Il n'est pas très rapide, son temps de mise en conduction de la sortie avoisine 15 µs. Aucun souci à se faire, puisque le contrôleur MLI du Swiss Pi ne peut pas monter à plus de 1,5 kHz. Aussi, retenez que la fréquence de commande sera audible dans le moteur, surtout à des valeurs faibles de MLI.

Pour empêcher les parasites du moteur de polluer la tension d'alimentation du Swiss Pi et du RPi, il y a encore une isolation galvanique entre les ponts en H et le Swiss Pi. Pour les signaux MLI, il s'agit d'un isolateur numérique du type ADUM1200ARZ qui est à deux voies et jouit d'une bande passante supérieure à celle des photocoupleurs. Il n'est malheureusement pas disponible comme composant traversant, mais un boîtier SOIC à huit contacts se soude encore facilement à la main. Pour le retour optionnel des signaux d'erreur des ponts en H, ce sont deux photocoupleurs ordinaires qui sont utilisés.

Les commandes de moteurs peuvent être couplées par deux, ce qui donne un total de huit moteurs à CC que le Swiss Pi peut commander.

LISTE DES COMPOSANTS : COMMANDE DE MOTEUR

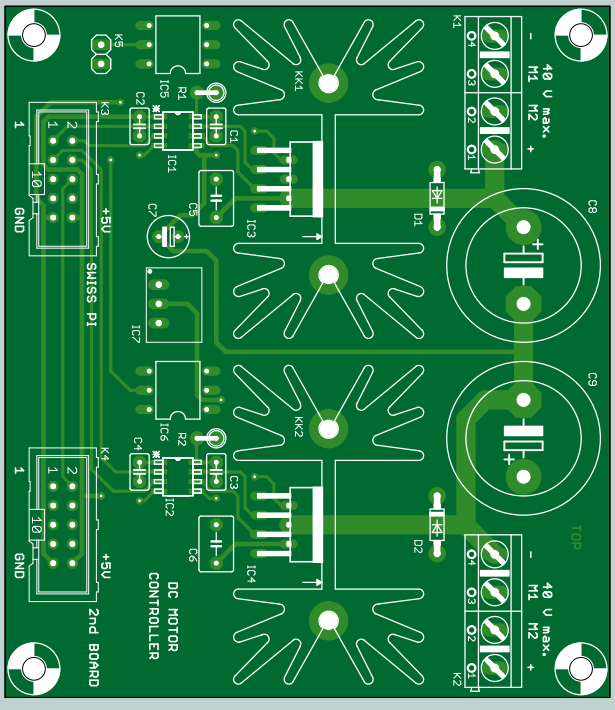
Résistances :
R1, R2 = 330 Ω

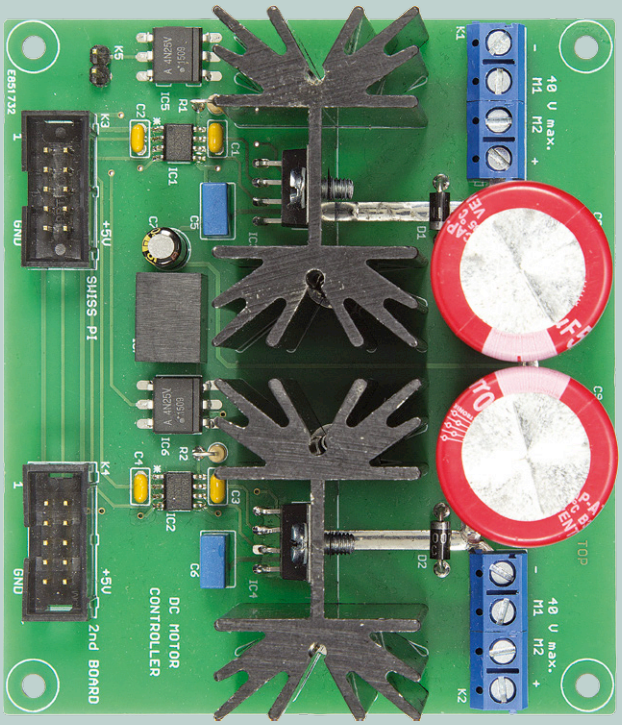
Condensateurs :
C1 à C4 = 100 nF
C5 à C6 = 470 nF/63 V*
C7 = 22 µF/50 V*
C8, C9 = 4 700 µF/50 V*

Semi-conducteurs :
D1 à D2 = 1N4007
IC1, IC2 = ADUM1200ARZ
IC3, IC4 = TLE5206-2
IC5, IC6 = 4N25
IC7 = convertisseur CC/CC SIP3, 5 V/1 A vers 40 V (p.ex. Würth 173010542)*

Divers :
K1, K2 = bornier à vis, à 4 pôles, au pas de 5 mm
K3, K4 = embase à collerette à 10 broches
KK1, KK2 = radiateur, p.ex. SK 129 38,1 STS de Fisher Elektronik (38,1 × 42 × 25 mm)

*si la commande de moteur sert pour des tensions plus faibles, on peut adapter ces composants. IC7 ne doit pas pouvoir fournir 1 A, en fait les convertisseurs CC/CC qui donnent plus de 28 V ne sont disponibles que pour 1 A.





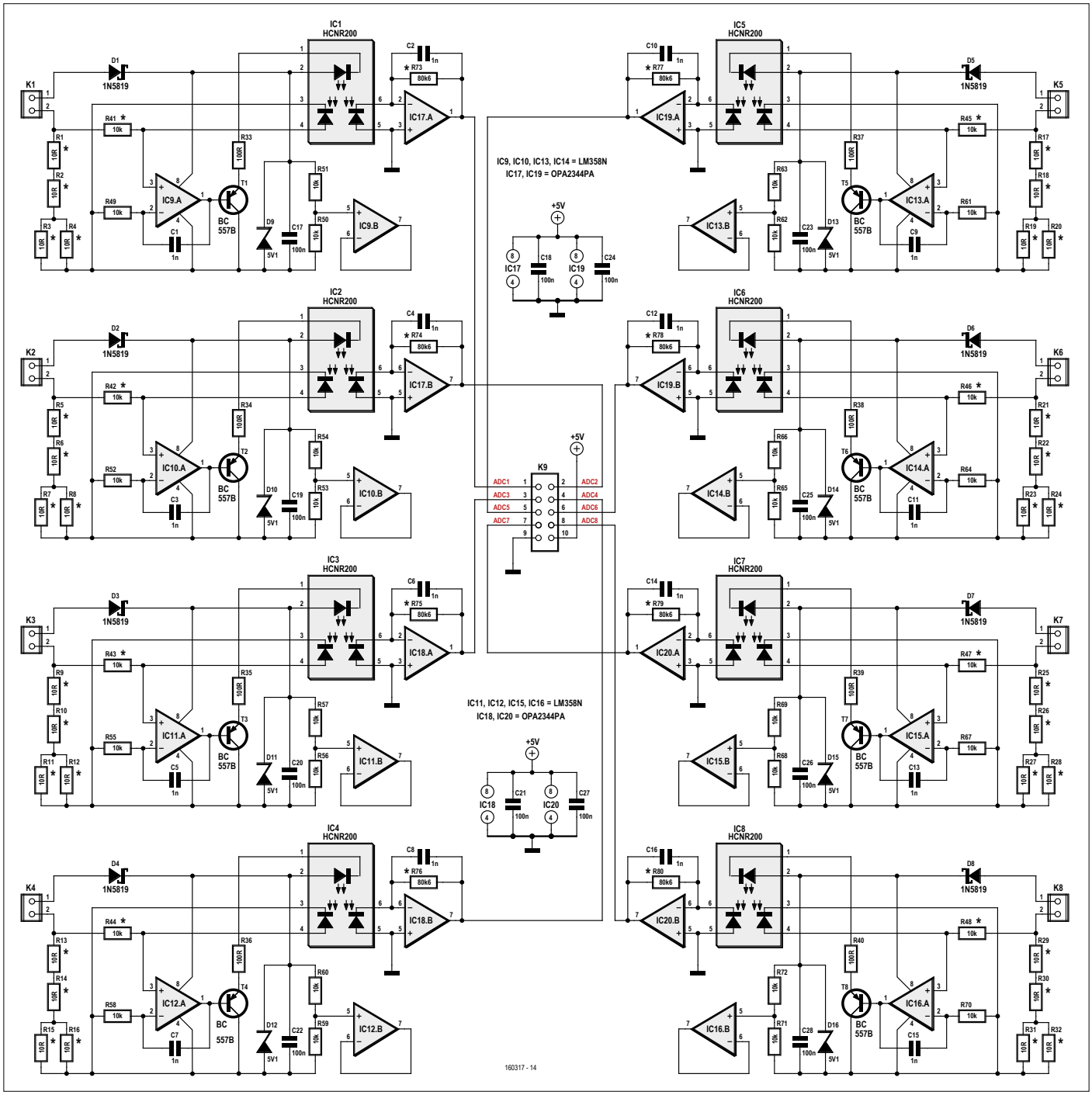


Figure 4. Le schéma complet de la carte d'interface pour boucles de courant semble énorme, mais en fait, c'est le même circuit en huit exemplaires.

Enfin, un convertisseur CC/CC fournit l'alimentation de 5 V pour la logique du côté des ponts en H. Si vous voulez utiliser une tension d'alimentation plus élevée, il importe de choisir un convertisseur CC/CC capable de supporter une tension d'entrée de 40 V.

Boucles de courant

Dans l'industrie, malgré l'arrivée de solutions purement numériques, on se sert encore régulièrement de boucles de courant pour la transmission des mesures de capteurs. La boucle de courant est simple et robuste et peut couvrir de grands rayons d'action. L'offre de ce genre de capteurs est donc aussi vaste.

L'interface proposée ici est construite à partir de photocoupleurs analogiques HCNR200 d'Avago (**fig. 4**). Ils sont composés de deux photodiodes appariées et d'une LED. À l'émission, l'une des photodiodes est utilisée en rétroaction de la puissance émise par la LED. Comme le courant dans la photodiode côté récepteur est pratiquement pareil à celui dans la photodiode de l'émetteur, la caractéristique de transfert est d'une bonne linéarité.

Le circuit reproduit presque à la lettre la note d'application d'Avago, nous n'y avons apporté que quelques minimes retouches, parmi lesquelles le remplacement des amplis op de l'émetteur par des LM385 au lieu des LM158 d'origine. Non seulement beaucoup moins chers, ils sont aussi plus faciles à se procurer. La diffé-

LISTE DES COMPOSANTS : BOUCLES DE COURANT

Résistances :

R1 à R32 = 10 Ω*
R33 à R40 = 100 Ω
R41 à R48 = 10 kΩ*
R49 à R72 = 10 kΩ
R73 à R80 = 80,6 kΩ*

Condensateurs :

C1 à C16 = 1 nF
C17 à C28 = 100 nF

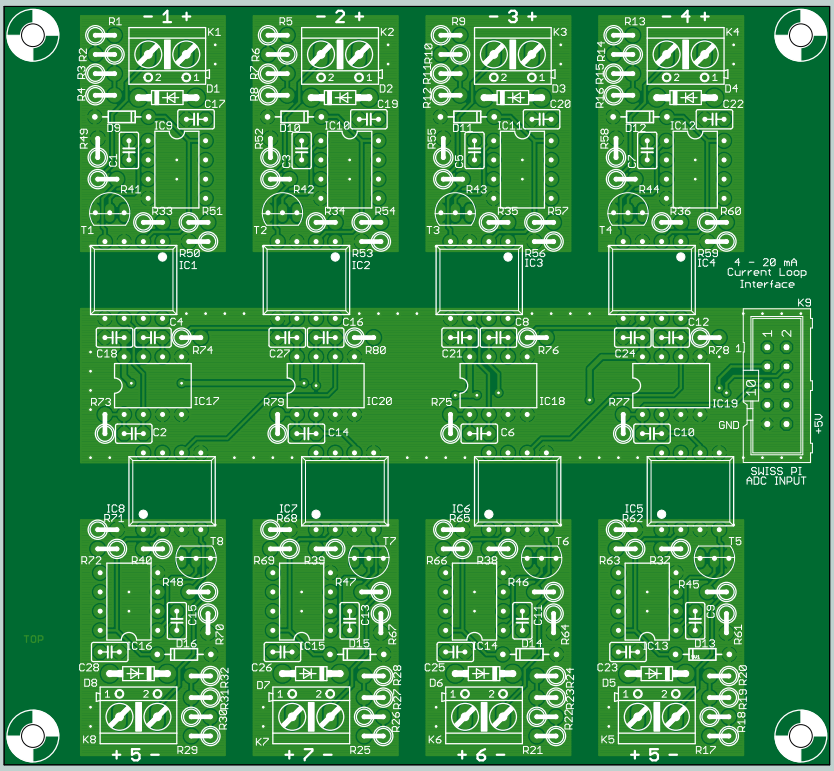
Semi-conducteurs :

D1 à D8 = 1N5819
D9 à D16 = diode Zener 5V1, 400 mW
IC1 à IC8 = HCNR200
IC9 à IC16 = LM358N
IC17 à IC20 = OPA2344PA
(LM6142BIN/NOPB)

Divers :

K1 à K8 = bornier à vis, à 2 pôles, au pas de 5 mm
K9 = embase à collerette à 10 broches

*1% à film métallique ou mieux



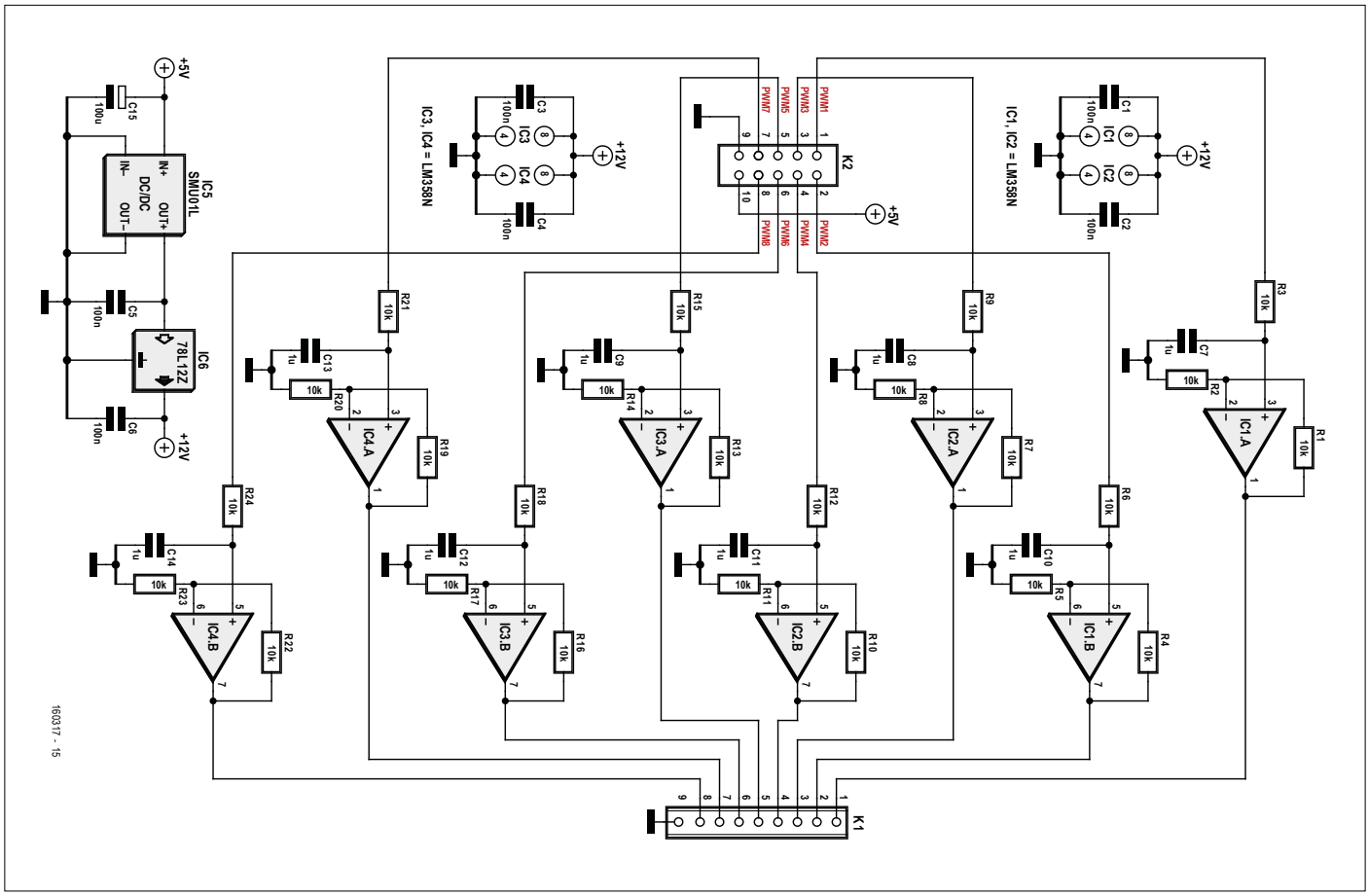
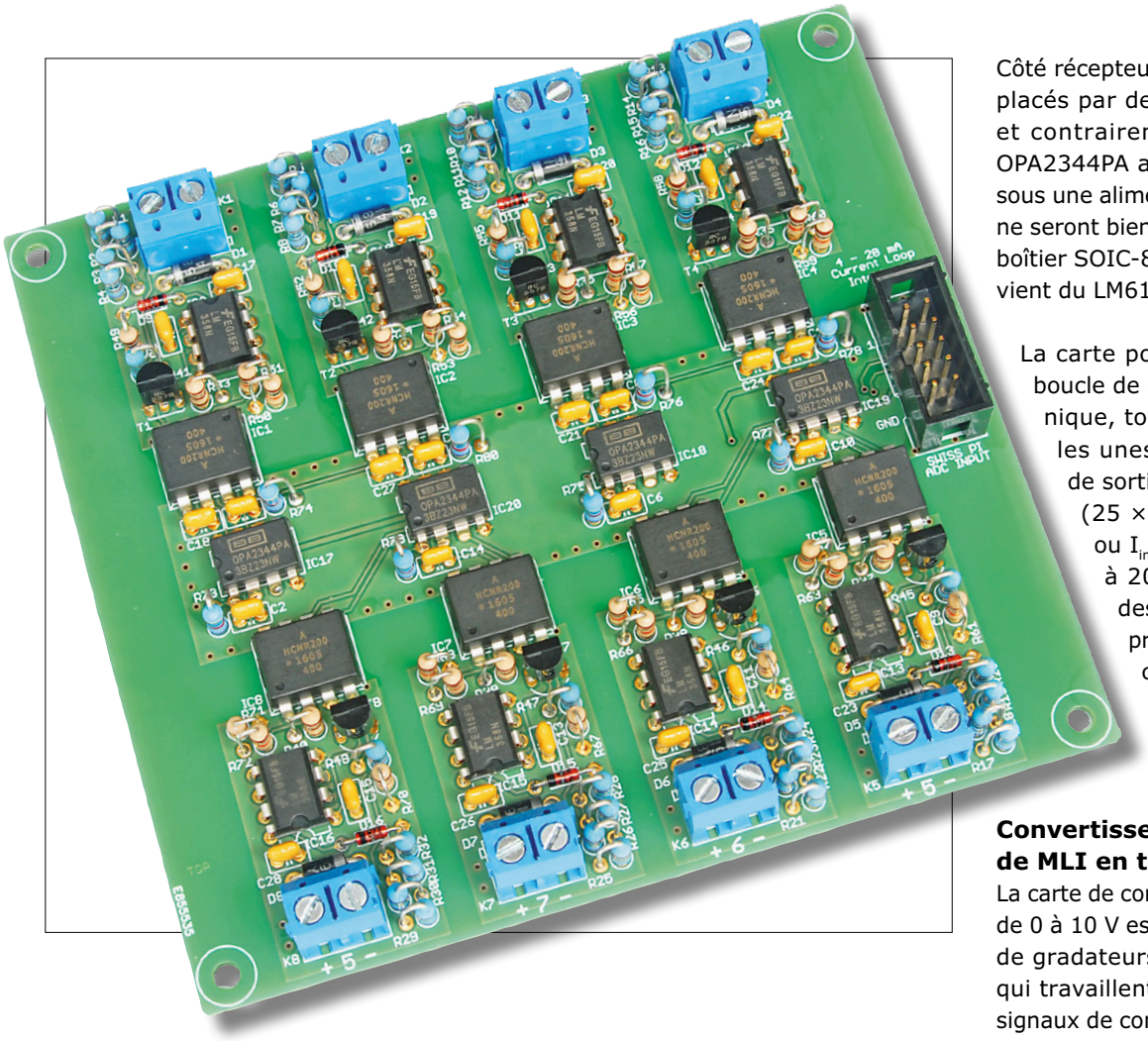


Figure 5. Le convertisseur redresse le signal MLI pour en donner une tension continue proportionnelle entre 0 et 10 V.



rence principale réside dans le domaine de température spécifié. Le LM158 tolère de −55 °C à 125 °C. Comme le RPi ainsi que le Swiss Pi sont réalisés avec des composants pour applications commerciales pour une plage de température de 0 °C à 70 °C, il n'y a aucune raison de prendre des LM158.

Côté récepteur, les LM158 ont été remplacés par des amplis op OPA2344PA et contrairement aux premiers, les OPA2344PA atteignent facilement 4 V sous une alimentation de 5 V. Hélas ! Ils ne seront bientôt plus disponibles qu'en boîtier SOIC-8, mais une autre solution vient du LM6142BIN/NOPB.

La carte porte huit interfaces pour boucle de courant à isolation galvanique, totalement indépendantes les unes des autres. La tension de sortie de l'interface vaut I_{in} × (25 × 80.600)/(10.000 + 25) ou I_{in} × 201. Le domaine de 4 à 20 mA correspond donc à des tensions de sortie comprises entre 0,804 et 4,02 V, ce qui reste dans la plage acceptable du convertisseur A/N du Swiss Pi.

Convertisseur de MLI en tension continue

La carte de conversion de MLI en tension de 0 à 10 V est destinée à la commande de gradateurs, pour LED notamment, qui travaillent généralement avec des signaux de commande dans cette plage.

Le signal MLI issu du Swiss Pi est d'abord converti en tension continue dans un filtre passe-bas, puis est tamponné et doublé par un amplificateur opérationnel de la **figure 5**.

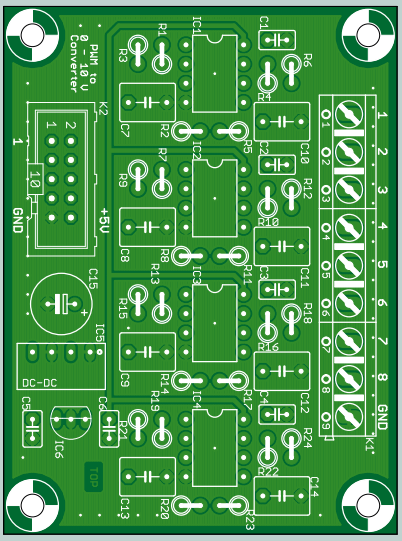
LISTE DES COMPOSANTS : MLI VERS TENSION

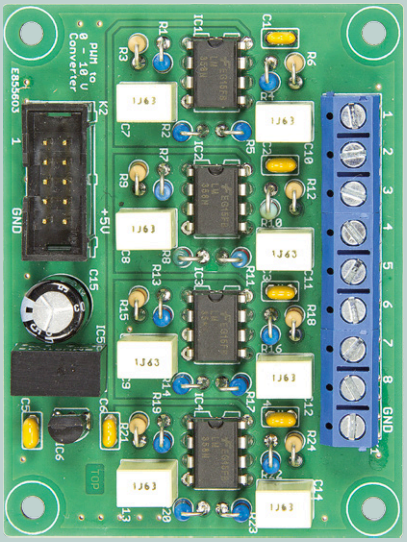
Résistances :
R1 à R24 = 10 kΩ

Condensateurs :
C1 à C6 = 100 nF
C7 à C14 = 1 µF
C15 = 100 µF

Semi-conducteurs :
IC1 à IC4 = LM358N
IC5 = SMU01L-15
IC6 = 78L12Z

Divers :
K1 = bornier à vis, à 9 pôles, au pas de 5 mm
K2 = embase à collerette à 10 broches





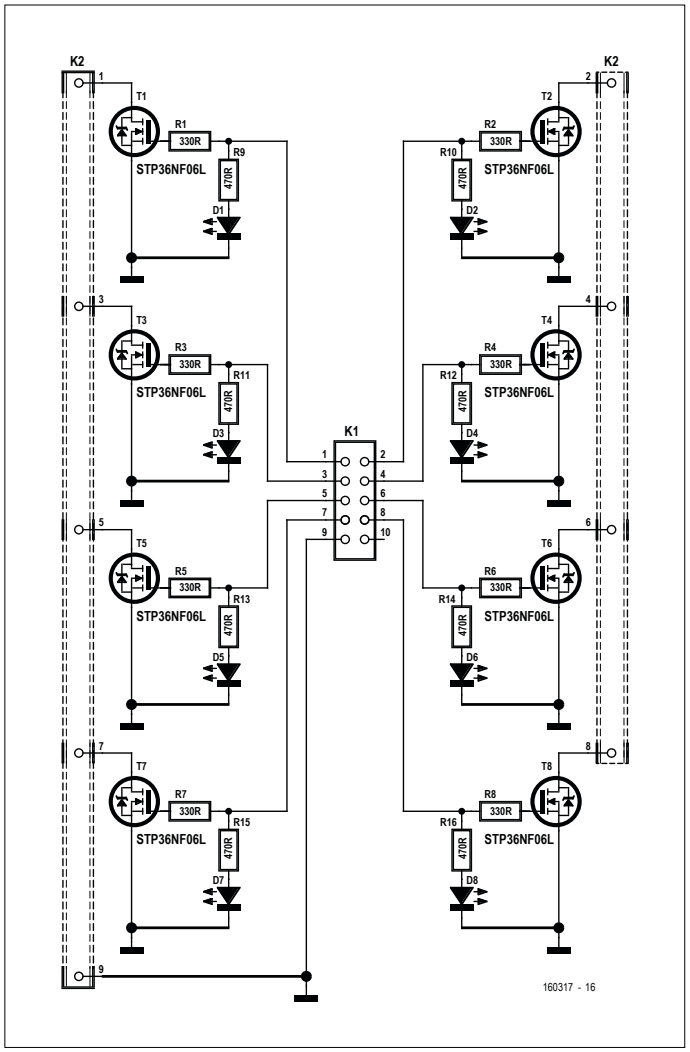


Figure 6. Pour la carte à MOSFET, le choix s'est porté sur le STP36NF06L qui supporte jusqu'à 30 A et 60 V.

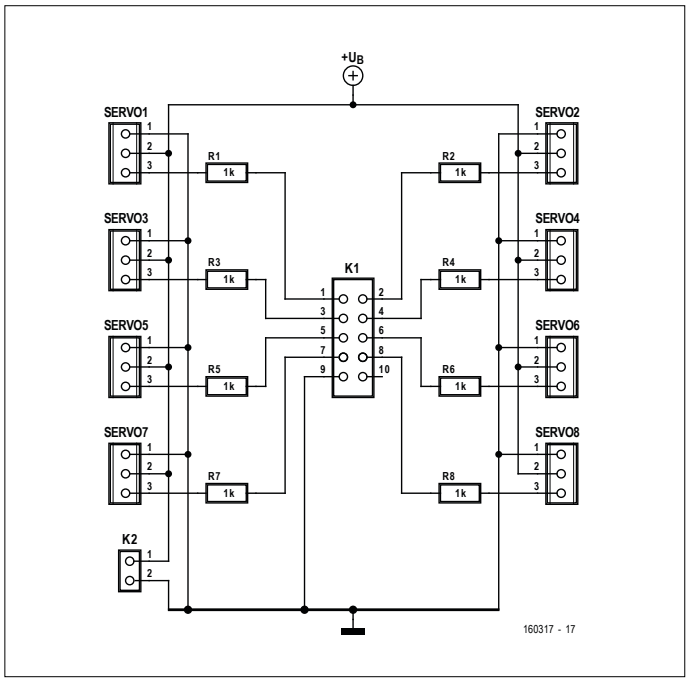


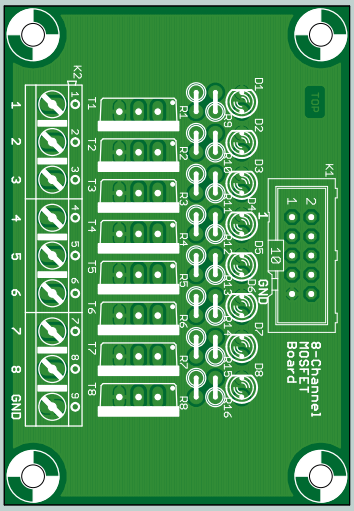
Figure 7. Avec cette simple carte, le Swiss Pi pourra piloter jusqu'à huit servomoteurs.

LISTE DES COMPOSANTS : CARTE À MOSFET

Résistances :
R1 à R8 = 330 Ω
R9 à R16 = 470 Ω

Semi-conducteurs :
D1 à D8 = LED 3 mm, rouge
T1 à T8 = STP36NF06L

Divers :
K1 = embase à collerette à 10 broches
K2 = bornier à vis, à 9 pôles, au pas de 5 mm



Comme la consommation des amplis op est faible et que leurs sorties 0 à 10 V sont en principe peu chargées, c'est un simple convertisseur CC/CC qui élève la tension pour les amplis. Il transforme le 5 V du Swiss Pi en 15 V, puis un 78L12 en fait du 12 V irréprochable. On n'a donc pas besoin d'alimentation extérieure.

Attention : l'excursion de la tension en sortie ne va pas nécessairement de 0 à 10 V, en pratique c'est de 0 V à deux fois la tension d'alimentation de la puce MLI sur le Swiss Pi.

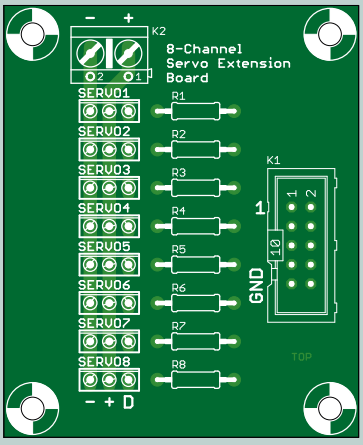
Carte à MOSFET

La carte de la **figure 6** peut servir à commuter des charges en courant continu. La commande provient soit des lignes ordinaires de GPIO du Swiss Pi, soit des canaux MLI. Le pilotage de rubans de LED RVB est un bel exemple d'application.

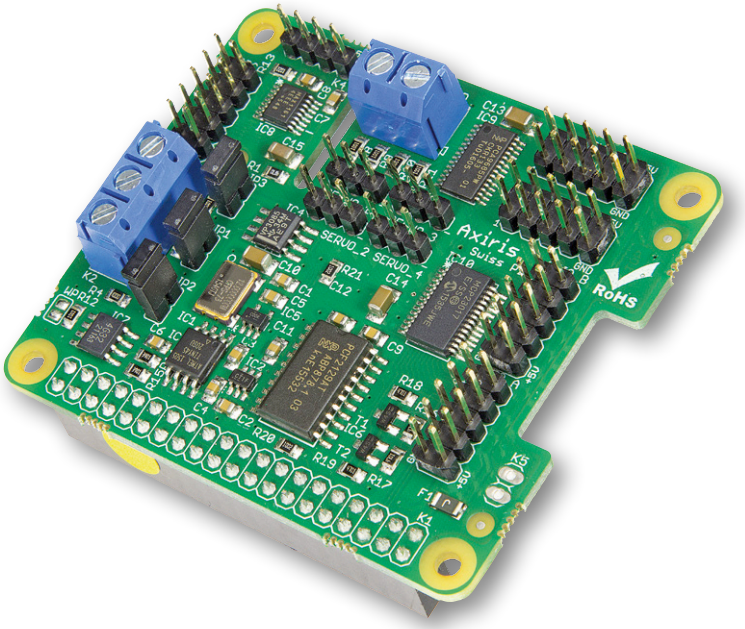
LISTE DES COMPOSANTS : SERVOMOTEURS

Résistances :
R1 à R8 = 1 kΩ

Divers :
K1 = embase à collerette à 10 broches
K2 = bornier à vis, à 2 pôles, au pas de 5 mm
K3 à K10 = embase à 3 broches, au pas de 2,54 mm



Comme MOSFET, nous avons choisi le STP36NF06L à canal N qui convient pour des tensions jusqu'à 60 V sous 30 A ($R_{DS(on)}$ = 0,032 Ω pour V_{GS} 10 V/15 A). C'est un MOSFET commandé par niveau logique, ce qui signifie qu'il est déjà en conduction totale avec une tension de grille de 5 V. Avec une tension V_{GS} de 5 V, sa $R_{DS(on)}$ est de 0,045 Ω, d'où en pratique, il peut commuter jusqu'à environ 5 A sans nécessiter de refroidissement.
Chaque canal est en outre doté d'un témoin à LED.



Carte pour d'autres servomoteurs

Cette carte-ci permet de commander plus de servomoteurs que les quatre prévus sur le Swiss Pi (**fig. 7**). Avec deux de ces cartes, c'est une ribambelle de seize servos qui s'y ajoutent. Les résistances protègent le Swiss Pi au cas où la fiche tripolaire de l'un des servos serait branchée à l'envers.

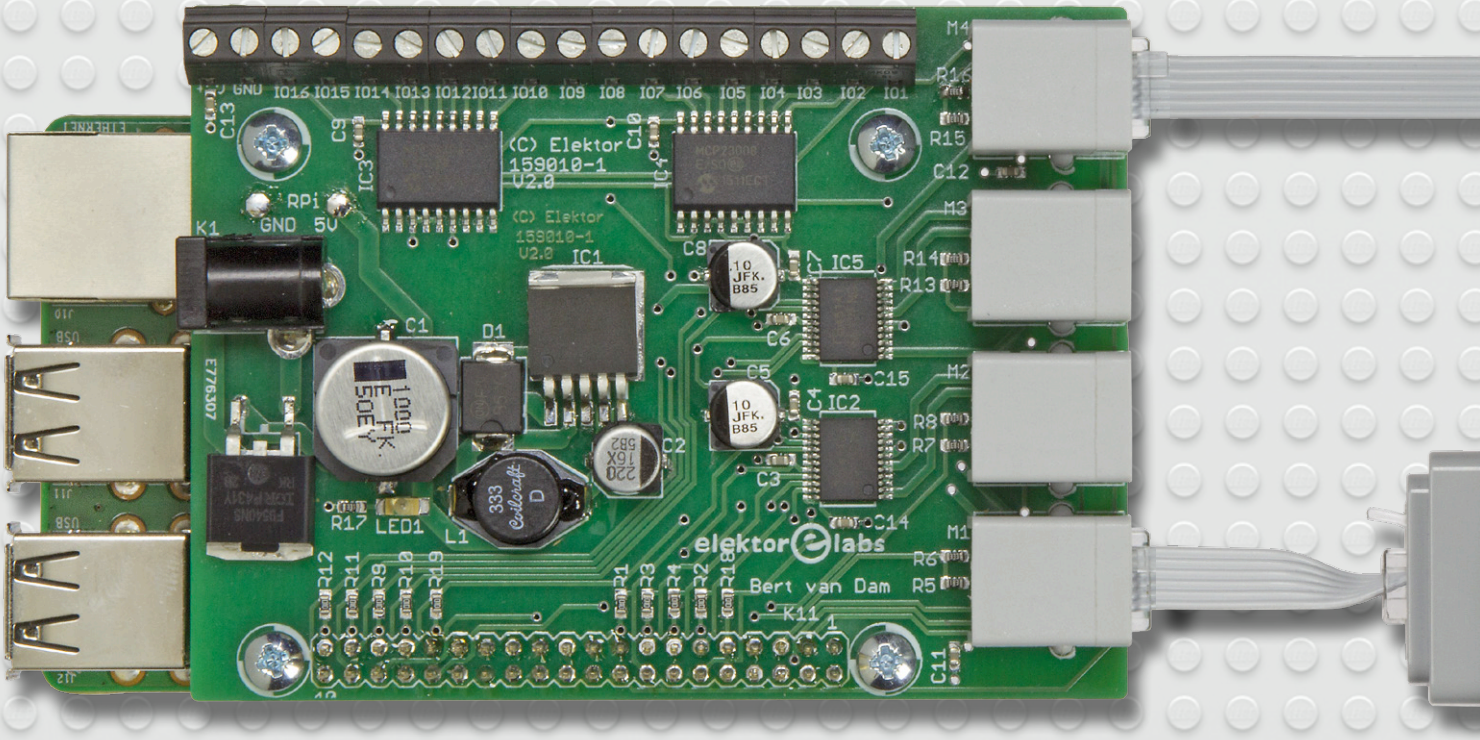
Avec les différents logiciels d'exemple parus dans le précédent article [3], vous n'avez plus que l'embarras du choix de projets qui combinent Swiss Pi et Raspberry Pi.

(160317 – version française : Robert Grignard)

Liens

[1] www.elektormagazine.fr/150584
[2] www.elektormagazine.fr/160317
[3] www.elektormagazine.fr/160237

carte de commande LEGO® pour Raspberry Pi



Cette carte *HAT* (*Hardware Attached on Top*) pour Raspberry Pi permet d'ajouter au puissant nano-ordinateur quatre voies pour des moteurs LEGO de type EV3 et seize entrées/sorties numériques tamponnées. La programmation peut se faire dans différents langages, tels que C ou Python. Pour vous mettre le pied à l'étrier, nous avons construit une « boîte inutile » avec cette carte et quelques pièces LEGO.

La carte décrite ici combine la souplesse d'emploi de deux systèmes bien connus :

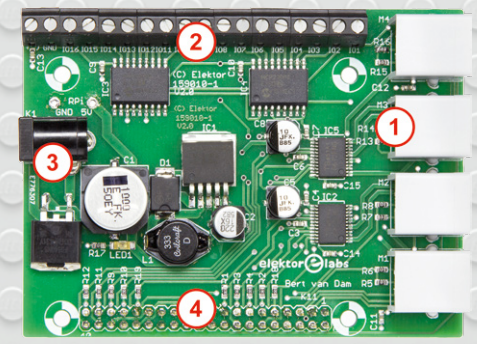


Figure 1. Connexions de la carte de commande LEGO. 1) Quatre connecteurs LEGO pour moteur ; 2) bornier à vis pour seize E/S ; 3) connecteur pour l'alimentation 9 V ; 4) recopie des broches d'E/S du Raspberry Pi.

d'une part LEGO, pour la construction mécanique, et d'autre part Raspberry Pi, pour les aspects logiciel, gestion des capteurs et communication avec l'internet.

Lorsque cette carte est montée sur un Raspberry Pi 2 ou 3, vous disposez de quatre voies pour moteur EV3 (ou équivalent), pour mettre en mouvement des maquettes LEGO. Il y a aussi seize entrées ou sorties numériques tamponnées, pour la gestion et la commande de capteurs, LED, etc. Vous ne pourrez pas utiliser les capteurs LEGO (dans ce cas, il vaut mieux utiliser le contrôleur EV3 de LEGO), mais le choix est vaste par ailleurs : infrarouge, ultrasons, température, etc. Laissez libre cours à votre imagination !

Le RPi est alimenté par la carte, une seule alimentation de 9 V suffira donc (courant de 5 A recommandé) pour le nano-ordinateur, la carte de commande, et les moteurs. Pour des applications mobiles, des accus feront l'affaire, le régulateur de tension pour le RPi est un convertisseur *Buck* à haut rendement.
Le connecteur GPIO du RPi est « ramené » sur la carte de commande LEGO, ce qui permet de connecter des composants ou capteurs I²C ou SPI additionnels. N'hésitez surtout pas à combiner projets et capteurs du livre « Raspberry Pi – 45 applications utiles pour l'électronicien » [2] avec des pièces de LEGO.

Cerise sur le gâteau, le RPi permet de communiquer avec l'internet, par câble

LEGO est une marque déposée du Groupe LEGO.

avec quatre voies pour moteur et seize entrées/sorties numériques

Bert van Dam (Pays-Bas)

ou Wi-Fi. Une clé USB/4G branchée sur le RPi vous permettra en outre de communiquer avec votre module où que vous soyez (ou presque).

Vous pourrez ainsi construire des maquettes qui combinent les moteurs LEGO et un tas de capteurs, le tout commandé et géré par un puissant nano-ordinateur, le Raspberry Pi !

Tableau 1. Points forts et faibles du LEGO EV3 et du Raspberry Pi.

LEGO – points forts	LEGO – points faibles
• Construction mécanique ; on peut réaliser tout ce que l'on veut. • Les moteurs, bien que pour des applications sérieuses il faille des servomoteurs et des vérins.	• Le peu de connexions pour des capteurs. Un robot un tant soit peu équipé a besoin de quatre capteurs ultrasonores, et ce n'est qu'un début ! • Vidéo et autres applications, auxquelles LEGO n'a pas pensé.
Raspberry Pi – points forts	**Raspberry Pi – points faibles**
• Système ouvert, très puissant avec un langage facile à apprendre, Python. Et encore plus en C. • Nombreuses connexions d'E/S, que l'on peut étendre « à l'infini » avec des circuits d'extension.	• Connecteurs d'E/S fragiles et non protégés. • Gourmand en courant.

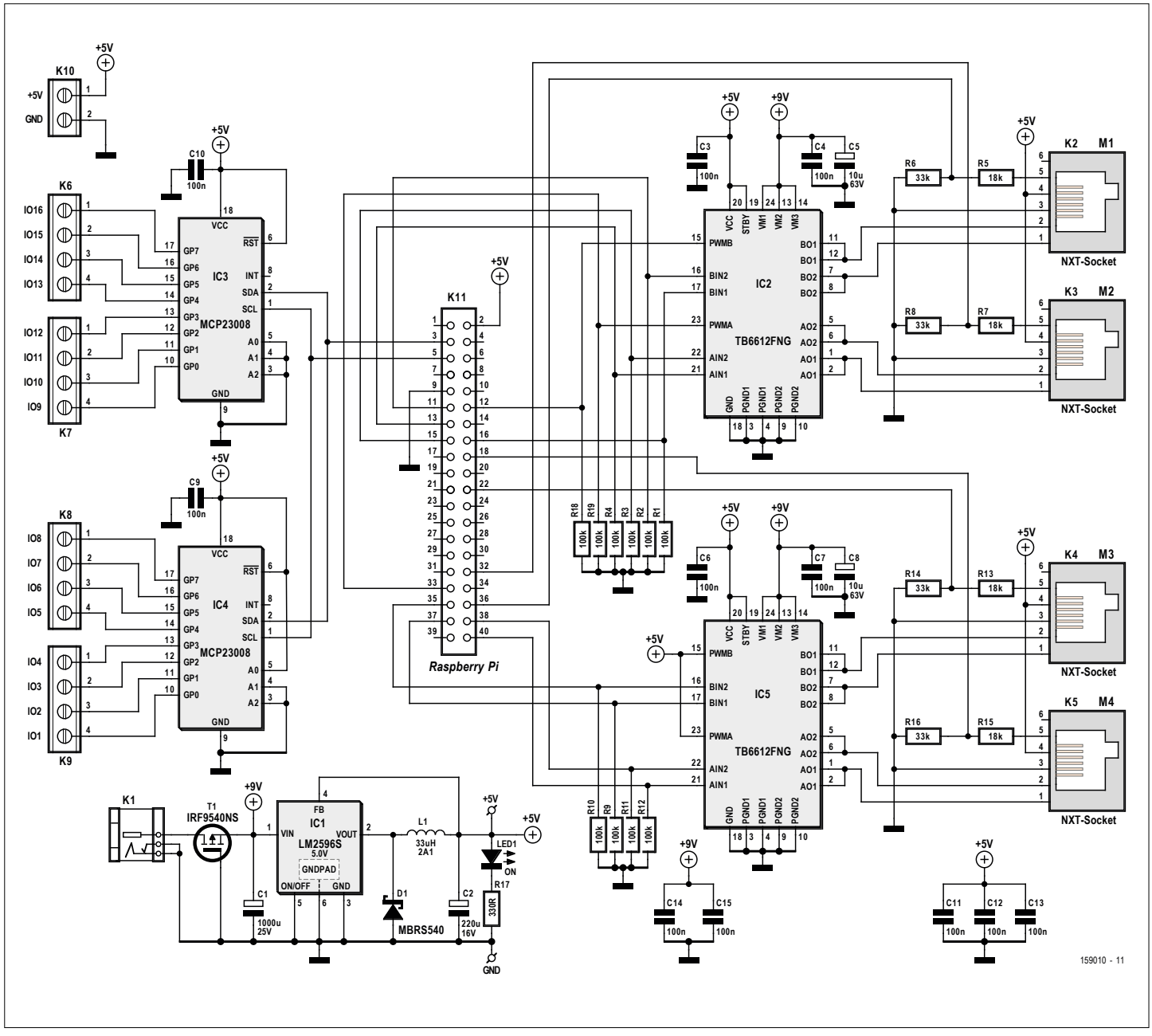


Figure 2. Le circuit comporte deux circuits de commande de moteur, deux circuits d'extension d'E/S, et un convertisseur abaisseur de tension *Buck*.

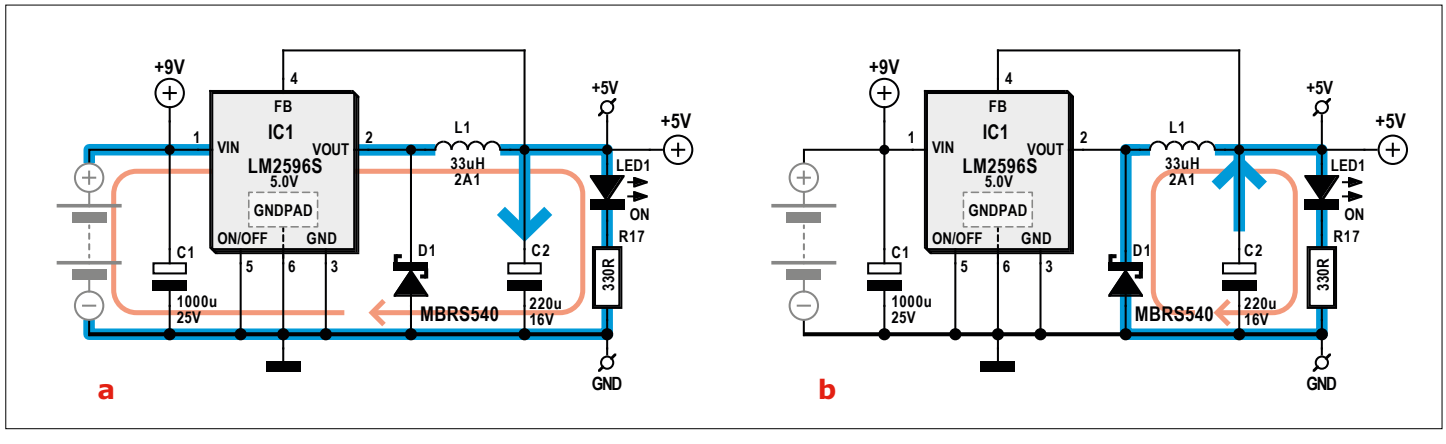


Figure 3. Convertisseur *Buck* : a) démarrage du « volant d'inertie » b) alimentation par le « volant d'inertie ».

Considérations préliminaires

La carte de commande LEGO n'est pas une simple copie de la brique EV3 ; elle combine les points forts de LEGO et ceux de RPi. Le **tableau 1** reprend les points forts et faibles des deux systèmes. Il en ressort que notre carte doit présenter les caractéristiques suivantes (et c'est bien le cas) :

- quatre voies pour des moteurs LEGO EV3 (LEGO 45502, 45503 ou compatible), toutes avec signaux de tachymétrie, et deux avec MLI (*PWM*) ;
- seize E/S tamponnées (extensibles par I²C) ;
- alimentation à haut rendement pour le RPi (convertisseur *Buck*) ;
- bibliothèque Python 2 gratuite ;
- recopie du connecteur GPIO du RPi (ce qui permet des connexions supplémentaires, telles qu'E/S protégées ou analogiques).

Résultat

Le schéma complet de la carte de commande LEGO est en **figure 2**. Les composants les plus importants sont les deux circuits de commande de moteur, des TB6612FNG de Toshiba (chacun peut commander deux moteurs, avec un courant de sortie maximal de 1,2 A), et les deux circuits d'extension d'E/S MCP23008 de Microchip, pourvus d'interfaces I²C à haute-vitesse et SPI. Ci-après l'analyse de quelques sous-ensembles intéressants du circuit.

Alimentation

Afin de n'utiliser qu'une seule source de tension, le RPi est alimenté par la carte de commande. La carte elle-même est alimentée en 9 V, fournis par des accus ou une alimentation secteur (courant nominal recommandé : 5 A).

Il nous fallait encore une manière économe de passer du 9 V au 5 V (alimentation du RPi), pour ne pas décharger les batteries trop vite. Nous avons donc opté pour un convertisseur *Buck* (hacheur série abaisseur de tension) ; il est basé sur un circuit LM2596S-5.0V de Texas Instruments (appelé par TI *Simple Switcher® Power Converter*), qui fonctionne à 150 kHz et peut fournir 3 A.

La **figure 3** illustre le fonctionnement d'un convertisseur *Buck*. La boucle formée par la diode D1, l'inductance L1 et le condensateur C2, constitue une sorte de volant d'inertie électronique. Le LM2956 commence par « pousser » la tension à fond, ce qui lance le volant d'inertie. Ce n'est pas instantané, car le condensateur doit se charger via l'inductance, dans laquelle un champ magnétique est créé par le courant. À ce moment, la diode ne conduit pas encore, et le courant passe par le LM2956.

Lorsque la tension atteint sa limite haute, le LM2956 bloque sa sortie. Le champ magnétique dans l'inductance s'effondre, la tension s'inverse, et la diode conduit. Un courant parcourt alors la diode et la charge (le courant ne peut plus passer par le LM2956, celui-ci étant bloqué), et le condensateur se décharge. Lorsque la tension atteint sa limite basse, le LM2956 se débloque et le cycle recommence.

Le LM2956 fonctionne donc en tout ou rien. Le rendement est excellent, et il n'y a aucun risque d'échauffement du circuit. Vu le découpage de la tension d'entrée, la valeur de C1 est importante aussi. C1 et C2 doivent être à faible résistance série (*ESR – Equivalent Series Resistance*).

Entrées/Sorties

Les circuits d'extension MCP23008 (IC3 et IC4) sont pilotés par I²C, et veillent à ce que les connexions d'E/S du Raspberry Pi ne soient pas utilisées directement. Ceci offre en outre quelques avantages :

- les connexions d'E/S fonctionnent sous 5 V ;
- les broches d'E/S peuvent fournir un courant plus élevé que celles du RPi ;
- en cas d'incident, le dommage sera en principe limité au seul circuit d'extension concerné.

Le courant que peut fournir chaque broche d'E/S du MCP23008 est de 25 mA, avec un maximum de 125 mA pour l'ensemble du circuit, dont la dissipation ne peut dépasser 700 mW. Ceci est valable pour chaque MCP23008 ; si vous avez besoin de « beaucoup » de courant, veillez donc à répartir la tâche entre les deux circuits.

Les broches d'E/S du RPi étant ramenées (et donc accessibles) sur la carte de commande, on peut y raccorder des composants I²C supplémentaires, il suffit que leur adresse soit différente (la carte de commande LEGO utilise les adresses '0x20' et '0x21'). On peut par ex. ajouter des circuits d'extension d'E/S, et des convertisseurs N/A ou A/N, pour produire ou lire des signaux analogiques.

Moteurs

Les deux circuits TB6621FNG (IC2 et IC5) possèdent deux ponts en H, nous pouvons donc commander un total de quatre moteurs. La consommation des moteurs LEGO est en approximation la suivante :

	petit EV3	**gros EV3**
roue libre	85 mA	80 mA
en avant lent	500 mA	1000 mA
bloqué	780 mA	1800 mA

Les moteurs ne peuvent rester bloqués que durant un court laps de temps, sinon une sécurité les met hors circuit. En outre, il vaut toujours mieux éviter un blocage : si la sécurité flanche, le moteur peut brûler.

Le TB6621FNG peut fournir 1,2 A par sortie (donc par moteur), avec une pointe de 3 A ; c'est amplement suffisant.

Si vous voulez installer une bibliothèque de pilotes pour les TB6621FNG, n'oubliez pas que les broches MLI (*PWM*) du circuit doivent être au niveau haut pour que le TB6621FNG travaille. Avec un signal MLI, cela fonctionnera toujours correctement, quel que soit le sens de rotation du moteur.

Circuit imprimé

La **figure 4** montre le dessin du circuit imprimé. Les dimensions du circuit et la position du connecteur sont choisies de telle manière que la carte s'adapte parfaitement à un RPi 2 ou 3 (doté d'un connecteur d'E/S à 40 broches). La plupart des composants sont en boîtier CMS, Elektor propose donc une version montée et testée de la carte. Vous pouvez bien entendu monter le circuit vous-même, le dessin est disponible sur la page du projet [1]. Notez que le connecteur à 40 broches (K11) est placé sous le circuit imprimé, et soudé sur le dessus. Les quatre connecteurs NXT destinés aux moteurs ne sont pas très courants, et vous devrez chercher un fournisseur sur l'internet.

Projet de démonstration : la boîte inutile

Le concept de la boîte inutile (*useless box*) est une boîte qui ne sert à rien en

Liste des composants

Résistances :

R1-R4, R9-R12, R18, R19 = 100 kΩ, 1%, 100 mW, CMS 0603
R5, R7, R13, R15 = 18 kΩ, 1%, 100 mW, CMS 0603
R6, R8, R14, R16 = 33 kΩ, 1%, 100 mW, CMS 0603
R17 = 330 Ω, 100 mW, CMS 0603

Condensateurs :

C1 = 1000 µF / 25 V, aluminium, d = 12,5 mm, h = 13,5 mm, ELPP-CP-125-135
C2 = 220 µF / 16 V, aluminium, d = 6 mm , h = 7,7 mm, ELPP-CP-063-066
C3, C4, C6, C7, C9-C15 = 100 nF / 16 V, 10%, X7R, CMS 0603
C5,C8 = 10 µF / 63 V, CMS D

Bobine :

L1 = 33 µH, 2,1 A (p.ex. Coilcraft DO3316)

Semi-conducteurs :

D1 = MBRS540, 40 V / 5 A, ELPP-DO-214AB
LED1 = LED rouge, CMS 1206
T1 = IRF9540NSPBF (MOSFET canal P), D2-PAK
IC1 = LM2596S-5.0 (convertisseur *Buck*), TO-263-5
IC2, IC5 = TB6612 (commande des moteurs), 24-SSOP (Digikey TB6612FNGC8ELCT-ND)
IC3, IC4 = MCP23008 (extension d'E/S), SOIC-18

Divers :

K1 = connecteur d'alimentation, 12 V / 3 A, broche centrale Ø 1,95 mm
K2-K5 = connecteur femelle NXT
K6-K9 = bornier à vis, 4 broches, pas de 3,81 mm
K10 = bornier à vis, 2 broches, pas de 3,81 mm
K11 = connecteur à 2×20 broches RPi, extra long (p.ex. Kiwi-electronics ADA-1979)
Circuit imprimé réf. 159010-1 (www.elektor.fr)

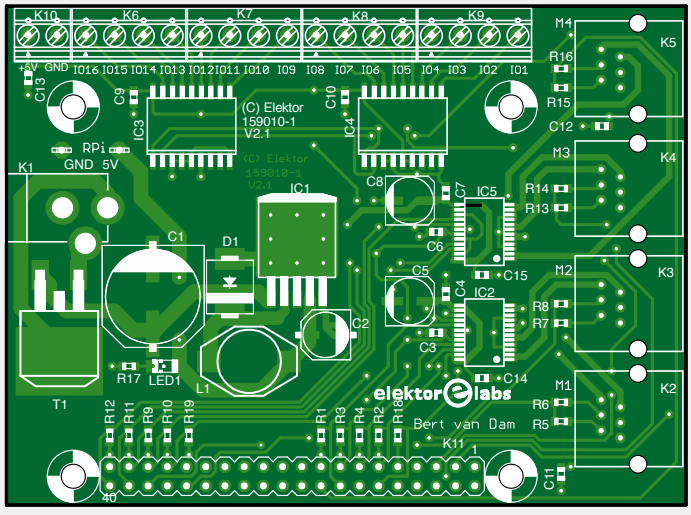


Figure 4. Dessin du circuit imprimé. Le connecteur à 40 broches (K11) est installé sous le circuit.

particulier, avec un bouton dessus. Avec notre boîte, lorsque vous actionnez l'interrupteur, un « doigt » sort de la boîte et actionne à son tour l'interrupteur (dans l'autre sens). Nous n'avons conçu que le circuit interne, à vous de construire la boîte ; vous pouvez en faire une en bois ou carton.

La vidéo et les photos disponibles sur la page du projet ne vous montrent d'ailleurs que la partie mécanique, sans la boîte. Vous pourrez en tout cas saisir le fonctionnement du dispositif.

De quoi avez-vous besoin ?

- La carte SD du livre « Raspberry Pi – 45 applications pour l'électronicien » ;
- les téléchargements gratuits de cet article (dont la bibliothèque et le code source du projet de démonstration) ;
- un moteur LEGO (« gros » moteur de type 45502) avec câble, et quelques pièces LEGO (disponibles dans la boîte LEGO Mindstorms EV3) ;
- un Raspberry Pi 2 ou 3 ;
- la carte de commande LEGO (**fig. 4**) ;
- deux résistances (1 et 10 kΩ, ¼ W) ;
- un petit interrupteur facile à actionner.

Le matériel

Nous utilisons un « gros » moteur LEGO de type 45502 – que l'on peut trouver entre autres dans une boîte LEGO Mindstorms EV3 (voir contenu à la **figure 5**). Une courte vidéo (voir en [1]) montre comment réaliser un bras articulé. L'interrupteur est placé sur le montage, de telle manière que le bras en plastique noir puisse l'actionner.

L'ensemble est ensuite raccordé à l'alimentation via les deux résistances, et à la broche d'E/S 16 (**fig. 6**). La résistance de 10 kΩ seule suffirait, mais la seconde est une sécurité. Sans cette résistance de 1 kΩ, on pourrait endommager les E/S si on invertissait entrée et sortie et raccordait ce fil à masse.

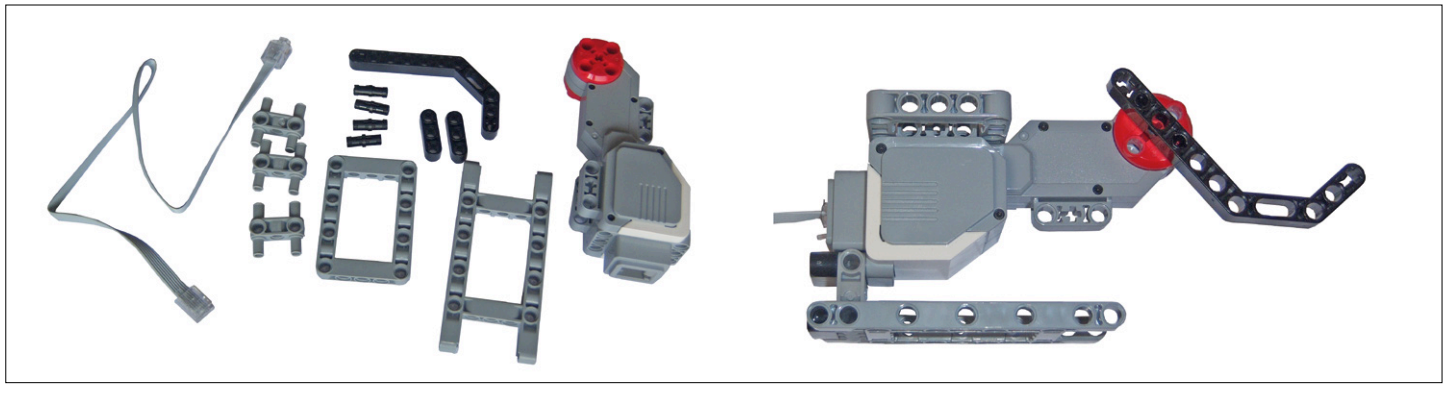


Figure 5. Pièces du bras articulé et sa version finale.

Le logiciel

Le logiciel est étonnamment simple, mais attention : la bibliothèque attend des programmes en Python 2. Toutes les informations sur cette bibliothèque et les fonctions vous sont fournies gracieusement dans un manuel, téléchargeable en [1]. Il faut d'abord installer la bibliothèque 'rpirobot', puis entrer l'instruction 'execfile'. Les ports de moteur sur la carte de commande LEGO sont alors correctement initialisés et les tâches MLI (*PWM*) activées. Ensuite nous configurons en entrée la broche d'E/S 16 de la carte de commande, vu que l'interrupteur y est raccordé.

```
execfile('rpirobot.lib')

# Switch on io 16, input
iodir(16,1)

print "Useless Box Program running"
```

Nous utilisons une structure de contrôle '*try/except*' (**listage 1a**), de telle manière que le programme attende que l'utilisateur actionne l'interrupteur ou appuie sur 'Ctrl-C' (arrêt).
Lorsque l'interrupteur est actionné, le programme attend 0,5 s (pour que l'utilisateur ait le temps de retirer son doigt), puis fait tourner le bras LEGO avec 'PWM 60' (60% de la vitesse maximale). Nous sommes dans une boucle pour exécuter des tâches (*threads*), mais comme rien ne se passe, c'est peu utile. Nous avons donc programmé une fonction 'sleep', d'une durée de 1 µs ; personne ne remarquera ce retard. Dès que l'interrupteur est actionné par le bras, le frein moteur est enclenché.

L'étape suivante **(listage 1b**) sert à ramener le bras LEGO à sa position de repos, à vitesse réduite ('PWM 30', 30% de la vitesse maximale). Il n'y a pas d'interrupteur de fin de course pour stopper le mouvement, le programme utilise donc les données de tachymétrie. Le programme compte les pas, et arrête le moteur après 100 (28% d'un cercle) avec la commande 'float'. Le bras se trouve alors à environ 1 cm du « sol ».

La structure '*except*' (**listage 1c**) traite l'action 'Ctrl-C' de l'utilisateur. Le programme envoie l'interruption à la bibliothèque RPi et s'arrête ensuite lui-même.

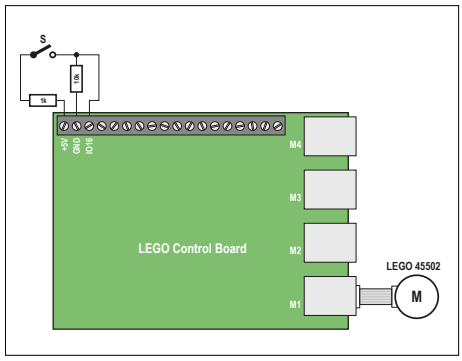


Figure 6. Schéma de connexion de l'interrupteur et des deux résistances.

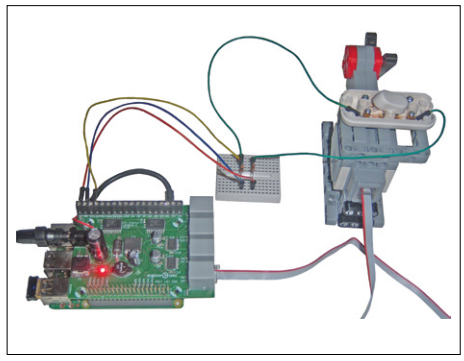


Figure 7. Le montage complet.

Sans cette structure, le programme s'arrêterait aussi avec 'Ctrl-C', mais la bibliothèque resterait ouverte, ce qui n'est pas souhaitable.

Pas à pas

- Mettre le bras LEGO dans sa position de repos, c'est-à-dire le plus loin possible de l'interrupteur (par ex. contre la table où repose le bloc moteur).
- Démarrer le RPi, et patienter jusqu'à ce que le processus de démarrage soit terminé.
- Copier le dossier *test* dans le RPi (par ex. avec WinSCP).
- Entrer dans le dossier *test* avec l'instruction 'cd test'.
- Exécuter 'python uselessbox.py'.
- Attendre jusqu'à ce que le texte 'Useless Box Program running' apparaisse sur l'écran du RPi, presser l'interrupteur, et retirer le doigt. Le bras LEGO actionne à nouveau l'interrupteur.
- Répéter cette dernière étape autant que l'on veut, puis arrêter le programme avec 'Ctrl-C'.

C'est un exemple d'utilisation très simple, mais la carte de commande LEGO offre bien plus de possibilités, avec ses quatre moteurs et ses seize broches d'E/S. ⏮

(150597 – version française : Jean-Louis Mehren)

Liens

[1] Page du projet : www.elektormagazine.fr/150597

[2] Livre « Raspberry Pi, l'alliance de la programmation et de l'électronique – 45 projets utiles pour l'électronicien », Elektor, ISBN 978-2-86661-196-5 : www.elektor.fr/rpi

Listage 1a.

```
try:
  while 1:
    # wait for the user
    # to push switch on
    if ioread(16):
      # push switch off
      time.sleep(0.5)
      reverse(1)
      pwm(1,60)
      while ioread(16):
        time.sleep(0.000001)
      brake(1)
```

Listage 1b.

```
    # retract lever
    forward(1)
    pwm(1,30)
    counter1=0
    lastpin1=tacho(1)
    while counter1<100:
      time.sleep(0.000001)
      if tacho(1)!=lastpin1:
        counter1+=1
        lastpin1=tacho(1)
    # float motor
    float(1)
```

Listage 1c.

```
except:
  # user aborts program with
Ctrl-C
  print "Program aborting"
  abort()
  print "Done"
```

webradio à tubes fluorescents (1)

la technique actuelle s'affiche à l'ancienne

C'est chouette une radio de cuisine qui lit les CD, donne l'heure et mesure le temps de cuisson sans être encombrante. Mais quand elle ne donne plus que du bruit, il faut la remplacer. Pourtant, son bel afficheur était bien pratique, complet et compact. Pourquoi ne pas le recycler dans une radio d'un nouveau genre ?

Michael Buser (Allemagne)

C'est une bonne idée, surtout quand on est dans l'électronique. Et pour passer du désir à la réalisation pratique, il suffit de faire une recherche sur l'internet pour trouver des études détaillées de ce type d'afficheur.
Mais à peine avais-je puisé aux bonnes sources tout le nécessaire, que le VFD (*Vacuum Fluorescent Display*) a été réduit à l'état de FD, je l'ai laissé choir, ce qui a malheureusement fait se dissiper le Vide dans l'atmosphère ☹.
Par bonheur, on trouve de pareils afficheurs dans les boutiques de déstockage, si bien que j'ai pu acquérir à bon compte un nouveau vieux, un VFD du type FV651G (**figure 1**), superbe, bien que pas tout à fait aussi beau que l'ancien.

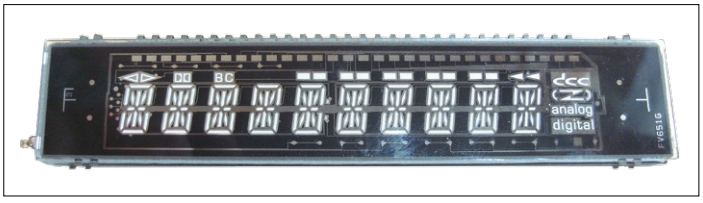


Figure 1. Le VFD FV651G est l'afficheur fluorescent utilisé.

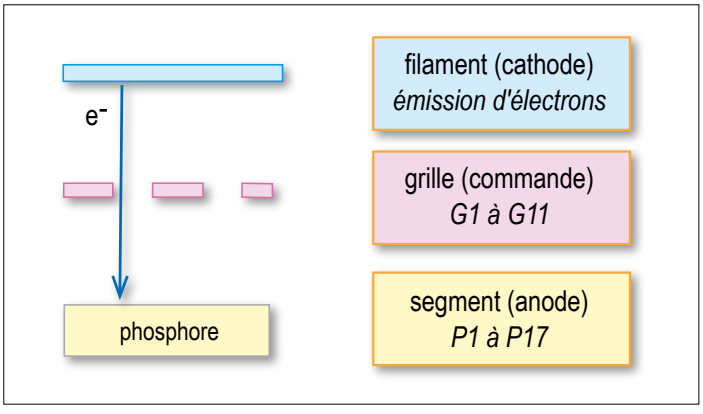


Figure 2. La structure du VFD. Les segments d'anode couverts de phosphore s'éclairent sous la projection d'électrons.

Un VFD, comment ça marche ?

Les VFD reposent sur le même principe que les tubes triodes (cf. **figure 2**), mais appliqué à chaque segment lumineux. Les détails de construction se trouvent sur [1] et [2]. En gros, ils travaillent comme suit : comme dans les autres tubes, le filament assure le chauffage de la cathode pour produire l'émission d'électrons. Les anodes sont constituées par les segments de l'afficheur, porteurs d'une couche fluorescente. Les électrons incidents provoquent la phosphorescence. Le potentiel de grille commande le flux d'électrons. Négatif, il empêche de s'allumer tous les segments, d'habitude au nombre de dix-sept. Positif, il leur permet de s'allumer, mais un potentiel d'anode négatif en repousse les électrons, c'est la manière de sélectionner les segments à éteindre. En revanche, si le segment est assez positif, les électrons le frappent et sa couche est excitée. Les contacts de grilles G1 à G12 déterminent les positions adressées et les contacts P1 à P17, chacun des segments.
Selon sa fiche technique, le type FV651G demande une tension alternative de 3,7 V pour le chauffage et 32 V comme tension de grille. C'est cette dernière qui rend pesante la commande. L'afficheur possède onze positions, la dernière à droite présente des symboles spéciaux adaptés à l'usage premier auquel il était destiné. Chaque position peut avoir jusqu'à 17 éléments lumineux à commander séparément ; le double point, 17e segment, n'est disponible qu'à une seule position.
Comme tous les segments sont interconnectés, il faut les piloter en multiplex. Cela représente donc un tableau de 11 colonnes sur 17 lignes, à réaliser de préférence par logiciel.

Chauffage

Dans l'ancienne radio de cuisine, c'est un transformateur secteur avec un secondaire à prise médiane qui alimentait les filaments. Comme la nouvelle est normalement tributaire d'un bloc secteur indépendant de 12 V, il fallait trouver une parade. Cette tension alternative ne doit pas forcément être à 50 Hz. La solution est à la **figure 3**, avec ce bon vieux temporisateur NE555 et un transformateur bobiné maison.
Originalité par rapport au circuit habituel, D1 produit un rapport cyclique presque symétrique. Le transformateur est bobiné sur un noyau de ferrite en matériau N27 d'un diamètre extérieur de 16 mm. Côté primaire, il y a 40 spires de fil de cuivre émaillé, 2 × 18 au secondaire. En charge, il fournit un peu plus

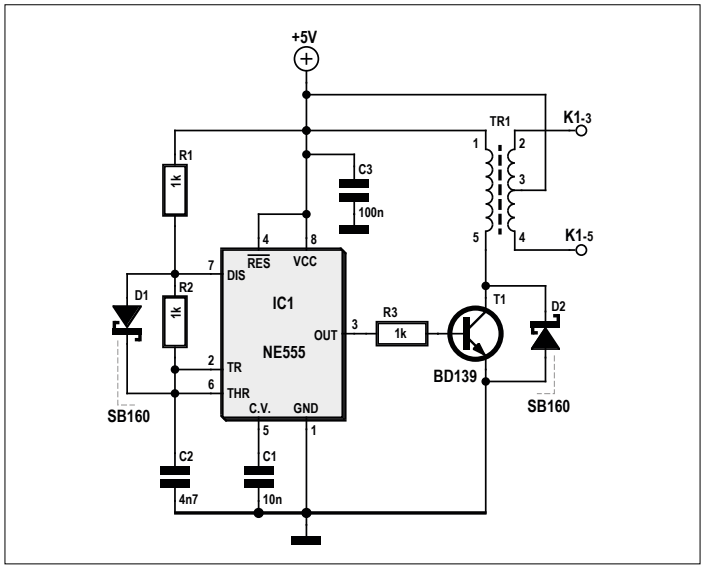


Figure 3. Le circuit d'appoint conçu par Kerry D. Wong alimente le chauffage.

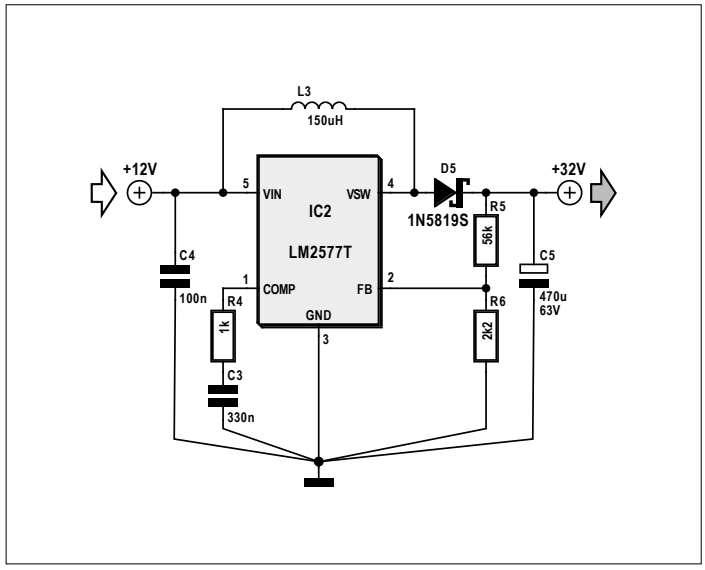


Figure 4. Autre circuit d'appoint pour produire la tension anodique avec un convertisseur survolteur.

de 2 V_{eff} sur chaque secondaire. Le circuit a été développé par Kerry D. Wong [3]. Il ne faut pas relier à la masse la tension de chauffage pour éviter l'apparition de lumières fantômes sur des segments non activés, mais polariser la prise médiane du transformateur au +5 V.

Tension anodique

Il faut aussi partir du bloc de 12 V pour créer la tension d'anode au moyen d'un convertisseur survolteur (**figure 4**). Seules les valeurs des résistances diffèrent du circuit de base de la fiche technique du LM2577.

Selon le constructeur, on calcule la tension de sortie par la formule :

$$V_{OUT} = 1{,}23\ V \times (1 + R5 / R6)$$

Les valeurs choisies pour R_5 et R_6 conduisent à la tension anodique souhaitée de 32,5 V.

Commande de grilles

Il y a en tout 28 électrodes de l'afficheur à piloter : P1 à P17 et G1 à G11. À cause de la haute tension d'anode de 32 V, il convient d'utiliser des tampons à tension de sortie élevée. La puce UDN2981 supporte 50 V et convient très bien ici ; avec huit pilotes à bord, quatre boîtiers suffiront. On les commande par le niveau usuel de 5 V.

Pour ne pas devoir sacrifier à cette fin toutes les lignes d'E/S de l'ATmega32 prévu, on intercale chaque fois un verrou à 8 bits. Ils sont groupés par quatre et commandés en parallèle ; les groupes sont choisis par le signal de validation LE (*Latch Enable*). On réduit ainsi le nombre de sorties nécessaires au pilotage de l'afficheur à douze, 8 bits de données et 4 signaux LE. Le contrôleur peut encore interroger quelques composants. Le circuit complet de la section affichage se trouve dans le document *Muxer_Schaltplan.pdf* dans le fichier d'archive Muxer.zip (téléchargement en [4]). Les fichiers Eagle correspondants y sont aussi.

Micrologiciel pour ATmega

Pour le microcontrôleur, l'afficheur est représenté par une valeur de 32 bits répartis sur quatre octets, un par verrou. Mais en réalité, sa principale tâche est de transférer ces quatre octets sur le port à 8 bits, à la fréquence de multiplexage, et de produire les signaux de commande des verrous. Le **tableau 1** donne la liste des significations des bits dans les mots de 32 bits. On n'utilise pas les quatre bits de poids le plus faible. Les définitions correspondantes sont dans les fichiers *fv651.c* et *fv651.h* du téléchargement. La structure de données `TDispControl` résume les variables requises pour la commande :

```
typedef struct {
  uint32_t data;
  uint8_tgitter;
  uint16_t anode;
} TDispControl;
```

La variable `data` contient la trame de 32 bits pour les quatre verrous, `gitter` est le nombre qui désigne la position actuelle dans l'afficheur (de 0 à `MAX_DIGITS`). La variable `anode` contient la trame de bits pour les 17 segments. C'est la routine `flush_VFD()` du **listage 1** qui transfère le contenu de `data` dans les quatre verrous.

Tableau 1. Signification des 32 bits

octet	3								2								1								0							
bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
électrode	G11 à G01											P1 à P17																	inutilisés			
fonction	10	9	8	7	6	5	4	3	2	1	0			a	b	f	k	j	h	m	g	n	p	r	c	e	d					
	position (grille)											segments (anodes)																				

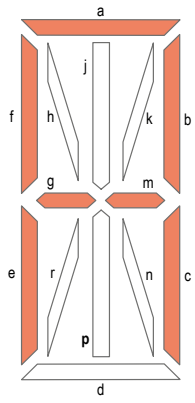


Figure 5. Pour que les segments s'allument, il faut que les bits correspondants du tableau 2 soient au niveau haut (1).

Tableau 2. Commande des segments de la figure 5

bit	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
électrodes	P1 à P17																
segments				a	b	f	k	j	h	m	g	n	p	r	c	e	d

On découpe le mot de 32 bits en quatre octets adressés successivement au port A, lequel est relié aux entrées des verrous. Après chaque transfert, il faut donner une impulsion à l'entrée de validation du verrou concerné pour que l'octet voulu y soit mémorisé. L'impulsion produite par un `nop` dure 760 ns. Avec une horloge à 16 MHz, l'opération complète prend 9 µs.

Tableau ASCII

Reste à savoir comment les trames de bits pour les caractères à afficher arrivent dans la variable `data`. C'est la fonction `mux_VFD()` qui s'en occupe. Pour que la lettre « A » s'inscrive correctement sur l'afficheur, comme à la **figure 5**, il faut allumer les segments a, b, c, e, f, g et m. On utilise pour cela une table de signes qui contient la formule adéquate de bits pour chaque signe à montrer. Le **tableau 2** donne l'exemple de la lettre A. La valeur décimale correspondante est donc 14 534. Le tableau contient ainsi une valeur de 16 bits pour chaque signe utilisé. Il est également repris dans le téléchargement, on peut le modifier au besoin.

Les mots de 16 bits qui en découlent sont stockés dans la mémoire du microcontrôleur et leurs codes ASCII servent d'index. Pour des raisons de place, on ne peut pas inclure dans le tableau résultant tous les signes possibles, il convient donc de vérifier qu'y sont représentés tous ceux réellement nécessaires. Le code du tableau ASCII ainsi que les fonctions de vérification se trouvent dans le **listage 2**.

En appelant la valeur ASCII de A = 65, la deuxième des trois conditions `if` trouve et renvoie la valeur dans la table située à la position 65 – LUT_A + *LUT_A_OFFSET* = 10.

On voit dans les minuscules un certain nombre de zéros. Petit exercice : remplissez le tableau pour que toutes les minuscules soient affichées…

Là, il faut encore un peu chipoter sur les bits pour qu'ils viennent aussi à la position voulue dans des mots de 32 bits. Dans la structure `TDispData`, `digits` contient les dix caractères à afficher et `symbols` un bit pour chaque symbole spécial affiché en 11e position.

Listage 1. Traitement des mots de 32 bits

```
void flush_VFD() {
    uint8_t data;

    data = (ctrl.data & 0xFF);          //1st byte
    PORT_VFD_DATA = data;
    LE_VFD_0_ON;                        //impulse with nop =
    asm volatile („nop“);               //760ns
    LE_VFD_0_OFF;                       //measureds with scope

    data = ((ctrl.data >> 8) & 0xFF);   //2nd byte
    PORT_VFD_DATA = data;               //delay between 1st and 2nd impulse =
    LE_VFD_1_ON;                        //2us
    asm volatile („nop“);
    LE_VFD_1_OFF;

    data = ((ctrl.data >> 16) & 0xFF);  //3rd byte
    PORT_VFD_DATA = data;               //delay between 1st and 3rd impulse =
    LE_VFD_2_ON;                        //4.7us
    asm volatile („nop“);
    LE_VFD_2_OFF;

    data = ((ctrl.data >> 24) & 0xFF);  //4th byte
    PORT_VFD_DATA = data;               //delay between 1st and 4th impulse =
    LE_VFD_3_ON;                        //7.5us
    asm volatile („nop“);
    LE_VFD_3_OFF;
}
```

```
typedef struct {
  uint8_tdigits[MAX_CHARS];  // un pour chaque position
  union TSymbolssymbols;
  // afficher ou non symbole spécial
} TDispData;
```

Le temporisateur appelle toutes les 5 ms la fonction `mux_VFD()`. Celle-ci lit les caractères dans `digits`, produit le motif de bits et envoie les caractères à l'afficheur. Cette routine est assez volumineuse, aussi je vous renvoie aux commentaires du code dans le téléchargement.

La variable `display` du type `TDispData` constitue avec les fonctions du **listage 3** l'interface pour l'unité d'affichage.

