Capacimètre pour batteries
de fortes capacités jusqu'à 50Ah.

Par Nulentout : Mercredi 30 Août 2017.

Contrairement à mes didacticiels actuels où la minuscule carte NANO Arduino est privilégiée pour concevoir de tout petits appareils électroniques "de loisir", dans cette application c'est une bonne vieille référence Arduino UNO qui sera intégrée dans le projet décrit. Cette dernière a déjà servi pour mettre au point une foule de petites applications variées servant d'exemples simples pour utiliser des modules du commerce que vous pouvez aller consulter sur https://onedrive.live.com/?authkey=%21AAQqqyaHRuisCEI&id=B2A7F8B18C2B4F0E%212352&cid=B2A7F8B18C2B4F0E et tout particulièrement la page 20 relative au SHIELD pour afficheur LCD utilisé sur cet appareil. Autant dire que son ATmega328 a déjà été programmé des centaines de fois prouvant la fiabilité de ce microcontrôleur. Avant de poursuivre, **à quoi sert concrètement ce type d'appareil de mesure ?**

Nombreuses sont les applications pour lesquelles nous faisons l'achat d'un "Pack énergie 12V" réputé servir à démarrer le moteur thermique d'une automobile dont la batterie de bord vient de rendre l'âme. Naturellement, c'est l'usage de base d'un tel produit commercial tel que celui de la Fig.1 qui pour ma part sert les belles nuit d'été à l'alimentation en 12Vcc d'un petit télescope motorisé pour le pointage automatique. C'est du reste la raison pour laquelle ces modules énergétiques sont pourvus de prises **2** de type "allume cigare". Ils sont utilisés en camping pour l'éclairage de la tente, le fonctionnement de la radio et bien d'autres choses encore. Du reste, c'est souvent pour des emplois de ce type qu'ils sont acquis, bien plus que pour les grosses pinces crocodile **1**.

Arrive souvent le cas où il vous semble que le gros module ne tient plus la charge. Il peut s'agir également de la batterie de votre motocyclette, cette de votre berline ou pourquoi pas celle du tracteur agricole si indispensable à la ferme. Dans tous les cas, il vaut mieux pouvoir à l'avance savoir à quel moment il est devenu temps de changer la batterie, c'est à dire avant de "tomber en carafe". Il est également bien utile à mon sens de mesurer la performance d'une batterie ou d'un "pack alimentation" tout neuf dont on vient de faire l'acquisition pour s'assurer que son aptitude à emmagasiner de l'énergie correspond bien à la caractéristique annoncée par le fournisseur.

C'est ici que l'appareil décrit dans ces lignes va apporter une réponse objective. Le principe d'utilisation restant élémentaire. On commence par effectuer une recharge complète de l'individu qui va être testé. Étant "gavé" à pleine charge de ce qu'il peut encore emmagasiner, on va procéder à sa décharge complète et mesurer tout ce qu'il restitue. Si l'énergie rendue dépasse les deux tiers de la capacité théorique, on peut considérer que l'élément est encore en bonne santé. Si à peine un quart de ce que prévoit la théorie est rendu, alors il est temps d'envisager le remplacement, car la batterie en question peut nous abandonner sans préavis. Vous comprenez qu'il ne s'agit pas de mathématiques, c'est l'ordre de grandeur par rapport à la valeur théorique du produit à l'état neuf **3** qui compte, inutile de pousser à vingt chiffres significatifs la finesse du mesurage. Ceci étant précisé, et uniquement pour le plaisir de l'analyse et de la programmation, on va dans cette description rechercher une précision très exagérée au regard de l'application ... juste pour le plaisir.

Fig.1

Pour des raisons d'équité commerciale, la photographie a été retouchée pour effacer la marque et la référence de ce produit.

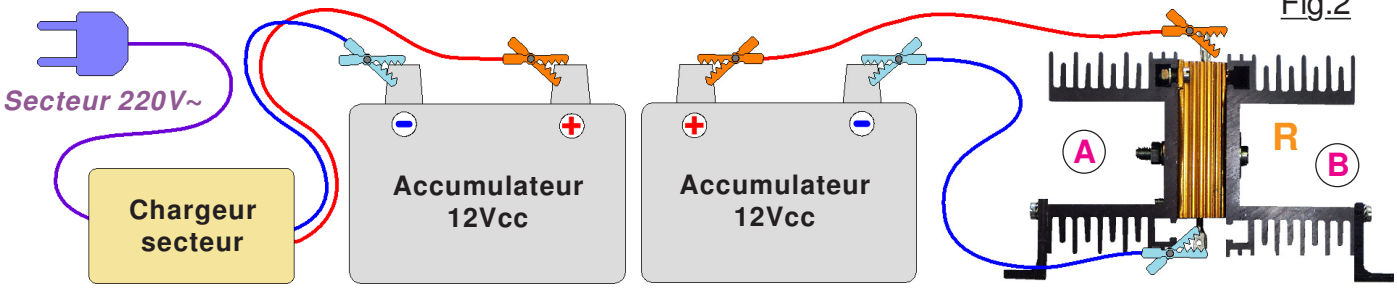
ATTENTION : Le principe de mesurage de la capacité d'un accumulateur consiste à effectuer un cycle Charge complète / Décharge complète. Hors, même s'il est considérable, **le nombre de cycle pour une batterie reste limité**. Il ne faut donc effectuer cette manipulation que de façon ponctuelle, soit quand le produit est neuf pour en vérifier les caractéristiques réelles, soit pour la batterie de votre véhicule quand vous avez un doute sérieux. ***Surtout, il importe de recharger entièrement cette dernière sans tarder après le test, un accumulateur ne devant jamais rester déchargé sur une longue période.***

Danger : Si vous observez la Fig.1 en **4** vous pouvez constater que sur un court-circuit la batterie peut fournir jusqu'à 600A ! L'étincelle qui se produit peut faire fondre en partie le métal des pinces, avec projection de minuscules perles en fusion vers les yeux. ***Pour les manipulations de branchement protégez votre vue avec des lunettes ...***

01) Principe de mesurage de la capacité d'une batterie :

Que ce soit une grosse batterie au plomb pour démarrer le récalcitrant V8 de notre gros 4x4 urbain utilisé pour faire les courses au supermarché, ou la minuscule petite pile AAA que l'on va introduire dans la minuterie permettant de cuire correctement les œufs durs à la cuisine, dans tous les cas la **CAPACITÉ** précise par définition le "courant total" que pourra fournir le dispositif en cours d'évaluation. Cette entité se mesure en **Ampères Heures**. *(Ou les sous-multiples.)* Seule différence en fonction de la nature de l'élément mesuré, il faut ***adapter l'intensité qui sera maintenue durant la décharge***, cette dernière devant rester raisonnable sans pour autant conduire à un temps de vérification exagéré. Prenons un exemple : Sur le corps du module de la Fig.1 le fournisseur précise que son produit neuf présente une caractéristique de **22Ah**. On en déduit que si l'appareil utilisateur consomme en permanence 10A, la durée de fonctionnement sera de 22 / 10 soit environ 2,2 heures avant d'avoir à effectuer une recharge. Par contre, si la prise "allume cigare" ne débite que 500mA, la décharge totale va exiger 22 / 0,5 = 44 heures. Si vous laissez le bouton sur marche, la LED verte de fonctionnement restera allumée consommant ses 10mA en permanence. Elle videra toute l'énergie emmagasinée en 22 / 0,01 = 2200h soit presque 92 jours ...

Fig.2



1) Recharge complète de l'élément à tester. *2) Décharge complète dans une résistance de "charge".*

Principe du mesurage de la capacité d'un accumulateur.

Comment déterminer l'intensité "raisonnable" ?

Pour être significative, la décharge complète *(Qui se fait à courant pratiquement constant)* ne doit pas s'avérer trop brutale. Par exemple, on décharge à 600A *(Théoriquement possible puisqu'inscrit en **4** de la Fig.1 sur le produit commercial.)* ce qui exige dans ces conditions une durée de 22 / 600 = 0,037 heures. Ou si vous préférez environ deux minutes. La mesure qui en résulterait ne serait pas précise, car de courte durée et à un débit n'ayant rien à voir avec l'usage habituel. Par ailleurs, si la décharge est trop "gentille", la mesure va exiger une durée exagérée. Pour pouvoir tester toutes nos batteries, 4A à 5A me semblent aussi bien compatible avec les batteries de motocyclettes que pour de gros éléments pouvant aller jusqu'à 50AH. Si vous désirez de plus fortes capacités, il suffit d'ajoutez à convenance des résistances de décharge en parallèles de celles de cette description. *(Et revoir la constante définissant **R** dans le logiciel.)* La Fig.2 résume le principe de la mesure. On commence par recharger à saturation l'individu à tester. Puis, mesurant le débit qu'il fournit dans une résistance de charge **R**, on le déleste entièrement de l'énergie emmagasinée.

*(Résistance de charge **R** au sens de l'exigence d'un effort électrique. Vocable habituel oxymore ici, puisque cette charge électrique engendre ... une décharge de l'échantillon testé !)*

02) Chaleur tropicale ! :

Tributaire des résistances de puissance disponible dans le commerce, l'idée consiste à répartir l'effort thermique dans plusieurs éléments que l'on calculera "large" pour des raisons de fiabilité. N'oublions pas que l'énergie consommée sur la batterie pour la décharger sera transformée en chaleur par effet Joules. Plusieurs approches sont possibles. On peut employer un grand nombre d'éléments de faibles puissances, ou un petit groupe mais de composants plus musclés. Le choix final résulte d'un compromis ***Volume du bloc thermique / Disponibilité des composants*** dans le commerce en ligne. Le choix final c'est porté sur un groupe de trois éléments de 8Ω qui mis en parallèle vont aboutir à une résistance théorique de 2,67Ω environ. Vous pouvez observer sur la Fig.3 que ces références sont conçues pour pouvoir dissiper chacune 50W, nous allons voir qu'elles sont calculées très très large. Réputés précis à 5% ces composants sont parfaits pour notre application. ***Comme toutes les résistances de ce type qui fondamentalement sont prévues pour dissiper de la chaleur, il importe de rayonner le plus rapidement possible cette dernière dans l'environnement immédiat***. La première méthode consiste à immobiliser ces éléments sur un radiateur. *(Encore un oxymore puisque le but de ce composant consiste à refroidir !)* Sur la Fig.3 seule la moitié **A** du bloc thermique est photographiée pour montrer les résistances. Pour améliorer la jonction thermique entre **R** et **A**, de la graisse silicone spéciale a été déposée sous la semelle des belles résistances oranges. Quand les boulons d'immobilisation ont été serrés, la graisse thermique à été chassée, on peut en observer un résidu "Bave d'escargot" sur la photographie.

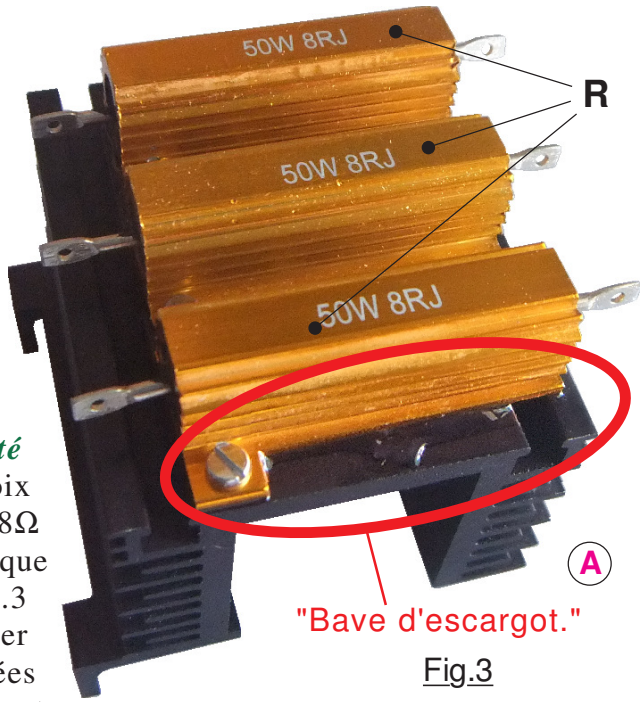


Fig.3

Calculer la puissance qui sera convertie en chaleur reste élémentaire, il suffit d'appliquer la fidèle loi d'Ohm bien connue : I = U / R suivie de P = U x I. Des mesures précises ont montré qu'une fois réunies par du gros fil électrique et alimentées par deux fiches bananes, la résistance globale avoisine 2,7Ω ce qui confirme la précision des composants approvisionnés. La valeur est légèrement plus élevée que celle du calcul théorique, mais il faut compter sur la résistance des fils et surtout celle dans les deux prises pour fiches bananes. Dans ces conditions, I = 12 / 2,7 soit approximativement 4,44A. Compte tenu des choix effectués, nous obtenons bien l'intensité souhaitée comprise entre 4A et 5A. La puissance qui va se dégager et transformée en chaleur par **R** sera donc de 4,44 x 12 = 53,3W pour une tension nominale aux bornes de la batterie. Ça va chauffer dur dur !

03) Ventiler s'avère indispensable :

Visiblement, la puissance que pourrait absorber le groupe de trois éléments est presque trois fois supérieure. Il se trouve que des composants plus modestes sont commercialisés pratiquement à des tarifs identiques. Surdimensionner va donc dans le sens de la fiabilité. Pour le prototype décrit, le choix du radiateur de récupération c'est porté sur un modèle qui autorise le serrage en sandwich particulièrement propice au refroidissement. Inutile de préciser qu'avant de brider **A** et **B** sur **R**, une couche de graisse silicone a également été étalée sur le dessus des résistances. Pour les dimensions, un composant "moyen" a été sélectionné pour ne pas conduire à un appareil trop volumineux. Le bloc complet présente malgré tout un ambonpoint respectable.

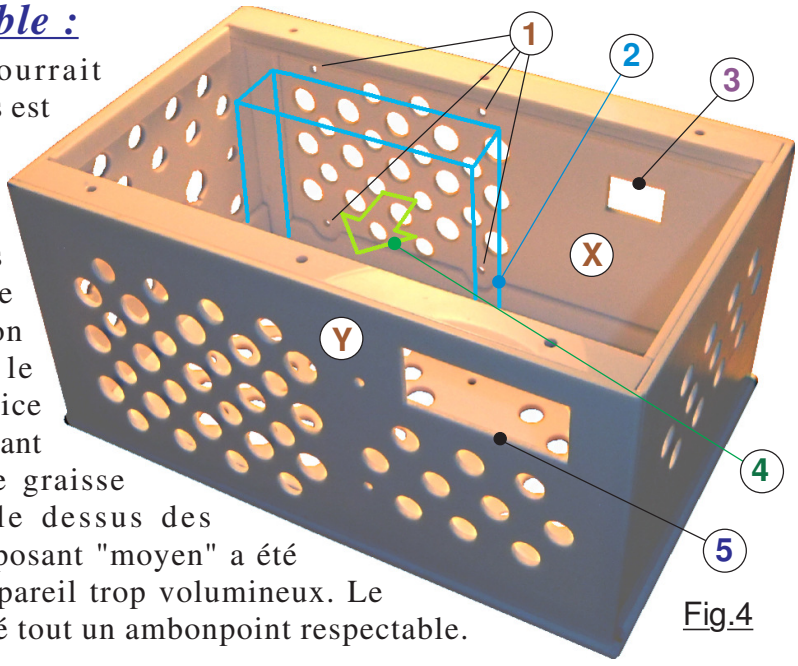


Fig.4

Il n'en reste pas moins vrai que chauffant à 54W durant une période notable, vous ne laisserez pas les mains sur ses ailettes de refroidissement. La chaleur dégagée doit impérativement être extraite de l'appareil et chassée à l'extérieur. C'est la raison pour laquelle le coffret sera percé d'une multitude de trous d'aération et surtout l'air chaud sera chassé à l'extérieur, ***poussé efficacement par un ventilateur***. La Fig.4 présente le coffret avant intégration des composants internes. Le ventilateur est un modèle 12V prévu pour les ordinateurs. Son corps est carré de 80mm x 80mm de coté pour une épaisseur de 25mm. Ses trous de fixation sont visibles en **1** sur l'image. La photographie est surchargée en **2** pour symboliser l'emplacement de ce radiateur qui se trouve à 12mm de la face **X** séparé par des entretoises. La flèche verte **4** indique le sens de soufflage du ventilateur, aspirant l'air frais à traves les orifices de **X** et chassant l'air chauffé par **A** et **B** à travers les trous d'aération pratiqués en face sur **Y**. Le couvercle étant lui-même percé de nombreux orifices et le flux du ventilateur étant confiné par un déflecteur visible sur la Fig.6 dont il sera question plus avant, la température du radiateur mesurée coté **A** se stabilise à environ 38°C, valeur tout à fait raisonnable.