Présentation des caractères

L'instruction suivante donne un exemple d'inscription de la lettre A à la quatrième position de l'afficheur :

```
display.digits[4] = 65
```

L'appel de la fonction `setVFDDisplayData(4, 'A')` exécute la même chose, mais est un peu plus ordonné et effectue une vérification de la plage de valeur. Les fonctions de commande de l'afficheur se trouvent dans les fichiers *fv651g.c* et *fv651g.h* du téléchargement. Le code source complet est un projet Studio 6.2 d'Atmel dans le fichier *Software_1.zip*.
Ce micrologiciel complète la section affichage de la radio de

Listage 2. Tableau ASCII et fonctions de vérification

```
uint16_t lookupVFDSegTable(uint8_t index) {
    static const flash uint16_t lookupSeg[SEGMENT_LOOKUP_TABLE_SIZE] = {
        14343,  4100, 12483, 12485,  6340, 10437, 10439, 12292, 14535, 14532,  //0...9
        14534, 14535, 10243, 14343, 10435, 10434, 10375,  6342,   528,   536,  //A...J
         1584,  2051,  7430,  6438, 14343, 14530, 14375, 14562, 10437,  8720,  //K...T
         6151,  4388,  6190,  1320,   784,  8193,                               //U...Z
          103,  2247,   195,  4295,    83,     0,     0,  2130,     0,     0,  //a...j
            0,     0,     0,     0,   103,     0,     0,    66,     0,     0,  //k...t
            7,    36,    46,     0,    37,    73                               //u...z
    };
    if (inRange(LUT_0, LUT_9, index)) {return(lookupSeg[index - LUT_0]);}    //48...57
    if (inRange(LUT_A, LUT_Z, index)) {return(lookupSeg[(index - LUT_A) + LUT_A_OFFSET]);}  //65...90
    if (inRange(LUT_a, LUT_a, index)) {return(lookupSeg[(index - LUT_a) + LUT_a_OFFSET]); }  //97...12
    return (0);
}
```

Listage 3. Interface d'affichage

```
void init_VFD();                              //initialize all data structures
void mux_VFD();                               //displays next digit, timer controlled

void clearVFDDisplay(uint8_t withSymbols);    //clears display
    withSymbols = true                        //deletes all symbols
    withSymbols = false                       //lets symbols untouched

void clearVFDDisplayData(uint8_t index);      //deletes position „index“ in display
                                              //0 = most right digit

void setVFDDisplayData(uint8_t index, uint8_t data);  //shows „data“ on digit „index“

void setVFDDisplayText(uint8_t index, const char *data);  //shows string „*data“,
                                                          //starts @ position „index“

void setVFDDisplayInt(uint8_t index, uint8_t anzStellen, uint8_t wert);
                                    //shows number „wert“ whith „anzStellen“
                                    //digits starting from rigth

void clearRunText();

void setRunText( const char *data );

void tickRunText();
```

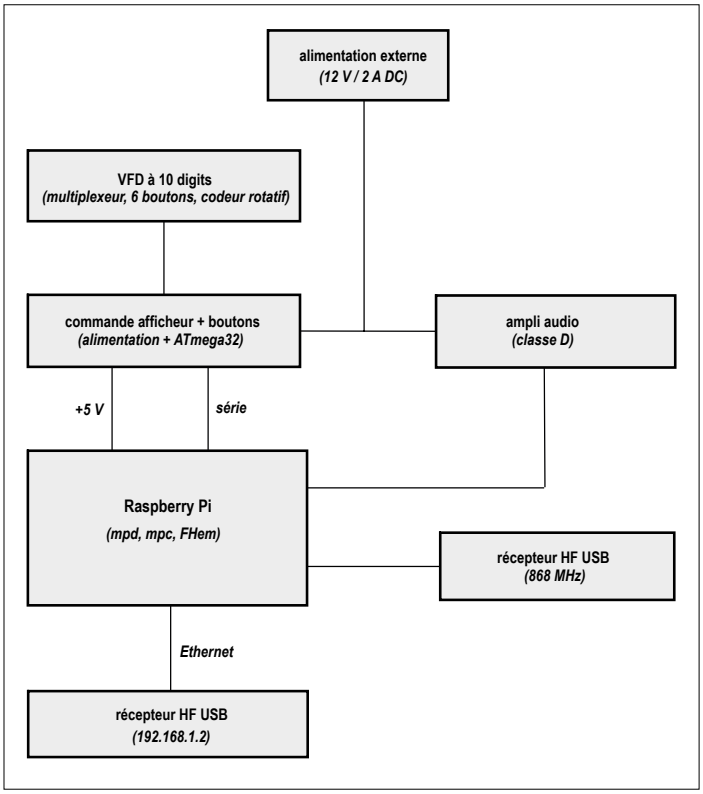


Figure 6. Diagramme fonctionnel de l'ensemble de la radio de cuisine.

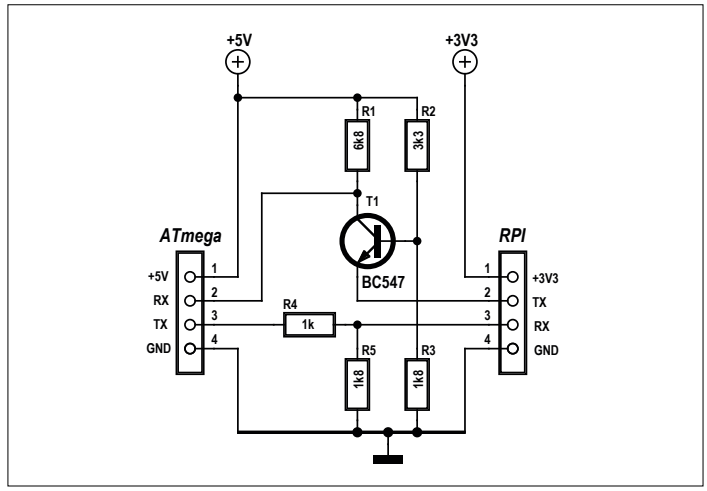


Figure 7. Entre le 5 V de l'ATmega et le 3,3 V du RPi, il faut un convertisseur de niveau sur l'interface sérielle.

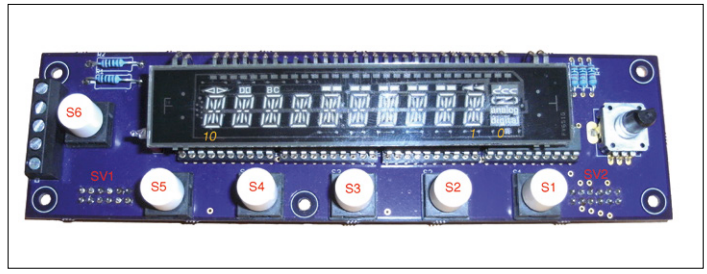


Figure 8. Le circuit imprimé Muxer avec afficheur, pilotes, verrous, boutons et codeur rotatif.

cuisine, avec les boutons pour la manipulation. Il y manque évidemment le principal, la webradio elle-même, fondée sur un Raspberry Pi. C'est ce que nous verrons la prochaine fois.

Diagramme fonctionnel de la webradio

Comme avant-goût, la **figure 6** vous donne déjà un aperçu de cette radio de cuisine branchée. La carte RPi est reliée à l'ATmega par interface sérielle. Mais comme le RPi travaille sous 3,3 V et l'ATmega sur 5 V, il faut changer de niveau (**figure 7**). De l'ATmega vers le RPi, il suffit d'un simple diviseur de tension par R4 et R5. Mais dans l'autre sens, il faut élever le niveau. La solution est simple : le diviseur de tension R2/R3 fixe le potentiel de base de T1 à 1,8 V environ. Son émetteur est sur Tx. Pour un « 1 », T1 bloque, alors son collecteur est amené à 5 V par la résistance de polarisation R1. Pour un « 0 », T1 conduit et son collecteur descend au niveau bas. Comme autre solution, on pourrait alimenter les deux puces en 3,3 V et se passer de convertisseur de niveau. Mais alors, on devrait installer, pour le signal d'horloge de l'ATmega, un quartz à 12 MHz.

Circuits imprimés et construction

Mon premier circuit imprimé s'appelle *Muxer* (**figure 8**), il porte l'afficheur, les verrous, les pilotes, six boutons-poussoirs et un codeur rotatif. Les deux embases à picots SV1 et SV2 sont installées sur la face soudures. Les pilotes et verrous se trouvent sous l'afficheur. Le schéma [4] indique les lignes de port utilisées pour SV1 et SV2.
Le deuxième circuit imprimé (**figure 9**) est celui de l'alimen-

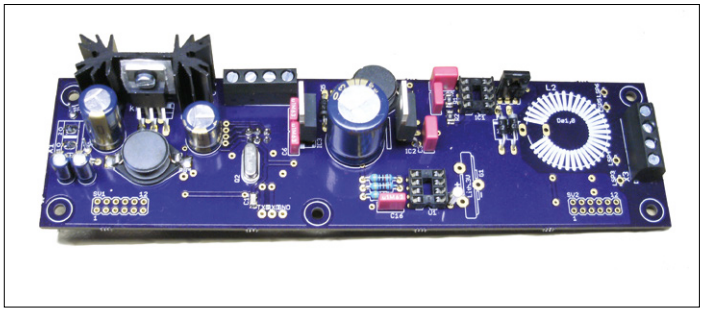


Figure 9. Circuit imprimé partiellement assemblé avec l'alimentation et le contrôleur.

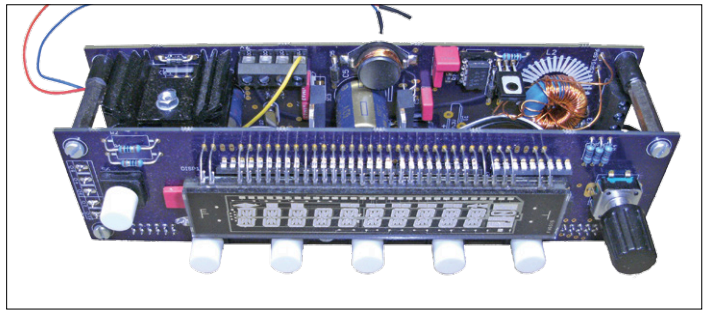


Figure 10. Les deux circuits imprimés du module réunis en sandwich.

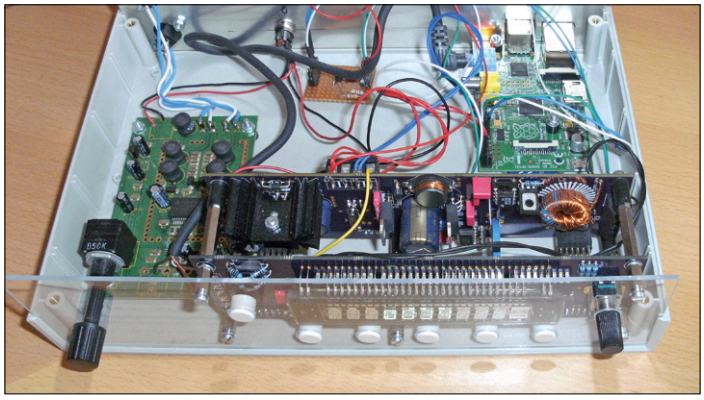


Figure 11. Le prototype terminé dans son boîtier moulé, avec le RPi et les circuits d'appoint, dont le convertisseur de niveau.

tation et du microcontrôleur ATmega32. On y trouve aussi l'horloge en temps réel et sa pile de secours, ainsi que les connexions entre l'interface sérielle de l'ATmega et la carte RPi. Après assemblage et essais, les deux circuits imprimés sont unis en un module compact qui se loge dans un boîtier moulé. Le circuit imprimé à gauche convertit le 12 V de l'alimentation externe en un puissant 5 V qui servira aussi à la carte RPi.

Conclusion

Il n'est pas indispensable que ce module serve d'afficheur à une radio de cuisine moderne, connectée à l'internet, mais vous pouvez le faire. Auquel cas, vous aurez besoin de la deuxième et dernière partie de cet article qui décrira le protocole de communication et le logiciel nécessaire à la webradio basée sur RPi. ⏮

(150720 – version française : Robert Grignard)

Liens

[1] *Technik der Vakuumfluoreszenz-Anzeigen* : http://elektroniktutor.de/technologien/vfd.html

[2] *A guide to VFD operation* : www.noritake-itron.com/SubPages/ApplicNotesE/vfdoperapn.htm

[3] Alimentation du filament : www.kerrywong.com/2013/06/05/vfd-filament-driver-using-555

[4] Téléchargements : www.elektormagazine.fr/150720

L'auteur

Après une formation de technicien en télécommunications et un cursus à l'université technique en électricité et communications, Michael Busser a travaillé une vingtaine d'années au FTZ (Bureau central des télécoms) et ensuite sur différents projets chez T-Systems. Après une formation de troisième cycle pour les enseignants des écoles professionnelles à l'université de Kaiserslautern, il est depuis deux ans professeur d'électrotechnique et d'informatique au BBS TGHS à Bad Kreuznach. Vous pouvez communiquer avec lui à l'adresse : michael.busser@t-online.de.

webradio à tubes fluorescents (2)

RPi + ATmega + logiciel

Michael Busser (Allemagne)

Nous voulons une radio de cuisine à VFD, solide et fiable, mais branchée sur l'internet. Le premier article a traité la fonction, l'électronique et la commande de l'affichage. Mais sans logiciel, nous ne voyons encore rien. L'ATmega a besoin de micrologiciel pour l'affichage, mais il faut aussi du code au Raspberry Pi (RPi) pour en faire une webradio opérationnelle.

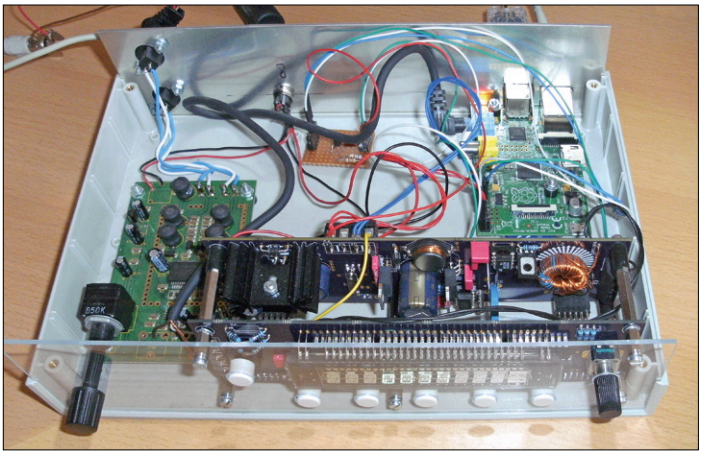


Figure 1. Toutes les cartes sont logées dans un boîtier moulé et y sont câblées.

Figure 2. Couvercle dessus et micrologiciel dedans, la webradio de cuisine à VFD opérationnelle, l'électronique toujours visible derrière la façade en acrylique.

Avec les schémas, les dessins des circuits imprimés et toutes les explications du premier article [1], vous avez construit un beau module VFD compact, tout y est convenablement installé, testé et les tensions sont conformes, pourtant, il ne fait rien d'utile.

Même si toutes les cartes sont câblées (**figure 1**), mises en boîte et le couvercle fermé (**figure 2**), il n'y a toujours rien à voir sur les tubes. C'est que le microcontrôleur de l'unité d'affichage a besoin d'un micrologiciel qui comprenne les commandes. C'est précisément l'objet de ce second article, qui apporte aussi le code pour le RPi.

Protocole de communication

Pour que deux systèmes informatiques puissent échanger des données, il faut s'être mis d'accord au préalable sur la manière de procéder : c'est le rôle du protocole. J'en ai faufilé un moi-même qui utilise un bus de domotique à deux fils. Il dispose de caractéristiques qui ne sont pas forcément nécessaires ici, mais il repose sur la conception habituelle de ces protocoles. Le micrologiciel pour l'ATmega du module d'affichage a été rédigé en entier dans l'environnement de développement Studio d'Atmel (cf. l'écran de la **figure 3**).
Le **tableau 1** montre la structure des données. La couche de transport prend la forme de l'envoi d'un message (type

Tableau 1. Trame du protocole

STX *début de transmission*	contenu du message	ETX *fin de transmission*
0x02	data	0x03

Tableau 2. Éléments du message

STX	SRC	DST	MT	MODE	charge utile (0 à 15 octets)	CS	ETX

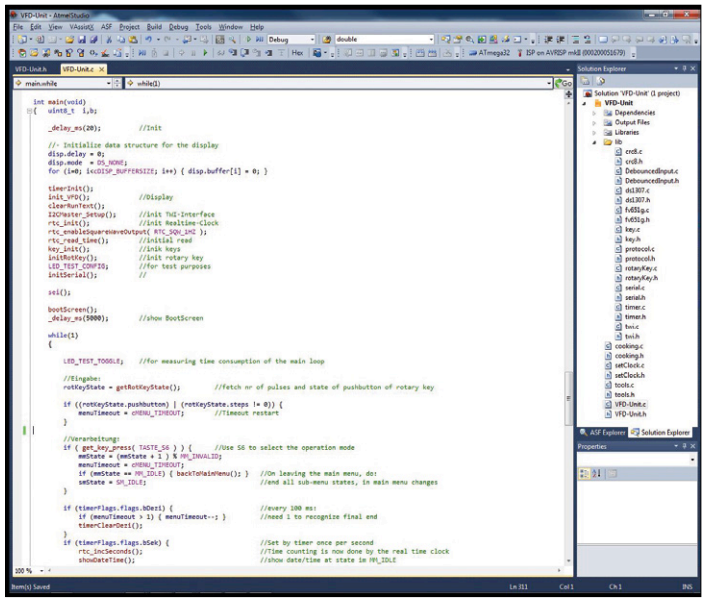


Figure 3. Fenêtre de l'environnement de développement Studio d'Atmel avec le micrologiciel pour l'ATmega du module d'affichage.

TDataBuf), avec comme préambule un STX. On marque la fin d'une transmission d'un ETX. On ne peut donc pas utiliser les signaux STX et ETX dans le reste du flux de données. Si vous le faites, masquez-les en les faisant précéder du caractère DLE (= 01xF). Le récepteur élimine alors le signe de camouflage des données et poursuit avec les caractères suivants du message reçu. Si c'est le caractère de masquage que vous devez transmettre, il faut envoyer deux signes DLE.

Tableau 3. Codes du message

champ	signification
SRC	Source *(Quelladresse)* : 0 à 255 sauf valeurs 2 et 3
DST	Destination *(Zieladresse)* : 0 à 255 sauf valeurs 2 et 3
MT	Type du message : 0x10 StatusReq 0x11 StatusResp 0x12 DisplayReq 0x13 DisplayResp 0x14 Key_Req 0x15 Key_Resp 0x16 Time_Req 0x17 Time_Resp Note : 0x02 et 0x03 sont interdits. Reste libre pour des extensions.
MODE	Bit signification 0 à 3 : longueur de la charge utile en octets : 0 à 15 4 : 0/1 = avec / sans somme de contrôle 5 : 0/1 = confirmation par ACK nécessaire / ou pas 6 : 0 = réservé 7 : toujours 1 Note : 0x02 et 0x03 sont interdits. De là, bit 7 toujours 1.
CS	Somme de contrôle du message

L'interface est formée des fonctions suivantes (cf. serial.c et serial.h) :

```
void initSerial(void);
void sendMsg(TDataBuf msg);
uint8_t peekMsg(TDataBuf *msg);
```

La fonction initSerial() doit être exécutée une fois au lancement du programme. Elle assure l'initialisation de l'UART et déverrouille différentes interruptions. On peut fixer le débit binaire en plaçant dans serial.h la mention :

```
#define UART_BAUD_RATE 9600UL
```

Les autres paramètres de l'interface sont 8N1. La fonction sendMsg() s'occupe des messages de type TDataBuf, les inscrit dans le tampon interne d'émission (quand il est libre) et retourne tout de suite à l'appelant. La réception et l'émission des messages sont commandées par interruption et se passent en arrière-plan. Dès réception d'un message valide, il est placé dans l'un des deux tampons de réception. Un appel de la fonction peekMsg() renvoie un 1 et le message valide est disponible dans msg.
Le mécanisme de masquage est déjà à l'ouvrage ici. Il n'y a qu'un tampon d'émission, mais deux de réception, de manière à ce qu'un message puisse être traité pendant la réception du suivant.
Le module de code est indépendant de la forme du message à transmettre qui est défini dans protocol.c et protocol.h. Un bus à deux fils comme le RS485 nécessite encore un signal de commande pour commuter entre l'émission et la réception. L'état normal est la réception. L'émetteur n'est actif que pendant la durée de la transmission. Il faut dans serial.h les deux définitions suivantes :

```
#define SENDER_OFF
#define SENDER_ON
```

Comme le récepteur est toujours en service, il reçoit aussi le message envoyé par le poste lui-même. C'est utile pour détecter les collisions de bus. Dans protocol.h, on n'utilise que la fonction ownAdr() qui renvoie l'adresse propre du nœud sur le bus.

Le **tableau 2** détaille les différents éléments d'un message et le **tableau 3** contient leurs significations. On voit dans le **listage 1** l'agencement d'un message de type *struct* en C. Avec le regroupement de type union, la structure TDatagramm et la chaîne d'octets TBuf occupent la même adresse en mémoire. On peut ainsi considérer un message simplement comme une chaîne d'octets, ce qui simplifie l'émission et la réception.

Communication ATmega

Voici les différents types de messages :

```
#define MT_STATUS_REQ 0x10
#define MT_STATUS_RES MT_STATUS_REQ + 1

#define MT_DISPLAY_REQ 0x12
```

```
#define MT_DISPLAY_RESPMT_DISPLAY_
      REQ + 1

#define MT_KEY_REQ  0x14
#define MT_KEY_RESP MT_KEY_REQ + 1

#define MT_TIME_REQ 0x16
#define MT_TIME_RESPMT_TIME_REQ + 1
```

Avec `DISPLAY_REQ`, le RPi peut enfin afficher quelque chose sur le VFD. D'autre part, `KEY_REQ` informe le RPi qu'un bouton a été actionné sur le module d'affichage. Avec `TIME_REQ`, l'ATmega peut se procurer le temps réel par le RPi, lequel le reçoit par l'internet via NTP.
L'ATmega peut envoyer et recevoir des messages dans un format défini par l'interface sérielle d'un simple appel de la fonction ad hoc dans `protocol.c` (**listage 2**). La réception a lieu dans une boucle du programme principal (**listage 3**), qui vérifie constamment la validité des messages reçus et les traite, si nécessaire.

Voici un exemple de déroulement d'un message.
Toutes les 60 min, l'ATmega envoie au RPi un message du type `MT_TIME_REQ`, ce qui provoque l'appel de la fonction `sendMsgTimeReq()` :

```
if (timerFlags.flags.bMin) {
    doNTP_Sync--;
    if (doNTP_Sync == 0) {
        doNTP_Sync = NTP_SYNC;
        sendMsgTimeReq();
      // appel temps réel de RPi
    }
    timerClearMin();
}
```

Le RPi envoie en retour un message du type `MT_TIME_RESP` qui sera traité dans la boucle de réception du programme principal. Le texte source complet est trop long pour une reproduction ici. Vous pouvez l'obtenir dans le fichier *Software_2.zip* du téléchargement [1].

Logiciel RPi

On utilise comme système d'exploitation pour le RPi la version *wheezy-raspbian* du 09/09/2014. Comment en faire une image à copier sur carte SD avec la configuration de base, vous l'apprendrez en détail sur [2].
Il y a en [3] une notice explicative sur la manière de configurer RPi en webradio

Listage 1. Construction du message

```
typedef struct {       // construction d'un message
  uint8_t     src;     // adresse source (émetteur)
  uint8_t     dst;     // adresse de destination (récepteur)
  uint8_t     mt;      // type du message
  TMode       mode;    // octet de mode, décrit la structure du message
  TPayload    Payload; // dépend du niveau de l'application
  uint8_t     cs;      // somme de contrôle, par ex. CRC-8
} TDatagramm;

typedef union {
    TBuf        Buf;          // accès vi table d'octets
    TDatagramm  Datagramm;    // accès via structure
} TDataBuf;
```

Listage 2. Fonctions des messages

```
void sendMsgTimeReq(void) {
  TDataBuf msg;

  msgClear( &msg );
  msg.Datagramm.src = ownAdr();
  msg.Datagramm.dst = masterAdr();
  msg.Datagramm.mt  = MT_TIME_REQ;
  msg.Datagramm.mode.bAckReq = NO_ACK_REQUIRED;
  msg.Datagramm.mode.bCRC = NO_CHECKSUM; // WITH_CHECKSUM;
  msg.Datagramm.mode.count = 0; // ne pas compter octets de tête / fin
  msgPrepare( &msg );
  sendMsg( msg );
}
```

Listage 3. Analyse des messages

```
if (peekMsg(&recMsg) == 1) {// si message reçu : analyser
    switch (recMsg.Datagramm.mt) {
        case MT_STATUS_REQ:     break;
        case MT_STATUS_RESP:    break;
        case MT_DISPLAY_REQ: i=0;
            b=1;
            for (i=0; i < cDISP_BUFFERSIZE; i++) {
                if (recMsg.Datagramm.Payload[i] == 0) {b = 0;}
                if (b == 1) {
                    disp.buffer[i] = recMsg.Datagramm.Payload[i];
                } else {
                    disp.buffer[i] = 0;
                }
            }
            disp.delay           = cDISP_SHOWTEXT;
            disp.mode            = DS_TEXT;
            showText();
            break;
        case MT_DISPLAY_RESP:   break;
        case MT_KEY_REQ:        break;
        case MT_KEY_RESP:       break;
        case MT_TIME_REQ:       break;
        case MT_TIME_RESP:      setTimeOnMsgReq( &recMsg );    break;
        }
}
```

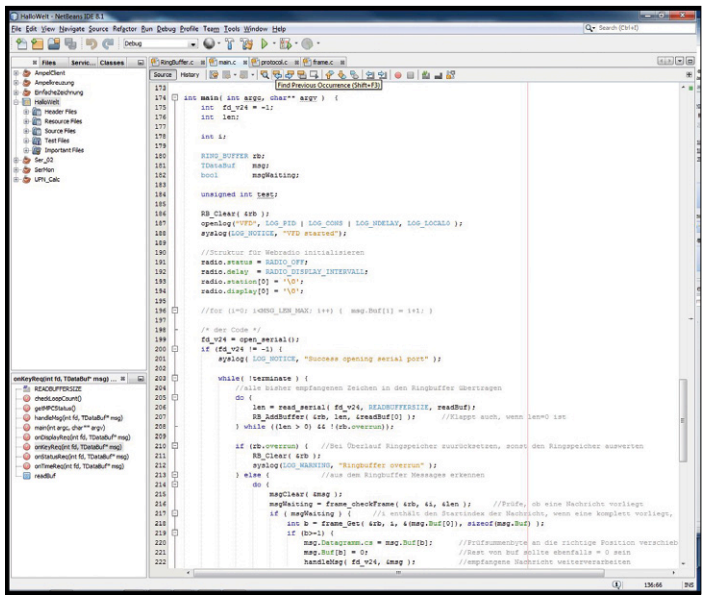


Figure 4. Fenêtre de l'EDI de NetBeans avec le micrologiciel qui transforme un Raspberry Pi en une webradio.

à l'aide de *mpd* (*Music Player Daemon*) et *mpc* (*Music Player Client*). Mais bien d'autres sources existent sur l'internet à ce sujet. Il suffit d'une recherche sur « RPI mpd » pour s'en convaincre.
Pour faire tourner un premier programme RPi en C, il vous faut un environnement de développement (en principe, la ligne de commande suffirait). J'ai choisi NetBeans parce que Java était déjà installé sur mon ordinateur. La **figure 4** vous montre de quoi il s'agit. L'installation de NetBeans comme EDI pour RPi est décrite pas à pas sur [4].

Une fois que vous en êtes arrivé là, vous pouvez confortablement créer sous Windows des programmes en C pour RPi. La compilation et le débogage se feront sur RPi.
Par bonheur, le code C pour l'ATmega tourne aussi sans trop de modifications sur RPi, pour autant qu'on n'ait pas besoin de fonction spéciale ou de module supplémentaire, il n'y en a pas là.

Le programme RPi envoie des messages de journalisation à un serveur Syslog, ce qui aide au débogage. Le Syslog doit être configuré sur le Rpi. La fonction `dumpMsg` réceptionne un message et l'écrit sous forme lisible sur le Syslog.

Dans la partie `main` du programme RPi, tous les caractères qui viennent d'arriver sont lus sur le port sériel et déposés dans un tampon circulaire :

```
do {
    len = read_serial(fd_v24, READBUFFERSIZE,
      readBuf);
    RB_AddBuffer(&rb, len, &readBuf[0]);
                // fonctionne même si len=0
} while ((len > 0) && !(rb.overrun));
```

La méthode `frame_checkFrame()` est appelée régulièrement pour vérifier si un message complet se trouve déjà dans le tampon. Sinon, elle retourne un 0 à la place d'un 1. Les paramètres `startindex` et `length` indiquent le début et la longueur du message.
Un appel de la fonction `frame_get()` renvoie alors le message en question dans `msg`. Il est ensuite traité par la fonction `handleMsg()` (**listage 4**). Le message d'exemple `MT_TIME_RESP` arrive ainsi à la fonction `handleMsg()` qui enchaîne un appel à `onTimeReq()` (**listage 5**). C'est là qu'on fait la réponse à la requête et on la renvoie directement à l'expéditeur par la fonction `frame_Send()`.
Le RPi s'occupe de la configuration du client NTP. La fonction `localtime()` à l'intérieur de la fonction `getTimeResp()` effectue la demande du temps système. On peut voir comment le message de réponse est composé dans la structure `TDataBuf`.

Listage 4. Fonction *handleMsg()*

```
void handleMsg(int fd, TDataBuf *msg) {
    if (msg->Datagramm.mode.bCRC) {// test de la somme de contrôle
        syslog( LOG_NOTICE, "checksum ignored");  // tbd
    }
    if (msg->Datagramm.dst == masterAdr()) {
        switch (msg->Datagramm.mt) {
        case MT_STATUS_REQ:     onStatusReq(fd, msg);     break;
        case MT_DISPLAY_REQ:    onDisplayReq(fd, msg);    break;
        case MT_KEY_REQ:        onKeyReq(fd, msg);        break;
        case MT_TIME_REQ:       onTimeReq(fd, msg);       break;
        default:    syslog(LOG_ERR, "unknown message dropped");
            dumpMsg(msg);
        }
    } else {// ce n'est pas mon message
        syslog( LOG_NOTICE, "message not for me, ignoring it");
    }
```

Listage 5. Fonction *onTimeReq()*

```
void onTimeReq(int fd, TDataBuf *msg) {
    TDataBuf reply;
    msgClear(&reply);
    reply.Datagramm.src = masterAdr();
    reply.Datagramm.dst = msg->Datagramm.src;
    getTimeResp(&reply);
    // dumpMsg(&reply);
    frame_Send(fd, &(reply.Buf[0]), reply.Datagramm.mode.count
      + MSG_HEADER_LEN, reply.Datagramm.cs );
    syslog( LOG_NOTICE, "onTimeReq");
}
```

Webradio

On réalise la webradio proprement dite avec le *mpc* de Music Player Client. C'est là que l'on crée des listes de lecture des stations de radio par internet. Vous trouverez les détails sur [5]. Il y a une aide en ligne *mpc* disponible par invite de commande. Mon programme, écrit en C, commande le mpc en l'appelant par l'option correspondante de `system()` comme on pourrait le faire avec l'invite de commande.

Un fragment de la fonction `onKeyReq()` dans le **listage 6** vous éclairera davantage. On l'appelle en appuyant sur un bouton du module d'affichage.

Si vous appelez mpc sans paramètre, vous verrez dans la première ligne la station et le titre qui passe en ce moment. La fonction `getMPCStatus()` le fait à intervalles réguliers et envoie ensuite le nom de la station par `MT_DISPLAY_REQ` à l'ATmega qui affiche alors le nom de la station de radio choisie à la place de la date du jour.

Avec mpd, les flux radio doivent toujours être enregistrés dans une liste de lecture. On la trouve dans le répertoire */var/lib/mpd/playlists*.

Listage 6. Fragment de la fonction *onKeyReq()*

```
void onKeyReq(int fd, TDataBuf *msg) {
    uint8_t keys = msg->Datagramm.Payload[0];
      // définition variable / vérification bouton
    if (keys & KEY_S0) {syslog( LOG_NOTICE, "onKeyReq:  S0");}
    if (keys & KEY_S1) {terminate = true; syslog( LOG_NOTICE, "onKeyReq: S1");}
    if (keys & KEY_S2) {syslog( LOG_NOTICE, "onKeyReq: S2");}
    if (keys & KEY_S3) {syslog( LOG_NOTICE, "onKeyReq: S3");}
    if (keys & KEY_S4) { // station suivante
        if (radio.status == RADIO_ON) {
            system("mpc next");}
            syslog( LOG_NOTICE, "onKeyReq: S4");
        }
    if (keys & KEY_S5) {                    // on / off radio
        if (radio.status == RADIO_OFF) {    // si radio off
            system("mpc play");             // jouer dernière station
            radio.status = RADIO_ON;
        } else                              // radio off
            system("mpc stop");
            radio.status = RADIO_OFF;
        }
        syslog(LOG_NOTICE, "onKeyReq:  S5");
    }
}
```

Liens

[1] Logiciel et 1ère partie : www.elektormagazine.fr/150720

[2] Installation du RPi : www.netzmafia.de/skripten/hardware/RasPi/RasPi_Install.html

[3] RPi en Webradio : www.youtube.com/watch?v=jf3M1RVpQbM

[4] *Netbeans* pour RPi : http://bit.ly/2aBZ14A

[5] Tutoriel webradio : www.youtube.com/watch?v=jf3M1RVpQbM

[6] Lancement automatique : www.netzmafia.de/skripten/hardware/RasPi/RasPi_Auto.html

Lancement du programme

Actuellement, il faut encore lancer le programme du RPi manuellement parce que son développement n'est pas terminé. La longueur des répertoires créés lors de l'utilisation de NetBeans est assez embarrassante. Si vous suivez la notice pour configurer NetBeans (dans votre répertoire racine), le programme exécutable se retrouve dans le répertoire suivant :

```
/root/.netbeans/remote/192.168.1.24/mib2-Windows-
      x86_64/D/Projekte/RaspberryPi/RPi/HalloWelt/
      dist/Debug/GNU-Linux
```

La partie du chemin en caractères gras change en fonction de l'environnement : adresse IP du PC sous Windows, « mib2 » (nom d'hôte du PC) et « D\Projekte\... » (chemin du projet sous Windows).

Mais on peut facilement faire en sorte que le RPi lance automatiquement le programme. Comment ? C'est Netzmafia [6] qui l'explique. Lors d'un démarrage à partir du shell, le programme adhère à ce shell. Quand on l'arrête, le programme s'arrête aussi. Le remède consiste en un simple « & » après la commande :

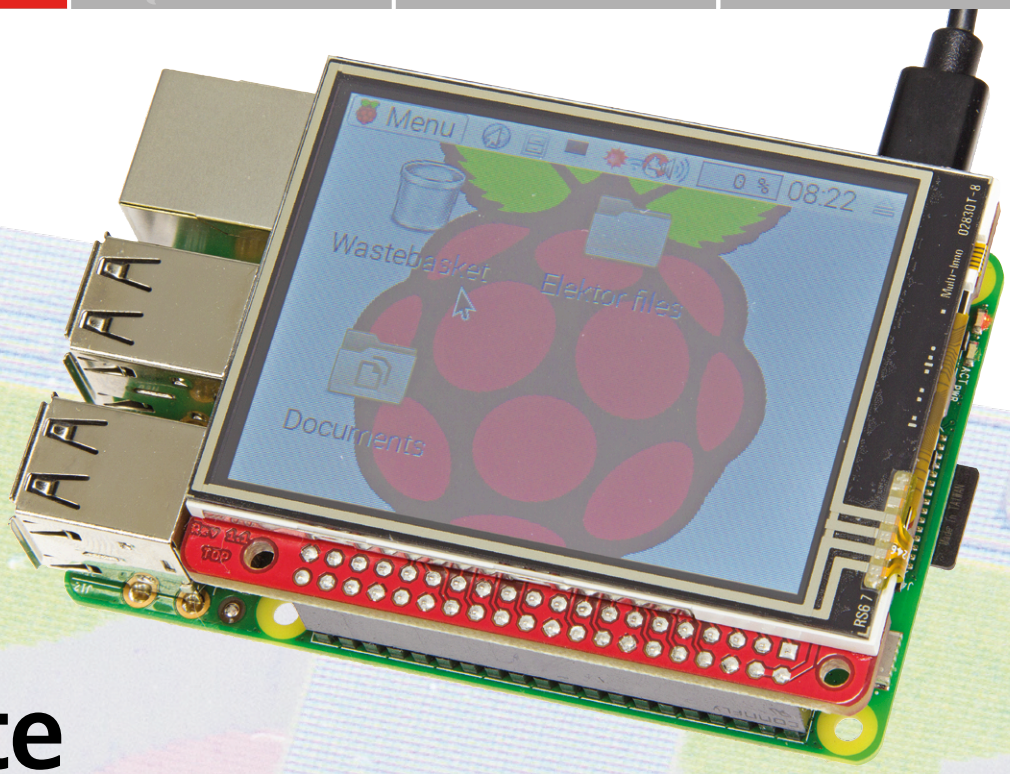
./hallowelt &

Le programme est alors exécuté en tâche de fond. Le shell donne aussi un numéro de processus et un PID. Quand on veut le ramener à l'avant-plan, on utilise alors :

fg %1

Le programme est alors capable de tourner sur le RPi. Des développements complémentaires sont possibles et bienvenus. On peut importer directement dans NetBeans le programme emballé dans *HalloWelt.zip*. Bonne chance pour vos expériences !

(160207 – version française : Robert Grignard)



raccorder un écran tactile à une carte Raspberry Pi

facile ou difficile ?

Il existe de nombreux modules à enficher pour la carte Raspberry Pi (appelés *HAT – Hardware Attached on Top*), pour diverses applications. Certains fonctionnent dès qu'ils sont connectés, d'autres doivent être programmés ou configurés. Ce n'est pas toujours très clair...

Luc Lemmens (labo d'Elektor)

Lorsqu'il s'agit d'un écran, il faut connaitre la procédure pour faire apparaître une image, au moins pour savoir s'il fonctionne ou pas. Elektor propose depuis quelque temps dans son e-choppe un écran tactile pour la carte Raspberry Pi [1]. Il s'agit d'un écran couleur de 320 x 240 pixels avec une diagonale de 2,8 pouces (7,11 cm), doté d'un revêtement tactile résistif ; en outre, il n'est pas très cher. Ses dimensions sont quasi identiques à celles de la carte Raspberry Pi, l'ensemble est donc très compact et facile à loger dans un boîtier.

Plusieurs lecteurs nous ont demandé quel logiciel installer pour cet écran ; en effet, il semblerait que les distributions Linux standard de la fondation Raspberry Pi ne le prennent pas en charge.

Le fabricant du module, la société allemande Watterott, a créé une page sur *GitHub* [2], où sont disponibles toutes les informations sur le produit et le logiciel. Sur cette page, vous trouverez des images toutes prêtes de différentes distributions (Raspbian/Debian) pour carte SD ; ces images contiennent les outils nécessaires pour piloter l'écran. Vous trouverez sur l'internet différentes pages qui expliquent comment transférer l'image d'une distribution Linux sur une carte SD.

Si vous ne voulez pas sacrifier une installation existante, téléchargez de cette page un *script* qui installera la mémoire d'image de cet écran à transistors en couches minces *(FBTFT – Framebuffer for thin-film transistor screen)*. Nous avons testé ce *script* au labo d'Elektor, il fonctionne parfaitement. Si vous préférez l'installation manuelle (par exemple pour une autre distribution Linux), ou si le programme ne fonctionne pas, cette page propose aussi des explications sur l'installation des pilotes *(drivers)*. N'oubliez pas que dans tous les cas vous devrez calibrer l'écran.

Cerise sur le gâteau de ce site : quelques projets autour de la combinaison écran-Raspberry Pi, entre autres une *webradio* et une mini-TV. Ça vaut assurément le détour ! ⏮

(150824 – version française : Jean-Louis Mehren)

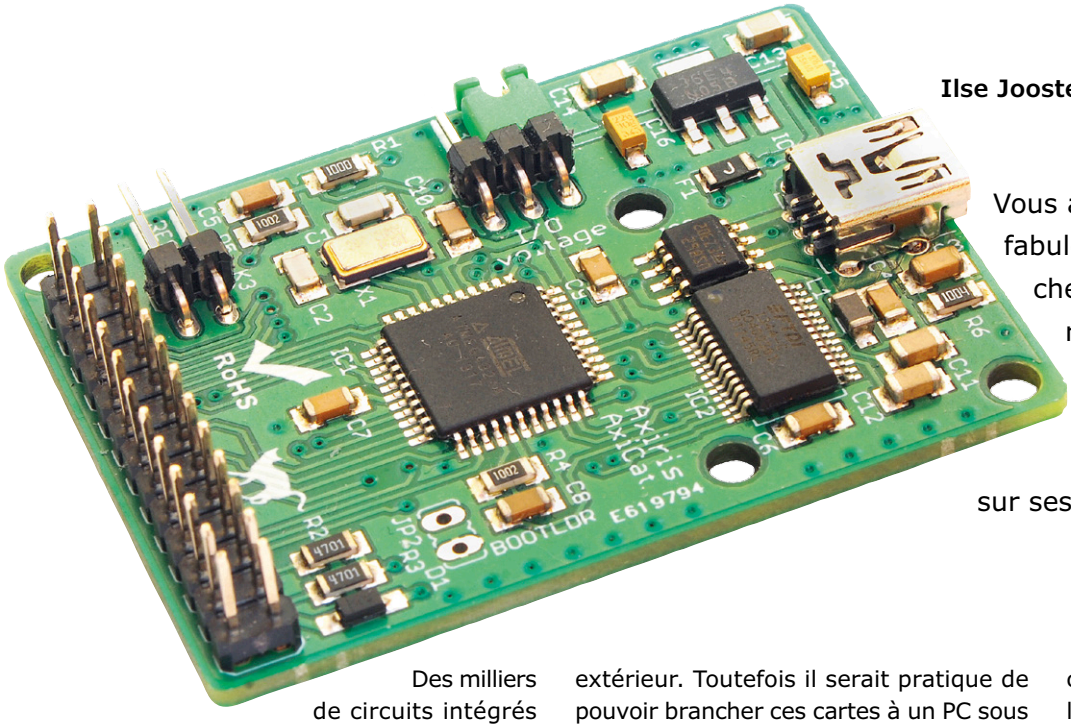
Liens

[1] www.elektor.fr/touch-display-for-raspberry-pi

[2] github.com/watterott/RPi-Display

AxiCat

outil de développement polyvalent sur USB



Ilse Joostens & **Peter S'heeren** (Belgique)

Vous avez écrit du code pour un fabuleux circuit intégré et vous cherchez comment connecter, rapidement et facilement, votre montage à un PC sous Linux ou Windows. Ne cherchez pas plus : AxiCat retombe toujours sur ses pattes !

Des milliers de circuits intégrés intéressants s'offrent à vous, développeur professionnel ou amateur enthousiaste. Plusieurs d'entre eux sont des CMS, ce qui est malcommode, mais par bonheur, de nombreux distributeurs plus ou moins connus et qui pensent aux constructeurs anonymes, proposent une large gamme de BoB (*BreakOut Board*), des cartes de liaison compactes. La majorité d'entre elles disposent d'une interface I²C ou SPI pour communiquer avec le monde extérieur. Toutefois il serait pratique de pouvoir brancher ces cartes à un PC sous Linux ou Windows pour tester rapidement leurs fonctions, avant de les inclure dans un projet. C'est possible avec l'adaptateur multiprotocoles AxiCat avec interface USB, sans même devoir écrire une seule ligne de code.

Schéma et construction

On le voit à la **figure 1**, le cœur du dispositif, c'est IC1, un microcontrôleur ATmega164A d'Atmel. D'autres membres de cette famille conviennent aussi, comme les 324A, 644A et 1284, la seule différence réside dans la capacité de la mémoire interne. Normalement, l'AxiCat est livré avec le 164A ou le 624A, selon leur disponibilité. La tension d'alimentation est fixée par le cavalier JP1, à 5 V ou 3,3 V. Le choix s'est porté sur une horloge à quartz de 12 MHz plutôt qu'une fréquence plus élevée, ainsi le fonctionnement du microcontrôleur sous 3,3 V reste fiable.

La liaison USB passe par IC2, un FT245RL de FTDI. Contrairement au plus célèbre FT232, il n'a pas d'interface sérielle, mais d'un registre FIFO interne avec bus de données sur 8 bits de large. Son avantage est de travailler à pleine vitesse en USB à 12 MHz pour tenir un débit respectable entre le bus USB et le microcontrôleur IC1. La couche sous-jacente du protocole de communication USB entre le pilote sur le système hôte et le FT245RL assure aussi un transfert exempt d'erreur vers le FIFO du FT245RL. Cela résulte du fait que les données sont lues sans erreur par le microcontrôleur IC1 et arrivent toujours sans erreur par IC1 au système hôte, ce qui rend superflue une procédure de correction d'erreur dans le protocole de communication. Tout bénéfice pour la vitesse. IC3, un USB6B1, protège les lignes USB

Caractéristiques techniques

- basé sur un puissant microcontrôleur ATmega164A à 12 MHz
- convient aussi bien à Windows qu'à Linux
- niveaux logiques : 3,3 V ou 5 V
- 17 lignes d'E/S bidirectionnelles avec résistance de polarisation configurable
- bus I²C avec résistances de polarisation, fonction maître ou esclave, vitesse maximale de 400 kHz
- lignes de sélection SPI maître avec quatre esclaves, vitesse maximale de 6 MHz
- deux ports sériels indépendants
- maître *1-wire* avec possibilité de puissante polarisation et accélérateurs intégrés (énumération)
- alimenté par bus USB
- en mode asynchrone, toutes les interfaces sont utilisables simultanément
- peut servir de programmateur *in situ* pour microcontrôleurs AVR
- chargeur d'amorçage pour faciliter la mise à niveau du micrologiciel et charger son propre logiciel
- brochage du connecteur GPIO compatible avec Raspberry Pi A et B.

contre les DES, sécurité supplémentaire pour la carte.

La tension de 3,3 V vient d'IC4, un régulateur LM1117-3,3 à faible déchet. Elle est à disposition sur le connecteur GPIO K2 et peut s'utiliser à votre convenance avec le 5 V de l'USB pour alimenter le microcontrôleur et la section d'E/S de IC2. Le fusible réarmable (polyfuse) F1 limite à 350 mA le débit total pour éviter une surcharge du port USB. Autant dire que le courant maximal sur le connecteur GPIO doit rester inférieur à 350 mA, lignes d'E/S, alim 5 V et 3,3 V ensemble. La diode D1 dans la ligne d'alimentation 5 V vers les broches 2 et 4 du connecteur GPIO K2 empêche que les cartes d'extension alimentées par un Raspberry Pi puissent injecter du courant en direction de l'AxiCat et du port USB de l'ordinateur. Le connecteur GPIO K2, une embase mâle à 2 × 13 picots, n'est pas choisi au hasard, de même que la disposition des contacts (cf. **figure 2**), mais ils correspondent *grosso modo* au connecteur GPIO sur les premières versions du RPi (modèles A et B). L'avantage majeur de cette disposition est de pouvoir raccorder sur AxiCat un grand nombre de cartes d'extension et de HAT pour RPi. Les cartes d'extension pour RPi qui utilisent d'autres lignes d'E/S sur le connecteur à 40 contacts des nouvelles versions de

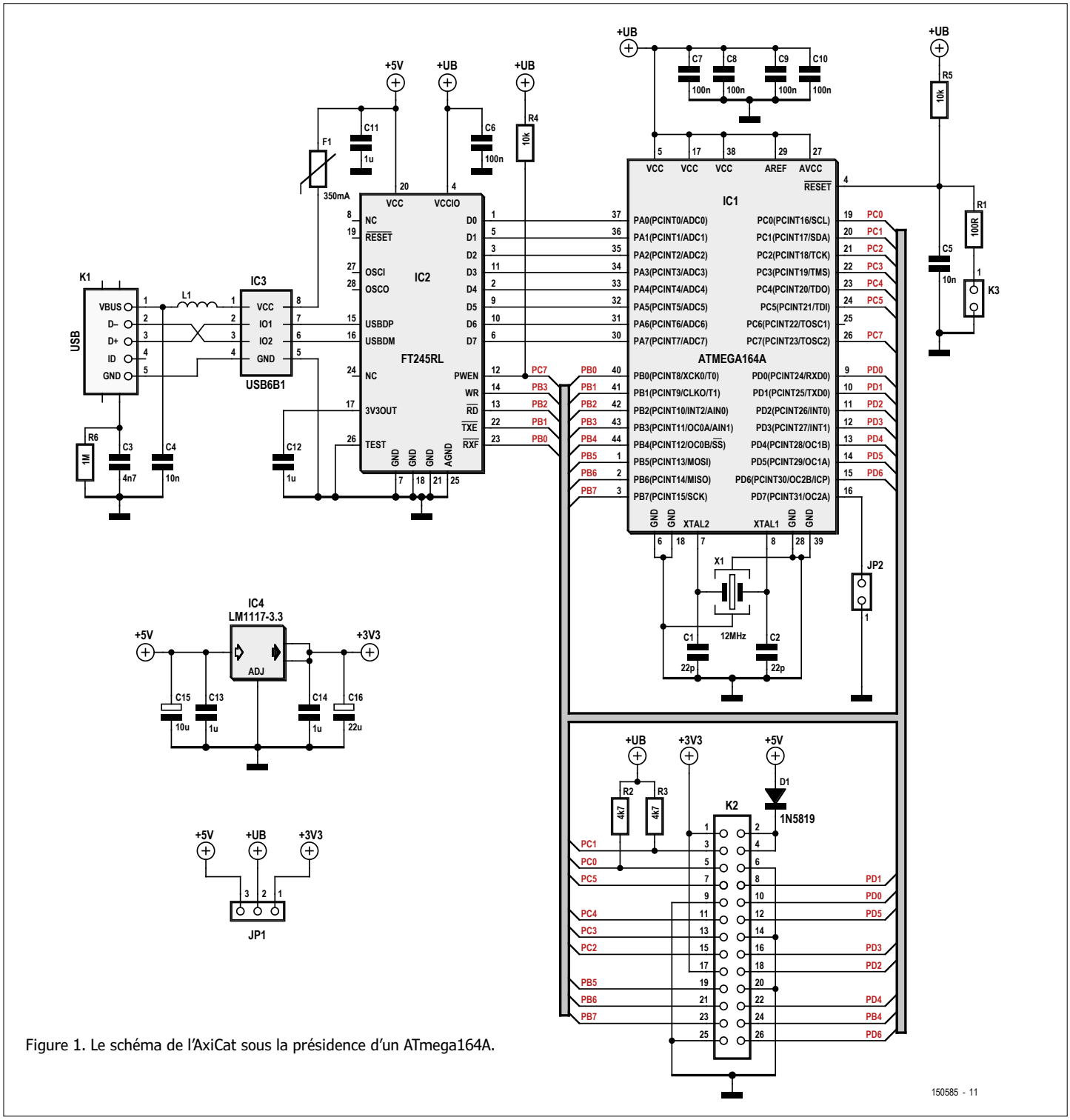


Figure 1. Le schéma de l'AxiCat sous la présidence d'un ATmega164A.

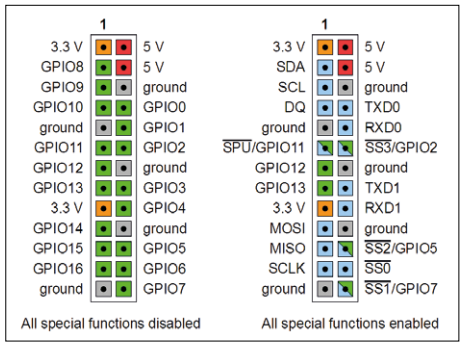


Figure 2. Le brochage du connecteur le rend compatible avec les Raspberry Pi modèle A et B. Vous pouvez ainsi connecter directement sur l'AxiCat un grand nombre de HAT Raspberry Pi.

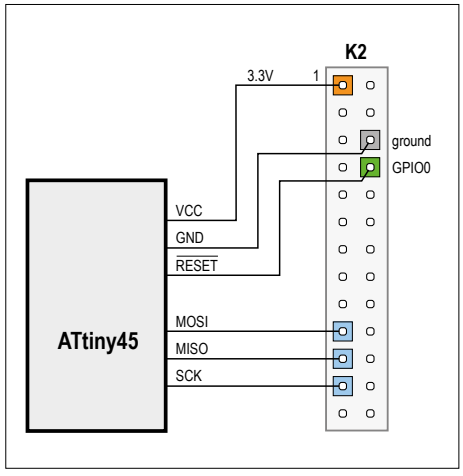


Figure 3. La manière de transformer un AxiCat en programmateur et de le relier à un ATtiny45.

RPi (modèles A+, B+, 2 et 3) ne sont pas utilisables.
Enfin, le connecteur cavalier K3 permet une RàZ manuelle d'AxiCat. Si le cavalier JP2 est mis lui aussi, le chargeur d'amorçage interne est activé. C'est ce qui permet d'envoyer à l'AxiCat son propre micrologiciel.

Protocole sériel

Le protocole sériel définit le flux de données entre l'hôte et l'AxiCat, à savoir des paquets d'octets transmis en duplex. Les paquets envoyés par l'hôte sont des commandes, ceux de l'AxiCat sont des réponses. Le fichier `axicat.h` disponible sur [1] contient les définitions en langage C.
On discerne des commandes synchrones et des asynchrones. La différence réside dans la manière dont l'AxiCat les exécute après réception. Une commande synchrone s'exécute immédiatement. Si une réponse est définie pour cette commande, l'AxiCat répond tout de suite à l'hôte. Les commandes synchrones concernent entre autres la lecture d'information d'état, la modification de réglages, l'accès aux lignes GPIO.
Certaines commandes demandent plus de temps d'exécution. Ce sont celles qui comportent une transaction de bus, comme un transfert SPI ou l'énumération en *1-wire*. Pour éviter un blocage du flux entrant de commandes, l'AxiCat exécute donc en arrière-plan une commande asynchrone longue, et ce n'est que quand elle a été exécutée que l'AxiCat envoie à l'hôte la réponse.
Les commandes asynchrones s'exécutent dans un tampon de travail (*work buffer*). Il y en a pour les composants suivants : I²C maître, I²C esclave Tx, I²C esclave Rx, SPI maître, 1-Wire maître, UART n°0 Tx, UART n°0 Rx, UART n°1 Tx et UART n°1 Rx. Notez que les tampons d'exécution de l'UART ne stockent pas de commandes, mais les octets de données à transmettre (Tx) ou reçus (Rx).
Les tampons de travail permettent à l'AxiCat d'exécuter des transactions de bus sur différentes interfaces en parallèle. La grandeur réelle de chaque tampon est réglable dans le micrologiciel et dépend principalement de la quantité de RAM disponible dans le microcontrôleur.
Il y a aussi des réponses automatiques (*unsolicited responses*). On ne dispose pas de commande par exemple pour lire des octets de données arrivés par l'UART. Alors AxiCat envoie lui-même un paquet de réponses quand un UART reçoit des octets.

Micrologiciel

`AxiCat Main` est le programme principal qui fait vivre l'AxiCat. Il est écrit en C. C'est là que se trouve l'implémentation du protocole sériel. Nous utilisons WinAVR-20100110. WinAVR est une version Windows de gcc pour AVR à 8 bits et AVR Libc.
Au cours du développement d'AxiCat, nous avons souvent observé le code assembleur produit et régulièrement réécrit le code source pour obtenir un code machine plus efficace et plus compact. En effet c'était nécessaire d'une part pour atteindre une rapide itération de la boucle principale, cruciale pour les performances d'AxiCat, et d'autre part pour faire entrer le programme principal dans les 16 Ko de mémoire flash de l'ATmega164A.
Nous avons décidé d'utiliser le moins possible des interruptions, parce que le compilateur gcc déclenche souvent des routines d'interruption qui stockent puis récupèrent de nombreux registres sur la pile, ce qui fait perdre beaucoup de temps. Nous préférons utiliser des tâches qui ne paralysent rien, elles font partie de la boucle principale. `TIMER1` est la seule fonction matérielle à routine d'interruption et sert à compter les millisecondes.
On choisit la taille des tampons de travail en fonction de la RAM disponible et du code que crée le compilateur. Le programme principal utilise des tampons de 2^n octets avec un maximum de 128 octets pour garder un code compact et efficace avec un AVR à 8 bits.
Le programme principal autorise le réveil à distance par USB. AxiCat peut sortir l'hôte de l'état de veille. L'avantage vient de ce que le système d'exploitation de l'hôte permet le réveil à distance.
Le chargeur d'amorçage `Axicat Bootloader` s'exécute au démarrage d'AxiCat. Il vérifie d'abord l'état du cavalier JP2. S'il est absent, le chargeur saute vers le programme principal. S'il est là, le chargeur reste actif, on a la possibilité avec lui (`aximcubldr`) de charger un nouveau programme principal de l'hôte vers l'AxiCat.

Logiciel

`AxiCat Server` est un programme qui met à disposition les fonctions d'AxiCat par des ports réseau et des E/S ordinaires, sous la forme d'un protocole de communication. Le serveur accepte des connexions client multiples, de sorte que plusieurs intervenants peuvent travailler en même temps avec AxiCat. Le serveur opère comme un multiplexeur entre plusieurs clients et un seul AxiCat, au profit d'une grande souplesse d'utilisation.
Le protocole de communication repose sur des commandes et les réponses correspondantes. Un client lance une commande, ensuite le serveur lui répond dès que la commande a été effectuée. Le protocole de communication est en code ASCII avec une syntaxe qui permet une utilisation simple, tant manuelle que programmée.
Le serveur propose une cinquantaine de commandes pour travailler avec AxiCat. En voici quelques exemples.

Configurer la broche 1 de GPIO en sortie, la mettre au niveau bas et en lire l'état :

```
commande    iod 1 out
réponse     iod ok
commande    iow 1 0
réponse     iow ok
commande    ior 1
réponse     ior 01 1 0 out
```

horloge en temps réel PCF2129A : activer I²C maître, écrire dans le registre index 0, lire les registres 0 à 9 :

```
commande    ime
réponse     ime ok
commande    imw 81 0
réponse     imw 081 00001 ack
commande
réponse     imr 081 08h 16h 04h
30h 22h 16h 06h 00h 08h 16h nack
```

activer SPI maître, transférer quatre octets et sélectionner esclave n°2 :

```
commande    sme
réponse     sme ok
commande    smt 2 0FFh 0FFh 10100b 5
réponse     smt 2 000 255 032 005
```

Beaucoup de réponses du serveur sont des valeurs numériques. Le format de ces nombres est totalement réglable. On peut choisir entre binaire, décimal, hexadécimal, avec ou sans zéros non significatifs. Le protocole de communication est purement asynchrone pour que le client ne doive pas attendre de réponse avant d'envoyer la commande suivante. Le client peut même envoyer une salve de commandes pour augmenter l'efficacité du système. Au besoin, on peut attribuer un numéro d'identification à chaque commande, il sera renvoyé dans la réponse.

Le serveur permet d'envoyer directement des octets de données entre chacun des deux UART et un port réseau. Cette option vous permet, avec un terminal comme PuTTY, d'écrire et de lire directement des données par l'UART.

`AxiCat Application Layer` est un module qui met à disposition une API en C par-dessus le protocole sériel. Cette API marche avec toutes les fonctions du protocole sériel, y compris le transfert asynchrone, ce qui permet ainsi au programmeur de commander au mieux et en parallèle les différentes interfaces d'AxiCat. Le code source d'AxiCat Server donne un bel aperçu de l'usage de cette API. Le module est utilisable dans différents programmes : *AxiCat Server, AxiCat AVR ISP, I/O Card Explorer, 1-Wire Automation Server*.

Le programme `AxiCat AVR ISP` transforme AxiCat en programmateur *in situ* pour les microcontrôleurs AVR à 8 bits. Il peut flasher des mémoires, des EEPROM et régler les fusibles. Il peut lire les fichiers HEX et raw d'Intel.

AVR ISP travaille avec un bus SPI (mais sans *slave select*) et une ligne de RàZ. On relie le bus SPI d'AxiCat au microcontrôleur et une ligne de GPIO (au choix) à la broche de RàZ du microcontrôleur, par exemple comme à la **figure 3**.

L'outil `AxiCat Command Line` convertit des commandes textuelles simples en paquets d'octets et les envoie directement à l'AxiCat. Il sert principalement à développer et déboguer le programme principal dans l'AxiCat. Il permet d'envoyer à l'AxiCat des paquets non valides pour tester aisément la robustesse du programme principal. C'est aussi un moyen pratique pour étudier le protocole sériel.

En conclusion

Au labo d'Elektor, nous utilisons surtout l'AxiCat pour vérifier le micrologiciel et le charger dans nos propres cartes d'extension pour Raspberry Pi (Swiss Pi et PiWire+). Nous développons aussi tout le logiciel à l'aide d'AxiCat. Écrire et déboguer le code C va toujours plus vite sur PC que sur RPi. En fin de parcours, nous compilons et testons le logiciel sur un vrai RPi.

Vous pouvez vous procurer à l'e-choppe l'AxiCat en module assemblé prêt à l'emploi [2] et sur [3], vous trouverez de nombreuses autres informations et tout le logiciel nécessaire. ⏮

(150585 – version française : Robert Grignard)

Liens

[1] Téléchargement du logiciel : www.elektormagazine.fr/150585

[2] AxiCat à l'e-choppe : www.elektor.fr/150585

[3] Informations et logiciel : www.axiris.eu/en/index.php/i-o-cards/axicat

Liste des composants

Résistances :
(toutes 1206, sauf indication contraire)
R1 = 100 Ω
R2, R3 = 4,7 kΩ
R4, R5 = 10 kΩ
R6 = 1 M Ω

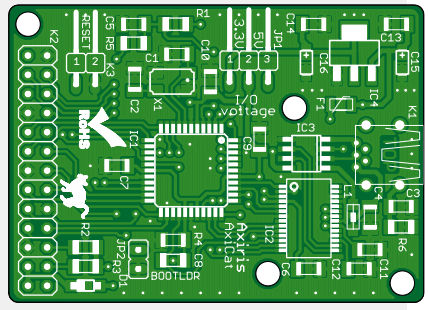
Condensateurs :
(tous 1206, sauf indication contraire)
C1, C2 = 22 pF
C3 = 4,7 nF
C4, C5 = 10 nF
C6 à C10 = 100 nF
C11 à C14 = 1 µF
C15 = 10 µF 10 V tantale, boîtier A
C16 = 22 µF 10 V tantale, boîtier A

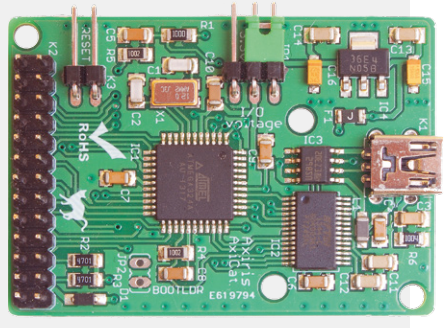
Inductance :
L1 = perle de ferrite 3 A
(Farnell réf. 1653393)

Semi-conducteurs :
D1 = diode Schottky 1N5819, 1 A, 40 V, SOD123
IC1 = ATmega164A-AU*
IC2 = FT245RL
IC3 = USB6B1
IC4 = LM1117IMP-3.3

Divers :
F1 = polyfuse 350 mA, 1206
JP1 = embase mâle à 3 picots coudés, au pas de 2,54 mm, avec cavalier
K1 = connecteur mini-USB type B
K2 = embase mâle à 2x13 picots, au pas de 2,54 mm
K3 = embase mâle à 2 picots coudés, au pas de 2,54 mm, avec cavalier
X1 = quartz 12 MHz

*cf. texte





PROJET DE LECTEUR

testeur portatif OBD2 à base de Raspberry Pi

avec un nouveau logiciel de diagnostic

Thomas Beck

Le module Pi-OBD-HAT est une carte d'extension pour Raspberry Pi qui transforme ce nano-ordinateur en une interface de diagnostic OBD2. Un inconvénient des solutions précédentes était qu'on devait les interfacer avec un émulateur de terminal. Le logiciel de diagnostic HHGui présenté ici est d'un emploi bien plus simple et commode.

Le logiciel HHGui (interface graphique pour Pi-OBD portatif) est une variante de mon émulateur portatif OBD2, HHEmu [1], une espèce de couteau suisse pour les développements autour de l'« analyseur OBD2 NG/DIAMEX Handheld Open » (HHOpen). Ce testeur OBD2 portatif a déjà été présenté dans Elektor (09/2009, voir [2]). L'analyseur OBD2 NG *(Nouvelle Génération)* est basé sur le module DXM-OBD2 [3], un prédécesseur de l'interface AGV4000C du module Pi-OBD-HAT *(OnBoard Diagnostic - Hardware Attached on Top)*. Aujourd'hui HHGUi est adapté à ce module Pi-OBD-HAT.

Historique du développement

L'itinéraire inhabituel du développement du logiciel montre qu'il est possible de porter un micrologiciel écrit à l'origine

pour un microcontrôleur vers un Raspberry Pi ou un PC – dans l'idéal sans avoir à en modifier le code source.
À l'origine, HHEmu est un outil de développement pour mettre à jour à partir d'un PC le micrologiciel du µC AVR de l'analyseur ODB2, et sans véhicule pour pouvoir le tester. C'est pourquoi HHEmu contient également un simulateur OBD2 qui fournit des données simulées au micrologiciel via un module DXM-OBD2 également simulé.
Dans la phase suivante du développement, HHEmu est doté d'un accès aux ports série d'un PC, ce qui, en combinaison avec l'extension Bluetooth (04/2010, voir [4]), offre à l'analyseur OBD2 NG des possibilités remarquables. Le micrologiciel exécuté sur un PC dans HHEmu peut alors établir, via le port série d'un adaptateur Bluetooth, une connexion avec l'extension Bluetooth d'un véritable analyseur NG et donc commander le module DXM. Il est possible de tester un nouveau micrologiciel sur un véhicule sans avoir à le charger au préalable dans l'analyseur OBD2. Une étape importante est franchie, HHEmu devient alors LE logiciel de diagnostic des interfaces de diagnostic OBD2 basées sur DXM. De plus, le simulateur OBD2 de HHEmu peut servir de générateur de données pour le logiciel de diagnostic d'autres fabricants ou pour le développement de nouveaux logiciels. Outre la plus grande facilité de test du micrologiciel, le portage vers le PC montre un autre avantage : les fonctions de chaque nouvelle mise à jour du micrologiciel se retrouvent automatiquement à l'identique dans le logiciel de diagnostic.

Nous voilà au portage de HHEmu vers Raspberry Pi : après toutes les adaptations au module Pi-OBD et la suppression de tous les menus spécifiques à l'analyseur OBD2 NG, la première version de HHGui est prête pour la publication.

Modes de fonctionnement

Du fait de son histoire passée, HHGui est en premier lieu un logiciel de diagnostic OBD2 qui imite l'interface utilisateur de l'analyseur OBD2 NG et utilise en interne le micrologiciel de celui-ci, adapté au module Pi-OBD. L'étendue des fonctions actuelles du micrologiciel est détaillée dans l'**encadré**.
HHGui affiche ses menus utilisateur et les données OBD2 comme sur un afficheur à cristaux liquides, lequel est simulé et adapté sur l'écran graphique, plus grand, du RPi. On a là un compromis entre l'utilisation de l'écran tactile officiel de 7 pouces recommandé pour le RPi et d'écrans plus petits de 320×240 pixels. Toutefois dans ce cas, une modification du module Pi-OBD est nécessaire, voir la description en ligne dans la première mise à jour du projet [5].
La deuxième fonction implémentée dans le HHGui est un simulateur OBD2, qui simule le module Pi-OBD ainsi que jusqu'à huit contrôleurs *(Electronic Control Units)*, ce qui offre la possibilité de tester tout le fonctionnement du logiciel en l'absence du module Pi-OBD ou de véhicule. Comme un véhicule réel donné ne présente qu'un sous-ensemble de toutes les fonctions OBD2 (dont une grande partie se répartit entre modèles à essence et modèles diesel), on dispose d'un moyen simple de tester l'ensemble des fonctions. De plus, cette variante est totalement sans danger, l'effacement involontaire de données OBD2 dans le véhicule étant impossible. La gamme de fonctions OBD2 du simulateur est identique à celle du micrologiciel. La description des commandes à touches du simulateur OBD2 est disponible en ligne, voir le projet dans le labo d'Elektor en [1].

Possibilités du diagnostic OBD2

Revenons à la fonction principale : le logiciel de diagnostic OBD2. Quels services peut-on ou non en attendre pour un véhicule ?
Quand on lit la spécification la plus importante sur le sujet ISO 15031-5 [6] (ou SAE J1979 [7], avec le même contenu), on se rend compte qu'avec les services OBD2, on ne peut lire et effacer partiellement que des données et des défauts relatifs aux émissions. Ces données sont pour la plupart fournies par le contrôleur du moteur. Pour les véhicules à boîte de vitesses automatique, le contrôleur de celle-ci fournit aussi des données OBD2. Plus rarement, d'autres contrôleurs ou des contrôleurs de plusieurs moteurs (par ex. sur les véhicules hybrides) peuvent fournir des données. Les configurations telles que le verrouillage automatique des portières ou la suppression du signal sonore de la ceinture de sécurité ne sont pas du domaine des services OBD2. À l'exception du voyant de contrôle du moteur, on ne peut pas non plus éteindre les voyants d'alarme du tableau de bord, ni réinitialiser les indicateurs ou les intervalles de maintenance. Tout cela n'est possible qu'avec un moyen de diagnostic spécifique du constructeur, qui varie

Étendue des fonctions OBD2 du logiciel de diagnostic HHGui

HHGui supporte les services listés ci-dessous, d'après ISO 15031-5. On trouvera une description plus précise de ces services sur la page du projet [5]. Des informations sur les sous-fonctions sont disponibles sur [6], [7] et [9]. Les services et sous-fonctions effectivement disponibles dépendent de chaque véhicule et, plus précisément, de ses contrôleurs OBD2. HHGui interroge chaque contrôleur sur ses sous-fonctions disponibles et n'affiche que celles-ci.

- Supporte au maximum huit contrôleurs, selon ISO 15765-4 [10]
- Mode OBD2 0x01 : lecture des valeurs instantanées (*Current Data*) ; sous-fonctions supportées (*Parameter Identifier, PID*) : 0x00 à 0x60, 0x70, 0x80, 0x8D
- Mode OBD2 0x02 : lecture des données d'environnement des défauts (*Freeze Frame Data*), PID supportés : voir mode 0x01
- Mode OBD2 0x03 : lecture des défauts confirmés (*Confirmed Diagnostic Trouble Codes, DTC*)
- Mode OBD2 0x04 : effacement des défauts, des données d'environnement des défauts, du statut des voyants indicateurs de défaut (*Malfunction Indicator Lamps, MIL*), des données du moniteur OBD2 et autres
- Mode OBD2 0x07 : lecture des défauts en attente (*Pending DTC*)
- Mode OBD2 0x09 : lecture des informations du véhicule (*Vehicle Information*) ; sous-fonctions supportées : 0x00, 0x02, 0x04, 0x06, 0x08, 0x0A, 0x0B
- Mode OBD2 0x0A : lecture des défauts permanents (*Permanent DTC*)

d'un constructeur à l'autre.
Vous êtes déçu !? Toutefois il y a un avantage à cette prééminence des constructeurs dans la standardisation du diagnostic OBD2 : aujourd'hui, 20 ans après son introduction aux États-Unis (en l'an 2000 en Europe pour les véhicules à essence), cela fonctionne pour pratiquement n'importe quel véhicule. Depuis 2008, le nombre de protocoles d'échange de données OBD2 s'est fortement réduit, bientôt à un seul, les véhicules neufs ne peuvent plus utiliser que le protocole CAN.
Si l'on examine plus attentivement les sous-fonctions OBD2, on découvre un nombre ahurissant de données de capteurs, de compteurs et autres (en tout, plus de cent), propres à éveiller la curiosité de l'explorateur. De plus, contre toute attente, on n'y trouve pas que des données du contrôleur du moteur. Certaines d'entre elles, étiquetées par le diagnostic comme provenant du moteur, sont en fait issues d'autres contrôleurs, comme celui des freins.

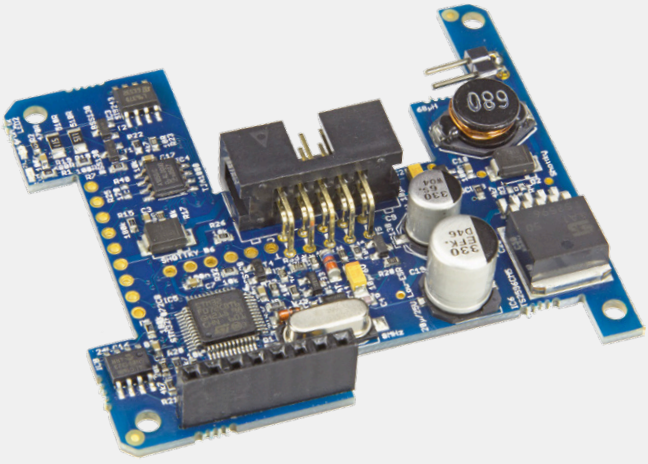
Parmi les données OBD2 faciles à interpréter, on compte celles qui sont déjà affichées par le véhicule, mais après une transformation :

- par filtrage passe-bas (température extérieure, niveau de carburant et d'une manière générale tous les afficheurs à aiguille réalisés avec des moteurs pas-à-pas)
- avec une fonction plateau (température du liquide de refroidissement) [8]
- avec des décalages ajoutés pour respecter des contraintes réglementaires d'affichage minimum, des tolérances, ou simplement des désirs du constructeur (vitesse, tachymètre)

Dans ces cas, le diagnostic OBD2 fournit des données « brutes », non transformées, par ex. la vitesse réelle et non celle affichée.
Ensuite, il y a les données qu'on aimerait bien voir, mais que le constructeur n'a pas prévu d'afficher, par ex. la température de l'huile ou la pression du turbocompresseur. Même la puissance fournie par le moteur est une information utile.
Pour l'utilisateur lambda, il est intéressant de connaître les données OBD2 qui signalent que le véhicule est en état de passer le prochain contrôle technique. Pour cela il faut que tous les programmes de contrôle OBD2 (les moniteurs OBD2) aient été exécutés avec succès au moins une fois depuis le dernier effacement des défauts. Cet état est atteint quand HHGui affiche dans le menu *Inspection/Maintenance-Readiness* l'état « …monitoring ready: YES » pour tous les moniteurs OBD2.
À côté de ces données OBD2 faciles à interpréter, il y en a beaucoup d'autres dans le domaine du réglage *lambda* (rapport air/carburant), plutôt réservées aux spécialistes. Les lister toutes ici serait beaucoup trop long. Comme les versions les plus récentes des normes mentionnées ci-dessus ne sont disponibles qu'à titre (très) onéreux, il est recommandé de consulter Wikipedia [9].

PI-OBD : le matériel

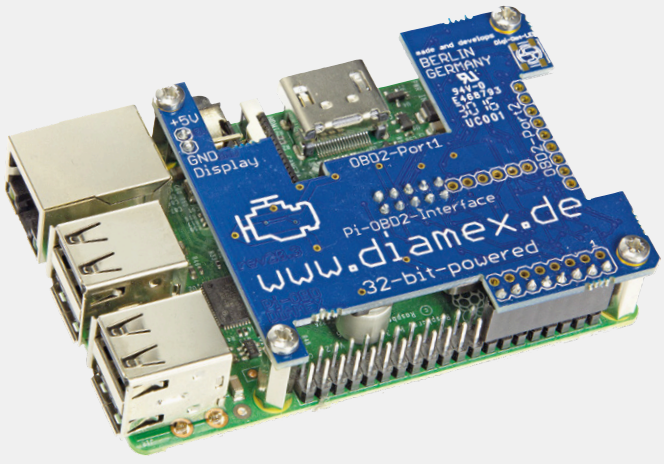
Le module Pi-OBD-HAT (disponible dans l'e-choppe) de Diamex transforme le Raspberry Pi en un gestionnaire de diagnostic OBD2. Si l'on équipe le RPi avec un écran tactile de 7 pouces, on a en main un système de diagnostic autonome pour véhicule, avec une interface OBD2.
Le Raspberry Pi ainsi que l'écran sont alimentés par



l'alimentation de bord du véhicule.
Pour cet appareil, toutes les versions du RPi conviennent en principe, toutefois, compte tenu de la vitesse plus élevée de leur processeur, il est indiqué de donner la préférence au RPi 2 modèle B ou RPi 3 modèle B. Le module Pi-OBD n'occupe que huit broches de l'interface GPIO du RPi, pour l'alimentation, le port série, une réinitialisation et l'activation du démarrage du Pi-OBD. La communication s'effectue au travers du port série configuré 8N1 (8 bits/sans parité/1 bit de stop) à la vitesse de 115.200 baud.

Montage

Pour commencer, on enfiche le module Pi-OBD sur le RPi.



Le connecteur femelle à 8 broches doit être enfoncé sur la rangée extérieure du connecteur GPIO du RPi. Ensuite le module est fixé solidement au RPi au moyen des entretoises fournies. Le RPi est alimenté par les broches du connecteur GPIO dès que le module Pi-OBD est connecté par câble à la prise OBD2 du véhicule. Excepté pendant le test et la configuration, aucune alimentation par la prise micro-USB du RPi n'est nécessaire. Pour alimenter l'écran de 7 pouces

Passons maintenant à la partie intéressante pour le développeur.

HHGui : structure et fonctionnement

Principe de fonctionnement

Le micrologiciel est exécuté dans un fil *(thread)*, toutes les entrées et sorties sont traitées dans d'autres fils. Pour cela, il faut simuler des registres et des interruptions du µC.

Réalisation effective

HHGui se compose de six fils. La **figure 1** en montre un synoptique simplifié avec les interactions entre les fils. Les noms des fils dans les fichiers sources sont différents du fait de la longue histoire de ce logiciel. Le fil ***Pi-OBD-Module*** y porte le nom de `DxmThread` et le fil ***OBD2-Analyser NG*** s'appelle `HhopenThread`. Dans la suite, nous allons utiliser ces noms plus courts.

Le `HhopenThread` est le fil dans lequel est exécuté le micrologiciel de l'analyseur OBD2. Ce fil communique de manière

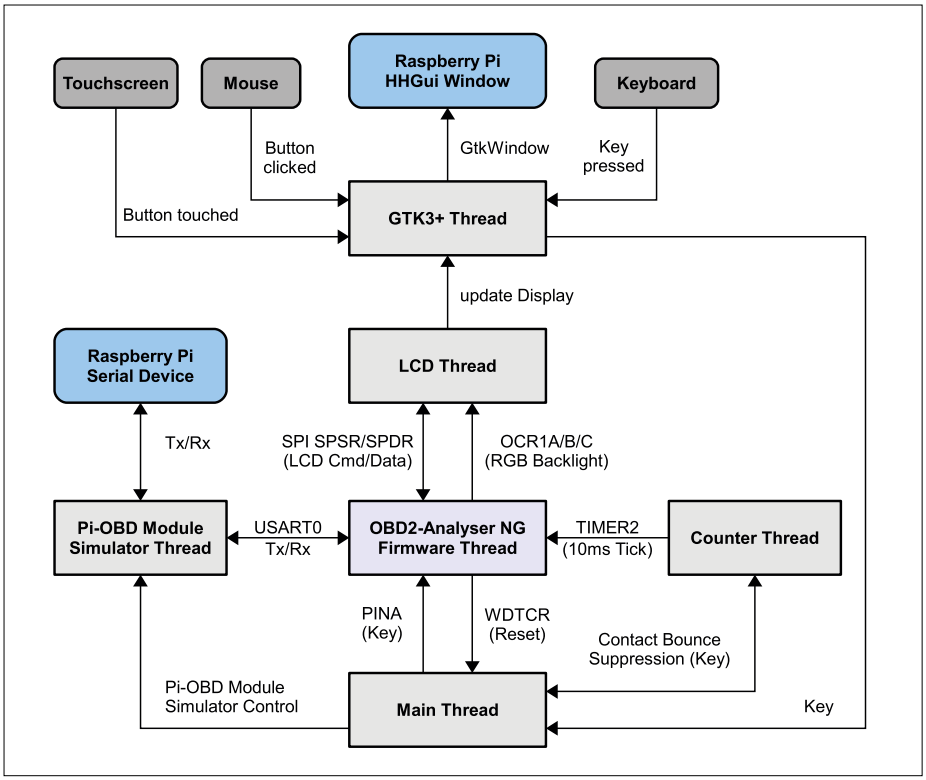


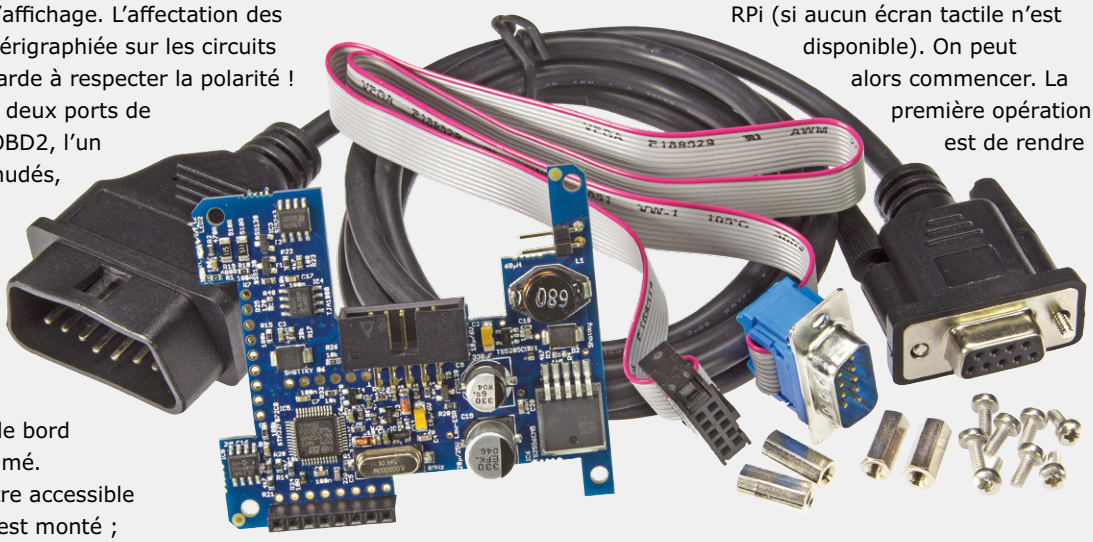
Figure 1. HHGui : diagramme fonctionnel.

du RPi, il faut connecter les deux broches d'alimentation 5 V de son module d'affichage aux broches correspondantes du module Pi-OBD. La **photo** montre cette connexion avec, en bas, le RPi équipé du module OBD et, en haut, l'arrière de l'écran avec le module d'affichage. L'affectation des différentes broches est sérigraphiée sur les circuits imprimés. Prenez bien garde à respecter la polarité !

Le module est équipé de deux ports de raccordement du câble OBD2, l'un (port 2) pour des fils dénudés, l'autre (port 1) pour un câble plat terminé à l'autre bout par un connecteur Sub-D.

On peut monter un connecteur plat (mâle ou femelle) sur le port 2 qui se trouve sur le bord extérieur du circuit imprimé.

Ce port a l'avantage d'être accessible même quand le module est monté ; attention toutefois à une possible erreur de polarité. Pour le port 1, équipé d'une prise carénée avec détrompeur, une telle erreur est impossible, la présence d'un connecteur Sub-D interdit l'inversion. Il faut toutefois connecter le câble plat avant l'assemblage du RPi et du module, cette opération est impossible par la suite.



Mise en service

La mise en service du Pi-OBD-HAT nécessite des connaissances de base du Raspberry Pi, de l'utilisation de Linux et de l'installation de logiciels. Pour la configuration et l'utilisation, il faut raccorder un clavier et une souris au RPi (si aucun écran tactile n'est disponible). On peut alors commencer. La première opération est de rendre disponible le port série du RPi, opération dont la description sortirait du cadre de cet article, d'autant plus qu'elle présente des différences entre les différents modèles du RPi. Cette procédure est toutefois décrite en détail dans le mode d'emploi [11]. Après un test de fonctionnement au moyen d'un émulateur de terminal, l'appareil est prêt à l'emploi.

variée avec les autres fils. Le registre SPDR sert à transférer les données à afficher au fil `LcdThread`, lequel reçoit également les données RVB pour le réglage du rétroéclairage par les registres OCR1A/B/C. Les commandes AT ainsi que les requêtes OBD sont transmises au fil `DxmThread` au travers du registre UDR0 par lequel passent aussi les réponses. Le fil `MainThread` demande un redémarrage du micrologiciel ou de HHGui par l'intermédiaire du registre WTCR et reçoit les actions sur les touches par le registre PINA. Le fil `HhopenThread` est piloté par les fils `CounterThread` et `DxmThread` par appel de ses fonctions de traitement des interruptions. D'autres détails suivent dans les descriptions des cinq autres fils.
Le fil `MainThread` est responsable de l'exploitation des paramètres des lignes de commande, de l'initialisation du port série, de la création des autres fils et, lors de la fin d'exécution, de la restitution des ressources. Entretemps, ce fil exploite dans une boucle sans fin les états des touches qui lui sont fournis par le fil `GtkThread`. Parmi ces touches, il faut distinguer les touches de commande du simulateur OBD2 des touches Up/Down/ESC/OK pour le fil `HhopenThread`. Pour ce dernier, une fonction de suppression du rebond des touches incluse dans le micrologiciel doit être exécutée : les touches doivent être présentées au fil `HhopenThread` à travers le registre PINA, dans l'état « pressé » pendant 40 ms, pour que le micrologiciel identifie une pression sur une touche. Le délai de 40 ms est évalué en collaboration avec le fil `CounterThread`. Les touches qui ne sont utilisées que pour le simulateur OBD2 provoquent des modifications des réglages dans le fil `DxmThread`.
La boucle sans fin de gestion des pressions des touches est quittée sur une réinitialisation du chien de garde provoquée par le fil `HhopenThread` (suite à une pression prolongée de la touche ESC) ou bien par la fin de HHGui sur pression de la touche q (*quit*).
Le fil `LcdThread` analyse les commandes et les données à afficher que lui transmet le fil `HhopenThread` à travers l'interface SPI simulée et les convertit en directives pour la bibliothèque graphique utilisée GTK3+. Les valeurs RVB lues au travers des registres OCR1A/B/C pour le rétroéclairage, combinées avec la valeur du contraste (lue par l'interface SPI), déterminent les couleurs affichées. Comme l'afficheur utilisé par l'analyseur OBD2 NG utilise un contrôleur ST7565R, le fil `LcdThread` contient un simulateur de ce contrôleur pour les commandes ST7565R utilisées par le micrologiciel.
Le fil `GtkThread` est nécessaire, car la boucle principale de GTK3+ (fonction *gtk_main()* de la bibliothèque GTK3+) est bloquante. Les fonctions de dessin de GTK3+ ne peuvent être appelées que de l'intérieur du fil `GtkThread` (ou dans le contexte de la boucle principale de GTK3+). Quand le fil `LcdThead` demande un rafraîchissement de l'affichage, il installe pour cela une fonction *callback* qui est appelée ultérieurement par la boucle principale de GTK3+. Le fil `GtkThread` exploite les événements d'entrée de toutes sortes et les transforme soit en pressions sur les touches Up/Down/ESC/OK, soit en événements de touches qui pilotent le simulateur OBD2.
Le fil `CounterThread` sert en premier lieu à émuler les interruptions cadencées à 10 ms du *Timer2*. Pour cela, il appelle toutes les 10 ms la fonction de traitement des interruptions (*Interrupt Service Routine, ISR*) `TIMER2_COMP_vect()`. D'autre part, il collabore avec le fil `MainThread` pour la fonction anti-rebond des touches, comme déjà mentionné.
En mode *logiciel OBD2*, le fil `DxmThread` fonctionne comme adaptateur d'interface et transmet les données de manière transparente. Les données reçues par le registre USART de données d'entrée-sortie UDR0 (commandes AT pour le module Pi-OBD ou requêtes OBD2) sont envoyées par le port série configuré au véritable module Pi-OBD. Après chaque octet lu, le micrologiciel est averti par l'appel de l'*ISR* `USART_UDRE_vect()` que cet octet a été envoyé et que l'octet suivant peut être écrit dans le registre. Les réponses du module Pi-OBD sont reçues par le port série et envoyées au micrologiciel via le registre UDR0. La réponse (OK, caractère > d'invite du module Pi-OBD, erreur ou données OBD2) est également écrite octet par octet dans le registre UDR0 et signifiée au micrologiciel par appel de l'*ISR* `USART0_RX_vect()`, lequel vient lire les données dans ce registre.
En mode *simulateur OBD2*, le fil `DxmThread` simule le module Pi-OBD2 et un ou plusieurs contrôleurs. Il n'y a alors pas de communication à travers le port série physique. ⏮

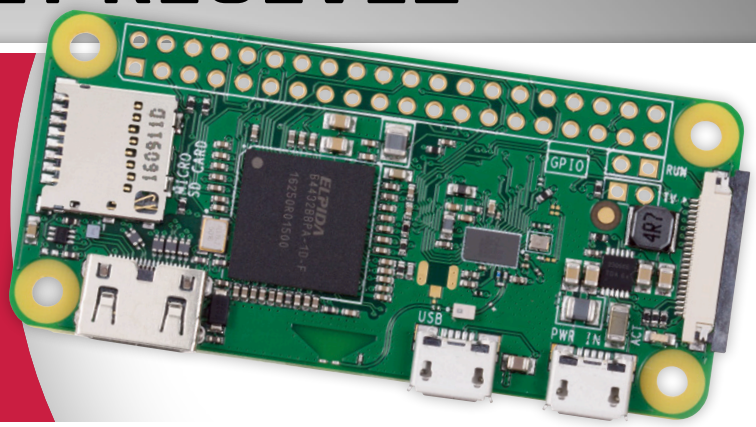
(160204 – version française : Helmut Müller)

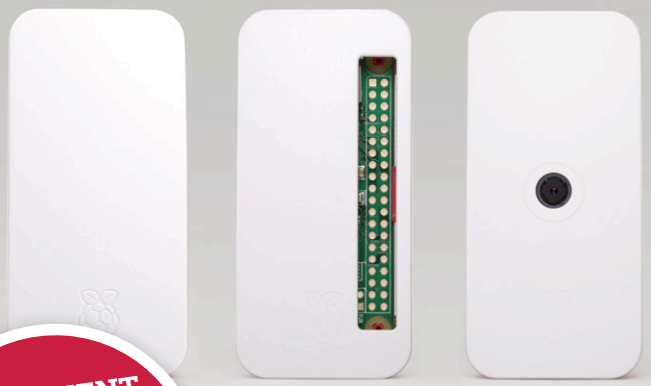
Liens

[1] HHEmu : www.elektormagazine.fr/labs/firmware-update-and-emulator-for-obd2-analyser-ng-wireless-obd2
[2] *Analyseur OBD2 (Nouvelle Génération)*, Elektor 09/2009 : www.elektormagazine.fr/magazine/elektor-200909/11390
[3] Module DXM : www.diamex.de/dxshop/DIAMEX-DXM-OBD2-Modul
[4] *OBD-2 sans fil, module d'extension Bluetooth pour l'analyseur OBD2 NG*, Elektor 04/2010 : www.elektormagazine.fr/090918
[5] HHGui : www.elektormagazine.fr/labs/obd2-for-raspberry-pi
[6] www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=66368
[7] http://standards.sae.org/j1979_201408/
[8] Fonction plateau : http://www.t4-wiki.de/wiki/Plateau-Funktion
[9] https://en.wikipedia.org/wiki/OBD-II_PIDs
[10] www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=67245
[11] Pi-OBD-HAT : www.elektor.fr/pi-obd-hat-obd2-module-for-raspberry-pi

ABONNEZ-VOUS ET RECEVEZ

RPI ZERO W GRATUIT





SEULEMENT **54,95 €** PAR AN (6 NUMÉROS)

TOUS LES 2 MOIS, LES DERNIÈRES NOUVELLES DU RASPBERRY PI ET LES MEILLEURS PROJETS !

Souscrivez dès maintenant un abonnement d'un an au magazine MagPi, nous vous offrons :

- Six numéros du magazine MagPi
- Une carte Raspberry Pi Zero W
- Un boîtier avec trois couvercles différents
- Un connecteur pour module de caméra
- Un câble HDMI/mini-HDMI et un câble micro-USB/USB OTG

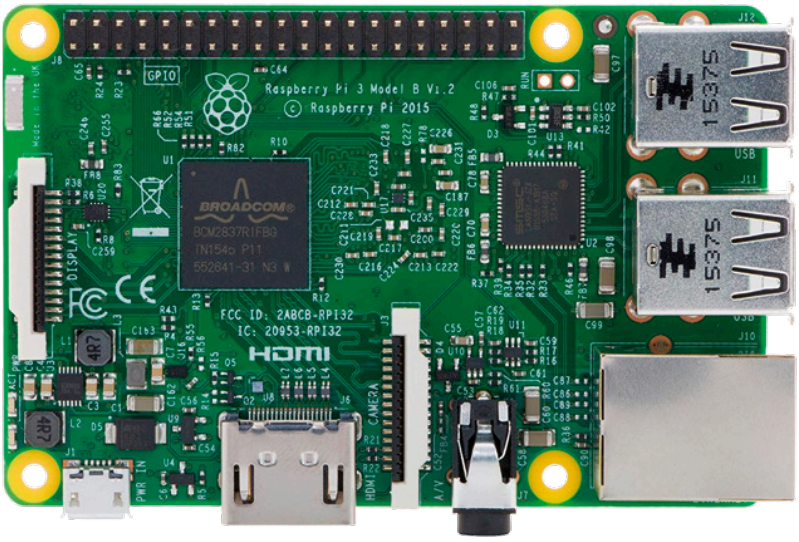
Vos avantages :

- Prix au numéro réduit
- Chaque numéro directement dans votre boîte aux lettres
- Cadeau de bienvenue d'une valeur de 22,95 €
- Découverte de chaque nouveau numéro avant sa sortie en kiosque

ABONNEZ-VOUS : WWW.MAGPI.FR

Android sur Raspberry Pi (1)

mesurer, commander, réguler : les broches GPIO



Google sort une nouvelle version d'Android, spécialement conçue pour les ordinateurs monocarte ainsi que la mesure, la commande et la régulation. Nous avons testé Android sur le Raspberry Pi et nous vous montrons pas à pas comment réaliser un premier projet.

Tam Hanna (Slovaquie)

Les progrès rapides de Microsoft dans le domaine du Raspberry Pi ont certainement dû donner des maux de tête à Google : après l'ordinateur de bureau, la tablette et l'ordiphone, l'Internet des Objets est bien parti pour devenir le prochain champ de bataille.

Convergence, c'est le maître-mot – Google est contraint de se lancer dans le domaine de l'Internet des Objets pour ne pas laisser à Microsoft la liberté d'en profiter pour une nouvelle attaque du marché des portables.

Le système « Brillo », annoncé il y a un an, a eu peu de succès. Pour ce deuxième essai, on ne fait pas dans le détail et on appelle le nouveau produit simplement « Android Things » (en français *Objets Android*).

Attention à la qualité !

Si vous achetez des cartes en Chine, soumettez-les à des contrôles rigoureux. Beaucoup de fabricants vendent des lots de matériel de mauvaise qualité qui annoncent au contrôleur la capacité nominale, mais qui en pratique perdent des données.

Accès plus simple au matériel

Commençons par le plus évident. Malgré tout le tapage – les analogies avec « Windows 10 IOT Core » ne sont que des coïncidences – *Android Things* n'est en gros qu'une version standard d'Android, avec une extension pour l'interaction avec le matériel. La **figure 1** montre la structure du système d'exploitation.

La bibliothèque « Things Support Library » se compose de deux modules. Le premier est l'Interface de Programmation des Périphériques (*Peripheral I/O API*) qui, à l'heure où nous écrivons ces lignes, offre un accès aux périphériques suivants (il n'est pas clair si *OneWire* sera ajouté par la suite) :

- GPIO avec MLI
- I²C
- SPI
- UART

Le deuxième est l'Interface de Programmation des Pilotes Utilisateur (*User Driver API*). Il s'agit d'une interface de programmation qui permet aux développeurs de mettre à la disposition du reste du système les données fournies par leurs capteurs maison. Vous trouverez en [1] davantage d'informations sur les modules contenus dans la bibliothèque *Things Support Library*.
Il est temps de nous lancer dans nos premiers essais avec le matériel.

Images complètes

À l'heure où nous rédigeons cet article, Google, qui a apparemment tiré les leçons de l'échec de Brillo, propose des images d'*Android Things* pour les plateformes Edison, NXP Pico et Raspberry Pi 3. Ici, nous allons nous intéresser au Raspberry Pi 3, de loin le plus répandu. Pour commencer, téléchargez l'image disponible en [2] et copiez-la sur une carte SD selon la procédure habituelle (voir aussi l'**encadré** « Attention à la qualité »). Dans la suite, l'auteur utilise une carte de 8 Go. L'utilisation de cartes de plus grande capacité ne devrait pas poser de problèmes, alors que la taille de l'image, à savoir 4,6 Go, interdit l'usage de cartes plus petites.
Raccordez ensuite l'écran et un câble réseau au RPi et démarrez en mettant sous tension (le clavier et la souris ne sont pas nécessaires, car *Android Things* n'affiche qu'un écran de bureau sans possibilité d'interaction directe). Nous n'insisterons pas sur le débogage en mode liaison Wi-Fi, parce que la latence plus élevée due à la liaison sans fil peut introduire des délais parasites dans la chasse aux erreurs. Si vous voulez néanmoins vous y risquer, suivez les instructions de connexion au réseau Wi-Fi en [3].
Même sur un Raspberry Pi 3, le premier démarrage d'*Android Things* peut prendre une minute ou deux. Tout au début apparaît un message d'absence d'Ethernet, qu'on peut ignorer si le câble réseau est effectivement branché.
Ceci fait, le système affiche l'écran de démarrage représenté sur la **figure. 2**. L'adresse IP – ici 10.42.0.44 – va nous être utile dans un instant.

Environnement de développement

Ne serait-ce que par manque de place, il nous faut supposer ici que vous disposez sur votre PC de développement d'un environnement *Android Studio* opérationnel, sinon, vous trouverez conseils et soutien en [4]. Pour les étapes suivantes, l'auteur utilise une station de travail AMD à 8 cœurs sous Ubuntu 14.04 ; sous Windows et Mac Os, cela ne devrait être guère différent.
Prochaine étape : positionnez-vous sur le PC de développement sous le répertoire racine de la Passerelle de Débogage d'Android (*Android Debug Bridge, ADB*). Puis connectez-vous au RPi au moyen des commandes de console (en gras) suivantes :

```
tamhan@TAMHAN14:~/Android/Sdk/
   platform-tools$ ./adb connect
   10.42.0.44

connected to 10.42.0.44:5555

tamhan@TAMHAN14:~/Android/Sdk/
   platform-tools$ ./adb devices

List of devices attached
10.42.0.44:5555 device

tamhan@TAMHAN14:~/Android/Sdk/
   platform-tools$
```

La commande `adb devices` n'est pas absolument indispensable, nous la mentionnons ici parce qu'elle permet de lister tous les périphériques connectés à la passerelle.
Notez qu'une connexion entre ADB et RPi peut être perdue, par exemple si le PC passe en veille.

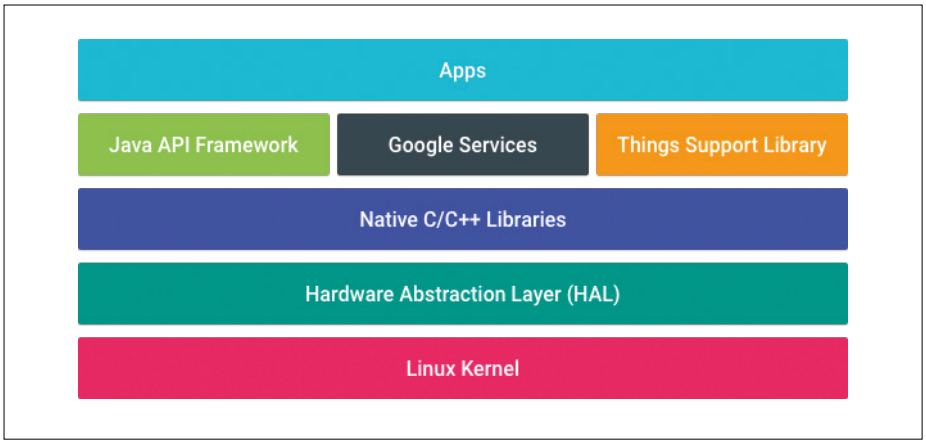


Figure 1. *Android Things* est une variante d'Android (source : Google).

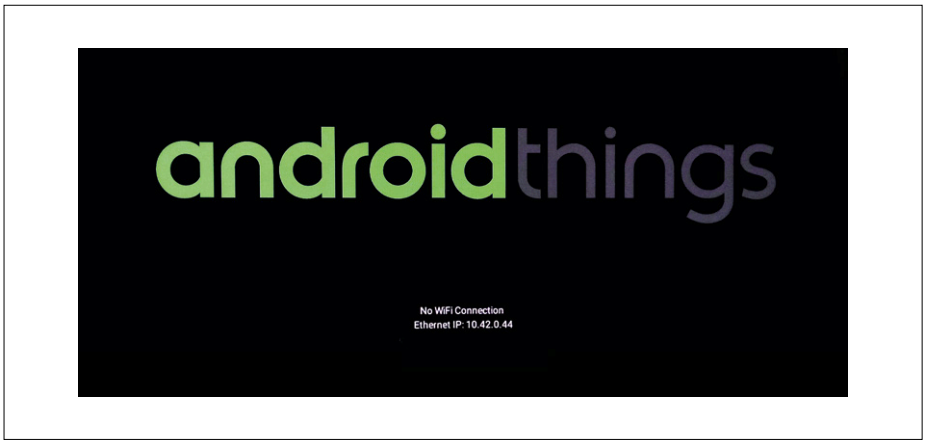


Figure 2. Notre *Android Thing* est prêt à l'emploi – notez son adresse IP.

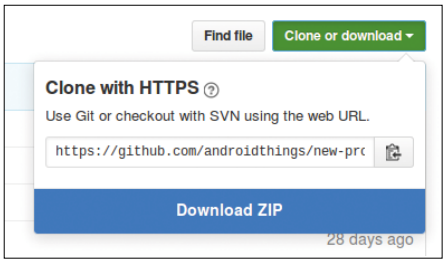


Figure 3. GitHub est et reste une pénibilité.

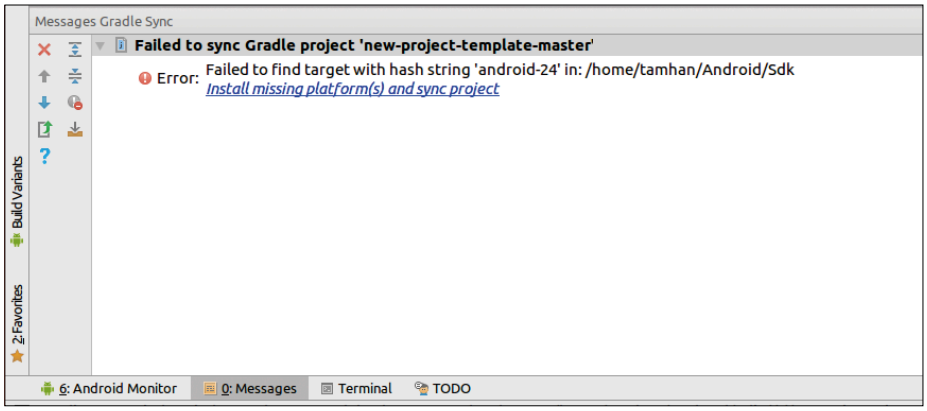


Figure 4. Ici, il manque un kit de développement logiciel compatible.

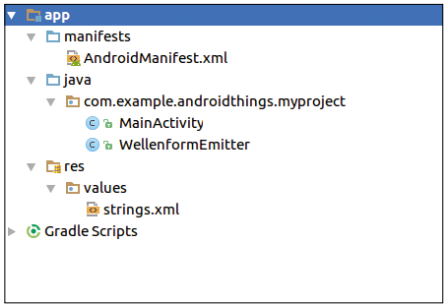


Figure 5. Cette application se passe de XML.

Figure 7. Celui qui bloque le fil *GUI* est sanctionné (source : Google).

Vu l'état de développement précoce d'*Android Things*, il n'y a actuellement pas de modèle disponible dans le générateur de projet d'*Android Studio*. À la place, vous devez accéder au dépôt GitHub en [5] et cliquer sur le bouton apparaissant sur la **figure 3** pour télécharger un exemple de squelette de projet.

À ce moment, si vous avez un projet ouvert dans *Android Studio*, fermez-le en cliquant sur *File -> Close Project*. *Android Studio* affiche alors la fenêtre d'accueil.

```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     |  15 |  14 |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     |  16 |  15 |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 |   1 |  18 |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 |   4 |  23 |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 |   5 |  24 |
 |  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 |   6 |  25 |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     |  10 |   8 |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     |  11 |   7 |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   |  31 |   1 |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 |  26 |  12 |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 |  27 |  16 |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 |  28 |  20 |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 |  29 |  21 |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
```

Figure 6. Ce tableau présente les broches du Raspberry Pi.

Vous devez alors extraire l'archive téléchargée à l'étape précédente dans un répertoire facile d'accès. Cliquez ensuite sur *Open existing Android Studio Project* et indiquez ce répertoire. Un clic sur *OK* et *Android Studio* démarre la construction du projet.

Dans le cadre de la synchronisation du projet avec *Gradle* (moteur de production intégré), *Android Studio* essaie automatiquement de télécharger les composants manquants. Un éventuel problème est signalé par un message d'erreur dans la fenêtre des messages (**fig. 4**). Un clic sur le lien conduit habituellement à sa résolution.

Ne vous étonnez pas si *Android Studio* se plaint fréquemment de versions périmées. *Android Things* exige la toute dernière version de divers composants du système d'exploitation, qui doivent être installés en plusieurs étapes.

Après un démarrage réussi d'*Android Studio*, il est bon de cliquer encore une fois sur *Build -> Make Project* pour lancer une compilation complète. Ceci est important dans la mesure où une connexion à l'internet est indispensable pour une première compilation. Ensuite, vous avez une bonne chance de pouvoir travailler aussi hors connexion avec le squelette de votre projet.

De menus changements

Le code de `MainActivity`, qui sert de point d'entrée, se distingue des applications normales d'Android par le fait que Google y insère des appels supplémentaires à `Log`. Ils sont nécessaires, car *Android Things* peut travailler sans écran. Dans ce cas, on obtient, grâce à ces appels, un minimum d'informations sur la console de débogage.

Assurez-vous aussi que le répertoire *res* est vide, comme le montre la **figure 5** : derrière cette *Activity*, il n'y a pas de procédure XML, ce qui se traduit dans le code (**listage 1**) par l'absence d'appels XML de chargement de ressources.

À noter aussi la structure du fichier de manifeste, reproduit partiellement dans le **listage 2**, qui présente deux modifications intéressantes : d'abord, la partie *Library* assure que les bibliothèques mentionnées plus haut seront bien liées au projet, ensuite, la *MainActivity* est pourvue d'un filtre additionnel *intent-filter* qui la désigne comme point d'entrée pour les appareils *Android Things*.

Ceci a son importance dans la mesure où *Android Things* doit pouvoir se passer d'un point de début de programme. Le RPi est en fait reconverti en un simple outil asservi à son application hôte.

Entrées/Sorties

Pour une première démonstration, intéressons-nous au moteur GPIO. Un vieil adage stipule que les performances en temps réel d'un système d'exploitation diminuent linéairement ou même exponentiellement avec sa complexité : c'est particulièrement désagréable avec les systèmes basés sur Java où le ramasse-miettes fait de temps en temps des siennes. Rappelons au passage que le Raspberry Pi est une plateforme en 3,3 V. Lui raccorder des périphériques en 5 V conduit à coup sûr au désastre !

Nous pouvons maintenant nous intéresser au code de l'application dans la *MainActivity* (**listage 3**). Commençons par ajouter deux variables membres. L'API des périphériques est réalisée sous la forme d'un service système. Un service est une ressource généralement disponible au niveau le plus large : les applications qui veulent l'utiliser n'ont qu'à se procurer un identifiant pour y accéder. La classe `Gpio` sert à contrôler la commutation des broches.
C'est la fonction `onCreate` (**listage 3**) qui procure l'accès au service.
Même si Google ne traite pas les fonctions GPIO à 100% de la même manière qu'Arduino, on voit néanmoins clairement ce qui se passe ici. Le tableau de la **figure 6** donne les identifiants BCM associés aux diverses broches. On obtient ce tableau en entrant la commande `GPIO Readall` sur un RPi (sous Raspbian standard, pas sous Android).

Mise au point

Pour évaluer la fiabilité ou la performance en temps réel d'un système d'exploitation, on peut effectuer un changement d'état périodique d'une sortie GPIO aussi rapidement que possible. Si l'on observe le spectre obtenu sur un analyseur de domaine de modulation, on peut en tirer quelques conclusions. Théoriquement, il n'y a pas d'objection à commuter la broche directement dans la fonction `onCreated`, qui est appelée au démarrage du programme. Mais si l'on essaie cela, on obtient le message d'erreur de la **figure 7**.

Listage 1. Squelette de la *MainActivity* – Point de démarrage du programme sans interface utilisateur

```
public class MainActivity extends Activity {
    private static final String TAG = MainActivity.class.
   getSimpleName();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, „onCreate“);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, „onDestroy“);
    }
}
```

La raison de ce comportement plutôt étrange tient à une spécificité du système Android : la partie interface utilisateur des programmes est exécutée dans un fil (*thread*) dédié, appelé *GUI*. Une application qui bloquerait ce fil se ferait « sortir » sans pitié par le système. Une boucle sans fin serait un blocage classique qui n'aurait à coup sûr droit à aucune pitié.
Pour contourner ce problème, nous pouvons transférer notre routine dans un autre fil.

Pour créer un nouveau fil, le moyen le plus simple est d'ajouter un *Runnable* au moyen d'une classe avec la structure suivante :

```
public class WaveformEmitter
   implements Runnable {
    @Override
    public void run() {
    }
}
```

Un *Runnable* n'est en somme rien d'autre

Listage 2. Fichier de manifeste avec l'inclusion de la bibliothèque

```
<manifest . . .>
    <application
      - - ->
        <uses-library android:name=“com.google.android.Things“/>

        <activity android:name=“.MainActivity“>

            . . .

            <intent-filter>
                <action android:name=“android.intent.action.MAIN“/>
                <category android:name=“android.intent.category.
   IOT_LAUNCHER“/>
                <category android:name=“android.intent.category.
   DEFAULT“/>
            </intent-filter>

        </activity>
    </application>
</manifest>
```

Listage 3. La classe Gpio gère la commutation des broches

```
public class MainActivity extends Activity {

    private static final String TAG = MainActivity.class.
   getSimpleName();
    PeripheralManagerService service;
    Gpio myGPIO;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, „onCreate“);

        PeripheralManagerService service = new
   PeripheralManagerService();
        try{
            myGPIO = service.openGpio(„BCM6“);
            myGPIO.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
            myGPIO.setValue(true);
        }
        catch (Exception e){Log.d(TAG, „Fehler:“ + e.getMessage());}
```

qu'une espèce de conteneur qui fait s'exécuter le code dans un nouveau fil. Outre la méthode `run` spécifiée ici, on peut bien entendu ajouter d'autres membres afin de stocker les informations nécessaires à l'exécution du fil.

Au complet, la classe résultante se présente comme dans le **listage 4**.

Outre le constructeur chargé de prendre en compte l'instance de la classe `Gpio`, nous avons peuplé la méthode `run()`,

Listage 4. La boucle sans fin de production du signal est exécutée dans son propre fil – un Runnable sert de conteneur

```
public class WaveformEmitter implements Runnable {
    Gpio myGpio;
    public WaveformEmitter(Gpio _which){
        myGpio=_which;
    }
    @Override
    public void run() {
        try {
            while(1==1){
                myGpio.setValue(true);
                myGpio.setValue(false);
                myGpio.setValue(true);
                myGpio.setValue(false);
                myGpio.setValue(true);
                myGpio.setValue(false);
            }
        }
        catch (Exception e){}
    }
}
```

qui est responsable de la forme d'onde émise. Nous produisons ici une forme d'onde caractéristique composée de trois rectangles, ce qui est intéressant dans la mesure où l'exécution de la boucle `while` est mise en évidence de manière séparée.

Maintenant la question est de savoir comment activer le *Runnable* dans `onCreate`. Une erreur classique de débutant serait l'appel direct de `run()`. Ce faisant, on exécute le code dans le contexte de la méthode appelante (c'est-à-dire à nouveau dans le fil *GUI*). La manière correcte d'activer le second fil est de créer une nouvelle classe de fil à laquelle on passe l'objet « charge utile » (*payload*) `Gpio` en paramètre (**listage 5**). La méthode `start()` permet de démarrer le fil.
Nous sommes maintenant prêts à transférer le programme sur le RPi. Comme l'*ADB* fonctionne comme une couche d'abstraction, il suffit d'un clic sur *Run*, le RPi se comporte alors comme un ordiphone connecté au PC par une liaison USB. Comme *MainActivity* n'inclut aucune interface utilisateur, l'écran connecté au RPi devient blanc, ce qui montre ainsi que notre *Activity* s'est mise joyeusement au travail.

Exploitation

Connectez maintenant un analyseur de domaine de modulation au RPi pour bénéficier de la copie d'écran de la **figure 8** (l'auteur propose en [6] une vidéo en anglais qui fournit des informations complémentaires sur le rôle et l'intérêt d'un analyseur de domaine de modulation).

Outre une instabilité occasionnelle, on remarque deux pics : la plage à 2,073 kHz montre les deux « passages linéaires » alors que l'exécution de la boucle `while` se traduit par une fréquence moindre de « seulement » 2,062 kHz.
Il est évident qu'Android ne peut en aucun cas concourir avec un Unix classique ; la machine virtuelle Java intermédiaire réclame son tribut. On trouvera sur l'internet divers articles sur le comportement en temps réel d'Android, par exemple en [7].

Conclusion

Le programme que nous venons de réaliser est un exemple classique de gaspillage de ressources dans le domaine de l'embarqué. Un programmeur raisonnable

obtiendra une forme d'onde stable sans système en temps réel, rien qu'avec une poignée de lignes de code et un PIC.

Il faut reconnaître que le « *bit banging* » (exécution d'un protocole série par logiciel) ne fait pas vraiment partie du domaine d'application favori d'*Android Things*. Pour que la plateforme puisse montrer sa vraie puissance, il faut des applications plus exigeantes. Dans le prochain article, nous vous montrerons comment exploiter des données de capteurs et les afficher sous forme de diagramme. D'ici là, je vous souhaite un bon codage ! ⏮

(160361 – version française : Helmut Müller)

Listage 5. Démarrage du fil de production du signal dans la *MainActivity*

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    . . .

    WaveformEmitter myEmitter=new WaveFormEmitter(myGPIO);
    new Thread( myEmitter ).start();
}
```

Liens

[1] https://developer.android.com/things/sdk/index.html
[2] https://developer.android.com/things/preview/download.html
[3] https://developer.android.com/things/hardware/raspberrypi.html
[4] https://developer.android.com/studio/install.html
[5] https://github.com/androidthings/new-project-template
[6] www.youtube.com/watch?v=lBLEfVUVGyU
[7] www.utdallas.edu/~cxl137330/courses/fall14/RTS/papers/4a.pdf

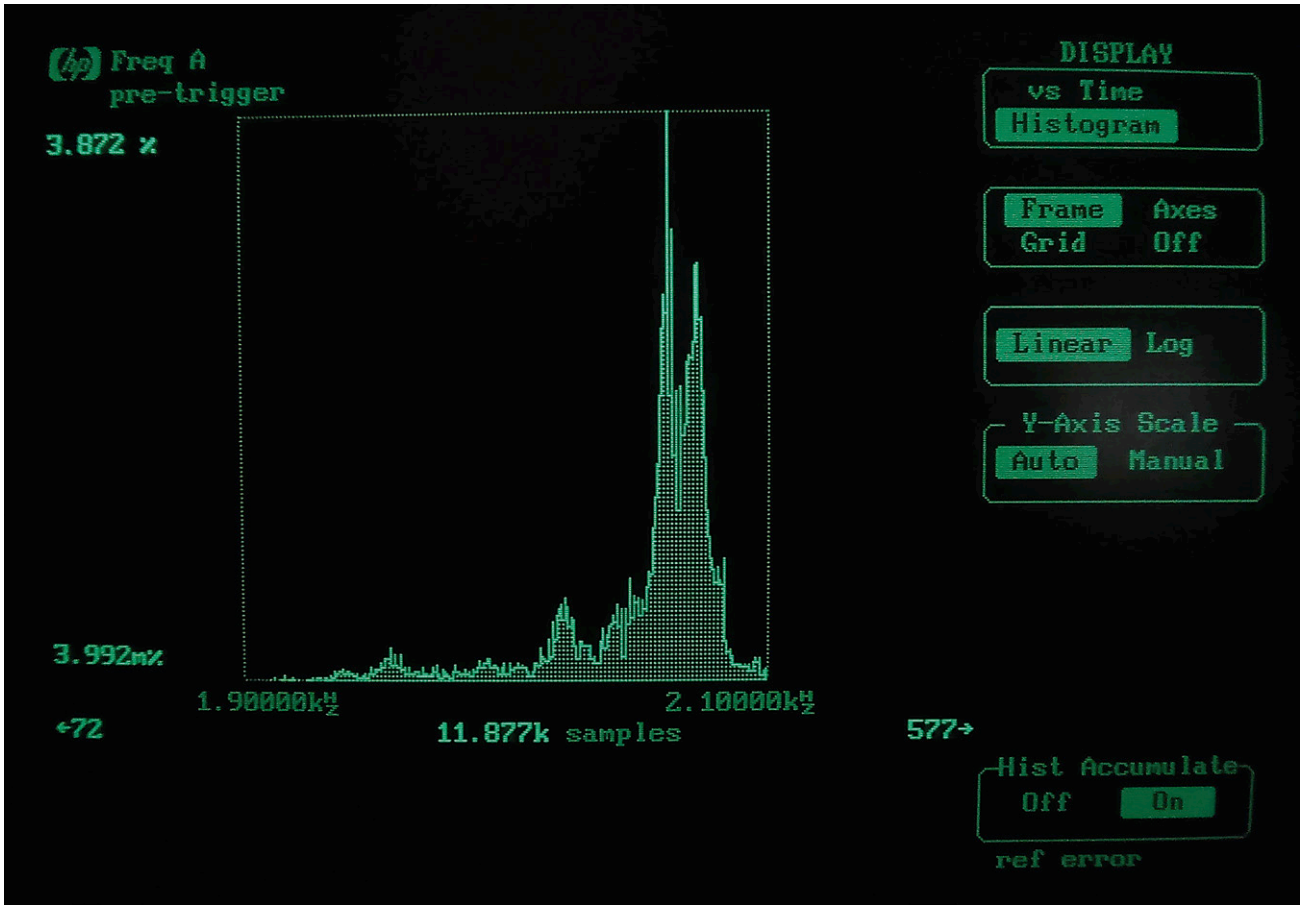
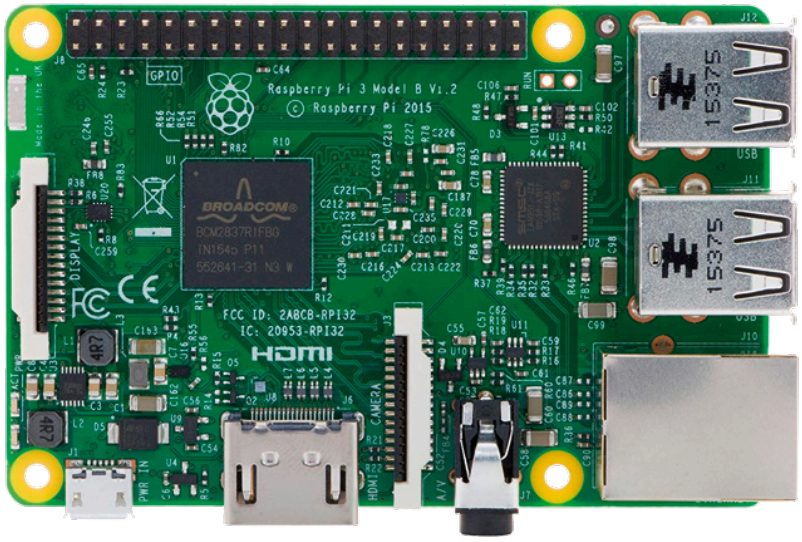


Figure 8. Les Arduinos produisent de bien plus beaux histogrammes.

Android sur Raspberry Pi (2)

afficheur piloté via le bus SPI



Dans l'article précédent, nous avons fait tourner un premier programme sous Android : une sortie numérique commutait entre les états haut et bas dans une boucle sans fin, ce qui nous a permis d'examiner son comportement en temps réel. Dans le cas d'une gestion de périphérique, la commande « manuelle » des broches n'a guère de sens; il faut passer par les interfaces matérielles du contrôleur. Nous montrons ici comment piloter un petit afficheur OLED via le bus SPI.

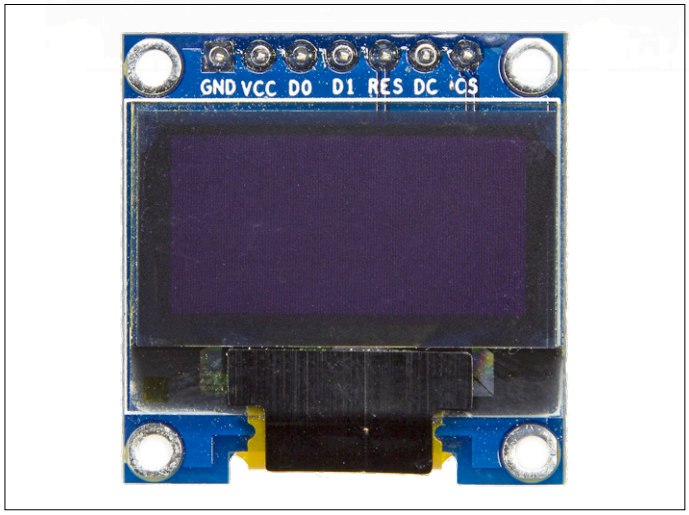


Figure 1. Cet afficheur est bien plus petit qu'un moniteur HD, aussi bien par sa taille que par sa consommation.

Tam Hanna (Slovaquie)

Si l'on veut gérer des périphériques avec Android Things, on a pour l'instant le choix entre SPI et I2C. Dans la suite, nous allons utiliser un afficheur OLED piloté par SPI, largement répandu et disponible chez AliExpress pour quelques dollars (**figure 1**).

Circuit

Un Raspberry Pi sous Android se comporte d'un point de vue matériel comme un Raspberry Pi ordinaire. Le circuit de la **figure 2** n'est donc guère différent de celui utilisé dans la série « Windows sur la carte RPi » [2].

Android Things identifie les bus SPI (de même que les broches GPIO) par des chaînes de caractères. Notre première tâche est d'identifier ceux du RPi qui sont à notre disposition :

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    . . .

    PeripheralManagerService manager = new
         PeripheralManagerService();
    List<String> deviceList = manager.getSpiBusList();

    if (deviceList.isEmpty()) { } else {
        Log.i(TAG, "List of available devices: " +
              deviceList);
    }
}
```

Le programme affiche dans la console de débogage une liste de tous les bus disponibles. Au moment de la publication, le RPi 3 de l'auteur fournit deux références :

```
I/MainActivity: List of available devices: [SPI0.0,
      SPI0.1]
```

Ici, il faut éviter un petit piège tendu par Gradle ou Android Studio : si vous voulez recueillir les données au niveau d'un point d'arrêt, il faut prendre garde de démarrer le programme en cliquant sur l'icône de débogage. Si vous cliquez sur la flèche du démarrage normal, le débogueur n'est pas lié au programme.

Dans un premier projet de démonstration (dont on peut comme d'habitude télécharger le code depuis le site web d'Elektor [3]), nous devons commencer par initialiser le bus SPI. Pour cela, faisons appel à une nouvelle méthode appelée `configureAndFindDevice`. En premier lieu, munissons la classe d'activité principale `MainActivity` d'un objet GPIO global qui contrôle la ligne de réinitialisation de l'afficheur. En complément, ajoutons un objet `SpiDevice` qui encapsule la communication entre le RPi et l'afficheur.

```
public class MainActivity extends Activity {

    SpiDevice myDevice;

    Gpio myPinReset;
```

L'étape suivante inclut l'initialisation – plus ou moins classique – du bus SPI à l'intérieur de `configureAndFindDevice` (**listage 1**). Un appel à `setFrequency` fixe la fréquence de communication : les valeurs de 4 MHz ou 8 MHz ont donné toute satisfaction à l'auteur.
Les lecteurs attentifs de l'article précédent se demandent sans doute ce que `Thread.Sleep` vient faire dans le fil GUI. En fait, le délai de 10 ms n'est pas gênant et ne justifie aucunement la mise en place d'un fil spécifique.

Nous incorporons l'appel de cette méthode à `onCreate` de sorte qu'elle soit exécutée au démarrage du programme (**listage 2**). Ici, nous initialisons une autre broche GPIO qui permet de commander la ligne DC (*Data Control*). De plus, nous appelons la fonction `initDisplay` responsable de l'initialisation de divers paramètres de l'afficheur – le contenu de cette méthode est présenté dans la section suivante.

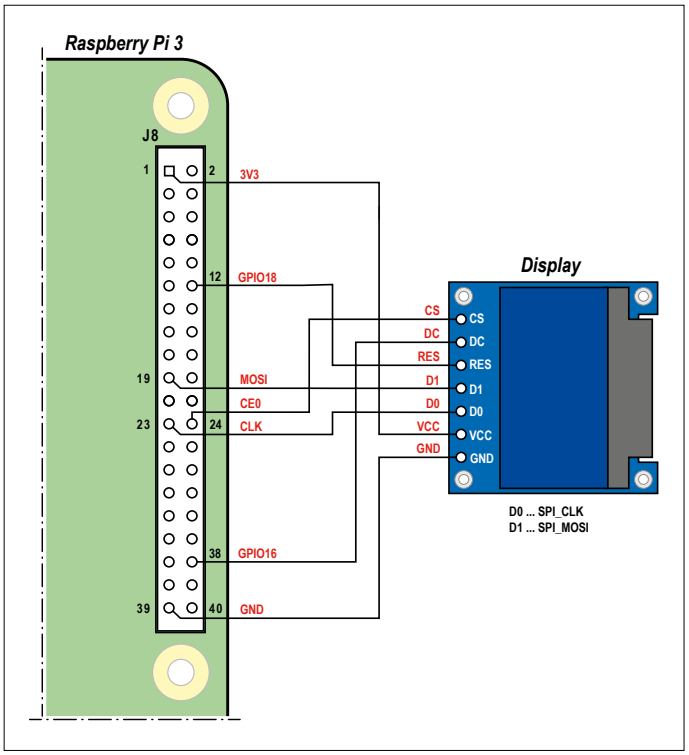


Figure 2. Il suffit de quelques fils pour raccorder le RPi à l'afficheur.

Pour être parfaitement complet du point de vue didactique, notez qu'il faut détruire l'objet `SPIdevice` en fin d'activité – c'est le rôle de la méthode `onDestroy` (**listage 3**).

Listage 1. Initialisation de l'interface SPI

```
void configureAndFindDevice(){

    PeripheralManagerService manager = new
                PeripheralManagerService();

    try {

        myPinReset = manager.openGpio("BCM18");
        myPinReset.setDirection(Gpio.
            DIRECTION_OUT_INITIALLY_LOW);
        myPinReset.setValue(false);

        myDevice = manager.openSpiDevice("SPI0.0");
        myDevice.setMode(SpiDevice.MODE0);
        myDevice.setFrequency(4000000);
        myDevice.setBitsPerWord(8);
        myDevice.setBitJustification(false);
        Thread.sleep(10);
        myPinReset.setValue(true);
    }

 catch (Exception e) {. . .}

}
```

Listage 2. Appelée au démarrage du programme, la méthode onCreate effectue les initialisations de toutes les broches.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    . . .

    configureAndFindDevice();

    try {

        PeripheralManagerService manager = new
                PeripheralManagerService();
        myPinDataCommand = manager.
                openGpio("BCM16");
        myPinDataCommand.setDirection(Gpio.
                DIRECTION_OUT_INITIALLY_LOW);

    }catch(Exception x){}

    initDisplay();

}
```

Si ces opérations de nettoyage font défaut, il peut se produire au débogage des exceptions du genre « android.os.Service SpecificException: BCM18 is already in use ».
Notons encore que des classes d'activité du système d'exploitation Android peuvent être supprimées en cours de fonctionnement. Il s'agit là d'une décision de conception des développeurs qui provoque des troubles en bien des lieux de l'univers Android.

Listage 3. À la fin du programme, on fait le ménage.

```
@Override

protected void onDestroy() {

    super.onDestroy();

    Log.d(TAG, "onDestroy");

    try {
        if(myDevice!=null) myDevice.close();
        if(myPinDataCommand!=null) myPinDataCommand.
              close();
        if(myPinReset!=null) myPinReset.close();
        myDevice = null;

    } catch (IOException e) {

        Log.w(TAG, "Unable to close SPI device", e);

    }

}
```

Nous initialisons…

Après la configuration réussie du *framework* SPI, nous pouvons nous occuper de la transmission des données qui mettront l'afficheur dans le mode correct ou seront affichées sur l'écran. Même si la feuille de caractéristiques du SSD1306 est très répandue, il est toujours bon, lors du premier emploi d'un circuit intégré, de rechercher chez Adafruit ou parmi la documentation d'autres fabricants, des bibliothèques complètes qui présentent le mode opératoire.

Le code d'`InitDisplay` est présenté dans le **listage 4**.

Pour commencer, on effectue un appel supplémentaire à `Thread.sleep` : cela garantit un temps de démarrage suffisant au contrôleur lorsqu'il se réveille après la réinitialisation. On envoie ensuite trois commandes qui démarrent la pompe de charge et mettent l'afficheur en marche. Avant l'envoi de la commande `sendData`, qui gère l'envoi de données à afficher, on s'assure que le tampon d'affichage contient des données plus ou moins aléatoires.

La déclaration des constantes des commandes prend une forme particulière, dont la plus notable est que chaque constante nécessite une conversion de format (`cast`) explicite vers un octet, car, en son absence, le compilateur Java créerait une arborescence :

```
private byte[] CMD_DISPLAY_ON = { (byte)0xAF };
```

Ensuite, nous pouvons utiliser les deux méthodes d'envoi des données vers le contrôleur de l'afficheur. Remarquons que pour la réception des données aussi bien que pour la collecte des commandes, le SD1306 se comporte comme un périphérique SPI. C'est le niveau de tension présent sur l'entrée numérique qui détermine le mode. Si le niveau est haut, les données vont

Listage 4. Affichage de lignes sur l'afficheur.

```
void initDisplay()

{

    try {

        Thread.sleep(100);
        SendCommand(CMD_CHARGEPUMP_ON);
        SendCommand(CMD_MEMADDRMODE);
        SendCommand(CMD_DISPLAY_ON);

        for(int i=0;i<myDisplayBuffer.length;i++){
            myDisplayBuffer[i]=22;
        }

        sendData();
    }

        catch(Exception x){Log.e("Elektor",x.
getMessage());}

}
```

dans le tampon d'affichage ; s'il est bas, elles sont stockées dans la mémoire de commande. Cela conduit au code suivant :

```
    void SendCommand(byte[] _what){

        try {
            myPinDataCommand.setValue(false);
            myDevice.write(_what, _what.length);
        }

        catch (Exception x){
            Log.e("Elektor",x.getMessage());
        }
    }
```

La version pour l'envoi de données d'affichage ne présente qu'un minimum de différences avec `SendCommand`. Au lieu de prendre comme données le code de commande, la fonction accède à un tableau défini comme membre (`myDisplayBuffer`).

```
    void sendData(){

        try {
            myPinDataCommand.setValue(true);
            myDevice.write(myDisplayBuffer,
                 myDisplayBuffer.length);
        }

        catch (Exception x){
            Log.e("Elektor",x.getMessage());
        }
    }
}
```

Et voilà une première version du programme déjà opérationnelle. Exécutez-la ; si tout est correct, une série de lignes horizontales apparaissent sur l'écran (**figure 3**).

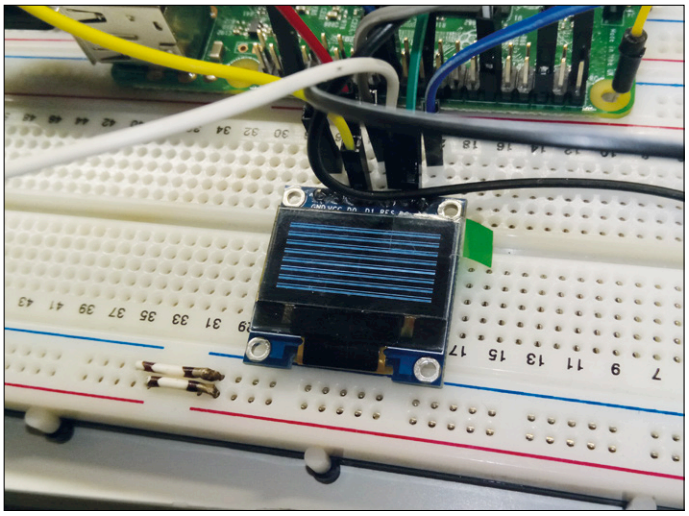


Figure 3. Le motif de test montre que tout fonctionne bien. Un afficheur OLED noir donne, par contre, l'impression d'être « mort ».

… et communiquons

Pour n'afficher qu'un motif constitué de lignes, nul besoin d'un système d'exploitation en temps réel dédié ; une mémoire d'affichage de 1 Ko sur un AVR fera parfaitement l'affaire. La puissance du RPi n'entrera en jeu que lorsque le système d'exploitation aura à afficher des informations complexes.

Une application serait de pouvoir afficher sur l'écran des éléments de commande GUI (comme un diagramme) sous la forme d'une matrice de points (*bitmap*). Pour cela, nous allons incorporer ***MPAndroidChart*** à notre programme – il s'agit d'une bibliothèque de diagrammes très puissante qui joue également un rôle dans le développement d'applications Android classiques. Commencez par ouvrir le fichier *build.gradle* du projet et complétez-le avec le code suivant :

```
allprojects {
    repositories {
        jcenter()
        maven { url "https://jitpack.io" }
    }
}
```

Il faut compléter le fichier *build.gradle (Module:app)* de l'appli de la manière suivante :

```
dependencies {
    provided 'com.google.android.
          things:androidthings:0.1-devpreview'
    compile 'com.github.PhilJay:MPAndroidChart:v3.0.1'
}
```

Après l'enregistrement de ces deux fichiers, Android Studio affiche un bandeau jaune dans lequel il faut cliquer sur *Sync Now*, ce qui demande à l'EDI d'actualiser l'ensemble du projet. L'une des plus grandes forces de Gradle est sa capacité de télécharger sur l'internet les bibliothèques et autres informations nécessaires au cours de la compilation.

Ensuite, faites un clic droit sur le dossier du projet et choisissez l'option *New → File → Android Resource File*. Dans la fenêtre de dialogue qui s'affiche, saisissez les paramètres tels que représentés sur la **figure 4** pour créer un fichier de mise en page (*layout*).

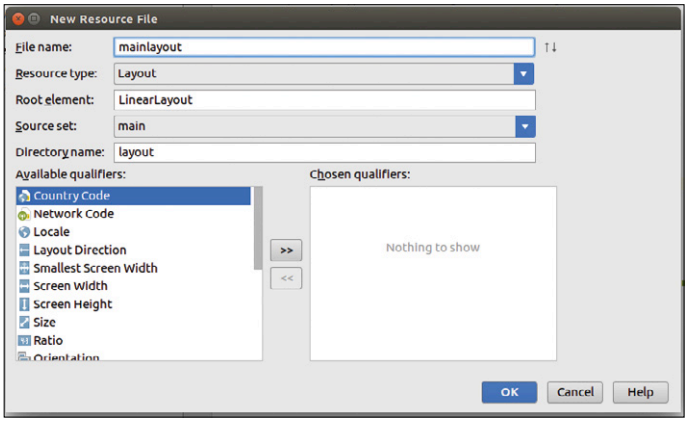


Figure 4. Ces paramètres servent à créer un nouveau fichier de mise en page.

Par défaut, le sous-système de mise en page d'Android produit un élément de commande affichable dont la taille est ajustée dynamiquement, alors que nous voulons un *widget* d'une taille fixe de 128 × 64 pixels. Il nous faut donc modifier le code de la manière suivante :

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=http://schemas.android.com/
            apk/res/android
    android:orientation="vertical"
            android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.github.mikephil.charting.charts.LineChart
        android:id="@+id/chart1"
        android:layout_width="128px"
        android:layout_height="64px"/>

</LinearLayout>
```

Le détour par l'écran principal du RPi utilisé ici n'est pas absolument nécessaire, mais il facilite le travail, car il permet d'observer l'affichage produit sur un moniteur Android Studio connecté.

Nous pouvons maintenant nous occuper de la bibliothèque graphique, qui a besoin d'une directive pour l'affichage de diagrammes. Celle-ci prend la forme suivante :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.mainlayout);
```

La première différence avec une appli Android Things habituelle est l'appel de `setContentView`, qui s'occupe du chargement des informations contenues dans le fichier XML de mise en page. Ensuite nous déclarons un tableau rempli de données d'affichage plus ou moins aléatoires :

```
    ArrayList<Entry> values = new ArrayList<Entry>();

    for (int i = 0; i < 20; i++) {
        float val = (float) (Math.random() * 25) + 3;
        values.add(new Entry(i, val));
    }
```

Nous en arrivons au paramétrage : la bibliothèque *MPAndroidChart* est conçue pour le traitement de vastes ensembles de données, ce qui rend son initialisation relativement complexe, dont voici seulement quelques extraits :

```
    LineChart mChart = (LineChart) findViewById(R.
         id.chart1);
    mChart.getDescription().setEnabled(false);
    LineDataSet s1=new LineDataSet(values,"");

    . . .

    ArrayList<ILineDataSet> dataSets = new
          ArrayList<ILineDataSet>();
    dataSets.add(s1);
    LineData data = new LineData(dataSets);
    mChart.setData(data);
```

Le premier problème, à l'origine de la taille énorme du code, est l'élimination des commandes pour les caractères (inutiles ici) : la résolution de l'afficheur de 128 × 64 pixels est tellement petite qu'elle n'a sans doute jamais été testée par les développeurs de *MPAndroidChart*.

Le problème suivant est de convertir le résultat en une matrice de points qui peut être envoyée à l'afficheur. Ici, il importe surtout d'éliminer l'information de couleur, notre afficheur ne gère bien entendu que le noir et blanc. À cause de la procédure assez particulière appliquée aux caractères par *MPAndroidChart*, nous devons retarder la conversion ; c'est possible avec un pilote, voici le squelette de son code :

```
final Handler handler = new Handler();
handler.postDelayed(new Runnable() {

    @Override
    public void run() {

        //Code ist hier

    }
}, 700);
```

`postDelayed` accepte un objet qui doit comporter une méthode `run()` qui contient elle-même le code à exécuter après écoulement du délai. C'est assez compliqué – commençons par la conversion du widget en une instance de la classe `Bitmap` :

```
    public void run() {
        LineChart mChart = (LineChart) findViewById(R.
                id.chart1);

        Bitmap returnedBitmap = Bitmap.createBitmap(128,
                 64,Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(returnedBitmap);

        Drawable bgDrawable =mChart.getBackground();
        if (bgDrawable!=null)
            bgDrawable.draw(canvas);
        else
            canvas.drawColor(Color.WHITE);

        mChart.draw(canvas)
```

À ce moment, `returnedBitmap` contient une version couleur du diagramme produit par *MPAndroidChart*. Nous devons donc convertir cette matrice de points (*bitmap*) en un tampon de trame (*frame buffer*). Et voici notre première cause d'irritation : la structure de la mémoire de l'écran ; elle est divisée en un groupe de pages !

La solution la plus simple à ce genre de problème est de traiter les octets de mémoire l'un après l'autre dans des boucles `for`. Dans notre cas, il nous faut trois boucles :

```
        byte counter=0;
```

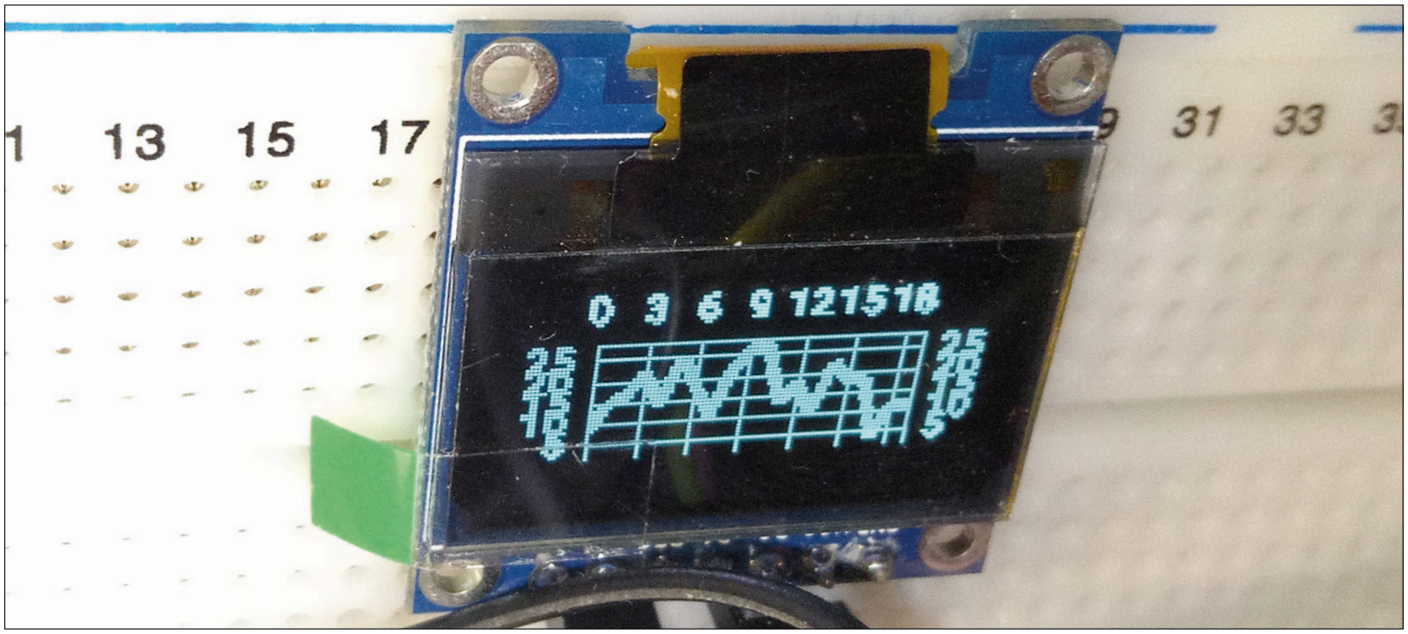


Figure 5. Grâce à Android, on peut afficher sans gros efforts un sympathique diagramme linéaire sur une grille.

```
byte store=0;
int counter2=0;
for(int ctr=0;ctr<8;ctr++)
    for(int x=0;x<returnedBitmap.getWidth();x++)
{
        for (int y = ctr*8; y < (ctr+1)*8; y++)
{
```

L'étape suivante consiste à rassembler huit pixels dans un octet. Pour cela, nous commençons par acquérir l'information de couleur de chaque pixel pour ensuite la convertir en blanc ou noir par une évaluation simple de ses composantes :

```
int myVal=returnedBitmap.getPixel(x,y);
if(Color.red(myVal)!=255 || Color.
   green(myVal)!=255 || Color.blue(myVal)!=255)
        { //Set
      store=(byte)(store | ((byte)(1 <<
           (counter))));
}
```

Le remplissage d'un octet se fait donc en huit pas. Les informations sont rangées une à une dans `store` au moyen d'instructions de manipulation de bits ; dès qu'un octet est plein, il est copié dans `myDisplayBuffer`.

```
            counter++;
            if(counter==8) {
                myDisplayBuffer[counter2]=store;
                store=0;
                counter=0;
                //Byte raus
                counter2++;
            }
        }
    }
```

Pour finir, nous envoyons quelques commandes au contrôleur de l'afficheur pour le mettre dans un état bien défini. `sendData` envoie alors les informations à l'afficheur via le bus SPI :

```
        SendCommand(CMD_RESETCOLADDR);
        SendCommand(CMD_RESETPAGEADDR);

        sendData();

    }
```

Le diagramme de la **figure 5** montre le fruit de nos efforts.

Conclusion

Nous voici arrivés à la fin de nos expérimentations avec Android Things. Ce n'est certes pas une plateforme pour tout le monde, mais si vous arrivez à en tirer quelque chose, vous aurez à votre service des fonctions nombreuses et puissantes. Il y a beaucoup d'applications qui ne sont pas ou seulement très difficilement à la portée des contrôleurs à 8 bits, alors mettre à la tâche un puissant processeur à 32 bits permet de gagner beaucoup de temps et d'effort. Nous vous souhaitons beaucoup de succès dans ce domaine !

(160369 – version française : Helmut Müller)

Liens

[1] www.elektormagazine.fr/160361
[2] www.elektormagazine.fr/150520
[3] www.elektormagazine.fr/160369

bienvenue dans votre e-choppe

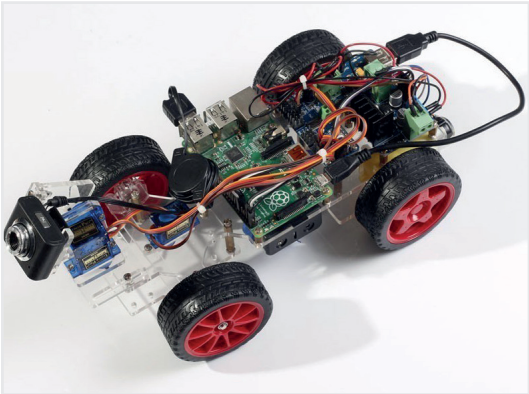
Raspberry Pi
APPROVED RESELLER

la rédaction recommande

Le kit « Smart Video Car for Raspberry Pi » de Sunfounder est idéal comme outil de développement de logiciel à base de RPi. Les pièces du kit sont bien usinées, le manuel est très bon et le logiciel fonctionne. Le véhicule est télécommandé par Wi-Fi et il exécute d'emblée (après montage tout de même) les fonctions *move forward* (en avant), *move backward* (en arrière), *go left* (à gauche), et *go right* (à droite) et, pour orienter la caméra *pan* (horizontalement) et *tilt* (verticalement).

La télécommande est un ordinateur (Windows, Linux, Mac, autres) sur lequel Python 2.7 et un navigateur web sont installés (et ça va de soi, un accès Wi-Fi). Il existe aussi une appli Android.

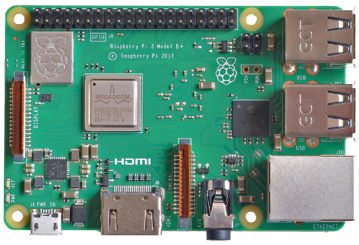
Clemens Valens
Labo d'Elektor



Lisez le banc d'essai sur www.elektor.fr/smart-video-car-banc-dessai

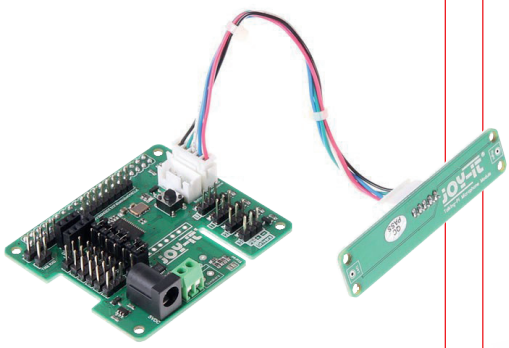
Notre sélection :

1. Raspberry Pi 3 (modèle B+)
   www.elektor.fr/rpi3b-plus



2. Livre (en anglais) « Raspberry Pi 3 Basic and Advanced Projects »
   www.elektor.fr/livre-rpi3
3. Kit de démarrage (de luxe) Raspberry Pi 3 B+
   www.elektor.fr/rpi-starterkit
4. pi-top 2 – ordinateur portable modulaire RPi
   www.elektor.fr/pi-top2
5. Module WI-FI ESP-01 8266
   www.elektor.fr/150445-91
6. Raspberry Pi Zero WH
   www.elektor.com/rpi-zero-wh
7. Écran tactile de 10" pour RPi (HDMI/VGA/BNC/AV)
   www.elektor.fr/td-rpi

Talking Pi

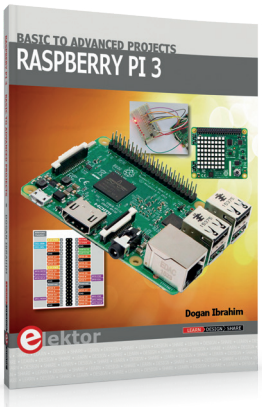


Talking Pi est un nouvel assistant intelligent et universel pour commande vocale avec un Raspberry Pi. Le code source de ce module d'extension compatible avec le projet Google Home / AIY est ouvert. Quand vous entrez ou sortez d'une pièce, un circuit de commande vocale permet p.ex. d'allumer ou d'éteindre les lumières, de commander des prises, d'activer la machine à café ! À vous d'imaginer d'autres applications, il n'y a pas de limites.

Prix (membres) : 31.46 €

www.elektor.fr/talking-pi

Raspberry Pi 3 Basic and Advanced Projects



Ce livre (en anglais) traite du Raspberry Pi 3, mais surtout de son utilisation dans diverses applications de commande et de surveillance. Les montages proposés sont intéressants parce qu'ils reposent sur du matériel récent : Sense HAT, Swiss Pi, MotoPi, module Camera, et beaucoup d'autres capteurs analogiques et numériques de pointe.

Prix (membres) : 31,46 €

www.elektor.fr/livre-rpi3

Boîtier Pi Desktop

Le boîtier Pi Desktop est bien plus qu'une enceinte pour loger le RPi. Cet écrin pour le Raspberry Pi et ses bijoux comporte entre autres une carte d'extension avec microcontrôleur STM8S00 et connecteur GPIO par lequel les broches d'E/S du RPi restent accessibles. Le Pi Desktop transformera votre RPi 2 ou 3 en véritable ordinateur avec jusqu'à 1 To de mémoire (connecteur mSATA pour disque dur) et un bouton marche-arrêt géré par logiciel.

Prix (membres): 53.96 €

www.elektor.fr/rpi-desktop-case

Raspberry Pi 3B+ avec kit de démarrage (de luxe !)

Ce superbe assortiment d'accessoires du Raspberry Pi 3B+ réunit pour vous tout ce qu'il faut pour démarrer en beauté avec le plus populaire des mini-ordinateurs .

Contenu du kit :

- 1x Raspberry Pi 3B+ avec radiateur installé
- 1x boîtier officiel pour Raspberry Pi (noir)
- 1x alimentation micro-USB officielle pour Raspberry Pi (5,1 V | 2,5 A) – noire
- 1x câble HDMI haut débit (1 m)
- 1x câble réseau Cat. 5e (2 m)
- 1x carte microSD (16 Go, Class 10) avec adaptateur SD (NOOBS installé)

Prix (membres) : 80,96 €

www.elektor.fr/rpi-starterkit

pi-top 2 (ordinateur portable modulaire RPi)

Grâce à pi-top 2, votre Raspberry Pi se transforme en ordinateur portable avec un écran full HD de 35,6 cm (14 pouces) ! Rien à souder, rien à programmer : en quelques minutes, vous avez devant vous un puissant portable de facture moderne, avec clavier et pave tactile.

Prix (membres) : 269.96 €

www.elektor.fr/pi-top2

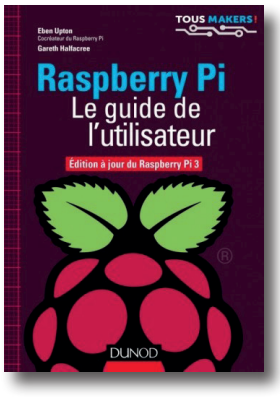
Raspberry Pi 45 applications utiles à l'électronicien

Ce livre allie programmation et électronique. Il explique comment réaliser des projets plaisants : régulation de température, commande de moteurs électriques, traitement de signaux analogiques, luxmètre, régulation de vitesse de moteur, serveur ouèbe avec CGI, applications client-serveur... Chaque projet est accompagné du pourquoi de la solution choisie. En les réalisant, vous apprenez beaucoup sur le Raspberry Pi, Python et les composants utilisés .

Prix (membres) : 37,50 €

www.elektor.fr/rpi

Raspberry Pi Le Guide de l'utilisateur



Ce manuel d'utilisation écrit par le créateur du Raspberry Pi lui-même s'adresse à tous ceux qui souhaitent tirer le meilleur parti de cette carte. Il permet de comprendre le matériel et ses principes de fonctionnement (installation, confi guration) et d'apprendre les bases de deux langages de programmation simples, Scratch et Python.

Prix (membres) : 24,21 €

www.elektor.fr/guide-rpi

le bus I²C

1re partie : le protocole

Josef Möllers

Le bus *Inter-Integrated Circuit*, en abrégé bus I²C ou encore TWI (*Two Wire Interface*) chez certains fabricants, permet de relier facilement deux puces à l'aide de deux fils quand il y a peu de données à transmettre et à petite vitesse. En trois épisodes, nous examinerons ses caractéristiques essentielles et les moyens de le mettre en œuvre avec Raspberry Pi, Arduino et d'autres systèmes.

Il n'est pas nécessaire d'aller dans les détails techniques pour découvrir à quel point ce bus I²C est d'emploi facile. Nous examinerons d'abord le protocole de transmission des données, puis nous ferons des essais de communication avec différents systèmes, depuis Arduino et RPi jusqu'au PC lui-même et finalement, nous irons à la rencontre de quelques puces I²C spécialisées et verrons comment lever une panne de transfert d'information.

C'est Philips Semiconductors qui a conçu le bus I²C dans les années 1980. Si vous ne tenez pas à aborder le bus I²C comme une boîte noire associée à des bibliothèques de logiciel tout fait, mais que vous voulez tout savoir jusqu'au niveau du bit, vous trouverez la spécification officielle complète de NXP (qui a repris Philips Semiconductors) sur [1] et dont la lecture est évidemment recommandée.

Il y a sur l'internet une infinité de ressources sur le thème du bus I²C. Citons par exemple le site en anglais [2] de la société hambourgeoise *telos Systementwicklung GmbH* où l'on trouve à peu près tous les aspects de la communication sur ce bus.

Petit détour par la couche physique

En général, un microcontrôleur programmable commande des blocs périphériques à fonction précise, comme extension de port, horloge ou EEPROM. Le contrôleur est le maître, il pour-

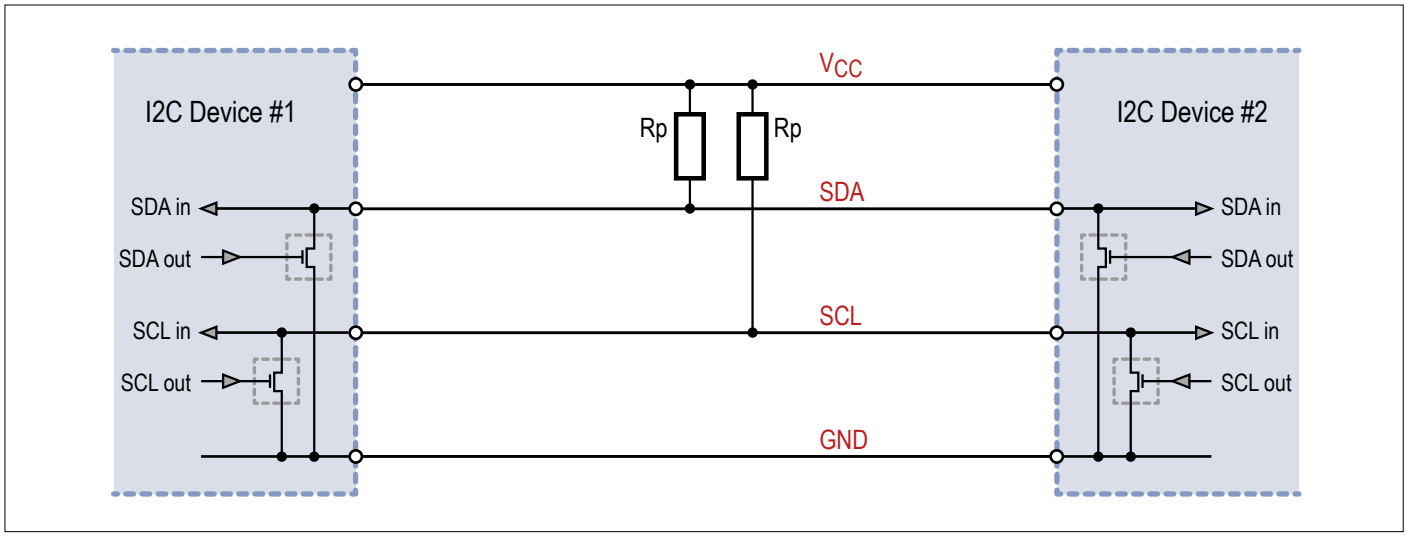


Figure 1. Le bus I²C constitue une porte ET câblée.

voit au signal d'horloge, tandis que les puces périphériques sont les esclaves. Par I²C, on peut aussi faire converser deux microcontrôleurs, si l'un occupe la fonction de maître. Quand il faut commander plusieurs puces esclaves, tous les blocs, en rang d'oignons, partagent les mêmes lignes de bus. Mais on peut aussi connecter plusieurs maîtres au même bus I²C. L'une des deux lignes du bus véhicule le signal d'horloge SCL, l'autre le signal de données SDA. Un câble plat à quatre conducteurs suffit ainsi pour commander les périphériques et les alimenter. Pour limiter la diaphonie, mieux vaut utiliser pour la masse le fil situé entre ceux des deux signaux.

Les deux lignes adhèrent à la logique positive : une tension élevée sur SDA représente un 1 et pour une impulsion d'horloge, la ligne SCL est aussi placée brièvement au niveau haut. Au repos, les résistances de polarisation haute maintiennent les deux lignes au niveau haut. Les tampons dans les puces I²C sont en configuration à collecteur ouvert ou à drain ouvert de manière à ce que les deux lignes constituent toujours une grande porte ET câblée (**figure 1**). Une ligne ne peut être haute par rapport au point commun que si **toutes** les sorties sont au niveau haut. Si l'une des sorties passe au niveau bas, la ligne commune est également au niveau bas. Chaque membre du bus doit donc vérifier le niveau des lignes : s'il veut y mettre un 1, il doit être en mesure d'exclure la possibilité qu'un autre membre y place un 0 en même temps.
La grandeur de la résistance est peu critique, elle devrait se situer dans la plage de quelques milliers d'ohms. Sur RPi, les résistances de polarisation (1,8 kΩ vers le 3,3 V) sont déjà installées à demeure. Sur les microcontrôleurs Atmel de la série ATmega, ces résistances sont commutées par logiciel, il ne faut donc pas en ajouter, puisqu'elles valent déjà entre 20 et 50 kΩ. Selon la spécification, le niveau haut doit atteindre au moins 0,7 V et le niveau bas être inférieur à 0,3 V ; en pratique, les puces de 5 V fonctionnent aussi avec une résistance vers le 3,3 V, par exemple sur un RPi. Si d'aventure un circuit ne fonctionnait pas, cela peut provenir de différences de niveau, il faut alors installer un convertisseur de niveau.
Il existe aussi des modules esclaves à résistances de polarisation intégrées. Il faut les enlever ou ne pas les monter s'il s'agit d'un kit, parce que quand plusieurs esclaves mettent ensemble leurs résistances, elles se retrouvent en parallèle et la résistance résultante est alors trop petite. Sur un bus I²C, il ne faut qu'**une** paire de résistances de polarisation, de préférence du côté du maître.

Il faut aussi faire attention quand on mélange des blocs de tensions différentes. Certains de ces composants alimentés sous 3,3 V ne supportent pas une tension supérieure à l'entrée. Relier les résistances de polarisation au +5 V risque de détériorer la puce. Pour que le bus travaille en toute sécurité, il y a lieu de s'en tenir aux niveaux prescrits. Pour assembler sur le même bus des composants de 5 V et de 3,3 V, des convertisseurs de niveau bidirectionnels sont nécessaires, comme ceux décrits sur [3]. Deux MOSFET et les résistances nécessaires relient alors la partie sous 5 V à celle sous 3,3 V sur le bus (**figure 2**).

Le protocole

Le bus I²C est un simple bus maître/esclave sur lequel la communication démarre toujours d'un maître qui s'adresse à un esclave avec lequel il échangera des données. Un esclave ne peut pas entreprendre lui-même une conversation comme il est possible de le faire en SCSI, par exemple. Mais un microcontrôleur peut aussi bien jouer le rôle de maître que d'esclave. En théorie, il serait possible d'inverser les rôles en cours d'exécution, mais d'habitude, un membre du bus est soit l'un, soit l'autre.

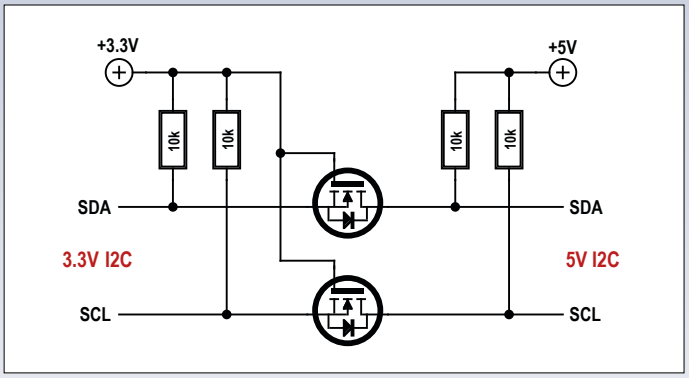


Figure 2. Des MOSFET compartimentent le bus entre la partie sous 5 V et celle sous 3,3 V.

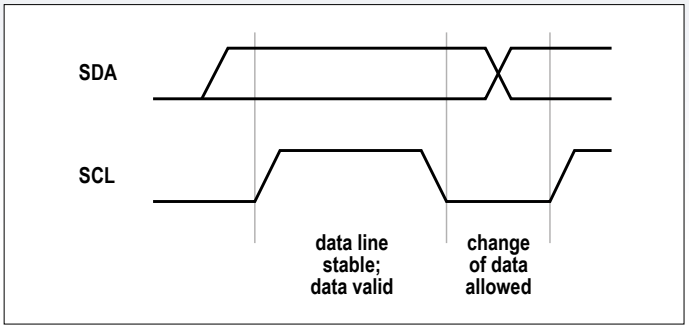


Figure 3. Diagramme temporel d'un transfert de données.

C'est toujours le maître qui décide de la vitesse de transmission. Il y a officiellement quatre débits binaires possibles :

- 100 Kb/s (mode standard, Sm, par défaut),
- 400 Kb/s (mode rapide, *Fast mode*, Fm),
- 1 Mb/s (mode plus rapide, Fm+) et
- 3,4 Mb/s (mode à haute vitesse *High Speed*, Hs).

Le maître peut transmettre des données à l'esclave, et inversement, mais c'est toujours le maître qui délivre le signal d'horloge sur la ligne SCL. Le transfert de données (**figure 3**) se déroule donc synchronisé par l'horloge. Pour envoyer un bit, l'émetteur met le bit de donnée sur la ligne SDA, en logique positive. Le maître relâche la ligne d'horloge SCL qui est tirée au niveau haut et après un certain temps la ramène au niveau bas. C'est ainsi qu'un bit est transmis et l'émetteur peut envoyer le bit suivant. Pendant la transmission, aussi longtemps que la ligne SCL reste haute, la ligne SDA ne peut pas varier.

1. Pour lancer la transmission, le maître abaisse d'abord la ligne de données SDA, puis celle d'horloge SCL (**figure 4**). Cette séquence s'appelle la condition de départ, *START condition*.

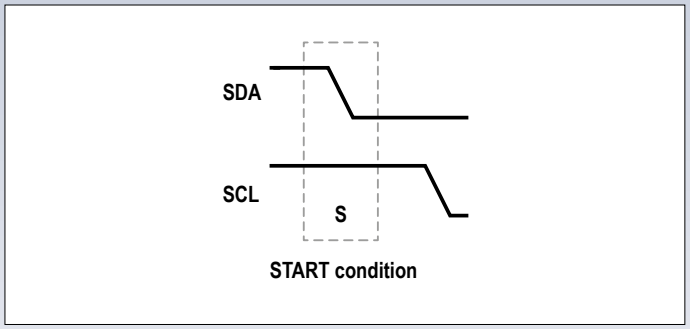


Figure 4. Pour démarrer une communication, il faut remplir la condition de *START*.

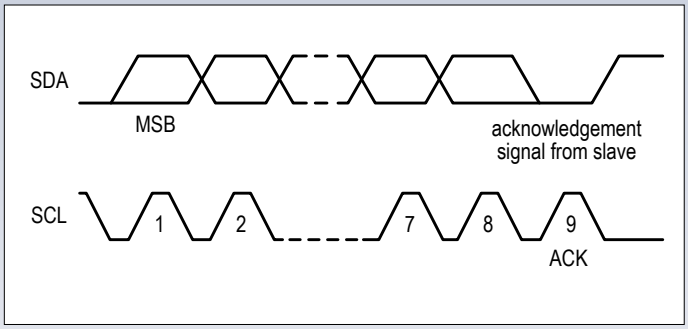


Figure 5. La manière de passer les bits d'adresse.

2. Après quoi, le maître, comme l'indique la **figure 5**, envoie d'abord l'adresse de l'esclave, bit de poids fort en tête (MSB), suivie du bit de lecture/écriture (L/E) (0 = donnée de l'esclave au maître ou 1 = donnée du maître à l'esclave). L'adresse d'esclave compte généralement sept bits (le standard I²C permet aussi des adresses plus longues sur 10 bits, mais c'est rarement utilisé). Le fabricant des blocs périphériques fournit lui-même l'adresse, en partie du moins. La plupart du temps, le fabricant laisse à disposition quelques bits que l'on peut définir soi-même par câblage extérieur de manière à associer sur le bus plusieurs blocs identiques en leur conférant des adresses différentes. Les adresses des composants périphériques sont indiquées dans les fiches techniques, mais on peut les obtenir avec un outil tel que `i2cdetect`. Au total, l'adresse d'esclave et le bit de L/E font 8 bits, donc un octet.

3. Après l'envoi de l'adresse, le maître transmet à coups d'horloge SCL les données utiles, toujours sous forme d'octets avec le bit de poids fort MSB en tête (**figure 6**). Si le bit de L/E dans l'octet d'adresse était un 0, le maître attend que l'esclave mette au bon moment les bits de donnée sur SDA pour produire lui-même les impulsions en mesure sur SCL. En revanche, si ce fameux bit était un 1, c'est au tour du maître de passer les bits de donnée sur SDA et toujours de produire les coups d'horloge. Pour chaque octet, y compris ceux d'adresse avec l'indicateur de direction L/E, chaque destinataire doit envoyer un neuvième bit comme accusé de réception. Un niveau bas pour une bonne arrivée (ACK), un niveau haut pour un déni de réception (NACK). Un NACK ne signifie d'ailleurs pas que la transmission a raté ou que les données étaient fausses, mais seulement que c'est la fin de la transmission. Quand il s'agit de l'octet d'adresse, un NACK signale qu'aucun esclave n'a reconnu l'adresse ou que la combinaison de l'adresse et du bit de L/E n'était pas valide.

4. À la fin de la transmission, le maître libère d'abord la ligne d'horloge SCL, puis la ligne de données SDA et donc les deux lignes retournent au niveau haut à cause de résistances de polarisation. Cette séquence (**figure 7**) s'appelle *STOP condition*. Le cas échéant, le maître peut abaisser en premier la ligne de données SDA.

Les conditions *START* et *STOP* sont des exceptions à la règle qui veut que la ligne SDA ne puisse pas varier tant que la ligne SCL est au niveau haut.

Le nombre d'octets qu'une communication peut transmettre est en fait illimité. En revanche, des protocoles plus élaborés, comme *Power Management Bus* (PMBus) ou *System Management Bus* (SMBus), qui sont basés sur le bus I²C, définissent des limites supérieures ou des formats d'enregistrement, qui contiennent par exemple la longueur de la transmission. En fait, la transmission « sans frontière », n'importe quelle plateforme ou bibliothèque ne la permet pas, parce que généralement, il faut préciser dès le début de la transmission le nombre d'octets à recevoir. Par exemple, la bibliothèque *Wire* d'Arduino définit dans ce protocole un temps limite, par exemple pour la mise en pause de l'horloge (*clock stretching*, cf. plus loin) ou la détection d'erreur par contrôle de redondance (*Packet Error Checking, PEC*).

Une communication (entre *START* et *STOP*) est toujours unidirectionnelle. Le maître fixe le sens de la communication avec le bit de L/E pour toute sa durée.

Seules quelques puces ont une structure si élémentaire qu'elles ne peuvent communiquer que d'une seule manière, par exemple le PCF8574 (que nous utiliserons dans la dernière partie de cette série). Avec la plupart des composants I²C, le maître doit, avant de pouvoir lire, effectuer une opération d'écriture pour envoyer à l'esclave un paramètre tel qu'un numéro de registre ou une adresse mémoire. S'il veut ensuite envoyer des données à l'esclave, il peut le faire en effectuant une écriture à l'endroit désigné par le paramètre. Si le maître veut lire des données issues de l'esclave, il doit lancer une nouvelle opération de lecture. Pour éviter qu'un autre maître s'immisce dans la conversation en envoyant au même esclave un autre paramètre, le maître actif peut remplacer la condition de *STOP* par une nouvelle condition de *START*, ce qui s'appelle alors une condition de *REPEATED START* :