04) Schéma électrique et électronique :

Électronique car il y a la présence de la carte Arduino UNO. Pour le reste, il s'agit d'électricité assez élémentaire. La Fig.5 présente l'ensemble de ce qui sera intégré dans notre appareil de mesures. Un minimum de commentaires s'impose pour en dégager les points particuliers. Le dessin décrit globalement l'agencement électrique, et fait de surcroit apparaitre les éléments de connectiques qui jouent un rôle important. En effet, le circuit de puissance, celui qui sera parcouru par un courant important, doit présenter des résistances de câblage faibles. Donc du gros fil, et surtout des connecteurs sérieux. Par exemple en **X** et **Y** on trouve les jonctions sur cosses dont il sera question plus loin. Comme nous allons le constater, le taux d'intégration est particulièrement élevé. Il faut pourtant permettre d'extraire et de déposer n'importe quel élément en vue d'une maintenance

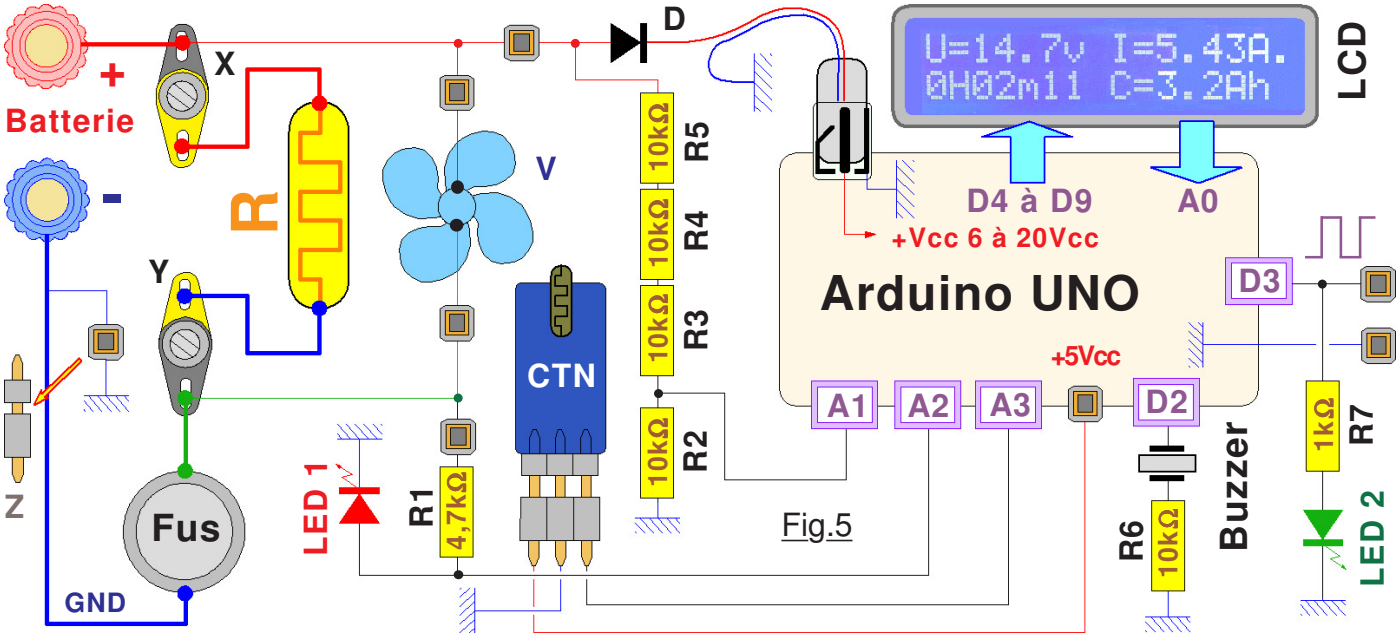


Fig.5

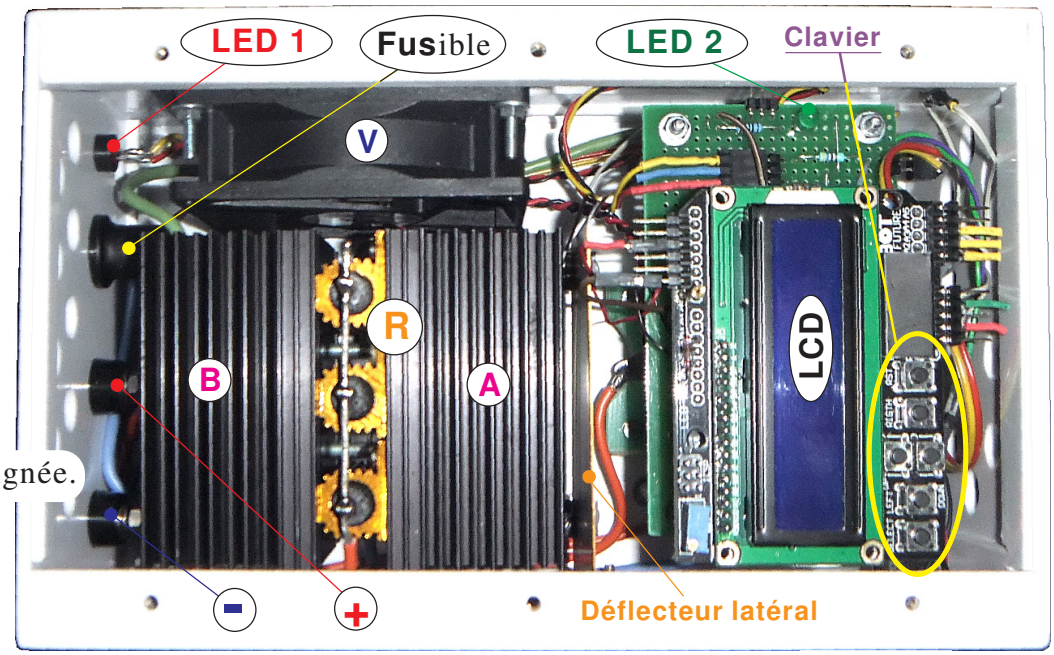
éventuelle. Du reste, avant d'envisager de déposer, il faut déjà ... pouvoir intégrer dans le coffret et effectuer les raccordements électriques. Toute la circuiterie faible puissance, celle qui intègre la carte Arduino UNO et l'afficheur **LCD** est branchée ou isolée par des connecteurs HE14 symbolisés par . Dans la pratique ces prises sont constituées d'une paire HE14 mâle et femelle telle que celle représentée en **Z** à une ou plusieurs lignes. La batterie est branchée sur la grosse borne pour fiche banane rouge. Ensuite le courant va sur le groupe de résistances **R** via le bornier **X**. Puis cheminant à travers les cosses **Y** le courant traverse le **Fus**ible pour retourner à la batterie par la deuxième borne bleue.

Maintenant que tout est en place pour procéder à la décharge de la batterie, il importe d'effectuer le mesurage et d'établir le "dialogue Homme/Machine. C'est la carte Arduino UNO qui sera chargée d'effectuer les mesures, les conversions analogiques numériques ainsi que les calculs. Elle gèrera l'afficheur à cristaux liquides **LCD** pour fournir les informations à l'opérateur. Le SHIELD **LCD** du commerce comporte un clavier à cinq boutons poussoir largement suffisant pour permettre à l'opérateur de piloter le processus. Notez au passage que le ventilateur **V** est placé en

parallèle de **R** et peut être séparé en débranchant son connecteur HE14. Cette faculté permet entre autre de mesurer sa consommation alors que l'appareil est entièrement assemblé. Comme il n'est jamais exclus de brancher la batterie à tester à l'envers, autant **R** et **V** ne nous en tiendraient pas rigueur, autant Arduino et **LCD** pourraient se détruire, le régulateur 5Vcc intégré en premier, l'afficheur ensuite le courant négatif cheminant dans la circuiterie. Aussi, la diode **D** élimine définitivement le risque de dégradation par inversion de polarité. La sortie **D2** pilote un petit **Buzzer** qui sera chargé de prévenir l'opérateur quand le mesurage sera terminé, ou d'alerter en cas de surchauffe du bloc thermique. Comme le petit transducteur fait beaucoup de bruit, il est "calmé" par la résistance **R6** de 10kΩ. En **D3** l'ATmega328 inverse l'état de la **LED 2** toutes les secondes. Cette dernière témoigne du fonctionnement normal du programme. Surtout, le connecteur HE14 auquel on accède par l'ouverture **3** de la Fig.4 permet de mesurer avec une grande précision la durée de la boucle de base qui influence celle des mesures. Le courant dans **LED 2** est limité par **R7** de 1kΩ.

Pour calculer le courant qui est fourni par la batterie, il faut déterminer la tension aux bornes de **R**. Dans ce but, la tension d'entrée est divisée par quatre par le groupe de résistances **R2** à **R5** car sur **A1** il ne faut pas dépasser 5Vcc. Il serait logique de remplacer **R3** à **R5** par une résistance unique de 30kΩ, mais cette valeur n'est pas très courante. Aussi, comme personnellement je commande des 10kΩ par paquet de trente, je peux en utiliser deux de plus, et ce d'autant plus que dans ce montage la place ne manque pas. Pour ne pas que la mesure ne soit perturbée par la tension de conduction aux bornes de **D**, le pont diviseur est branché en amont. Une inversion de tension n'est plus interdite, mais avec 30kΩ en série, l'ATmega328 ne risque strictement rien. On constate que sur la Fig.5 le "point chaud" du **Fus**ible est relié à l'entrée **A2** par l'intermédiaire d'une résistance de 4,7kΩ. On pourra ainsi déduire la chute de tension aux bornes du **Fus**ible qui se retranche à celle de la batterie pour calculer exactement la tension aux jonctions de **R**. *(Ici franchement c'est du pinaillage, mais comme le microcontrôleur n'est pas surchargé, autant lui éviter de s'ennuyer !)* Plus utile, si on enlève le fusible ou si ce dernier se détruit pour une quelconque raison, la **LED 1** s'allumera prévenant optiquement l'opérateur. La tension qui au nominal ne dépasse pas 0,13V sur le **Fus**ible monte alors entre 1,8V et 1,9V lorsque la tension batterie varie entre 5V et 20V. Le programme est alors informé de la non présence d'un **Fus**ible et peut prévenir sur le **Buzzer**.

Comme on ne manque pas d'entrées disponibles et que la taille du programme le permet, un petit module équipé d'une thermistance de type **CTN** est ajouté à l'ensemble pour mesurer en permanence la température du bloc thermique. En fonctionnement normal stabilisé, la température avoisine les 38°C. Supposons que pour une raison quelconque le ventilateur ne fonctionne plus. Des mesures *(Hors coffret pour ne pas abimer ce dernier.)* ont montré que la température peut augmenter jusqu'à 94°C. Bien avant d'atteindre cette valeur inacceptable, par exemple à 45°C, on déclenchera une alerte sonore "assez agassive" pour attirer rapidement l'attention, l'écran **LCD** précisant la raison de ce tapage. Les subtilités de ce montage globalement simple étant assimilées, avant de passer à l'aspect programmation, nous allons concrétiser ce schéma et aborder l'aspect matériel. La volonté d'aboutir à un coffret de dimensions raisonnable sera une pierre d'achoppement avec chausse-trappes et pièges à la clef. L'étude d'implantation devra être particulièrement soignée.



L'appareil terminé.

Fig.6

05) Réalisation du bloc thermique :

Trouver dans le commerce en ligne un radiateur identique à celui qui équipe le prototype relève d'une mission impossible. Il a été trié dans un gros carton contenant une foule de dissipateurs thermiques de récupérations multiples effectuées sur des ensembles assez anciens ... donc introuvables. Il vous faudra commander un élément assez volumineux pour pouvoir recevoir les trois résistances de puissance formant le groupe R. De ce fait votre bloc de consommation électrique sera forcément de volume différent, avec un impact direct sur les dimensions du coffret. Toutefois, pour vous aider je vous propose les dessins précis et à l'échelle du boitier. De toute façon vous aurez à modifier le produit approvisionné, car les radiateurs du commerce sont généralement conçus pour recevoir un ou deux composants d'encapsulage de type TO3. Hors l'entraxe des deux pattes de fixation des résistances, qui sont placées en diagonales sur leurs semelles, est bien plus grand que celui d'un boitier de type TO3. Percer des trous au diamètre ϕ M3 sera probablement incontournable. Pas de panique, les dissipateurs thermiques du commerce sont pratiquement tous composés d'un alliage d'aluminium extrudé pour former les ailettes de refroidissement, car pour la convection c'est la surface avec l'air ambiant qui est importante. Ce matériau relativement mou accepte d'être percé avec une petite chignole d'électronicien pour peu que l'on ne dépasse pas les 4,5mm de diamètre. Pour cette dimension il vaut mieux faire un avant-trou de diamètre 2,5mm. Sans que ce soit spécialement aisé pour le commun des mortels, c'est tout à fait faisable.

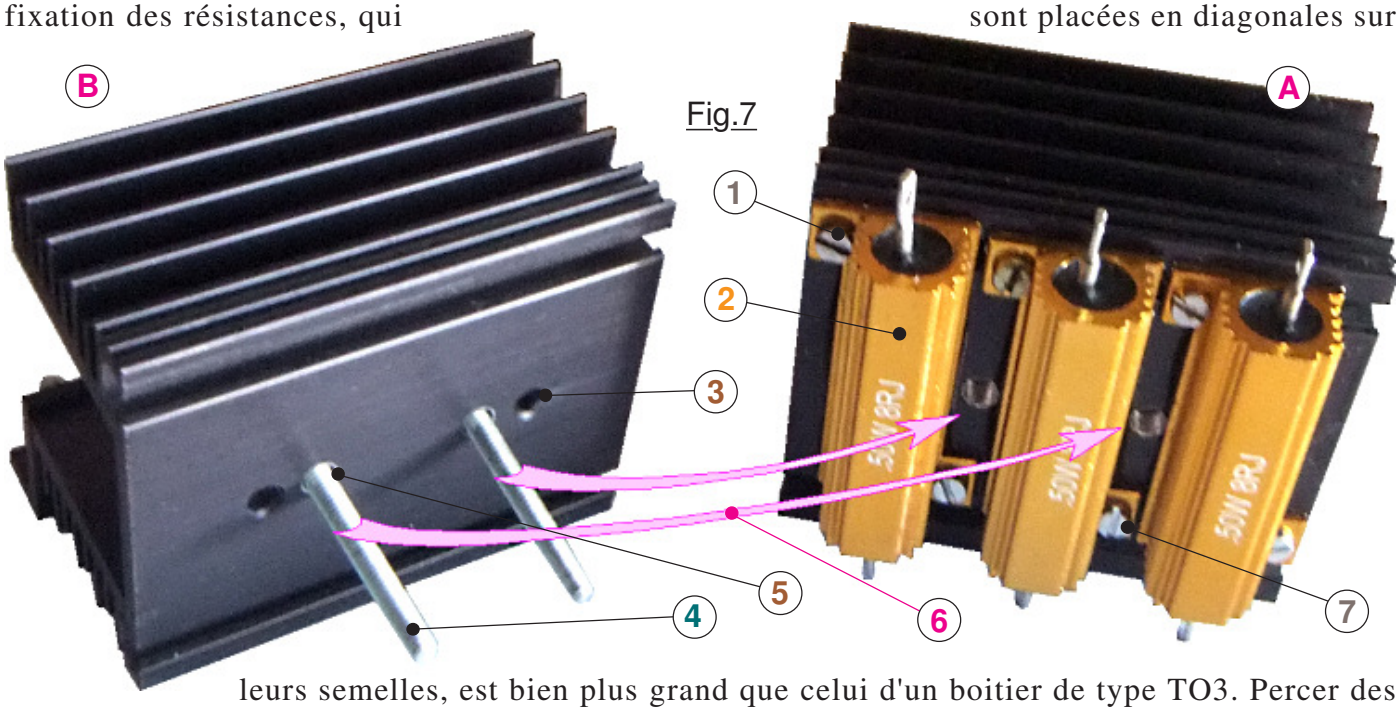


Fig.7

À titre d'exemple je vais détailler les opérations chirurgicales pratiquées sur le radiateur de ce didacticiel, car bien que différent du vôtre il pourra vous donner des idées pour les usinages à effectuer. Critère prioritaire quand vous choisirez votre composant : Présenter une surface plane assez grande pour recevoir les trois résistances 2, avec sur l'autre coté un dégagement qui accepte la présence des écrous de liaison.

La Fig.7 présente les deux blocs modifiés juste avant l'assemblage final. Les résistances 2 sont immobilisées en 1 par un boulon ϕ M3 dont l'écrou 8 et la rondelle d'appui sont glissée dans la rainure du demi bloc A. L'autre patte est immobilisée par les vis 7 dans un trou directement taraudé à ϕ M3 dans le bloc d'aluminium. En 3 on observe les trous d'origine qui permettaient d'assembler A sur B avec les boulons ϕ M4 repérés 4 sur le dessin. En répartissant les trois résistances 2 sur la surface, ces deux trous sont masqués. C'est la raison pour laquelle deux autres orifices de traversée ont été pratiqués en 5 à ϕ 4,5mm entre les trois résistances. Quand elles sont immobilisées sur A, on enduit la surface de contact extérieure avec la graisse thermique au silicone, et l'on assemble le total par l'introduction 6 des vis 4 à travers A.

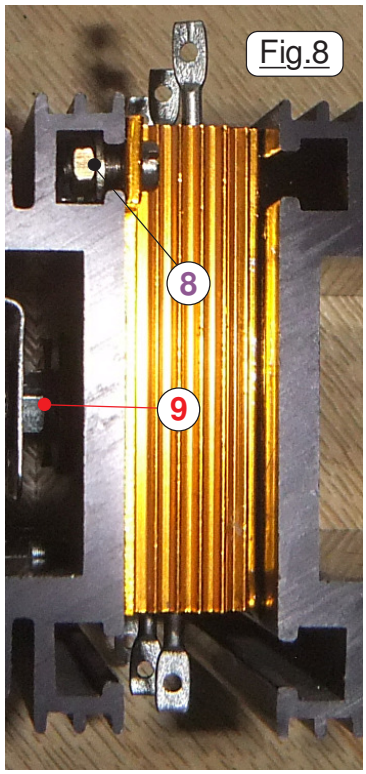


Fig.8

Finalement, la modification à apporter au radiateur sélectionné et résumée sur la Fig.9 *(Qui reprend les repères de la Fig.7)* se résume à peu de chose. Elle consiste à percer les deux trous **5** de ϕ 4,5mm sur **A**, et à tarauder à ϕ M3 les trois trous filetés **10** pour implanter les vis **7**. Cette deuxième opération est particulière, car tout le monde ne dispose pas d'un jeu de tarauds ϕ M3. Il vous faudra par conséquences choisir un radiateur dont l'usinage à effectuer soit dans vos possibilités. Outre le perçage et le taraudage des orifices

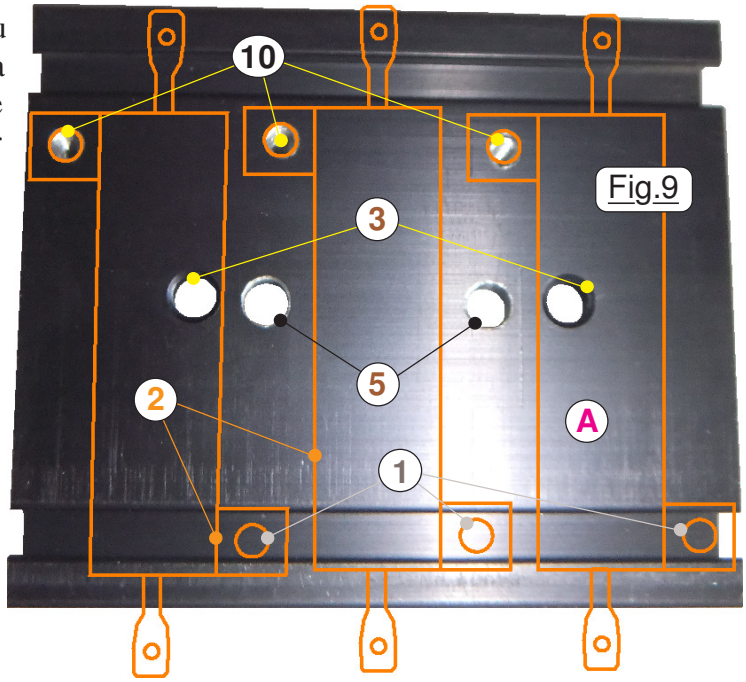


Fig.9

pour les vis de liaison, on peut avoir quelques façonnages à effectuer sur l'accastillage. Dans notre cas, les rondelles **11** sont un peu trop larges pour pouvoir être insérées dans la rainure du bloc **A**. Un peu le limage comme montré dans l'encadré **E** sera suffisant pour résoudre ce petit problème. La Fig.10 montre en gros plan la configuration de cette rondelle avant l'assemblage de **2** sur **A**. Inutile de serrer fortement les petits boulons ϕ M3 car il suffit d'établir un contact thermique entre le