```
Start – Adresse+Write – Registernumber – (Repeated)
   Start – Adresse+Read – Data– … – Stop
```

De nombreux blocs périphériques, comme des mémoires ou le RV-8523 de RTC, incrémentent automatiquement le numéro de registre ou l'adresse mémoire après chaque accès, d'autres, comme le capteur de température LM75, ne le font pas. Avec les

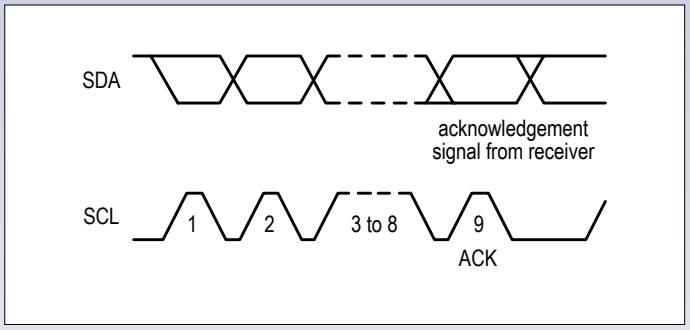


Figure 6. Le transfert des bits de données.

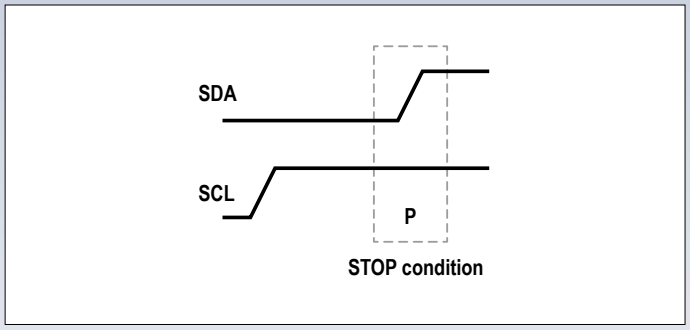


Figure 7. La clef pour clore la communication : *STOP*.

premiers, on peut dans une seule communication transmettre tout le banc de registres ou de mémoire, avec les autres, il faut répéter chaque fois l'adresse. D'un autre côté, quand on veut tout le temps lire le même registre, par exemple le registre de température du LM75, on peut se passer de répéter le numéro de registre. Les détails sur le sujet sont dans la fiche technique. Le protocole I²C ne prévoit pas les interruptions. Bien sûr, un esclave pourrait déclencher une interruption chez le maître par une ligne séparée. Après quoi, celui-ci appelle l'esclave par I²C, mais le protocole du bus ne dispose d'aucune norme dans ce but.

Mise en pause de SCL et arbitrage

Quand un esclave doit transmettre des données, il peut s'écouler un certain temps avant que la puce ne soit prête à le faire. L'esclave doit alors pendant ce délai arrêter ou différer l'horloge du maître. Comme les deux lignes de signal du bus forment une grande porte ET, l'esclave en a le pouvoir en passant momentanément la ligne SCL au niveau bas. Le maître s'en rend compte puisque SCL ne remonte pas quand il libère la ligne et il attend. Ce mécanisme s'appelle *Clock Stretching*, étirement du coup d'horloge, mais à proprement parler, ce n'est pas vrai, puisque l'esclave ne peut qu'allonger la pause entre deux impulsions d'horloge, pas l'impulsion elle-même.

Il est vrai qu'on peut raccorder plusieurs maîtres sur un bus I²C. Aussi, entre les conditions *START* et *STOP*, le bus affiche complet, aucun autre maître ne peut émettre de condition *START* pendant ce temps-là. Un maître (potentiel) doit donc continuellement tenir à l'œil le bus. Un maître ne peut émettre de *START* que quand le bus est libre. Imaginons que deux maîtres veuillent lancer un *START* au même instant. Il y a fort à parier que les deux adresses d'esclaves ou, au bout du compte, les bits de L/E seront différents et alors, un maître aura voulu transmettre un 1 et l'autre un 0. À cause du ET câblé, celui qui veut envoyer un 0 le verra sur le bus, tandis que l'autre, avec son 1, ne l'apercevra pas et devra cesser immédiatement la transmission. C'est donc celui qui a présenté un 0 qui est vainqueur, il ne remarque rien, acquiert tout de suite le contrôle du bus et poursuit sa transmission.

On trouve parfois à ce sujet le terme d'arbitrage, même dans de la documentation officielle. C'est inapproprié parce qu'il n'y a pas le moindre déphasage ou quoi que ce soit de comparable à des priorités ou des délais d'attente. Celui qui veut transmettre attend simplement que le bus se libère pour commencer, tout en surveillant ce qui se passe sur le bus. On ne trouve rien d'autre dans la documentation de NXP.

Tout maître peut d'ailleurs à tout moment interrompre une communication avec une condition *de STOP*. Un récepteur aussi peut le faire d'un simple NACK dans l'octet d'accusé de réception, mais jamais un esclave comme émetteur. Même si l'esclave, de son côté, a cessé d'envoyer les données dont il dispose, le maître, imperturbable, continue à lire ce qui se trouve sur SDA, éventuellement sans signification.

Dans la deuxième partie

La prochaine fois, nous établirons des communications avec différents microcontrôleurs, comme ATmega et Arduino, mais aussi Raspberry Pi et un PC, utilisés comme maître I²C et esclaves. Des exemples de programmes en C et Python montreront comment s'adresser à un membre du bus, lui envoyer des commandes et recevoir ses réponses. ⏮

(160095 – version française : Robert Grignard)

Liens

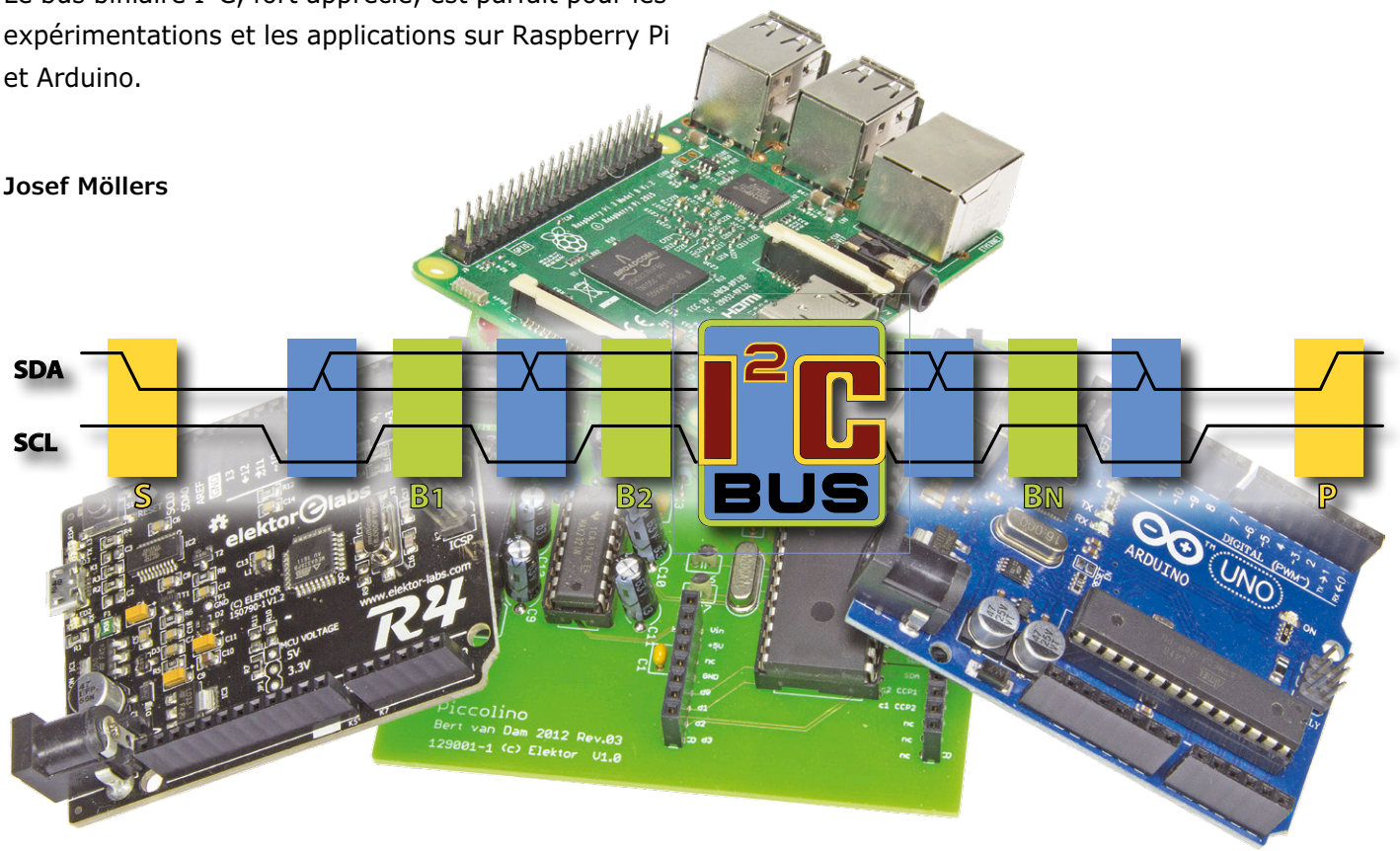
[1] www.nxp.com/documents/user_manual/UM10204.pdf
[2] www.i2c-bus.org/
[3] http://playground.arduino.cc/Main/I2CBi-directionalLevelShifter

le bus I²C

2e partie : le bus en action

Le bus bifilaire I²C, fort apprécié, est parfait pour les expérimentations et les applications sur Raspberry Pi et Arduino.

Josef Möllers



Raspberry Pi, BeagleBone, Arduino, Genuino, ATmega, PIC, presque tous les PC : à peu près tout ce qu'on trouve aujourd'hui comme outil de calcul sur la paillasse des labos ou des faiseurs est équipé d'un ou plusieurs ports I²C. À l'aide d'un capteur de température LM75, nous décrivons ici l'utilisation des interfaces I²C du Raspberry Pi, de l'ATmega et de l'Arduino en mode maître et, quand c'est possible, en mode esclave.

Raspberry Pi

Le Raspberry Pi dispose de deux bus I²C physiques, mais habituellement un seul est utilisable directement. Les résistances de rappel vers 3,3 V sont déjà présentes et actives en permanence sur le RPi. Les signaux SDA et SCL sont présents sur les broches 3 (SDA) et 5 (SCL) du connecteur GPIO, opportunément tout près des 3,3 V (broche 1), 5 V (broche 2) et masse (broche 6). Ces broches appartiennent au bus I²C 1. On peut donc installer une mini-carte d'extension pour un capteur de température ou de position, ou bien une horloge en temps réel, en occupant un minimum de place.

Attention : il est impératif que la tension sur une broche du RPi **ne dépasse pas 3,3 V**, lui appliquer 5 V peut conduire à la destruction du RPi ! C'est pourquoi on doit contrôler soigneusement tout circuit avant de le connecter aux broches du RPi. En particulier, il faut s'assurer qu'un esclave I²C ne possède pas de résistances de rappel vers le 5 V. Au besoin, il faut les supprimer, elles n'ont de toute façon aucune raison d'être.

Grâce à un environnement confortable, le RPi se prête bien à la prise en main de nouveaux esclaves, jusqu'alors inconnus. La **figure 1** montre le raccordement au RPi d'un LM75 installé sur une plaque d'essai.

Mode maître

Pour pouvoir utiliser le bus I²C du RPi sous Raspbian, il faut commencer par charger deux pilotes additionnels. Pour cela, on lance `raspi-config` et on sélectionne dans la rubrique `Advanced Options` la sous-rubrique `I2C` pour activer l'interface et charger les modules du noyau. Autre méthode, on ajoute avec l'éditeur de texte les lignes :

```
i2c-dev
  i2c-bcm2708
```

au fichier /etc/modules.

Après un réamorçage, les deux pilotes (ainsi que d'autres pilotes éventuellement nécessaires) sont chargés et les liens vers les

périphériques `/dev/i2c-<n>` existent, ce qu'on peut vérifier avec les commandes ($...) suivantes :

```
$ lsmod | fgrep i2c_
i2c_dev           XXXX  0
i2c_bcm2708       YYYY  0
$ ls /dev/i2c-*
dev/i2c-1
```

Les nombres *XXXX* et *YYYY* indiquent la taille des modules, les deux zéros le fait qu'ils ne sont encore utilisés par personne. Ces commandes fonctionnent aussi en mode non-privilégié. On installe ensuite le paquet `i2c-tools` :

```
sudo apt-get install i2c-tools
```

qui contient entre autres un détecteur de bus et de modules I²C. Raspbian contient déjà les pilotes de certains périphériques I²C, comme l'horloge en temps réel (RTC) RV-8523. Par défaut, le RPi gère le bus I²C dans le mode standard, à la vitesse de 100 kbits par seconde.
Pour commencer, on peut, en utilisant l'outil `i2cdetect` à la console, obtenir une vue d'ensemble des esclaves présents sur le bus I²C numéro 1 et reconnus par le RPi. Le résultat est celui de la **figure 2** si le LM75 répond présent à l'adresse `0x48`. Avec les commandes `i2cget`, `i2cset` et `i2cdump`, on peut communiquer avec le LM75 sans avoir à programmer soi-même (`0x00` est le numéro du registre et il doit toujours être indiqué) :

```
pi@raspberrypi ~ $ i2cget -y 1 0x48 0x00 w
0xa010
```

Il faut permuter les deux octets de la réponse (`0x10a0`). Seuls les neuf bits de poids fort de la réponse sont significatifs (`0x021`). Le LM75 fournit la température par incréments de 0,5 K, on a donc mesuré une température de 16,5 °C.
Théoriquement, le matériel du RPi peut aussi fonctionner en mode esclave, mais ce mode n'est pas supporté par le pilote Linux.

Programmation en C et Python
Pour la programmation **en C**, on a besoin de cinq fonctions :

- `open()` pour l'accès au périphérique (ou port) I²C,
- `ioctl()` pour la configuration des paramètres de l'esclave I²C,
- `read()`et `write()` pour l'échange de données avec l'esclave,
- `close()` pour terminer la session (fermer l'accès au périphérique I²C).

Pour définir l'argument `I2C_SLAVE` utilisé ci-dessous, il faut ajouter le fichier `linux/i2c-dev.h` à la liste des fichiers inclus :

```
# include <linux/i2c-dev.h>
```

On ouvre l'accès au port I²C avec `open()` :

```
fd = open(«/dev/i2c-1», O_RDWR);
```

Ensuite on sélectionne l'adresse de l'esclave par un appel à

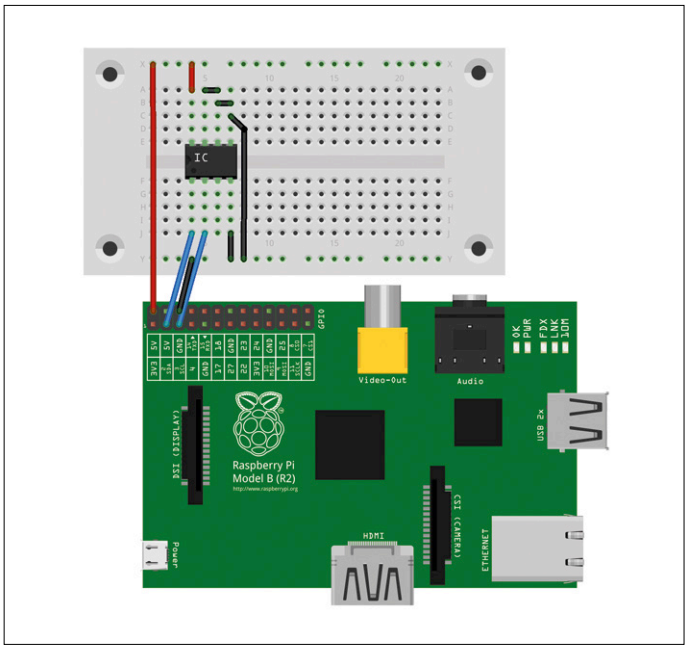


Figure 1. Connexion d'un LM75 sur carte d'essai avec un Raspberry Pi.

`ioctl()` :

```
ioctl(fd, I2C_SLAVE, 0x48);
```

Maintenant, nous pouvons lire et écrire :

```
unsigned char buf[2];
float T;
buf[0] = 0;
write(fd, buf, 1);   /* write register number 0 */
read(fd, buf, 2);    /* read temperature register */
T = ((buf[0]<< 8) | buf[1]) / 256.0;
```

Enfin, on termine la session :

```
close(fd);
```

Bien entendu, pour détecter les erreurs, il faut améliorer le traitement des valeurs retournées par les fonctions appelées. Par exemple pour savoir si l'on dispose du droit d'ouvrir `/dev/`

```
pi@raspberrypi ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 2. *i2cdetect* découvre le LM75 à l'adresse 48.

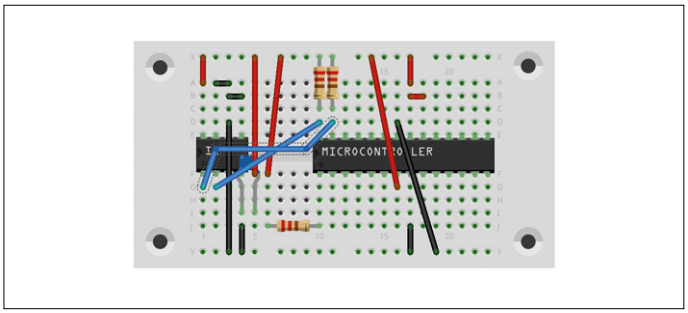


Figure 3. L'ATmega88 en compagnie du capteur sur une carte d'essai.

`i2c-1`, s'il existe un esclave à l'adresse `0x48` ou si l'échange de données a réussi.

Pour programmer le bus I²C **en Python**, il faut avoir installé le paquet Raspbian `python-smbus` :

```
sudo apt-get install python-smbus
```

On peut alors lire la température mesurée par le LM75 :

```
#! /usr/bin/python
import smbus
import time

bus = smbus.SMBus(1)
address = 0x48
```

Listage 1. Boucle principale de lecture du capteur LM75.

```
# include <i2cmaster.h>

# define LM75 (0x48 << 1)
                              // see datasheet
int
main(void)
{
unsigned char val[2];

i2c_init();
                              // initialisation I2C

i2c_start(LM75 | I2C_WRITE);
                              // addressing to write
i2c_write(0x00);
                              // temperature register
i2c_rep_start(LM75 | I2C_READ);
                              // addressing to read
val[0] = i2c_read();
                              // degrees Celsius
val[1] = i2c_read();
                              // tenth part bit
i2c_stop();
                              // ready
for(;;);
}
```

```
w = bus.read_word_data(address, 0)
print format(w, '04x')
```

Le « `0` » dans l'appel de la méthode `read_word_data` indique le numéro du registre interne de l'esclave, ici le registre de température (le protocole SMBus qui encapsule l'I²C exige l'envoi préalable d'un numéro de registre). Cela peut poser problème, par ex. pour la puce d'expansion d'entrées-sorties PCF8574 qui ne possède pas de registre d'adresses !
La valeur lue pose le même problème que `i2cget` : il faut permuter les deux octets du mot `w`.

ATmega

Les microcontrôleurs ATmega d'Atmel disposent d'un contrôleur I²C intégré qui gère le mode standard (100 kHz) ainsi que le mode rapide (400 kHz) et qui peut être utilisé en mode maître, esclave ou combiné. Les ATmega324PB et ATmega328PB possèdent même deux bus I²C. Dans les exemples suivants, j'utilise un ATmega88 enfiché sur la platine d'essai à côté du capteur de température (**fig. 3**).
Ici, le mieux est d'utiliser la bibliothèque `i2cmaster` de Peter Fleury [1], mais en notant que cette bibliothèque n'active pas les résistances internes de rappel vers le haut. Il faut donc fournir ces résistances soi-même. Elles apparaissent en haut de la figure 3, au niveau de la ligne d'alimentation 5 V. La bibliothèque configure le port I²C dans le mode standard (100 kHz). L'accès au capteur LM75 prend alors la forme du **listage 1**.
Avec la bibliothèque `lcdlibrary` du même auteur, on peut réaliser un thermomètre numérique rien qu'en ajoutant un afficheur à cristaux liquides. On peut aussi utiliser une carte d'interface I²C (qui embarque le plus souvent un PCF8574) ou bien directement un afficheur avec une interface I²C intégrée.
Le contrôleur I²C intégré de la série ATmega n'est pas seulement capable de fonctionner en mode maître pour gérer des esclaves, il peut aussi travailler en mode esclave, sous le contrôle d'un maître. Il peut être maître pour communiquer avec un LM75, puis esclave pour être lu par un RPi. Utiliser l'ATmega comme esclave peut être intéressant dans le cas où celui-ci acquiert des données sur ses broches d'entrées-sorties et les soumet à un prétraitement avant de les faire suivre à un RPi.
Mais ce mode de fonctionnement est plus complexe et en interaction plus étroite avec le reste du logiciel. La bibliothèque disponible en [2] est un bon point de départ puisqu'elle permet d'utiliser le contrôleur I²C des ATmega en esclave.
Le déroulement décrit ci-dessous suit les états décrits dans les feuilles de caractéristiques d'Atmel. Les tableaux correspondants se trouvent dans chaque feuille de caractéristiques d'Atmel, ainsi qu'au chapitre 22.7 du manuel de l'ATmega48/88/168.
Pour une meilleure efficacité, l'exécution du code est pilotée par interruptions. L'unité de contrôle I²C provoque une interruption dans les conditions suivantes :

- après l'émission d'une *START condition* ou d'une *Repeated Start condition*,
- après l'émission de l'adresse et du bit de lecture/écriture,
- après l'envoi d'un octet de données,
- quand l'ATmega a perdu l'arbitrage (lorsqu'il se produit une collision lors de l'envoi d'une *START condition* ou de l'octet d'adresse ou du bit de lecture/écriture),
- quand l'ATmega a reconnu une *STOP condition* alors qu'il a

été adressé comme esclave
- quand l'ATmega a reçu un octet de données
- quand l'ATmega a reconnu une *START condition* ou une *Repeated Start condition* alors qu'il a été adressé (de nouveau) comme esclave,
- quand une transaction invalide a été détectée sur le bus

Comme les quatre premières conditions concernent le mode maître, nous ne nous intéresserons ici qu'aux quatre dernières.
L'unité de contrôle I²C doit être initialisée avec l'adresse d'esclave désirée. On peut ensuite l'activer et commencer le travail :

```
TWAR = (I2C_Slave_Addr << 1);
TWCR = _BM(TWEA) | _BM(TWEN) |
   _BM(TWIE);
```

Il n'y a pas lieu de régler la vitesse en mode esclave, étant donné qu'elle est imposée par le maître.
Si une interruption se présente, il faut en déterminer la raison. Pour cela, on lit le registre d'état `TWSR` et on exploite les cinq bits de poids fort.
Le **listage 2** présente une partie de la routine de service de l'interruption (ISR) pour l'adressage comme récepteur, comme émetteur, ainsi que pour la *STOP condition.*
Les descriptions et les tableaux des feuilles de caractéristiques ATmega sont très complets, aussi bien pour l'explication des codes d'état que pour les réponses requises du programme et les actions matérielles qu'elles provoquent.

Arduino

Comme beaucoup de cartes Arduino sont basées sur des microcontrôleurs de la famille ATmega, toutes les explications ci-dessus restent en principe valables. Toutefois l'Arduino bénéficie d'une bibliothèque souvent utilisée et très pratique du nom de `Wire`.
Pour l'Arduino, le raccordement est aussi simple que pour le RPi (**fig. 4**). Selon le type (ou le clone) d'Arduino, il faut prendre garde à ce que les broches d'un Arduino alimenté en 3,3 V ne soient jamais soumises à une tension de 5 V. Donc, mesurez d'abord, connectez ensuite et, le cas échéant, supprimez les résistances de rappel vers le haut de l'esclave !

La bibliothèque `Wire` est déjà contenue dans l'Environnement de Développement Intégré (EDI) ; de même que pour la bibliothèque `i2cmaster` de Peter Fleury, il faut commencer

Listage 2. Routine de traitement d'interruption de la *STOP condition*. Le code complet et abondamment commenté est à télécharger sur la page du projet [5].

```
ISR(TWI_vect)
{
    /*
     * These variables need to be preserved across interrupts
     */
    static unsigned char i2c_idx,       /* Index into twi_msg[] */
                         i2c_tosend;    /* Number of bytes to send */

    switch (TWSR & 0xf8)
    {
    ...
    /*
     * RECEIVE Code
     * See Table 19-4.  Status Codes for Slave Receiver Mode
     * [Page 229]
     */
    case 0x60:
        /*
         * Own SLA+W has been received; ACK has been returned
         * TWDR: No TWDR action
         * STA=X STO=0 TWINT=1 TWEA=1
         * Data byte will be received and ACK will be returned
         */
        TWCR = (TWCR & ~_BM(TWSTO)) | (_BM(TWINT) | _BM(TWEA));
        i2c_idx = 0;
        break;
    …
    case 0xA0:
        /*
         * A STOP condition or repeated START condition has been
         * received while still addressed as slave
         * TWDR: No action
         * TWA=0 STO=0 TWINT=1 TWEA=1
         * Switched to the not addressed Slave mode;
         * own SLA will be recognized;
         * GCA will be recognized if TWGCE = "1"
         */
        TWCR = (TWCR & ~(_BM(TWSTA) | _BM(TWSTO))) | (_BM(TWINT) | _BM(TWEA));
        break;
    …
    case 0xA8:
        /* Own SLA+R has been received; ACK has been returned
         * TWDR: Load data byte
         */
        /*
         * The address (register number) has been received,
         * Start sending payload
         */
      TWDR = 0x42;
      break;
    }
}
```

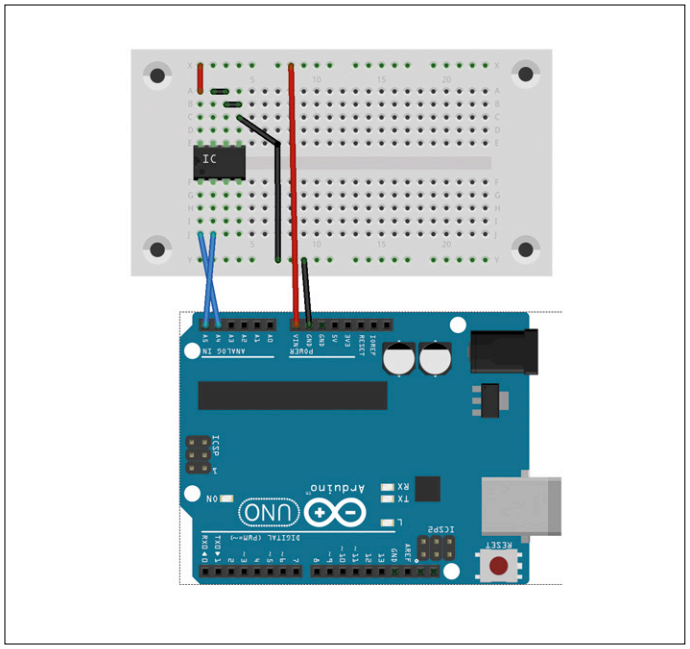


Figure 4. La connexion à l'Arduino est pratiquement identique à celle au RPi.

par initialiser la bibliothèque, c'est-à-dire le port I²C :

```
#include <Wire.h>
void setup() {
 Wire.begin();
}
```

L'appel de `Wire.begin()` active les résistances internes de rappel vers le haut. Si cela pose problème du fait des valeurs assez élevées de ces résistances, on peut leur connecter en parallèle deux résistances de 10 à 20 kΩ. Ou bien, on les désactive immédiatement et on n'utilise que des résistances externes d'environ 4,7 kΩ. Pour cela, on ajoute les deux lignes suivantes immédiatement après l'appel à `Wire.begin()` :

```
digitalWrite(SDA, 0);
digitalWrite(SCL, 0);
```

Remarque : ceci n'est pas officiel et pourrait totalement changer dans l'avenir. Le mieux est de faire une mesure au multimètre après l'initialisation.
Attention : même si l'on désactive aussitôt les résistances internes de rappel, elles ont quand même eu une courte période

Le bus I2C sur PC

Chaque PC possède une interface I²C et, le plus souvent, plusieurs. On rencontre des esclaves I²C dans les afficheurs (DDC [3]) et les barrettes de mémoires (SPD [4]). Même les capteurs de température peuvent être connectés par I²C. Malheureusement, pratiquement aucun constructeur ne fournit d'information sur les types de puces utilisées, ni comment accéder aux bus I²C de l'extérieur (à supposer que ce soit possible et qu'ils ne soient pas enfermés à l'intérieur de quelques modules).

Si l'on est sous Linux, on peut essayer de charger le module *i2c-dev* :

```
$ sudo modprobe i2c-dev
```

et voir ensuite s'il existe des périphériques *i2c* dans */dev* :

```
$ ls /dev/i2c*
/dev/i2c-0  /dev/i2c-1  /dev/i2c-2  /
dev/i2c-3 /dev/i2c-4 /dev/i2c-5
```

Cela marche aussi en mode non-privilégié. Sous Debian et ses avatars (Raspbian et Ubuntu), il y a les outils *i2c-tools* :

```
apt-get install i2c-tools
```

Il arrive qu'un seul des bus soit doté d'esclaves :

```
root@bounty:~# i2cdetect -y 5
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- 37 -- -- 3a -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- -- -- -- -- -- -- 58 -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Il n'est toutefois pas toujours très facile de savoir de quel genre d'esclaves il s'agit. Ici, on a à l'adresse 0x50 la PROM EDID de l'afficheur :

```
root@bounty:~# i2cdump -y 5 0x50
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 00 ff ff ff ff ff ff 00 1a b3 d4 07 ec 22 02 00    ........?????"?.
10: 0a 16 01 03 80 34 20 78 2a ef 95 a3 54 4c 9b 26    ?????4 x*???TL?&
20: 0f 50 54 a5 4b 00 81 80 81 00 81 0f 95 00 95 0f    ?PT?K.???.???.??
30: a9 40 b3 00 01 01 28 3c 80 a0 70 b0 23 40 30 20    ?@?.??(<??p?#@0
40: 36 00 06 44 21 00 00 1a 00 00 00 fd 00 38 4c 1e    6.?D!..?...?.8L?
50: 52 10 00 0a 20 20 20 20 20 20 00 00 00 fc 00 42    R?.?      ...?.B
60: 32 34 57 2d 35 20 45 43 4f 0a 20 20 00 00 00 ff    24W-5 ECO?  ....
70: 00 59 56 32 45 31 34 30 30 31 32 0a 20 20 00 8e    .YV2E140012?  .?
```

Si vous disposez encore d'un port VGA, essayez de connecter un esclave I²C à la broche 12 (SDA) et à la broche 15 (SCL). Les broches 6 (SCL) et 7 (SDA) d'un connecteur DVI, ainsi que les broches 15 (SCL) et 16 (SDA) d'un connecteur HDMI sont également de bons candidats. Mais ces bus sont probablement sous le contrôle total de la carte graphique et de son micrologiciel.

d'activité. Cela peut avoir des conséquences graves pour un esclave 3,3 V avec des résistances externes de rappel vers 3,3 V, connecté directement à un Arduino 5 V. Dans ce cas, il faut impérativement intercaler un convertisseur de niveau ! Après l'initialisation, on peut programmer un envoi de données à un esclave I²C dans la fonction `loop()`, par ex. l'adresse du registre de température du capteur LM75 :

```
void loop() {
 Wire.beginTransmission(0x48);
 Wire.write(byte(0x00));
 Wire.endTransmission();
```

ou bien une lecture de données :

```
Wire.requestFrom(0x48, 2);
 c1 = Wire.read();
 c2 = Wire.read();
}
```

Comme on le voit, il faut indiquer à la bibliothèque `Wire` le nombre d'octets à recevoir de l'esclave (ici: `2`). Avec cet appel, on en lit déjà deux. Ensuite, on peut les acquérir un par un.
La bibliothèque `Wire` peut aussi être utilisée pour le mode esclave. Là aussi, il faut commencer par l'initialiser. Pour cela, on appelle `Wire.begin()` avec un paramètre représentant l'adresse choisie pour l'esclave, ce qui active le mode esclave. De plus, il faut spécifier une fonction de traitement des événements exécutée lorsque l'Arduino est adressé comme esclave, par ex. dans le cas où il doit recevoir des données :

```
#include <Wire.h>
void setup() {
 Wire.begin(0x48);
 Wire.onReceive(receive);
}
```

Ici aussi, la méthode `Wire.begin()` active les résistances de rappel vers le haut, chose plutôt inattendue pour un esclave. Il faut donc les désactiver immédiatement, comme ci-dessus.

Avertissement : cela n'empêche pas un Arduino 5 V d'être utilisable comme esclave d'un maître 3,3 V, mais une impulsion à 5 V sur des lignes 3,3 V peut détruire le maître. Si l'on veut connecter un Arduino à un RPi, on doit obligatoirement faire en sorte que la connexion électrique ne s'établisse qu'après l'initialisation de l'Arduino.
Dès l'arrivée de données, le gestionnaire d'événements `receive` spécifié ci-dessus est appelé avec le nombre d'octets à lire en paramètre :

```
void receive(int n) {
 while (n--) {
  uint8_t c = Wire.read();
  // traitement de c
 }
}
```

Pour l'envoi de données au maître, on appelle, au lieu de `onReceive()`, la méthode `onRequest()` et on lui passe un gestionnaire d'événements qui met à disposition les données.

Comme la quantité de données à transmettre ou à recevoir est inconnue, ce gestionnaire n'a pas de paramètre. Les données doivent toutefois être transmises par **un** appel à `Wire.write()` :

```
#include <Wire.h>
void setup() {
 Wire.begin(0x48);
 Wire.onRequest(transmit);
}
void loop() {
 while (1) delay(1000);
}
void transmit() {
 uint8_t msg[N];
 // production du contenu de msg[]
 Wire.write(msg, N);
}
```

Ces deux méthodes peuvent être combinées dans un croquis :

```
#include <Wire.h>
void setup() {
 Wire.begin(0x48);
 Wire.onReceive(receive);
 Wire.onRequest(transmit);
}
```

On peut ainsi reproduire un LM75 précis, utiliser un capteur 1-Wire (DS18B20) à la place d'un LM75 ou émuler une horloge en temps réel qui reçoit l'heure par DCF77 ou GPS.

Dans la troisième partie

La partie suivante de la série présentera quelques composants I²C très appréciés : outre le capteur de température LM75, la puce d'expansion d'entrées-sorties PCF8574 et l'horloge en temps réel RV8523. Cette petite série d'articles se conclura sur le thème de la recherche et de l'analyse d'erreurs avec des outils appropriés et à la portée d'un labo d'amateur. ⏮

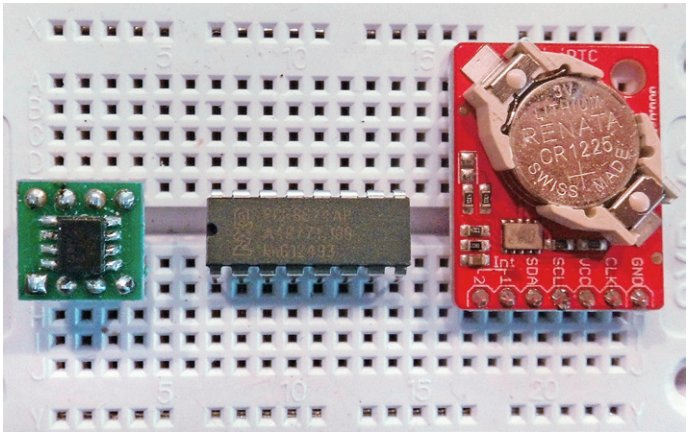
(160418 – version française : Helmut Müller)

Liens

[1] http://homepage.hispeed.ch/peterfleury/avr-software.html
[2] www.mikrocontroller.net/topic/182638
[3] https://en.wikipedia.org/wiki/Display_Data_Channel
[4] https://en.wikipedia.org/wiki/Serial_presence_detect
[5] www.elektormagazine.fr/160148

le bus I²C

3e partie : composants et analyse des erreurs



La dernière partie de cette série est consacrée à trois puces I²C typiques : un capteur de température, un extenseur d'entrées-sorties et une horloge en temps réel. Nous apprendrons comment lire et écrire leurs registres et de quels outils matériels et logiciels on peut se servir pour cela.

Josef Möllers

Il existe de nombreux composants et modules I²C, du capteur de température aux afficheurs graphiques, en passant par les horloges en temps réel et les capteurs de position et de mouvement. Il n'y a qu'à en rechercher une liste sur l'internet ou bien taper « I2C » dans le champ de recherche du site web de votre fournisseur de matériel électronique favori. Et voilà puces et modules qui vous arrivent par pages entières [1], nous vous en présenterons quelques-uns ici.

Les modules I²C ne nécessitent pas une connexion bien compliquée avec leur hôte. Avec un câble ruban à 4 conducteurs pour les données et l'alimentation, le tour est joué. Sur l'image du chapeau de cet article, on distingue les trois composants dont il est question ici, enfichés côte à côte dans une platine d'essai, de gauche à droite : un capteur de température LM75 monté sur un adaptateur, un extenseur d'entrées-sorties PCF8574 et un module RTC avec la puce d'horloge en temps réel RV-8523, à gauche, en dessous de la pile.

LM75

Le LM75 est **LE** standard du capteur de température connecté par bus I²C. Dans son adresse à 7 bits, seuls les 4 bits de poids fort sont fixes, les 3 autres bits sont déterminés par un circuit extérieur. Il est donc possible de connecter huit LM75 avec les adresses 0x48 à 0x4F. Si, par exemple, on utilise le LM75 pour contrôler la température d'un gros ordinateur, on peut

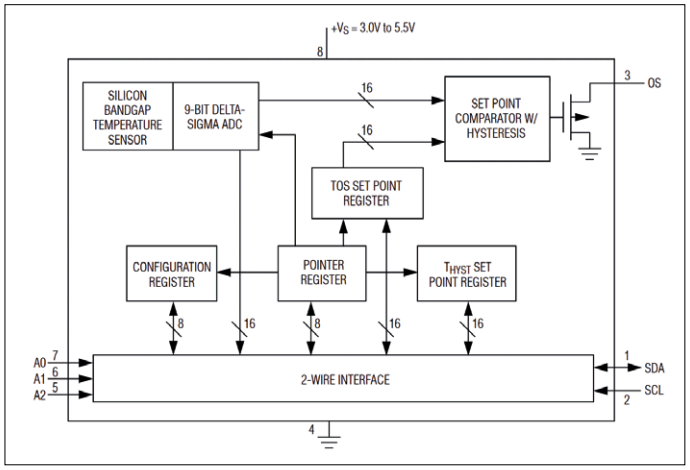


Figure 1. Logique interne du capteur de température LM75 (source : Maxim).

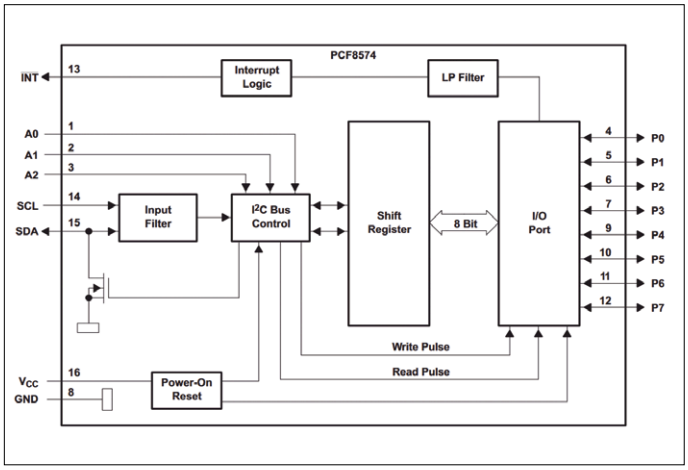


Figure 2. Structure interne de l'extenseur de port PCF8574 (source : Texas Instruments).

en disséminer huit dans l'enceinte de la machine.
Le LM75 dispose de quatre registres internes, adressables sur deux bits (**fig. 1**) :

- 00H : registre à 16 bits de température, en lecture seule,
- 01H : registre à 8 bits de configuration,
- 02H : registre à 16 bits d'hystérésis,
- 03H : registre à 16 bits de comparaison.

Même si, à la mise sous tension, c'est le registre de température qui est sélectionné par défaut, on doit toujours, avant de lire un registre (y compris le registre de température), transmettre son adresse. En écriture, cela est systématique, car le premier octet transmis au LM75 est toujours interprété comme une adresse de registre.
Les données suivent : pour les registres à 16 bits, un premier octet de poids fort, suivi de l'octet de poids faible.
Contrairement à d'autres puces I²C, le LM75 n'incrémente pas automatiquement l'adresse du registre après un accès, cette adresse ne change pas. S'il n'y a qu'un maître et qu'on ne fait que lire la température, il n'est pas nécessaire de remettre l'adresse du registre à zéro à chaque fois, il suffit de lire les deux octets de température lors de chaque transaction.
La précision du LM75 laisse malheureusement à désirer : la feuille de caractéristiques indique une incertitude de 2 °C. Il existe des alternatives plus précises, comme le TMP275, qui sont le plus souvent compatibles en termes de protocole de communication et de registres.
Le LM75 est également très sensible aux perturbations de l'alimentation. Il est donc impératif de respecter la règle d'or : « une puce, un condensateur de 100 nF », sous peine de voir le maître amasser du « n'importe quoi ».

PCF8574

Le PCF8574 est un extenseur distant d'entrées-sorties numériques à 8 bits, un gestionnaire de port parallèle contrôlé par le bus I²C (**fig. 2**). Il existe en deux variantes qui ne se distinguent que par leur adresse I²C : les quatre bits de poids fort du PCF8574 valent 0100, ceux du PCF8574A valent 0111. On peut ainsi piloter seize de ces puces sur un bus I²C.
En interne, il n'existe qu'un seul registre qui est directement relié aux entrées-sorties. Si l'on écrit un masque de bits dans le registre, on change l'état des entrées-sorties. Si l'on met l'état de l'un des bits à « 1 », on peut aussi l'utiliser comme entrée. Si on lit le registre, on obtient une information sur l'état des broches. Sur la **figure 3**, on a connecté au RPi un PCF8574 avec une LED reliée à P0.
Si l'on écrit un « 1 » dans le PCF8574,

```
i2cset -y 1 0x40 0x01
```

la LED s'allume. Si l'on écrit un « 0 »,

```
i2cset -y 1 0x40 0x00
```

la LED s'éteint.
Plus exactement, on a envoyé un numéro de registre 0x01 puis 0x00, car, selon la documentation et l'aide en ligne, i2cset exige à la suite de l'adresse I²C un numéro de registre, qui est interprété par l'extenseur de port comme un masque de bits pour ses sorties.

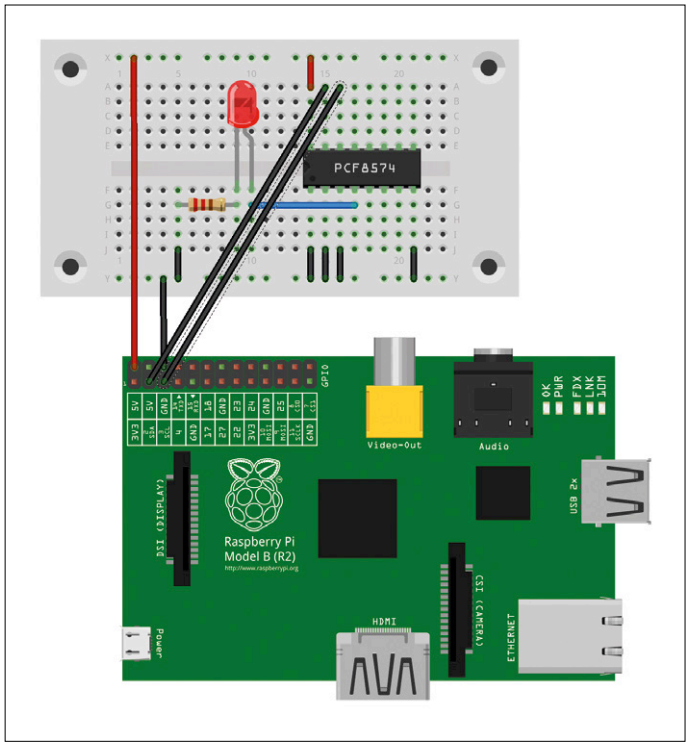


Figure 3. L'extenseur de port PCF8574 connecté à un RPi.

On a quelquefois intérêt à séparer électriquement un processeur et un périphérique, de sorte qu'un court-circuit accidentel, avec le 12 V par exemple, ne détruise « que » le PCF8574.
Le PCF8574 est aussi utilisé sur des cartes d'interfaçage simples qui apportent la connectivité I²C aux afficheurs LCD standard à une ou deux lignes équipés de contrôleurs HD44780. L'afficheur est alors contrôlé dans le mode à quatre bits, trois autres bits du PCF8574 fournissent les signaux E, RS et R/W. Le dernier bit peut être utilisé pour contrôler le rétroéclairage de l'afficheur. Sur certaines de ces cartes, il est prévu des résistances de rappel vers le 5 V. Il est recommandé de ne surtout pas les monter et même de les supprimer le cas échéant.

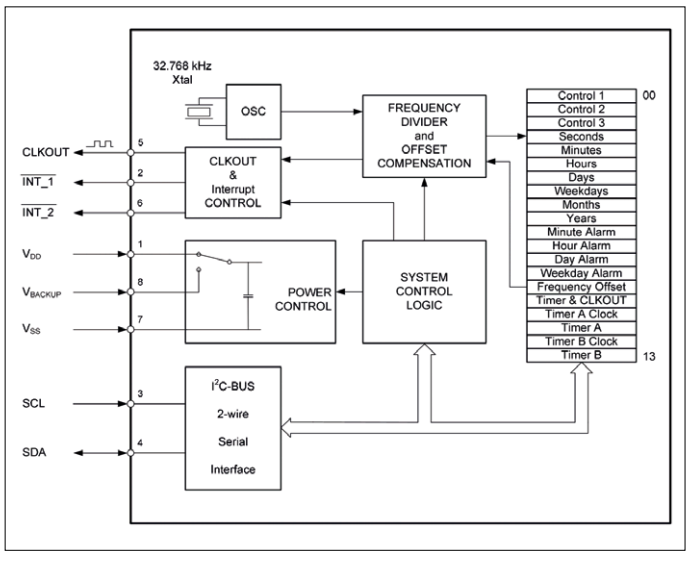


Figure 4. L'horloge en temps réel dispose de 14 registres adressables pour la commande, l'heure et l'alarme (source : Micro Crystal).

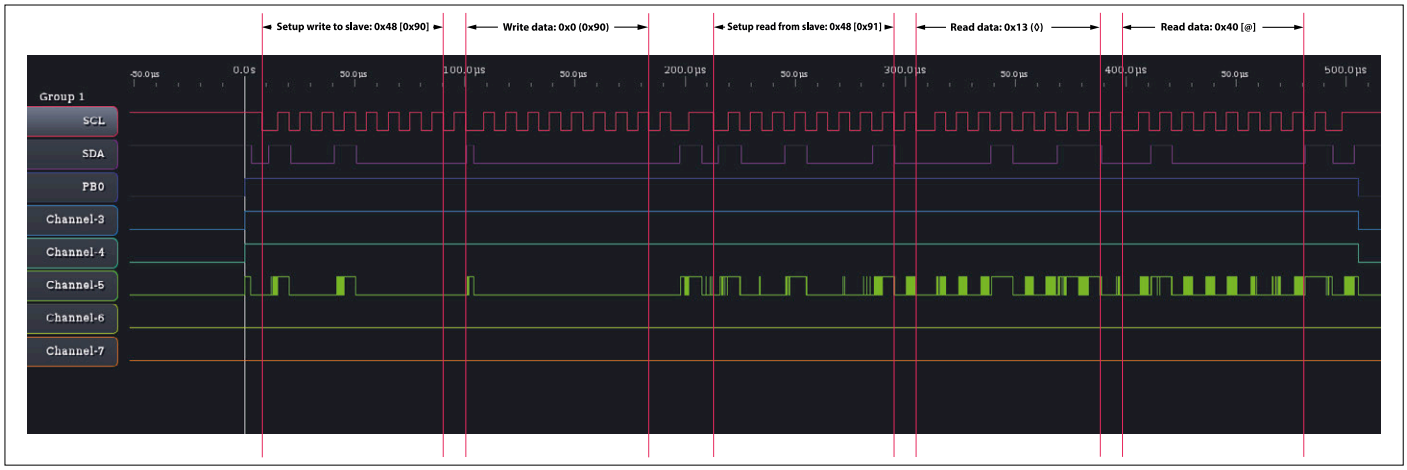


Figure 5. L'Open Logic Sniffer observe comment un ATmega88 lit les valeurs de température d'un LM75.

RV-8523

Le RV-8523 est une puce horloge en temps réel (RTC, *Real Time Clock*) avec vingt registres de 8 bits aux adresses 00H à 13H, listés sur la **figure 4**. Ce circuit possède une surveillance interne de la tension et peut commuter automatiquement son alimentation sur une pile. Comme le montre l'image du chapeau, la puce est souvent fournie montée sur une carte complète, comprenant le support de la pile.

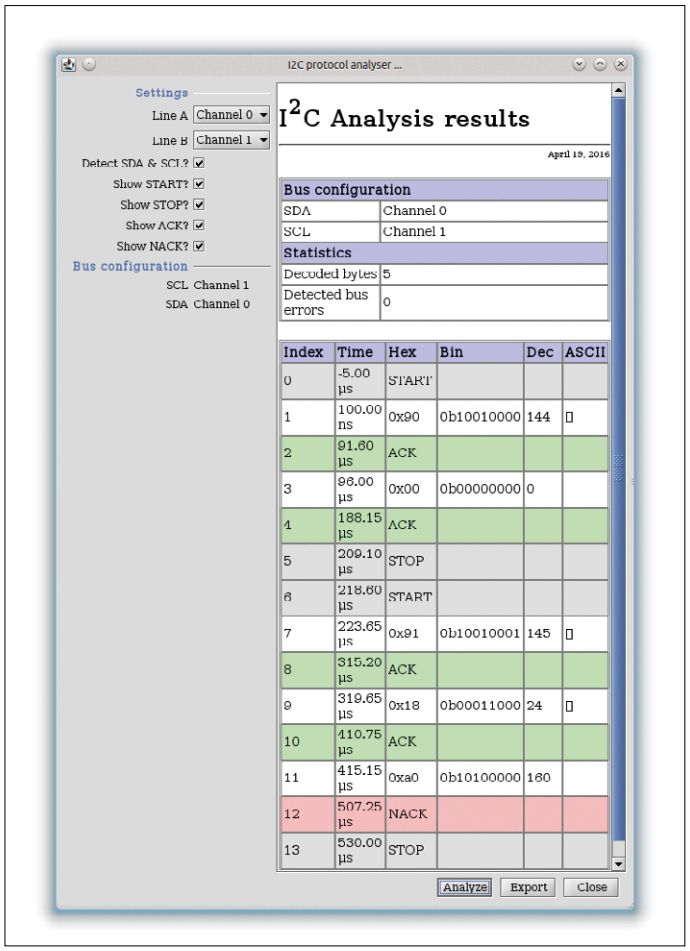


Figure 6. Le client OLS Java affiche les paramètres du bus sous forme de tableau.

Après avoir transmis l'adresse d'un registre (entre 0x00 et 0x13), on peut accéder à ce registre. Contrairement au LM75, cette adresse est incrémentée, avec un passage de 0x13 à 0x00. Avec un seul accès, on peut donc lire ou écrire l'ensemble des vingt registres.
Mais si l'on ne s'intéresse qu'à la date et à l'heure, on met l'adresse du registre à 3 et on lit 7 octets, comme dans cet exemple qui utilise la bibliothèque *Wire* d'Arduino :

```
Wire.beginTransmission(0x68);
 Wire.write(byte(0x03));
     // set register number to 3
 Wire.endTransmission();
 Wire.requestFrom(0x68, 7);  // read time and date
 seconds = Wire.read();
 tenseconds = (seconds >> 4) & 0x07; seconds &= 0x0f;
 minutes = Wire.read();
 tenminutes = (minutes >> 4) & 0x07; minutes &= 0x0f
 ...
```

Les valeurs sont déjà codées en BCD, une conversion de binaire en décimal est donc superflue.

Outre son horloge, le RV-8523 possède une fonction réveil qui produit une interruption à une heure prédéterminée. Le problème est que la sortie INT_1 passe bien au niveau bas à l'heure prévue, mais ne revient pas d'elle-même au niveau haut. La condition d'alarme doit être annulée explicitement (AF dans le registre de commande 2).
Certains systèmes d'exploitation, comme Raspbian, fournissent un pilote pour ce composant (rtc_pcf8523), ce qui évite d'avoir à programmer soi-même, du moins en ce qui concerne l'heure. La commande hwclock accède à l'horloge en temps réel et lit ou modifie son heure. Au démarrage du système, un script rc exécute cette commande et copie l'heure en temps réel dans l'horloge du système ; lors de l'arrêt, le temps système (éventuellement actualisé manuellement ou par internet) est copié dans l'horloge en temps réel. De cette manière, on peut sauvegarder l'heure sur un RPi, même non connecté à l'internet, en cas d'interruption de l'alimentation. Mais on n'échappe pas à la nécessité de programmer soi-même si l'on veut utiliser la fonction réveil.

Lorsqu'on a mis à l'heure la carte en temps réel sur un RPi, on peut ensuite la connecter à un ATmega ou un Arduino. La pile sur la carte assure la continuité du fonctionnement de l'horloge. Avec seulement quelques lignes de code, on dispose ainsi de la date et de l'heure sur un ATmega ou un Arduino.

Recherche et analyse d'erreurs

Tout ne fonctionne pas toujours du premier coup. On peut commencer la recherche d'erreurs avec un multimètre, mais si l'on a affaire à des problèmes de protocole, ça ne suffira pas le plus souvent.
Au repos, SDA et SCL sont au niveau haut. Pour commencer, on vérifie donc au **multimètre** si, après la mise sous tension (et de préférence quand le bus est au repos), SCL et SDA sont à un niveau suffisamment élevé. Comme déjà mentionné précédemment, le standard requiert un niveau haut d'au moins 0,7 V_{cc} (donc d'au moins 3,5 V pour 5 V). Toutefois, de nombreuses puces 5 V fonctionnent aussi avec des résistances de rappel au 3,3 V, du fait que le niveau haut dans la logique 3,3 V dépasse sensiblement les 3 V. Peut-être peut-on résoudre le problème rien qu'avec une ou deux résistances.

Si les tensions sont correctes, il faut recourir à d'autres moyens. Il existe une série d'**analyseurs logiques** dans une gamme de prix étendue. Qu'il s'agisse d'un petit ScanaQuad [2] ou d'un Red Pitaya [3] évolué ou, comme chez moi, d'un Open Logic Sniffer [4] de Dangerous Prototypes avec le client OLS Java de ols.lxtreme.nl [5] : ils sont tous utilisables pour analyser le flux de données sur le bus I²C.
La **figure 5** montre comment un ATmega lit le registre de température d'un LM75. Au début (à 0,0 s), on voit la condition de départ (*start*), à la fin (à 500,0 µs) la condition d'arrêt (*stop*) et à 200,0 µs la condition de répétition du départ (*repeated start*). Entre le point de départ et 200,0 µs environ, le LM75 est adressé en écriture et l'adresse de registre 0x00 lui est envoyée ; après la condition de répétition du départ, le LM est adressé à nouveau, cette fois pour la lecture, et les deux octets du registre de température 0x13 et 0x40 sont envoyés vers l'ATmega. Le déclenchement passe par PB0. Le transfert commence quand PB0 est mis à 1 ; en fin de transfert, PB0 est remis à 0. Autrement, on peut aussi lancer l'enregistrement par SDA = niveau bas.
Le client OLS Java analyse l'enregistrement et la communication I²C, et présente les octets échangés sous forme d'un tableau (**fig. 6**). Par erreur, la condition de répétition est représentée comme condition d'arrêt, suivie d'une condition de départ, alors que le diagramme montre clairement qu'aucune condition d'arrêt n'est envoyée à 200,0 µs.

Le **Bus Pirate** (**fig. 7**) de la société Dangerous Prototypes [6] est un outil d'analyse de protocoles sériels tels que I²C, SPI, ou du trafic sur un UART. Dangerous Prototypes commercialise elle-même le Bus Pirate et l'Open Logic Sniffer, mais met aussi librement à disposition le matériel et le logiciel, de sorte qu'on peut réaliser soi-même les deux modules. Selon le fabricant, la version 4 du Bus Pirate est « orientée vers le futur », mais elle ne fonctionne pas de manière aussi stable que la version 3.6, disponible chez le distributeur allemand Watterott Electronic [7].
Le **tableau 1** montre l'adressage et le transfert de données. On voit que dans les deux cas (écriture et lecture), le LM75 acquitte son adresse. Cela vaut aussi pour le numéro du registre (0x00),

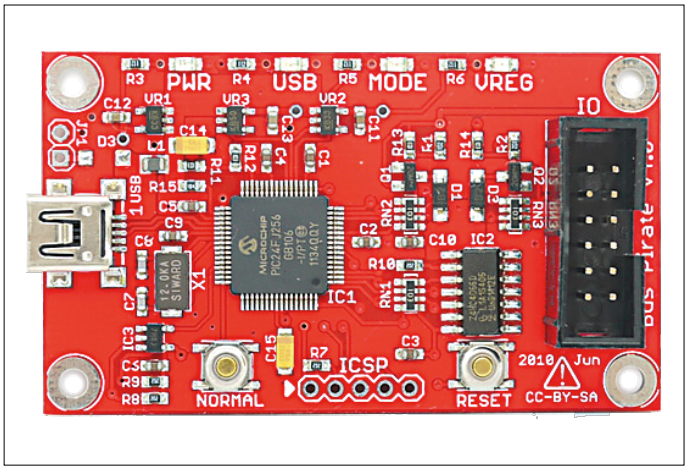


Figure 7. Le petit Bus Pirate est une interface de bus universelle pour PC (ici, la nouvelle version 4 ; source : Dangerous Prototypes).

car il se pourrait que d'autres octets soient transmis même si le registre de température est à lecture seule. L'ATmega88 acquitte le premier octet issu du registre de température avec ACK parce qu'il en attend un autre, et le second avec NACK parce que c'est le dernier à devoir être transféré. Ensuite l'ATmega88 termine la communication. Le Bus Pirate n'est pas capable d'identifier qui a envoyé ACK ou NACK.

Tableau 1. Adressage et transfert de données pour un LM75.

HiZ>m 1. HiZ 2. 1-WIRE 3. UART 4. I2C 5. SPI 6. 2WIRE 7. 3WIRE 8. LCD 9. DIO x. exit(without change)	Afficher le menu …
(1)>4 Set speed: 1. ~5KHz 2. ~50KHz 3. ~100KHz 4. ~400KHz	… I²C Bus …
(1)>3 I2C READY	… vitesse 100 kHz
I2C>(2) Sniffer Any key to exit [0x90+0x00+][0x91+0x13+0x40-]	Démarrer le *Bus Sniffer*
en gras: commande [*indique une* condition de START, *les chiffres donnent les valeurs transmises,* *+ un ACK,* *- un NACK et*] *une* condition de STOP	

Tableau 2. Lecture d'un capteur LM75 avec le Bus Pirate en mode maître.	
I2C>[0x90 I2C START BIT WRITE: 0x90 ACK	Envoyer la condition de *START*, l'adresse 0x48 et le bit d'écriture (0), le LM75 répond avec ACK
I2C>0x00 WRITE: 0x00 ACK	Envoyer l'octet 0x00, le LM75 répond avec ACK
I2C>] I2C STOP BIT	Envoyer la condition de *STOP*.
I2C>[0x91 r:2 I2C START BIT WRITE: 0x91 ACK READ: 0x14 ACK 0x20	Envoyer à nouveau la condition de *START*, puis l'adresse 0x48 et le bit de lecture (1), le LM75 renvoie ACK. Lire ensuite 2 octets. Le Bus Pirate acquitte le premier octet reçu avec ACK, le second seulement lorsqu'il est établi que c'est aussi le dernier, c'est-à-dire lorsqu'il a été averti d'émettre la condition de *STOP*.
I2C>] NACK I2C STOP BIT	Envoyer à nouveau la condition de *STOP*, le dernier bit reçu étant aussi le dernier à recevoir, l'acquitter au préalable avec NACK.
On peut aussi saisir l'ensemble sur une seule ligne :	
I2C>[0x90 0x00][0x91 r:2] I2C START BIT WRITE: 0x90 ACK WRITE: 0x00 ACK I2C STOP BIT I2C START BIT WRITE: 0x91 ACK READ: 0x14 ACK 0xA0 NACK I2C STOP BIT	

Le Bus Pirate peut aussi fournir les résistances de rappel (10 kΩ). Pour la version 3, il faut que la broche 5 (VPU) du bornier d'entrées/sorties soit raccordée à la tension de rappel souhaitée. Cette tension alimente alors un commutateur analogique auquel sont connectées toutes les résistances de rappel sur le Bus Pirate.

Cela est en principe superflu et même dangereux si un RPi y est également connecté. Dans la nouvelle version v4, la connexion est gérée par logiciel. Il est facile de déterminer si une résistance de rappel est connectée et à quelle tension par une mesure sur les broches 7 (SCL) et 8 (SDA). Les résistances de rappel peuvent être activées ou désactivées par les commandes :

```
I2C>P
Pull-up resistors ON (activer)
```

et

```
I2C>p
Pull-up resistors OFF (désactiver)
```

Ensuite, avec le Bus Pirate, on peut, de la même manière qu'avec la commande Raspbian i2cdetect, rechercher les esclaves sur le bus I²C, ce qui affiche leurs fonctions d'écriture et de lecture séparément :

```
I2C>(1)
Searching 7bit I2C address space.
Found devices at:
0x90(0x48 W) 0x91(0x48 R)
```

C'est l'alternative si, ne disposant pas d'un RPi, on avait un doute sur l'adresse affectée à un esclave. On peut aussi utiliser le Bus Pirate comme maître et exécuter manuellement le protocole I²C. Le **tableau 2** donne l'exemple d'une lecture de température sur un LM74.

En résumé

Le bus I²C est un moyen extrêmement simple de connecter des périphériques à un processeur lorsque les volumes de données sont faibles et qu'aucun traitement d'interruptions n'est nécessaire. Deux fils, voilà tout ce qu'il faut ! Et comme il s'agit d'un protocole multipoint, on peut connecter plusieurs esclaves sur ces deux fils. La ligne ne devrait toutefois pas dépasser une longueur d'un mètre. Grâce à des bibliothèques gratuites, comme celle de Peter Fleury, l'effort de programmation reste limité et de nombreuses causes d'erreurs sont exclues d'office. Les instruments de mesure nécessaires à la recherche d'erreurs soit sont disponibles la plupart du temps (multimètre), soit ne coûtent pas une fortune. Une infinité de capteurs et autres modules I²C sont impatients de prendre contact avec la carte à microcontrôleur sur la paillasse de votre laboratoire ! ⏮

(160373 – version française : Helmut Müller)

Liens

[1] Par exemple : http://rn-wissen.de/wiki/index.php/I2C_Chip-%C3%9Cbersicht

[2] www.elektor.fr/logic-analyzer-scanaquad-sq100

[3] www.elektor.fr/stemlab-125-10-starter-kit

[4] http://dangerousprototypes.com/docs/Open_Bench_Logic_Sniffer

[5] http://ols.lxtreme.nl/

[6] http://dangerousprototypes.com/docs/Bus_Pirate

[7] www.watterott.com/en/Bus-Pirate

Remerciements

Je remercie mes (anciens) collègues Franz Otte et Michael Kleineberg pour leurs indications et conseils concernant le matériel, ainsi que mes (anciens) collègues Reinhard Bernhardi-Grisson, Norbert Bandzius et Thomas Schlüssler pour leur relecture de mon texte et leurs remarques.

PROJET DE LECTEUR

passerelle IoT et nœuds sans fil

1ère partie : le matériel

Il manque toujours une fonction spécifique à un système de domotique du commerce. C'est pourquoi l'auteur a conçu son propre système où les différents nœuds (*nodes*) communiquent sans fil avec la passerelle (*gateway*) centrale. Cette dernière utilise le protocole MQTT pour envoyer des données de mesure à un serveur OpenHAB, qui les traite et les supervise. Ensuite, plusieurs autres utilisateurs ont repris son projet et l'ont étoffé. En deux articles, l'auteur décrit les éléments les plus importants de son système – des informations plus détaillées et le logiciel associé sont disponibles gratuitement sur l'internet.

Hennie Spaninks (Pays-Bas)

Les systèmes domotiques font rêver. C'est magique de voir les lampes s'allumer et s'éteindre toutes seules, de constater que le chauffage « sait » quand vous êtes chez vous ou de recevoir un message qui signale un quidam à la porte. On trouve dans le commerce de nombreux systèmes, mais ils « souffrent » en général des défauts suivants :

- Ils sont souvent unidirectionnels ; on peut envoyer une instruction, sans avoir la certitude qu'elle est arrivée. Si on lit sous une lampe, pas de problème, mais lorsqu'on est en vacances à l'étranger, savoir que tout va bien rassure.
- Souvent, ils ne sont pilotables qu'avec une appli spécifique : chaque fabricant a la sienne et les protocoles utilisés sont loin d'être compatibles. D'où un ordinophone encombré d'une pléthore d'applis.
- L'attrait de la domotique est qu'elle permet d'automatiser les choses : envoyer des instructions en réponse à la valeur de mesure d'un capteur. Comme la plupart des systèmes disponibles dans le commerce ne sont pas compatibles, il faut l'intervention d'un tiers (*IfThisThenThat* par ex.) pour qu'il se passe quelque chose.
- Les systèmes sans fil font souvent appel au Wi-Fi. Dans les maisons en béton armé, la portée est limitée.
- La sécurité est loin d'être optimale : risque de voir le voisin allumer vos lampes.

Voilà suffisamment de motifs pour retrousser ses manches et construire son propre système domotique !

Panorama

Pour les capteurs et actionneurs (« nœuds terminaux », *end nodes*), nous optons de préférence pour des versions abordables et frugales en énergie. La commande centrale doit bien entendu reconnaître les protocoles réseau standard tels que TCP/IP.

C'est pourquoi nous avons opté pour un système à deux niveaux : les nœuds terminaux, une passerelle et une unité

de commande centrale. On en retrouve la structure dans le synoptique de la **figure 1**.

- Pour la communication avec les nœuds terminaux, nous utilisons une connexion sans fil en duplex à fréquence relativement basse. En Europe, 433 MHz et 868 MHz sont les fréquences éligibles. Choix du 868 MHz, la bande de 433 MHz est assez encombrée. La société HopeRF fournit des émetteurs-récepteurs (*transceiver*) pilotables via un bus SPI qui travaillent sur cette fréquence ; nous avons choisi la version à la puissance d'émission la plus élevée, le RFM69HW. Ce module embarque un cryptage matériel, la sécurité des données est donc garantie.
- Le RFM69HW n'a pas de pile TCP/IP, d'où l'échange, via la liaison radio, d'un bloc de données fixe de 66 octets. Il nous faut une passerelle qui traduise les données du réseau sans fil vers TCP/IP et vice versa.
- Pour la distribution des données du « producteur » (nœuds de capteurs) au « consommateur » (ordiphone par ex.), nous optons pour un protocole standard différent qui tourne sous TCP/IP : MQTT, un protocole de messagerie conçu pour envoyer de façon simple des messages courts à de multiples clients. Comme MQTT est un standard établi, les flux de données de notre système de domotique sont utilisables avec d'autres systèmes MQTT.
  MQTT requiert un serveur qui sert de courtier (*broker*, centrale de messages). Un client (ordiphone par ex.) peut s'abonner à un flux de données pour recevoir les messages véhiculés par ce flux. Pour réaliser le courtier MQTT, nous utilisons Mosquitto [1], une implémentation *open source* d'un courtier MQTT, disponible pour différentes plateformes (pour Raspberry Pi entre autres).
- Notre centrale domotique devrait pouvoir être utilisée avec les équipements de tiers. Elle doit donc en plus être capable de communiquer par le biais d'autres protocoles. Nous avons opté pour un système basé sur OpenHAB [2]. Ce logiciel *open source* peut convertir des messages de protocoles différents vers un flux de messages universel. On peut ainsi également piloter des lampes Hue.
- OpenHAB dispose d'une appli standard (Android et iOS) qui, via l'internet, communique de manière sécurisée avec votre propre système OpenHAB. Aucun souci donc de pare-feu ou de cryptage au niveau du lien reliant notre téléphone à notre système domotique. OpenHAB est accessible sur le réseau local via une interface web.
- OpenHAB comporte une base de données sous-jacente ; ceci permet de convertir des valeurs de mesure en graphiques simples. Il est facile, sous OpenHAB, de créer des lignes (d'instruction) et des scripts, pour automatiser des processus. OpenHAB dispose aussi d'interfaces externes pour les messages d'alarme par courriel ou via l'appli OpenHAB.
- Nous utilisons des composants disponibles et bon marché. Arduino nous servira de contrôleur des nœuds sans fil et un Raspberry Pi de plateforme pour Mosquitto et OpenHAB.

Protocole pour les nœuds terminaux

Plusieurs fonctions sont définies dans les nœuds terminaux : lire des capteurs de température et d'humidité, des interrupteurs, des détecteurs PIR, voire piloter des relais et des écrans LCD.

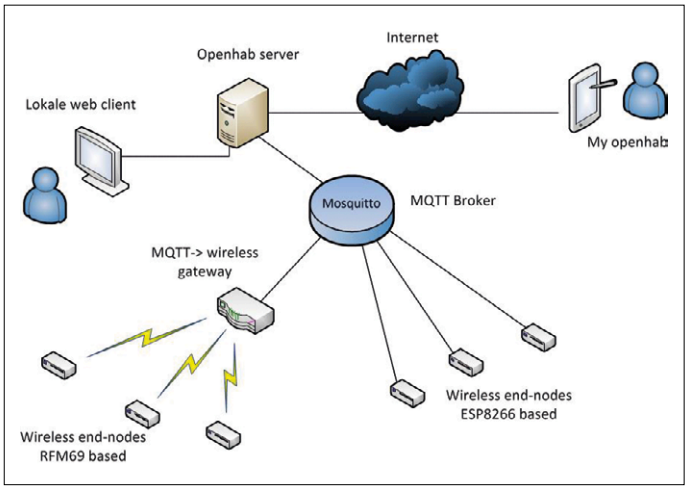


Figure 1. Synoptique du système de domotique.

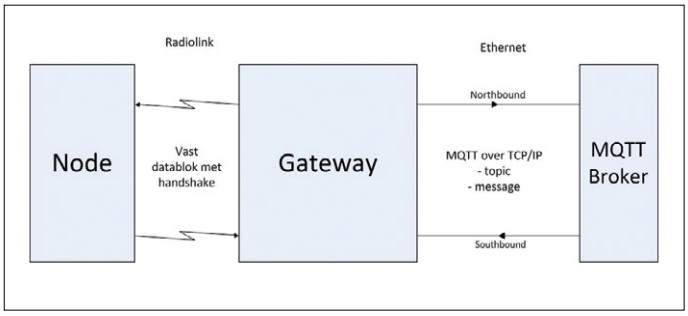


Figure 2. Schématisation des flux de données véhiculés par la passerelle.

Tableau 1. Identificateurs d'appareil (*Device identifiers*)

Plage des DevID	Fonction
0-15	Appareils système
16-31	Sortie binaire (relais, lampe)
32-39	Appareils à sortie de type entier (gradateur, MLI)
40-47	Entrée binaire (interrupteur, module PIR)
48-63	Entrée décimale (température, humidité)
64-71	Entrée entière (clavier, interrupteur)
72	Transfert de chaîne (affichage LCD, lecteur RFID)
73-90	Pour usage ultérieur
90-99	Messages d'erreur

Les nœuds terminaux sont en mesure de fournir des valeurs de mesure à intervalle régulier. Il est possible aussi, sur demande, d'envoyer ces valeurs (après une instruction de lecture).

La **figure 2** montre le flux de données par la passerelle. Un nœud terminal peut être programmé pour différentes fonctions. Pour veiller à ce que la passerelle gère de façon correcte le trafic de chaque nœud, on a effectué une classification fixe des fonctions (*device* = appareil). En fonction du code d'appareil (DevID), la passerelle sait comment traiter les données entrantes. Le **tableau 1** donne un aperçu des DevID reconnus par la passerelle. Les *devices* 0 à 15 sont des fonctions système. Le **tableau 2** en donne la description. Pour finir, le **tableau 3** indique les messages d'erreur que produit la passerelle.

Tableau 2. Fonctions système			
DevID	**Nom**	**RW**	**Fonction**
00	Uptime	R	Minutes depuis démarrage du nœud
01	TxInterval	RW	Intervalle d'envoi en secondes (0 = pas de transmission périodique)
02	RSSI	R	Intensité du signal radio
03	Version	R	Version du logiciel du nœud terminal
04	Voltage	R	Tension de la pile
05	ACK	RW	Drapeau de confirmation des instructions émises
06	Toggle	RW	Drapeau de fonction bascule du bouton-poussoir du nœud terminal
07	TImer	RW	Drapeau de fonction *timer* du bouton-poussoir du nœud terminal
08	Btnpress	RW	Drapeau d'envoi du message « bouton pressé »
09	TXreply	R	Nombre de répétitions requises sur la liaison radio

Tableau 3. Messages d'erreur		
Erreur ID	**Nom**	**Description**
90	Link error	La liaison radio est interrompue.
91	Syntax error	Le message MQTT comporte une erreur de syntaxe.
92	Invalid device	Le DevID adressé n'existe pas dans le noeud terminal.
99	Wakeup	Message envoyé lors du démarrage du nœud.

Tableau 4. Exemples MQTT		
Sujet	**Message**	**Description**
home/rfm_gw/sb/node02/dev16	ON	Allume la LED du nœud 02.
home/rfm_gw/sb/node02/dev16	READ	Demande l'état de la LED du nœud 02.
home/rfm_gw/sb/node03/dev01	300	Change l'intervalle TX du nœud de 3 à 5 min.
home/rfm_gw/sb/node03/dev01	0	Désactive les transmissions périodiques du nœud 3.
home/rfm_gw/sb/node18/dev48	READ	Lit la température du nœud 18.
home/rfm_gw/sb/node05/dev02	READ	Lit l'intensité du signal radio du nœud 5.
home/rfm_gw/sb/node05/dev03	READ	Lit la version du logiciel du nœud 5.

Messages MQTT

Le protocole MQTT travaille avec des abonnements, appelés *topics* (sujets) (cf. [3]). Si un client est abonné à un sujet, le courtier veillera à ce que tous les messages présents dans ce sujet lui soient livrés.

Le sujet MQTT utilisé a le format :

`home/rfm_gw/direction/nodeID/deviceID`

- `direction` indique ici le sens du flux de données. `nb` (*Northbound*) signifie du nœud vers serveur OpenHAB (valeurs de capteur), `sb` (*Southbound*) de OpenHAB vers le nœud (instructions).

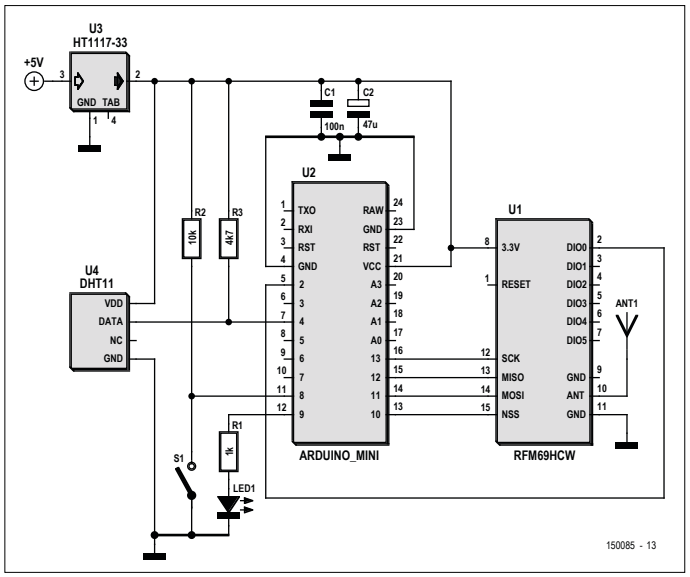


Figure 3. Schéma d'un nœud terminal.

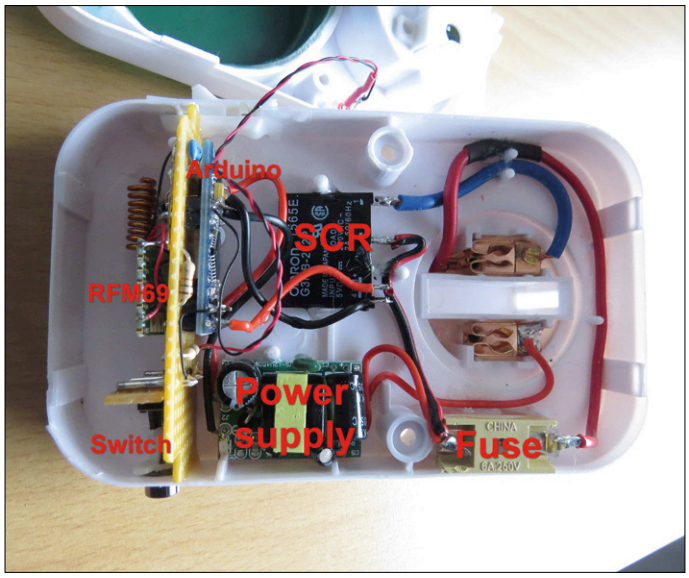


Figure 4. Nœud DIG dans un boîtier enfichable.

- `nodeID` est l'**ID**entifiant du nœud. Chaque nœud a un identifiant unique (adresse) déterminé au cours de la compilation du code. La passerelle a toujours le `nodeID` 01.
- `deviceID` est l'identifiant de la fonction, comme décrit plus haut.

Le message MQTT (*playload* = charge utile) dépend de l'appareil et de la direction :

- *Southbound* : le message contient des instructions pour le nœud (ON, OFF, READ) ou des valeurs à remettre au nœud (le nombre de secondes de l'intervalle de transmission par ex.).
- *Northbound* : le message contient la valeur du paramètre associé au DevID.

La passerelle s'abonne à tous les messages *Southbound* pour tous les nœuds du réseau avec le sujet suivant avec un caractère générique (*wildcard*) :

```
home/rfm_gw/sb/#
```

OpenHAB tient bien sûr à recevoir tous les messages venant des nœuds et s'abonne donc au sujet :

```
home/rfm_gw/nb/#
```

Le **tableau 4** donne quelques exemples de messages MQTT.

Nœud terminal

Le cœur du nœud terminal est un Arduino. La conception tient compte des conditions connexes suivantes :

- Le module RFM69HW travaille sous 3,3 V. Les entrées ne supportent pas plus de 3,9 V. Pour ne pas compliquer les choses, la carte Arduino Pro Mini est utilisée sous 3,3 V et peut être reliée au RFM69HW sans adaptation de niveau. Alternative : la carte Arduino Buono R3 qui peut être commutée à 3,3 V.
- À l'émission, le module RFM consomme brièvement un courant intense (130 mA) ; il faut donc une alimentation de capacité suffisante.
- La communication entre l'Arduino et le RFM69HW se fait via le bus SPI ; on utilise pour cela les broches standard de l'Arduino.
- Le RFM69HW ne doit pas être utilisé sans antenne ; dans la plupart des cas, un fil de 8,6 cm fera l'affaire.

La **figure 3** donne le schéma d'un nœud terminal qui comporte un capteur de température/humidité DHT11 relié à la broche 4 de l'Arduino. Un bouton-poussoir attaque la broche numérique 8 et une LED est reliée à la broche 9.
Le module RFM est couplé à l'Arduino via le bus SPI (SCK, MOSI, MISO et NSS). La tension d'alimentation de 3,3 V fait appel à un régulateur AMS1117 épaulé par les condensateurs de découplage requis pour encaisser les crêtes de courant.
Le logiciel chargé dans l'Arduino détermine la fonction du nœud terminal. À l'heure actuelle, les nœuds terminaux suivants sont fonctionnels :

- DHT est un nœud terminal avec des capteurs de température et d'humidité. Le nœud comporte une sortie numérique et un bouton-poussoir.
- DIG est une version simple du nœud DHT. Il se résume à un bouton-poussoir et une sortie numérique.
- RFID est un nœud doté d'un lecteur RFID. En cas de détection d'un système RFID, le nœud envoie l'ID(entifiant) RFID au serveur OpenHAB.
- LCD est un nœud avec un écran LCD. Il est possible depuis OpenHAB d'envoyer des lignes de texte à l'écran.
- RC est un nœud équipé d'un émetteur de 433 MHz. Il est possible, à partir de ce nœud, de commander les interrupteurs du système de domotique *KlikAanKlikUit* [4].

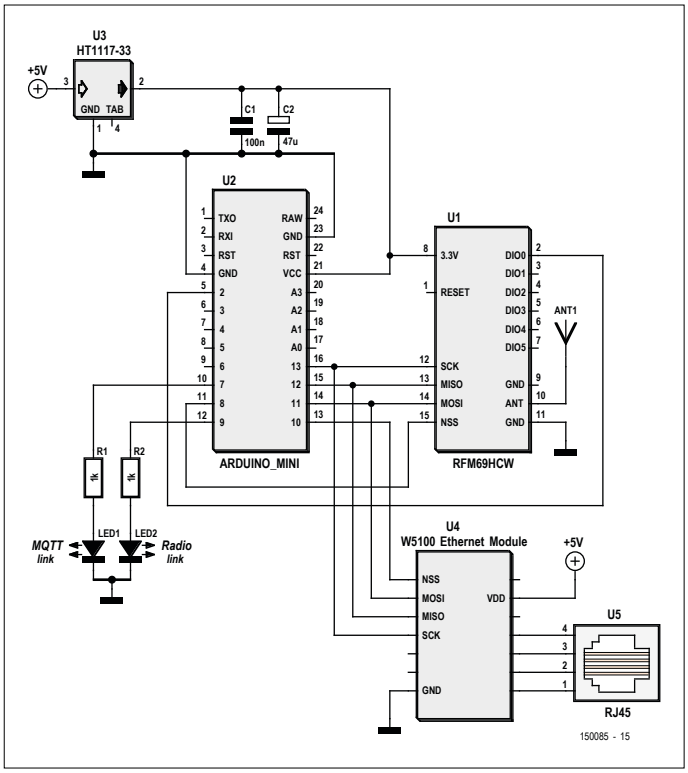


Figure 5. Schéma de la passerelle.

Le logiciel est modifiable aisément pour y ajouter des fonctions ou en modifier. La structure du nœud terminal dépend beaucoup de la fonction. On pourra par ex. utiliser un programmateur mécanique bon marché pour y caser un nœud DIG, solution qu'illustre la **figure 4**.

La passerelle (*gateway*)

La passerelle est constituée d'un Arduino, d'un RFM69HW et d'un module Ethernet W5100. On en retrouve le schéma en **figure 5**. La passerelle aussi utilise un Arduino qui travaille sous 3,3 V.
On voit sur le schéma que les RFM69 et W5100 se partagent le bus SPI dans la passerelle. Les signaux SCK, MOSI et MISO sont connectés en parallèle. On fait appel, pour la sélection du bon appareil sur le bus, à différents signaux *Slave Select* (SS) : la broche 8 pour le RFM69 et la broche 10 pour le module Ethernet.

À nouveau, c'est un AMS1117 qui se charge de la tension d'alimentation de 3,3 V. Les deux LED indiquent l'état de la passerelle.

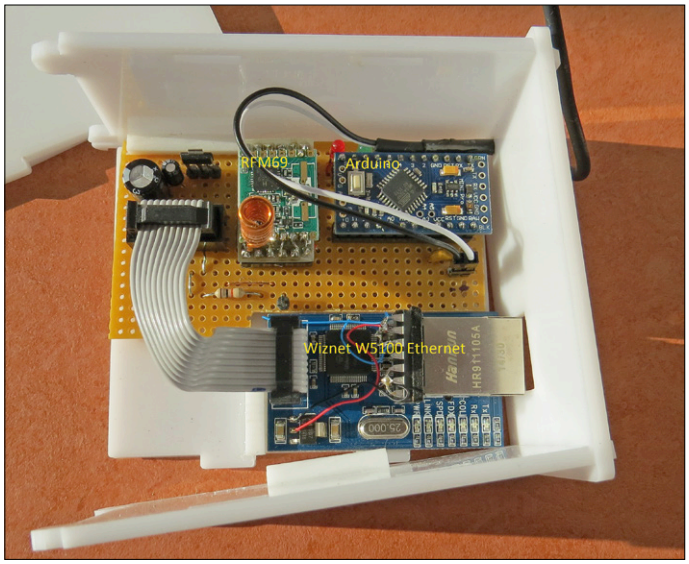


Figure 6. Voilà la passerelle telle que réalisée par l'auteur.

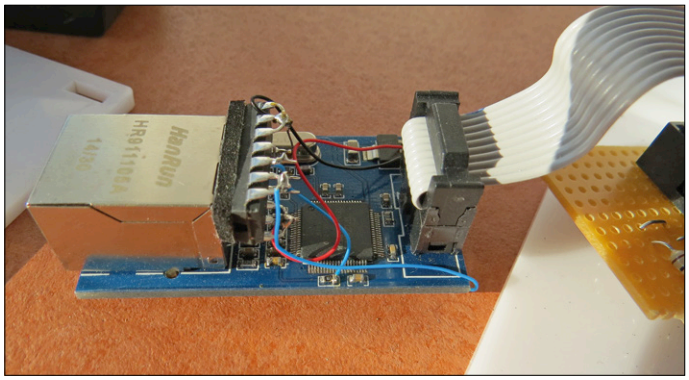


Figure 7. Le 4011 est fixé au boîtier du connecteur réseau et produit le signal SEN pour le W5100 à partir du signal CS.

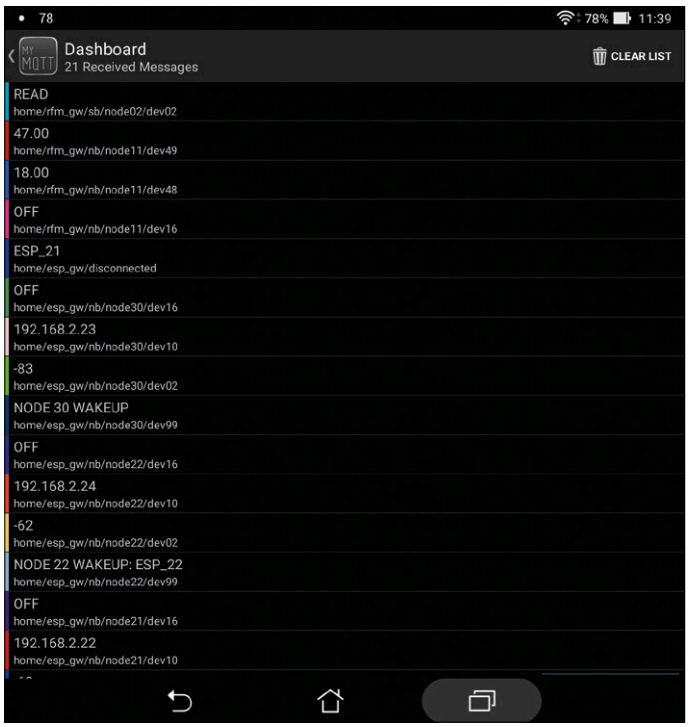


Figure 8. Cette capture d'écran montre un exemple du trafic MQTT.

La connexion Ethernet peut se faire à l'aide d'un module distinct. Si on a choisi une carte Arduino Buono, on pourra utiliser un *shield* Ethernet Wiznet. Une passerelle par système de domotique suffit ; c'est pourquoi l'auteur a réalisé sa passerelle sur une carte de prototypage (**fig. 6**).
Il y a deux aspects auxquels il faut veiller lors de la réalisation de la passerelle :

Le bogue W5100 (*W5100 bug*)
Le matériel du W5100 ne convient pas directement à une utilisation de plusieurs appareils sur le bus SPI. Lorsque le W5100 n'utilise pas le bus, les signaux de bus doivent être « libérés » pour qu'un autre appareil puisse utiliser le bus. En standard, le W5100 ne le fait pas. On résout ce problème en alimentant le signal SEN par le biais du signal CS après passage par un inverseur (cf. [5] pour plus de détails), ledit inverseur a été intégré dans les versions ultérieures du *shield* W5100. Ce n'était pas le cas du module acheté par l'auteur sur Ebay, il l'a donc rajouté sous la forme d'un circuit intégré (4011) (cf. **fig. 7**). Cette figure montre également qu'il est facile de connecter le signal SEN au travers d'une résistance de forçage au niveau haut (*pull-up*) sur le circuit imprimé. Ne pas oublier de mettre à la masse les entrées inutilisées du 4011 !

Court-circuit par le biais de l'embase ICSP
En cas d'utilisation d'un Arduino de format standard (tel que le Buono) en combinaison avec un *shield* W5100, l'Arduino se voit alimenté en 5 V par le *shield* Ethernet au travers de l'embase ICSP. La tension d'alimentation de 3,3 V de l'Arduino est ainsi rehaussée avec le risque d'endommager le RFM69HW. Pour éviter cela, supprimez la broche VCC de l'embase ICSP ou repliez-la vers l'extérieur, ce qui empêche ainsi tout risque de contact.

À suivre…
Dans le prochain article, nous nous intéresserons au logiciel de la passerelle et des nœuds ainsi qu'à la configuration de Mosquitto et d'OpenHAB. Les impatients trouveront le logiciel en [6]. Il faut disposer de Mosquitto pour tester la passerelle et un client MQTT s'avérera utile lors des tests. Sur Android, l'appli *MyMQTT* fait parfaitement l'affaire. Sur Windows, on utilisera *Chrome Lens* ou *MQTT.FX*. La **figure 8** montre à quoi ressemblent les messages.
Ce projet a déjà été réalisé en plusieurs exemplaires. Le forum [7] sert de plateforme de discussion pour des améliorations, en cas de problèmes et autres expériences.

(150085 - version française : Guy Raedersdorf)

Liens
[1] https://mosquitto.org
[2] www.openhab.org
[3] www.elektormagazine.fr/news/mon-voyage-dans-le-nuage-iot-13-un-petit-capteur
[4] www.klikaanklikuit.nl
[5] http://john.crouchley.com/blog/archives/662
[6] https://github.com/computourist/RFM69-MQTT-client
[7] http://homeautomation.proboards.com/board/2/openhab-rfm69-based-arduino

PROJET DE LECTEUR

passerelle IoT et nœuds sans fil

2e partie : le logiciel

L'auteur a cherché un système domotique commercial qui réponde à ses exigences. Resté sur sa faim, il a décidé de concevoir le sien : dans la bande des 868 MHz, plusieurs « nœuds terminaux » (capteurs et actionneurs) communiquent avec une passerelle centrale. Celle-ci utilise le protocole MQTT pour envoyer des données de mesure à un serveur OpenHAB, qui les traite et les supervise [1]. Dans le 1er article [2], nous avons abordé le matériel. Ici nous nous intéressons au logiciel qui anime le tout.

Hennie Spaninks (Pays-Bas)

Nous traiterons ici du code pour les nœuds terminaux, la passerelle et le serveur sur RPi. L'auteur a écrit lui-même les croquis (*sketches*) Arduino et il a bien entendu fait appel à diverses bibliothèques à code source ouvert.

Nœud terminal
Croquis Arduino pour le nœud terminal : du code pour piloter la liaison radio, lire les capteurs et commander les sorties.

Passerelle (*gateway*)
Croquis Arduino pour la passerelle : du code pour piloter la liaison radio et décoder les messages. Un client MQTT échange les messages avec le courtier (*broker*).

Serveur domotique (Raspberry Pi)
Sur le Raspberry Pi tournent à la fois le courtier MQTT (Mosquitto) et le serveur domotique OpenHAB.
Pour compiler le code Arduino, nous utilisons l'EDI Arduino standard, épaulé par les bibliothèques mentionnées plus loin. Pour le Raspberry Pi, on utilisera de préférence la version la plus puissante (RPi 3), encore que cela marche aussi avec le Raspberry Pi 2. Notre SE sera Raspbian.

Nœud terminal
Le code pour les différents nœuds terminaux est disponible en [3]. Le noyau du code est identique pour tous les nœuds terminaux. Nous prenons comme exemple le nœud terminal DHT [4]. Ce nœud comporte une entrée numérique (un bouton-poussoir) sur l'entrée 8, une sortie numérique (relais/LED) sur la broche 9 et un capteur d'humidité/température sur la broche 4. Si on presse le bouton, le nœud peut exécuter localement deux fonctions :

- Démarrer un temporisateur (*timer*) et mettre la sortie au niveau haut. Réinitialiser la sortie une fois la durée du temporisateur écoulée. La durée de la temporisation est réglée avec l'appareil 7 (cf. tableau 2 dans [2]). Un « 0 » bloque la fonction.
- Basculer la sortie après pression du bouton-poussoir. Ceci

permet d'activer et de désactiver la sortie manuellement en local. Pour ce faire, l'appareil 6 doit être sur « ON ».

Indépendamment de ces deux fonctions, chaque changement de niveau de l'entrée numérique (broche 8) provoque l'envoi d'un message d'état via l'appareil 40. Les valeurs du capteur d'humidité/température peuvent être lues via l'appareil 48/49. Des transmissions périodiques sont aussi possibles.

Ce code fait appel à trois bibliothèques :

- Bibliothèque RFM69 de Felix Ruso [5].
- Bibliothèque SPI pour la commande du bus SPI.
- Bibliothèque DHT pour la lecture du capteur DHT11.

Le code commence par quelques paramètres de configuration (à partir de la ligne 60) :

```
#define NODEID 2
   // unique node ID within the closed network
#define GATEWAYID 1
   // node ID of the Gateway is always 1
#define NETWORKID 100
   // network ID of the network
#define ENCRYPTKEY “xxxxxxxxxxxxxxxx”
   // 16-char encryption key; same as on Gateway!
//#define DEBUG
   // uncomment for debugging
#define VERSION “DHT V2.2”
   // this value can be queried as device 3
```

`NODEID` attribue au nœud terminal une adresse unique sur le réseau sans fil. Cet ID(entificateur) est également utilisé dans le sujet (*topic*) MQTT pour s'adresser au nœud. Le `nodeID` de la passerelle est toujours « 1 ». Un nœud terminal ne communique qu'avec la passerelle ; il n'y a donc pas de trafic internœuds terminaux.
`NETWORKID` permet de définir des réseaux fermés. Seuls les nœuds du même réseau peuvent communiquer entre eux. Dans ce projet, nous utilisons un réseau fermé, de sorte que l'ID du réseau doit être le même sur la passerelle et tous les nœuds terminaux.
`ENCRYPTKEY` est la clé de cryptage du trafic sans fil. Cette clé comporte seize caractères et doit être la même pour tous les nœuds du réseau.
Le paramètre `DEBUG` est une bascule d'activation du mode de débogage. Si ce paramètre n'est pas défini (en faisant de cette ligne un commentaire par l'ajout de « // »), aucune information de débogage n'est produite. La suppression de « // » active le débogage, des messages d'état sont alors envoyés sur le port série.

Une commande adéquate du module RFM69 requiert quelques paramètres supplémentaires :

```
#define FREQUENCY RF69_868MHZ
#define IS_RFM69HW    // uncomment only for RFM69HW!
#define ACK_TIME 50  // max # of ms to wait for an ack
```

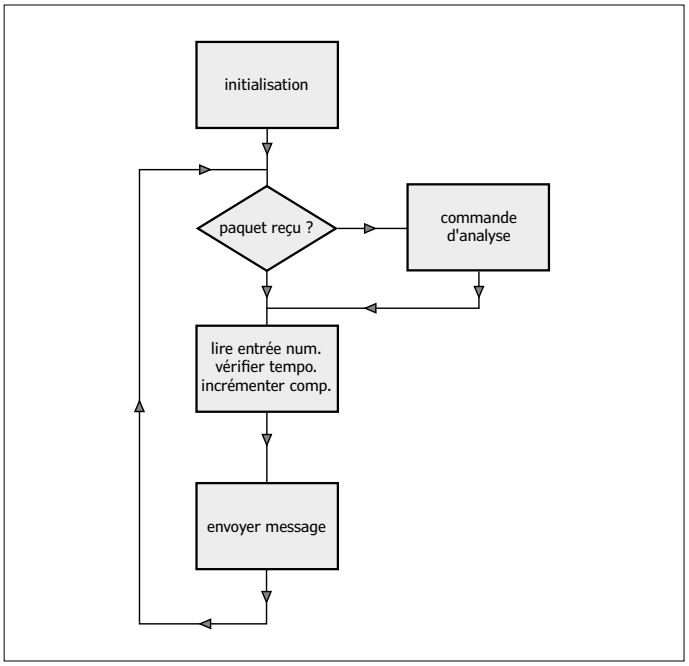


Figure 1. Ordinogramme du nœud terminal.

`FREQUENCY` indique la fréquence utilisée (en Europe, 433 ou 868 MHz). `IS_RFM69HW` indique que vous utilisez la version forte puissance. `ACK_TIME` donne la durée d'attente maximale, après envoi d'un message, avant réception d'un acquittement.

À partir de la ligne 78, on trouve quelques définitions pour le capteur DHT, les broches et variables utilisées. Les commentaires dans le code source indiquent les fonctions de ces variables. À partir de la ligne 135 suit l'initialisation du programme (`setup()`), dans laquelle est définie la broche de sortie et initialisé le module sans fil. Voir l'ordinogramme de la **figure 1**.
On commence (ligne 167) par voir si le module RFM69 a reçu un paquet radio. Si c'est le cas, il y a appel de la fonction `parseCmd`. En fonction du *deviceID* et de l'instruction de lecture/écriture dans le bloc de données reçu, le drapeau d'envoi adéquat est levé. Après cela, on vérifie s'il y a eu pression du bouton et expiration de la temporisation. Il y a ensuite incrémentation du compteur de temps de fonctionnement. Si l'intervalle de transmission périodique est écoulé, il y a positionnement des drapeaux d'envoi de ces paramètres.
Enfin, à la ligne 235, il y a appel de la fonction `sendMsg`. Selon le drapeau d'envoi levé, `sendMsg` prépare les données requises pour la transmission radio, réinitialise le drapeau d'envoi et envoie le paquet après un appel de la fonction `txRadio()`.
En cas de demande de la température ou de l'humidité (appareils 48 et 49), le module DHT est interrogé (lignes 478 et 485).

Passerelle (*gateway*)

La passerelle remplit les fonctions suivantes :

- Réception et décryptage d'un message MQTT, création et envoi du paquet radio.
- Réception et décryptage d'un message radio, création et publication du message MQTT.

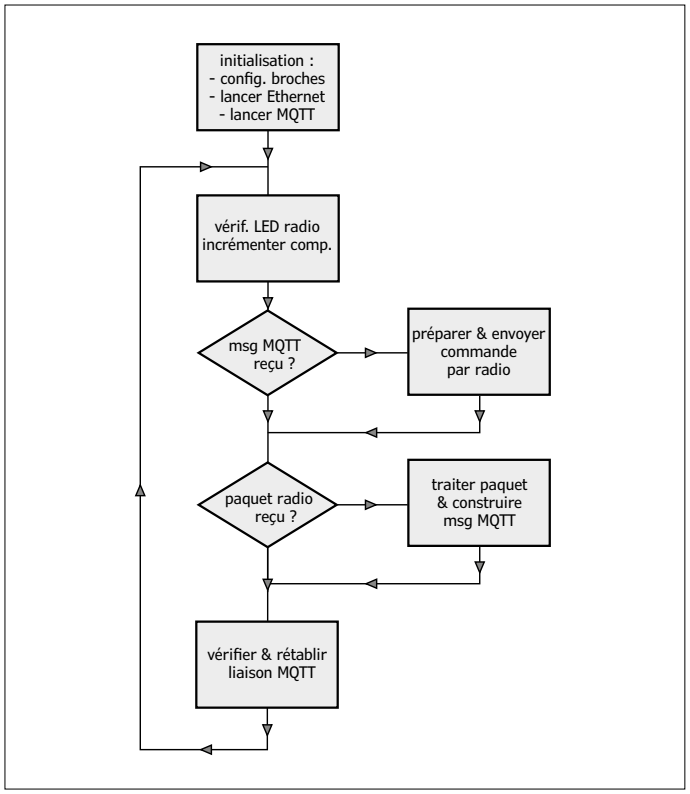


Figure 2. Ordinogramme de la passerelle.

- Réception, décryptage et réponse aux demandes sur la passerelle elle-même, telles que temps de fonctionnement, version, etc.

La passerelle est reliée à une connexion Ethernet fixe et reçoit ses informations de réseau par DHCP. La passerelle comporte deux LED. La LED radio indique l'activité de la liaison sans fil. La LED MQTT s'allume lorsqu'il existe une liaison avec Mosquitto.

Comme le programme de la passerelle occupe beaucoup de mémoire, la fonction de débogage est subdivisée en deux parties : l'une pour déboguer la liaison sans fil et l'autre pour déboguer la connexion au courtier MQTT. Il ne peut y avoir qu'une seule partie active, sinon le programme ne se compile pas.
La description du programme repose sur la dernière version (2.4), disponible en [6].

Le croquis de la passerelle utilise les bibliothèques suivantes :

- *RFM69*
- *SPI*
- *Ethernet*
- *PubSubClient* [7]

Lors de la compilation du croquis de la passerelle, nous modifions quelque peu des paramètres par défaut :

- Dans la passerelle, le bus SPI est utilisé par deux appareils. Les deux appareils utilisent « 10 » comme adresse d'appareil par défaut. Nous choisissons le code d'appareil 8 pour le module RFM et 10 pour le module Ethernet (voir configuration à la ligne 80 du code de la passerelle).
- Le module RFM utilise des interruptions pour indiquer la réception d'un paquet radio. Tant que le pilote Ethernet (*W5100.h*) est actif sur le bus SPI, le traitement de ces interruptions se fait incorrectement d'où blocage de la passerelle. Nous utilisons la solution proposée par Martin Harazinov et adaptons le fichier *w5100.h* pour résoudre ce problème (cf. [8]). Dans le fichier *w5100.h* (à trouver dans *../Documents/Arduino/libraries/Ethernet/src/utility*), on ajoute aux lignes suivantes (à partir de la ligne 342) des déclarations `cli()` et `sei()`.

```
#else
inline static void initSS()   { DDRB |= _BV(2); };
inline static void setSS()    { cli(); PORTB &= ~_
   BV(2); };
inline static void resetSS()  { PORTB |= _BV(2);
   sei(); };
#endif
```

  Cela permet de bloquer les interruptions tant qu'il y a du trafic Ethernet sur le bus SPI.

Le croquis de la passerelle commence par un bloc de déclaration très similaire à celui du nœud terminal. On y définit tous les paramètres de la liaison sans fil. C'est ensuite le tour des paramètres suivants pour Ethernet :

- `Ip`, pour l'adresse IP de la passerelle. C'est un repli (*fallback*) ; la passerelle utilise DHCP et reçoit donc une adresse dynamiquement.
- `Mac`, pour l'adresse MAC de la connexion Ethernet. Remarque : celle-ci doit être unique dans un sous-réseau IP, donc attention s'il y a plusieurs Arduino dans le réseau.
- `Mqtt_server`, l'adresse du serveur Mosquitto (nous en reparlerons plus loin).

La **figure 2** montre l'ordinogramme du code de la passerelle. L'initialisation commence à la ligne 135. On y configure les broches de sortie des deux LED, lance la connexion Ethernet et démarre la connexion avec le courtier MQTT.

Le client MQTT est initialisé à la ligne 129. On y demande un abonnement, défini dans la fonction `mqtt_subs` (ligne 382). L'objet de l'abonnement est *home/rfm_gw/sb/#* (ligne 121) pour recevoir tous les messages *Southbound*. Dès qu'un message MQTT arrive, la variable `mqttToSend` est mise à *TRUE*. Dans le programme principal, on commence par voir si la LED du sans fil doit être éteinte puis on augmente le temps de fonctionnement de la passerelle.
À la ligne 223, on vérifie avec la variable `mqttToSend`, s'il faut envoyer des messages sur le réseau sans fil, fonction dont se charge `sendMsg` (ligne 248). Le cas échéant, l'envoi de la transmission se répète, au maximum cinq fois, toutes les demi-secondes, jusqu'à ce que la réception du paquet soit confirmée par le nœud terminal.
Ensuite, à la ligne 225, on contrôle si l'on a reçu un paquet d'un des nœuds terminaux. Si c'est le cas, on appelle la fonction `processPacket` (ligne 288). Un message MQTT de la passerelle se réfère toujours à *Northbound*. Il est constitué de `nodeID` et `deviceID`, et stocké dans la chaîne `buff_topic` à la

ligne 303. En fonction du `deviceID`, on détermine quelle donnée du paquet radio est valide (*integer*, *float*, *string* ou état), et quel est le formatage requis pour le message MQTT. Pour finir, le message MQTT se trouve dans la chaîne `buff_mess`. Enfin on vérifie à la ligne 227, si la liaison MQTT est toujours active. Si ce n'était pas le cas, on tente à la ligne 230 de la redémarrer, toutes les deux secondes. La LED MQTT est allumée (ligne 235) lors de l'établissement de la liaison.

OpenHAB

On voit en **figure 3** l'architecture d'OpenHAB qui repose sur un bus d'événement(s). Par le biais de ce bus, il y a échange d'événements entre les interfaces d'entrée et de sortie et le reste d'OpenHAB. Cette construction permet d'utiliser côte à côte plusieurs interfaces d'E/S (« *bindings* »). Il est possible ainsi de créer des « *bindings* » spécifiques pour différents protocoles. À l'heure actuelle il existe des « *bindings* » pour des compteurs intelligents, les systèmes audio SONOS, les lampes Hue de Philips, les thermostats Nest ou encore les voitures Tesla. La page [9] en donne un aperçu.

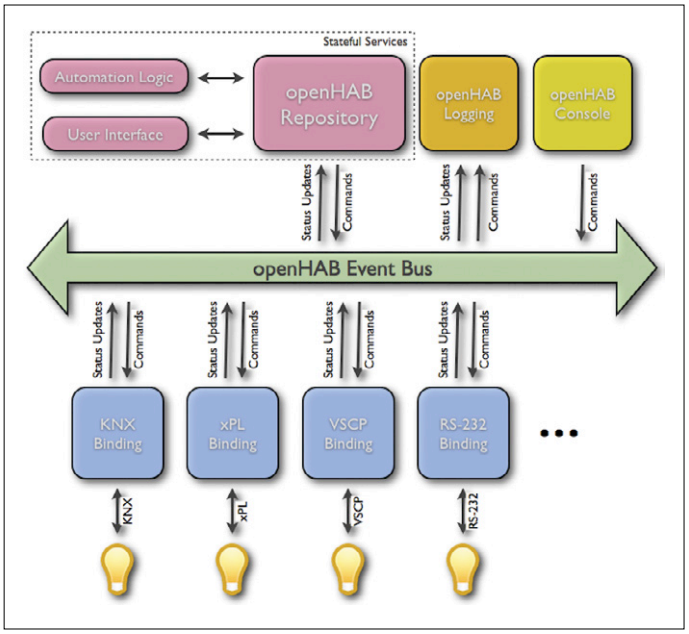


Figure 3. Architecture d'OpenHAB (source : OpenHAB.org).

Installation

Nous utilisons un Raspberry Pi 3 avec une carte SD de 16 Go ainsi que la version 2 d'OpenHAB. La façon la plus simple d'installer OpenHAB est d'utiliser OpenHABian. C'est une image avec un système Linux minimal combiné à des scripts d'installation. On pourra télédécharger cette image depuis [10]. Avec l'outil adéquat (WinDiskImager [11] par ex.), nous écrivons cette image sur une carte SD. Cette dernière est enfichée dans le Raspberry Pi et après le « *boot* », l'installation démarre. Cela peut durer un certain temps.
Au cours de cette installation, il y a également installation de *Samba* qui permet d'accéder aux dossiers de configuration d'OpenHAB par le réseau, ce qui facilite les changements de configuration.

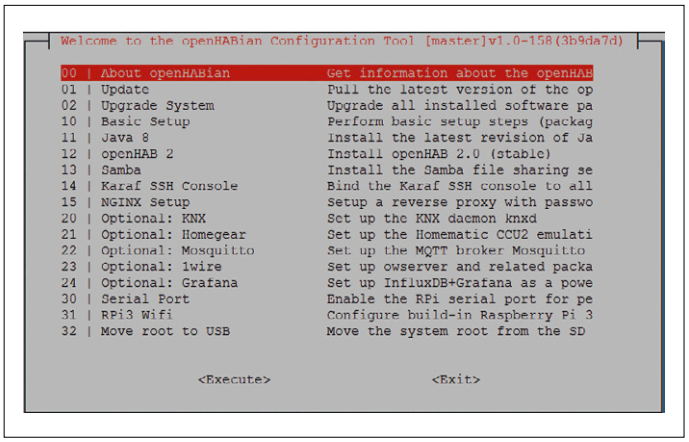


Figure 4. Configuration d'OpenHABian.

Lorsque l'installation est terminée, nous démarrons Mosquitto avec l'outil de configuration :

```
pi@openHABianPi: sudo openhabian-config
```

... et voyons apparaître, si tout se passe bien, la fenêtre de la **figure 4**. Le choix 22 installe Mosquitto. Il faut aussi que le Raspberry Pi reçoive toujours la même adresse IP. Nous y veillons en modifiant le fichier */etc/network/interfaces*. Il faut modifier `address` et `gateway` en fonction de la configuration souhaitée, par ex. :

```
auto eth0
    iface eth0 inet static
        address 192.168.2.7
        netmask 255.255.255.0
        gateway 192.168.2.254
```

Un redémarrage active le nouveau paramétrage.

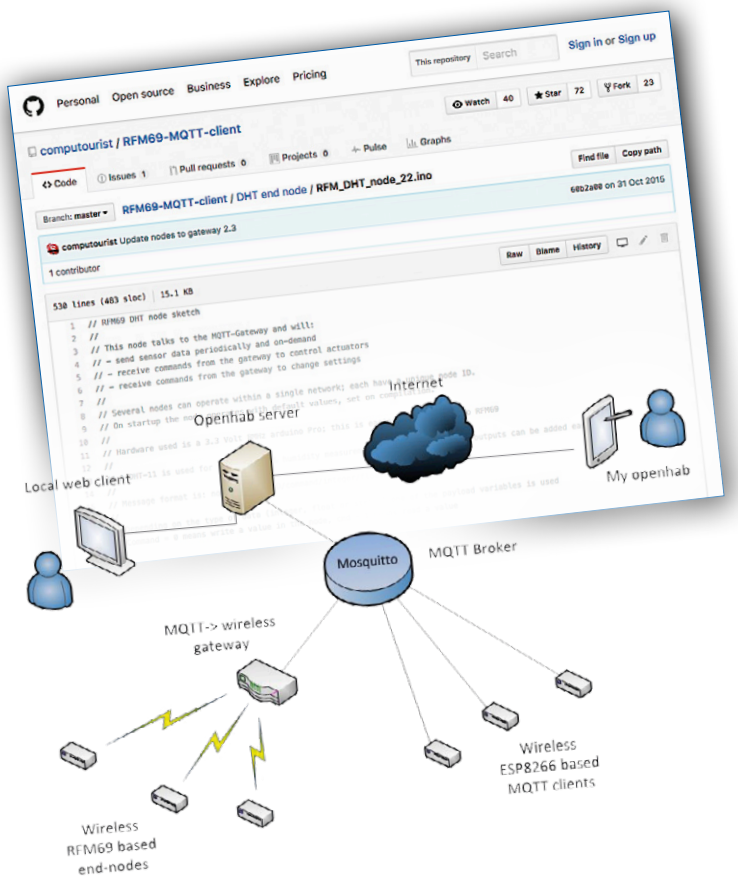
Configuration

Pour qu'OpenHAB fonctionne, il faut configurer correctement la liaison MQTT. Le fichier à modifier se trouve sur le RPi dans */etc/openhab2/services/mqtt.cfg*. L'URL du courtier (*broker*) est le seul paramètre à modifier comme suit :

```
# Define your MQTT broker connections here for use
   in the MQTT Binding or MQTT
# Persistence bundles. Replace <broker> with an ID
   you choose.
# URL to the MQTT broker, e.g. tcp://localhost:1883
   or ssl://localhost:8883
mosquitto.url=tcp://localhost:1883
```

C'est un renvoi vers l'hôte local (*localhost*) pour le serveur Mosquitto. Veillez à ce qu'ici `mosquitto` soit le nom (ID) du courtier. Dans la configuration d'objet (*item*) (cf. le texte plus loin), nous renvoyons à cet ID(entifiant) pour établir la communication. Le port 1883 est utilisé par défaut pour le trafic MQTT. OpenHAB utilise trois fichiers de configuration pour décrire un système :

- *Items* dans le dossier */etc/openhab2/items*, qui contient les définitions des paramètres utilisés et leur lien vers les dossiers *item*.



- *Sitemap* dans le dossier */etc/openhab2/sitemaps*, qui décrit la mise en page de l'interface utilisateur.
- *Rules* dans le dossier */etc/openhab2/rules,* qui indique les actions à effectuer (*scripting*).

À titre d'exemple, nous configurons un nœud DHT terminal avec le *nodeID* 2. Il doit envoyer périodiquement la température, l'humidité et l'état de sortie. Dans le dossier *item* (*/etc/openhab2/items*), nous créons un fichier *home.items* avec le contenu du **listage 1**. Le Wiki OpenHAB [12] explique la structure de la configuration d'un élément (*item*). Ensuite, dans le dossier */etc/openhab2/sitemaps*, nous créons le fichier *home.sitemap* avec le contenu suivant :

```
sitemap home label="MyHome"

Frame label="Controls" {
       Switch item=OUT2 label="Myhome lamp"
   icon="light" }

Frame label="Sensoren" {
       Text item=TEMP2
       Text item=HUM2
       Text item=RSSI2   }
```

La première ligne donne le nom de cette configuration (`home`), pour que lors de l'accès à OpenHAB par un navigateur ou une appli, il y ait chargement de la cartographie (*sitemap*) correcte. Ensuite on définit cadre pour superviser la sortie du nœud 2 et un autre pour l'affichage des température, humidité et intensité de champ. Enfin, nous créons un fichier *home.rules* dans */etc/openhab2/rules* comportant les lignes suivantes :

```
//
//  refresh rules
//

rule "refresh RSSI2"   // periodically refresh
   value for signal strength
    when
       Time cron "0 0/1 * * * ?"
   then
       sendCommand(getRSSI2, "READ")
end
```

Grâce à la commande `cron`, on envoie, une fois par minute, le message MQTT « READ » vers l'appareil 2 sur le nœud 2 aux fins d'obtenir la force de champ.

Il est temps de procéder aux premiers essais. Ouvrez un navigateur et allez à :

```
http://192.168.xx.xx:8080/basicui/app?sitemap=home
```

Il faut bien entendu remplacer l'adresse IP par l'adresse correcte. La fenêtre de la **figure 5** montre le résultat de cette instruction. Il se peut que toutes les valeurs ne soient pas disponibles (on a dans ce cas-là affichage d'un « - »), mais au bout d'un certain temps, après une première actualisation par le nœud, cela devrait être résolu.

Débogage

Et, si tout ne se passe pas comme prévu ?

MQTT fonctionne-t-il ?

Si la connexion MQTT entre la passerelle et le courtier fonctionne, la LED MQTT s'allume, ce qui indique que les messages transitent de la passerelle vers Mosquitto. Étape suivante : installer un client de test MQTT (MQTT.fx [13] ou MQTTLens [14], voir Chrome Web Store). MQTT.FX a un onglet *broker status*

Listage 1. Configuration d'un *item* OpenHAB pour un nœud terminal température/humidité de l'air.

```
Number RSSI2 «RSSI [%d dBm]» {mqtt=»<[mosquitto:home/rfm_gw/nb/node02/dev02:state:default]»}
String getRSSI2 «get RSSI» {mqtt=»>[mosquitto:home/rfm_gw/sb/node02/dev02:command:*:default]»}
Number TEMP2    "Temperature [%.1f °C]" {mqtt="<[mosquitto:home/rfm_gw/nb/node02/dev48:state:default]"}
Number HUM2    "Humidity [%.1f %%]"{mqtt="<[mosquitto:home/rfm_gw/nb/node02/dev49:state:default]"}
Switch OUT2 "attic node" <PushBtn> {mqtt=">[mosquitto:home/rfm_gw/sb/node02/
   dev16:command:ON:ON],>[mosquitto:home/rfm_gw/sb/node02/dev16:command:OFF:OFF]"}
```

pour demander des informations sur Mosquitto. Abonnez-vous au sujet *home/rfm_gw/#* et des messages devraient apparaître ; si tout est OK, il s'agit de messages provenant du nœud terminal. Sinon, il est temps de compiler la passerelle ou le nœud avec l'option *DEBUG* et de connecter un PC au port sériel. Démarrez PUTTY [15], paramétrez le port COM correct et réglez la vitesse à 115200 baud. La sortie du nœud/passerelle indiquera l'existence (ou non) d'une connexion et les valeurs en cours de transfert. L'envoi de commandes client MQTT au nœud (par ex. *READ* à */home/rfm_gw/node02/dev03*) montrera où bloque le flux de données.

Figure 5. Fenêtre de navigateur.

OpenHAB fonctionne-t-il ?

Si la connexion MQTT fonctionne, l'étape logique suivante consiste à voir du côté d'OpenHAB. L'examen des fichiers journaux (*log*) est instructif. L'instruction

```
> tail -f /var/log/openhab2/events.log
```

permet d'afficher les dix dernières lignes du journal d'événements. On devrait y voir des mises à jour entrantes venant du nœud ou de l'interface utilisateur. Petit coup d'œil aussi à */var/log/openhab2/openhab.log,* fichier où sont rangés les événements système d'OpenHAB.

Et après...

Votre première extension sera sans doute un accès à l'internet ; m*yopenhabl* le permet. OpenHAB a une appli qui communique avec votre propre système OpenHAB par l'internet via une passerelle sécurisée (voir [1]).
OpenHAB permet également de créer des graphiques des variations dans le temps des valeurs de mesure ; cela requiert d'activer une base de données locale et de spécifier les valeurs de mesure à mémoriser. On peut travailler localement, mais aussi accéder à des bases de données externes (voir [16]).
Le nœud décrit mesure la température et l'humidité et peut commuter une sortie numérique. L'utilisation d'un circuit électronique différent pour le nœud, et la modification du logiciel ouvrent d'autres perspectives. On trouvera sur Github [3] des solutions pour un LCD et un lecteur RFID [17]. On y décrit également un nœud qui commande depuis OpenHAB des interrupteurs du système de domotique *KlikAanKlikUit*.

Parallèlement à la solution décrite ici, une variante à base d'ESP8266 a été développée. Ce nœud établit une liaison par Wi-Fi avec le courtier MQTT et OpenHAB. On trouvera en [8] la description du projet.

Jetez donc un coup d'œil au forum [19] où différents utilisateurs présentent des solutions pour des détecteurs PIR, des volets roulants, des détecteurs de niveau et bien d'autres applications encore. ⏮

(160318 – version française : Guy Raedersdorf)

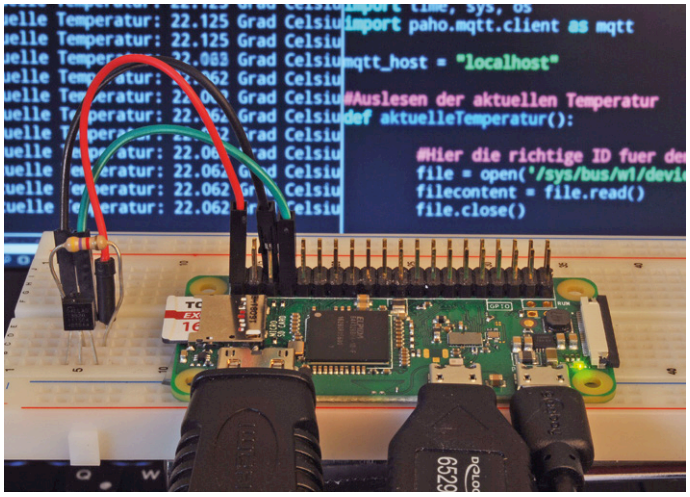
Liens

[1] www.myopenhab.org
[2] www.elektormagazine.fr/150085
[3] https://github.com/computourist/RFM69-MQTT-client
[4] https://github.com/computourist/RFM69-MQTT-client/blob/master/DHT%20end%20node/RFM_DHT_node_22.ino
[5] https://github.com/LowPowerLab
[6] https://github.com/computourist/RFM69-MQTT-client/blob/master/Gateway_2.4/RFM_MQTT_GW_24.ino
[8] http://harizanov.com/2012/04/rfm12b-and-arduino-ethernet-with-wiznet5100-chip
[7] https://github.com/knolleary/pubsubclient
[9] http://docs.openhab.org/addons/bindings.html
[10] http://docs.openhab.org/installation/openhabian.html
[11] https://sourceforge.net/projects/win32diskimager
[12] https://github.com/openhab/openhab1-addons/wiki/Explanation-of-items
[13] http://mqttfx.org
[14] https://github.com/sandro-k/MQTTLensChromeApp
[15] www.putty.org
[16] http://docs.openhab.org/configuration/persistence.html
[17] https://github.com/computourist/RFM69-MQTT-client
[18] https://github.com/computourist/ESP8266-MQTT-client
[19] http://homeautomation.proboards.com/board/2/openhab-rfm69-based-arduino

petit mais complet : Raspberry Pi Zero W

premiers pas avec la mini-framboise

La version mini du nano-ordinateur Raspberry Pi avec le suffixe Zero était disponible dès fin 2015. En février est apparue une variante Zero W (« Wireless «, sans fil) avec le Wi-Fi et le Bluetooth. Dans cet article, nous examinerons les procédures de démarrage de cette nouvelle carte, ses performances et les avantages que procure la technologie sans fil appliquée à ce minuscule ordinateur monocarte. À titre d'exemple pratique, nous connecterons un capteur de température dont nous enverrons les données sur le réseau sans fil avec le protocole MQTT.



Markus Ulsass (Allemagne)

Au prix de dix dollars hors taxes, le nouveau Raspberry Pi Zero W s'avère près de deux fois plus cher que son prédécesseur, mais il est maintenant doté de fonctions radio Wi-Fi et Bluetooth, indispensables pour beaucoup de projets. Cela permet de réaliser à peu de frais des projets de robots, de domotique intelligente et d'Internet des Objets qui tirent profit de la petite taille et de la consommation électrique réduite du Zero W. La puce monocœur SoC BCM2835 de Broadcom est cadencée à 1 GHz (512 Mo de RAM). Le Zero W n'est donc pas aussi rapide que son grand frère, le Raspberry Pi 3, mais cela reste bien suffisant pour la plupart des projets.
En France, kubii [1] est le distributeur officiel du Zero W. Malgré les frais de port, la boutique britannique Piromoni [2] offre une alternative guère plus chère et des livraisons raisonnablement rapides.
Le Zero W est équipé de ports mini-HDMI et micro-USB-On-The-Go. La carte est livrée dépourvue de connecteur à 40 broches, il faut donc l'ajouter au besoin à la commande ou choisir l'un des kits proposés par les boutiques.

Installation du système d'exploitation et premiers pas

En possession d'un Zero W et de tous les ingrédients nécessaires, y compris une alimentation (voir l'encadré « Liste du matériel «), on doit commencer par installer le système d'exploitation sur une carte micro-SD d'une taille d'au moins 8 Go.
Pour cela, on télécharge depuis [3] l'image courante du système « Raspbian Jessie with Pixel «. Pour copier cette image sur une carte SD, on se sert d'un utilitaire comme Etcher, qu'on peut télécharger depuis [4] (**fig. 1**). L'intérêt de ce programme est qu'il est disponible pour tous les environnements PC et qu'il ne nécessite pas une décompression préalable de l'image téléchargée : une fois qu'on l'a installé et démarré, on lui indique l'image appropriée qui est copiée sur la carte SD choisie.
Ensuite on enfiche la carte SD sur le Zero W, on connecte un écran sur la sortie HDMI (au besoin à travers un adaptateur), un clavier et une souris (par l'USB) et, en dernier, l'alimentation. Attention : des deux prises micro-USB, il faut utiliser pour l'alimentation celle de droite, l'ordinateur posé à plat devant soi, la carte SD se trouvant à gauche (**fig. 2**).
Le Raspberry Pi Zero W doit alors démarrer dans l'interface graphique Pixel du système d'exploitation. On en a alors terminé avec l'installation d'un système Linux opérationnel sur le micro-ordinateur.

Réglage fin

NdT : *les procédures qui suivent s'appliquent à la distribution de Raspbian du 21/06/2017. Elles peuvent changer dans les*

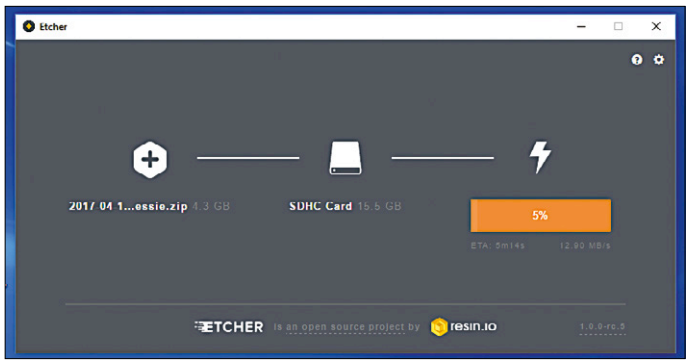


Figure 1. Etcher est un utilitaire de copie d'une image Raspbian sur une carte SD.

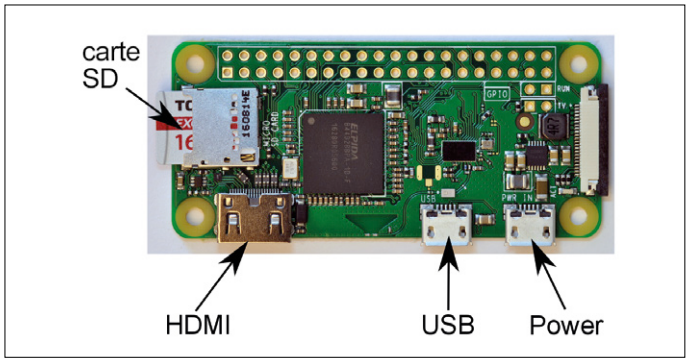


Figure 2. Connexions au Zero W.

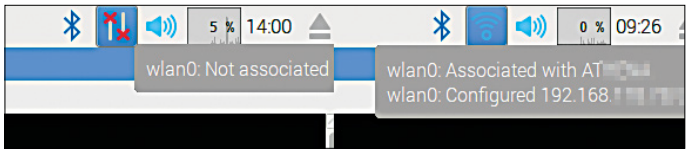


Figure 3. Avant (à gauche) / après (à droite). Après la configuration, le Zero W devrait établir une connexion Wi-Fi au réseau local. On obtient fréquemment des informations complémentaires en survolant les icônes avec la souris.

détails d'une distribution à l'autre.
Si l'on ouvre maintenant une fenêtre de terminal (touche *Window* gauche pour afficher le menu déroulant principal, puis *Accessories* > *Terminal,* ou bien par clic sur l'icône *Terminal* dans le bandeau de menu en haut de l'écran), on constate que l'environnement et le clavier initiaux sont en anglais étasunien. Pour modifier la langue, on saisit la commande *sudo raspi-config* (qu'on obtient en tapant *sudo rqspi) config* sur le clavier français). Le menu principal de l'outil *Raspberry Pi Software Configuration Tool* apparaît, dans lequel on choisit l'option *4 Localisation Options.* Dans le sous-menu, on sélectionne *I1 Change Locale* ; il apparaît une liste d'options qu'on fait défiler (touche flèche vers le bas) jusqu'à afficher fr-FR.UTF-8 UTF-8 qu'on sélectionne avec la touche espace, puis <*OK*> (atteint avec la touche de tabulation). Dans la fenêtre suivante, on valide FR.UTF-8 comme *Default locale for the system environment* et on confirme par <*OK*>. On redémarre par la commande *sudo reboot*.
La langue de l'environnement est maintenant le français, toutefois le clavier n'a pas changé. On retourne donc dans *raspi-config* > *4 Localisation Options* où l'on choisit *I3 Change Keyboard Layout.* Dans la fenêtre suivante, on choisit *PC générique 105 touches (intl)*, suivi de *Autre*, puis successivement de *Français, Français variante (sans touche morte), disposition par défaut pour le clavier, touche Alt de droite.* On finit par retourner au menu principal qu'on quitte par <*Finish*>. On peut alors constater que la touche A ne produit plus un q, mais bien un a.

Comme le Zero W utilise par défaut toujours le même mot de passe *raspberry* pour l'utilisateur *pi*, il faut absolument changer ce mot de passe. Pour ce faire, on utilise l'option *1 Change User Password* de *raspi-config.* On entre un nouveau mot de passe que l'on prend soin de noter.
Pour que le Zero W puisse être identifié sans ambiguïté sur le réseau, il faut également changer son nom d'ordinateur (par défaut : *raspberrypi*). Pour cela, on modifie avec l'éditeur *nano* une ligne dans deux fichiers texte. Avec la commande *sudo nano /etc/hostname*, on ouvre le premier fichier dans lequel on remplace *raspberrypi* par *zerow*. On enregistre avec *ctrl-O* puis *Entrée* et on quitte avec *ctrl-X*. Avec *sudo nano /etc/hosts*, on ouvre le second fichier où l'on localise la ligne *127.0.0.1 raspberrypi* dans laquelle on effectue la même substitution. On enregistre et on quitte par les mêmes commandes *ctrl-O, Entrée* et *ctrl-X*.
NdT : l'option 2 Hostname de raspi-config *effectue ces opérations. Il faut redémarrer pour que cette modification prenne effet.*

Communication avec l'extérieur

Pour que notre Zero W puisse se connecter au réseau local par Wi-Fi, il faut encore saisir les données du point d'accès (SSID, mot de passe). On clique sur le symbole des deux barres verticales et des deux croix rouges à droite dans le bandeau de menu en haut de l'écran (**fig. 3**). On choisit le

Listage 1. Le script Python pour faire clignoter une LED.

```
#!/usr/bin/python
#LED_Blink.py
import RPi.GPIO as GPIO #Inclure la bibliothèque GPIO
import time #Bibliothèque nécessaire au Sleep

LED = 14
GPIO.setmode(GPIO.BCM) #Utiliser l'identification de broche GPIO
GPIO.setwarnings(False) #Ignorer les avertissements
GPIO.setup(LED, GPIO.OUT) #Configurer en sortie la broche de la LED
PAUSEON = 1.0 #Temps d'allumage
PAUSEOFF = 1.0 #Temps d'extinction

while True:
    GPIO.output(LED, GPIO.HIGH) #Allumer la LED
    time.sleep(PAUSEON) #Temps d'allumage
    GPIO.output(LED, GPIO.LOW) #Eteindre la LED
    time.sleep(PAUSEOFF) #Temps d'extinction
```

LISTE DU MATÉRIEL

Raspberry Pi Zero W
Carte micro-SD (min. 8 Go)
Adaptateur HDMI->mini-HDMI
Adaptateur micro-USB-OTG (par ex. Delock 65296)
Bloc d'alimentation micro-USB (min. 1 A)
Barrette à 2×20 broches (au pas de 2,54 mm)
Platine d'essai à trous
Fils de connexion
LED (rouge)
Résistance de 470 Ω
Résistance de 4,7 kΩ
Capteur de température DS18B20

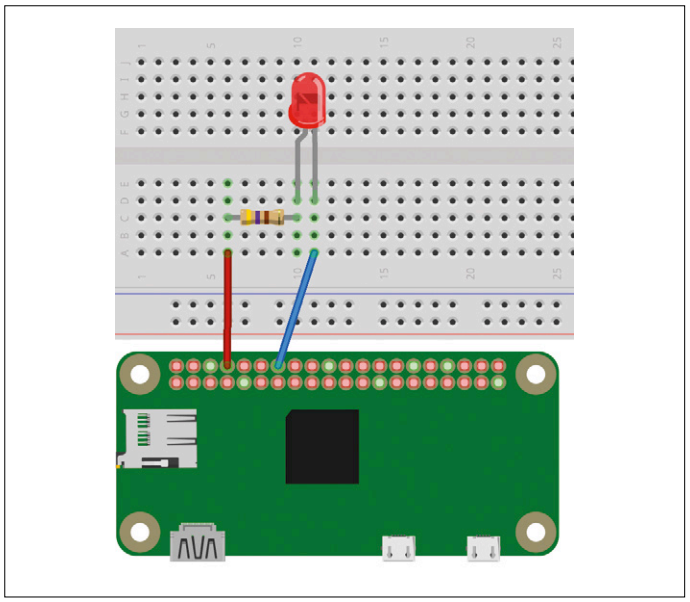


Figure 4. L'image Fritzing de la connexion de la LED au Zero W.

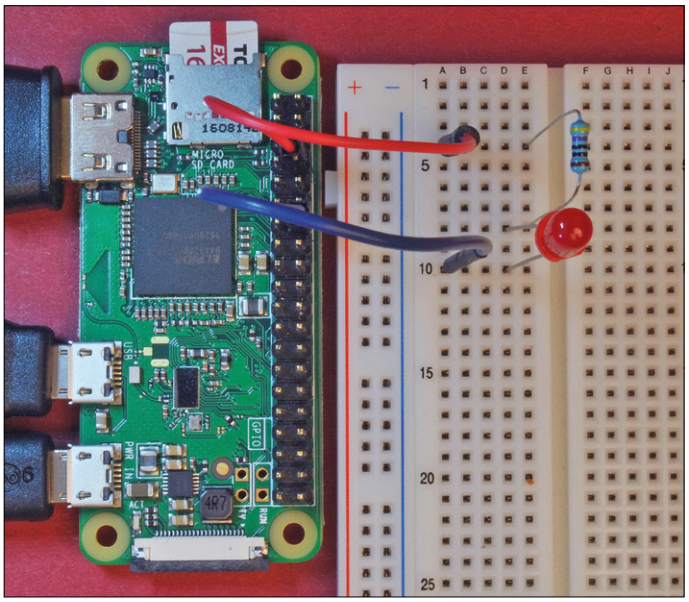


Figure 5. La connexion du Zero W avec la LED et la résistance sur la platine d'essai.

point d'accès désiré, on entre le mot de passe et le Zero W devrait se connecter au réseau.
Avec une nouvelle image du système, il est bon de mettre à jour la liste des paquets et le système. Dans un terminal, la commande *sudo apt update && sudo apt upgrade* effectue cette opération. On confirme par *Y* et on attend quelques minutes que tous les téléchargements se terminent.

« Hello world « par LED

Après en avoir terminé avec les préparatifs de base, occupons-nous de connecter un premier circuit simple. Pour cela, il faut tout d'abord souder un connecteur à 40 broches sur le Zero W pour pouvoir lui raccorder par fils une LED sur platine d'essai. Avant de connecter la résistance et la LED, on commence par arrêter le Zero W et le mettre hors tension, afin d'éviter les courts-circuits ou tout autre effet négatif lors de la connexion directe de matériel aux broches GPIO. Pour cela on saisit la commande *sudo shutdown –h now* (ou *sudo halt*) dans la fenêtre de terminal ou on clique sur *Shutdown* dans le menu déroulant, puis sur l'option *Shutdown.*

Pour notre « Hello world « par LED, nous connectons une LED rouge en série avec une résistance de 470 Ω à la broche GPIO14. Le brochage du Zero W est disponible sous [5]. On raccorde la broche GPIO14 à la résistance, celle-ci à l'anode de la LED, dont la cathode est raccordée à l'une des broches 0 V du Zero W. La **figure 4** montre une image Fritzing ; la **figure 5**, une vue réelle du circuit.

C'est un petit script Python (*LED_Blink.py*) qui va nous servir à faire clignoter la LED. Il est présenté sur le **listage 1**, mais peut aussi être téléchargé sous [6] depuis le site web d'Elektor. Nous pouvons créer le code avec l'éditeur *nano* ou bien avec *geany*, éditeur un peu plus confortable (*Menu -> Programmation -> Geany*), présent par défaut dans le système Raspbian. Remarquons qu'il faut éviter d'utiliser les lettres spéciales (avec accent, tréma, cédille...), en particulier dans les commentaires et que les indentations, éléments de la syntaxe de Python, doivent être faites avec des espaces (et non avec des tabulations), sous peine de provoquer des erreurs (*Indentation Error*). Une fois le programme enregistré, il est démarré avec la commande *python LED_Blink.py* saisie dans la fenêtre de terminal. La LED devrait alors clignoter au rythme d'une fois par seconde.
On arrête le programme par *ctrl-C*.

Mesure de température avec le DS18B20

Le DS18B20 de Maxim Integrated est un capteur de température répandu et facile à connecter. Pour commencer, on éteint le système pour réaliser le circuit sur la platine d'essai et le connecter au Zero W en toute sécurité.
La broche de gauche du DS18B20 (vu avec son immatriculation de face) est connectée au 3,3 V du Zero W, celle de droite à une broche 0 V. Celle du milieu est connectée à la broche GPIO4. Finalement, il faut encore installer une résistance de

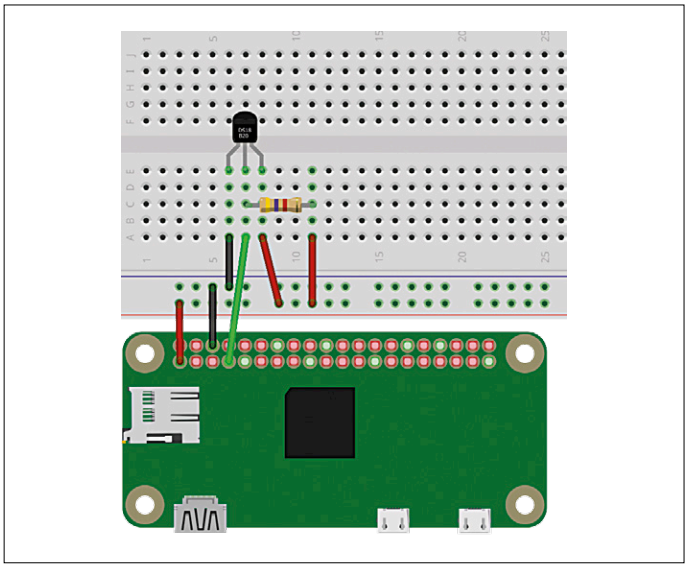


Figure 6. Le circuit avec le capteur de température DS18B20.

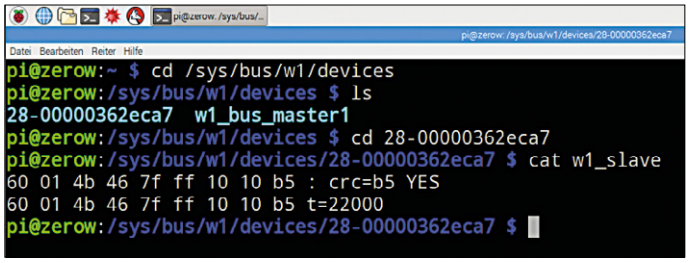


Figure 7. Le test du capteur connecté.

rappel de 4,7 kΩ entre la broche de données et la tension d'alimentation (voir l'image Fritzing, **figure 6**).

Après le raccordement et un contrôle du câblage, si tout est correct, on redémarre le Zero W. Mais avant de pouvoir utiliser le capteur de température One-Wire, le système doit le reconnaître. Pour cela, à l'option *P7 1-wire* de *sudo raspi-config*, il faut avoir répondu *Y* à la question *Would you like the one-wire interface to be enabled?* puis avoir redémarré le système. Maintenant il faut vérifier que le capteur est correctement reconnu par le système. Pour cela, dans une fenêtre de terminal, on se positionne dans le répertoire *cd /sys/bus/w1/devices* dont on liste le contenu avec la commande *ls*. Il doit alors s'afficher une identification alphanumérique du capteur.
Avec la commande *cd <Sensor-Id>*, soit, dans notre cas, *cd 28-00000362eca7*, on se positionne dans le sous-répertoire correspondant où, avec la commande *cat w1_slave*, on liste les données courantes du capteur, qui sont affichées sur deux lignes (**fig. 7**). t=22000 représente dans ce cas une température de 22°C.
Le capteur est alors configuré correctement et prêt à l'emploi.

Connexion du capteur de température au reste du monde par MQTT

On peut vouloir communiquer les valeurs de température lues par notre Zero W à d'autres utilisateurs. Pour cela, nous allons utiliser MQTT, un protocole léger d'échange d'informations entre machines. MQTT utilise un mécanisme appelé *Publish-Subscribe* où des capteurs peuvent par ex. publier des données sous des sujets particuliers (*Publish*), auxquels des clients peuvent s'abonner (*Subscribe*) pour recevoir ces données. Les noms des sujets rappellent, par la présence de barres de séparation obliques, la syntaxe des arborescences de répertoires (on aurait par ex. *appartement/cuisine/température*) ; on peut même avoir des caractères génériques. Au lieu de fichiers, on identifie ainsi des capteurs, ou même des actionneurs.
Notre capteur de température pourrait publier ses valeurs de mesure sous le sujet « /maison/extérieur/température/capteur1 », auquel un client (par ex. un système domotique comme openHAB, fhem ou Node-Red) se serait abonné.
MQTT nécessite l'intervention d'un intermédiaire (appelé *broker* dans le jargon MQTT) qui gère la circulation des données entre l'émetteur et le récepteur. On choisit un courtier (*broker*) populaire dénommé Mosquitto, que l'on installe sur le Zero W avec la commande *sudo apt install mosquitto mosquitto-clients*.
Pour vérifier que tout est correctement installé, on peut tester l'envoi ainsi que la réception des données sur le même système. Pour cela, il faut commencer par démarrer le courtier avec *sudo systemctl start mosquitto*. Pour qu'il soit automatiquement réactivé à chaque redémarrage du système, on saisit aussi la commande *sudo systemctl enable mosquitto*.
On simule dans une fenêtre de terminal

Listage 2. Script Python pour l'envoi de la température par MQTT.