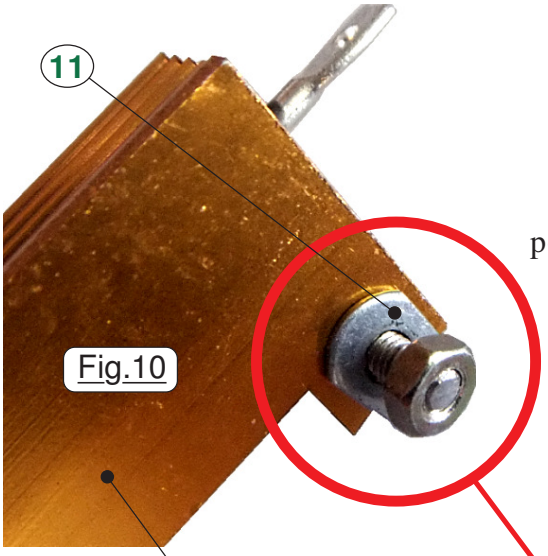
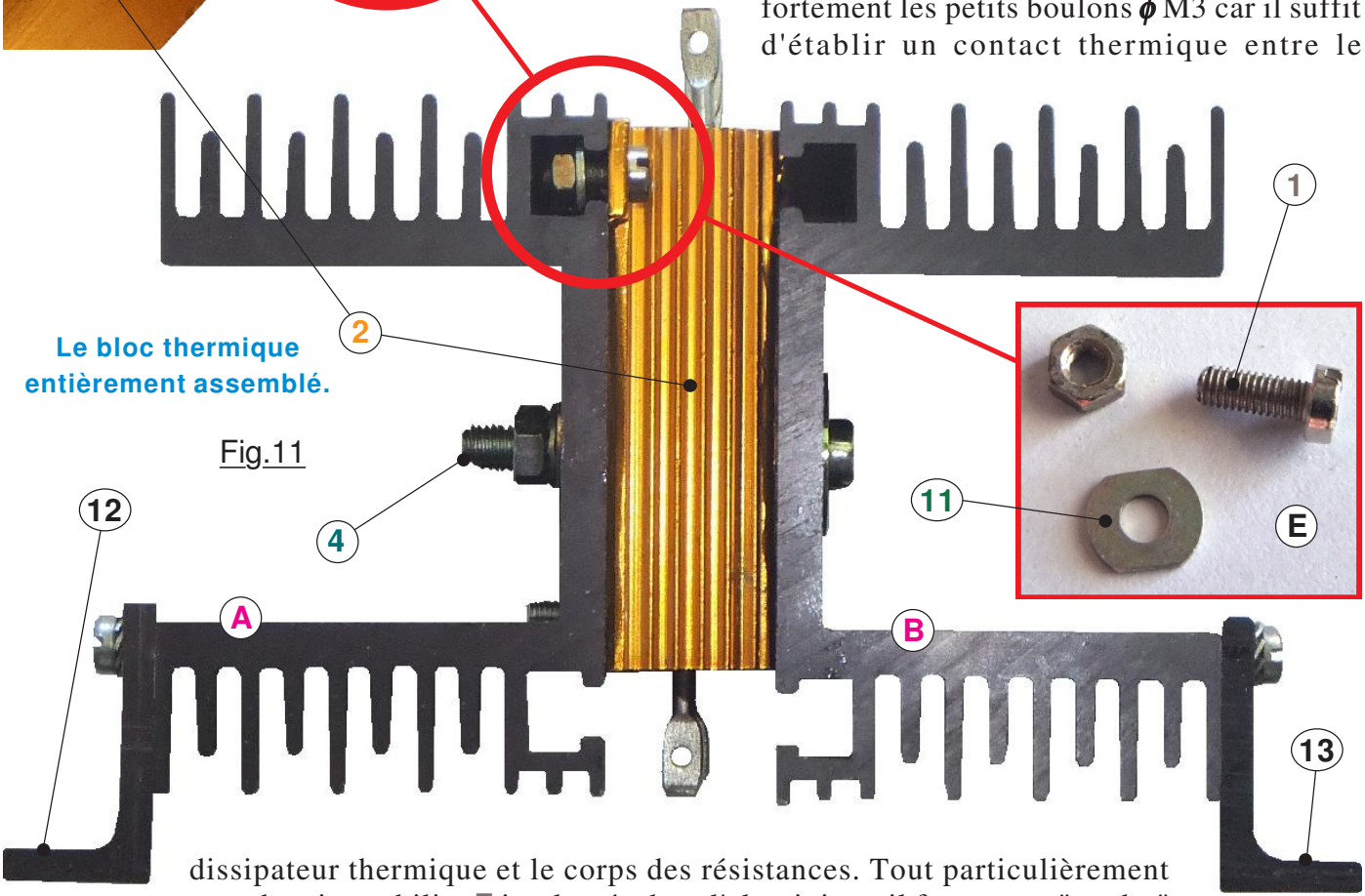


Fig.10



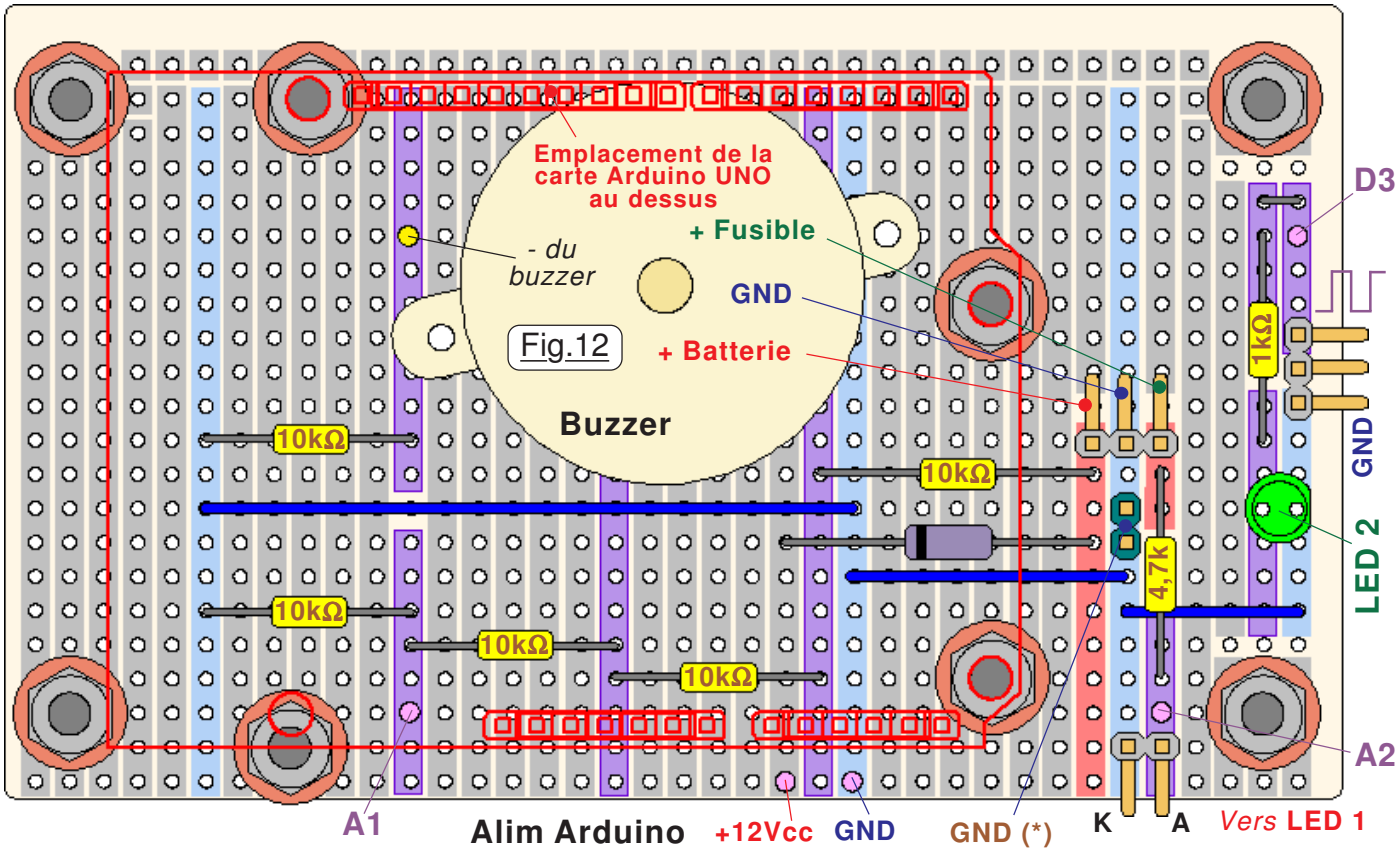
Le bloc thermique entièrement assemblé.

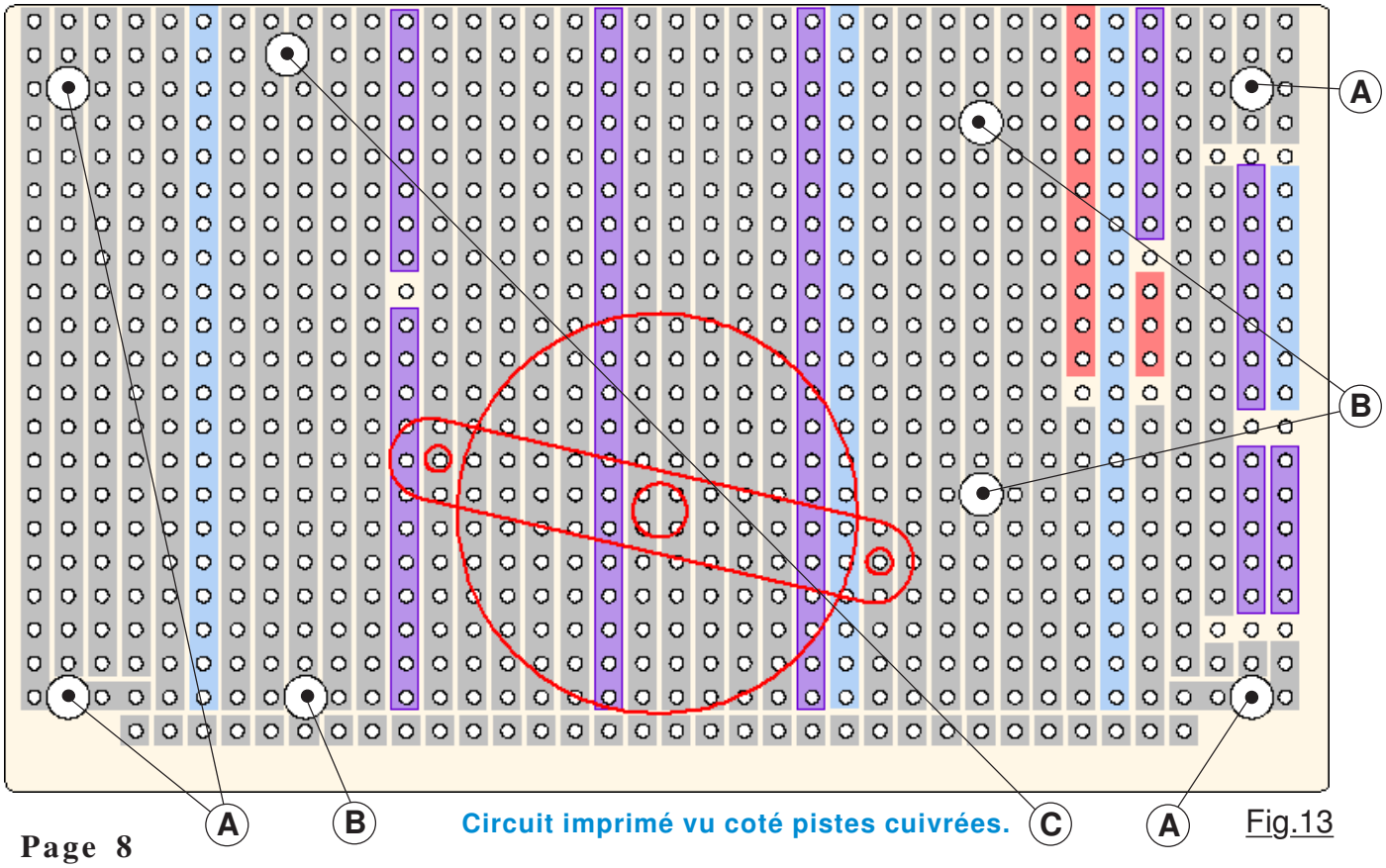
Fig.11

dissipateur thermique et le corps des résistances. Tout particulièrement quand on immobilise **7** implantée dans l'aluminium, il faut rester "tendre" car le filetage est relativement fragile. Par contre, pour l'assemblage entre les deux blocs on peut y aller plus fermement car les boulons en acier sont résistant, et le total doit former un bloc complet bridé bien rigide. Aligner à plat les quatre pattes de fixation inférieures **12** et **13**.

06) Le circuit imprimé principal :

Assurant le support de la carte Arduino UNO et des divers composants périphériques, la plaque électronique est réalisée à partir de cartes prépercées de prototypage faciles à trouver sur Internet. Celles que j'utilise sont pourvues de bandes de cuivre. En coupant ces pistes et en "grattant" le cuivre avec un cutter, on réalise les isolements nécessaires. Le dessin de la Fig.12 présente à grande échelle l'implantation des composants. Ce dessin montre les pistes cuivrées comme si le support était transparent. Les bandes de cuivre sont coloriées, facilitant l'interprétation de leur





Circuit imprimé vu coté pistes cuivrées. Fig.13

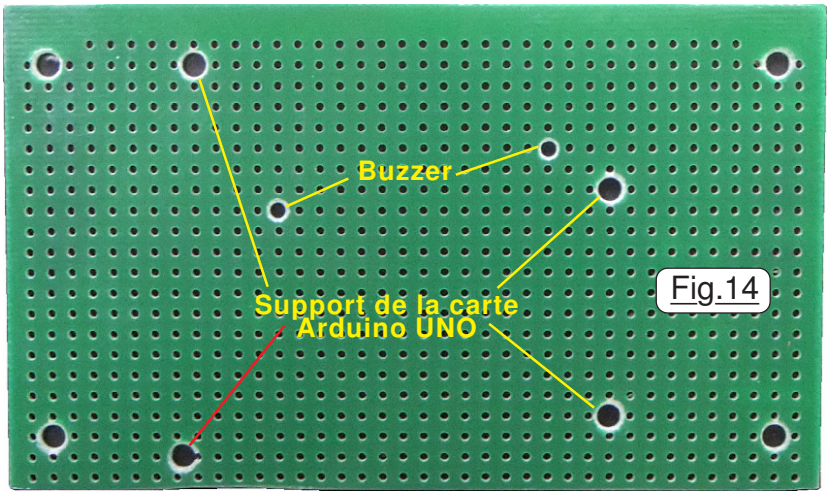


Fig.14

affectation et mettant en évidences certaines coupures de pistes peu repérables. Bleu clair pour **GND**, violet pour des E/S avec l'ATmega328. Le **Buzzer** est un modèle de grandes dimensions, car il était disponible et la place ne manque pas. S'il est placé en biais, c'est pour une raison simple : Faire coïncider ses trous de liaison avec ceux pré-percés de la plaque de prototypage. Le petit cercle jaune montre l'emplacement de la liaison avec le fil du **Buzzer**. Les cinq autres petits cercles roses indiquent les autres liaisons filaires. La Fig.13 montre à une échelle identique le circuit vu côté pistes cuivrées. Le contour du **Buzzer** est repéré en rouge bien qu'il se trouve sur l'autre face. Les quatre trous **A** dans les angles sont agrandis à ϕ 3mm pour pouvoir traverser avec des vis classiques au diamètre nominal ϕ M3. Notez au passage que ces trous coïncident avec ceux déjà présents sur la plaque. Il en est de même pour les trois trous **B** de liaison avec les entretoises qui supportent la petite carte Arduino UNO. Attention : Le dernier trou **C** pour supporter ce module est un peu décalé et ne respecte pas les écartements standards en pouces. De ce fait, il se trouve exactement entre les deux orifices prépercés et exige un pointage précis. Les Fig.14 et Fig.15

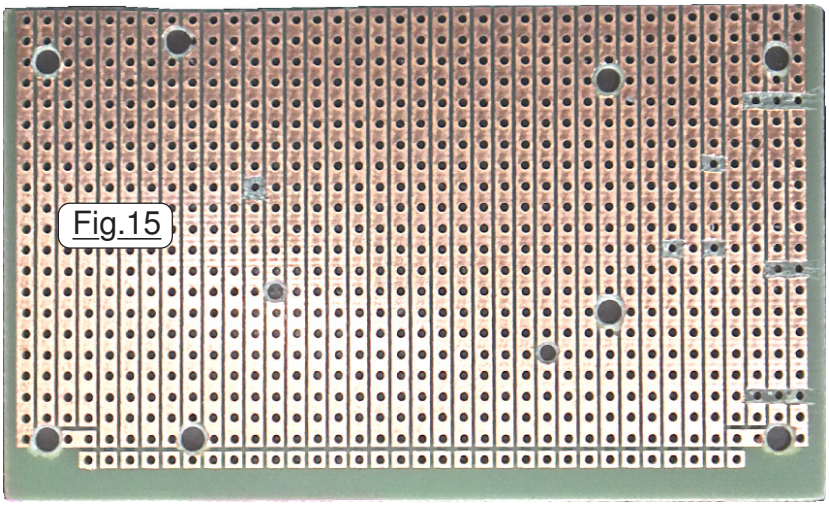


Fig.15

présentent le circuit avant de commencer à y souder les divers composants. On y remarque les divers trous de traversée de la visserie ainsi que les coupures pratiquées sur les pistes cuivrées. Notez au passage que la liaison avec le coffret exige de surélever de façon importante ce circuit imprimé pour que l'afficheur LCD implanté en "gigogne" soit proche du couvercle. De ce fait la plaque reposera sur des entretoises longues. Les vis traversant en **A** seront en fait réalisée par sciage à la bonne longueur d'une tige filetée ϕ M3 facile à se procurer dans les magasins de bricolage. Il importe de remarquer que la visserie est systématiquement complétée par des rondelles pour répartir les efforts de la tête de vis ou des écrous. Sur le dessus du circuit imprimé on peut utiliser des rondelles métalliques banales. En revanche, sur le dessous coté pistes il faut impérativement intercaler des rondelles isolantes. Par exemple sur la Fig.16 montrant le circuit achevé on observe les deux petits boulons qui assurent la liaison avec le **Buzzer**. Ici encore nous avons à faire à des composants récupérés, et vous avez compris qu'un appareil électroménager ne va jamais à la déchetterie sans au préalable subir un "cannibalisme général". Rien ne se perd, rien ne se crée, tout se récupère comme dirait un certain Lavoisier !