```
#!/usr/bin/python
#DS18B20_MQTT.py
#Importation des modules requis
import time, sys, os
import paho.mqtt.client as mqtt

mqtt_host = "localhost"

#Lecture de la température courante
def currentTemperature():

        #Substituer ici l'identification réelle du capteur de
température
        file = open('/sys/bus/w1/devices/28-00000362eca7/w1_slave')
        filecontent = file.read()
        file.close()

        #Mise de la température au bon format
        temperaturestring = filecontent.split("\n")[1].split(" ")[9]
        temperaturevalue = float(temperaturestring[2:]) / 1000
        return(temperaturevalue)

def on_connect(client, userdata, flags, rc):

    print("Connected with MQTT-Broker (IP): " + mqtt_host)

client = mqtt.Client()
client.on_connect = on_connect

client.connect(mqtt_host, 1883, 60)

client.loop_start()

while True:
    time.sleep(2)

    temperature = currentTemperature()
    print("Current temperature: " + str(temperature) + " degrees
Celsius")
    client.publish("/sensor1", temperature)
```

avec *mosquitto_sub -h localhost -t /sensor1* un abonné qui doit ultérieurement recevoir des données – nous simplifions ici le libellé du sujet pour un gain de temps et une meilleure lisibilité. Pour l'instant, il ne se passe rien, le système est en attente de données. Ouvrons maintenant une autre fenêtre de terminal et saisissons *mosquitto_pub -h localhost -t /sensor1 -m «22»* (**fig. 8**). Nous simulons ainsi l'envoi (*mosquitto_pub*) d'une donnée de température, ou (pour rester dans le langage de MQTT) sa publication (*publish*). On devrait donc voir s'afficher 22 dans la première fenêtre.
Notre système de publication des valeurs de température fournies par le capteur est maintenant complet. Nous allons l'automatiser dans une dernière phase.

Envoi de la température par MQTT

Pour l'envoi automatique de données par MQTT, nous utilisons un script Python. Pour cela, il nous faut une bibliothèque installée avec la commande *sudo pip install paho-mqtt*.

Notre script (**listage 2**) commence par importer les modules nécessaires et définit l'adresse IP de l'ordinateur utilisée par Mosquitto (« localhost « pour le courtier local, l'adresse IP pour un courtier distant).
Ensuite, la température courante est lue par l'appel de la fonction *currentTemperature()* à laquelle on doit bien entendu passer, au lieu de celle du code exemple, l'identité réelle du capteur utilisé. Les données lues sont converties dans un format approprié et retournées par la fonction.
Une connexion avec le courtier MQTT local ou distant (*mqtt_host*) est alors créée et, dans une boucle while, la valeur de la température est lue toutes les deux secondes, affichée sur la ligne de commande et envoyée au courtier MQTT sous le sujet « /sensor1 «.
Après avoir saisi le programme dans un éditeur ou l'avoir téléchargé, on ouvre une fenêtre de terminal dans laquelle on le démarre avec *python DS18B20_MQTT.py*. Sur la ligne de commande devrait apparaître la notification de la connexion au courtier MQTT local du Zero W, suivie des valeurs de température courantes (**fig. 9**).
Maintenant, nous ouvrons une autre fenêtre de terminal et nous nous abonnons en tant que client MQTT au sujet « /sensor1 « sur l'ordinateur local avec la commande *mosquitto_sub -h localhost -t /sensor1*.

Tout cela et bien plus encore

Avec le Zero W, la fondation Raspberry Pi a réussi, juste à temps pour le cinquième anniversaire du premier Raspberry Pi, à produire une carte fantastique. Qui aurait pu imaginer, il y a cinq ans, qu'un micro-ordinateur Linux d'une taille aussi minuscule puisse voir le jour ?
Le port USB qui, chez le prédécesseur, était encore occupé par un clavier ou un module Wi-Fi, est devenu disponible pour d'autres fonctions. Même la consommation électrique est devenue très faible, presque du niveau des objets connectés autoalimentés. On trouvera une comparaison des consommations des différents modèles de RPi sous [7].
Avec toutes ces caractéristiques exceptionnelles, il n'est pas étonnant que cette carte ne soit disponible qu'en petites quantités. C'était déjà le cas du premier RPi, et on ne peut qu'espérer que la disponibilité du Zero W s'améliore rapidement. ⏮

(160451 – version française : Helmut Müller)

Liens

[1] www.kubii.fr/
[2] https://shop.pimoroni.com/
[3] www.raspberrypi.org/downloads/raspbian/
[4] https://etcher.io/
[5] https://pinout.xyz/
[6] www.elektormagazine.fr/160451
[7] https://blog.adafruit.com/2017/03/03/how-much-power-does-pi-zero-w-use-piday-raspberrypi-raspberry_pi/

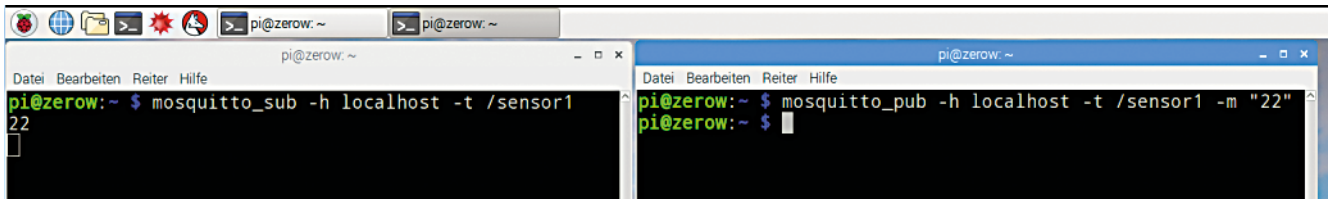


Figure 8. Le test de la fonction MQTT.

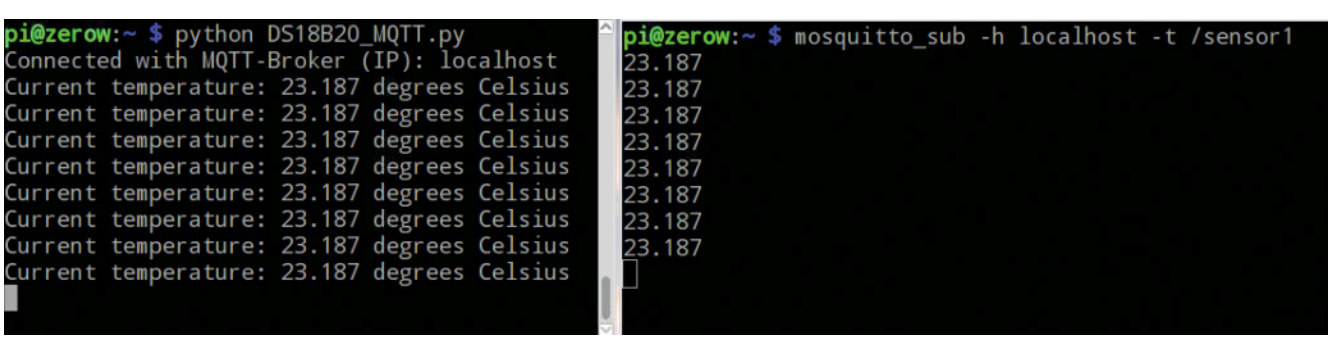


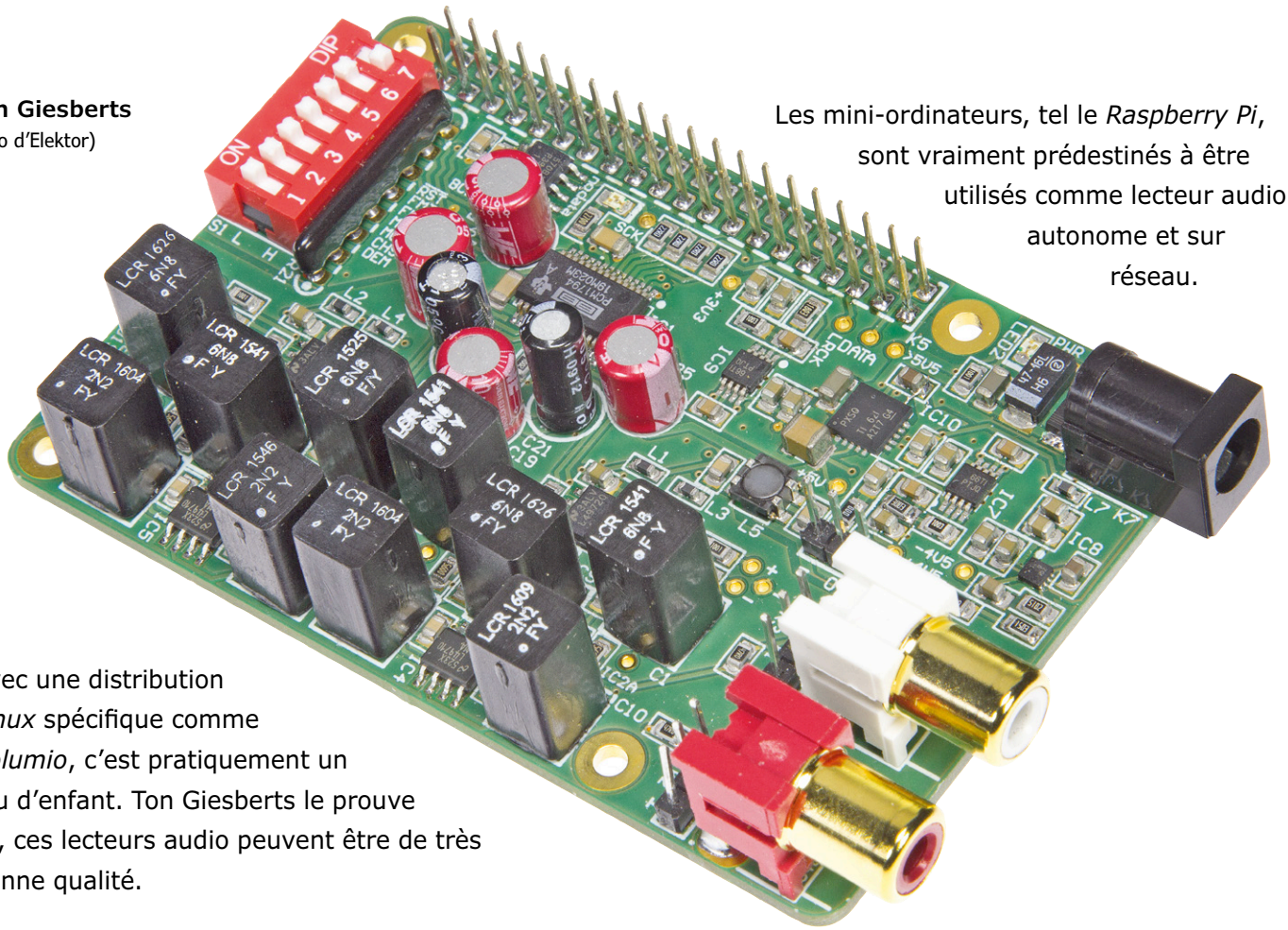
Figure 9. Test de l'envoi des données du capteur de température : à gauche l'affichage local, à droite la réception des données transmises par MQTT.

CN/A audio pour RPi

lecteur audio sur réseau avec *Volumio*

Ton Giesberts
(labo d'Elektor)

Les mini-ordinateurs, tel le *Raspberry Pi*, sont vraiment prédestinés à être utilisés comme lecteur audio autonome et sur réseau.



Avec une distribution *Linux* spécifique comme *Volumio*, c'est pratiquement un jeu d'enfant. Ton Giesberts le prouve ici, ces lecteurs audio peuvent être de très bonne qualité.

L'offre en lecteurs audio autonomes qui peuvent être mis sur un réseau est limitée, qui plus est si on souhaite aussi un écran tactile. Il y a bien quelques amplificateurs de marque, volumineux et au prix à l'avenant ; en général, ils n'ont pas d'écran tactile pour la commande non plus. En tout cas, si l'on veut un appareil compact et portable, ce n'est pas dans cette voie qu'il faut chercher.

On en est vite réduit à se tourner vers une solution faite maison, et une application sur la base du *Raspberry Pi* constitue un bon point de départ. On trouve sur le marché des convertisseurs CN/A de bonne qualité pour ce mini-ordinateur, mais pour Elektor nous voulions encore mieux. Nous avons donc décidé de concevoir un convertisseur pour le *RPi*, de la meilleure qualité possible, et avec des composants irréprochables.

Vue d'ensemble

Les circuits intégrés CN/A de *Burr-Brown* (désormais *Texas Instruments*) restent sans doute ce qui se fait de mieux, et nous les avons déjà utilisés par le passé. Nous avons choisi pour ce projet le PCM1794A, un des meilleurs du fabricant ; il s'agit d'un CN/A à 24 bits, qui accepte des taux d'échantillonnage de 10 à 200 kHz, et qui comprend un filtre à suréchantillonnage (×8). Les caractéristiques de ce circuit intégré, fourni en boîtier *SSOP* (*Shrink Small Out-Line package*) à 28 broches, sont excellentes à tous points de vue : plage dynamique de 132 dB (9 V_{eff}, mono) et 127 dB (2 V_{eff}, stéréo), distorsion de 0,0004%, et sorties en courant symétriques (7,8 mA crête à crête). Il accepte en outre tous les formats de données connus : standard, justifié à gauche, et – important pour notre projet – I²S. L'alimentation de la partie numérique du circuit intégré est de 3,3 V, mais les entrées numériques acceptent les signaux 5 V. La fiche de caractéristiques est disponible sur le site de *TI* [1].

Les sorties en courant symétriques devront bien entendu être transformées en sorties en tension asymétriques à l'aide d'un convertisseur I/V, et filtrées. Forts de ces connaissances, nous avons dessiné le schéma de principe (**figure 1**) et établi notre « liste de courses » : un *Raspberry Pi 3* (la version 2 convient aussi), un CN/A tel que décrit ci-dessus, et un écran tactile compatible avec le *RPi* (nous avons choisi un modèle *Waveshare* de 3,5 pouces). La partie logicielle inclut *Raspbian* pour le *RPi*, *Volumio* (ou un programme équivalent, p. ex. *Mood Player*), et un pilote pour l'écran tactile.

Considérations sur l'alimentation principale

Le circuit intégré CN/A IC1 (voir le schéma en **figure 2**) a besoin de deux alimentations : 3,3 V pour la partie numérique et 5 V pour la partie analogique. Nous pourrions utiliser les tensions d'alimentation présentes sur le connecteur GPIO du *RPi*, mais elles sont tellement « polluées » qu'il ne resterait rien de notre quête du haut de gamme. Nous avons donc besoin de sources indépendantes pour les tensions de 3,3 V et 5 V ; il est à noter qu'on pourrait alors alimenter le *RPi* à partir de la platine CN/A, mais cela aurait aussi des effets néfastes sur la qualité finale. L'inconvénient est qu'il faudrait deux connecteurs pour l'alimentation positive, et un troisième serait nécessaire pour la tension négative des amplis op utilisés pour les convertisseurs I/V et les filtres de sortie ; pour éviter cela, nous utilisons des régulateurs, et un inverseur pour obtenir la tension négative à partir du +5 V. En y regardant de plus près, on constate que l'alimentation de la partie analogique est en fait de +5,2 V. Cette valeur quelque peu supérieure – sans danger pour le PCM1794A – est voulue, pour que la valeur maximale de la tension de sortie sans déformation du signal soit de 1 V. On rencontre souvent des spécifications de 2 V en sortie, mais il nous faudrait soit des tensions d'alimentation plus élevées, soit des amplis op avec une capacité de sortie rail à rail. La plupart des amplis op ayant cette capacité n'ont cependant pas les qualités des amplis op conçus spécifiquement pour l'audio haut de gamme.

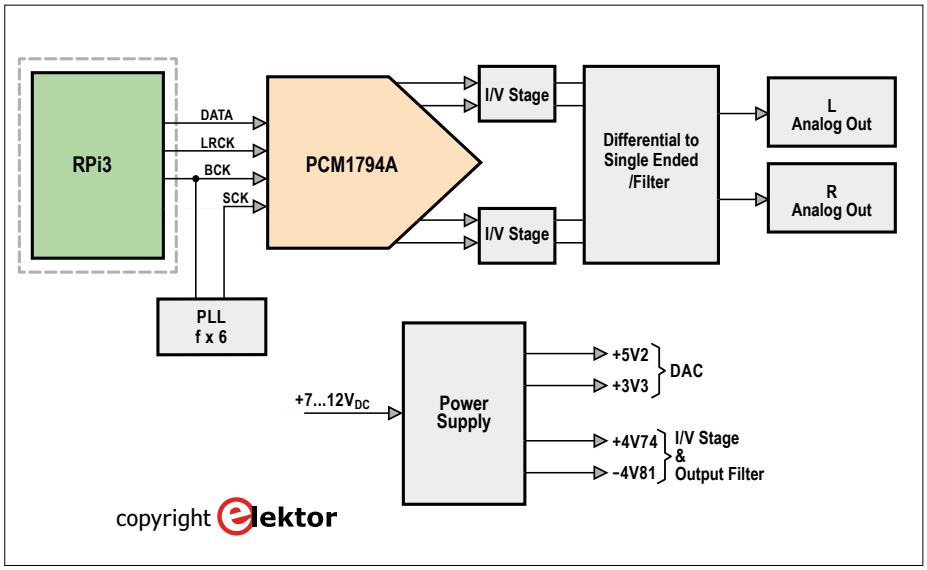
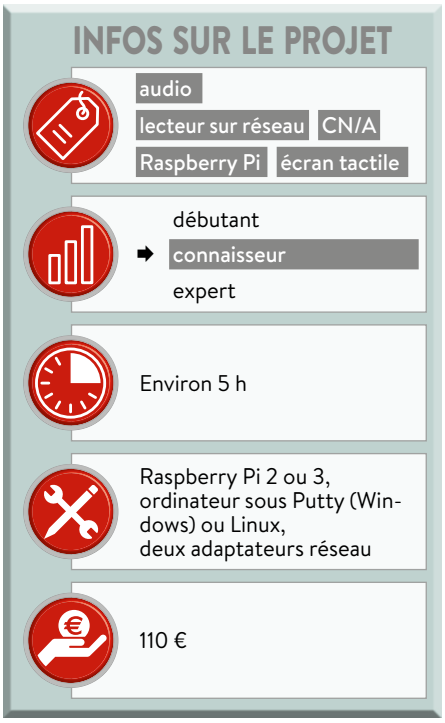


Figure 1. Schéma de principe du CN/A audio.

Pour l'obtention du 5,2 V, nous avons choisi le TPS7A4700, un régulateur de tension à faible chute : 307 mV pour un courant maximal de 1 A. La valeur de la tension de sortie est fixée en reliant certaines broches à la masse, connectées en interne à des résistances ; la tension régulée minimale est de 1,4 V, égale à la valeur de la référence de tension du circuit. Pour obtenir 5,2 V en sortie, ce sont les broches 6, 10 et 11 qui sont reliées à la masse :

$$V_{sortie} = 1,4V + 3,2V + 0,4V + 0,2V = 5,2V$$



Autres tensions

La tension d'alimentation positive des amplis op est fixée à 4,74 V, obtenue à partir du 5,2 V via le régulateur linéaire à très faible bruit IC7, un TPS7A4901. La valeur de la tension de sortie est fixée par le diviseur de tension R28/R29 :

$$V_{sortie} = \left(\frac{R28}{R29} + 1\right) \times 1,185V$$

La tension de 3,3 V est obtenue via un régulateur identique, IC9, et le diviseur de tension R32/R33.

Pour la tension d'alimentation négative des amplis op, nous avons choisi un LM27761, un régulateur à découpage inverseur à faible bruit. L'inversion de tension s'effectue par commutation de condensateurs ; la tension obtenue est ensuite stabilisée par un régulateur linéaire à faible bruit. La valeur de la tension est fixée à −4,81 V par le diviseur de tension R30/R31 :

$$V_{sortie} = -\left(\frac{R30}{R31} + 1\right) \times 1,22V$$

Rappelons que ces valeurs inhabituelles (±4,5 V est plus courant) ont été calculées pour garantir une tension maximale du signal en sortie, sans déformation, de 1 V (0 dB).

L'entrée d'alimentation (via K7) est protégée contre une inversion de polarité par

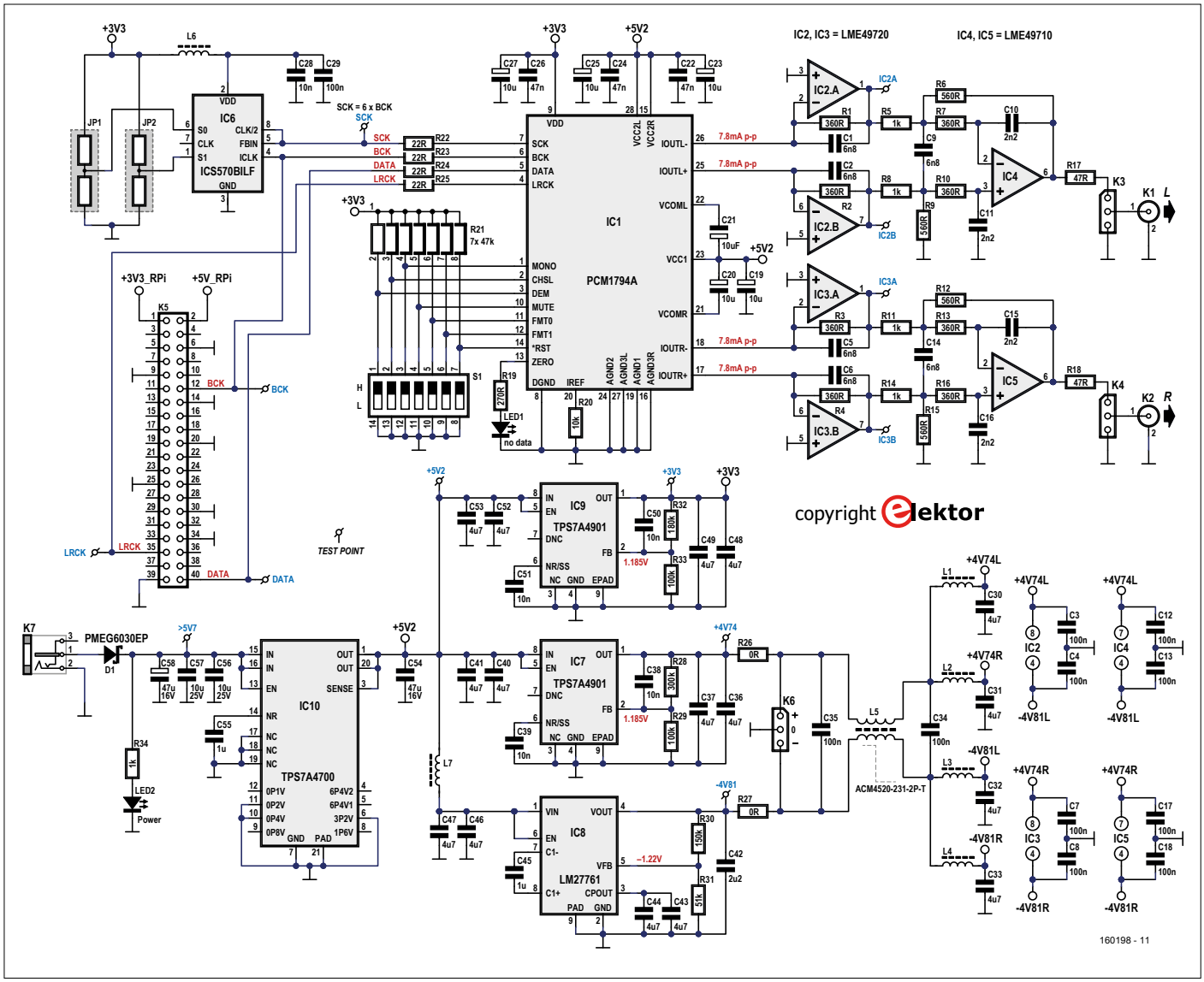


Figure 2. Le schéma détaillé du convertisseur CN/A pour le RPi.

la diode Schottky D1, qui n'occasionne que 0,3 V de chute de tension. Le courant nécessaire dépend de la fréquence d'échantillonnage, sa valeur est reprise dans le **tableau 2**.

Horloge principale

Pour que la carte CN/A audio fonctionne correctement, *Raspbian* (le système d'exploitation du *RPi*) doit être configuré pour un CN/A *HiFiBerry*. Les signaux I²S (*Inter-IC Sound*) seront alors disponibles sur le connecteur d'extension GPIO. Ces signaux comprennent un signal d'horloge continu (*SCK*) ou horloge bit (*BCLK/BCK*), un signal de sélection de mot (*WS*) ou horloge gauche-droite (*LRCLK/LRCK*), et un signal de données série (*SD/SDATA*). Pour la synchronisation, il faut aussi un signal d'horloge principal (*master clock*), mais cela n'est pas inclus dans les spécifications I²S originales. Ce signal n'est donc pas présent sur le connecteur GPIO, et le CN/A doit le produire lui-même. Nous avons pour ce faire ajouté un multiplicateur à boucle à verrouillage de phase (*PLL*), un ICS570BILF (IC6). Ce circuit intégré dispose d'un tampon sans délai de propagation, les flancs montants des signaux d'entrée et de sortie coïncident donc parfaitement.

Le facteur de multiplication peut être sélectionné entre 0,5 et 32 à l'aide des entrées à trois états S0 et S1. La plage du signal de sortie va de 10 à 170 MHz. Nous avons choisi un facteur de multiplication de 6 (S0 et S1 ouvertes), et nous utilisons la sortie *CLK 2*, qui est aussi reliée à l'entrée de contre-réaction. Cette configuration offre la plage d'entrée la plus étendue (2,5 à 12,5 MHz) pour une plage de sortie de 15 à 75 MHz. On pourrait croire qu'un signal dont la fréquence d'échantillonnage est de 32 kHz ne peut être reproduit, mais il n'en est rien : ça marche très bien.

Un petit souci

Lors du test de notre CN/A avec des signaux de 32 ou 24 bits et une fréquence d'échantillonnage entre 32 et 192 kHz, tout fonctionnait parfaitement. Nous avons cependant rencontré un problème avec des signaux de 16 bits : pas de signal de sortie ! Pour résumer notre longue investigation : le PCM1794A supporte bien les signaux audio de 16 et 24 bits, mais apparemment pas en mode I²S ; la fiche de caractéristiques n'est pas très claire à ce sujet...

Est-ce que cela signifie que notre lecteur n'accepte pas les signaux de 16 bits ? Oui et non... *Volumio*, du moins la ver-

sion 1.55 que nous utilisons, permet la conversion de la fréquence d'échantillonnage avec trois niveaux de qualité. Cette possibilité sera sans doute aussi offerte par la version 2 du programme.

Le convertisseur N/A

Le PCM1794A intègre un filtre à suréchantillonnage (×8) et accepte les fréquences d'échantillonnage de 10 à 200 kHz. Les signaux I²S parviennent au CN/A via des résistances de 22 Ω (R22 à R25), pour éviter les parasites RF : pour une fréquence d'échantillonnage de 192 kHz, la fréquence d'horloge principale est de près de 74 MHz ! Les lignes de sélection matérielles (*MONO, CHSL, DEM, MUTE, FMT0, FMT1, RESET*) passent par un interrupteur DIP à 7 pôles, et un réseau de 7 résistances de rappel de 47 kΩ ; le changement de configuration en est grandement facilité. La sortie *ZERO* (broche 13) commande LED 1, qui indique l'absence de données audio.

Pour le découplage des alimentations, nous utilisons des condensateurs à électrolyte solide (C19, 23, 25 et 27) ; leur résistance série équivalente (*ESR*) n'est que de 40 mΩ à 100 kHz, mais leur cou-

LISTE DES COMPOSANTS

Résistances (1%, 0,125 W, CMS 0805, sauf indication contraire)

R1, R2, R3, R4, R7, R10, R13, R16 = 360 Ω
R5, R8, R11, R14, R34 = 1 kΩ
R6, R9, R12, R15 = 560 Ω
R17, R18 = 47 Ω
R19 = 270 Ω
R20 = 10 kΩ
R21 = 47 kΩ, réseau de 7 résistances en boîtier SIP, 125 mW, 2%
R22, R23, R24, R25 = 22 Ω
R26, R27 = 0 Ω
R28 = 300 kΩ
R29, R33 = 100 kΩ
R30 = 150 kΩ
R31 = 51 kΩ
R32 = 180 kΩ

Condensateurs

C1, C2, C5, C6, C9, C14 = 6,8 nF, 63 V, 1%, polystyrène, EXFS/HR 6 800 pF +/- 1%, LCR Components (pas de 5 mm ou CMS 0805)
C3, C4, C7, C8, C12, C13, C17, C18, C29, C34, C35 = 100 nF, 50 V, 10%, X7R, CMS 0805
C10, C11, C15, C16 = 2,2 nF, 63 V, 1%, polystyrène, EXFS/HR 2 200 pF +/- 1%, LCR Components (pas de 5 mm ou CMS 0805)
C19, C23, C25, C27 = 10 µF, 35 V, 0,04 Ω, diamètre max. de 6,3 mm, pas de 2 ou 2,5 mm, 870055673001 (WCAP-PTHR Series), Würth Elektronik
C20, C21 = 10 µF, 63 V, 1,06 Ω, diamètre max. de 6,3 mm, pas de 2 ou 2,5 mm, UPM1J100MDD, Nichicon
C22, C24, C26 = 47 nF, 50 V, 10%, X7R, CMS 1206
C28 = 10 nF, 50 V, 10%, X7R, CMS 0603
C30, C31, C32, C33, C36, C37, C40, C41, C43, C44, C46, C47, C48, C49, C52, C53 = 4,7 µF, 25 V, 10%, X5R, CMS 0805
C38, C39, C50, C51 = 10 nF, 50 V, 10%, X7R, CMS 0805
C42 = 2,2 µF, 10 V, 10%, X7R, CMS 0805
C45, C55 = 1 µF, 16 V, 10%, X7R, CMS 0805
C54 = 47 µF, 16 V, 20%, X5R, CMS 1210
C56, C57 = 10 µF, 25 V, 10%, X5R, CMS 1206

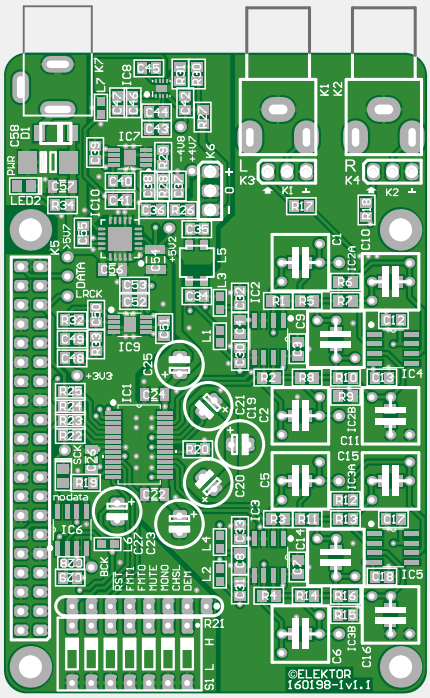


Figure 3. Le circuit imprimé, côté composants.

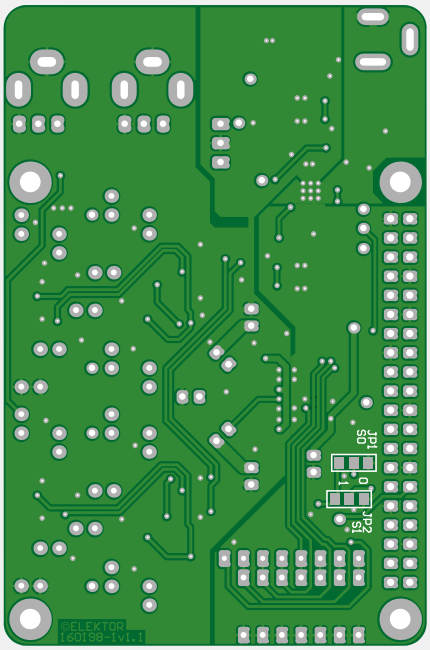


Figure 4. Le circuit imprimé, côté soudure. En principe, il ne faut pas monter JP1 et JP2.

C58 = 47 µF, 16 V, 10%, tantale, CMS-C (2312), TR3C476K016C0350, Vishay

Bobines

L1, L2, L3, L4, L6, L7 = 600 Ω @ 100 MHz, 0,15 Ω CC, 1,3 A, CMS 0603, BLM18KG601SN1D, Murata
L5 = 230 Ω @ 100 MHz, 2 x 0,05 Ω CC, 2,6 A, CMS, ACM4520-231-2P-T, TDK

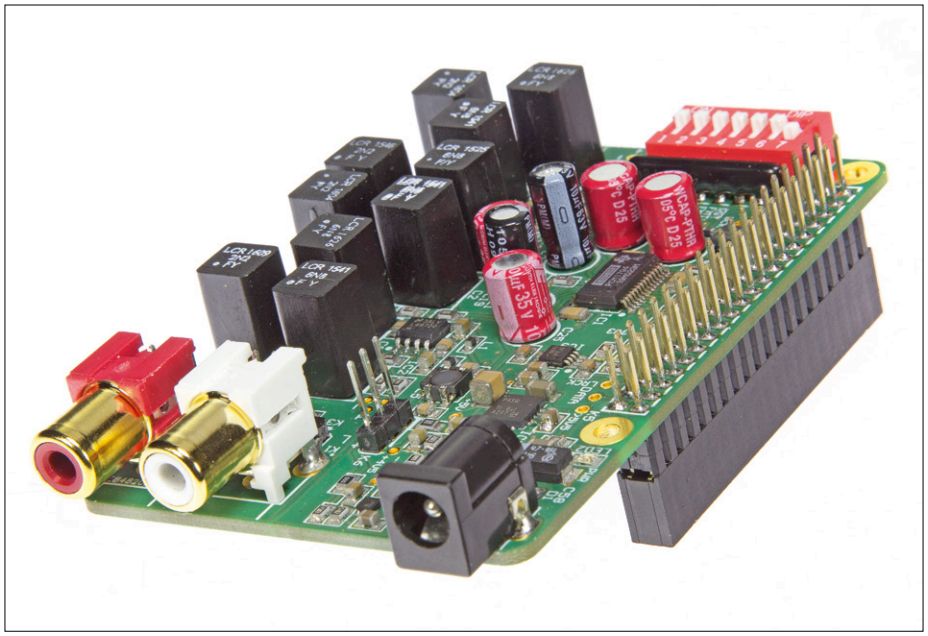
Semi-conducteurs

D1 = PMEG6030EP, 60 V, 3 A, CMS SOD-128
LED1, LED2 = LED verte, faible consommation, CMS 0805
IC1 = PCM1794ADB, CMS SSOP-28
IC2, IC3 = LME49720MA/NOPB, CMS SOIC-8
IC4, IC5 = LME49710MA, CMS SOIC-8
IC6 = ICS570BILF, CMS SOIC-8
IC7, IC9 = TPS7A4901DGNT, CMS MSOP 8
IC8 = LM27761DSGT, CMS WSON-8
IC10 = TPS7A4700RGWT, CMS VQFN-20

Divers

K1 = connecteur audio RCA blanc pour circuit imprimé, doré, PJRAN1X1U02AUX, Switchcraft
K2 = connecteur audio RCA rouge pour circuit imprimé, doré, PJRAN1X1U03AUX, Switchcraft
K3, K4, K6 = 3 picots à souder, pas de 2,54 mm
K3, K4 = cavalier, pas de 2,54 mm
K5 = connecteur GPIO à 40 broches, 2×20, femelle, extra haut
K7 = connecteur alimentation CC, 3 A, 1,95 mm, Lumberg NEB 21 R
S1 = interrupteur DIP à 7 pôles
4 entretoises M2,5 de 17 mm avec bout fileté
4 entretoises M2,5 de 14 mm avec bout fileté
8 écrous et rondelles M2,5

Circuit imprimé 160198-1



Figuur 5. Le connecteur GPIO est inséré côté soudure.

Tableau 1 : positions de S1		
S1-1	L	Désaccentuation 44,1 kHz désactivée
S1-2	L	Filtre numérique à flanc raide
S1-3	L	Sortie mono
S1-4	L	Sortie coupée (*mute*)
S1-5, S1-6	L	Mode I^2S
S1-7	H	Remise à zéro (*reset*) désactivée

Tableau 2 : courant en fonction de la fréquence d'échantillonnage	
Fréquence d'échantillonnage	**Courant (K7)**
Pas de données	120 mA
32 kHz	127,5 mA
44,1 kHz	131,7 mA
48 kHz	133 mA
96 kHz	149,5 mA
192 kHz	182,1 mA
(mesuré avec une alimentation de 8 V, les deux sorties terminées sur 10 kΩ)	

Tableau 3 : distorsion harmonique				
DHT + bruit				
Fréquence d'échantillonnage		48 kHz	96 kHz	192 kHz
1 kHz	B = 22 kHz	0,0008 %	0,0009 %	0,0013 %
	B = 80 kHz	0,0028 %	0,0012 %	0,0014 %
7 kHz	B = 22 kHz	0,00095 %	0,0011 %	0,0013 %
	B = 80 kHz	0,003 %	0,0014 %	0,0016 %
DIM				
50 Hz : 7 kHz = 4:1		0,0014 %	0,002 %	0,0036 %

rant de fuite maximal est relativement élevé, 100 µA. Nous ignorons si ce courant de fuite peut influencer les réglages de courant internes du circuit intégré, nous utilisons donc aussi des condensateurs électrolytiques « ordinaires » : *ESR* de 1,06 Ω à 100 kHz, courant de fuite de 4 µA. On pourra expérimenter avec divers types de condensateurs, il y a suffisamment de place sur le circuit intégré pour cela.

Le filtre de sortie

Les produits de la fréquence d'échantillonnage sont limités par le filtre à suréchantillonnage, mais il y a tout de même des composantes HF présentes en sortie du CN/A. Pour atténuer ces fréquences, nous utilisons un filtre de Butterworth passe-bas du 3e ordre. Les filtres et les convertisseurs I/V sont imbriqués et bâtis autour des amplis op IC2 et IC3. La fréquence de coupure est de 64,5 kHz. Une fréquence de 352,8 kHz, correspondant à 8 fois une fréquence d'échantillonnage de 44,1 kHz, est atténuée de plus de 40 dB ; pour une fréquence d'échantillonnage de 192 kHz, l'atténuation de l'harmonique 8 est supérieure à 80 dB. Le choix de la fréquence de coupure du filtre est toujours un compromis entre une atténuation suffisante avec des fréquences d'échantillonnage basses, et la bande passante du signal audio ; ici, cette bande passante est tout de même plus de trois fois supérieure au spectre audible.

Les sorties du CN/A sont en courant et symétriques, ce qui permettrait l'utilisation d'un filtre externe. Pour notre carte audio, les sorties doivent être converties en tension, et désymétrisées. Les convertisseurs I/V sont bâtis autour d'IC2 et IC3 ; les sorties symétriques d'IC2A/IC2B et IC3A/IC3B sont ensuite désymétrisées par IC4 et IC5, qui sont aussi configurés en filtres passe-bas du 2e ordre. Grâce à la combinaison avec les convertisseurs I/V, nous avons bien finalement des filtres du 3e ordre (avec une fonction de transfert de type Butterworth).

Les signaux de sortie sont disponibles sur K1 et K2, via K3 et K4. Les signaux peuvent transiter directement de K3/K4 vers K1/K2, ou on peut insérer un potentiomètre double comme réglage de volume entre les connecteurs.

Le circuit intégré CN/A ne permet pas le réglage du volume. On pourrait utiliser *Volumio*, mais ce serait au détriment de la résolution, et donc de la qualité, ce qui n'est bien entendu pas acceptable.

Nous avons conçu un circuit séparé pour la commande du volume et de la balance par télécommande. Ce circuit sera décrit dans le prochain numéro d'Elektor, il est aussi sur le site du labo [3].
Un dernier détail : l'alimentation symétrique des amplis op est filtrée par une inductance en mode commun et des filtres LC (un par canal).

Le circuit imprimé

Nous n'avons besoin que d'une seule alimentation de 7 à 8 V pour la platine, le *RPi* ayant sa propre alimentation. Nous ne recommandons pas d'utiliser une alimentation de 9 à 12 V, bien que ce soit en théorie possible : la dissipation d'IC10 serait trop importante.
Les **figures 3** et **4** montrent les deux faces du circuit imprimé, dont les dimensions sont identiques à celles du *RPi*. Les connecteurs d'alimentation et de sortie sont d'un même côté du circuit, l'interrupteur DIP S1 de l'autre (là où se trouve sur le *RPi* l'antenne *Wi-Fi*). Le connecteur GPIO à 40 broches K5 est inséré du côté soudure (voir photos).
Les pattes de K1 à K4, K6 et K7 côté soudure doivent être coupées le plus possible à ras du circuit imprimé, pour éviter tout contact avec les connecteurs réseau et *USB* du *RPi* ; un morceau d'isolant entre les deux circuits serait même conseillé.
Nous avons utilisé des entretoises M2,5 de 17 mm avec bout fileté pour fixer le CN/A au *RPi*. Les îlots de soudure pour les condensateurs des filtres de sortie (C1, C2, C6, C9-C11, C14-C16) sont tels que divers modèles peuvent être utilisés, p. ex. CMS 0805 ou conventionnels au pas de 5 mm. Nous recommandons l'utilisation de condensateurs au polystyrène avec une tolérance de 1%.
La broche 1 des circuits intégrés est identifiable grâce à une petite bande blanche ; ce n'est pas facile à voir pour IC7, à cause de la présence de C38. Le numéro des résistances et condensateurs CMS 0805 et 1206 est à l'emplacement même du composant, il n'y avait pas assez de place à côté ; les numéros ne sont donc plus visibles une fois les composants montés. Pour le montage à la main, avec un pistolet à air chaud ou au four, il est préférable d'utiliser une copie agrandie de la photo du circuit imprimé côté composants, pour faciliter la mise en (bonne) place des CMS. Une fois les composants traversants soudés, ce qui se fait en dernier, il sera très difficile de dessouder la plupart des CMS 0805 !
JP1 et JP2 sont montés côté soudure ; ils peuvent en principe rester ouverts (*SCK* = 6 × *BCK*). Les connecteurs audio *RCA* sont de marque *Switchcraft* ; leurs pattes de connexion sont plus rapprochées que sur des modèles standard, où l'espacement est de 10 mm. Si vous utilisez des connecteurs d'une autre marque, il est probable qu'il ne soit pas possible de les insérer sur le circuit imprimé.
L'écran tactile de 3,5 pouces de chez *Waveshare* doit être monté 16 mm au-dessus du CN/A. Nous avons utilisé pour ce faire des entretoises M2,5 de 14 mm avec bout fileté, et des écrous M2,5 pour la distance additionnelle. Ne poussez pas le connecteur de l'écran au maximum vers le bas ! En cas de doute, il vaut mieux utiliser un connecteur supplémentaire et monter l'écran un peu

Figure 6. Le lecteur audio en réseau complet.

Volumio

Volumio est une distribution *Linux* à code ouvert (*open source*) gratuite, spécifiquement adaptée à la reproduction musicale. Le logiciel tourne sur de nombreux appareils, dont les mini-ordinateurs comme le Raspberry Pi. Après installation, l'appareil cible devient un lecteur audio « sans tête » (*headless audiophile music player*) ; il faut en effet un autre ordinateur, téléphone ou tablette pour le commander.

Ceci est possible grâce à l'interface utilisateur de *Volumio*, une application *web* qui tourne sur tout appareil disposant d'un navigateur. L'interface est simple et intuitive ; elle permet la reproduction de fichiers musicaux. Les communications entre *Volumio* et l'application web se font via le réseau local.

Les instructions pour l'installation de *Volumio* se trouvent sur la page du projet au labo [2]. Nous avons utilisé la version 1.55 ; la version 2, qui date de décembre 2016, ne supporte pas (encore) l'écran tactile utilisé dans ce projet.

La **figure 7** vous montre la commande de *Volumio* via l'écran tactile (à gauche) et un ordinateur (à droite).

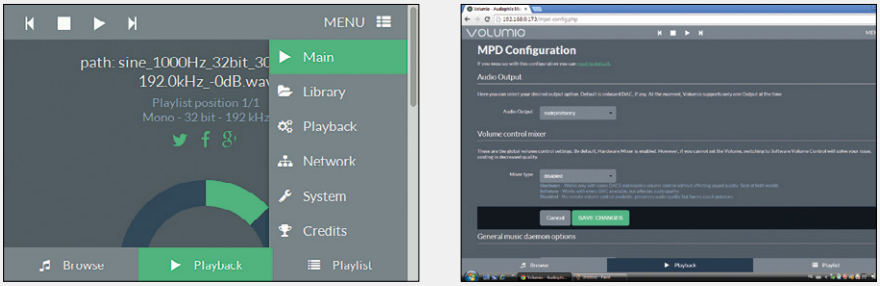


Figure 7. *Volumio* peut être commandé via l'écran tactile (à gauche), ou un ordinateur (à droite).

plus haut. La distance entre le connecteur à 26 broches de l'écran et le circuit imprimé du CN/A est d'environ 2 mm. Le résultat final est visible en **figures 5** et **6**.

Mesures

Nous avons bien entendu soumis notre CN/A audio à toute une batterie de tests. Les principaux résultats sont repris aux **tableaux 2** et **3**. La position des interrupteurs S1, utilisée pour les mesures, est résumée dans le **tableau 1**.

Le courant est proportionnel à la fréquence d'échantillonnage, voir **tableau 2**.

Le **tableau 3** résume les mesures de distorsion harmonique totale (DHT), augmentée du bruit, et de distorsion par intermodulation (DIM).

Sur la page du projet au labo [2], quelques graphiques obtenus avec un analyseur *Audio Precision* sont aussi disponibles. ⏮

(160198 – version française : Jean-Louis Mehren)

Liens

[1] www.ti.com/lit/ds/symlink/pcm1794a.pdf

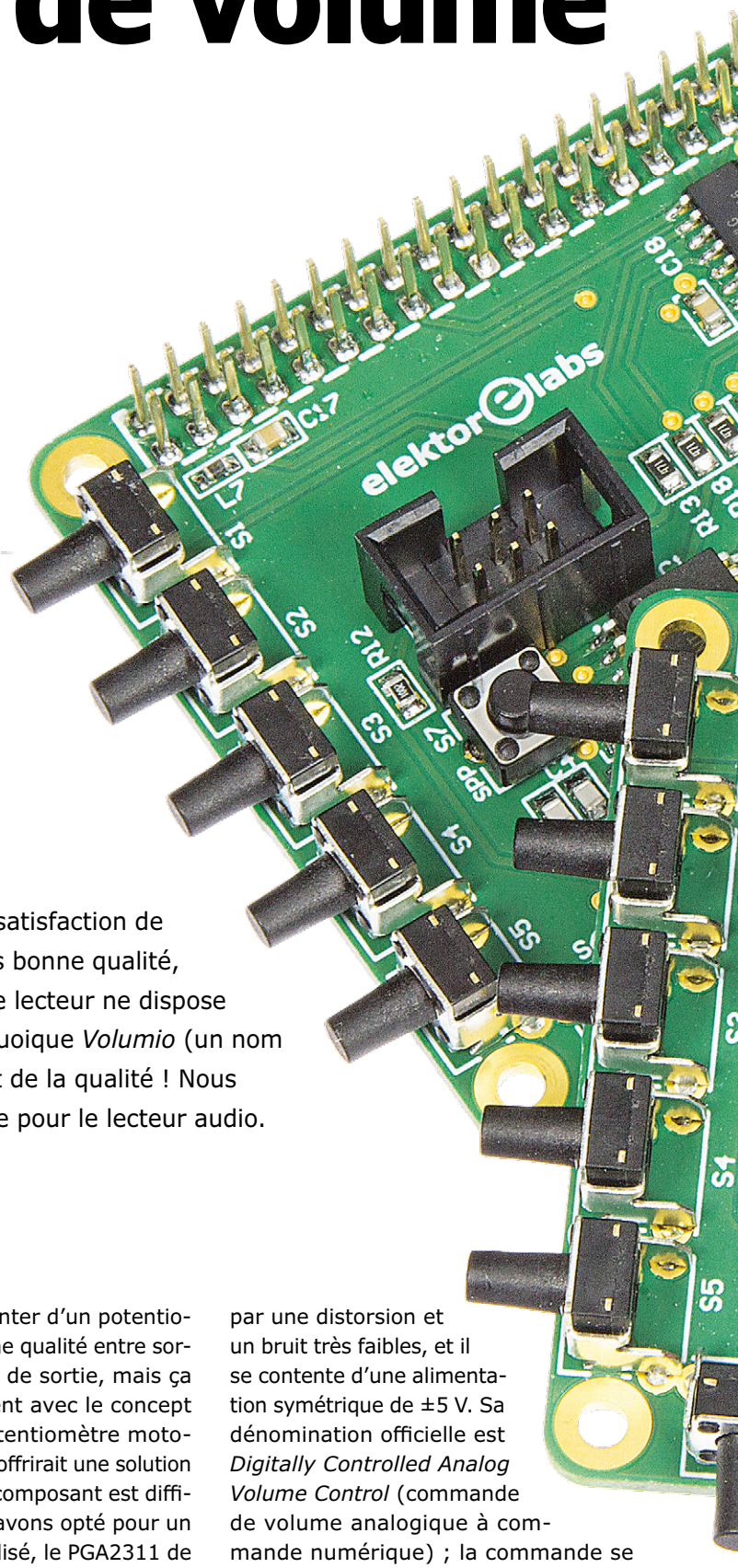
[2] www.elektormagazine.fr/labs/audio-dac-for-rpi-networked-audio-player-using-volumio

[3] www.elektormagazine.fr/labs/volume-control-160321-for-raspberry-pi-dac-160198

PROJET DU LABO

commande de volume pour le CN/A audio

faible ou fort, mais toujours de qualité



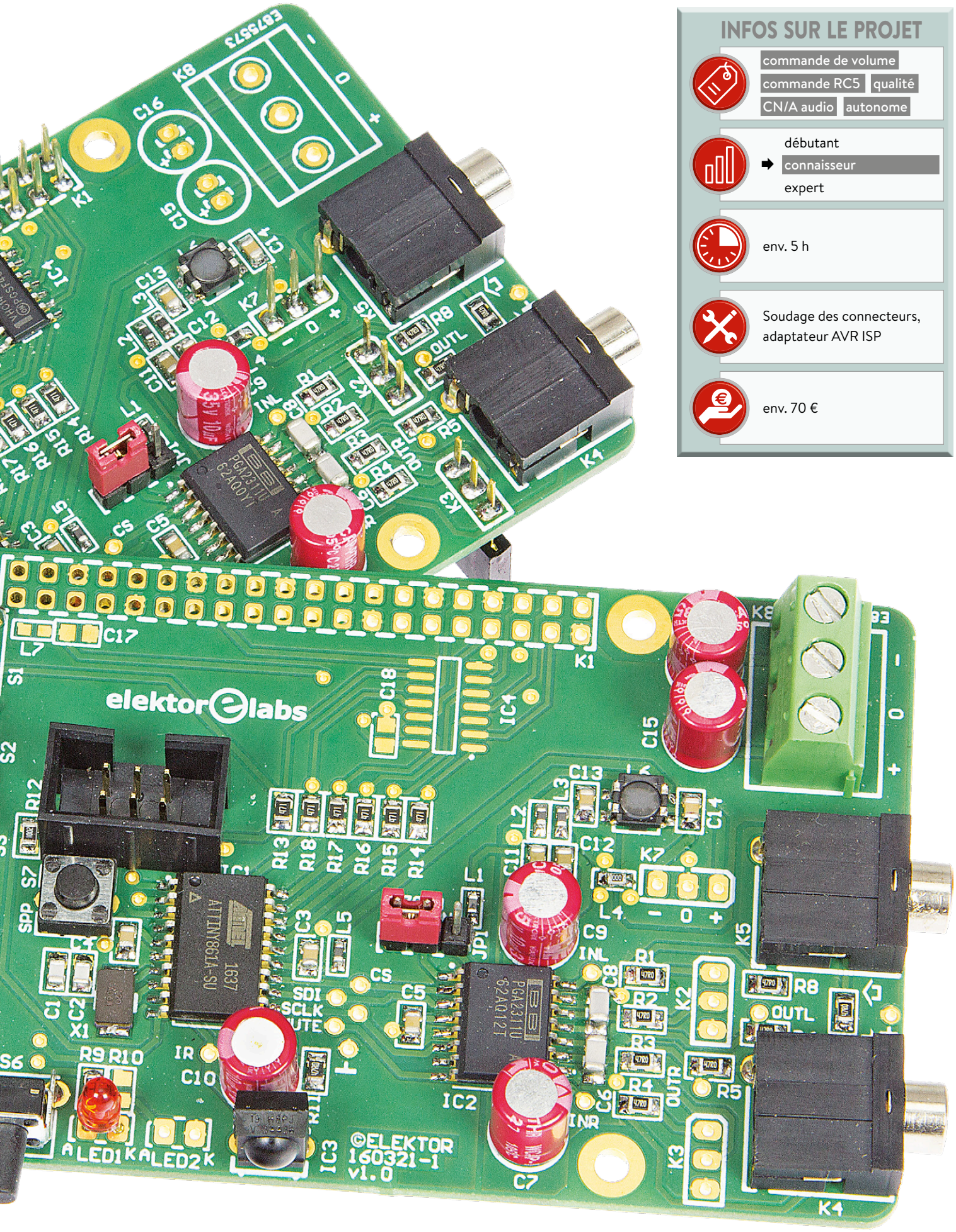
Ton Giesberts (labo d'Elektor)

Dans le numéro précédent, nous avons présenté – à la satisfaction de nombreux lecteurs – un lecteur audio sur réseau de très bonne qualité, sur la base d'une combinaison Raspberry Pi/*Volumio*. Ce lecteur ne dispose cependant pas d'une véritable commande de volume, quoique *Volumio* (un nom mal choisi ?) le permette par logiciel, mais au détriment de la qualité ! Nous avons donc décidé de réaliser une commande de volume pour le lecteur audio.

Caractéristiques

- Commande de volume d'excellente qualité
- Adapté au duo RPi + CN/A audio
- Construction en sandwich compacte
- Utilisation d'une télécommande RC5
- Possibilité d'utilisation autonome

On pourrait se contenter d'un potentiomètre stéréo de bonne qualité entre sortie du CN/A et ampli de sortie, mais ça ne cadre pas vraiment avec le concept de l'appareil. Un potentiomètre motorisé à télécommande offrirait une solution acceptable, mais ce composant est difficile à trouver. Nous avons opté pour un circuit intégré spécialisé, le PGA2311 de *Texas Instruments*. Ce circuit se distingue par une distorsion et un bruit très faibles, et il se contente d'une alimentation symétrique de ±5 V. Sa dénomination officielle est *Digitally Controlled Analog Volume Control* (commande de volume analogique à commande numérique) ; la commande se fait via une interface série, nous avons



INFOS SUR LE PROJET

commande de volume
commande RC5 qualité
CN/A audio autonome

débutant
➡ connaisseur
expert

env. 5 h

Soudage des connecteurs, adaptateur AVR ISP

env. 70 €

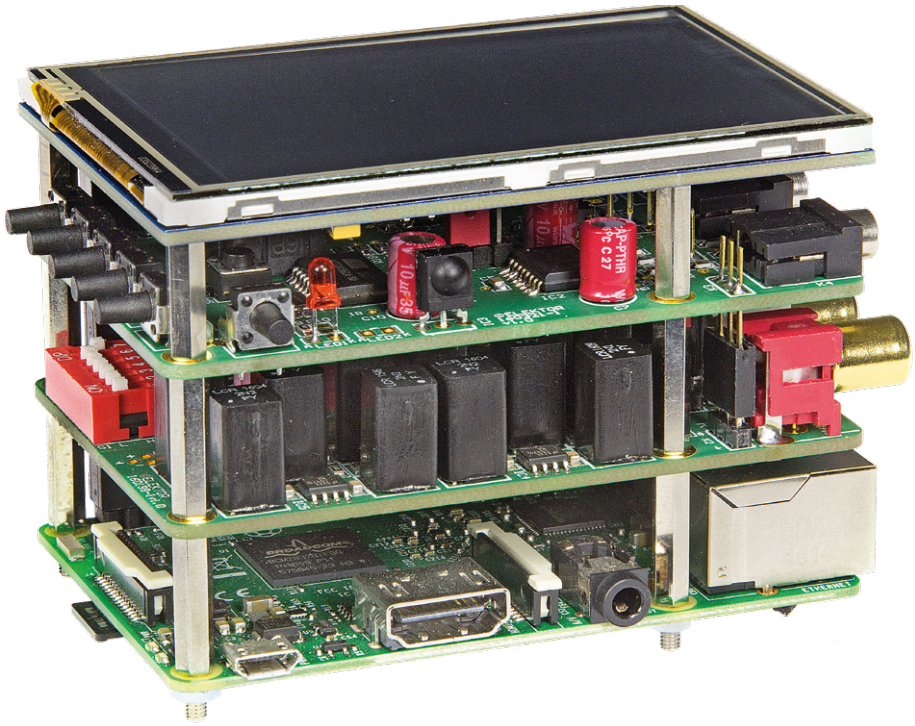


Figure 1. Le *Big DAC* dans toute sa splendeur.

choisi un microcontrôleur *ATtiny* d'*Atmel* pour la supervision. Le circuit détecte aussi le passage par zéro et possède une fonction de mise en sourdine.

Généralités

Cette commande de volume a été conçue pour le CN/A audio pour Raspberry Pi décrit dans le numéro précédent, et le circuit imprimé dessiné en conséquence : il peut être enfiché directement sur la carte du CN/A audio. Nous obtenons alors un gros sandwich, avec le Raspberry Pi, le CN/A audio, la commande de volume et l'écran ; le résultat est visible en **figure 1**. Sans beaucoup d'imagination, nous l'avons appelé *Big DAC*, par analogie à la marque commerciale d'une chaîne de restauration rapide bien connue.

Volumio – et/ou une autre application qui tourne sur le RPi – est commandé par six boutons-poussoirs, un septième sert au réglage du volume préférentiel.

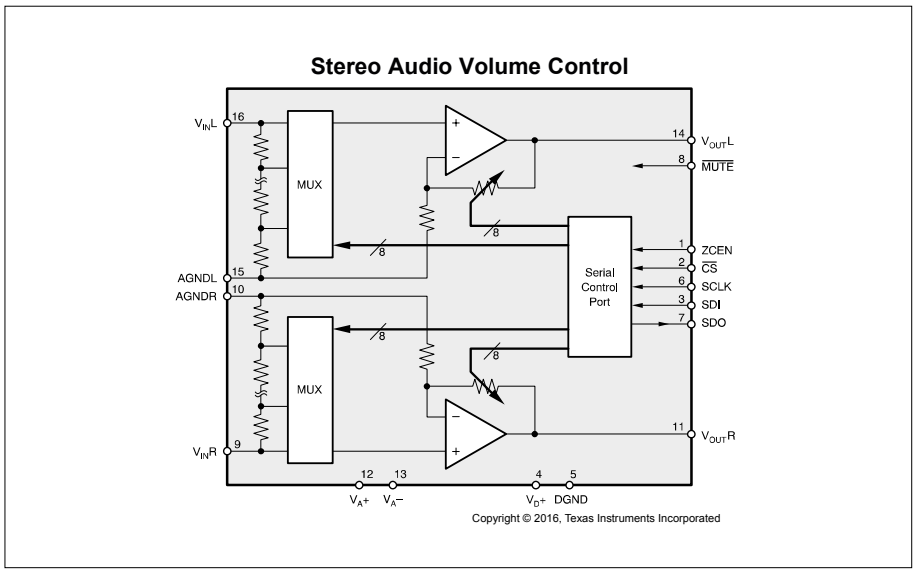


Figure 3. Le volume est commandé par des amplificateurs à gain variable.

Nous avons également prévu une télécommande infrarouge (protocole RC5). Autre bonne nouvelle, pour ceux qui voudraient une commande de volume autonome, sans RPi ni CN/A audio : c'est tout à fait possible avec notre projet, et on économise même quelques composants !

Schéma

Nous n'avons pas jugé utile d'inclure un schéma de principe ; le schéma de la commande de volume, repris en **figure 2**, peut sembler compliqué à un lecteur non averti, mais les apparences sont parfois trompeuses.

Nous allons décrire les éléments du circuit pas à pas, en commençant par l'indispensable : l'alimentation. Sans elle, le plus beau circuit au monde ne serait qu'inutile...

Alimentation

Si la commande de volume est utilisée avec le CN/A audio, aucune alimentation séparée n'est nécessaire. Les tensions pour le circuit intégré de commande de volume (+4,74 V et −4,81 V, notés *+4V7_A* et *−4V8_A* sur le schéma), et pour le microcontrôleur (+4,7 V, noté *+4V7_D*) proviennent du circuit imprimé du CN/A audio via le connecteur K7. Ce connecteur est soudé sur la face inférieure et s'enfiche sur le connecteur K6 du CN/A audio. La tension de +3,3 V nécessaire pour les inverseurs d'IC4, qui commandent les lignes GPIO du connecteur d'extension, provient du RPi (broche 1 du connecteur d'extension).

Les tensions d'alimentation du PGA2311 sont filtrées par la bobine en mode commun L6 et les bobines L2, L3 et L4. La bobine L5 filtre la tension d'alimentation du microcontrôleur, et L7 la tension de 3,3 V. L1 veille à un bon isolement entre les tensions numérique et analogiques du PGA2311. Le rôle de L4 est très important et un peu particulier : cette bobine prévient le risque de formation d'une boucle de masse HF ; c'est nécessaire puisque la masse des alimentations (K7) et les connexions de masse de K2 et K3 sont déjà reliées entre elles sur le circuit imprimé du CN/A audio.

Si on utilise cette carte de commande de volume avec le CN/A audio, il **ne** faut **pas** monter le connecteur K8, ni les condensateurs C15 et C16.

Si la commande de volume est employée seule, ce sont IC4, C17, C18, L7, K2, K3 et K7 qui **ne** doivent **pas** être montés.

Il est aussi conseillé de remplacer L4 par une résistance de 0 Ω. N'oubliez pas qu'une alimentation symétrique séparée de ±5 V est requise pour une utilisation de manière autonome.

PGA2311

Le schéma fonctionnel interne du PGA2311 est repris en **figure 3**. Nous avons deux amplificateurs (stéréo oblige), dont le gain est fixé par une série d'entrées de commande (*/CS, SCLK, SDI* et */MUTE*), gérées par le microcontrôleur IC1.

Le PGA2311 dispose d'un détecteur de passage par zéro, pour minimiser le

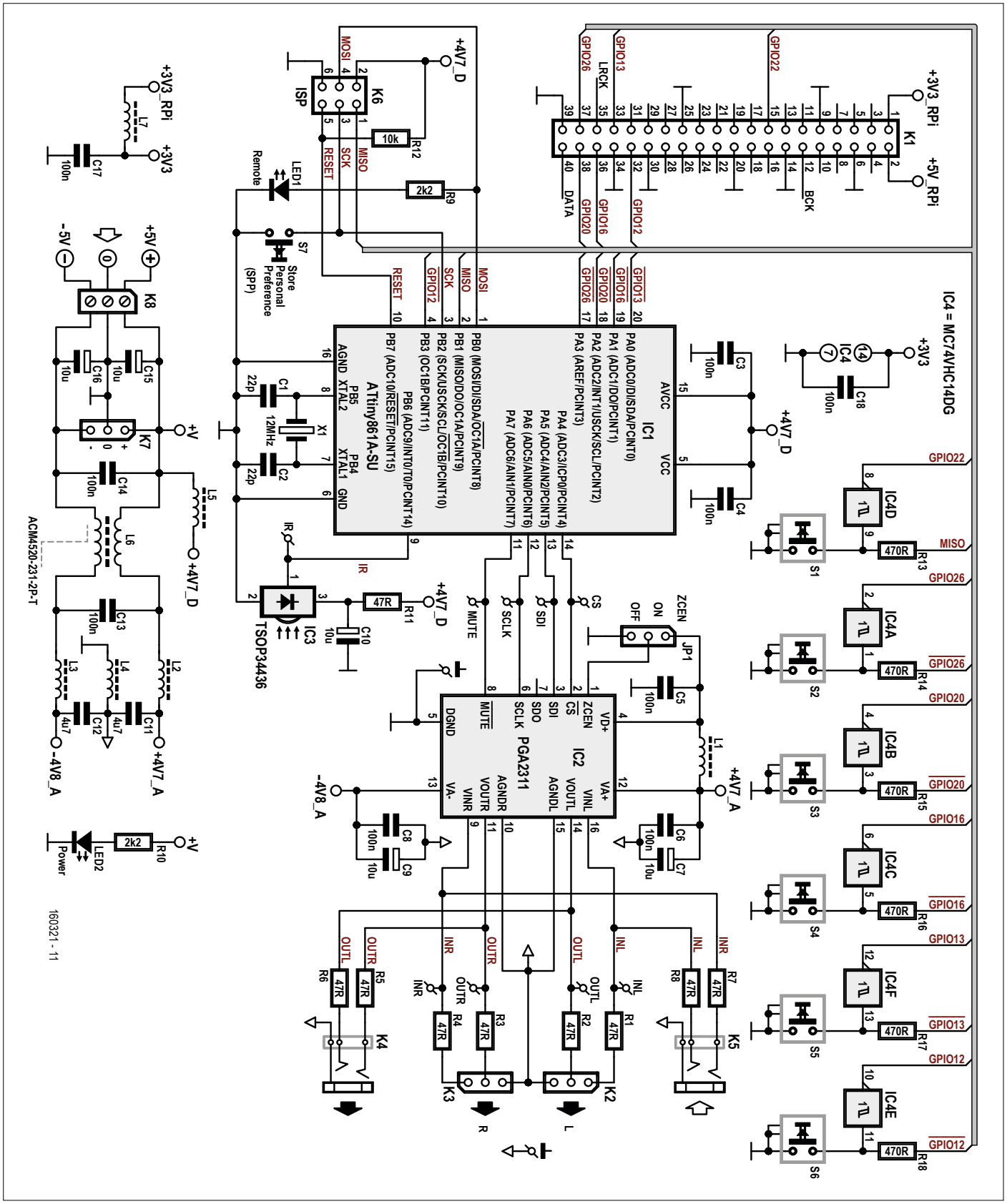


Figure 2. Le cœur du montage est – comment pourrait-il en être autrement ? – un microcontrôleur.

bruit lors d'un changement de volume. Lorsque la fonction est activée (*ZCEN – Zero Crossing Enable*, relié au +4,7 V), le changement ne se produit qu'après le prochain passage par zéro dans le sens positif du signal d'entrée. La fonction peut être désactivée en reliant *ZCEN* à la masse (cavalier JP1).

Pour des explications plus détaillées concernant le fonctionnement du PGA2311, reportez-vous à la fiche de caractéristiques du circuit [1].

Des points de test des entrées de commande $\overline{CS}$, *SCLK*, *SDI* et $\overline{MUTE}$ sont prévus sur le circuit imprimé, entre PGA2311 et microcontrôleur.

Les entrées et sorties analogiques sont reliées à deux connecteurs différents via

LISTE DES COMPOSANTS

Résistances

R1-R8, R11 = 47 Ω, ¼ W, 1%, CMS 0805
R9, R10 = 2,2 kΩ, 100 mW, 5%, CMS 0805
R12 = 10 kΩ, 100 mW, 5%, CMS 0805
R13-R18 = 470 Ω, 100 mW, 5%, CMS 0805

Condensateurs

C1, C2 = 22 pF, 50 V, 5%, C0G/NP0, CMS 0805
C3-C5, C13, C14, C17, C18 = 100 nF, 50 V, 10%, X7R, CMS 0805
C6, C8 = 100 nF, 25 V, 5%, C0G/NP0, CMS 1206
C7, C9, C10, C15, C16 = 10 µF, 35 V, 0,04 Ω, diam. 6,3 mm max., pas de 2/2,5 mm, Würth Elektronik 870055673001 (série WCAP-PTHR)
C11, C12 = 4,7 µF, 25 V, 20%, X5R, CMS 0805

Bobines

L1-L5, L7 = 600 Ω à 100 MHz, 0,15 Ω, 1,3 A, CMS 0603, Murata BLM18KG601SN1D
L6 = ACM4520-231-2P-T (TDK), 2 x 0,05 Ω, 230 Ω à 100 MHz, 2,6 A, CMS

Semi-conducteurs

LED1 = LED rouge, T-1, 3 mm
LED2 = LED verte, low-power, CMS 0805
IC1 = ATtiny861A-SU, SOIC-20, avec micrologiciel 160321-11 (ou 160321-12)
IC2 = PGA2311UA, SOIC-16
IC3 = TSOP34436, boîtier époxy à 3 broches
IC4 = M74VHC14DG, CMS SOIC-14 (ON Semiconductor)

Divers

K1 = connecteur GPIO à 40 broches, 2x20, femelle, extra haut
K2, K3, K7 = embase femelle à 3 contacts, verticale, au pas de 2,54 mm (soudage sur la face inférieure du circuit imprimé), avec longues broches traversantes
K4, K5 = connecteur pour jack audio stéréo de 3,5 mm, pour circuit imprimé, KLBR 4 Lumberg
K6 = connecteur HE10 mâle à 2 x 3 contacts, vertical, au pas de 2,54 mm
K8 = bornier à vis à 3 contacts, au pas de 5,08 mm, 630 V
JP1 = barrette mâle à 3 contacts, au pas de 2,54 mm
LED1 = barrette mâle à 2 contacts, au pas de 2,54 mm
JP1 = cavalier, au pas de 2,54 mm
S1-S6 = bouton-poussoir, SPST, FSMRA4JH TE Connectivity
S7 = bouton-poussoir, SPST, FSM4JRT TE Connectivity
X1 = quartz à 12 MHz, miniature, 5 x 3,2 mm, CMS, C_{last} = 18 pF
4 x entretoise M2,5 de 17 mm (05.12.173 Ettinger)
4 x vis M2,5 de 6 mm
4 x écrou M2,5
Circuit imprimé, réf. 160321-1

En option

Pour soutenir l'écran LCD : 4 x entretoise M2,5 de 14 mm (05.12.143 Ettinger), au lieu des 4 vis

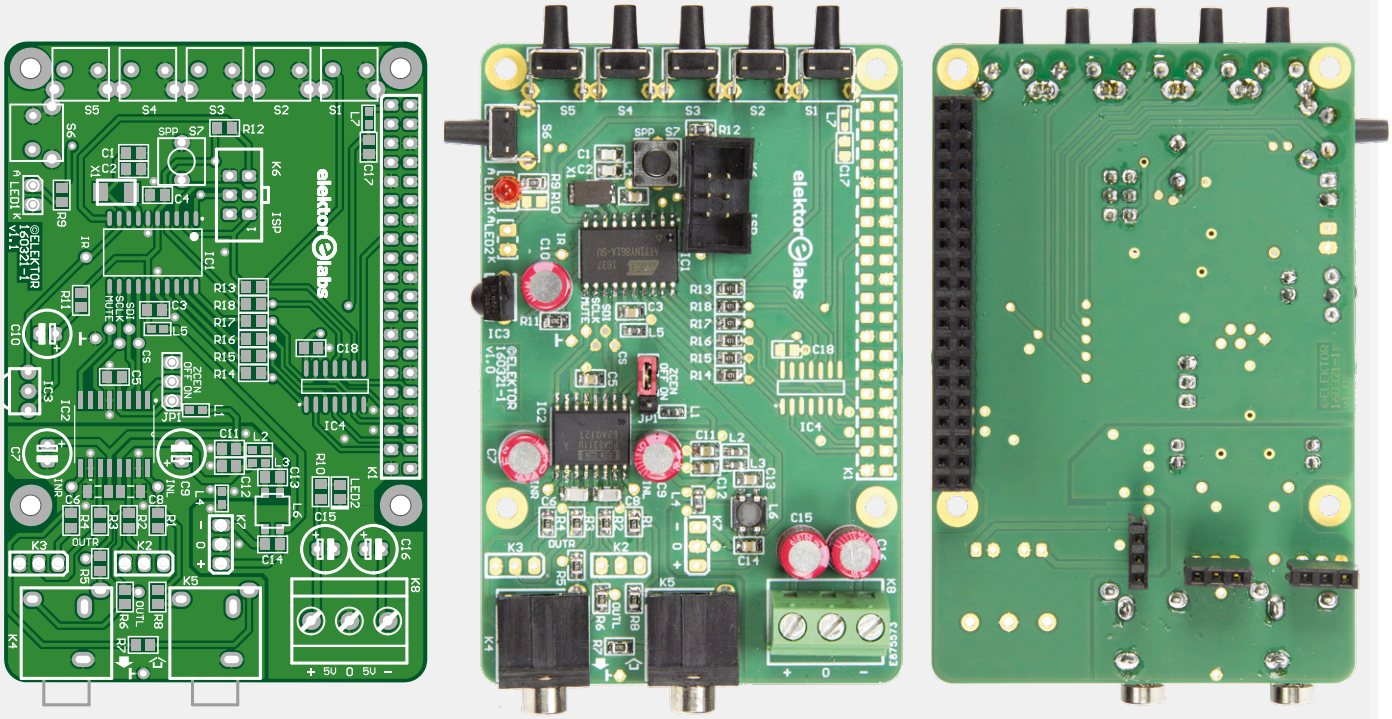


Figure 4. Le circuit imprimé s'adapte parfaitement à celui du CN/A audio.

des résistances de 47 Ω (R1-R8). Les connecteurs femelles à trois contacts que l'on peut imbriquer (*stack through*), K2 et K3, relient les entrées et sorties du CN/A audio (K3 et K4) et la commande de volume.
Les entrées et sorties d'IC2 sont en outre connectées à des jacks de 3,5 mm, présents sur le circuit imprimé (K4 et K5). Le signal de sortie original du CN/A peut donc être prélevé sur K5 ou bien sur les deux connecteurs audio RCA du CN/A audio. Le signal de sortie de la commande de volume se retrouve sur K4.
En mode autonome, l'entrée du PGA2311 se fait via K5, la sortie sur K4. Les résistances de 47 Ω protègent les sorties d'IC2 contre les effets d'une charge capacitive, et isolent les divers connecteurs entre eux.

Montage

Comme nous l'avons déjà dit, les dimensions du circuit imprimé sont identiques à celles du RPi et du CN/A audio (**fig. 4**). On peut donc constituer un sandwich, avec l'écran sur le dessus ; la **figure 5** montre le résultat d'ensemble.
Attention : les picots des connecteurs femelles de type *stack through* K2, K3 et K7 doivent être suffisamment longs pour que les connecteurs puissent être montés à une certaine distance du circuit imprimé (côté soudures), afin qu'ils s'enfichent correctement sur les connecteurs correspondants du CN/A audio. Il faudra bien entendu veiller à ce qu'ils restent bien parallèles au circuit imprimé.
Le même principe est valable pour le connecteur d'extension GPIO (K1), monté également côté soudures
Les circuits imprimés et l'écran sont solidement fixés les uns aux autres, séparés par des entretoises.

Le microcontrôleur

Pour commander le PGA2311, un microcontrôleur s'impose ; celui-ci enverra les bonnes données à l'interface série d'IC2. Nous avons choisi l'ATtiny861A d'*Atmel*, un circuit intégré à 20 broches, qui possède juste assez de lignes d'E/S pour l'interface série, les sept boutons-poussoirs, le récepteur infrarouge pour la télécommande, et une LED. Cette dernière est un modèle traversant ; avec un petit connecteur et du fil électrique, elle pourra être placée à un endroit au choix de l'utilisateur.
Les lignes de programmation in situ (*In-System Programming – ISP*) sont aussi toutes utilisées, sauf *Reset*. Lorsque l'on programme le microcontrôleur, via K6, il faut veiller à ne pas toucher les boutons-poussoirs.
Le récepteur IR (IC3) est un modèle standard de *Vishay*, un TSOP34436. La sortie est également un point de test. Le protocole utilisé pour la télécommande est le RC5, pour lequel une commande bien pratique existe en BASCOM (le langage dans lequel le logiciel a été écrit) :

```
Getrc5(Address, Command)
```

Deux versions du logiciel sont disponibles, en fonction de la version de *Volumio* ; elles peuvent être téléchargées sur la page du projet [3]. Nous les passons en revue ci-dessous.

Micrologiciel 160321-12

Cette version du logiciel a été écrite pour *Volumio 2* [2], qui dispose d'un module d'extension permettant d'attribuer une des fonctions ci-après à la plupart des broches GPIO (**fig. 6**) : *play/pause*, *volume +/–*, *previous*, *next* et *shutdown*.
Comme notre CN/A audio ne dispose pas de son propre réglage de volume, nous pouvons utiliser quatre fonctions pour commander *Volumio* ; pour d'autres CN/A avec commande de volume intégrée, nous avons aussi ajouté les fonctions *volume +* et *volume –*. Ces fonctions peuvent être attribuées à la plupart des broches GPIO, tant qu'elles ne sont pas utilisées pour l'I²S ou l'écran LCD tactile (rappelons encore que la version de *Volumio* utilisée, 2.041 du 12 décembre 2016, ne supporte pas encore notre écran LCD).
Les broches GPIO – dont la fonction est rappelée dans le **tableau 1** – sont pourvues de résistances de rappel.

Nous utilisons les commandes RC5 du **tableau 2**.

Figure 5. Le fameux sandwich !

Une commande de volume d'excellente qualité sur une surface réduite

Tableau 1 : connecteur d'extension du Raspberry Pi				
K1	**RPi (v3)**	**160321-12**	**I²S**	**LCD 3,5"**
1	3,3 V	3,3 V		3,3 V
2	5 V			5 V
3	GPIO2 (SDA1)			NC
4	5 V			5 V
5	GPIO3 (SCL1)			NC
6	GND	GND	GND	GND
7	GPIO4 (GPIO_GCLK)			NC
8	GPIO14 (TXD0)			NC
9	GND	GND	GND	GND
10	GPIO15 (RXD0)			NC
11	GPIO17 (GPIO_GEN0)			TP_IRQ
12	GPIO18 (GPIO_GEN1)		BCK	NC
13	GPIO27 (GPIO_GEN2)			NC
14	GND	GND	GND	GND
15	GPIO22 (GPIO_GEN3)	S1/Play/Pause		NC
16	GPIO23 (GPIO_GEN4)			NC
17	3,3 V			3,3 V
18	GPIO24 (GPIO_GEN5)			LCD_RS
19	GPIO10 (SPI_MOSI)			LCD_SI/TP_SCK
20	GND	GND	GND	GND
21	GPIO9 (SPI_MISO)			TP_SO
22	GPIO25 (GPIO_GEN6)			RST
23	GPIO11 (SPI_SCLK)			LCD_SCK/TP_SCK
24	GPIO8 (SPI_CE0_N)			LCD_CS
25	GND	GND	GND	GND
26	GPIO7 (SPI_CE1_N)			TP_CS
27	ID_SD			
28	ID_SC			
29	GPIO5			
30	GND	GND	GND	
31	GPIO6			
32	GPIO12	S6/Shutdown		
33	GPIO13	S5/Volume up		
34	GND	GND	GND	
35	GPIO19		LRCK	
36	GPIO16	S4/Volume down		
37	GPIO26	S2/Previous track		
38	GPIO20	S3/Next track		
39	GND	GND	GND	
40	GPIO21		DATA	

Tableau 2 : commandes RC5	
Play/pause	RC5 command 1 (Digit entry)
Previous track	RC5 command 2 (Digit entry)
Next track	RC5 command 3 (Digit entry)
Shutdown	RC5 command 12 (Standby)
Mute/de-mute	RC5 command 13
Personal preference settings	RC5 command 14
Increase sound volume	RC5 command 16
Decrease sound volume	RC5 command 17
Shift sound balance to the right	RC5 command 26
Shift sound balance to the left	RC5 command 27

Sur le bus GPIO, un niveau logique haut induit une tension de 3,3 V. Nous devons donc adapter le niveau, ce que nous faisons avec un sextuple inverseur à bascule de Schmitt, un MC74VHC14DG d'*ON Semiconductor*. Les entrées de ce circuit sont protégées contre les surtensions jusqu'à 7 V, ce qui est parfait pour nos besoins.

Les lignes d'E/S correspondantes du microcontrôleur doivent être à l'état haut au repos (inactives) ; c'est le cas des quatre lignes utilisées comme sortie (*PA2*, *PA3*, *PB1* et *PB3*). Les résistances de rappel pour deux lignes qui sont utilisées comme entrée pour la commande de volume (*PA0* et *PA1*) doivent être activées.

Les boutons-poussoirs S1 à S6 sont connectés au microcontrôleur via des résistances de 470 Ω ; celles-ci empêchent le court-circuitage des sorties du microcontrôleur lorsqu'un bouton-poussoir est pressé, ce qui met à la masse l'entrée de l'inverseur correspondant. La valeur des résistances est suffisamment basse pour que les sorties soient à l'état bas.

Lorsqu'on appuie sur S7, le volume à ce moment-là est enregistré comme volume préférentiel (ce niveau est utilisé au démarrage de l'appareil). Ce bouton-poussoir n'est pas aligné avec les autres, parce que nous considérons qu'il sera utilisé moins souvent. La LED1 s'illumine quatre fois lorsque S7 est activé, indiquant que le niveau du volume est enregistré dans la mémoire EEPROM du microcontrôleur.

Lors d'une première utilisation du microcontrôleur, le niveau standard est de −20 dB ; c'est assez élevé, ne soyez pas surpris !

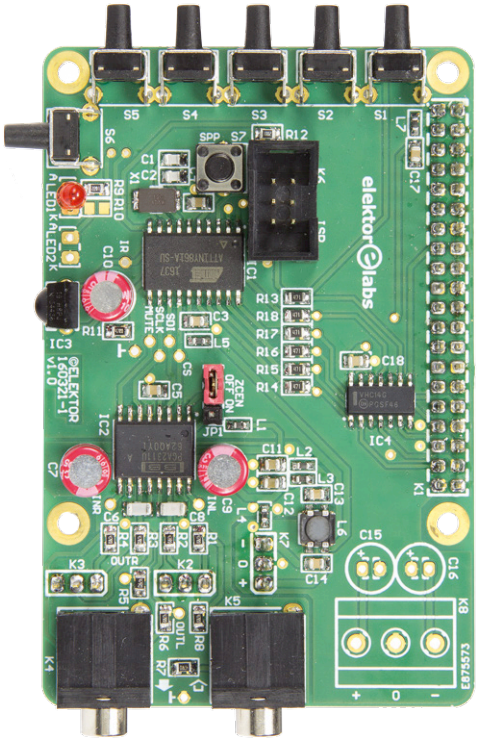
La LED1 s'illumine lors de la réception d'une commande RC5 (pas nécessairement une commande utilisée par la commande de volume). Le récepteur IR, adapté au protocole RC5, est connecté à *PB6*, car la commande *Getrc5* utilise *TIMER0* et son interruption. Si vous souhaitez utiliser un autre protocole pour la télécommande, choisissez un récepteur IR adapté à ce protocole ; les connexions sont quant à elles standard.

Lors de l'écriture du logiciel, nous avons noté un comportement anormal : si « *divide clock by 8* » est désactivé, la commande *Getrc5* ne fonctionne pas correctement. Lors de la programmation des fusibles, la case *CKDIV8* doit donc être cochée (**fig. 7**). C'est aussi la raison pour

laquelle la fréquence du quartz X1 a été choisie à 12 MHz.

Micrologiciel 160321-11

Cette version est en principe prévue pour la version 1.55 de *Volumio*, utilisée avec le RPi et le CN/A audio (voir numéro précédent). Elle pourrait bien entendu être utilisée avec *Volumio 2*, ou pour une autre application. Dans ce dernier cas, n'oubliez pas une alimentation séparée de ±5 V, isolée afin d'éviter le risque d'une boucle de masse HF. La bobine L4 devra aussi être remplacée par une résistance de 0 Ω (CMS 0603), ou un pont de soudure.
Après avoir monté K4 et K5 et connecté les appareils de votre chaîne audio avec



des câbles appropriés, le volume peut maintenant être commandé à distance (la tension maximale V_{max} est de 2 V). On peut aussi utiliser K2 et K3 plutôt que les prises jack de 3,5 mm ; il faut alors monter des connecteurs ou souder les fils directement sur le circuit imprimé.
Les boutons-poussoirs S1 à S6 ne concernent que la commande de volume ; les fonctions et les commandes RC5 associées sont reprises dans le **tableau 3**.

Le bouton S7 est toujours utilisé pour l'enregistrement du niveau de volume préférentiel.

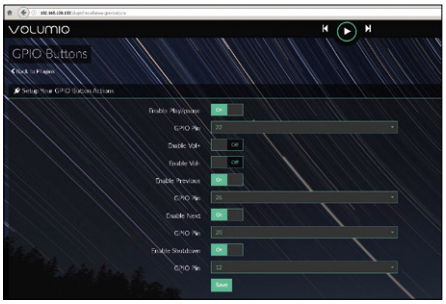


Figure 6. Le module d'extension GPIO de *Volumio*.

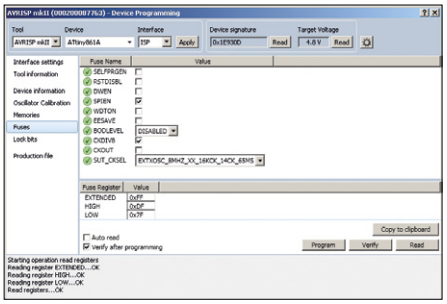


Figure 7. Important : la programmation des fusibles.

Si la fonction de sourdine est activée (broche 8 d'IC2), matériellement ou via le logiciel, les tampons internes du PGA2311 sont désactivés, et V_{OUTL} et V_{OUTR} mis à la masse via une résistance de 10 kΩ. Le bruit de l'amplificateur connecté en amont de la commande de volume pourrait alors augmenter.

Tableau 3 : boutons-poussoirs et commandes RC5 (logiciel 160321-11)		
S1	personal preference	RC5 command 14
S2	balance to the left	RC5 command 27
S3	balance to the right	RC5 command 26
S4	volume down	RC5 command 17
S5	volume up	RC5 command 16
S6	mute	RC5 command 13

Tableau 4 : résultats de quelques mesures	
Consommation	
LED1 éteinte	11 mA
LED1 allumée	12 mA
Tension de sortie maximale (1 kHz, THD = 0,1%)	
amplification nulle (0 dB)	2,3 V
amplification de 20 dB	2,6 V
atténuation de −20 dB, 3,9 V en entrée	0,39 V
THD+N	
1 kHz, B = 22 kHz	0,0003% (0 dB, V_{sortie} = 2 V)
1 kHz, B = 80 kHz	0,00042% (0 dB, V_{sortie} = 2 V)
20 kHz, B = 80 kHz	0,0022% (0 dB, V_{sortie} = 2 V)
1 kHz, B = 22 kHz	0,0016% (20 dB, V_{sortie} = 2 V)
1 kHz, B = 80 kHz	0,0025% (20 dB, V_{sortie} = 2 V)
20 kHz, B = 80 kHz	0,0035% (20 dB, V_{sortie} = 2 V)
IMD (50 Hz : 7 kHz = 4:1)	
amplification nulle (0 dB), $V_{entrée}$ = 2 V	0,0007%
amplification de 20 dB, $V_{entrée}$ = 200 mV	0,0038%
DIM (signal carré 3,15 kHz + sinus 15 kHz)	
amplification nulle (0 dB), $V_{entrée}$ = 2 V	0,0009%
amplification de 20 dB, $V_{entrée}$ = 200 mV	0,0008%
Bande passante (−3 dB)	
0 dB, triangle en sortie	1,1 MHz
20 dB	690 kHz
Diaphonie	
1 kHz	< −100 dB
20 kHz	< −90 dB

Volumio 2

Pour le moment, l'écran tactile de 3,5 pouces n'est pas supporté par *Volumio 2*. Lorsque le module supportant l'écran originel du RPi est installé, on peut raccorder un moniteur standard au connecteur HDMI du RPi.

Un conseil : ne raccordez pas de clavier ou de souris au RPi ; il faut même retirer le dongle de clavier/souris sans fil. L'installation doit être démarrée via l'interface utilisateur réseau (*Web UI*).

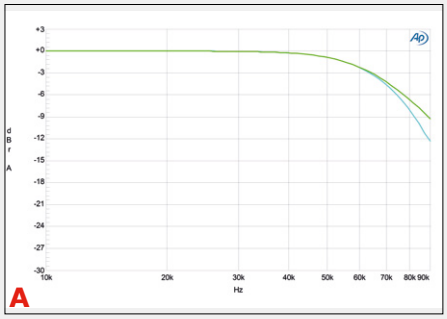
Mesures

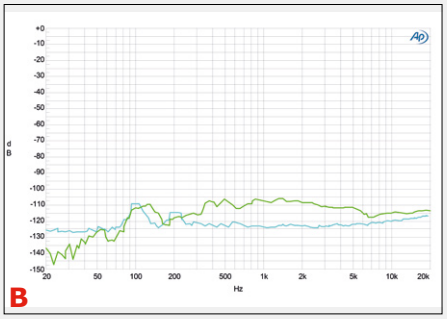
Nous avons réalisé quelques mesures sur notre commande de volume, comme nous l'avions fait pour le CN/A audio. Les résultats principaux sont repris dans le **tableau 4**.

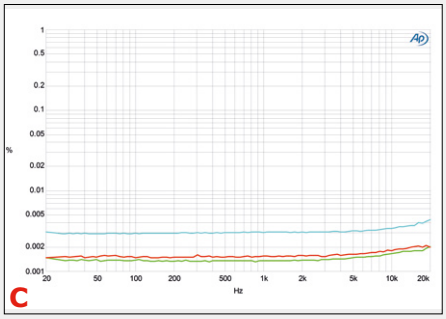
Notre analyseur de marque *Audio Precision* nous a aussi permis de réaliser quelques graphiques, pour la commande de volume, et pour le CN/A audio.

(160321 – version française : Jean-Louis Mehren)

Mesures sur le *Big DAC*







Un des trésors de notre labo est l'analyseur d'*Audio Precision*. Il n'a pas souffert du déménagement vers Aix-la-Chapelle, et nous avons pu l'utiliser pour quelques mesures sur le *Big DAC*, dont les résultats sont résumés ci-dessous.

CN/A audio

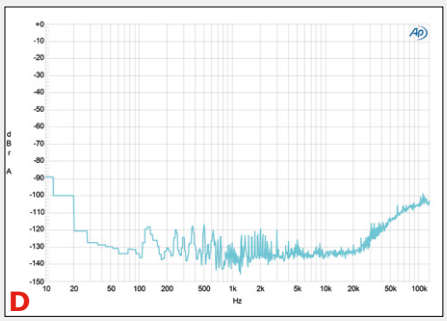
Le **graphique A** montre l'amplitude en fonction de la fréquence (signal sinusoïdal de 70 fréquences) pour une fréquence d'échantillonnage de 192 kHz. La courbe supérieure (en vert) est obtenue avec un filtre passe-bas à forte pente en sortie, la courbe inférieure (en bleu) avec un filtre moins sélectif. La fréquence de coupure est de 63,5 kHz (en théorie 64,5 kHz). À 90 kHz, l'atténuation est de 9,2 dB (vert) et 12,3 dB (bleu) ; à 45 et 51 kHz, l'atténuation est pour les deux filtres respectivement de 0,5 et 1 dB.

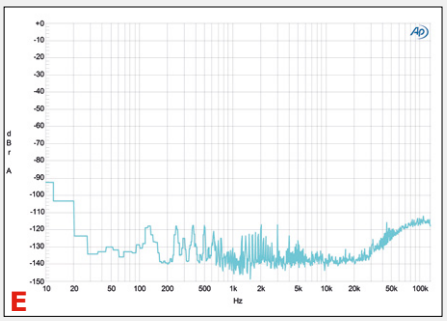
Le **graphique B** montre la séparation entre les canaux en fonction de la fréquence. La courbe supérieure (en vert) montre le couplage du canal gauche vers le droit, qui est un rien supérieur à l'inverse suite à une différence de longueur de pistes entre circuit intégré et sortie (la piste du canal droit est plus longue). Le résultat est cependant très bon pour un montage aussi compact.

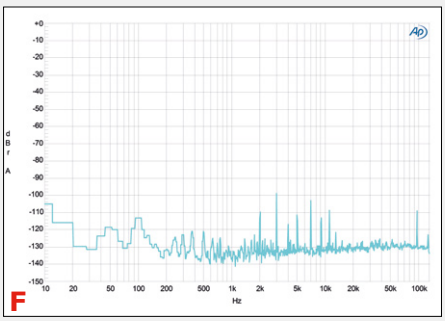
Le **graphique C** montre la distorsion harmonique totale (DHT) additionnée du bruit en fonction de la fréquence, pour des fréquences d'échantillonnage de 48, 96 et 192 kHz. La bande passante est de 80 kHz. Avec une bande passante de 22 kHz et une fréquence d'échantillonnage de 48 kHz, les résultats sont meilleurs.

Le **graphique D** montre la transformée de Fourier rapide (*FFT*) d'un signal sinusoïdal de 1 kHz à amplitude maximale et pour une fréquence d'échantillonnage de 32 kHz. Il est à noter que le bruit augmente en dehors de la bande des fréquences audio ; il s'agit sans doute d'une mise en forme du bruit (*noise shaping*). Avec une bande passante de 22 kHz, DHT + bruit = 0,0007%, 0,012% pour une bande passante de 80 kHz. Il y a un certain nombre de pics d'interférence, causés par le RPi qui est juste en dessous du CN/A. Les harmoniques 2 et 3 de la fréquence d'échantillonnage sont invisibles.

Le **graphique E** est identique au précédent, mais avec une fréquence d'échantillonnage de 44,1 kHz. Le bruit en HF est inférieur, et les harmoniques 2 et 3 de la fréquence d'échantillonnage sont un peu plus prononcées. La DHT (sans bruit) n'est cependant que de 0,00019%.





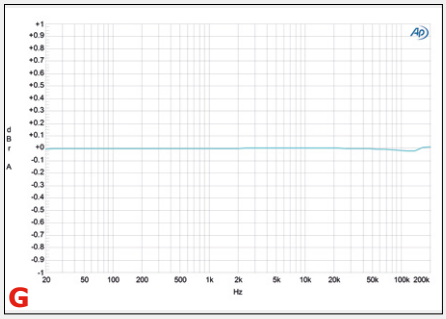


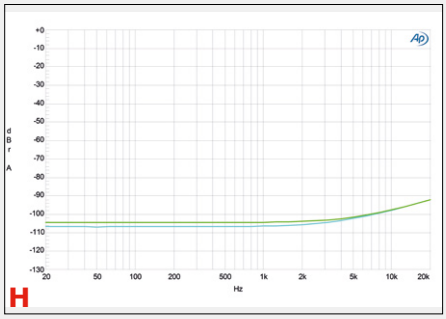
Liens

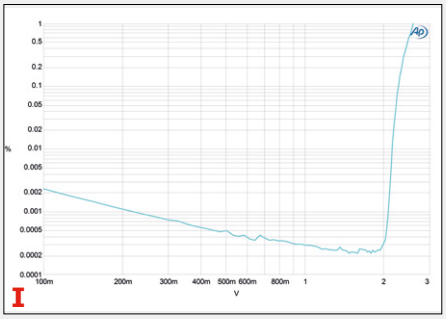
[1] www.ti.com/lit/ds/symlink/pga2311.pdf

[2] volumio.org

[3] www.elektormagazine.fr/160321







Le **graphique F** montre la transformée de Fourier rapide (*FFT*) d'un signal sinusoïdal de 1 kHz à amplitude maximale (sur 16 bits) à une fréquence d'échantillonnage de 32 kHz, mais converti à la volée en 24 bits/96 kHz (comme c'est possible avec *Volumio 1.55*). L'absence du bruit HF, qui était visible sur le graphique D, est remarquable.

Commande de volume

Le **graphique G** montre à nouveau l'amplitude en fonction de la fréquence, sans amplification (0 dB) et pour une tension de sortie de 2 V. La fréquence maximale de notre analyseur est de 200 kHz, et la petite bosse vers 100 kHz est sans importance (voyez l'échelle verticale...).

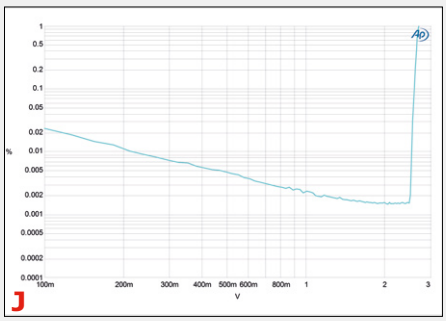
Le **graphique H** montre la diaphonie en fonction de la fréquence. La différence entre les deux courbes est d'environ 2 dB, le circuit est donc bien symétrique.

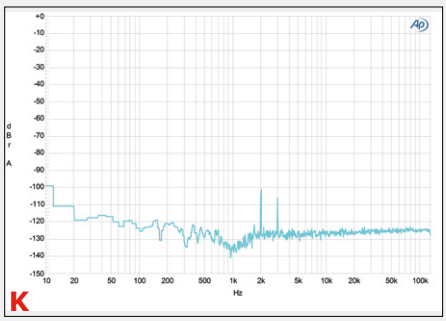
Le **graphique I** montre la DHT additionnée du bruit en fonction du niveau de sortie, sans amplification (0 dB). La bande passante a été réduite à 22 kHz, pour mieux voir où l'écrêtage commence. Avec une tension d'alimentation aussi faible que celle utilisée, la tension de sortie maximale est d'environ 1,9 V. Au-delà la distorsion augmente rapidement (0,1% à 2,3 V, 1% à 2,6 V).

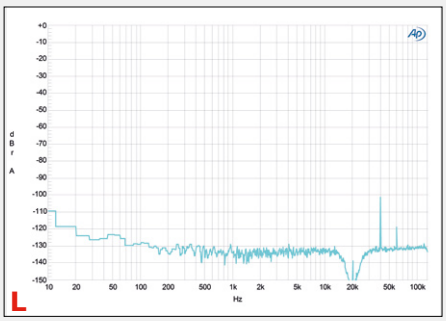
Le **graphique J** est identique au précédent, mais avec une amplification de 20 dB. La tension de sortie maximale sans augmentation de la distorsion est – et c'est surprenant – de 2,48 V avant écrêtage (DHT = 0,0015%). L'écrêtage se produit sans doute plus rapidement dans les étages d'entrée, puisque la tension d'entrée est plus élevée avec un facteur d'amplification réduit.

Le **graphique K** montre la transformée de Fourier rapide (*FFT*) d'un signal sinusoïdal de 1 kHz avec un gain de 20 dB et une tension de sortie de 2 V. On ne voit que les harmoniques 2 et 3, correspondant à une DHT (sans bruit) de 0,001%. La DHT plus le bruit pour une bande passante de 22 kHz sont de 0,0016%, 0,0025% pour une bande passante de 80 kHz.

Le **graphique L** est identique au précédent, pour un signal de 20 kHz. Les harmoniques 2 et 3 sont aussi bien visibles. DHT = 0,00086% ; DHT + bruit = 0,0035% (bande passante de 80 kHz).



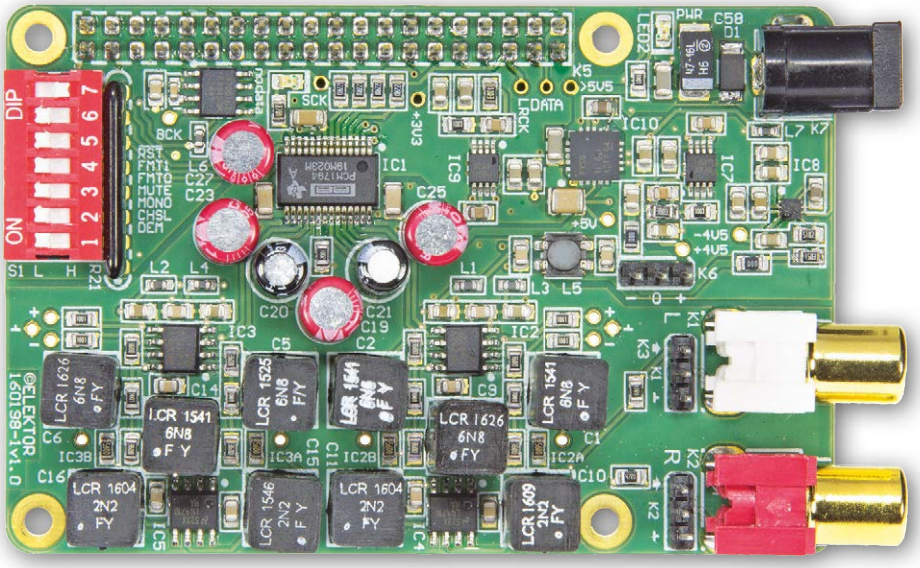




CN/A audio et commande de volume pour RPi d'Elektor ajustements et mises à jour

merci à tous pour vos questions et contributions

Compilé par **Ton Giesberts** (labo d'Elektor)



L'effet de synergie probable du croisement d'un poids lourd comme le Raspberry Pi avec l'art subtil de la conception audio haut de gamme, a fait que les articles *CN/A audio pour RPi* [1] et sa suite *Commande de volume pour le CN/A audio* [2] ont eu tous deux un succès fou. Mais du succès naissent les questions.

CN/A audio pour RPi, Elektor 7-8/2017.

Comme pour tous les projets audio personnels parus dans une publication papier et/ou en ligne, non seulement les utilisateurs affl uent pour acheter les cartes et jouer de la musique, mais ils réagissent aussi, dans un bon esprit d'ingénierie, avec d'utiles commentaires. En voici certains — attention, cela va terriblement vite.

RuneAudio

Un utilisateur a gentiment signalé que le réseau sans fi l (WLAN) ne fonctionne pas avec le système Volumio 1.55 ajouté au logiciel du projet en [3] (il s'agit d'`Elektor_Volumio_8GB_Image.zip` dans la section *Logiciels*). Après essai, en effet, le Wi-Fi ne fonctionne pas. Pour trouver une solution, j'ai refait l'installation de Volumio 1.55 comme décrit plus loin dans « Raspbian Stretch ne fonctionne pas ». J'ai utilisé `2017-07-05-raspbian-jessie` et l'interface graphique (GUI) de *Desktop Pixel* pour confi gurer le Wi-Fi. Pas de souci pour établir une connexion sans fi l ! Ensuite, j'ai installé Volumio. Pour installer l'écran tactile de 3,5 pouces TFT LCD de Waveshare, il faut le pilote `LCD-show-170703.tar.gz` [4]. J'ai confi guré le Wi-Fi et obtenu une adresse IP, même s'il a fallu quelques minutes. Le montage d'un NAS a aussi réussi. Mais la mise à jour d'une bibliothèque ne s'est pas terminée. Après redémarrage, toute l'interface Wi-Fi avait disparu…

Quelques jours plus tard, j'ai inséré la carte SD et réessayé, et Volumio a démarré normalement. J'ai configuré le Wi-Fi et étonnamment cela a fonctionné. J'ai obtenu une adresse IP et le NAS ajouté précédemment était accessible et opérationnel. Après avoir désactivé l'option « Attendre le réseau au démarrage » dans l'outil de configuration du logiciel du Raspberry Pi (`raspi-config`), le redémarrage, sans connexion LAN, a été ralenti de plus d'une minute. L'ajout de la ligne

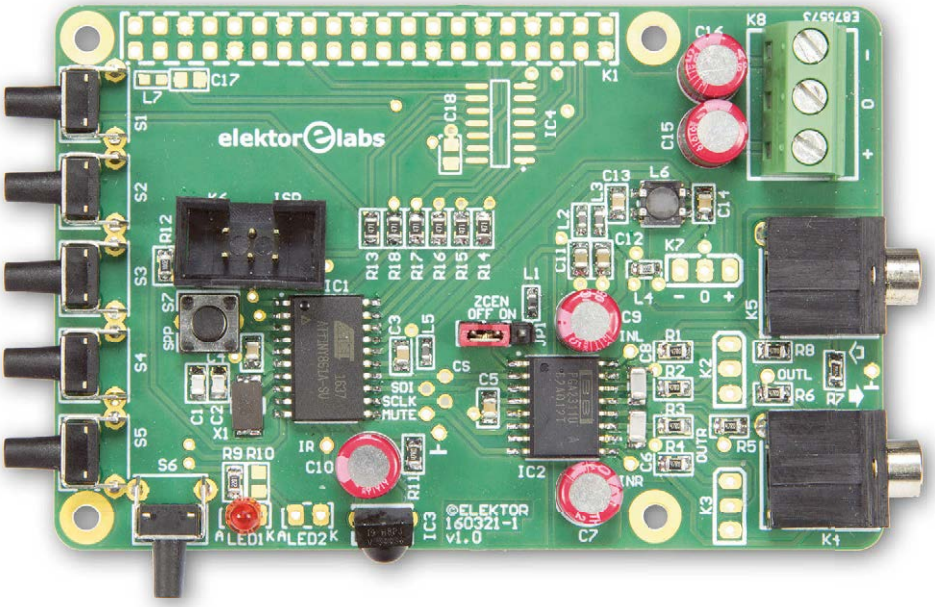
```
xserver-command=X -s 0 -dpms
```

dans