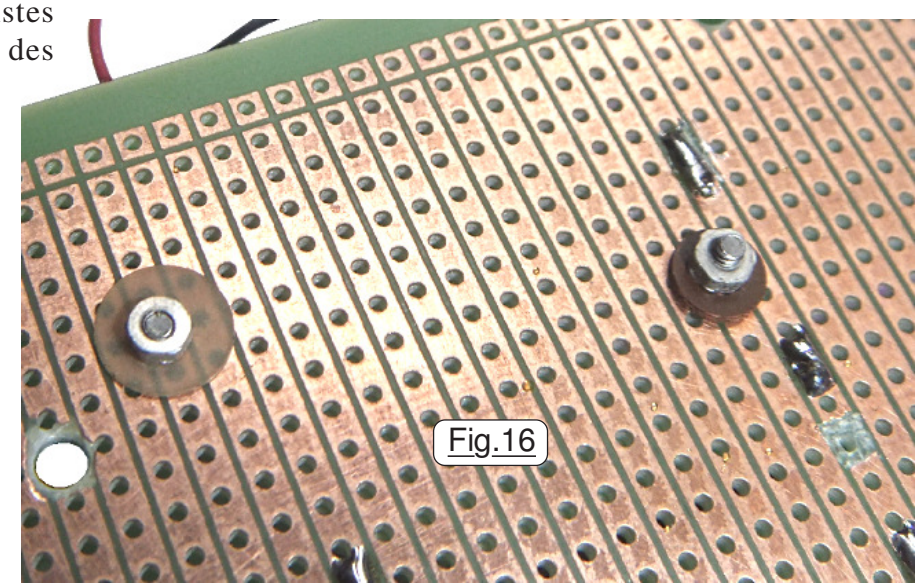


Fig.16

07) Soudure des composants :

Vraiment sans histoire, la soudure des divers composants ne pose strictement aucun problème. Comme toujours, après avoir achevé cette opération, un examen complet est effectué avec une loupe à fort grossissement pour s'assurer que toutes les soudures sont parfaites, et qu'elles ne débordent pas, créant une liaison interdite avec la piste voisine. La Fig.17 montre la plaque pratiquement terminée. Il ne manque que le fil de raccordement vers **A3** en **1** ainsi que le connecteur **3** et les essais peuvent commencer. Observez au passage que les soudures sur les petits connecteurs HE14 sont systématiquement isolées avec de la gaine thermo-rétractable. En **2** les liaisons femelles viennent se brancher sur le connecteur HE14 de type coudé. Ces connecteurs sont utilisés de façon assez particulière, on dirait qu'ils sont branchés à l'envers. Cette incongruité apparente est en réalité volontaire. Il se trouve que les connecteurs HE14 qui sont sur le SHIEL de l'afficheur LCD sont relativement hauts. *(Voir Fig.18 et Fig.19)*

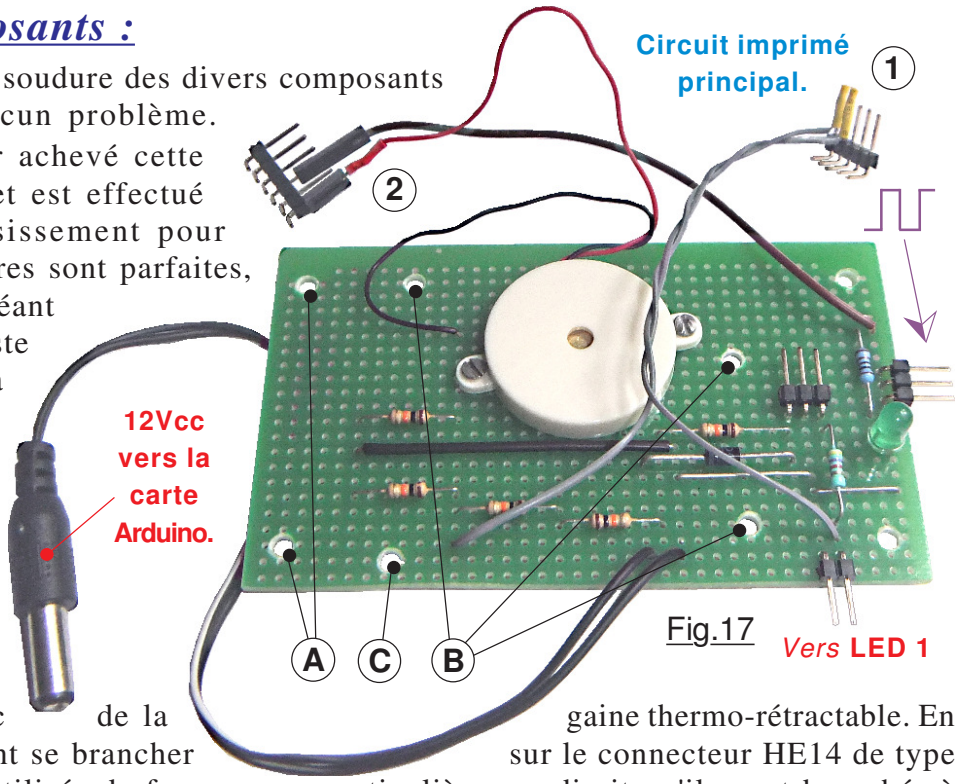


Fig.17

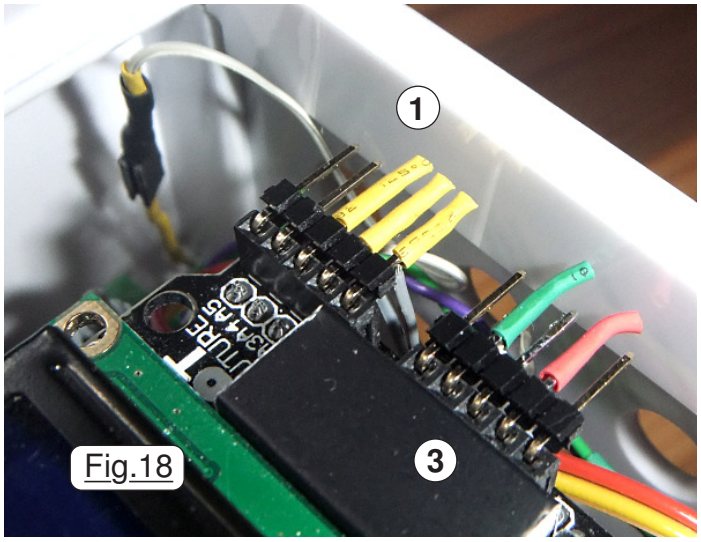
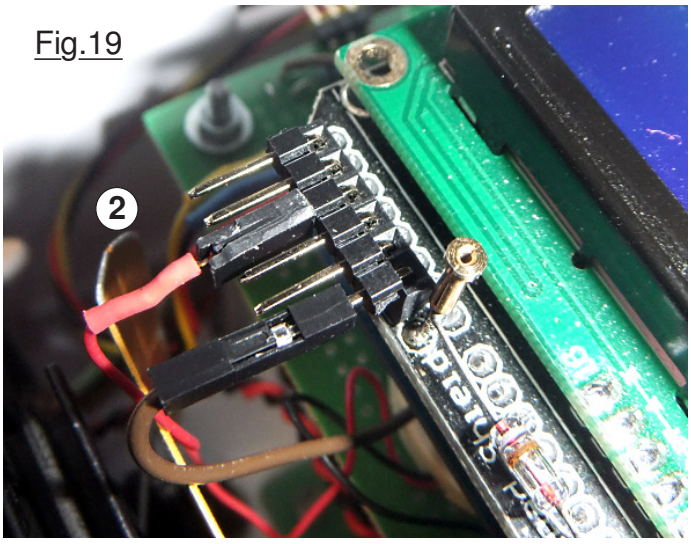


Fig.18

Fig.19



08) La mesure de température du radiateur :

Préambule à l'intégration du bloc thermique dans le coffret, il faut impérativement placer le support et le petit module électronique équipé de la thermistance CTN. Montré sur la Fig.20 ce tout petit circuit imprimé est très facile à se procurer sur Internet. La petite plaque ne comporte que deux éléments. La thermistance et une résistance de 10kΩ, ces deux éléments formant un diviser de tension qui alimenté avec le +5Vcc régulé de la carte Arduino sera numérisé par l'entrée analogique **A3**. Sur la Fig.21 la plaque **D** ressemble à un blindage de char d'assaut. Illusion d'optique, car cette image est saisie en macrophotographie. Dans la pratique il s'agit d'une petite plaque métallique de moins d'un demi-millimètre d'épaisseur. Cette pièce qui supporte le module électronique est découpée et pliée dans de la tôle galvanisée, puis vernie pour ne pas s'oxyder. En **A** on trouve la thermistance, le module étant immobilisé sur **D** par les deux vis **B** de diamètre nominal ϕ M2. Le petit circuit est écarté du support par les deux entretoises isolantes **F**.

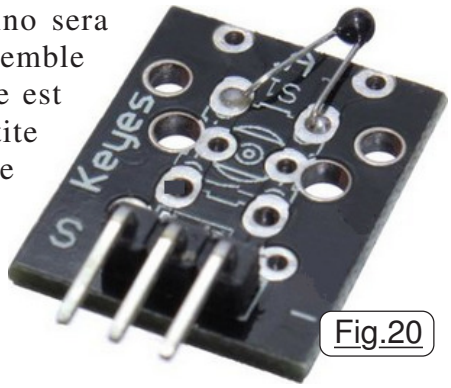


Fig.20

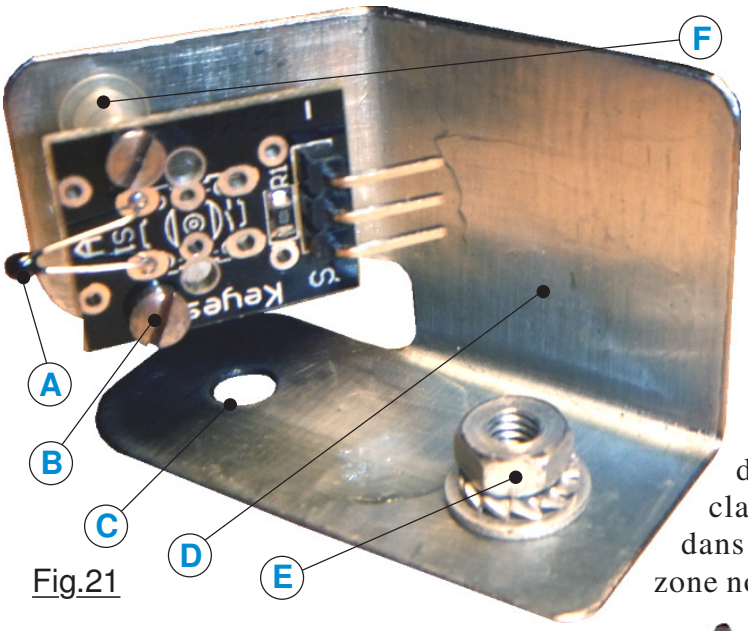


Fig.21

L'écrou **E** vient immobiliser le petit support à l'intérieur du bloc **A** sur les vis **4**. Le trou **C** laisse passer la deuxième vis de bridage **4** et empêche la plaquette de pivoter quand on serre l'écrou **E** avec une clef à pipe. Sur la Fig.21 la rondelle plate, la rondelle éventail de freinage et l'écrou **E** sont "posés" sur la plaquette métallique pour montrer dans quelle zone est effectuée la liaison. On se doute que sous ces trois éléments il y a un trou de diamètre 4mm comme en **C**, et séparé de ce dernier par l'entraxe existant entre les deux vis de bridage **4**. La Fig.11 située en page 7 montre clairement que les deux blocs **A** et **B** présentent dans leur partie centrale toute une zone non occupée formant une grande cavité. C'est dans le volume libre du bloc thermique **A** que se loge la petite plaquette **D** et son contenu. La vue de la Fig.22 montre bien l'emplacement de ce sous ensemble qui peut se débrancher aisément car le petit circuit imprimé est muni d'un connecteur HE14 à trois broches. Bien plus parlante qu'un long texte d'explications, la vue de coté sur la Fig.23 est surchargée en rouge pour montrer l'emplacement du connecteur HE14 femelle qui vient se brancher sur le petit module. On constate qu'il n'y a pas beaucoup de place entre ce dernier et la petite plaque métallique pliée **D**. Pour pouvoir facilement l'insérer et le débrancher, il faudra que les fils électriques de liaison soient soudés perpendiculairement aux cosses comme surchargé en jaune sur la photographie. Enfin, pour clore ce chapitre, la fig.24 met en évidence dans l'encerclé rouge la jonction entre la thermistance et le bloc **A** améliorée par un "petit paquet" de graisse thermique au silicone, ceci dit ... vous deviez certainement y avoir déjà pensé !

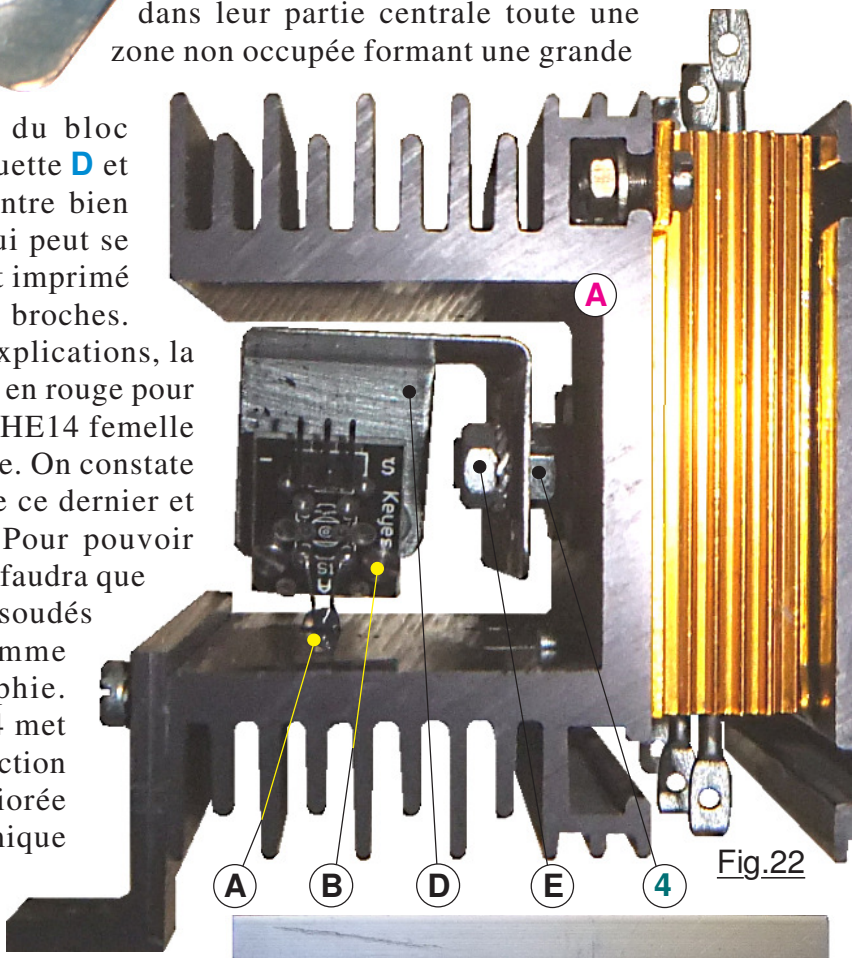


Fig.22

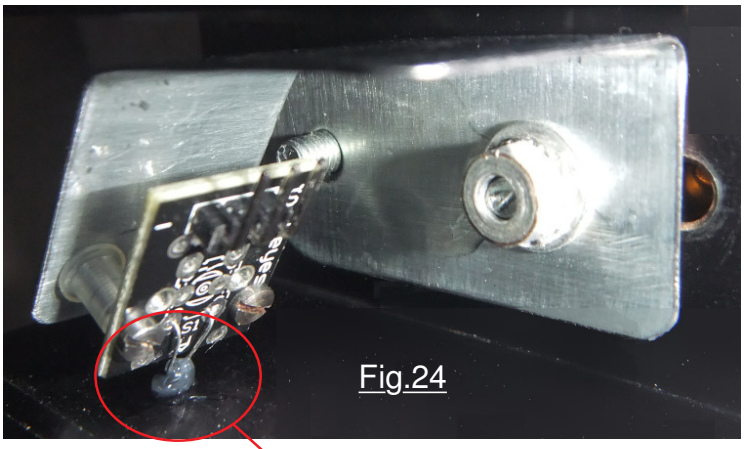


Fig.24

Graisse thermique au silicone

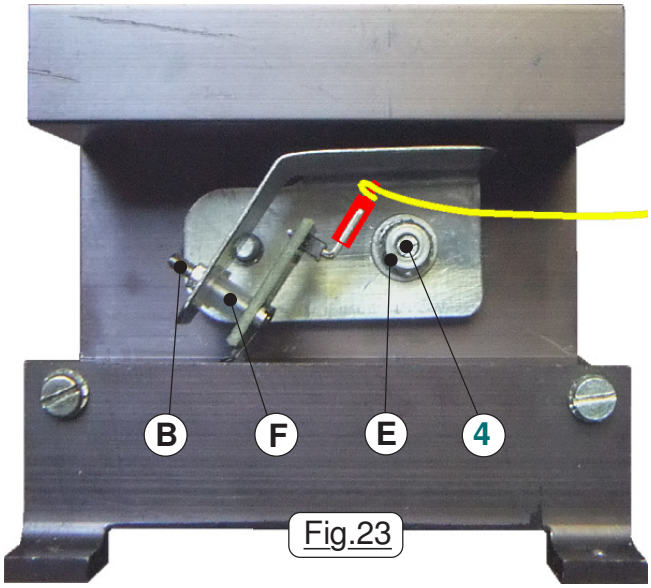


Fig.23

09) La réalisation du coffret :

Chacun utilisera les techniques qu'il maitrise, ces dernières étant fonction des matériaux dont il dispose et surtout de l'outillage présent dans son atelier de bricolage. Je ne vais pas reprendre dans ce descriptif la façon dont je réalise les boitiers personnels, ayant largement détaillé mes procédés dans divers didacticiels. Par exemple en allant sur le lien :
http://www.robot-maker.com/ouvrages/interface-puissance-arduino/realisation-pratique-coffret/
vous trouverez toutes les explications sur mes techniques de fabrication. Pour l'appareil décrit, les dimensions externes font 216mm de longueur, 133mm de largeur et 119mm de hauteur, couvercle compris. Pour compléter la Fig.4 de la page 3 la