```
/etc/lightdm/lightdm.conf
```

pour désactiver l'économiseur d'écran a provoqué le gel de l'installation. Je ne suis pas sûr que cette image fonctionne correctement, mais par chance il y a une autre solution : RuneAudio ! J'ai effectué plusieurs tentatives pour faire marcher le Wi-Fi dans l'installation de Volumio 1.55, mais en vain. Le Wi-Fi fonctionne très bien dans l'installation de RuneAudio. Si vous voulez installer vous-même RuneAudio, consultez dans la section *Mises à jour* de la page du projet : *From the lab - using RuneAudio* [3]. Nous avons ajouté une image de 8 Go avec RuneAudio et l'écran tactile de 3,5 pouces TFT LCD de Waveshare pleinement fonctionnel : `Elektor_RuneAudio_8GB_Image.zip`. Voir la section *Logiciels*. Il est facile de configurer une connexion Wi-Fi dans RuneAudio.

Après avoir *flashé* l'image sur une carte SD d'au moins 8 Go, connectez d'abord le réseau filaire et effectuez les réglages pour le Wi-Fi (sous `MENU/NETWORK`). Il vaut mieux utiliser l'interface web. Saisissez l'adresse IP du RPi avec RuneAudio dans un navigateur de votre choix. On trouve l'adresse IP de la connexion LAN dans `MENU/Network` à côté d'`ETH0` sous `NETWORK INTERFACES`.
Lorsqu'on essaye d'entrer le mot de passe sur l'écran tactile de 3,5 pouces, un clavier occupe presque tout l'écran et on ne voit pas ce qu'on tape. Saisir des caractères ainsi est difficile, mais possible. Quelquefois tous les caractères ne sont pas acceptés. Pour moi un clavier USB a fonctionné. Une chose étrange est survenue lorsque j'ai sélectionné le menu réseau. `Scanning for networks` s'est affiché et quelques secondes plus tard plusieurs réseaux Wi-Fi de mon voisinage ont été listés, excepté le mien ! J'ai décoché l'option `Enable 20/40 MHz Coexistence` dans le menu des paramètres sans fil de mon routeur et peu après mon réseau Wi-Fi était aussi listé. Assurez-vous que de nouveaux dispositifs soient autorisés à se connecter à votre routeur. D'autres paramètres de sécurité ou de configuration du réseau peuvent entraîner la défaillance de la connexion et/ou que le SSID de votre réseau sans fil ne soit pas trouvé !
Déconnectez le câble LAN et redémarrez. RuneAudio devrait maintenant se connecter par l'interface sans fil. Après la mise sous tension, un bref écran de copyright avec un login apparaît. Attendez et soyez patient — cet écran peut rester plus d'une minute avant que le démarrage ne se poursuive. RuneAudio n'a pas de menu pour rééchantillonner et par conséquent tous les fichiers audio et les stations radio de l'internet (comme Dirble) doivent être encodés en 24 ou 32 bits, sinon il n'y a pas de son. Si vous installez vous-même l'écran tactile, dans l'image originale de RuneAudio vérifiez les configurations suivantes, nécessaires pour notre CN/A : (dans `Elektor_RuneAudio_8GB_Image`, ces réglages sont déjà effectués)



Commande de volume pour le CN/A audio, Elektor 11-12/2017

- dans `MENU/MPD/Audio Output`, sélectionnez `HiFiBerry DAC (I2S)` comme interface de sortie audio ;
- dans `MENU/MPD/Volume control`, sélectionnez `disabled` (désactivé) ;
- dans `MENU/SETTINGS/Features management`, activez `Local Browser` ;
- dans `MENU/SETTINGS/Features management`, activez `USB Automount`.

Rasbian Stretch ne fonctionne pas

Une nouvelle version de Raspbian pour le Raspberry Pi est sortie mi-août 2017. L'installation de Volumio 1.55 comme décrit dans le texte de ce projet avec le fichier `2017-08-16-raspbian-stretch` a échoué. Il manquait des paquets comme `php5-imagick`. Vous pouvez installer le navigateur Midori, mais il ne démarre pas lorsqu'on l'appelle dans : `~/.config/lxsession/LXDE-pi/autostart`.
Le lancement au démarrage du navigateur par défaut Chromium ne posait pas de problème en ajoutant ceci à `autostart` : `@/usr/bin/chromium-browser --kiosk --disable-restore-session-state http://localhost`

Volumio s'est lancé au démarrage, mais après l'installation du pilote d'affichage de `Waveshare LCD-show-170703.tar.gz`, `dhcpcd` n'a pas pu établir de connexion (pas sûr que le pilote soit en cause). Donc, pas de réseau. De plus, les clés USB ne se montaient pas. Après plusieurs heures, il était temps d'utiliser une version plus ancienne de Raspbian : `2016-09-23-raspbian-jessie.img`, sans remise à niveau ni mise à jour pour RPi. Ça a marché. L'encadré « **Recette pour un lecteur de musique autonome** » est une description à jour de l'installation de Volumio 1.55. Après cette installation, il faut changer quelques réglages pour que le CN/A audio joue dans Volumio :

- dans `Playback/Audio Output`, sélectionnez `sndrpihifiberry` ;
- dans `Playback/Volume control mixer/mixer type`, sélectionnez `disabled` ;
- dans `System/I2S driver`, sélectionnez `Hifiberry` ;
- dans `Library`, sélectionnez `UPDATE LIBRARY`.

Tous les changements dans une clé USB ou un NAS déconnectés/reconnectés nécessitent une mise à jour pour être visibles dans Volumio.

« The Sound of Silence ? »

Le lecteur Werner Jäger a demandé un conseil pour vérifier si des données audio sont réellement émises du RPi vers le module CN/A. Dans son cas, aucun son ne sortait du Volumio — même si le logiciel semblait bien fonctionner et que l'interface graphique s'affichait sur le LCD via Midori.
Werner a préparé une carte micro-SD de 16 Go et a lancé le logiciel sur un RPi 2B. Il a ensuite pris l'image Raspi trouvée sur le mini-DVD et suivi toutes les instructions décrites ici. Il a

Recette pour un lecteur de musique autonome (avec Volumio 1.55, RPi 2 et LCD)

Ou comment installer Volumio 1.55 sur un Raspberry Pi 2 avec un LCD de 3,5 pouces, 480×320 pixels, pour obtenir un lecteur de musique autonome. D'abord installez l'image `2016-09-23-raspbian-jessie.img` sur une carte micro-SD d'au moins 8 Go. On trouve l'image ici :
http://downloads.raspberrypi.org/raspbian/images/raspbian-2016-09-28/
Connectez un moniteur (HDMI), un clavier et une souris au RPi. Mettez sous tension le RPi. Lorsque le démarrage est terminé, ouvrez un terminal et lancez :

```
sudo apt-get update
sudo raspi-config
```

pour étendre la partition sur la totalité de la carte micro-SD et peut-être changer la configuration du clavier, en anglais par défaut. On peut aussi activer SSH ici. Je recommande fortement de changer le mot de passe par défaut pour des raisons de sécurité. Après la configuration du LCD, travailler depuis un ordinateur distant à travers une connexion SSH peut faciliter la saisie de commandes dans un terminal. Les caractères sur le LCD de 3,5 pouces sont un peu petits. Le login par défaut reste : pi/raspberry.

```
sudo reboot
sudo apt-get install midori
```

(un navigateur internet que nous savons lancer en plein écran)

```
sudo nano ~/.config/lxsession/LXDE-pi/autostart
```

Ajoutez les lignes suivantes (n'oubliez pas de mettre # en tête de la ligne **xscreensaver**…) :

```
#@xscreensaver -no-splash
@xset s off #disable screensaver
@xset -dpms
@xset s noblank #disable screensaver blanking
@unclutter -idle 0 #disable mouse pointing
midori -e Fullscreen & ZoomOut & ZoomOut -
   a http://localhost
```

Pour désactiver l'économiseur d'écran en permanence :

```
sudo nano /etc/lightdm/lightdm.conf
```

et ajouter dans la section `[SeatDefaults]` (et nulle part ailleurs sinon ça ne marchera pas) les commandes suivantes :

```
xserver-command=X -s 0 -dpms
```

```
sudo apt-get autoremove minecraft-pi
sudo apt-get autoremove wolfram-engine
```

```
sudo reboot
```

(Alt-F4 pour quitter Midori après le redémarrage)

```
sudo apt-get install alsa-utils mpd mpc minidlna
   usbmount ntfs-3g -y
```

```
sudo nano /etc/mpd.conf
```

Commentez la ligne suivante avec un # :

```
bind_to_address "localhost"
```

Pour un réseau, c'est la ligne après le #

```
sudo apt-get install nginx sqlite3 php5-fpm php5-
   curl php5-sqlite php5-cli php5-gd php5-imagick -y
sudo apt-get install samba samba-common-bin -y
sudo apt-get install autofs -y
sudo nano /etc/auto.master
```

Ajoutez la ligne suivante à la fin :

```
/media/ /etc/auto.ext-usb
   --timeout=10,defaults,user,exec,uid=1000
```

```
sudo mkdir /var/www
```

(existe déjà, vérifiez-le d'abord avec : `ls /var/www`)

```
sudo mkdir /mnt/disk1
sudo mkdir /mnt/disk2
sudo mkdir -p /mnt/NAS
sudo mkdir -p /mnt/USB
sudo ln -s /mnt/NAS /var/lib/mpd/music/NAS
sudo ln -s /mnt/USB /var/lib/mpd/music/USB
sudo rm -r /var/www
```

configuré Volumio et redémarré plusieurs fois, tout en s'habituant à manipuler le GUI du lecteur audio, en particulier pour mettre à jour la « bibliothèque ». À la fin, quelques chansons, à 16 bits aussi bien que 24, se voyaient dans la liste de lecture et le temps de lecture défilait. Malheureusement, bien que le module CN/A et l'ampli stéréo soient alimentés, il n'y avait rien sur l'entrée audio analogique.
On a conseillé à Werner de placer les commutateurs DIP ainsi :

Liens

[1] « CN/A audio pour RPi », Elektor 7-8/2017 : www.elektormagazine.fr/160198
[2] « Commande de volume pour le CN/A audio », Elektor 11-12/2017 : www.elektormagazine.fr/160321
[3] Page du labo d'Elektor : www.elektormagazine.frm/labs/audio-dac-for-rpi-networked-audio-player-using-volumio#/comments/labs/1206
[4] Pilote de LCD : http://www.waveshare.com/wiki/3.5inch_RPi_LCD_(A)

Ceci efface tout le contenu de /var/www. Pour éviter les erreurs :

```
fatal: destination path '/var/www' existe toujours et n'est pas un répertoire vide
sudo git clone https://github.com/volumio/Volumio-WebUI.git /var/www
sudo chmod 775 /var/www/_OS_SETTINGS/etc/rc.local
sudo chmod 755 /var/www/_OS_SETTINGS/etc/php5/mods-available/apc.ini
sudo chmod -R 777 /var/www/command/
sudo chmod -R 777 /var/www/db/
sudo chmod -R 777 /var/www/inc/
sudo cp -var /var/www/_OS_SETTINGS/etc /
cd /var/lib/mpd/music ; sudo ln -s /mnt/disk1/Music
sudo ln -s /mnt/NAS /var/lib/mpd/music/NAS
sudo ln -s /mnt/USB /var/lib/mpd/music/USB
sudo nano /etc/modules
```

Ajoutez les lignes suivantes au fichier /etc/modules :

```
snd_soc_bcm2708
bcm2708_dmaengine
snd_soc_hifiberry_dac
```

Configurez maintenant ALSA

```
sudo nano /etc/asound.conf
```

Créez /etc/asound.conf avec le contenu suivant :

```
 pcm.!default  {
 type hw card 0
}
ctl.!default {
 type hw card 0
}

sudo reboot
```

À ce stade le LCD n'est toujours pas utilisé, seulement la sortie HDMI.
Volumio apparaît et réclame sa mise à jour. La première mise à jour provoque une erreur. Fermez Midori (Alt-F4) et relancez-le. Entrez http:/localhost dans la barre d'adresse et réessayez. Ou clic droit sur la page d'erreur, sélectionnez retour et réessayez. La deuxième fois ne devrait pas provoquer d'erreur et l'écran principal de Volumio devrait apparaître.

Pour utiliser l'écran de 3,5 pouces de Waveshare, copiez le pilote depuis le CD (dans notre cas le fichier s'appelait LCD-show-161112.tar.gz) dans un répertoire du Raspberry Pi. Le plus simple est d'utiliser une clé USB. Suivez les instructions du manuel utilisateur disponible sur le CD. Il précise d'exécuter tar xvf LCD-show.tar.gz.
Selon où vous avez copié le pilote (dans notre cas dans /home/pi/Documents), exécutez :

```
 tar xvf  /home/pi/Documents/LCD-show-160520.tar.gz
```

puis

```
cd /home/pi/LCD-show/
sudo ./LCD35-show
```

Le système va redémarrer et Volumio devrait apparaître en plein écran sur le LCD. Si vous voulez encore utiliser le Raspberry Pi pour autre chose, la touche F11 permet de quitter le mode plein écran et le Bureau est accessible. Pressez à nouveau sur F11 pour remettre Midori en plein écran (si c'est la fenêtre active).
Ajouter un emplacement à la bibliothèque peut nécessiter un redémarrage pour être visible.
Comme l'installation du LCD écrase ou modifie /boot/config.txt, nous sauvegardons l'activation d'AudioDAC pour terminer.

```
sudo nano /boot/config.txt
```

Pour activer I2S dans /boot/config.txt, décommentez ou ajoutez la ligne :

```
dtparam=i2s=on
and add line:
dtoverlay=hifiberry-dac

sudo reboot
```

- **1 à 6** : position L (ON)
- **7** : position H (OFF)

et le système a fini par fonctionner.

Prise en charge de Moode Audio

Si Moode Audio prend en charge un CN/A HifiBerry ou un CN/A générique I²S, notre CN/A devrait aussi fonctionner. Mais sachez que le PCM1794A ne supporte pas le mode I²S à 16 bits. Si de l'audio à 16 bits est lu dans une trame à 32 bits, alors l'audio à 16 bits fonctionne. Sinon une conversion de la fréquence d'échantillonnage est nécessaire pour changer le format des données audio de 16 à 32 bits. Volumio peut le faire.

(160632 – version française : Denis Lafourcade)

HangTux n°2

pendu sur le Raspberry Pi

Linux reste un défi pour beaucoup, même si l'arrivée de nombreux mini-ordinateurs qui tournent avec ce système d'exploitation (Raspberry Pi, BeagleBone, DragonBoard, Odroid, etc.) a accru le nombre d'utilisateurs. Si vous ne vous êtes pas encore lancé dans l'aventure, voici une application simple et amusante pour débuter : un jeu du pendu.

Roy Aarts (labo d'Elektor) &
Thijs Beckers (rédaction d'Elektor)

Dans l'article *HangTux*, paru dans l'édition de juillet-août 2016 du magazine [1], nous avons utilisé la carte Linux d'Elektor pour créer un jeu du pendu. Le programme du jeu tourne sur la carte, mais un ordinateur sous Linux sert de terminal. Les résultats sont aussi comptabilisés sur la carte, et un petit module d'extension affiche le nombre d'erreurs sur une barre de huit LED (style VU-mètre numérique). Cet article-ci s'inspire du précédent ; Nous avons cependant remisé la carte Linux d'Elektor pour la remplacer par le nano-ordinateur Raspberry Pi. Le but est toujours le même : jouer au pendu sous Linux. Le RPi doit bien entendu être configuré en ordinateur avec clavier, souris et écran ainsi qu'une distribution Linux comme système d'exploitation.

Matériel

Nous n'avons pas modifié le circuit, dont le schéma est représenté en **figure 1**. Les LED sont commandées par un circuit d'extension d'E/S à 8 bits, le MCP23S08. Il communique avec le Raspberry Pi via le bus SPI (*Serial Peripheral Interface*). Les composants sont tous traversants, et le circuit intégré est en boîtier DIP 18 ; le câblage du circuit imprimé ne devrait donc pas poser de soucis, même pour un néophyte en soudage.
Le circuit est alimenté en 3,3 V par le RPi. A priori cela ne doit pas poser de problème, mais il faut tenir compte de la consommation en courant plus élevée. Si votre adaptateur secteur atteint ses limites, il vaudrait mieux le remplacer par un modèle plus costaud.
La communication avec le MCP23S08 via le bus SPI repose sur des mots de 24 bits. Le premier octet adresse le circuit et propose l'écriture ou la lecture ; comme nous ne faisons qu'envoyer des données vers le circuit d'extension, le premier octet sera toujours '01000000'. Les bits d'adresse A0 et A1 sont toujours au niveau bas, les broches 4 et 5 d'IC1 sont dès lors raccordées à la masse.
Le second octet contient l'adresse du registre dans lequel les données seront écrites. Nous n'en utilisons que deux : IODIR (*I/O Direction*), pour configurer

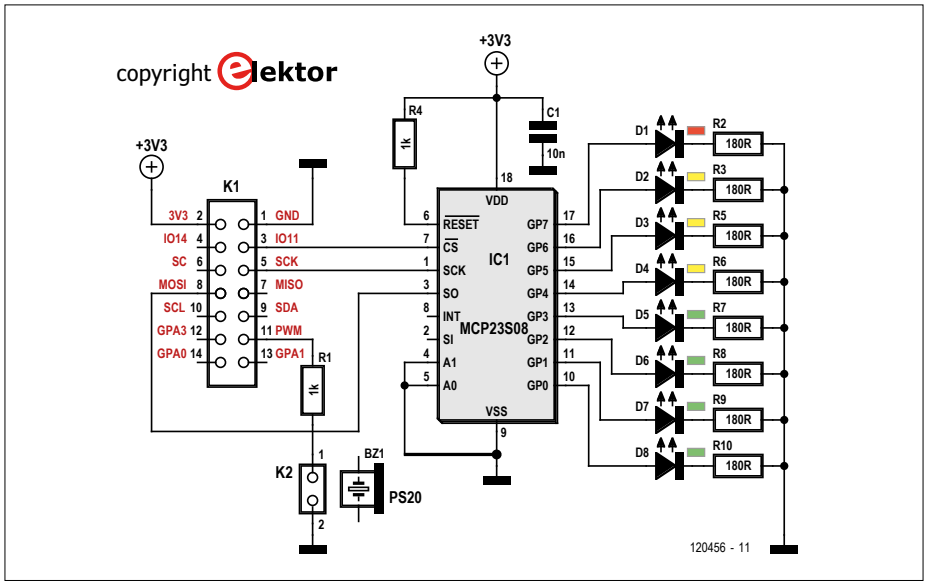


Figure 1. Le circuit est inchangé. Un circuit d'extension d'E/S commande les LED.

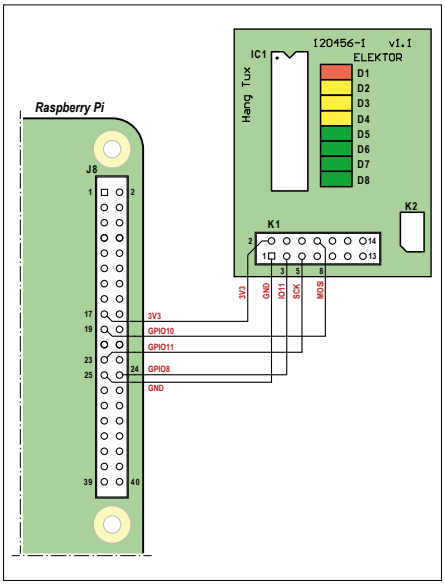


Figure 2. Raccordement du circuit imprimé au Raspberry Pi.

Figure 3. Démarrage du jeu dans le menu principal.

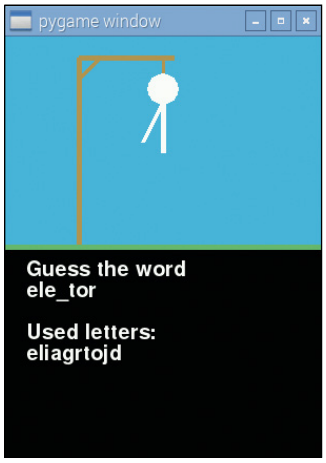


Figure 4. Pouvez-vous deviner de quel mot il s'agit ?

Figure 5. Bravo ! Vous avez gagné.

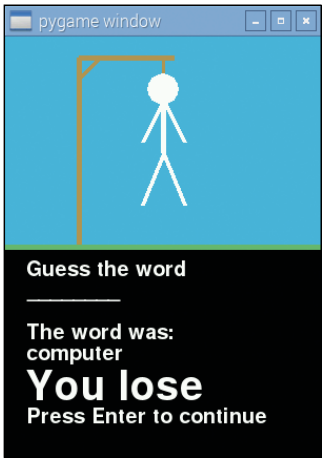


Figure 6. Pas de chance, essayez encore !

les broches GP0 à GP7 comme sorties, et OLAT (*Output Latch*) chargé de verrouiller le niveau logique désiré pour les broches de sortie. Le troisième octet contient les données à écrire dans un registre ou en sortie. Le signal CS (*Chip Select*) est commandé par le RPi.

Logiciel

Le programme est écrit en Python, il peut être téléchargé gratuitement sur le site du magazine [2]. La procédure d'installation est décrite ci-après.
Il faut tout d'abord configurer le bus SPI. Ouvrez une fenêtre de terminal sur le Raspberry Pi, et tapez l'instruction suivante :

```
sudo raspi-config
```

Allez ensuite à *advanced -> SPI*. Il vous est demandé si vous souhaitez lancer le SPI, répondez oui. Retournez à l'invite de commande et tapez les instructions suivantes :

```
sudo apt-get update
sudo apt-get install python-dev
   python3-dev
git clone git://github.com/doceme/
   py-spidev
cd py-spidev
sudo python setup.py install
sudo reboot
```

Copiez maintenant sur le RPi le dossier *hangman* (pendu) téléchargé en [2] ; placez-le dans */home/pi*. Pour cela utilisez une clé USB ou *WinSCP*. Enfin redémarrez (*reboot*) le RPi. Lorsque cette opération est terminée, copiez le fichier *hangman.sh*, provenant de *hangman*, sur le bureau. Ouvrez un terminal et tapez les instructions suivantes :

```
sudo cd Desktop
sudo chmod +x hangman.sh
```

Connectez le circuit d'extension au RPi suivant le schéma de la **figure 2**.
Vous voilà fin prêt pour lancer le jeu, en double-cliquant sur *hangman.sh* que vous avez copié sur le bureau.

Dictionnaires

Le jeu utilise des dictionnaires. Il y en a un en français et un en anglais dans le pack téléchargé, mais vous pouvez aussi y ajouter le vôtre. Il suffit de créer un fichier au format texte, d'y écrire les mots que vous voulez ajouter, un par ligne, et de le sauvegarder avec le nom *dictionary.txt* dans le dossier du jeu. Dans la première version d'HangTux, le nombre maximal de mots était de 100, ici il n'y a pas de limitation.
Le jeu utilise toujours le fichier dénommé *dictionary.txt*. Si vous voulez utiliser un autre dictionnaire, il suffit de le renommer.

À vous de jouer !

Il faut presser sur Entrée pour démarrer le jeu depuis le menu principal (**fig. 3**) ; cet écran permet aussi de quitter le jeu. L'écran montre quelles lettres ont été utilisées, lesquelles étaient correctes, combien il en reste à trouver, et le nombre d'essais restant (**fig. 4**).
La barre de LED indique aussi le nombre d'essais restant. Elle n'est pas vraiment nécessaire pour ce jeu, mais elle illustre parfaitement les possibilités de commande de matériel via le bus SPI. De plus en plus de composants électroniques ont besoin d'un système d'exploitation (variante de Linux), c'était donc un bon exercice, et certainement pas trop compliqué ! ⏮

(160127 – version française : Jean-Louis Mehren)

Liens

[1] www.elektormagazine.fr/120456
[2] www.elektormagazine.fr/160127

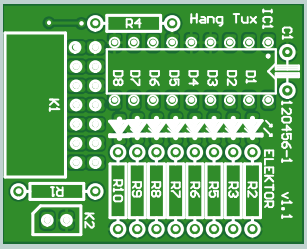
LISTE DES COMPOSANTS

Résistances
R1, R4 = 1 kΩ
R2, R3, R5 à R10 = 180 Ω

Condensateur
C1 = 10 nF

Semi-conducteurs
D1 = LED rectangulaire rouge, 5 mm
D2 à D4 = LED rectangulaire jaune, 5 mm
D5 à D8 = LED rectangulaire verte, 5 mm
IC1 = MCP23S08 (éventuellement avec support DIP 18)

Divers
K1 = embase mâle, à 2×7 contacts, pas de 2,54 mm
K2 = embase mâle, à 2 contacts, pas de 2,54 mm
Circuit imprimé réf. 120456-1



PROJET DU LABO

radio Internet avec Raspberry Pi

l'embarras du choix affranchi du PC

Roy Aarts (labo d'Elektor)

Avec ce projet, vous pouvez écouter n'importe quel émetteur radio en ligne sans gros frais en matériel.

Caractéristiques

- joue les flux audio des stations radio Internet
- sortie sur la prise audio intégrée
- fonctionnement autonome
- liste des stations radio programmable
- aucune soudure nécessaire

Les récepteurs radio Internet sont disponibles sous des formes et des tailles variées, mais peuvent s'avérer fort coûteux. Ce projet vous propose d'en réaliser un vous-même, pour un prix raisonnable, à l'aide d'un nano-ordinateur Raspberry Pi. Le RPi fournira sur sa prise « écouteurs » le signal audio décodé à partir d'un flux radio de l'internet. Pour cela, il vous faut un Raspberry Pi 2, un écran tactile de 2,8 pouces, une carte SD et, en option, une clé Wi-Fi. L'interface utilisateur écrite en python a recours à la bibliothèque *pygame*. Le logiciel est facile à comprendre et à adapter à votre goût.

Installation

Pour commencer, il faut installer *Raspbian* sur le Raspberry Pi. Téléchargez *Raspbian Jessie* depuis [1] et utilisez *Win32DiskImager* (téléchargement gratuit depuis [2]) pour graver l'image sur une carte SD vierge d'une capacité d'au moins 4 Go. Ensuite, insérez la carte SD dans la fente sur le Raspberry Pi et branchez un écran, un clavier, une souris ainsi qu'une clé Wi-Fi (ou directement un câble Ethernet). Enfin mettez sous tension. Lorsque le démarrage est terminé, activez le réseau en cliquant dans le coin supérieur droit (**fig. 1**). Quand la liaison réseau est obtenue, notez l'adresse IP du Raspberry Pi, vous en aurez besoin pour le configurer plus tard. Pour obtenir cette adresse, survolez l'icône qui a servi à activer le réseau.

Maintenant il faut installer les bibliothèques de l'écran tactile. Si vous n'avez pas encore branché cet écran, c'est le moment. Mais pas avant d'avoir arrêté le Raspberry Pi (coin supérieur gauche, cliquez sur `Shutdown`) ! Branchez l'écran et remettez sous tension. Nous allons installer les pilotes nécessaires à partir du PC. Téléchargez le programme *PuTTY* à partir de [3], installez et démarrez-le et entrez l'adresse IP de votre RPi dans le champ intitulé `Host address (or IP address)`. Si un message de diagnostic apparaît, cliquez sur `Yes`. Une fenêtre de terminal doit maintenant s'ouvrir avec une invite d'identification d'utilisateur. Identifiez-vous avec le nom d'utilisateur « pi » et le mot de passe « raspberry », puis entrez la commande :

```
sudo wget -N https://github.com/watterott/
   RPi-Display/raw/master/rpi-display.sh
```

terminée par « Entrée ». Les fichiers nécessaires sont alors téléchargés depuis github. Quand l'invite réapparaît, tapez :

```
sudo /bin/bash rpi-display.sh 270
```

Les fichiers github sont installés sur le système.
Après quelque temps, une liste de questions vient s'afficher. Répondez de la manière suivante :

- Activate the console on the TFT display? y/n **Y**
- Install fbcp (Framebuffer Copy)? y/n **N**
- Install xinput-calibrator? y/n **Y**
- Install tslib (touchscreen library)? y/n **Y**
- Calibrate touchscreen now? y/n **N**
- Reboot the system now? y/n **Y**

Le Raspberry Pi va redémarrer et la connexion avec le PC va être perdue. Si tout va bien, vous devriez voir l'interface utilisateur apparaître sur l'écran tactile après le redémarrage.

Configuration & Boîte à Musique

Pour configurer l'écran tactile, reconnectez-vous sur le RPi avec PuTTY. Identifiez-vous et saisissez la commande :

```
sudo TSLIB_FBDEVICE=/dev/fb1 TSLIB_TSDEVICE=/dev/
   input/touchscreen ts_calibrate
```

Vous allez voir apparaître l'interface de calibrage sur l'écran tactile (**fig. 2**). Une fois accomplies toutes les étapes affichées, l'écran est calibré.

Nous pouvons maintenant installer le client Boîte à Musique (`mpc`) et son « démon ». Pour cela, tapez les commandes suivantes :

```
sudo apt-get update
sudo apt-get install mpd mpc
```

Interface utilisateur

Maintenant il faut installer l'interface utilisateur. Pour cela nous utiliserons *winSCP* (en téléchargement gratuit depuis [4]) et le logiciel disponible en téléchargement sur le site Elektor pour ce projet [5]. Nous devons copier les fichiers de ce logiciel sur la carte SD du RPi. Nous abandonnons PuTTY et démarrons winSCP. L'identification de l'utilisateur s'effectue avec les mêmes paramètres, en répondant aussi `yes` à un éventuel message. Nous copions le répertoire « pi-radio » du logiciel Elektor sur le RPi (par ex. sous la racine « /home »). Puis nous copions les fichiers *launcher.sh* et *shutDown.sh* vers le répertoire *Desktop* du RPi. Le premier fichier démarre la radio, le second arrête le RPi. Nous pouvons maintenant fermer winSCP et redémarrer PuTTY.

Les deux fichiers que nous venons juste de copier vers le répertoire *Desktop* du RPi ne sont pas encore associés à des icônes cliquables. Pour cela, il faut entrer les commandes :

```
sudo chmod +x /home/pi/Desktop/launcher.sh
```

et

```
sudo chmod +x /home/pi/Desktop/shutDown.sh
```

Stations de radio

La radio est maintenant installée, mais aucun émetteur n'est encore audible. Pour ajouter un émetteur, nous devons disposer de son lien, c'est-à-dire l'adresse internet sur laquelle il émet son flux audio. Le site SomaFM [6] contient une liste de tels liens. Ce site référence un grand nombre de stations radio de différents genres, avec des liens utilisables pour notre récepteur. Pour obtenir un lien utilisable de SomaFM, il faut cliquer sur la station désirée puis sur l'onglet *Direct stream links*. Le lien qu'il nous faut suit l'indication : *Direct server:*.
Pour ajouter une station, on entre sous PuTTY la commande `mpc add` suivie du lien de la station, par exemple :

```
mpc add http://ice1.somafm.com/bagel-128-mp3
```

Les liens se terminant par `.m3u` ne fonctionnent pas. Rien n'oblige à se limiter à SomaFM, on peut ajouter de nombreuses stations proposées par d'autres sites. On peut tester un lien en le saisissant sur le navigateur de son PC. S'il est valide, un lecteur audio va démarrer et faire entendre la station.
Pour lister les stations répertoriées, vous pouvez utiliser la commande `mpc playlist` qui en fournit la liste complète. Pour supprimer une station, tapez `mpc del` suivi du numéro de la station sur la liste affichée par PuTTY ; par exemple : `mpc del 2` supprime la station en deuxième position.

Mode d'emploi

On peut maintenant démarrer le lecteur par un double clic de son icône sur l'écran tactile et en appuyant sur `Execute`. L'interface (**fig. 3**) a été gardée simple pour qu'on puisse l'utiliser avec les doigts plutôt qu'avec un stylet (fig. 2). Les commandes suivantes sont disponibles :

- `Play` démarre le lecteur sur la liste actuelle des stations.
- `Pause` arrête le lecteur.
- `Volume down` baisse le son.
- `Volume up` augmente le son.
- `Mute` coupe le son.
- `Refresh` redémarre le lecteur en rafraîchissant l'écran.
- `Exit` ferme l'interface utilisateur, le lecteur continuant à jouer.

Pour arrêter le RPi, double-cliquez sur l'icône `Shutdown` et tapez sur `Execute`. Bonne écoute ! ⏮

(160043 – version française : Helmut Müller)

Liens

[1] www.raspberrypi.org/downloads/raspbian
[2] https://sourceforge.net/projects/win32diskimager
[3] www.chiark.greenend.org.uk/~sgtatham/putty/download.html
[4] https://winscp.net/eng/download.php
[5] www.elektormagazine.fr/160043
[6] http://somafm.com

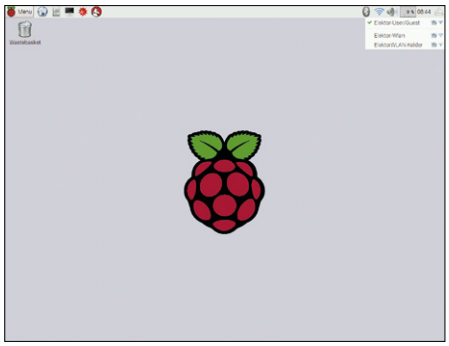
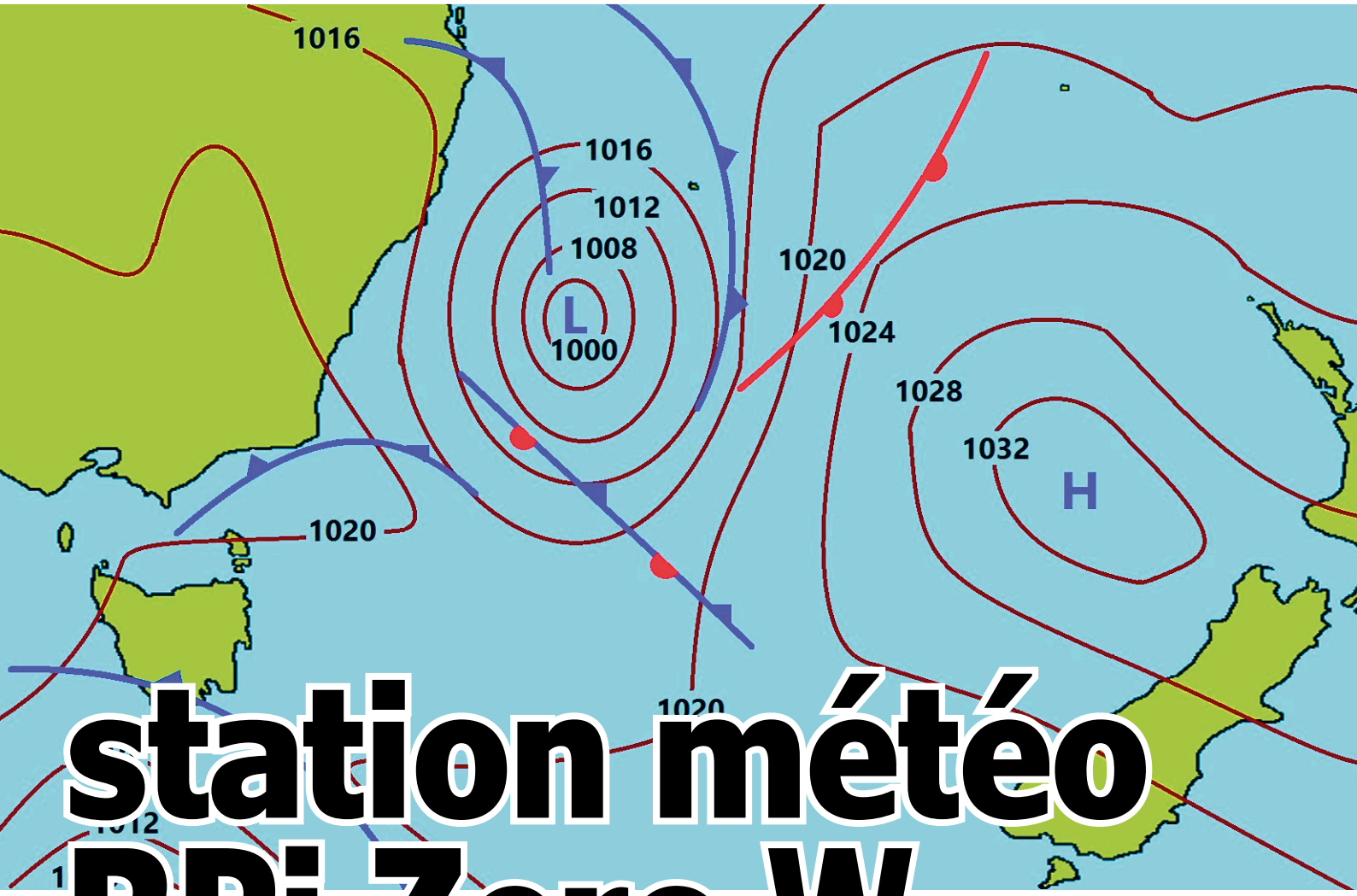


Figure 1. Configuration des paramètres réseau du RPi.

Figure 2. Calibrage de l'écran tactile avant utilisation...

Figure 3. L'interface répond aux doigts — aucun stylet nécessaire.

PROJET DE LECTEUR



station météo RPi Zero W

sans pièces mécaniques mobiles

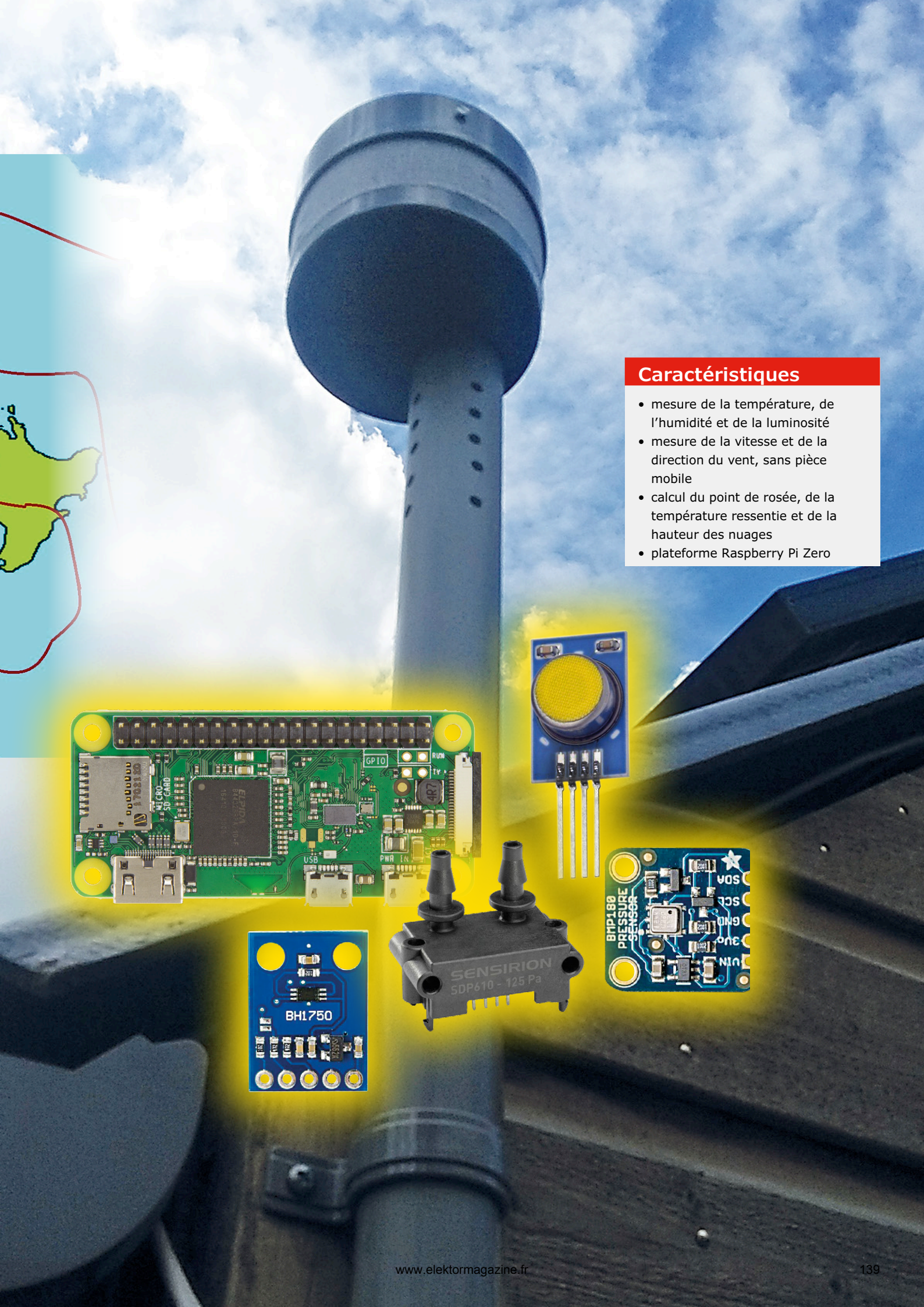
Zeno Otten (Pays-Bas)

De nombreux aspects de notre vie quotidienne sont régis par le temps qu'il fait : prendre son parapluie ou pas, aller à pied ou en voiture, ouvrir la fenêtre ou la fermer, faire la grasse matinée ou se lever, etc. Un simple coup d'œil par la fenêtre suffit pour avoir un aperçu des conditions actuelles, mais seule la mesure permet de faire des prévisions, et notre station météo est parfaite pour ça.

La réalisation d'une station météo n'est pas très rentable de nos jours, du moins si on ne considère que les aspects financiers. On trouve dans le commerce des appareils *made in Asia* bon marché, mais qui n'en sont pas moins performants, et un électronicien amateur ne peut pas les concurrencer. Ces appareils ont cependant quelques inconvénients, ce qui justifie l'approche *DIY* (*do it yourself*), en dehors de la satisfaction que l'on éprouve à fabriquer quelque chose soi-même. Pour mesurer la vitesse du vent, on utilise généralement une espèce de rotor avec des coupelles, et pour détecter sa

Caractéristiques

- mesure de la température, de l'humidité et de la luminosité
- mesure de la vitesse et de la direction du vent, sans pièce mobile
- calcul du point de rosée, de la température ressentie et de la hauteur des nuages
- plateforme Raspberry Pi Zero



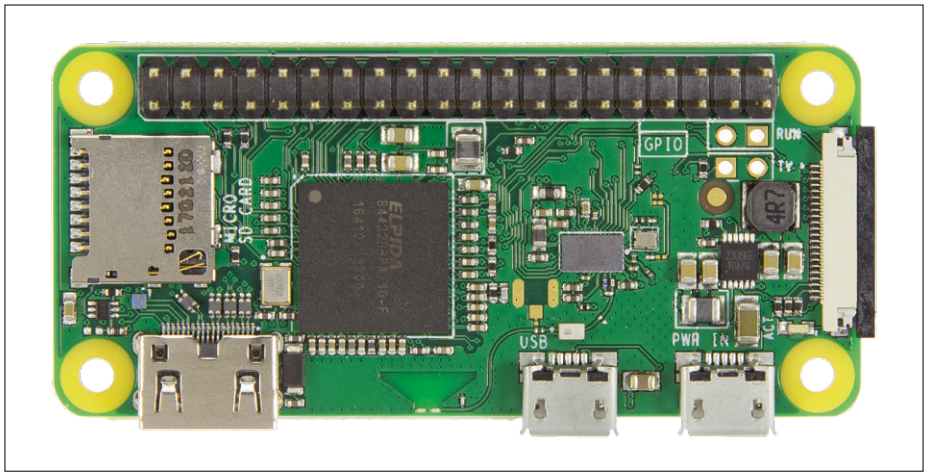


Figure 1. Le Raspberry Pi Zero W.

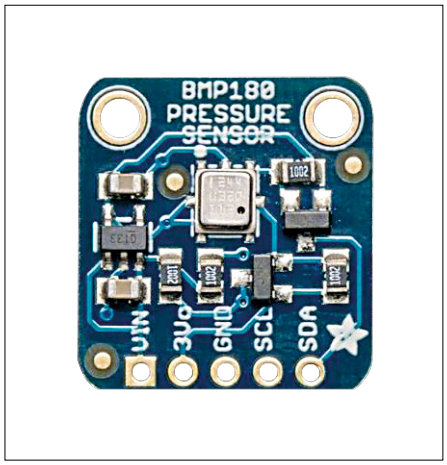


Figure 2. Le capteur de pression et de température BMP180.

Figure 3. Le capteur de température et d'humidité HYT221.

direction un simple fanion. Ce sont des pièces mécaniques, qui vont s'user et qu'il faudra remplacer tôt ou tard (en fonction de leur qualité). Ces pièces mécaniques ne sont pas toujours proposées sous forme de pièces détachées, et les problèmes commencent. La plupart du temps, la communication entre les capteurs et la station de base est effectuée à l'aide d'émetteurs-récepteurs qui travaillent par ex. dans la bande 433 MHz, et avec un protocole « maison ». Il est dès lors très difficile de modifier ou d'étendre l'installation.

Enfin, certaines grandeurs dérivées, comme le point de rosée ou la température ressentie (ou indice de refroidissement éolien), ne sont pas disponibles.

Le défi

Les raisons exposées ci-dessus étaient suffisantes pour relever le défi et concevoir sa propre station météo. Le cahier des charges est le suivant :

- pas de pièces mobiles ;
- mesure de la pression atmosphérique, de l'humidité, de la température, de la luminosité ambiante, de la vitesse et de la direction du vent ;
- calcul du point de rosée, de la température ressentie et de la hauteur des nuages ;
- utilisation du protocole I²C pour les capteurs ;
- tension d'alimentation de 3,3 V ;
- stockage des résultats des mesures dans une base de données ;
- affichage des résultats via une interface réseau ;
- communication Wi-Fi par réseau local ou internet ;
- possibilité d'extension (par ex. pour un pluviomètre) ;
- logiciel compréhensible et évolutif.

Le projet

Au vu de notre cahier des charges, le choix du cœur de notre circuit a été vite fait. Le Raspberry Pi Zero W est parfait pour la tâche : ce microcontrôleur a suffisamment de broches d'E/S, une bonne puissance de calcul, et tout ce qu'il faut pour réaliser facilement une liaison sans fil avec un réseau local ou internet. Si on ajoute que sa taille est celle d'une carte de crédit (**figure 1**) et qu'il ne coûte que 10 €, pas besoin de chercher plus loin. Elektor nous a présenté la « bête » dans le numéro de septembre/octobre [1], et un kit de démarrage complet [12] est disponible dans l'e-choppe pour ceux qui n'ont pas encore d'expérience du RPi (Zero).

Les capteurs communiquent avec le RPi Zero via le bus I²C ; deux fils sont aussi nécessaires pour leur alimentation. Le bus I²C, développé initialement par Philips, est très populaire ; il permet la communication entre microcontrôleurs, systèmes complets ou divers circuits intégrés via deux fils : *SDA* (*Serial Data*) pour les données, et *SCL* (*Serial Clock*) pour l'horloge. Avec l'alimentation, nous aurons donc des liaisons à quatre fils avec les capteurs.

Le protocole de communication est un système maître-esclave : le maître, dans notre cas le RPi, s'adresse à un esclave (un des capteurs) et lui demande la transmission des données disponibles (par ex. la température). Chaque esclave dispose d'une adresse unique, que l'utilisateur peut en général – et au moins partiellement – choisir lui-même.

Le bus utilise des résistances de rappel dont la valeur est de 4,7 à 10 kΩ ; les lignes *SDA* et *SCL* sont à l'état haut au repos. Ces résistances sont parfois intégrées au capteur, il faut donc veiller à ce que la valeur totale reste dans les limites acceptables.

Les capteurs

La mesure de la pression atmosphérique, de la température, de l'humidité et de la luminosité ne pose pas de problème particulier ; c'est un peu plus compliqué pour la vitesse et la direction du vent, mais chaque chose en son temps…

Pression atmosphérique

Pour la pression atmosphérique, nous uti-

Figure 4. Le capteur de luminosité ambiante BH1750.

lisons un BMP180 [2], un capteur barométrique qui mesure la pression absolue entre 300 et 1 100 hPa (mbar). Comme la pression atmosphérique est inversement proportionnelle à l'altitude – ceux qui ont déjà escaladé l'Everest le savent bien, le capteur peut aussi être utilisé comme altimètre.
Le capteur est monté sur une petite carte de liaison (*Breakout Board – BoB*), **figure 2**. La résolution est de 12 bits, soit 0,03 hPa. La tension d'alimentation est de 3,3 V et les résistances de rappel du bus I²C sont déjà montées sur le *BoB*. La communication avec le RPi se fait via les lignes *SDA* et *SCL*.
Il est possible que le BMP180 soit difficile à trouver. On peut utiliser le BME280, qui est disponible dans l'e-choppe [3]. Une adaptation du microgiciel sera peut-être nécessaire.

Température et humidité

Nous avons choisi un capteur robuste, le HYT221 [4], **figure 3** ; il est idéal pour un usage à l'extérieur. Il mesure l'humidité relative de 0 à 100% avec une précision de ±1,8%, et la température de −40 à +120 °C avec une précision de ±0,2 °C.
Les résistances de rappel pour le bus I²C sont intégrées au capteur, et la tension d'alimentation couvre la plage de 2,7 à 5,5 V.

Tableau 1 : vitesse du vent

vent violent à tempête	> 17 m/s
fortes bourrasques	14 à 17 m/s
forte brise	10 à 14 m/s
brise fraîche	8 à 10 m/s
brise légère à modérée	1,5 à 8 m/s
très légère brise	0,3 à 1,5 m/s
calme	0 à 0,2 m/s

Luminosité ambiante

La mesure de la luminosité ambiante est utilisée pour détecter la nuit, la couverture nuageuse, ou l'ensoleillement ; ces caractéristiques sont converties en valeurs concrètes. Le capteur est un BH1750 [5], dont la plage de mesure s'étend de 1 à 65 535 lx. Le capteur est monté sur un *BoB* (**figure 4**), de même que les résistances de rappel pour le bus I²C. La tension d'alimentation varie de 3,3 à 5 V.
Valeurs indicatives : 0,001 à 0,02 lx pour la nuit, 500 à 25 000 lx pour un ciel couvert, et 50 000 à 100 000 lx pour une journée ensoleillée.

Direction et vitesse du vent

Nous expliquons brièvement dans un cadre séparé comment ces deux valeurs sont obtenues à l'aide de deux capteurs de pression différentielle.
La plage de mesure est l'élément essentiel pour le choix du capteur. Le **tableau 1** reprend les vitesses de vent courantes dans nos régions. Si nous considérons une vitesse de 20 m/s pour une tempête moyenne et une masse

Intermède : « Éole, ce fils de Poséidon… »

La plupart des anémomètres pour la maison ou le jardin utilisent une espèce de rotor avec des coupelles (par ex. des balles de tennis de table coupées en deux) pour mesurer la vitesse, et un fanion ou une girouette pour la direction ; bref, des pièces mobiles. À notre époque, où microcontrôleur et logiciel règnent en maître absolu, l'électronicien amateur a horreur de la mécanique ; il nous fallait donc trouver une solution à ce problème.

Cette solution saute aux yeux quand on réfléchit à ce qui maintient un avion en l'air. L'aile d'un avion est plus courbée sur le dessus que sur le dessous. Dans un même laps de temps, l'air sur le dessus de l'aile doit parcourir un trajet plus long, et il en résulte une différence de pression (elle est plus faible sur le dessus de l'aile). Cette différence de pression induit une force vers le haut, qui maintient l'avion en l'air. Ce principe a été formulé au 18e siècle par Daniel Bernoulli.
Le tube de Pitot [11], utilisé en aéronautique pour mesurer la vitesse de l'air, repose sur un principe identique. La pression d'un fluide en mouvement, l'air par exemple, est mesurée face au flux et perpendiculairement à l'écoulement du fluide. La différence de pression permet de calculer la vitesse d'écoulement du fluide, en m/s, à l'aide de la formule établie par Bernoulli :

$$v = \sqrt{\frac{2\Delta p}{\rho}}$$

Δp est la différence de pression, et ρ la masse volumique du fluide (1,3 kg/m³ pour l'air).
Pour un anémomètre statique, c'est-à-dire sans pièces mobiles, nous utilisons un cylindre vertical autour duquel l'air s'écoule, comme représenté en **figure 5**.

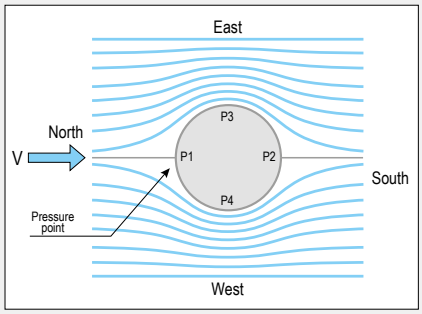


Figure 5. Écoulement de l'air autour d'un cylindre.

Deux capteurs de pression différentielle sont montés à l'intérieur du cylindre. Nous avons donc quatre points de mesure, P1 à P4, qui permettent la mesure de la différence de pression entre P1 et P2 d'une part, P3 et P4 d'autre part.
Pour obtenir la direction du vent, on compte pendant un intervalle de temps constant combien de fois la pression maximale apparaît à chaque point.
Le calcul de la moyenne permet de déterminer la direction.

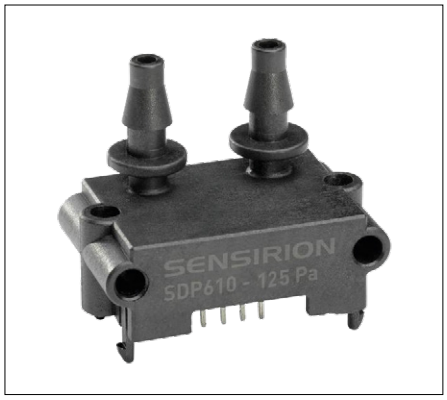


Figure 6. Le capteur de pression différentielle SDP600/610-125Pa.

volumique de l'air (ρ) de 1,3 kg/m³, la formule de l'encadré nous donne une différence de pression de 260 Pa.
En théorie, le capteur devrait donc au moins pouvoir mesurer cette différence de pression. Comme les tempêtes ne sont heureusement pas trop fréquentes sur la majeure partie de notre territoire, nous avons choisi le SDP600/610-125Pa de Sensirion [6], qui peut mesurer une différence maximale de 125 Pa. La précision de ce capteur est cependant meilleure.
La tension d'alimentation est de 3,3 V, et les données sont transmises via le bus I²C, mais il y a un hic : le fabricant a fixé l'adresse du capteur à *0x40* (valeur hexadécimale). Or nous avons besoin de deux capteurs ; nous devrions alors utiliser deux bus I²C, ce qui n'est guère pratique. Heureusement pour nous, l'adresse se trouve dans une mémoire EEPROM interne, et le fabricant nous a aimablement expliqué comment la modifier (ce qui annule la garantie…). Nous avons fixé l'adresse d'un des deux capteurs à *0x21*. Vous pouvez vous aussi demander au fabricant la procédure à suivre. Il suffit d'exécuter une routine avec un seul des deux capteurs connecté ; celle-ci est contenue dans le programme *weerstation.py* qui est disponible sur la page du projet [7].
Ce capteur ne possède pas les résistances de rappel pour le bus I²C, il faut les ajouter.

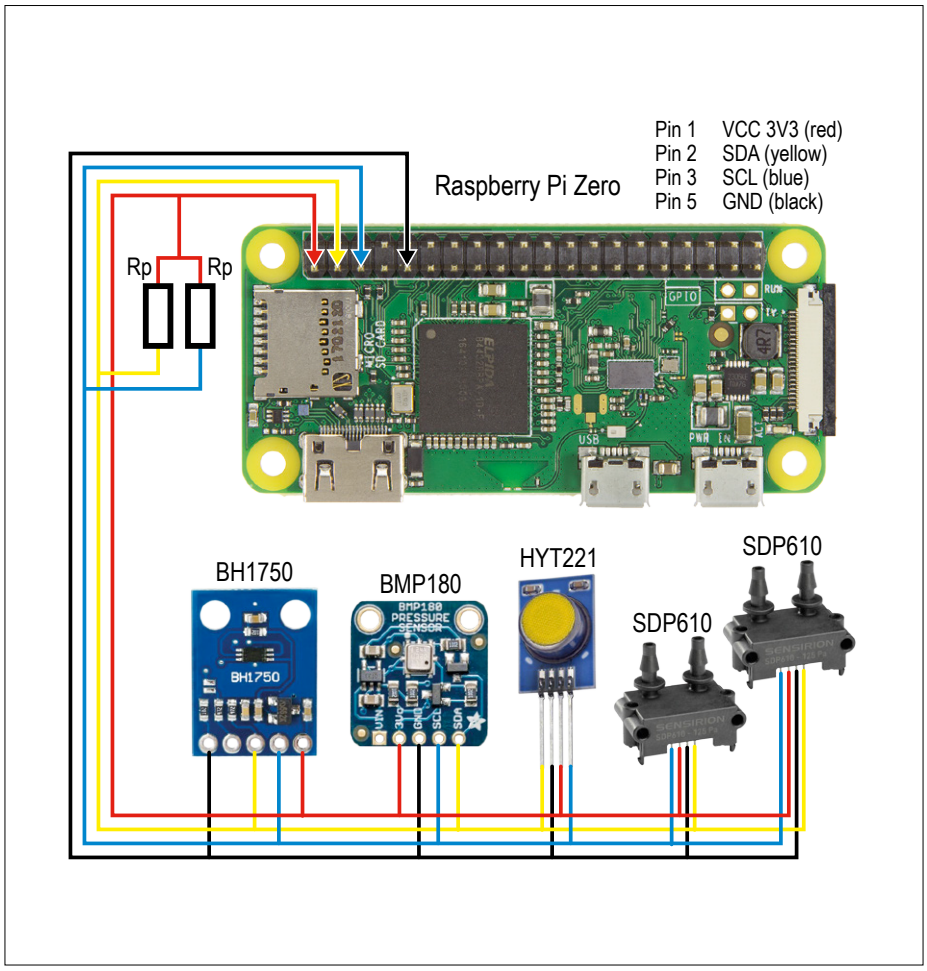


Figure 7. Connexion des capteurs au RPi Zero W via un bus à quatre fils.

Tableau 2 : composants I²C et leur adresse		
composant	**adresse (hexa)**	**fonction**
BMP180	0x77	capteur de pression
HYT221	0x28	capteur de température et d'humidité
SDP01	0x40	capteur de pression différentielle 1
SDP02	0x21	capteur de pression différentielle 2
BH1750	0x23	capteur d'éclairement lumineux

Les grandeurs dérivées

Jusqu'à présent, notre station météo n'est en rien différente de ce que l'on peut trouver dans le commerce ; nous ne mesurons que les grandeurs habituelles : température, humidité, pression atmosphérique, luminosité, vitesse et direction du vent. Ce qui rend notre projet intéressant, c'est le calcul de quelques valeurs dérivées moins connues.

Point de rosée

Le point de rosée est la température à laquelle l'air est saturé d'humidité. C'est un paramètre appréciable en météorologie ; il permet d'estimer la couverture nuageuse, le risque de brouillard, et le risque de formation de glace, ce qui est important pour le trafic aérien. Une valeur élevée du point de rosée indique également un plus grand risque d'orage. Le point de rosée peut être calculé à partir des valeurs de l'humidité relative et de la température de l'air, comme expliqué sur Wikipédia [8].

Température ressentie ou indice de refroidissement éolien (ou facteur vent)

Il y a température et température : il y a la valeur mesurée avec un thermomètre, et puis il y a celle que nous ressentons. La différence entre ces deux températures peut être significative ; avec un vent fort et une humidité relative élevée, une température de −3 °C peut être ressentie comme −10 °C, voire encore moins !
L'indice de refroidissement éolien, aussi appelé facteur vent, est calculé à partir des valeurs de l'humidité relative et

de la vitesse du vent. Il existe plusieurs modèles pour ce calcul, nous avons choisi celui du service météo australien [9].

Hauteur des nuages

La hauteur des nuages est aussi une donnée importante : plus ils sont bas, plus le risque de précipitations est élevé ; les agriculteurs l'ont d'ailleurs toujours su… Météo France explique comment calculer cette hauteur à partir des valeurs de la température de l'air et du point de rosée [10].

L'électronique

Si nous considérons le RPi et les capteurs comme des boîtes noires, il n'y a pas grand-chose à dire sur l'électronique. La **figure 7** reprend schématiquement l'ensemble du montage : un RPi Zero W, les capteurs, et un bus à quatre fils (deux pour l'I²C et deux pour l'alimentation). Nous n'avons pas repris l'alimentation du RPi sur ce schéma, on peut utiliser un chargeur/adaptateur USB relié au connecteur d'alimentation de la carte (en bas à droite sur la photo de la figure 1). Les deux résistances R_p sont les résistances de rappel vers le haut des capteurs de pression différentielle. La **figure 8** montre le prototype de l'auteur, installé dans un petit boîtier en plastique. Le câble d'alimentation passe par un serre-câble à étrier. Il en va de même pour le câble UTP à quatre fils de l'anémomètre, qui sur ce prototype a une longueur de 5 m ; cela ne pose pas de problème pour la transmission des données, et permet l'installation des capteurs sur le toit.
Le **tableau 2** reprend les adresses I²C des différents capteurs.

Le montage

Pour le montage de l'anémomètre, il faut faire preuve d'un peu d'ingéniosité. L'auteur a utilisé un tube en PVC de 10 cm de diamètre, dans lequel il a percé quatre trous, aux quatre points cardinaux. Des morceaux de tuyau flexible, par ex. celui utilisé pour une pompe d'aquarium, sont collés au tube en PVC et connectés aux ouvertures des capteurs de pression différentielle. La **figure 9** montre le résultat final. Les dessus et dessous du tube doivent bien entendu être étanches, afin de protéger l'électronique ; les extrémités des morceaux de tuyau doivent aussi être protégées des insectes, qui pourraient s'y installer, par exemple avec de la gaze. En tout cas, les bricoleurs peuvent s'en donner à cœur joie.

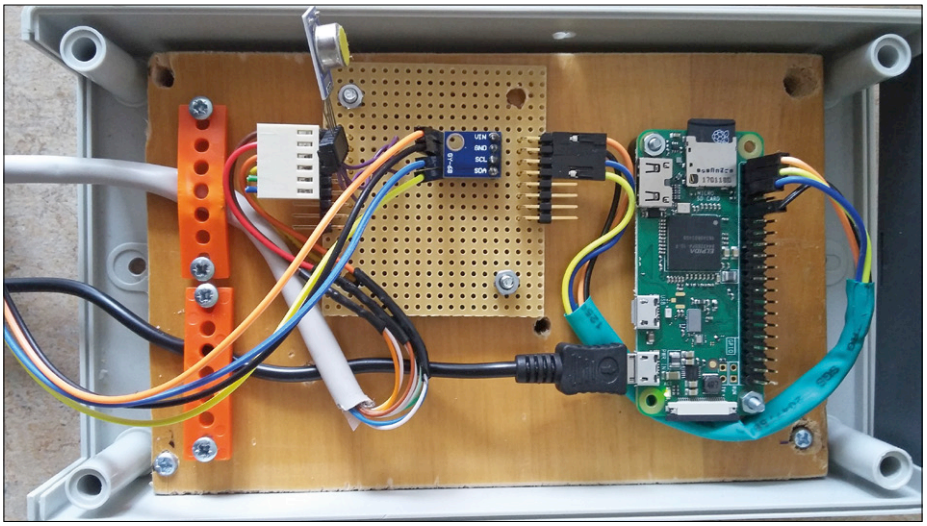


Figure 8. Le prototype de l'auteur.

Figure 9. Le capteur éolien dans son élément, au-dessus du toit.

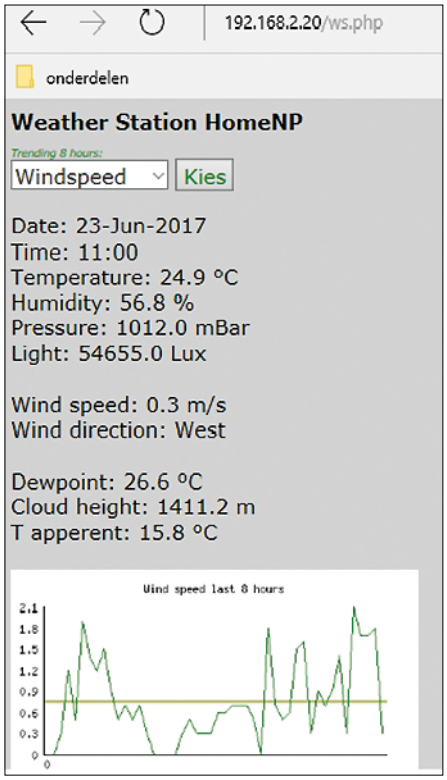


Figure 10. Page web avec les résultats des mesures.

Le micrologiciel

Le micrologiciel de la station météo comporte deux parties. Tout d'abord le programme *weerstation.py*. Celui-ci est écrit en Python, ce qui est pratique, puisque *Python 2.6* est partie intégrante de la distribution Linux *Jessie* du RPi Zero. Le programme collecte les données des capteurs toutes les dix minutes, exécute les calculs nécessaires, et entrepose les résultats dans une base de données MySQL.

La deuxième partie du micrologiciel est un script combiné HTML et PHP. Celui-ci tourne sur le serveur Apache du RPi Zero, lit les données de la base de données et les présente sur une page web (exemple en **figure 10**).

Cela n'aurait aucun sens de reproduire le code du micrologiciel ici, vous pouvez le télécharger en quelques clics de souris. Le programme est convenablement structuré et se passe de commentaires.

Nous ne citerons que les bibliothèques utilisées par *weerstation.py* :