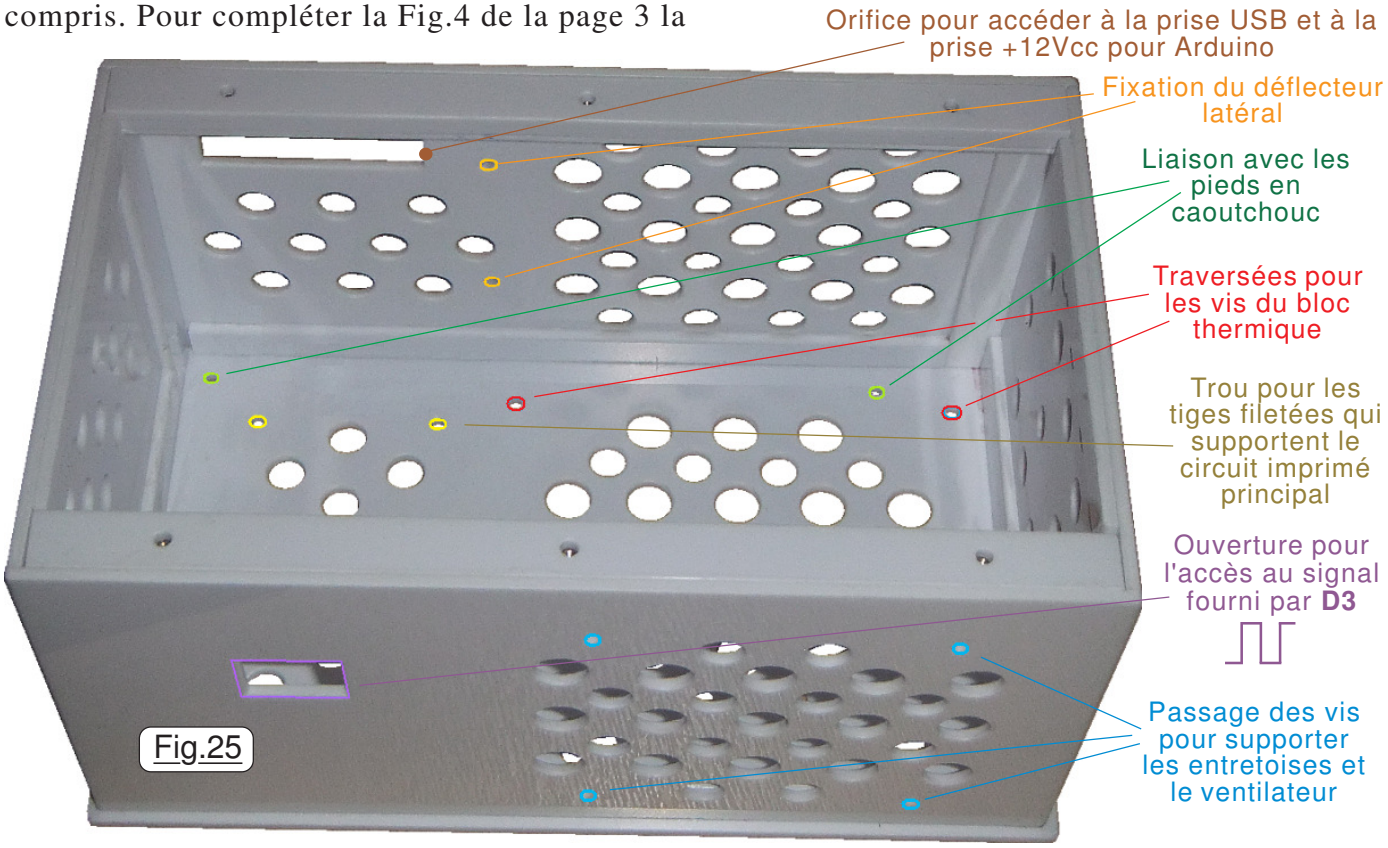


Fig.25

photographie de la Fig.25 présente le coffret, la face qui supporte le ventilateur étant orientée vers nous. Les divers trous pour les traversées de la visserie sont coloriés sur la photographie pour les mettre en évidence. Les deux longerons qui intègrent les écrous prisonniers de fixation du couvercle rigidifient considérablement "les grandes surfaces". Les deux flancs latéraux sont renforcés par une bande collée à plat juste sous le couvercle. L'Orifice pour accéder à la prise USB et à la prise +12Vcc pour Arduino sert également pour faciliter le serrage des écrous supérieur des tiges filetées qui traversent les divers "trous jaunes". Trois trous pour le passage des vis qui tiennent les pieds en caoutchouc sous la semelle sont situés dans les angles. *(Couleur vert clair.)* Celui visible à droite au fond est écarté vers l'intérieur, car à proximité il y a le trou rouge de fixation du bloc thermique. Ce dernier est surélevé de la semelle par des entretoises en nylon de 8mm de hauteur pour ne pas que la température des pattes de fixation n'affecte le fond du coffret. Les têtes des

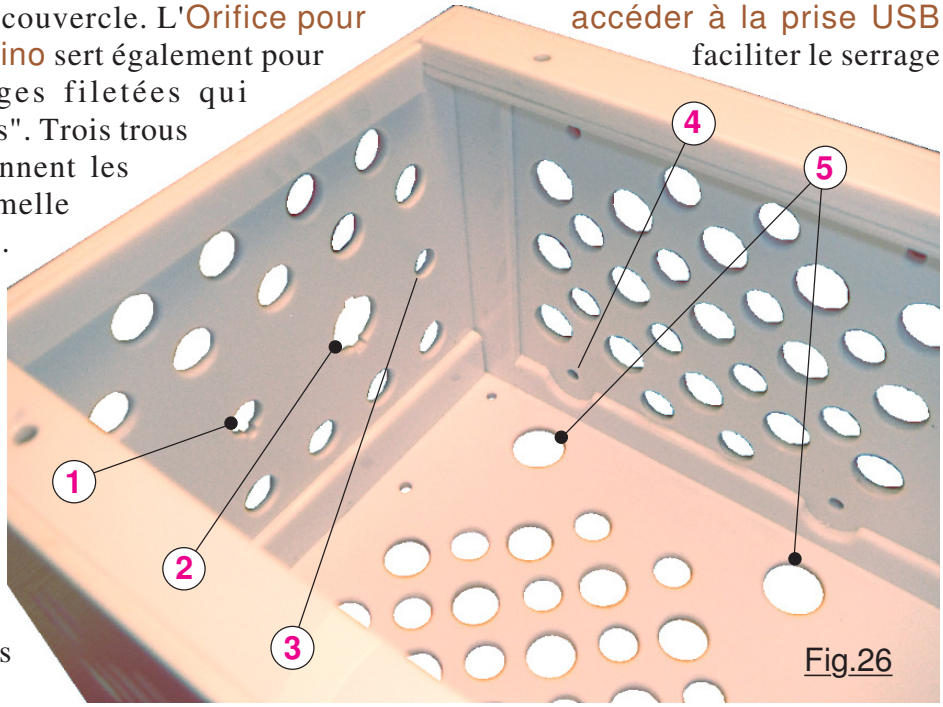


Fig.26

quatre vis de liaison portent sur des rondelles plates de grand diamètre. Pour achever ce chapitre, la Fig.26 montre une vue plongeante dans le coffret. En **1** se trouve le passage de la douille pour fiche banane avec l'encoche anti rotation. En **2** la traversée pour le support du fusible muni de deux ergots pour ne pas qu'il tourne quand on serre son écrou de liaison. Le trou **3** sert à supporter la LED rouge placée juste à proximité du fusible. En **4** on peut voir le trou de passage des vis de liaison du ventilateur avec le dégagement dans le renfort longitudinal collé à plat sur la façade verticale. Contrairement à tous les autres gros trous circulaires qui servent à la ventilation, les deux trous **5** permettent le passage des outils pour mettre en place et serrer les écrous inférieurs de liaison ventilateur.

10) L'intégration des divers modules dans le boitier :

C'est de loin la phase qui risque de vous poser de réels problèmes si elle n'est pas convenablement étudiée en anticipant tous les pièges. Désirant aboutir à un ensemble relativement compact pour ne pas qu'il encombre exagérément l'armoire de rangement, ***le taux d'occupation du volume interne a été poussé à l'extrême*** comme vous pourrez le constater sur les diverses photographies qui accompagnent ce didacticiel. Pour gagner un maximum en volume, la zone vide coté **B** du bloc thermique permet de loger les douilles pour fiche bananes et le support de fusible qui sont très proéminents. Du coup, pour arriver à tout imbriquer l'assemblage final ressemble assez à un jeu de Tétris. Avant de passer à la réalisation du coffret, il faut impérativement réaliser des dessins à l'échelle unitaire, étudier finement dans quel ordre on assemblera les éléments ***et vérifier que le passage de la visserie et surtout des outils de serrage ne relève pas de l'exploit***. Je n'insisterai jamais assez sur cet aspect VITAL du projet. L'agencement doit permettre non seulement de pouvoir tout assembler, souder, ***et surtout par la suite de permettre la dépose aisée des éléments principaux*** en vue d'opérations d'améliorations ou de maintenance. PRIMORDIAL ! Alors avant de commencer à souder les divers fils électriques, effectuez un assemblage complet pour vérifier le bienfondé de vos études et corriger les pièges éventuels ... toujours possibles !

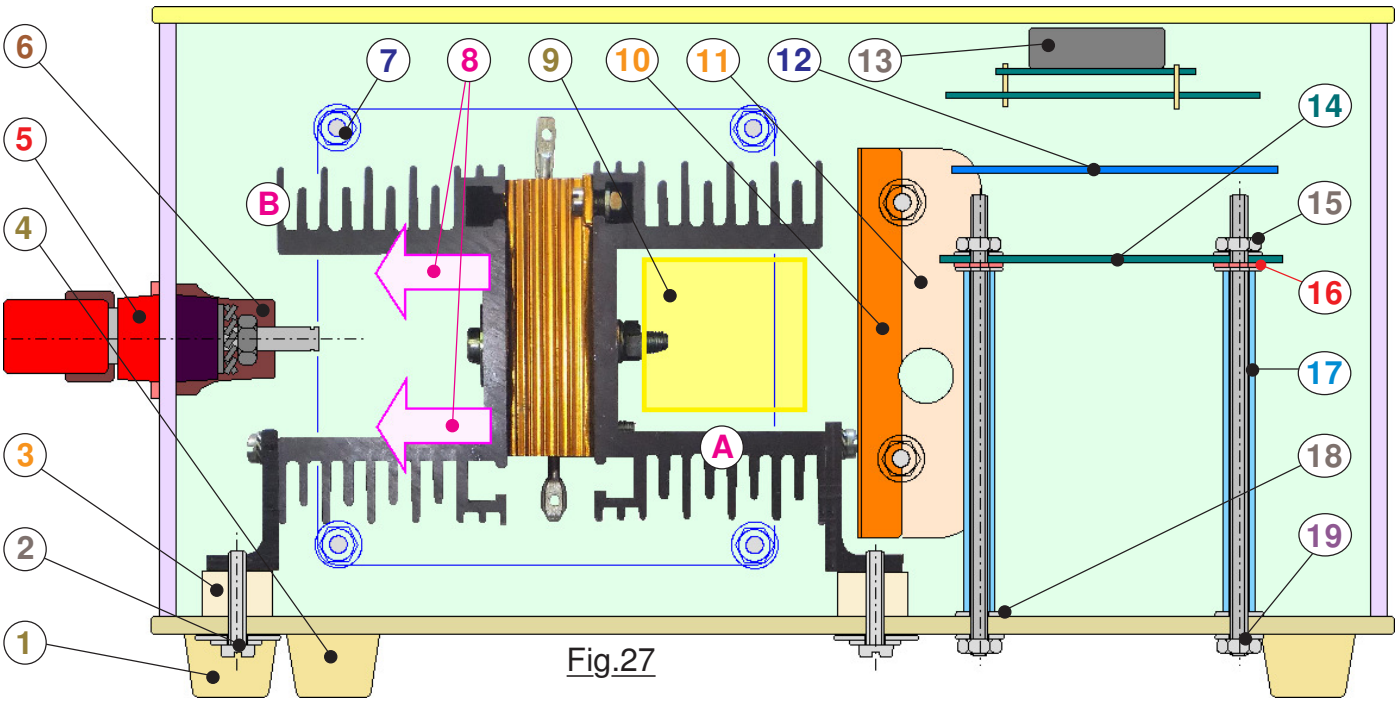
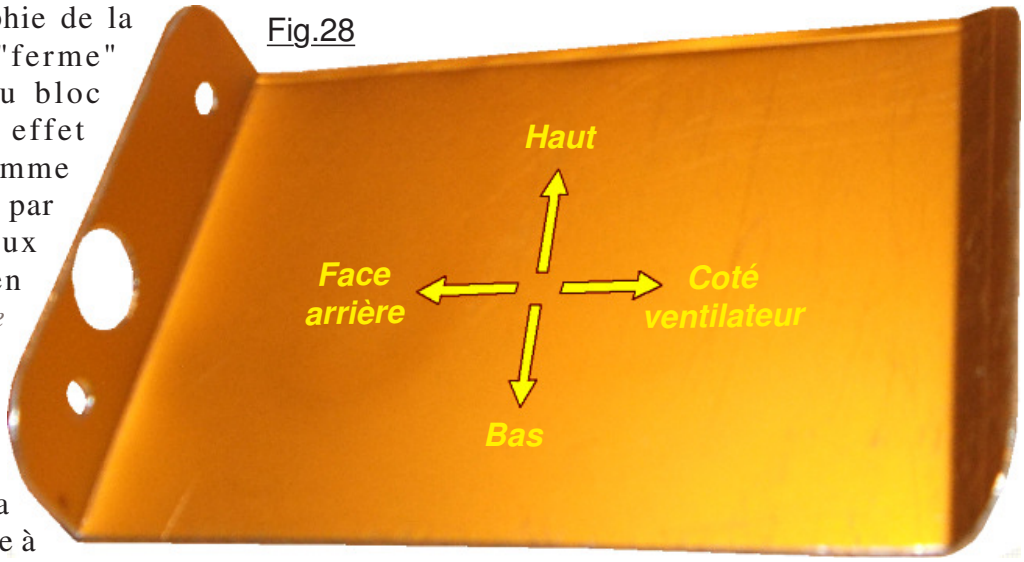


Fig.27

La coupe frontale de la Fig.27 montre le coffret sur lequel on a fixé les trois pieds d'angle **1**, puis celui qui est décalé **4**. Ensuite on a mis en place la **LED 1** et le ventilateur **7** ainsi que les douilles pour fiches bananes **5**. Porte fusible **6** en place, on procède au soudage des gros fils sur les deux bornes de puissance et sur le support de fusible. On a immobilisé la petite platine thermistance **9** sur **A**. On a soudé les gros fils sur les résistances, avec à leurs extrémités les grosses cosses de connexion. On enfile le bloc thermique par déplacement latéral **8**. On le soulève. Puis on intercale les entretoises **3** et l'on immobilise le gros module avec les vis **2**. Après tests électriques soignés de tout ce qui est actuellement en place, on procède à la fixation de déflecteur latéral **10**. En **11** couleur claire, la plaque forme une équerre qui autorise sa liaison avec la face verticale arrière. Cette partie est percée d'un gros trou ϕ 10 pour ne pas masquer l'orifice d'aération correspondant.

Montrée sur la photographie de la Fig.28, cette plaque "ferme" latéralement la zone du bloc thermique, formant un effet tunnel. L'air soufflé comme dans une conduite forcée par le ventilateur **7** est mieux canalisé. Cette plaque en aluminium anodisé *(Issue comme pas mal d'autres éléments mécaniques d'une récupération.)* protègera de surcroit l'électronique de la chaleur dégagée, si la température augmente suite à une panne du ventilateur qui n'est pas à exclure. Vous avez compris que l'orientation du bloc thermique est telle que **V** chasse l'air à travers les ailettes de refroidissement qu'il voit par la tranche. En **14** nous avons le circuit imprimé principal sur lequel est immobilisée la carte Arduino UNO en **12** sur laquelle s'enfiche le SHIELD LCD dont l'afficheur est représenté en **13**. *(Bien que semblant léviter sur le dessin, les deux modules sont représentés à l'échelle et respectent leurs positions respectives.)* Le circuit **14** est placé sur les entretoises **17** en intercalant quatre rondelles d'appui telles que **18**. Entre la rondelle métallique d'appuis sur l'entretoise **17** et les pistes cuivrées du circuit **14** sont intercalées les rondelles isolantes **16**. Enfin le total est serré entre les écrous **15** et **19**.

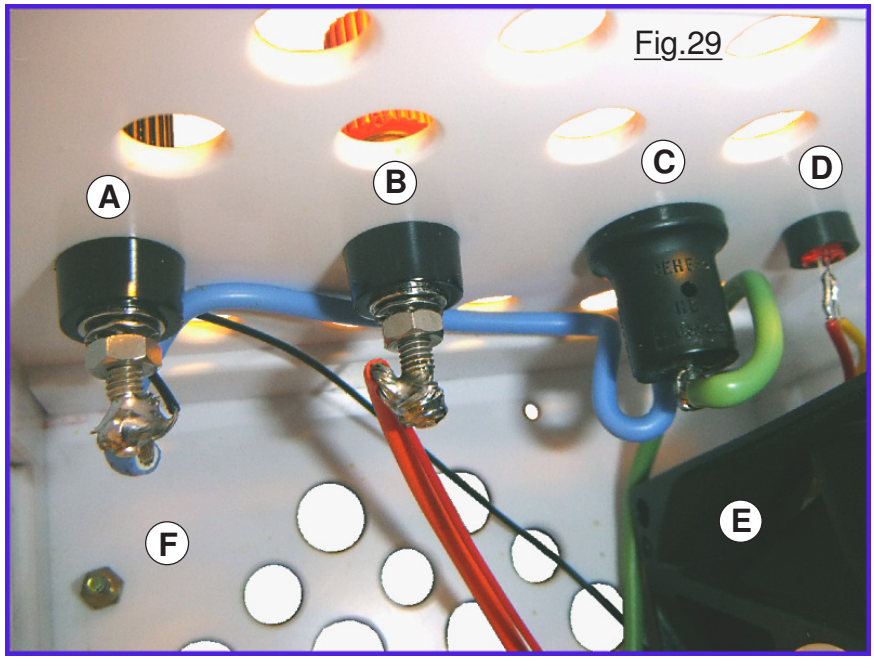
Fig.28

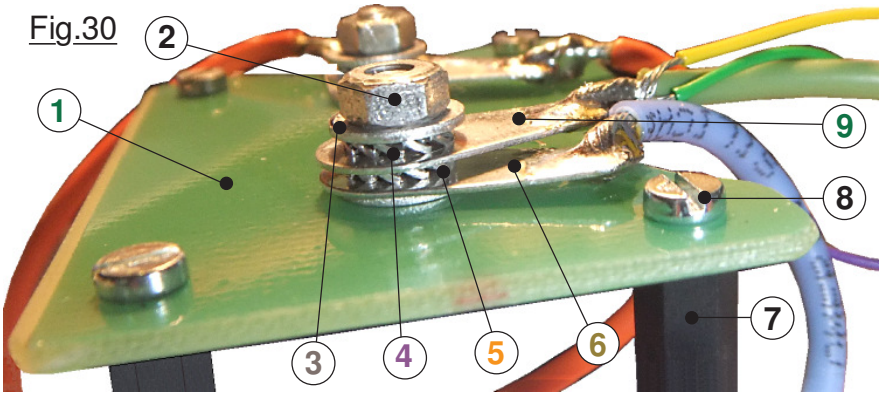


11) Câblage du circuit de puissance :