```
import MySQLdb as mdb
import sys
import time
import os
import smbus
from ctypes import c_short
import math
```

La page web *ws.php* (voir dossier de téléchargement) est publiée par le serveur Apache installé sur le RPi Zero. Chez l'auteur, le RPi Zero est relié au réseau local de la maison ; ainsi les données de la station météo sont consultables sur tous les appareils connectés au réseau. La page peut aussi être publiée sur internet via le routeur.

Conclusion

Ce projet ne sera pas proposé sous forme d'un kit prêt à l'emploi, c'est plutôt une source d'inspiration pour les lecteurs qui souhaitent réaliser un projet semblable. Nous serions heureux de recevoir vos remarques et suggestions. N'hésitez pas à nous présenter vos réalisations de station météo.

(160566 – version française : Jean-Louis Mehren)

Liens

[1] www.elektormagazine.fr/160451
[2] learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-
[3] www.elektor.fr/bme280-mouser-intel-i2c-version-160109-91
[4] www.farnell.com/datasheets/1643979.pdf
[5] domoticx.com/esp8266-wifi-lichtintensiteit-sensor-bh1750-gy-302-nodemcu
[6] www.sensirion.com/en/environmental-sensors/humidity-sensors/digital-differential-pressure-sensors-without-zero-point-drift
[7] www.elektormagazine.fr/160566
[8] fr.wikipedia.org/wiki/Point_de_rosée
[9] www.bom.gov.au/info/thermal_stress/?cid=003bl08
[10] www.meteofrance.fr
[11] fr.wikipedia.org/wiki/Tube_de_Pitot
[12] www.elektor.fr/raspberry-pi-zero-w-starter-kit

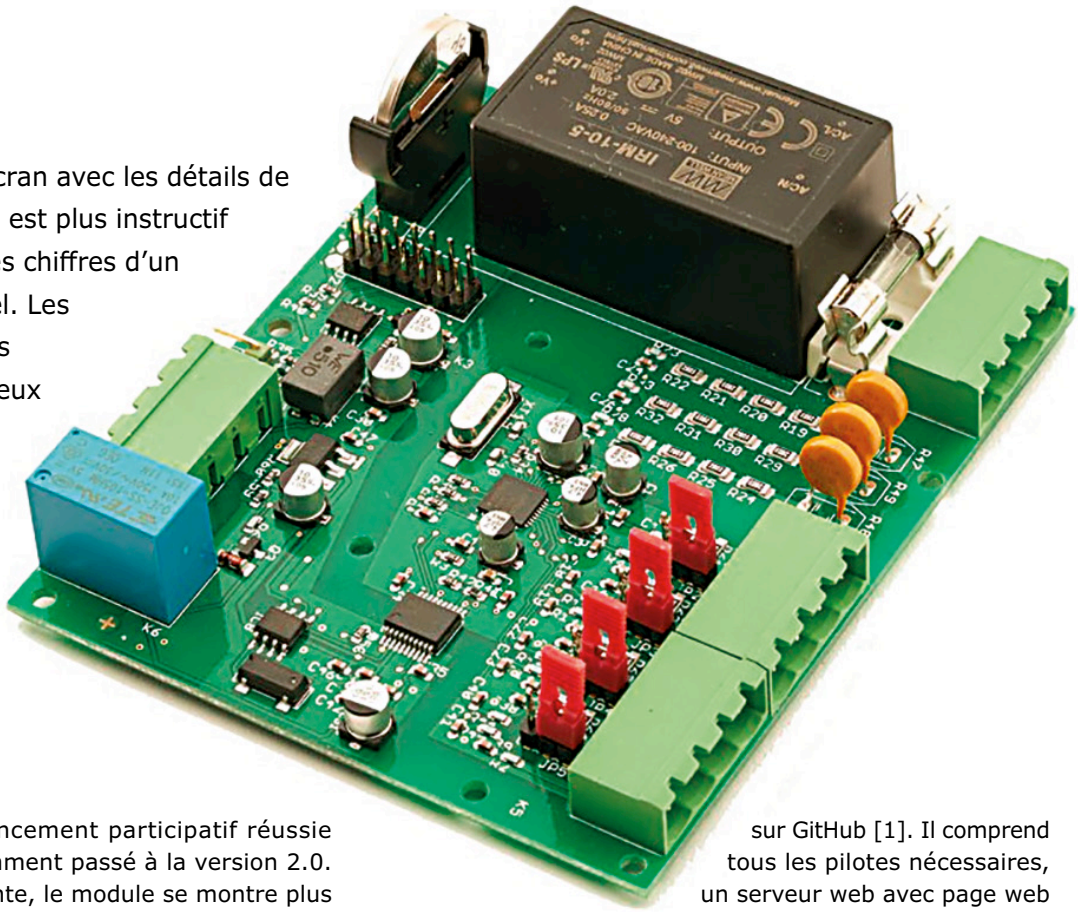
banc d'essai

on compte sur SmartPi

le compteur d'énergie intelligent pour Raspberry Pi

Clemens Valens (labo d'Elektor)

Pour l'usager lambda, un écran avec les détails de sa consommation d'énergie est plus instructif que le morne défilement des chiffres d'un wattheuremètre traditionnel. Les fournisseurs d'énergie et les opérateurs de réseaux ont eux aussi besoin de connaître en détail nos « habitudes de consommation ». Le compteur électrique « bidirectionnel » SmartPi se révèle à cet égard intéressant et bon marché.



Après une campagne de financement participatif réussie en 2016, le SmartPi est récemment passé à la version 2.0. Comparé à la version précédente, le module se montre plus pratique grâce à une alimentation intégrée et au placement sur un même côté des connecteurs les plus utiles. La carte gagne également en flexibilité puisque tous les connecteurs disposent maintenant de borniers à vis et qu'une interface RS-485 permet la connexion de dispositifs Modbus. Un port pour relais a été ajouté, et l'horloge en temps réel (RTC) a une pile de secours. Enfin, le nouveau boîtier possède un clip compatible rail DIN.

Spécifications

Le SmartPi mesure des intensités jusqu'à 100 A sur quatre canaux (trois phases + neutre), des tensions (jusqu'à 390 V_{eff}) sur trois canaux ainsi que des fréquences. Il calcule différentes puissances (active, réactive et apparente) et la consommation. Et puisqu'il mesure aussi la production d'énergie, le fabricant l'a à juste titre nommé « compteur intelligent bidirectionnel ».
Le SmartPi peut être utilisé avec des capteurs de courant inductifs à sortie secondaire de 50 mA ou 1 A, ce dernier type de sonde permettant, après modification d'un cavalier, la mesure d'intensités jusqu'à 700 A.
La carte d'extension SmartPi peut être achetée séparément, sous forme de kit configurable, ou entièrement assemblée. La version assemblée (celle de ce banc d'essai) est construite autour d'un Raspberry Pi 3 modèle B.
Le logiciel est écrit en Go, son code source ouvert est disponible sur GitHub [1]. Il comprend tous les pilotes nécessaires, un serveur web avec page web ainsi qu'une interface REST permettant l'intégration du SmartPi dans des applications maison.

Plus succinct, c'est un télégramme

Comme trop souvent de nos jours, le manuel livré avec le multimètre SmartPi est pour le moins succinct. Une visite pleine d'espoir sur le site du fabricant s'avère tout aussi décevante (à propos de ce site, notez que s'il est en allemand les pages consacrées au SmartPi sont disponibles en anglais). Entrez l'URL [2] pour vous y rendre, ou cliquez sur le lien vidéo (!) de l'e-choppe. La page Kickstarter [3] est sans doute la meilleure des sources d'information, sans oublier le forum [4].
La page *Installation* du site n'explique pas l'installation matérielle du module, mais celle du logiciel sur un Raspberry Pi. Quant au raccordement du module au réseau électrique d'une maison, il a carrément été laissé à titre de travaux pratiques. La page *Use* montre bien un schéma de connexion, mais ici non plus rien n'est dit sur le câblage du module au secteur. Il est même suggéré de confier l'installation à un électricien professionnel (**fig. 1**). C'est évidemment de bon conseil, mais nous aurions aimé davantage de détails ici.

Autre frustration, la page n'a pas été mise à jour et se réfère toujours au SmartPi 1.0 et à son alimentation externe.

Figure 1. L'installation du SmartPi par un technicien qualifié (je le suis. Vraiment).

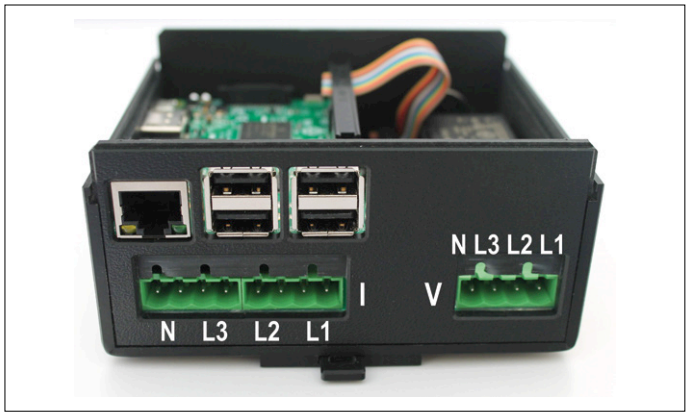


Figure 2. Le brochage des connecteurs de tension et de courant.

Connexion du SmartPi à un réseau de données

Le SmartPi arbore tous les connecteurs du RPi, dont le connecteur Ethernet. Une fois le SmartPi relié à mon routeur, il est apparu dans la liste des clients DHCP. Faites suivre l'adresse IP indiquée par votre routeur du numéro de port 1080, et l'interface utilisateur du SmartPi s'affichera aussitôt.
Une liaison Wi-Fi m'aurait été bien utile, car mon routeur est loin de mon tableau électrique, mais nulle part il n'est fait mention de cette possibilité. Comme le RPi 3 embarque un module Wi-Fi, j'ai tenté sa configuration via SSH. Le nom d'utilisateur par défaut est *pi*, avec comme mot de passe *smart4pi* ou *raspberry*.

SmartPi et Wi-Fi

Connectez le SmarPi via SSh avec PuTTY ou tout autre client SSH, et ouvrez le fichier wpa_supplicant.conf avec la commande :

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Ajoutez les lignes suivantes à la fin du fichier en remplaçant mySSID et myPassPhrase par votre SSID et votre mot de passe :

```
network={
    ssid="mySSID"
    psk="myPassPhrase"
    key_mgmt=WPA-PSK
}
```

Enregistrez le fichier avec *Ctrl-0* suivi de *Entrée*, et quittez nano avec *Ctrl-X*. Débranchez le câble Ethernet, redémarrez le SmartPi, et voilà, la connexion Wi-Fi fonctionne. Pourquoi diable n'ont-ils pas explicité tout ça ?

Modification du mot de passe du SmartPi

Profitons-en pour changer le mot de passe, une opération recommandée par la documentation officielle même si, là encore, rien n'est expliqué sur le comment du truc. Le truc, le voici. (Re)lancez le client SSH, connectez le SmartPi, puis lancez l'outil de configuration du Raspberry Pi avec :

```
sudo raspi-config
```

Sélectionnez `Change User Password` et entrez un nouveau mot de passe. Voilà, c'est fait.

Raccordement du secteur au SmartPi

Là non plus la documentation n'aide pas vraiment. Par chance les entrées du SmartPi correspondent aux indications phases (L) et neutre (N) étiquetées sur la face arrière du module (**fig. 2** ; cela dit il n'y a rien sur le mien).
Le SmartPi d'évaluation m'a été livré avec trois sondes de courant précâblées. J'habite en France, et le réseau monophasé de ma maison est séparé en deux groupes. J'ai placé dans chacun une sonde et fixé la troisième sur le fil allant au chauffe-eau (second groupe). La documentation précise que les sondes ont une flèche qui doit pointer vers le consommateur d'énergie (moteur, bâtiment, etc.), mais il m'a fallu inverser ces flèches pour lire des valeurs positives.
J'ai relié les trois entrées de tension et les ai connectées à un fusible indépendant.
Lorsque vous avez réussi à obtenir des valeurs positives de tension et de puissance sur les (trois) canaux, ne reste plus qu'à interpréter les données.

Tableau de bord

Ce qui suit s'applique à la version 0.6.4 du logiciel.
L'interface web (par défaut sur le port 1080, mais vous pouvez en choisir un autre) affiche trois intensités, trois tensions et trois fréquences (**fig. 3**).
L'onglet *Measurement* de la section *Settings* permet d'inverser le sens du courant, de sélectionner un type de sonde ainsi que

les canaux de tension à surveiller. On peut aussi paramétrer la tension et la fréquence du secteur. J'ai dû redémarrer le SmartPi à chaque changement de paramètre.
Pour une raison que j'ignore, la désactivation du canal du neutre est restée sans effet.
Le SmartPi a pour ainsi dire les fils qui se touchent lorsque les intensités sont faibles : les mesures deviennent incohérentes et les valeurs de la fréquence et du cos phi font du trampoline (de 0 à plus de 50 GHz, **fig. 4**). Il ne devrait pas être difficile de corriger ça dans le code.
Attendez quelques jours de collecte des données pour mieux comprendre les courbes affichées. Les données peuvent être exportées dans un fichier CSV et exploitées dans un tableur (on peut définir le caractère servant de virgule décimale depuis l'onglet *Expert* de la section *Settings*, mais pas le séparateur de champs…)
Les courbes sont agréables à l'œil et automatiquement mises à l'échelle ; parfois c'est bien, parfois ça l'est moins, en particulier lorsqu'il y a eu un large pic dans un passé récent. Il suffit de cliquer sur le bouton correspondant pour voir ou masquer une courbe d'intensité, tension, puissance, consommation ou production d'énergie. Passez la souris sur une courbe pour visualiser les valeurs et l'horodatage à la position du curseur. La production et la consommation d'énergie de la semaine passée sont indiquées au moyen d'histogrammes.

Options et interfaces

L'API REST mentionnée plus haut permet de faire communiquer le SmartPi avec d'autres applications. La section *Settings* du tableau de bord dévoile à cet égard les intentions des développeurs puisqu'on y trouve différentes options de communication et d'interfaçage. On peut ainsi configurer une connexion UMTS, un serveur FTP et un client MQTT. Une recherche dans le dépôt GitHub nous apprend que MQTT est supposé fonctionner, mais rien n'est dit sur FTP. On devine juste que le protocole a été ajouté pour charger automatiquement des fichiers CSV via un serveur FTP.
La prochaine version devrait inclure le port RS-485 (Modbus) et le contact commandé par relais.

Une plateforme pour amateurs avertis

Les heures que j'ai passées à bidouiller le matériel et le logiciel du SmartPi me laissent une impression mitigée. D'un côté c'est un produit au potentiel indéniable, d'un autre côté l'immaturité du code et la pauvreté de la documentation (du moins à ce jour) m'ont vraiment gêné. Autrement dit il est à réserver à des amateurs avertis.
Le logiciel est à l'évidence plein de bonnes intentions, mais il manque de précision et de finesse. Cela dit il est régulièrement mis à jour – vérifiez la version en cours sur le forum [4]. Les utilisateurs sans expérience de Linux ou du RPi seront vite dépassés. C'est dommage, car il ne devrait pas être difficile de compléter le menu de configuration (qui aurait alors une gestion du mot de passe, de la configuration Wi-Fi, etc.), d'ajouter des pages d'aide et d'enrichir le manuel de quelques paragraphes bien illustrés. Le SmartPi pourrait alors intéresser un public bien plus vaste. Ceci dit, si vous recherchez une plateforme sur laquelle construire un système de mesure intelligent de la consommation et de la production d'énergie, le SmartPi est assurément un bon point de départ [5].

(160485 – version française : Hervé Moreau)

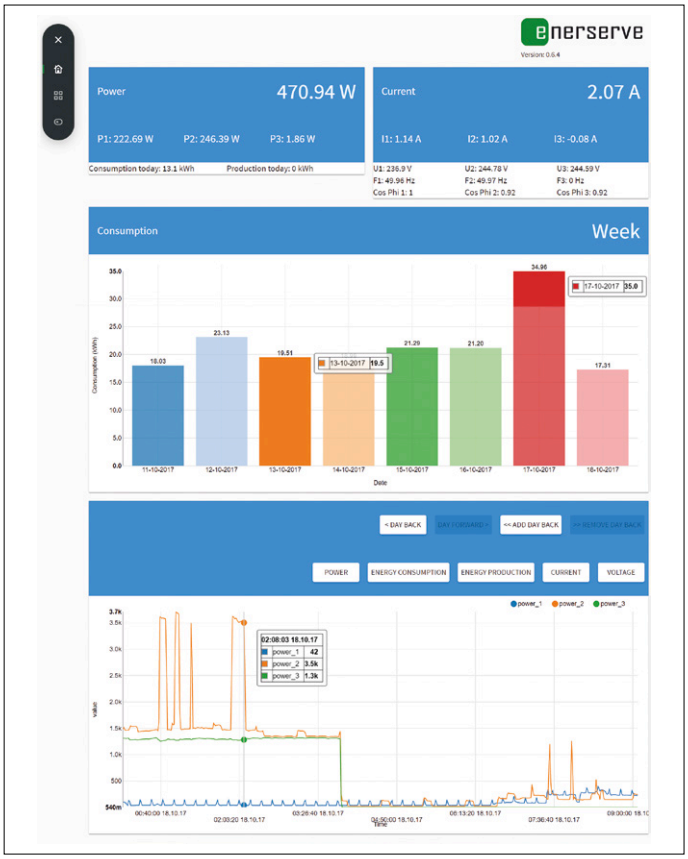


Figure 3. Le tableau de bord affiche les consommations actuelles et passées.

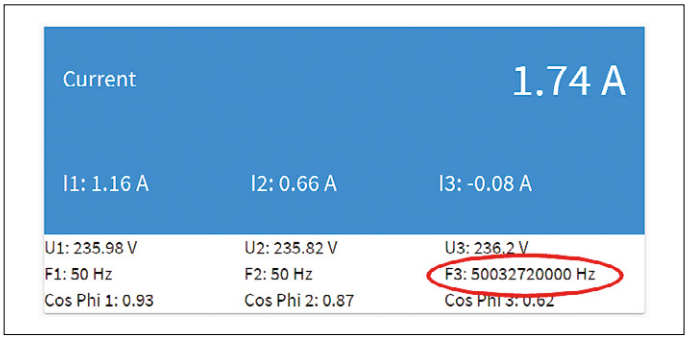
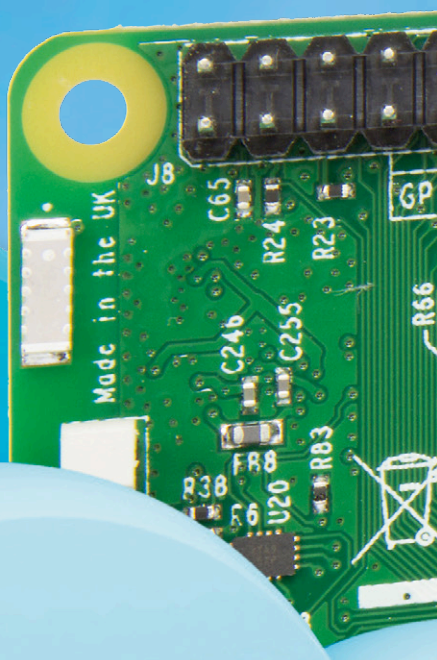


Figure 4. Une fréquence secteur de 50 GHz ? Qui peut y croire ?

Liens

[1] SmartPi sur GitHub : https://github.com/nDenerserve/SmartPi
[2] Page d'accueil : http://www.emanager.eu/en/products/smartpi
[3] Kickstarter : https://www.kickstarter.com/projects/1240982104/smartpi-turn-your-raspberry-pi-into-a-smartmeter
[4] Forum : https://forum.enerserve.eu/
[5] www.elektormagazine.fr/160485

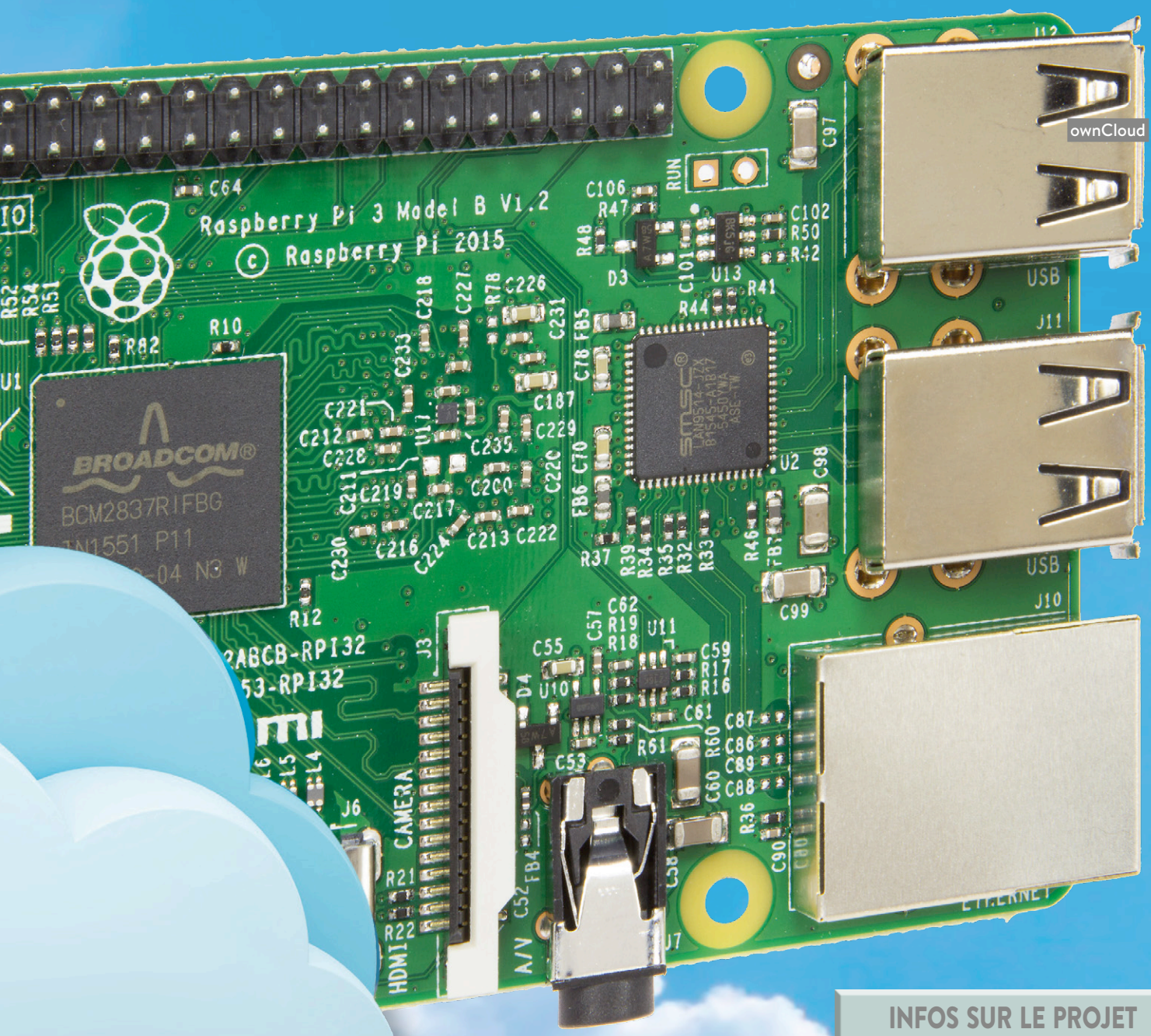
un nuage en forme de framboise



installez un *cloud* dans votre salon

Sébastien Guerreiro de Brito (Montrieux-en-Sologne, France)

Aujourd'hui le *cloud* est une solution populaire pour stocker et partager des fichiers et qui permet d'accéder à ses fichiers depuis n'importe quel ordinateur n'importe où dans le monde. Puisque beaucoup de gens se méfient de la fiabilité et la sécurité des *clouds*, cet article vous expliquera comment réaliser votre propre nuage et sa carte d'alimentation.



INFOS SUR LE PROJET

Raspberry Pi
Cloud Nuage
Apache2 HTTPS

débutant
➡ connaisseur
expert

env. 4 h

Ordinateur,
fer à souder pour CMS

env. 150 €

Comment réaliser un *cloud* ?

Un *cloud* est constitué d'une partie matérielle et d'une partie logicielle. Côté matériel, j'ai choisi la plateforme Raspberry Pi agrémentée d'un disque dur pour pouvoir stocker un grand nombre de fichiers. Parce que les disques durs consomment beaucoup de courant, j'ai décidé de concevoir une alimentation capable d'alimenter correctement l'ensemble Raspberry Pi-disque dur (voir l'encart et les **figures 1 et 2**)

Caractéristiques

- Accessible partout dans le monde
- Partagez vos fichiers, photos et applications
- Touchez le nuage

Pour la partie logicielle, j'ai choisi la solution « ownCloud ». Présenté comme une alternative à des services comme DropBox, ownCloud est un logiciel libre pour créer des plateformes de services de stockage et de partage de fichiers. Il présente les caractéristiques suivantes :

- C'est gratuit, à code source ouvert et sans aucune limitation.
- L'administration se fait par une page web.
- L'administrateur peut gérer les utilisateurs selon leur nom ou leur groupe.
- L'envoi d'un lien internet suffit pour effectuer un partage.
- Les fichiers sont cryptés durant leur transfert.
- Il existe un client pour synchroniser les fichiers choisis en local sous Windows ou Linux
- ...

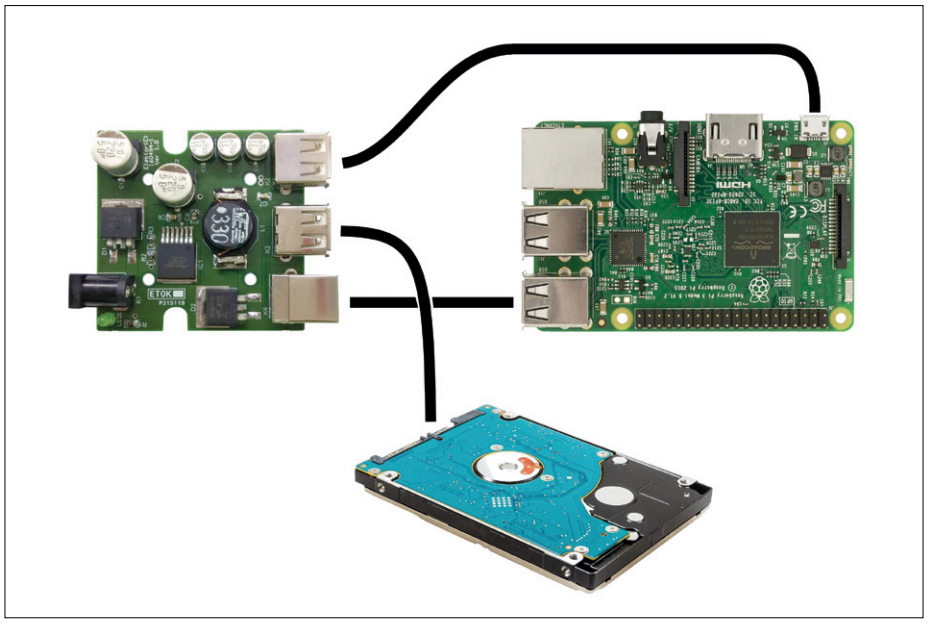


Figure 1. Voici comment raccorder un disque dur de type SATA alimenté uniquement par USB.

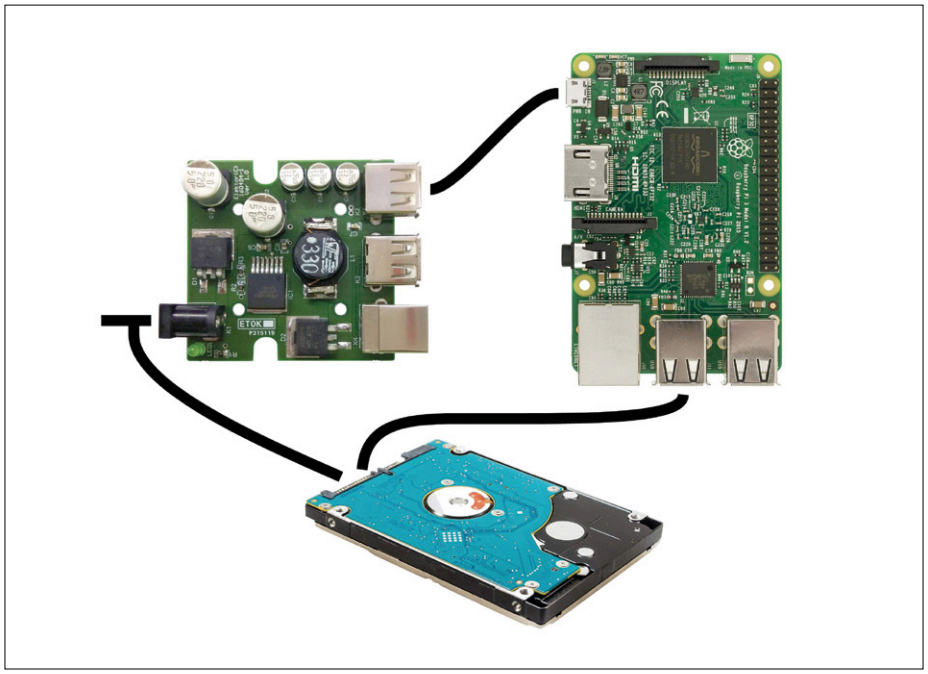


Figure 2. Un disque dur de type IDE dont le convertisseur USB possède sa propre alimentation est raccordé ainsi.

C'est plutôt pas mal et il ne semble pas y avoir beaucoup de différences avec d'autres systèmes gratuits et bien connus. Par contre, les avantages sont nombreux :

- On sait exactement où sont stockés les fichiers.
- La taille de stockage peut être conséquente, voire bien supérieure aux solutions existantes.
- On peut arrêter le système quand on le souhaite.
- Etc.

Préparer le Raspberry Pi

Pour commencer, il faut télécharger l'image Raspbian sur le site officiel du Raspberry Pi [1] puis l'installer. Pour cet article, nous avons utilisé `2017-11-29-raspbian-stretch-lite`.
L'image en version « *Lite* » suffit puisque nous n'aurons pas d'écran avec interface graphique. Suivez les instructions d'installation du site officiel ou référez-vous à l'un des nombreux tutoriels disponibles sur l'internet.
Une fois l'image gravée sur la carte microSD (c'est facile avec « Etcher », https://etcher.io/), insérez-la dans le lecteur du Raspberry Pi. Branchez un écran et un clavier, puis l'alimentation. Connectez-vous au Raspberry Pi. Je vous rappelle que le login par défaut est « pi » ; le mot de passe par défaut est « raspberry ». Lancez ensuite la configuration de la carte en tapant la commande

```
sudo raspi-config
```

À partir du menu de configuration, réglez le clavier selon votre langue (*localisation*). Je vous conseille d'activer le service SSH. Il vous sera utile pour intervenir sur la carte à distance. Pour des raisons de sécurité, profitez-en pour changer votre mot de passe.
Il est également conseillé de donner une adresse IP fixe au Raspberry Pi :

```
sudo nano /etc/network/interfaces
```

Ajouter les lignes :

```
auto eth0
iface eth0 inet static
address 192.168.yyy.xxx
```

```
          <= Adresse IP souhaitée
netmask 255.255.255.0
gateway 192.168.yyy.zzz
          <= Adresse IP de votre
   box
```

Appuyez sur Ctrl-O puis Entrée pour sauvegarder le fichier, Ctrl-X pour quitter l'éditeur. Vous pouvez également faire cela en configurant votre box pour qu'elle attribue toujours la même adresse IP au Raspberry Pi.

Configuration du disque dur de stockage

Commençons par créer le dossier sur lequel sera monté le disque dur :

```
sudo mkdir /mnt/usb
```

Puis changeons les droits sur ce dossier et montons le disque qui, dans notre cas, s'appelle sda1. Son système de fichiers est `ext4` :

```
sudo chmod -R a+w /mnt/usb
sudo mount /dev/sda1 /mnt/usb/
```

Passons maintenant au montage automatique du disque dur externe au démarrage du Raspberry Pi. Pour cela nous devons découvrir le PARTUUID du disque :

```
sudo blkid
```

Cherchez votre disque dans la liste et notez son PARTUUID puis ouvrez le fichier `fstab` :

```
sudo nano /etc/fstab
```

Ajoutez à la fin du fichier la ligne suivante :

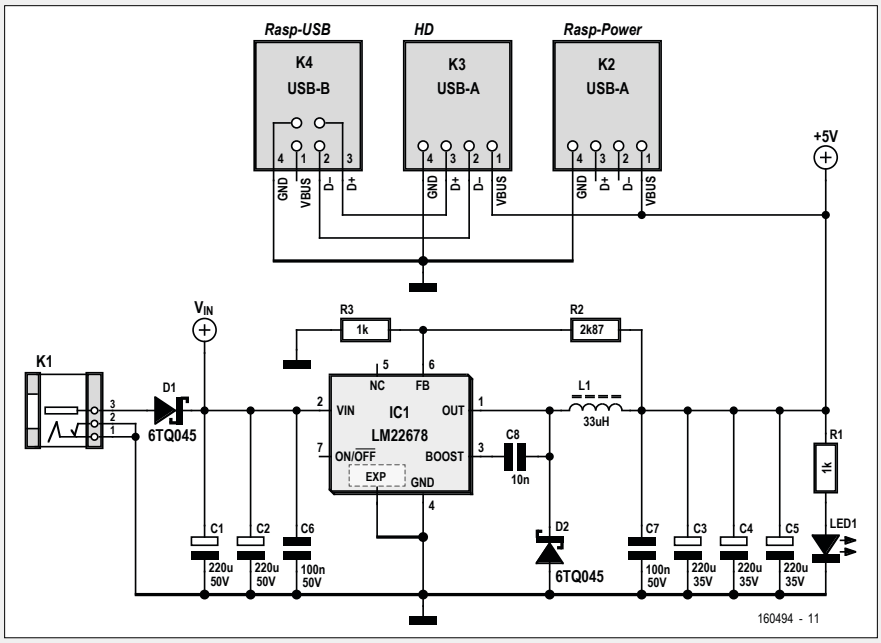
```
PARTUUID=6f20736b-01 /mnt/usb ext4
   defaults 0 2
```

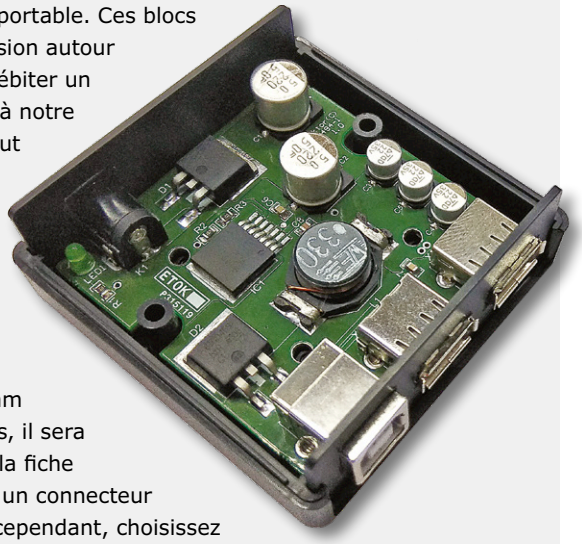
Ici « 6f20736b-01 » correspond à mon disque dur, remplacez cette valeur par l'identifiant de votre disque dur. Appuyez sur Ctrl-O puis Entrée pour sauvegarder le fichier, Ctrl-X pour quitter l'éditeur. Attention à ne pas faire des fautes de frappe, car les résultats peuvent être désastreux, vous seriez obligé de réinstaller l'image sur la carte SD, etc.
Redémarrez le Raspberry Pi avec la commande :

```
sudo reboot
```

La carte d'alimentation

Le schéma de l'alimentation est basé sur un régulateur à découpage LM22678. Il est capable de fournir un courant de 5 A maximum. Les plus gourmands des disques durs SATA peuvent consommer jusqu'à 2 A sous 5 V. Si on considère que le nano-ordinateur Raspberry Pi consomme au maximum 1 A sous 5 V, notre régulateur devrait faire l'affaire.





Pour alimenter notre *cloud*, l'idée est de prendre un vieux bloc secteur d'ordinateur portable. Ces blocs délivrent généralement une tension autour de 19 V et sont en mesure de débiter un courant de 5 A, ce qui convient à notre application. Notre régulateur peut supporter des tensions d'entrée jusqu'à 42 V, c'est donc parfait.
Pour être compatible avec un maximum de blocs d'alimentation, notre carte d'alimentation est équipée d'un connecteur femelle CC de 2,1 mm (*barrel jack*). Pour certains blocs, il sera peut-être nécessaire de couper la fiche d'origine et de la remplacer par un connecteur mâle CC de 2,1 mm. Attention cependant, choisissez un modèle capable de supporter un courant de 5 A.
La diode D1 protège le montage contre les inversions de polarité de la tension d'entrée.
Le régulateur IC1 existe en deux versions de tension de sortie : ajustable ou fixe de 5 V. J'ai utilisé la version ajustable, car je l'avais sous la main :-) La tension de sortie est fixée par le rapport entre les valeurs des résistances R2 et R3.

$V_{out}=1{,}285\times(R2/R3+1)$

Avec R2 = 2,87 kΩ et R1 = 1 kΩ, on obtient une valeur de V_{out} de 4,97 V. Dans le cas d'un régulateur non ajustable, il ne faut pas monter R3 et R2 = 0 Ω (ou un strap).
Les deux gros condensateurs chimiques en entrée du circuit servent à pallier les appels de courant du disque dur SATA au démarrage.

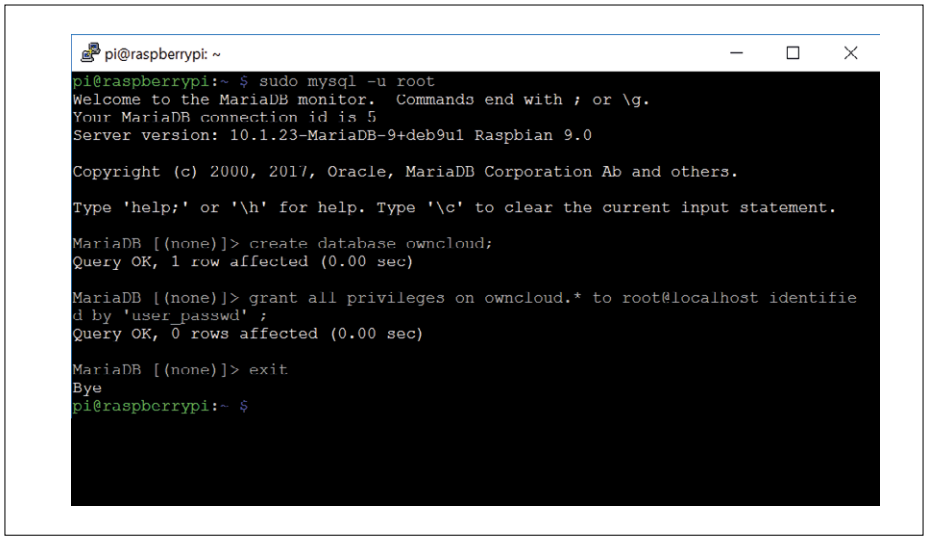


Figure 3. Préparation de la base de données pour l'utilisateur « root » avec mot de passe « user_passwd ».

Figure 4. Installation réussie, vous avez désormais accès à votre *cloud*.

Si tout se passe bien, vous pouvez lire (après connexion, bien sûr) le contenu de `/mnt/usb` avec la commande :

```
ls /mnt/usb
```

Installer ownCloud

L'application ownCloud a besoin de PHP version 5.6 ou plus récente et du serveur web Apache2. Nous installons donc Apache2, MySQL, PHP 7.0 plus une ribambelle de modules supplémentaires :

```
sudo -i
```

Maintenant nous sommes en mode « *Super User* » ce qui nous enlève l'obligation de précéder chaque commande par « sudo ».

```
apt-get update
```

Les lignes ci-dessous constituent une seule commande :

```
apt install -y apache2 mariadb-
   server libapache2-mod-php7.0 \
php7.0-gd php7.0-json php7.0-mysql
   php7.0-curl \
php7.0-intl php7.0-mcrypt php-
   imagick \
php7.0-zip php7.0-xml
   php7.0-mbstring
```

Ensuite nous téléchargeons le paquet ownCloud, nous le déballons et nous le copions dans notre répertoire. Attention, le numéro de version du paquet peut être différent de celui utilisé ici (10.0.4), modifiez-le si besoin :

```
wget https://download.owncloud.
   org/community/owncloud-
   10.0.4.tar.bz2
tar -xjf owncloud-10.0.4.tar.bz2
sudo cp -r owncloud /mnt/usb
```

Créer une base de données

Pour permettre à ownCloud de fonctionner, il faut créer une base de données :

```
mysql -u root
  create database owncloud;
  grant all privileges on
    owncloud.* to <user>@localhost
    identified by '<user_passwd>';
  exit;
```

Le nom `<user>` est le nom que nous donnerons à l'utilisateur de la base de données et `<user_passwd>` sera son mot de passe (**fig. 3**).

Configurer le serveur web

Apache2 a besoin de connaître quelques détails sur l'emplacement d'ownCloud. Nous plaçons ces informations dans un fichier de configuration :

```
nano /etc/apache2/conf-available/
   owncloud.conf
```

Comme contenu nous mettons :

```
Alias /owncloud "/mnt/usb/
   owncloud/"
<Directory /mnt/usb/owncloud/>
Options +FollowSymlinks
AllowOverride All
<IfModule mod_dav.c>
Dav off
</IfModule>
<IfVersion < 2.3>
order allow,deny
allow from all
</IfVersion>
<IfVersion >= 2.3>
Require all granted
</IfVersion>
SetEnv HOME /mnt/usb/owncloud
SetEnv HTTP_HOME /mnt/usb/owncloud
</Directory>
```

Appuyez sur Ctrl-O puis Entrée pour sauvegarder le fichier, Ctrl-X pour quitter l'éditeur.
Finissons la configuration avec quelques commandes supplémentaires :

```
ln -s /etc/apache2/conf-available/
   owncloud.conf /etc/apache2/
   conf-enabled/owncloud.conf
a2enmod rewrite
a2enmod headers
service apache2 restart
chmod a+w /mnt/usb/owncloud/config
```

Première connexion à ownCloud

Pour permettre au serveur web du Raspberry Pi (Apache2) d'accéder aux fichiers du cloud, nous allons devoir modifier les droits. L'utilisateur et le groupe par défaut pour Apache2 sont « www-data » Exécutez alors les commandes :

```
chown www-data:www-data -R /mnt/
   usb/owncloud
chmod 770 -R /mnt/usb/owncloud
```

Avec votre navigateur préféré, allez à l'adresse (remplacez « yyy.xxx » par les valeurs correspondantes pour votre réseau) :

```
http://192.168.yyy.xxx/owncloud
```

Vous serez alors invité à créer un compte administrateur pour gérer l'application (**fig. 4**). Entrez le chemin où seront stockées les données (`/mnt/usb/owncloud/data` dans notre exemple) puis sélectionnez le type de base de données installée (mysql). Pour terminer, entrez le nom de l'utilisateur que vous avez saisi (<user>), son mot de passe (<user_passwd>), le nom de la base de données (« owncloud » dans notre cas) et enfin, pour l'adresse de la base, laissez « localhost ».
Et voilà ! Vous pouvez désormais utiliser votre propre cloud (**fig. 5**), soit depuis votre navigateur internet, soit en utilisant un client ownCloud que vous pourrez télécharger sur le site d'ownCloud.

Se connecter via HTTPS

Lors de votre première connexion vous pourrez constater qu'ownCloud recommande une connexion sécurisée par HTTPS. Pour reconfigurer notre système, envoyez les commandes suivantes :

```
sudo -i
a2enmod ssl
service apache2 restart
a2ensite default-ssl.conf
service apache2 restart
```

Le module SSL a besoin de clés, créons-les :

```
cd /etc/apache2
mkdir certOC
cd certOC
openssl genrsa -out owncloud.key
   1024
openssl req -new -key owncloud.key
   -out owncloud.csr
```

Vous allez devoir répondre à un certain nombre de questions pour remplir le certificat, notez vos réponses quelque part au cas où (ou acceptez les valeurs par défaut).

Accès « *World Wide* » ?

Pour accéder à votre cloud depuis l'ordinateur de votre bureau ou de n'importe quel autre lieu sur la planète, il vous faut idéalement disposer d'une adresse DNS dynamique. Pour cela je vous conseille le service « NO-IP » [3]. Allez sur ce site et enregistrez le nom de domaine souhaité. C'est gratuit et la seule contrepartie est que tous les mois vous recevez un message pour continuer à utiliser votre compte.
Pour l'installation, c'est assez simple. Depuis votre carte Raspberry Pi, exécutez les commandes ci-dessous dans un dossier de travail de votre choix :

```
sudo wget https://www.noip.com/client/linux/noip-
   duc-linux.tar.gz
tar xf noip-duc-linux.tar.gz
```

Vérifiez la version de noip (2.1.9-1 dans notre cas) et modifiez la commande suivante si besoin :

```
cd noip-2.1.9-1
```

Avant de lancer l'installation, assurez-vous de connaître vos login et mot de passe pour NO-IP :

```
sudo make install
```

Maintenant nous avons besoin d'un script pour lancer le service au démarrage. Téléchargez le script « noip » sur [4] puis copiez-le dans `/etc/init.d/`. Pour que le service « noip » se lance au démarrage, il vous faut taper les lignes suivantes :

```
sudo chmod 755 /etc/init.d/noip
sudo update-rc.d noip defaults
sudo service noip configure
```

Démarrez le service à la main :

```
sudo service noip start
```

Vous pouvez vérifier l'état du service par la commande :

```
sudo service noip status
```

Enfin, pour pouvoir utiliser OwnCloud depuis l'extérieur, il faut modifier un paramètre dans le fichier `/mnt/usb/owncloud/config/config.php` :

```
'trusted_domains' =>
  array (
    0 => '192.168.yyy.xxx',
    1 => 'votre_nom_domaine',
  ),
```

Sur votre réseau local, il faut utiliser votre adresse interne (par exemple https://192.168.yyy.xxx/owncloud) pour paramétrer votre client de synchronisation.
Après avoir configuré votre box (voir [3]), vous pouvez désormais partager vos fichiers avec le monde entier !

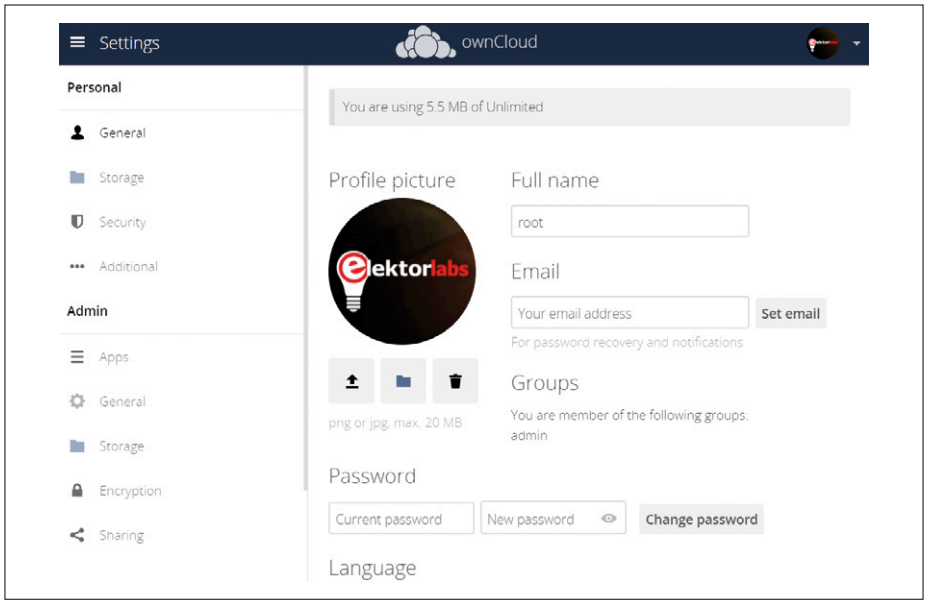


Figure 5. Ça y est, nous voilà devant le tableau de bord de notre *cloud*. Maintenant c'est à vous de piloter votre nuage.

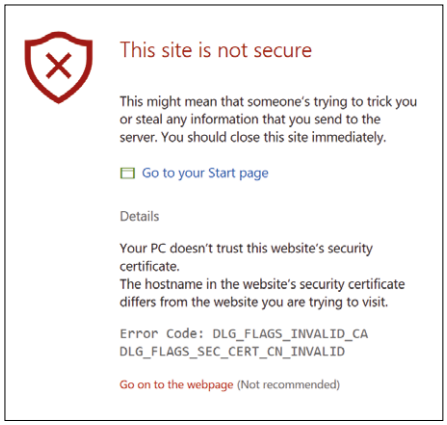


Figure 6. Pas de panique ! Vous pouvez faire confiance à vous-même et à vos propres systèmes, non ?

Ensuite :

```
openssl x509 -req -days 365 -in
   owncloud.csr -signkey owncloud.
   key -out owncloud.crt
```

On copie les clés :

```
cp owncloud.crt /etc/ssl/certs
cp owncloud.key /etc/ssl/private
```

Un deuxième fichier de configuration d'ownCloud est nécessaire :

```
nano /etc/apache2/sites-available/
   owncloud.conf
```

Avec comme contenu :

```
<VirtualHost *:443>
DocumentRoot /var/www
SSLEngine On
SSLOptions +FakeBasicAuth
   +ExportCertData +StrictRequire
SSLCertificateFile /etc/ssl/certs/
   owncloud.crt
SSLCertificateKeyFile /etc/ssl/
   private/owncloud.key
</VirtualHost>
```

Appuyez sur Ctrl-O puis Entrée pour sauvegarder le fichier, Ctrl-X pour quitter l'éditeur.
Enfin, on active la nouvelle configuration et on redémarre le serveur web

LISTE DES COMPOSANTS

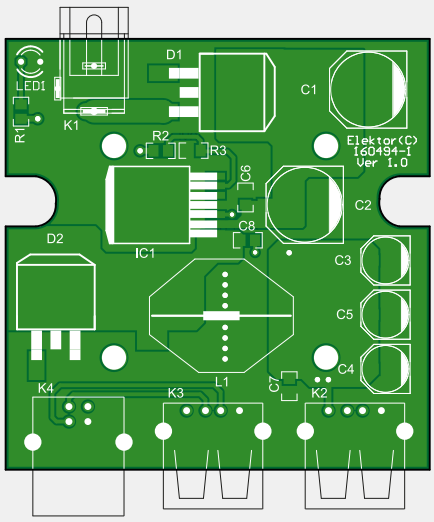
Résistances (0805, 0,1 W)
R1,R3 = 1 kΩ, 1%
R2 = 2,87 kΩ, 1%

Condensateurs
C1,C2 = 220 µF 50 V, CMS, Ø = 10 mm
C3,C4,C5 = 22 µF, 35 V, CMS, Ø = 5 mm
C6,C7 = 100 nF, 0805
C8 = 10 nF, 0805

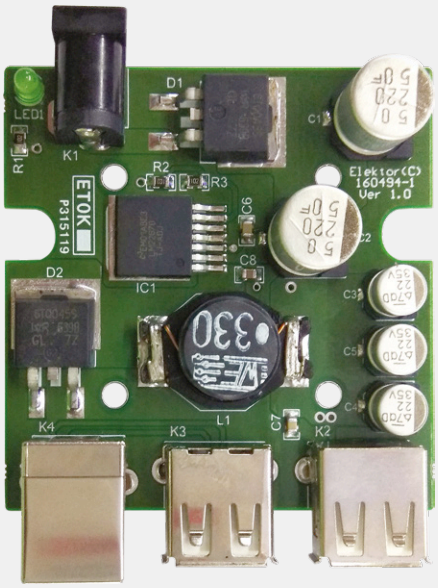
Inductance
L1 = 33 µH, 3,4 A, 70 mΩ (Würth 74457133)

Semi-conducteurs
D1,D2 = VS-6TQ045SPBF
IC1 = LM22678TJ
LED1 = LED, verte, 3 mm

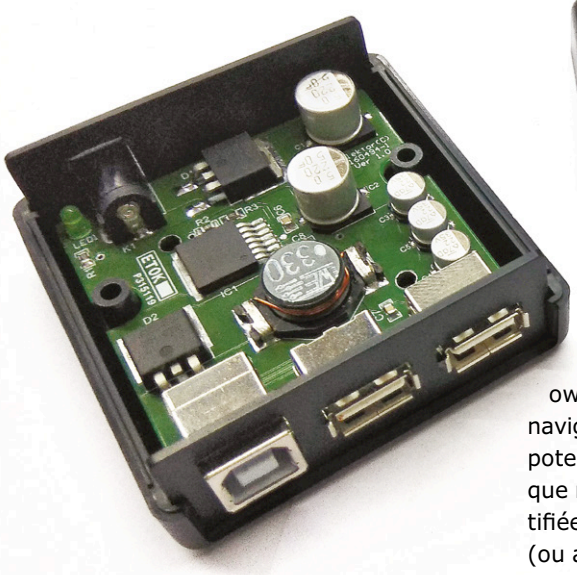
Divers
K1 = connecteur femelle CC, 5 A, 2,1 mm



K2,K3 = connecteur USB de type A
K4 = connecteur USB de type B
Coffret Hammond 1593KBK



Circuit imprimé, réf. 160494-1



Apache2 :

```
a2ensite owncloud.conf
apachectl configtest
service apache2 restart
```

Vous pouvez désormais vous connecter à votre cloud en utilisant l'adresse https://192.168.yyy.xxx/owncloud (avec un 's' à HTTP). Votre navigateur va vous prévenir d'un danger potentiel (**fig. 6**) parce que la signature que nous venons de créer n'est pas certifiée. Vous pouvez ignorer cette erreur (ou ajouter une exception) et continuer ainsi. Cela ne gêne en aucun cas le fonctionnement de votre nuage.

Et maintenant ?

Vous êtes désormais en possession de votre propre cloud, d'une taille conséquente et dont vous maîtrisez tout. Vous pouvez choisir d'allumer ou d'éteindre votre machine quand vous le souhaitez, lancer une sauvegarde automatique de votre cloud, etc. C'est vous qui êtes aux commandes, c'est vous qui pilotez le nuage. ⏮

(160494)

Liens

[1] Raspbian : www.raspberrypi.org

[2] ownCloud : owncloud.org

[3] Service NO-IP : www.noip.com

[4] Page de cet article : www.elektormagazine.fr/160494

framboise aromatisée à l'orange

économique, pratique, bonne !

Tam Hanna

Le nano-ordinateur Raspberry Pi est parfois cher pour l'application envisagée. La famille Orange Pi fabriquée en Chine comprend des douzaines de cartes alternatives qui, du point de vue du prix et de la puissance de calcul, sont plus attrayantes. Cet article présente l'un des membres de la famille : la carte Orange Pi Lite.

La *Raspberry Pi Foundation* d'Eben Upton a longtemps accaparé le marché amateur de l'ordinateur monocarte : le processeur, très bon marché, car issu des fonds de tiroir de Nokia, a permis au dirigeant de Broadcom de concevoir un ordinateur qu'au début, il a été difficile de concurrencer.

Un vieux dicton prétend que « tôt ou tard, à toute situation, des Chinois viendra la solution ». La société Allwinner, fabricant chinois de processeurs, ne pouvait évidemment pas tolérer la marche triomphale de son concurrent Broadcom sans réagir : sa réponse fut l'Orange Pi, apparu en 2014. Entretemps, la famille s'est considérablement agrandie : on trouve plus d'une douzaine de modèles différents sur AliExpress.

Quand on s'intéresse à l'électronique, on cherche toujours à obtenir le meilleur coût pour la plus grande puissance de calcul. La famille Orange Pi [1] de Shenzhen Xunlong est intéressante pour tous ceux qui veulent gérer du matériel avec un ordinateur Linux et qui peuvent se contenter de la puissance d'une carte de la classe d'un Raspberry Pi. Les cartes chinoises plairont à tous ceux qui accordent de l'importance à une bonne connectivité aux réseaux sans fil. D'autres préféreront une alternative au RPi parce qu'ils apprécient modérément l'agitation politique de la Fondation Raspberry Pi.

Disponibilité

Bien que les ordinateurs Orange Pi soient aujourd'hui disponibles chez l'un ou l'autre revendeur, le vrai connaisseur achète ses cartes directement en Chine chez le fabricant. AliExpress sert de distributeur ; Shenzhen Xunlong propose aussi sur ce site web des kits comprenant des accessoires divers.

Se pose la question de savoir quelles cartes vont intéresser quels utilisateurs. Le schéma de la **figure 1** montre la répartition des différentes cartes Orange Pi. On remarque qu'à l'exception de la famille Zéro, une bonne partie, dont le brochage est représenté sur la **figure 2**, est grosso modo compatible avec le RPi (voir

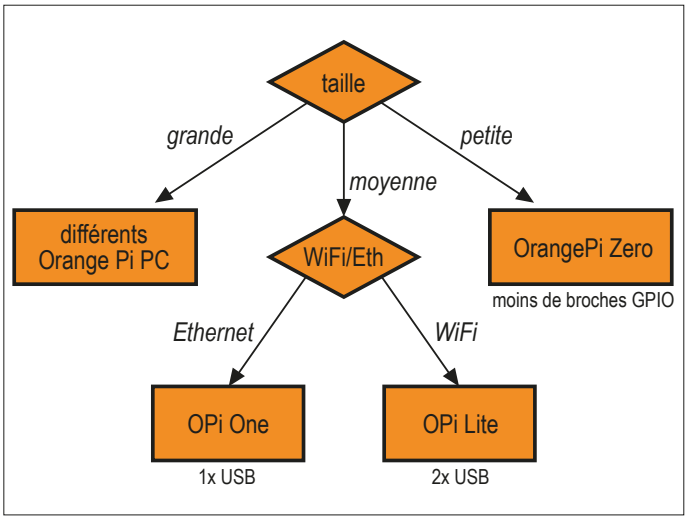


Fig. 1. Une présentation ordonnée de la famille.

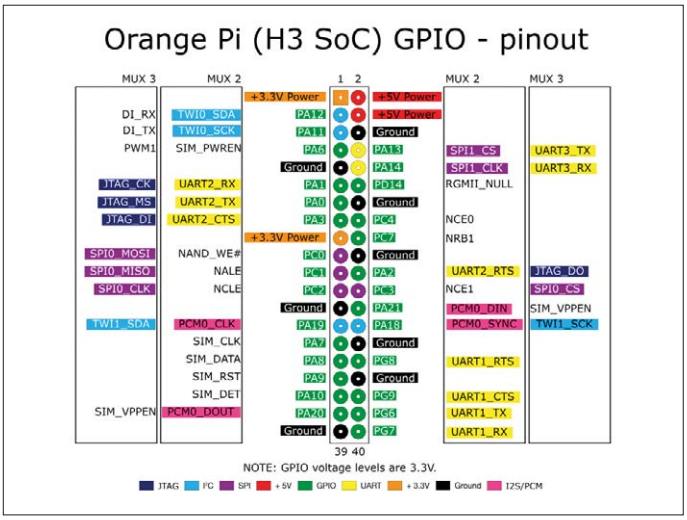


Fig. 2. L'assignation des 40 broches du connecteur rappelle pour partie celle du RPi (source : www.orangepi.org).

aussi la **figure 3**). Dans ce qui suit, nous nous intéresserons au nano-ordinateur Orange Pi Lite. Avec ses 28,8 g, il est plus léger que le RPi 3, environ douze dollars moins cher, mais comprend néanmoins deux ports USB et une antenne Wi-Fi. L'Orange Pi One ne présente, au lieu du Wi-Fi et des deux ports USB, qu'un seul port USB, mais y ajoute un port Ethernet, et il est encore moins cher (neuf dollars environ).

Android à bord

La mémoire flash embarquée de l'Orange Pi PC Plus contient une version chinoise d'Android qui, selon certaines rumeurs, ne serait guère qu'à moitié utilisable.

Configuration

Il y a une chose qu'on doit reconnaître à la Fondation Raspberry Pi : si vous achetez l'un de leurs ordinateurs, vous êtes paré du point de vue du logiciel et de l'écosystème. Un RPi peut s'alimenter avec n'importe quel chargeur micro-USB, ce qui simplifie considérablement la mise en service.

Assez curieusement, l'Orange Pi Lite présente également un connecteur micro-USB, mais qui est inutilisable pour l'alimenter du fait de la présence d'un composant intermédiaire (voir aussi [2]). L'alimentation s'effectue par un connecteur femelle ou bien les broches 2, 4 et 6 auxquelles sont connectés le +5V (broches 2 et 4) et la masse (broche 6). Ce « point faible » vaut pour toute la famille Orange Pi : si l'on tient absolument à alimenter son nano-ordinateur par une prise micro-USB, il faut renoncer aux produits de Shenzhen Xunlong.

Le problème suivant concerne le logiciel : les images système disponibles sous [3] ne sont guère plus que de la poudre aux yeux. Les liens ne fonctionnent parfois pas du tout et les fichiers qu'on parvient à télécharger ne bénéficient que d'une maintenance médiocre.

Dans la communauté des utilisateurs de l'Orange Pi, le projet Armbian est devenu une norme de fait. On peut le télécharger sous [4]. Notez que, pour l'Orange Pi Lite, il en existe une version Ubuntu et une version Debian. Les différences entre ces deux versions sont minimes, mais elles se manifestent tôt ou tard et exigent un complément de test.

Téléchargez l'image Debian [4B] et gravez-la sur une carte SD de taille suffisante. Branchez l'écran HDMI, la souris et le clavier sur le nano-ordinateur, puis l'alimentation. Ne soyez pas surpris que le démarrage dure jusqu'à une minute entière ; les ordinateurs Orange Pi prennent leur temps avant d'allumer leur écran.

C'est Linux !

Les images Armbian sont des systèmes informatiques classiques : au premier démarrage, on se trouve face à un écran de demande de connexion. Les paramètres corrects diffèrent d'une image à l'autre : pour l'Orange Pi Lite, il faut saisir `root` et `1234`.

À la première connexion, le système demande de changer le mot de passe par sécurité puis lance un assistant de configuration. La saisie du nom de l'utilisateur est obligatoire, les autres rubriques peuvent être ignorées par pression répétée sur la touche Entrée. Après ce travail, l'ordinateur redémarre et on se retrouve face à l'interface graphique d'Armbian (**fig. 4**).

Armbian est Debian : cette distribution Linux très ancienne est aussi la base de Raspbian, le système du Raspberry Pi. Il s'ensuit que le RPi et l'Orange Pi sont plus ou moins compatibles au niveau applicatif. La première petite différence apparaît dans la gestion du matériel. Alors que le RPi s'appuie sur la bibliothèque *Wiring Pi*, la société Shenzhen Xunlong utilise *WiringOP*.

Avant le premier emploi, la bibliothèque doit être téléchargée et compilée, ce qui, dans l'idéal, demande la séquence de commandes suivante :

```
cd ~
mkdir opigpiolib
git clone https://github.com/zhaolei/WiringOP.git -b
   h3
cd WiringOP
chmod +x ./build
sudo ./build
```

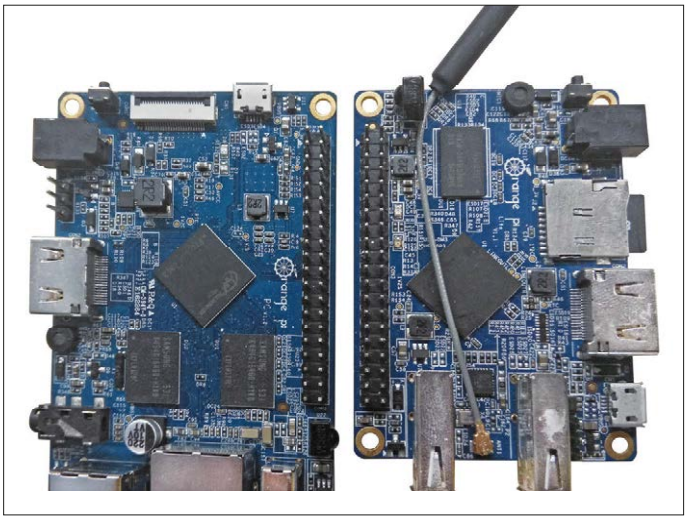


Fig. 3. Attention : la disposition du connecteur varie d'une carte à l'autre !

Fig. 4. Ça ressemble au Linux standard.

```
$ gpio readall
 +-----+-----+----------+------+---+-Orange Pi+---+---+------+---------+-----+-----+
 | BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+----------+------+---+----++----+---+---+------+---------+-----+-----+
 |     |     |     3.3v |      |   |  1 || 2  |   |   |      | 5v      |     |     |
 |  12 |   8 |    SDA.0 | ALT5 | 0 |  3 || 4  |   |   |      | 5V      |     |     |
 |  11 |   9 |    SCL.0 | ALT5 | 0 |  5 || 6  |   |   |      | 0v      |     |     |
 |   6 |   7 |  GPIO.7  | ALT3 | 0 |  7 || 8  | 0 | ALT3 | TxD3    | 15  | 13  |
 |     |     |       0v |      |   |  9 || 10 | 0 | ALT3 | RxD3    | 16  | 14  |
 |   1 |   0 |    RxD2  | ALT3 | 0 | 11 || 12 | 0 | ALT3 | GPIO.1  | 1   | 110 |
 |   0 |   2 |    TxD2  | ALT3 | 0 | 13 || 14 |   |   |      | 0v      |     |     |
 |   3 |   3 |    CTS2  |   IN | 0 | 15 || 16 | 0 | ALT3 | GPIO.4  | 4   | 68  |
 |     |     |     3.3v |      |   | 17 || 18 | 0 | ALT3 | GPIO.5  | 5   | 71  |
 |  64 |  12 |     MOSI |   IN | 0 | 19 || 20 |   |   |      | 0v      |     |     |
 |  65 |  13 |     MISO |   IN | 0 | 21 || 22 | 0 | ALT3 | RTS2    | 6   | 2   |
 |  66 |  14 |     SCLK |   IN | 0 | 23 || 24 | 0 | ALT4 | CE0     | 10  | 67  |
 |     |     |       0v |      |   | 25 || 26 | 0 | ALT3 | GPIO.11 | 11  | 21  |
 |  19 |  30 |    SDA.1 | ALT4 | 0 | 27 || 28 | 0 | ALT4 | SCL.1   | 31  | 18  |
 |   7 |  21 |  GPIO.21 | ALT3 | 0 | 29 || 30 |   |   |      | 0v      |     |     |
 |   8 |  22 |  GPIO.22 |  OUT | 0 | 31 || 32 | 0 | ALT3 | RTS1    | 26  | 200 |
 |   9 |  23 |  GPIO.23 |  OUT | 1 | 33 || 34 |   |   |      | 0v      |     |     |
 |  10 |  24 |  GPIO.24 |  OUT | 0 | 35 || 36 | 0 | ALT3 | CTS1    | 27  | 201 |
 |  20 |  25 |  GPIO.25 |  OUT | 0 | 37 || 38 | 0 | ALT3 | TxD1    | 28  | 198 |
 |     |     |       0v |      |   | 39 || 40 | 0 | ALT3 | RxD1    | 29  | 199 |
 +-----+-----+----------+------+---+----++----+---+---+------+---------+-----+-----+
 | BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+----------+------+---+-Orange Pi+---+---+------+---------+-----+-----+
```

Fig. 5. L'assignation des broches GPIO est assez complexe.

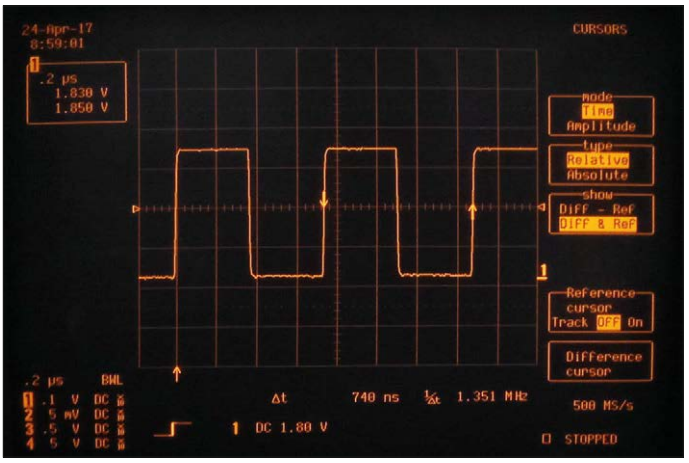


Fig. 6. Ces créneaux semblent tout à fait utilisables.

Pour le pas suivant, vous pouvez utiliser un petit programme de démonstration qui produit un signal carré sur l'une des broches de sortie GPIO. Pour cela, créez un nouveau fichier appelé `main.c` avec le contenu suivant :

```
#include <wiringPi.h>
int main (void) {
  wiringPiSetup ();
  pinMode (0, OUTPUT) ;
  for (;;) {
    digitalWrite (0, HIGH) ;
    digitalWrite (0,  LOW) ;
    digitalWrite (0, HIGH) ;
    digitalWrite (0,  LOW) ;
 }
  return 0 ;
}
```

La compilation s'effectue plus ou moins comme d'habitude sur le RPi :

```
$ gcc -Wall main.c -lwiringPi
$ sudo ./a.out
```

Pour certaines versions de l'Orange Pi, la compilation produit des erreurs qui indiquent l'absence de la bibliothèque *pthread* :

```
$ gcc -Wall main.c -lwiringPi
//usr/local/lib/libwiringPi.so: undefined reference
   to `pthread_join'
//usr/local/lib/libwiringPi.so: undefined reference
   to `pthread_create'
//usr/local/lib/libwiringPi.so: undefined reference
   to `pthread_cancel'
collect2: error: ld returned 1 exit status
```

Il s'agit habituellement là d'une erreur de configuration du compilateur GCC, qui n'inclut pas automatiquement *pthread*. Pour la corriger, il suffit d'ajouter un paramètre à la commande de compilation :

```
$ gcc -Wall main.c -lwiringPi -pthread
$
```

Dans le cas idéal, l'identification des différentes broches du connecteur GPIO s'obtient au moyen de la commande `gpio readall`. Pour simplifier, la **figure 5** présente un exemple de ce qui s'affiche pour un Orange PC Plus à 40 broches. Cela fait, nous pouvons maintenant connecter l'ordinateur à notre attelage habituel : l'analyseur de domaine de modulation et l'oscilloscope à mémoire numérique. Les **figures 6** et **7** montrent ce à quoi vous pouvez vous attendre.

L'activation de l'UART

Poursuivons avec la lecture de badges RFID. Le module Seeed 713-113020002, disponible chez

Difficultés alimentaires

Les cartes orange Pi sont très sensibles à une tension d'alimentation trop faible : outre des plantages lorsqu'on active l'interface graphique, on observe des erreurs de lecture de la carte SD. Si vous avez des problèmes au démarrage d'un nouvel ordinateur, commencez par vérifier la chute de tension dans le câble d'alimentation : ces ordinateurs s'autorisent jusqu'à 800 mA !

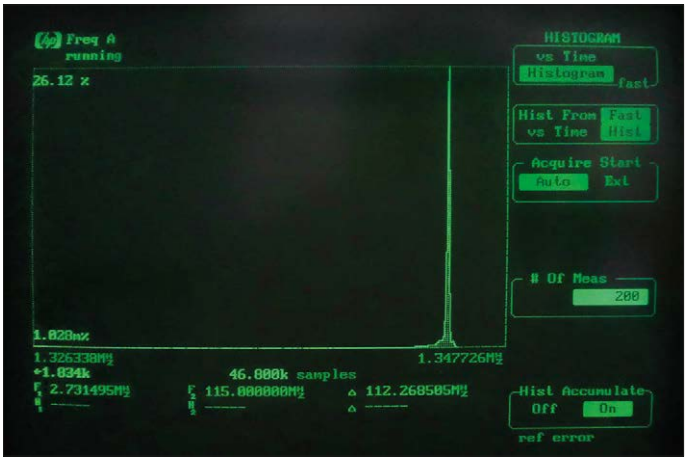


Fig. 7. Même la stabilité n'offre guère de prise à la critique.

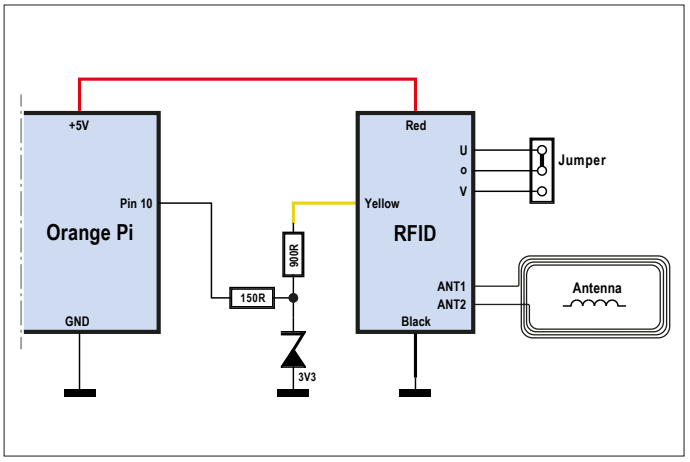


Fig. 8. Un adaptateur de niveau de tension est absolument indispensable.

Mouser pour douze dollars environ, y compris son antenne, est tout à fait ce qu'il nous faut pour nos premiers pas dans le monde de la technologie 125 kHz, vu la modestie de l'environnement matériel nécessaire. Du point de vue électronique, la **figure 8** montre tout ce dont on a besoin : alors que le RPi tolère le 5 V jusqu'à un certain point, l'Orange Pi exige dans tous les cas une adaptation de niveau (pas très compliquée).

Pour la configuration matérielle de ses systèmes sur une puce (SoC), Allwinner utilise un fichier binaire chargé par le noyau au cours du démarrage et qui, outre les régulateurs de tension et les horloges, sert à configurer divers autres sous-systèmes. L'UART utilisé par notre programme de démo n'est normalement pas accessible comme le montre l'examen de l'arborescence des périphériques (sous `/dev`), qui n'affiche pour l'instant qu'un seul UART (`ttyS0`) :

```
$ ls -l

. . .

crw--w---- 1 root tty      4,   9 Apr 12 06:07 tty9
crw--w---- 1 root tty    251,   0 Apr 12 06:08 ttyS0
crw------- 1 root root   248,   0 Apr 12 06:07 tv
```

Pour régler ce problème, il faut commencer par modifier le fichier de configuration lu par le matériel au cours de la phase de démarrage. Les scripts utilisés par les différents modèles d'ordinateurs se trouvent sur la carte SD dans le répertoire `/boot/bin`. Pour Armbian, on a le contenu suivant :

Le fichier réellement utilisé pour le démarrage se trouve en fait un niveau plus haut et s'appelle `script.bin`. Mais il s'agit là d'un lien logique, traduit par la commande `realpath` :

```
$ pwd
/boot

$ realpath script.bin
/boot/bin/orangepipcplus.bin
```

Les fichiers de configuration correspondants peuvent exister sous deux formes : outre la version binaire utilisée par l'ordinateur, il y a le fichier `.fex`, lisible par l'homme.

Pour convertir le fichier binaire, ici nommé `/boot/bin/orangepipcplus.bin`, comme on vient juste de le découvrir, on utilise la commande suivante :

```
$ sudo bin2fex orangepipcplus.bin orangepipcplus.fex
[sudo] password for spotter: . . .
fexc-bin: orangepipcplus.bin: version: 1.2
fexc-bin: orangepipcplus.bin: size: 36392 (84
   sections), header value: 36392
```

La structure des fichiers `.fex` rappelle celle des fichiers `.ini`, qui est documentée en détail sous [5]. Pour l'instant, le seul passage qui nous intéresse est celui qui configure l'UART associé au lecteur RFID. Éditez ce paragraphe pour le rendre identique à ce qui suit :

```
$ ls
aw-som-a20.bin          bananapiprolcd7.bin  lime-a10-lcd.bin  nanopiair.bin       orangepilite.bin    pcduino2.bin
bananapi.bin            beelinkx2.bin        lime-a10.bin      nanopim1.bin        orangepione.bin     pcduino3.bin
bananapilcd7.bin        cubieboard.bin       lime-a33.bin      nanopim1plus.bin    orangepipc.bin      pcduino3nano.bin
bananapim1plus.bin      cubieboard2.bin      lime.bin          nanopineo.bin       orangepipcplus.bin
bananapim1pluslcd7.bin  cubieboard2dual.bin  lime2-emmc.bin    olinux-som-a13.bin  orangepiplus.bin
bananapim2plus.bin      cubietruck.bin       lime2.bin         orangepi.bin        orangepiplus2e.bin
bananapipro.bin         lamobo-r1.bin        micro.bin         orangepi2.bin       orangepizero.bin
```

Listage 1. Lecture d'un badge au moyen de l'UART.

```
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <wiringPi.h>
#include <wiringSerial.h>

int main ()
{
  int fd ;
  int count ;
  unsigned int nextTime ;

  if ((fd = serialOpen ("/dev/ttyS3", 9600)) < 0)
  {
    fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
    return 1 ;
  }

  if (wiringPiSetup () == -1)
  {
    fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
    return 1 ;
  }

  nextTime = millis () + 300 ;

  for (count = 0 ; count < 256 ; )
  {
    delay (3) ;

    while (serialDataAvail (fd))
    {
      printf (" -> %3d", serialGetchar (fd)) ;
      fflush (stdout) ;
    }
  }

  printf ("\n") ;
  return 0 ;
}
```

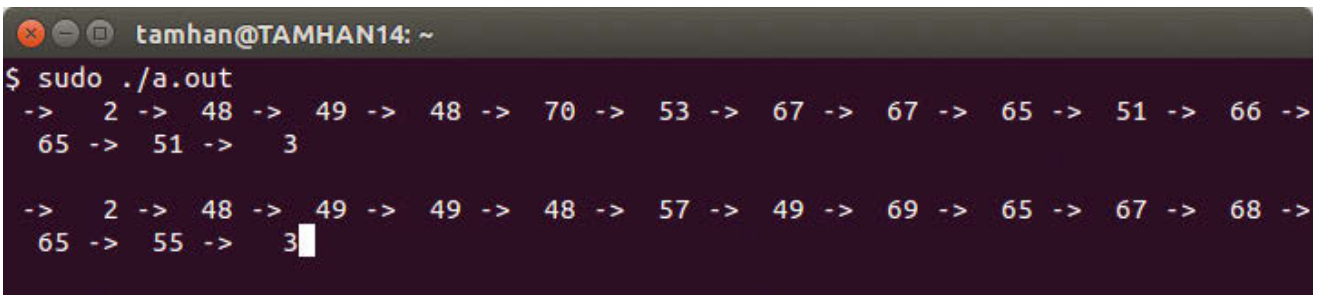


Fig. 9. Deux badges différents ont été reconnus correctement. Après chaque processus de lecture, appuyez sur Entrée pour retourner en début de ligne de commande.

```
[uart3]
uart_used = 1
uart_port = 3
uart_type = 4
uart_tx = port:PA13<3><1><default><default>
uart_rx = port:PA14<3><1><default><default>
uart_rts = port:PA15<3><1><default><default>
uart_cts = port:PA16<3><1><default><default>
```

À titre de comparaison, voici le code qui décrit un UART désactivé :

```
[uart2]
uart_used = 0
uart_port = 2
. . .
```

Le seul enregistrement des modifications du fichier `.fex` ne change en rien le comportement de l'ordinateur, même après un redémarrage obligatoire. Encore faut-il reconvertir le fichier `.fex` en un fichier `.bin`, ce qui s'effectue au moyen de la commande `fex2bin` :

```
$ sudo fex2bin orangepipcplus.fex orangepipcplus.
   bin
```

Après un redémarrage, le nouveau port série apparaît dans l'arborescence des périphériques sous `/dev` :

```
crw--w---- 1 root tty     251,   0 Apr 24 13:34
   ttyS0
crw-rw---- 1 root dialout 251,   3 Apr 24 13:34
   ttyS3
crw------- 1 root root    248,   0 Apr 24 13:34 tv
```

Lecture des badges RFID

L'activation et la configuration d'un UART ne représente qu'une petite partie des options. Un problème classique est le choix de la fréquence de l'horloge pour obtenir un compromis satisfaisant entre la puissance de calcul et la consommation électrique. On trouvera des compléments d'information sur les forums ; occupons-nous de la lecture des données RFID.

Au lecteur de badge, il faut associer un programme qui récupère les données sérielles fournies par l'UART. La première tâche de ce programme est d'ouvrir un accès vers l'UART, voir la première partie du **listage 1** [6].

Ensuite, il faut lire les informations reçues. Étant donné l'importance croissante du RFID, l'auteur propose une petite digression. Il existe deux types de lecteurs. Le premier, auquel appartient notre lecteur, lit les informations du badge et les envoie en flux sériel continu vers le destinataire.

Le second groupe est basé sur le protocole Wiegand qui utilise deux lignes de données pour transmettre des impulsions. Ce protocole fait l'affaire des vieux routiers de l'informatique, car, pour des raisons historiques, il est limité à 26, voire 34 bits (!) et est increvable. La plupart des lecteurs RF sont proposés pour l'un et l'autre protocole, mais l'auteur a une préférence pour le modèle UART.

Attention : collisions !

Quand plusieurs programmes accèdent simultanément au même bus sériel, il peut se produire des collisions. D'expérience, on sait qu'elles ne sont pas détectées par le système d'exploitation : les clients récupèrent alors une « salade de données ».

Pour des raisons de commodité, notre programme (**listage 1**) lit les octets un à un pour les afficher sur la ligne de commande. Tout est alors prêt pour un premier test. Connectez le lecteur de badge au nano-ordinateur, compilez le programme comme d'habitude et présentez un badge devant la bobine rouge. L'affichage sur la ligne de commande devrait ressembler à ce que montre la **figure 9**.

En faire plus

Comme on a pu le constater, l'Orange Pi est un ordinateur à base de Linux. Il en découle que les divers protocoles, tels que I²C et SPI se comportent à peu près comme on en a l'habitude avec le RPi et compagnie.

En examinant de plus près les interfaces de programmation (API) de l'I²C ou du SPI (ou en utilisant la bibliothèque *WiringOP*), on pourra facilement connecter des périphériques divers. Outre les afficheurs, les capteurs les plus variés sont disponibles.

Conclusion

Quand le premier Orange Pi a vu le jour, beaucoup doutèrent des chances de survie de l'entreprise. Trois ans plus tard, Shenzhen Xunlong est toujours en activité et a même mis sur le marché bon nombre de nouveaux modèles. De plus, pratiquement tous les modules proposés jusqu'ici sont toujours disponibles.

L'utilisation d'un Orange Pi s'avère particulièrement payante dans le cas d'une production en série : une diminution du coût de 50% n'est certes pas négligeable. Pour la production à l'unité se pose le problème des coûts de développement ; si l'on n'est pas un expert Linux, on sera plus à l'aise avec le RPi, car sa communauté d'utilisateurs est bien plus vaste. ⏮

(160456 – version française : Helmut Müller)

Liens

[1] www.orangepi.org

[2] www.youtube.com/watch?v=4zICevDOhr8

[3] www.orangepi.org/downloadresources/

[4] www.armbian.com/download

[4B] https://dl.armbian.com/orangepilite/

[5] http://linux-sunxi.org/Fex_Guide

[6] www.elektormagazine.fr/160456

Rejoignez la communauté Elektor

Devenez membre GOLD maintenant !

GOLD 2,45 € / semaine

- ✓ accès à l'archive d'Elektor
- ✓ 10% de remise dans l'e-choppe
- ✓ 6x magazine imprimé
- ✓ 6x magazine numérique
- ✓ des offres exclusives
- ✓ accès à plus de 1 000 fichiers Gerber
- ✓ le DVD annuel d'Elektor

Également disponible

abonnement « zéro papier » GREEN !

GREEN 1,78 € / semaine

- ✓ accès à l'archive d'Elektor
- ✓ 10% de remise dans l'e-choppe
- ✓ 6x magazine numérique
- ✓ des offres exclusives
- ✓ accès à plus de 1 000 fichiers Gerber

www.elektor.fr/membres