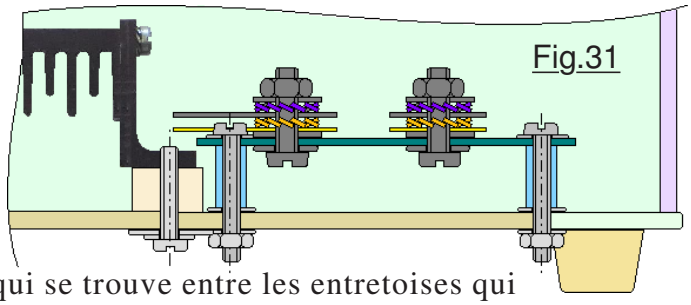
Techniquement, si l'on ne désire pas dégrader l'intensité qui traversera **R** il importe d'éviter les pertes en ligne. Il faut donc utiliser des fils électriques de section respectables et soigner les soudures. La Fig.29 illustre ce propos. Prise en macrophotographie, les liaisons ressemblent à des barreaux de chaise. Ceci étant précisé, les lignes ont été choisies en utilisant les plus gros fils souples de cuivre disponibles dans les stocks. À gauche nous avons en **A** la douille négative. Le petit fil noir permet de distribuer **GND** vers le circuit imprimé principal. Le gros fil bleu va au "point froid" du support de fusible **C**. Bien visible en **D** est immobilisée la **LED 1**. En **E** le ventilateur est en place sur la face avant. On peut observer en **F** l'écrou qui solidarise le pied en caoutchouc qui est décalé vers "le bas". De la douille **B** part le gros fil orange qui va vers les résistances **R** via le gros bornier dont il est question en fin de ce chapitre. Juste au dessus de cette liaison de forte section se trouve un fil rouge qui amène le **+12V** de la batterie sur le circuit imprimé **14**. Enfin, pour boucler le circuit de puissance, le fil de forte section vert va à l'autre extrémité des résistances **R** en passant par le bornier de puissance. Le "haut" des trois résistances de 8Ω est relié à un gros fil de couleur rouge terminé à son extrémité par une cosse calibrée en conséquence. Cette dernière sera en liaison avec la cosse située à l'extrémité du fil orange. Enfin le "bas" des trois résistances de 50W est réuni à une ligne bleue terminée également par une cosse de fort calibre. Le fil bleu et le fil rouge étant désolidarisés du bornier, le bloc thermique est indépendant pour sa dépose ou son insertion dans le coffret. *(Sans oublier de brancher ou débrancher le petit connecteur HE14 à trois broche du module thermistance CTN).*

Fig.29

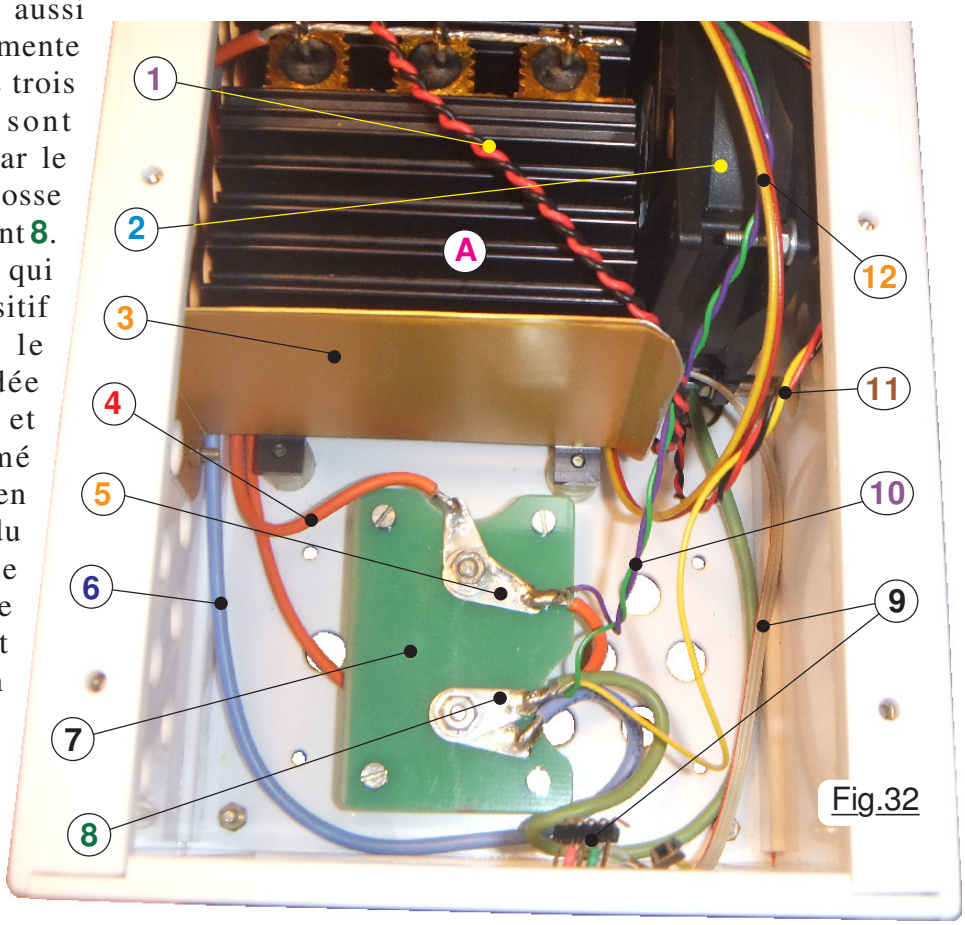




Montré en gros plan sur la Fig.30 le bornier de puissance réunissant les résistances au fusible et aux douilles de branchement de la batterie est constitué d'un morceau de circuit imprimé **1**. *(Cette plaque n'est pas cuivrée et sert de support isolant rigide.)* Chaque borne est constituée d'un boulon ϕ M4 dont on observe l'écrou **2** qui presse l'assemblage entre des rondelles plates telles que **3**. Les deux cosses réunies **6** et **9** sont "mordues" par les dents de la rondelle éventail **5**. Cette "agression" diminue considérablement la résistance de conduction de la liaison électrique ainsi établie. La rondelle **4** assure le freinage de l'écrou par élasticité. La rondelle plate **3** autorise toutefois un desserrage relativement aisé pour déposer si utile le bloc thermique. Pour simplifier la vue, sur le dessin de la Fig.27 le bornier de puissance n'est pas représenté. La zone située en bas et à droite est représentés sur la Fig.31 sur laquelle figure le dessin de ce bornier qui se trouve entre les entretoises qui supportent le circuit imprimé principal. Sur ce dessin elles ne sont pas représentées.



Encore un dernier petit regard indiscret sur la Fig.32 et nous allons pouvoir aborder l'intégration et l'assemblage de l'électronique de pilotage. Le dissipateur **A** domine avec en haut de l'image la réunion des trois résistances par le gros fil rouge que l'on retrouve en **4**. En face des ailettes de refroidissement on voit le ventilateur **2**. Rabattus au dessus pour dégager la zone inférieure la ligne **12** va à la **LED 1**. Guidant le flux d'air réchauffé par **A** le déflecteur **3** cloisonne le bloc thermique. Issu des trois résistances le gros fil rouge arrive en **4** branché sur le bornier par sa cosse indépendante. Sur la même borne est connectée la cosse **5** sur laquelle est soudé le gros fil orange qui va à la douille **B**. Sur cette cosse est aussi réuni le petit fil violet qui alimente le positif du ventilateur **2**. Les trois résistances de puissance sont soudées entre elles dessous par le gros fil bleu **6** qui va à une cosse indépendante sur le plot recevant **8**. De cette cosse **8** part le fil vert qui alimente par la ligne **10** le positif de **2**. Arrivant par dessous le ventilateur **2** la ligne torsadée rouge et noir **1** amène **GND** et **+12V** vers le circuit imprimé principal. Elle est complétée en **11** par le fil jaune arrivant du "point chaud" du support de fusible. Enfin en **9** on observe la ligne qui arrive du petit module thermistance avec à son extrémité le petit connecteur HE14 qui sera branché sur la carte Arduino via le circuit SHIELD LCD en gigogne.



12) Assemblage de la carte Arduino UNO sur le circuit imprimé principal :

Parfaite électroniquement et informatiquement, la carte Arduino UNO est affectée de deux faiblesses mécaniques qui en compliquent singulièrement la mise en œuvre. Le premier point noir est relatif à l'écartement des deux connecteurs dédiés aux broches **D0** à **D13** qui ne sont pas à la distance standard interdisant l'utilisation de plaques de prototypage prépercées. Dommage ! Dans ce projet cet inconvénient ne joue aucun rôle puisque nous utilisons un SHIELD du commerce dont le circuit imprimé conçu à la demande résout cette difficulté. Nous avons vu que l'un des trous de passage des vis de liaison n'est pas non plus à une position par rapport aux trois autres permettant son centrage sur des coordonnées en dixièmes de pouces. Ce détail ne fait que rendre le perçage du trou plus délicat sur une plaque prépercée. Rien de bien tragique. Il nous reste encore deux petites difficultés à contourner. Comme vous l'aurez forcément constaté, le trou de traversée situé dans l'angle de la prise USB ne permet pas d'utiliser une vis ϕ M3 classique car il est trop proche des deux connecteurs HE14. Bien visible sur la Fig.33 il suffit de sélectionner une vis en nylon et de façonner deux méplats à angle droit sur la tête fendue. La deuxième pierre d'achoppement réside dans les picots soudés des deux connecteurs HE14 qui dépassant sur le dessous et trop proche de ce trou de traversée. Il en résulte l'impossibilité d'utiliser une entretoise standard. Pour palier cet embarras, sur le prototype un canon d'isolement pour boitiers TO3 *(En bleu sur la Fig.34)* a été placés sous la carte Arduino. Le décolletage de diamètre réduit passe facilement entre la vis et les queues des composants qui dépassent du circuit imprimé. Vous pouvez aussi approvisionner des entretoises en nylon faciles à usiner avec une lime pour pratiquer un dégagement local. La Fig.35 apporte la preuve que ces petites complications sont faciles à contourner et surtout montre qu'avec des vis nylon présentant une longueur sous tête de 18mm, Arduino laisse largement assez de place entre sa carte électronique et le circuit imprimé principal. En **A** la ligne +Vcc issue de la batterie via la diode de protection **D** va vers le régulateur 5Vcc intégré d'Arduino. En **V** bien observable les vis nylon supportant la carte électronique. Également bien observable en **C** le connecteur coudé qui semble soudé sur le mauvais coté des broches. Nous pouvons passer à l'intégration de cet ensemble dans le coffret. Cette opération sans vraiment exiger des mains de sage-femme reste un peu délicate, car il faut placer les entretoises, les tiges filetées, les écrous inférieurs et la ribambelle de rondelles. Puis, les tiges filetées étant poussées vers le haut par du carton situé sur le dessous et le boitier sur ses pieds, on amène le circuit imprimé principal muni d'Arduino par le dessus. Tige filetée par tige filetée on fait coïncider avec son trou de passage, sur le dessus on place la rondelle et l'écrou. Quand les quatre colonnes sont en place, on peut alors les orienter en positions verticales et serrer les écrous supérieurs et inférieurs pour rigidifier le total. Bien en place, on peut alors brancher la fiche 12Vcc sur l'entrée Arduino. Sans que ce soit spécialement délicat, il importe

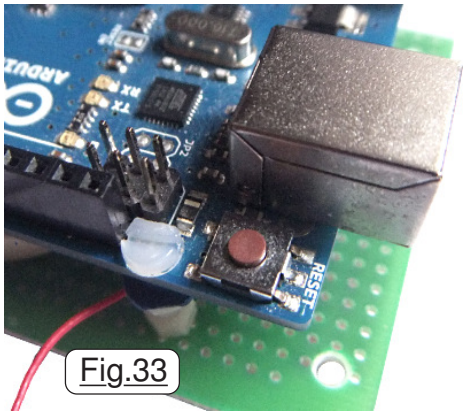


Fig.33

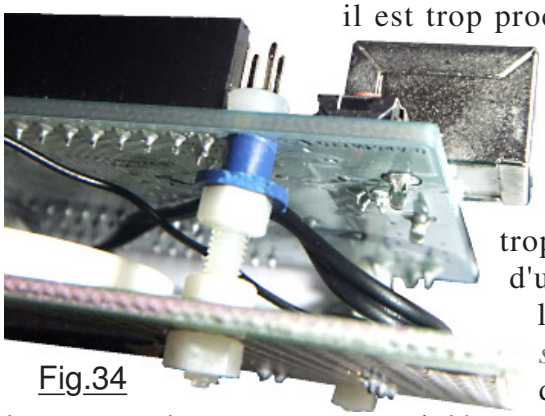


Fig.34

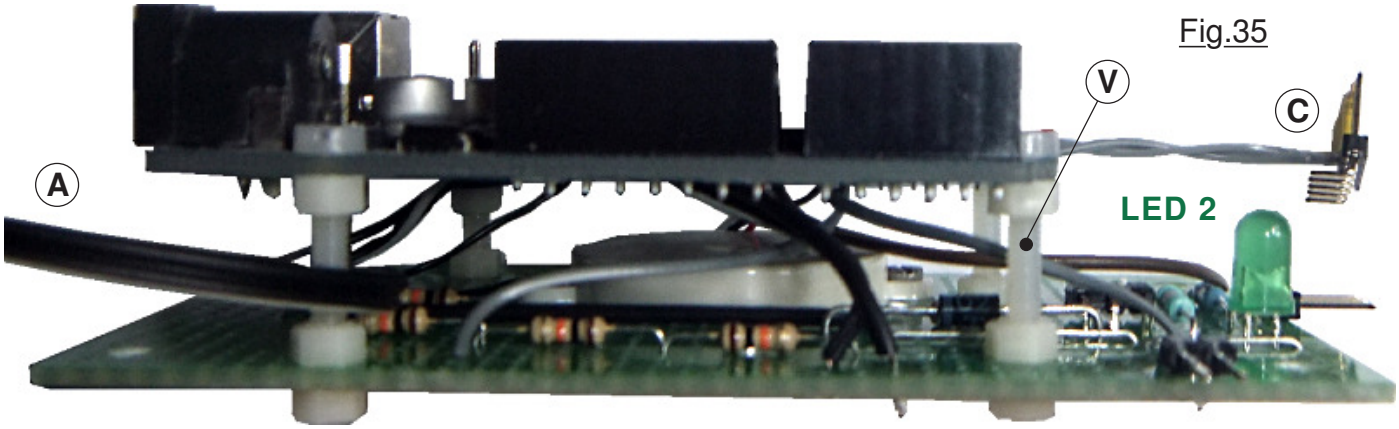


Fig.35

d'effectuer une dernière vérification électrique avant de procéder à cette opération d'assemblage. Dans ce but on réalise "en l'air" la configuration d'essai de la Fig.36 le SHIELD n'étant pas encore enfiché sur Arduino. Avant de brancher la fiche 12Vcc qui alimente la carte Arduino par la liaison **1**, on met en tension les deux douilles pour fiches bananes. ***Les essais ainsi que la programmation sont réalisés alors que le fusible est retiré de son support***, car consommer bêtement de l'énergie n'est pas très malin sans compter les 53W qui chauffent le local, alors que nous sommes en pleine canicule ! Par conséquent, quand une alimentation de laboratoire quelconque fournit de l'énergie et que la **LED 1** s'illumine, *(Car contrairement à ce que l'on voit en **9** la ligne qui va à la LED est branchée.)* le connecteur **6** étant en

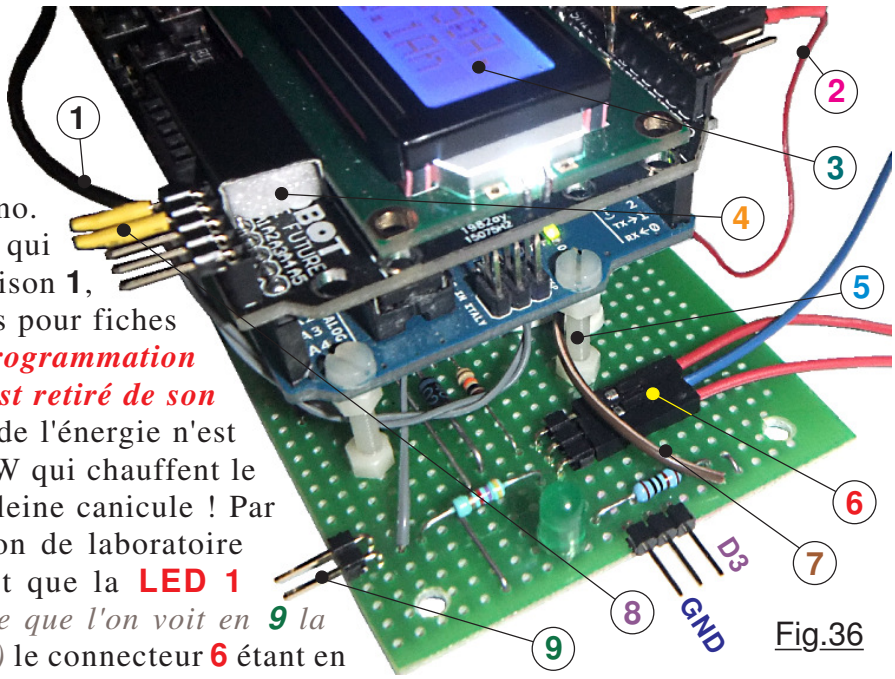


Fig.36

place. On commence par vérifier que la fiche qui va vers Arduino est correctement sous tension et surtout que ***le positif est bien sur son centre***. On peut alors l'enficher dans sa prise, Arduino prend vie. Avant cette manipulation, on a inséré le SHIELD. Bien avant, vous avez téléchargé le programme idoine, l'écran de l'afficheur **3** doit alors réagir normalement. Comme souvent dans ces lignes, faisons un petit tour du propriétaire : La carte Arduino est rigidement solidarisée sur le circuit imprimé principal par les boulons ϕ M3 en nylon **5**. Notez que la LED rouge du SHIELD qui témoigne de son alimentation normale est à mon avis bien trop présente et gène la lecture de l'afficheur, surtout quand on passe l'ensemble en veille et que le rétro-éclairage est éteint. En **4** un petit morceau de mousse synthétique bloque un carton noir qui masque cet éclairage trop vif. Le petit fil **7** va comme pour **2** sur le connecteur HE14 coudé pour la liaison avec **D3**. Enfin, l'autre connecteur coudé **8** qui semble soudé "à l'envers" est également inséré sur le SHIELD. Notez que lors de ces essais le fil du petit connecteur HE14 qui va sur **A3** n'est pas encore soudé car cette photographie remonte aux débuts de la réalisation matérielle du projet. Le module thermistance n'était pas encore envisagé. Avec la Fig.37 on s'octroi "un petit dernier pour la route". Les essais sont entièrement conformes aux prévisions, on peut tout tasser dans la boiboite. On va enfin passer au plaisir subtil de programmer en C++ le microcontrôleur et procéder à des mesures et des calibrages précis.

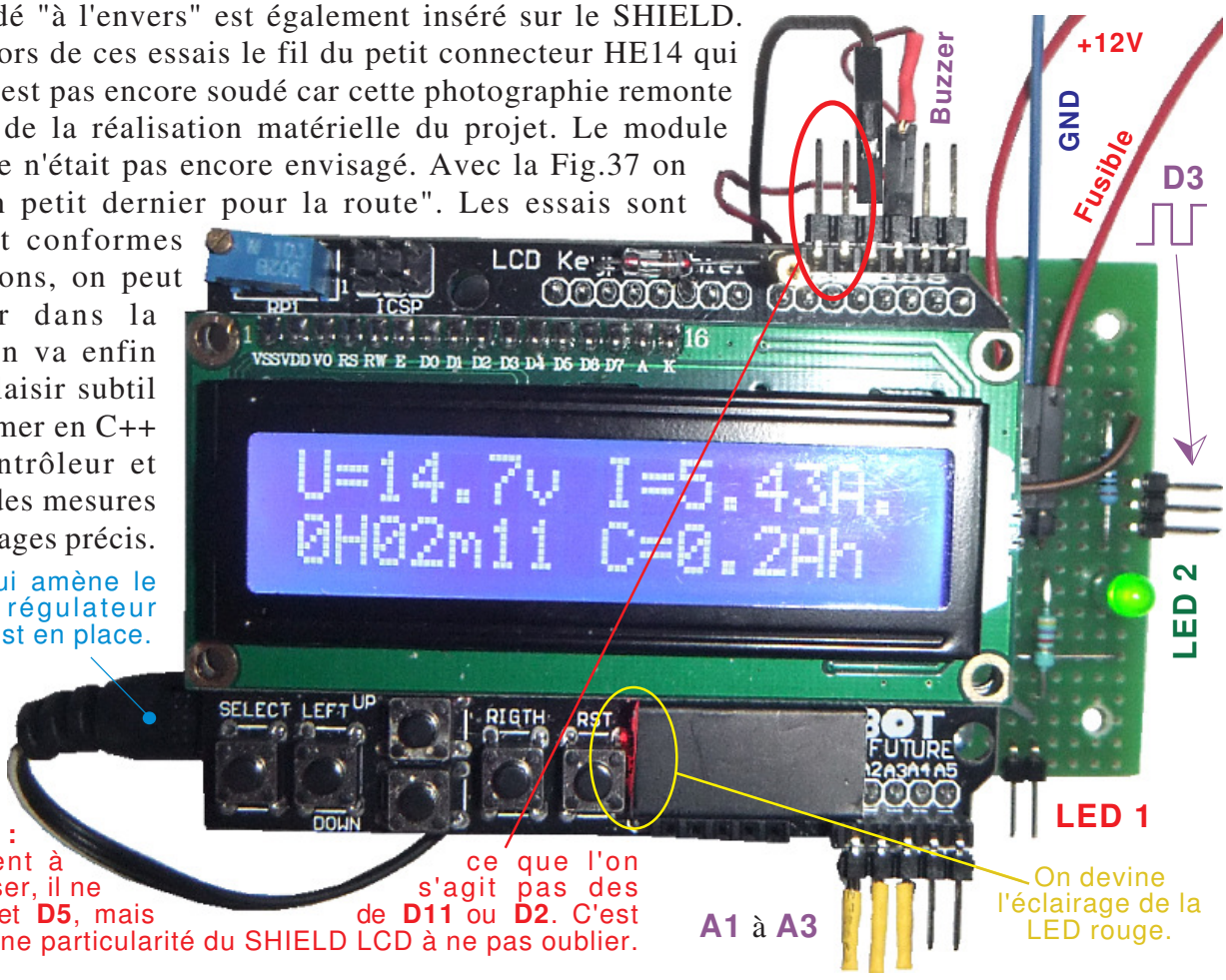


Fig.37

La fiche qui amène le +12V au régulateur d'Arduino est en place.

On devine l'éclairage de la LED rouge.

ATTENTION : Contrairement à ce que l'on pourrait penser, il ne s'agit pas des broches **D4** et **D5**, mais de **D11** ou **D2**. C'est une particularité du SHIELD LCD à ne pas oublier.

13) Galerie d'images pour illustrer ce descriptif :

Infiniment plus évocatrices que bien des développements textuels laborieux, quelques photographies permettent de révéler facilement une foule de détails dont il n'a pas été forcément question dans le tutoriel. L'assemblage matériel dans ce but s'achèvera sur quelques vues, complété par un minimum de commentaires. Sur la Fig.38 on a bien dégagé toutes les lignes électriques et les connecteurs pour insérer l'accastillage qui supportera l'ensemble électronique. Tout ce qui est actuellement présent dans le boitier a été finement vérifié. OUF, en Fig.42 page suivante on constate que tout est branché et fonctionne conformément aux prévisions. On va enfin pouvoir élaborer un programme séduisant et précis. Sur la Fig.39 orientée pour bien montrer les douilles de branchement et le support de fusible, on peut observer que le fil souple qui alimente la carte Arduino dépasse un peu par l'ouverture à l'arrière. C'est strictement sans importance. L'encadré de la Fig.40 effectué en gros plan permet de vérifier que par cette ouverture on peut aisément brancher la fiche USB pour travailler la programmation. Enfin sur la Fig.41 on observe les détails situés sur la semelle du coffret.

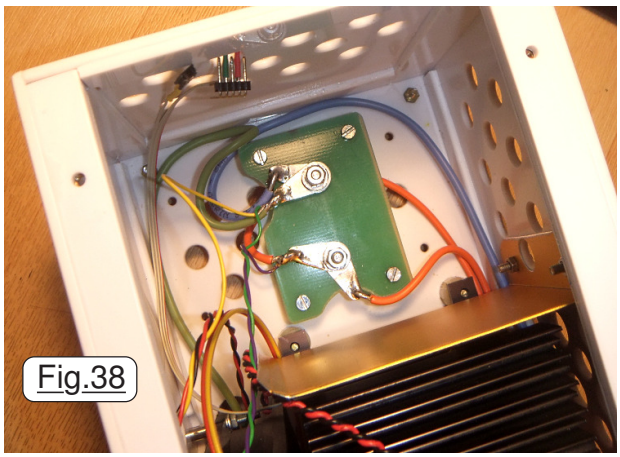


Fig.38

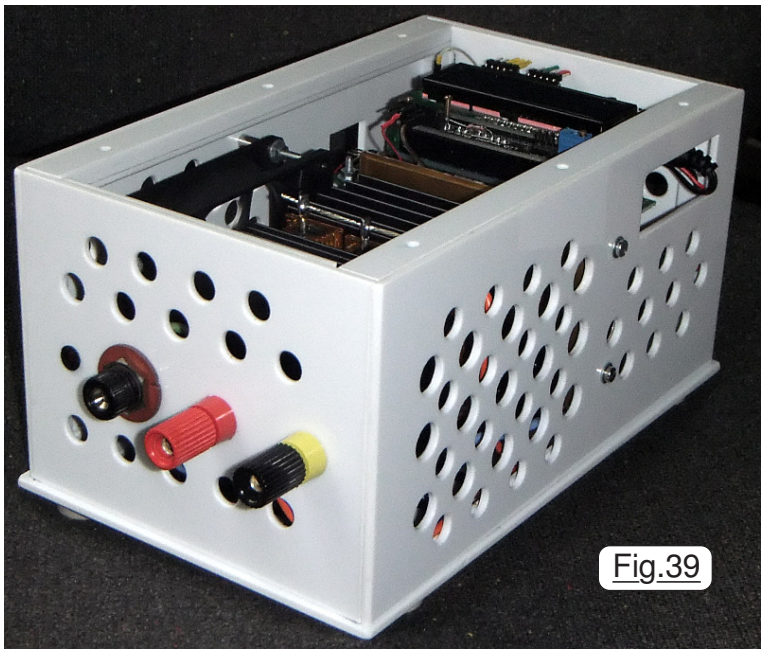


Fig.39

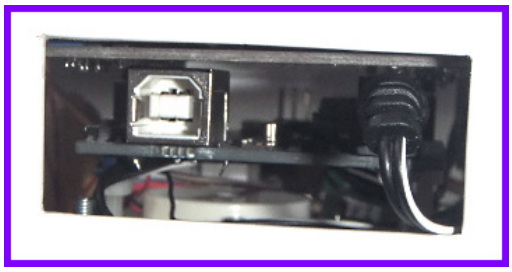


Fig.40

On distingue nettement les rondelles larges de liaison avec le bloc thermique. Est également bien visible le pied en caoutchouc décalé vers l'intérieur pour ne pas toucher la rondelle large. En **X** et **Y** nous avons les deux gros orifices qui facilitent la mise en place des écrous qui maintiennent à sa place le ventilateur. Il ne reste plus qu'à refermer avec le couvercle : Mission parfaitement remplie.

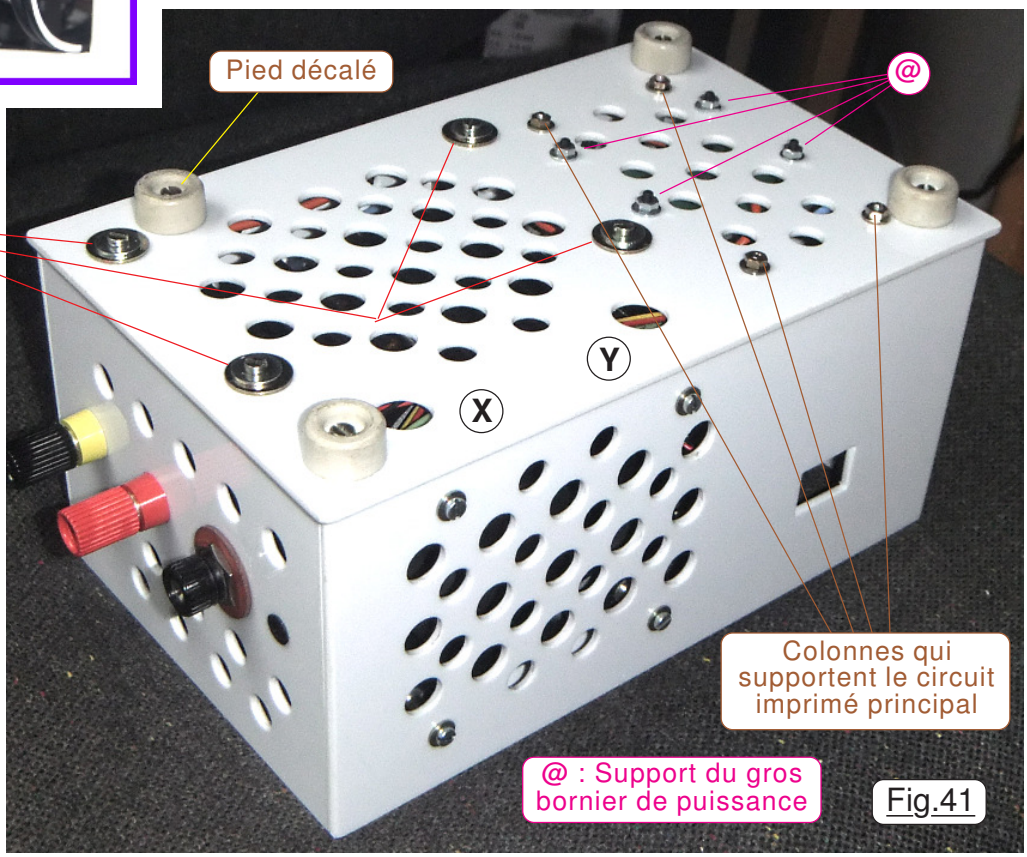


Fig.41

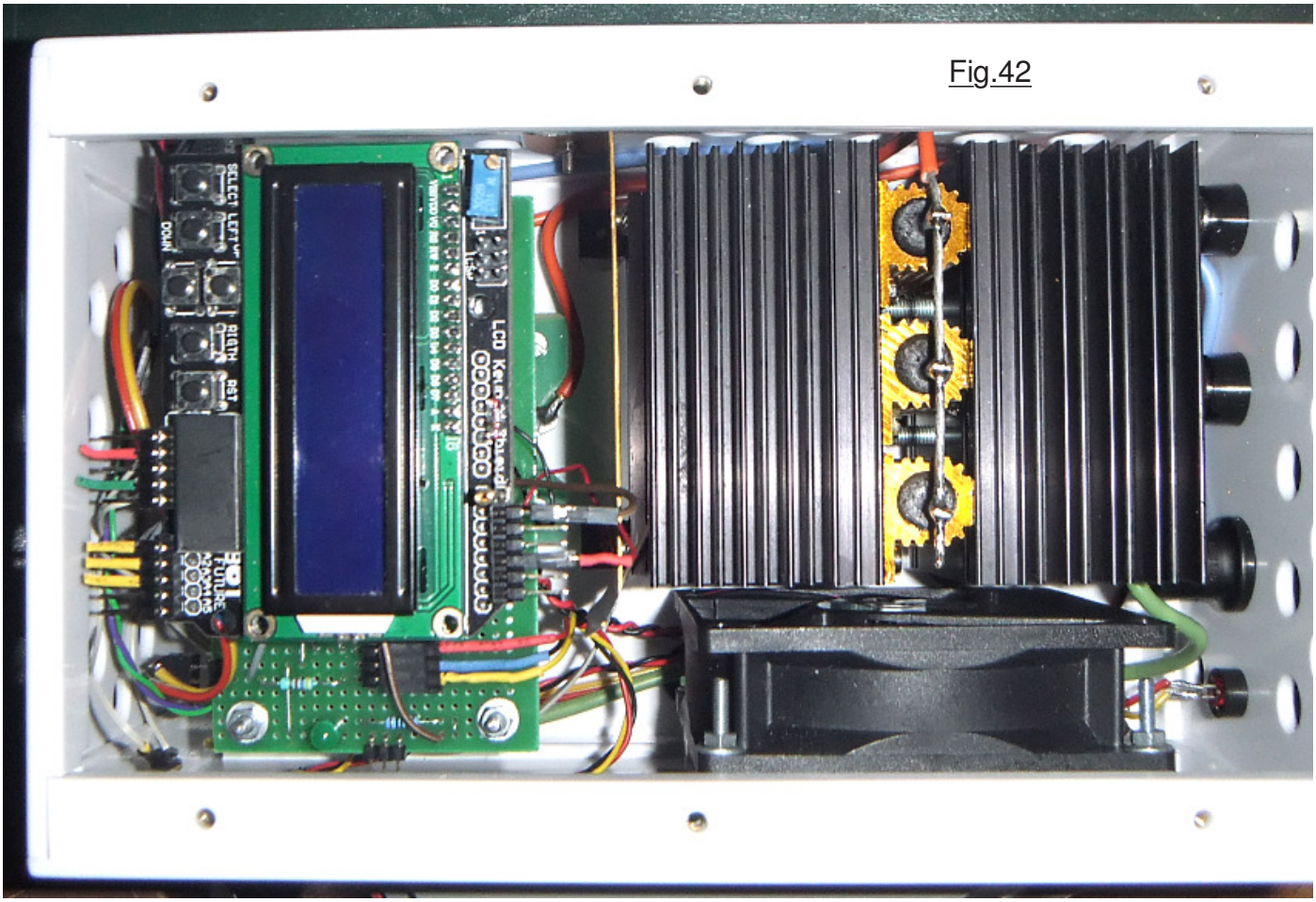


Fig.42

14) Dernière minute :

Rares sont les projets pour lesquels la dernière soudure marque la fin de la phase matérielle, c'est à dire où l'on a vraiment pensé à tout. On a beau analyser, vérifier, contrôler, passer en revus le moindre détail des schémas de câblage, généralement se glisse un petit lutin. Le détail insignifiant, facile à réparer mais ... qui oblige à démonter les éléments les plus délicats à mettre en place pour corriger la vermine insidieuse. Ben ici aussi un gai luron est venu nous tourmenter. Si vous observez attentivement la Fig.12 en page 8 vous constaterez la présence d'un petit connecteur à deux picots nommé **GND (*)** alors que l'astérisque ne devrait pas figurer et la couleur pour **GND** est bleue en standard. Quand le couvercle à été refermé, le logiciel de base dument téléchargé, immédiatement l'ATmega328 à présenté un comportement étrange. Comme nous étions en phase de développement du programme, la ligne USB alimentait la carte Arduino UNO et brancher la ligne **Alim Arduino** était parfaitement inutile. Pour des raisons purement philosophiques elle n'était pas branchée. Dès que la fiche a été insérée dans la prise dédiée, le fonctionnement est redevenu normal, comme sur le banc d'essai. Une analyse de la situation a rapidement montré qu'il n'y avait pas de ligne **GND** entre le circuit électronique principal et la carte Arduino. Après réflexion, la solution envisagée consiste à ajouter en **1** les deux picots verts **GND (*)** et à souder, comme montré sur la Fig.43 un fil sur le picot de masse du connecteur ICSP **2** de la petite carte électronique. *(Ayant un petit connecteur HE14 double, disponible, conserver les deux picots n'engage à rien bien qu'une seule broche soit en service dans l'appareil, autant le souder comme tel.)*

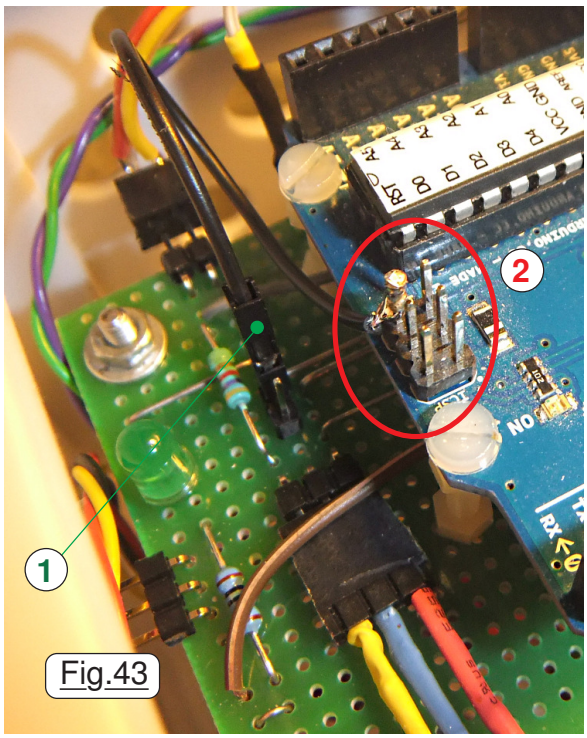


Fig.43

15) Développement du logiciel qui gèrera cet appareil de mesures :

Plusieurs petits programmes sont fournis en C++ écrits à l'aide de l'**IDE**, bien que seul P10_Programme_COMPLET.ino sera au final téléchargé dans la carte Arduino UNO. Ces petits démonstrateurs intermédiaires permettent au programmeur de mettre au point séparément certaines séquences qui ensuite viennent compléter le logiciel "définitif". Comme déjà annoncé au début de ce didacticiel, nous allons pousser l'analyse et la précision des mesures exagérément par rapport à l'utilité pratique de cet appareil. Cette recherche n'a pas d'autre justification que "la beauté du geste". Une étude conduite à outrance juste pour le plaisir. Commencez par enlever le fusible du support, la plus grande partie du programme sera développée sans avoir à gaspiller de l'électricité et chauffer inutilement notre laboratoire. Par contre, bien que la carte Arduino soit alimentée par la prise USB, si vous n'avez pas effectué la correction décrite en bas de la page 19, il faut laisser branchée la fiche du **+12V** car c'est elle qui alors amène **GND** Arduino sur **GND** du circuit imprimé principal.

Le démonstrateur NOYAU de base.

Fidèle à une technique qui a fait ses preuves depuis belle lurette, dès le début de développement d'un projet qui risque d'engloutir pas mal d'heures en programmation, un logiciel élémentaire portant la référence "double zéro" est mis en place. Nommé P00_PGM_de_base.ino, ce qui n'est pas très original, et surtout évocateur de sa mission, ce démonstrateur joue un rôle crucial pour faciliter le développement logiciel. Petit programme minimal pour écrire les routines complexes, il évacue les diverses options à valider sur RESET et ne comporte que les procédures indispensables à la séquence en cours d'étude. Ainsi la lecture du source est plus aisée, évitant la recherche d'une erreur dans un listage de plusieurs centaines de lignes. Quand le module logiciel est mis au point, il est alors intégré au programme complet, ainsi que ses subroutines dédiées. Naturellement, ce démonstrateur intègre la gestion de l'afficher LCD ainsi que la procédure d'exploitation du clavier. Les déclarations de constantes et de variables ainsi que l'affectation des E/S sont cumulées au fur et à mesure que les séquences testées et mises au point vont enrichir le logiciel complet. Ainsi il y a concordance permanente entre ce petit démonstrateur et le "programme définitif". Il vous est livré dans sa configuration en fin de développement pour un éventuel usage personnel.

Mise en service du module afficheur LCD.

Premier programme écrit en C++ pour appréhender le matériel, P01_Test_du_clavier_du_SHIELD.ino est autonome. Poliment il dit ">>> Bonjour. <<<". Ensuite, chaque action sur une touche est suivie d'un BIP un peu agressif et indique en texte sur la ligne du bas, quel petit bouton vient d'être sollicité. L'écran est configuré en mode "lumineux". Bien que rudimentaire, ce petit démonstrateur met en place la gestion du module LCD et de son afficheur à deux lignes. Il permet de tester le bon fonctionnement du SHIELD et confirme la pertinence des branchements relatifs à la carte électronique insérée en gigogne sur les connecteurs HE14 d'utilisation d'Arduino UNO. C'est un bon début, on peut passer à du plus consistant.

La Fig.44 résume la façon assez classique dont est géré un clavier à plusieurs boutons en n'utilisant qu'une seule entrée analogique. Autant de résistances que de poussoirs forment un diviseur de tension. Compte tenu des valeurs utilisées sur le SHIELD on obtient dans notre cas les tensions indiquée en violet. En bleu clair la valeur numérique que retourne le CAN. *(Convertisseur Analogique Numérique.)* Pour différencier chaque bouton on définit des seuils situés entre les deux valeurs numériques. La procédure void Tester_le_BP_A0() effectue la lecture du clavier. Si aucun **BP** n'est activé on en sort immédiatement pour ne pas pénaliser la boucle de base. Si un contact est détecté, la routine effectue l'anti-rebonds et filtre la touche utilisée. ***Cette technique ne permet pas de gérer plusieurs touches activées simultanément.***

Shéma de gestion des B.P.

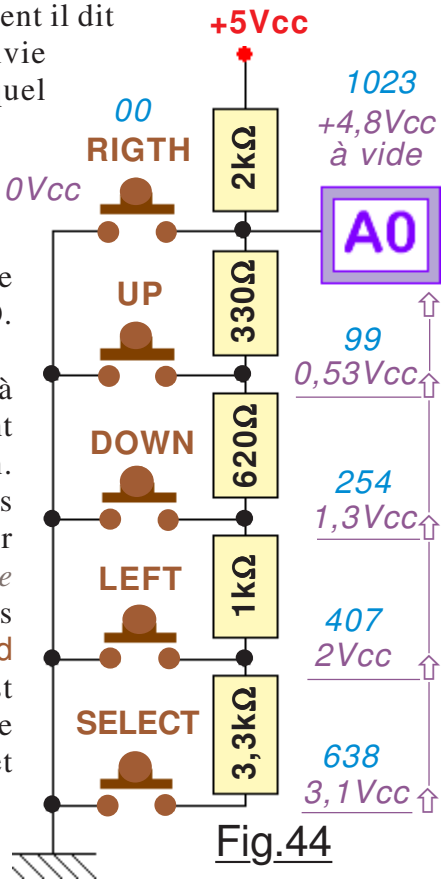


Fig.44

Mesurer la température du bloc thermique.

Résistance de consommation de courant et dissipateur de chaleur pourraient sans grand danger accepter de bien plus fortes températures que celles qui seront présentes quand la stabilité thermique sera obtenue, ventilateur d'extraction en service. Toutefois, sur panne de ce dernier, si le bloc thermique pourrait sans incident monter jusqu'au maximum d'environ 94°C en statique, le coffret en matière thermoplastique n'apprécierait pas vraiment. Mesurer la température du radiateur en permanence pour déclencher une alerte en cas d'incident devient une mesure de sauvegarde "prioritaire". Avant d'aborder le calcul de la capacité, on va rapidement balayer la façon dont sera "tâtée" et calculée la température mesurée. Pour l'alerte nous verrons plus avant.

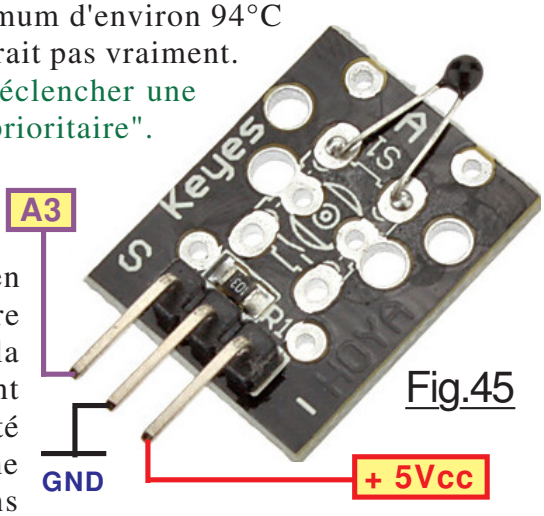


Fig.45

Le composant qui va transformer la température de contact en une valeur "électrique" est une ***thermistance*** CTN, c'est à dire un corps d'épreuve dont la ***résistance*** varie en fonction de la température à laquelle il est soumis. CTN, car contrairement au comportement banal des composants ordinaires en électricité qui voient leur résistance augmenter avec leur température, une CTN présente la particularité inverse. Quand nous serons capables de mesurer la valeur de la température du bloc de dissipation d'énergie, la traiter dans le programme complet sera facile, soit de la programmation "standard". Pour débroussailler le problème, P02_mesure_la_temperature.ino, également autonome, remplit cette tâche finement et va permettre de détailler quelques particularités. Revenons à notre Coefficient de Température Négatif.

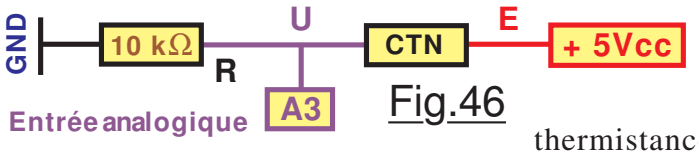


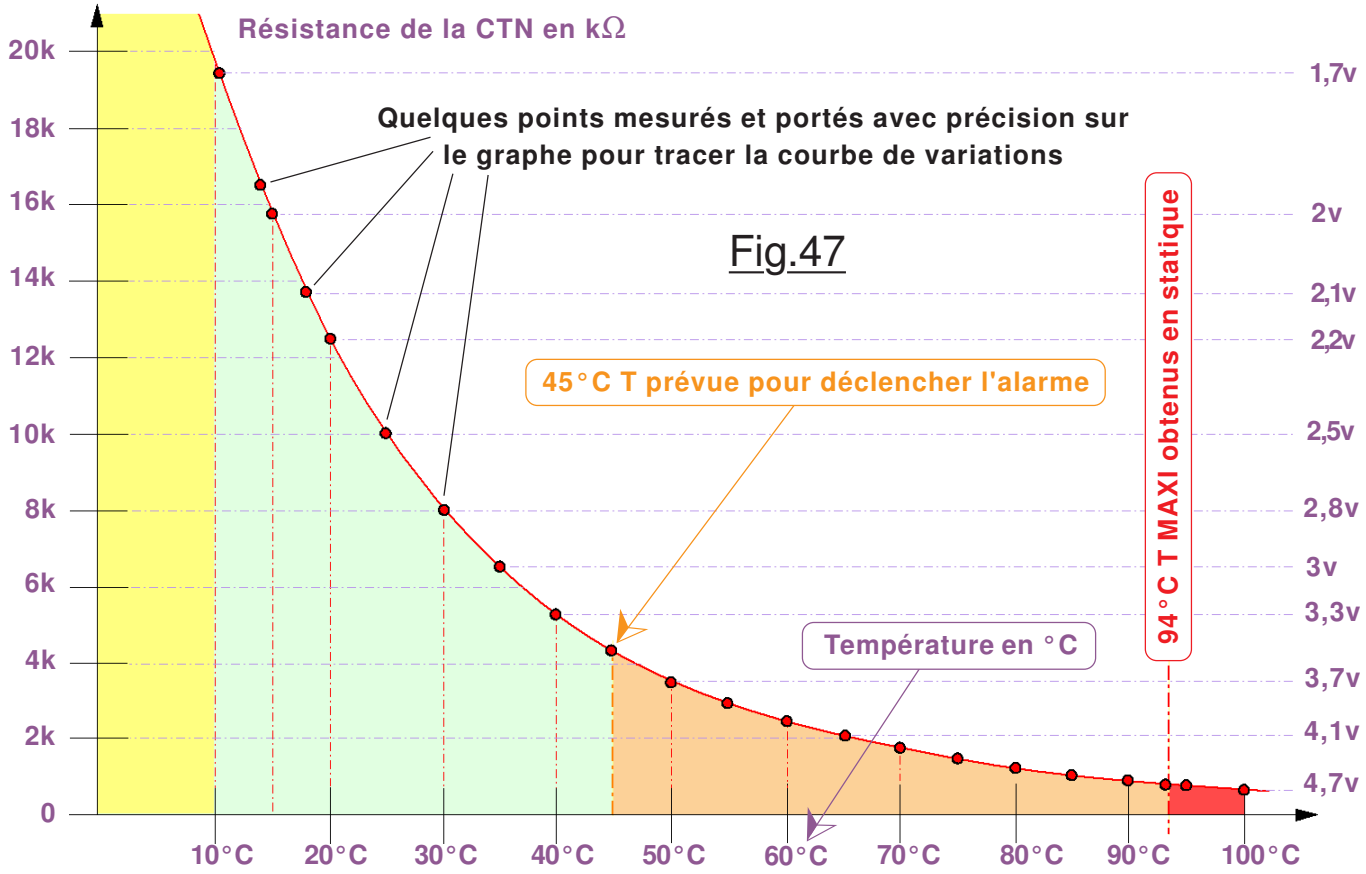
Fig.46

Incontestablement le petit module très courant dans le commerce en ligne dédié à Arduino, représenté sur la Fig.45 et équipé d'une thermistance **CTN**, est l'un des plus rudimentaires qui soit proposé comme module pour Arduino sur le marché. Comme on peut le voir sur la Fig.46 il est constitué d'un résistor de 10kΩ mis en série avec la thermistance à coefficient de température négatif. Le schéma du petit circuit imprimé se résume à deux fois rien. La Fig.45 précise les branchements à effectuer pour interfacer avec la carte ARDUINO sur l'entrée analogique **A3**. Attention, contrairement à ce que laisse présager la sérigraphie, il faut impérativement brancher le +5Vcc sur la broche réputée - et **GND** au centre. Mesurer la valeur de la thermistance n'est pas difficile puisqu'il s'agit d'un simple dispositif potentiométrique. La tension mesurée étant **U**, on peut facilement en déduire **CTN** en fonction de **R** :

Avec R = 10000 et E = 5Vcc.

$$\frac{E}{U} = \frac{R + CTN}{R} \Rightarrow \frac{E * R}{U} = R + CTN \Rightarrow \boxed{CTN = \frac{E * R}{U} - R}$$

Petit détail scabreux : La valeur de **R** nous est assez indifférente, c'est la température qui à nos yeux présente de l'intérêt. Bien que l'on pourrait royalement se contenter des informations données par le fournisseur de la CTN, car il maitrise avec précision la caractéristique de ses composants qui sont tous identiques à une dispersion dérisoire, pour une approche rigoureuse, des mesures physiques ont été réalisées ici, thermomètre logé dans les ailettes du radiateur. Les résistances de charge ont été soumises à une tension de 12V, le dissipateur n'étant pas ventilé. Ça chauffe, ça chauffe. Partant de la température ambiante, un grand nombre de points ont été mesurés. Sur la Fig.47 ont été sélectionnés certains d'entre eux pour répartir les points figuratifs et tracer la courbe de variation. *(Petits disques rouges bordés de noir.)* Manifestement, la variation n'est pas linéaire, et c'est bien dommage car la ***formule de transformation Résistance vers Température*** serait élémentaire. Le graphe est à deux entrées. À gauche nous avons la valeur de **CTN** en fonction de la température précisée en abscisse. À droite la tension qui en résulte sur **R**. Quelle que soit la température, la résistance du composant ne deviendra jamais nulle, elle tend vers une limite basse qui se situe vers 600 ohms. Normal, pour une température nulle il faut descendre vers le zéro absolu pour des comportements ordinaires. Ici il faudrait une chaleur solaire, il y aura longtemps que le composant se sera volatilisé ! Quand il fait froid, la résistance devient infinie, les mathématiciens parlent alors d'asymptote. Le pire, c'est que ces derniers parlent pour la variation visualisée d'une évolution logarithmique. Berkkkkkkk !



Artistiquement, que la courbe représentative soit logarithmique, n'est pas du tout moche. Elle est assez jolie la belle pente rouge sur la Fig.47, il serait presque judicieux de l'encadrer. Le problème, c'est le langage C++ du compilateur d'Arduino à sa naissance ne sait pas ce qu'est une "fonction log". Heureusement pour nous, des acharnés passionnés ont créé une bibliothèque mathématique qui inclus cette étrange chose. Il suffit dans le programme qui aura besoin de cette routine mathématique de déclarer la bibliothèque : #include <math.h>
De plus, nous avons de la chance, car maintenant ce petit trésor fait partie intégrante de l'**IDE** et nous n'avons pas besoin d'aller le chercher sur Internet et de le déclarer. Génial non ?

Établir la liaison entre **U** sur **A3** et la valeur de la température exprimée en degrés Celsius devient un jeu d'enfant. Analysons les séquences de programme qui se chargent de cette phase du logiciel. Deux lignes de programme traduisent globalement les actions qui se succèdent :

```
① int Mesure_NUM_CTN = analogRead(Capteur_KY_013);
② Temperature = Conversion_en_degres(Mesure_NUM_CTN);
```

La ligne ① procède sur **A3** nommée Capteur_KY_013 à la numérisation dans la variable Mesure_NUM_CTN de la tension **U**. La valeur est un int car peut aller jusqu'à 1023. Vous avez déjà compris qu'en ② la fonction Conversion_en_degres effectue tout le travail de transposition. Examinons cette séquence de programme :

```
③ double Conversion_en_degres(int Mesure_CTN) {
④   double Tmp_Kelvin;
⑤   Tmp_Kelvin = log(10000.0 * ((1023.0 / Mesure_CTN - 1)));
⑥   Tmp_Kelvin = 1 / (0.001129148 + (0.000234125 +
⑦                    (0.0000000876741 * Tmp_Kelvin * Tmp_Kelvin ))* Tmp_Kelvin );
⑧   Tmp_Kelvin = Tmp_Kelvin - 273.15; // Conversion des Kelvin en degrés Celcius.
⑨   return Tmp_Kelvin; }
```

La ligne ③ précise avec double qu'il s'agit d'une fonction qui retournera un réel codé sur 64 Bits. Notez que la séquence va utiliser la variable Mesure_CTN qui en réalité prend la valeur de Mesure_NUM_CTN qui est passée en paramètre lors de l'appel du calcul en ②. Dans la séquence étudiée on déclare en ④ la variable Tmp_Kelvin qui va subir diverses transformations. Bien que pas très longue, la ligne de programme ⑤ effectue un gros travail. La valeur 10000.0 représente la

 résistance **R** exprimée en Ω. Pas compliqué de comprendre que 1023.0 représente la

plus grande valeur numérisée quand U arrive à exactement +5Vcc. Pour le reste de la formule il faut titiller un peu les équations. Si l'on désire établir la relation entre la valeur numérisée et la tension U il faut employer la proportionnalité : U = Mesure_CTN * 5 / 1023. On reporte dans la formule encadrée qui devient :

$$CTN = \frac{E * R}{U} - R$$

$$CTN = \frac{\cancel{5} * 10000 * 1023}{Mesure\_CTN * \cancel{5}} - 10000 \Rightarrow CTN = 10000 * \left(\frac{1023}{Mesure\_CTN} - 1\right)$$

Pour simplifier cette forme, on met le 10000 en facteur commun, la formule devient celle située dans l'encadré colorié en rose. C'est exactement ce qui est codé dans la ligne ⑤ qui en outre transforme la valeur par la fonction log() pour tenir compte de l'évolution logarithmique de CTN sous l'influence de sa température. La ligne ⑥ tronquée dans la mise en page et complétée par ⑦ et correspond à la caractéristique intrinsèque du composant, comportement garanti par le fournisseur avec une bonne précision. Comme ce calcul de conversion de CTN vers température est effectué en degrés Kelvin, la ligne de calcul suivante ⑧ transforme en degrés Celsius. OUF !
Enfin, la dernière ligne de code ⑨ retourne le résultat sous forme d'un réel exprimé en °C.

Misère vont gémir certains à juste titre, qui ne désiraient pas des trifouilleries si compliquées, qui n'ont pas de thermomètre qui "monte aussi haut", ni d'alimentation de laboratoire pour torturer le bloc thermique, ***que vais-je devenir ?*** Pas de panique, contentez-vous d'ignorer ce paragraphe scandaleux. Le module que vous approvisionnerez présentera par "construction" des caractéristiques pratiquement identiques à celles du prototype. L'imprécision qui résultera de la dispersion de caractéristiques sera absolument négligeable. Vous téléchargez le programme et vogue la galère ...
Pour ceux qui désireraient modifier le logiciel et en faire une version locale, par exemple changer la température d'alarme, le tableau donné ci-dessous donne la correspondance entre la température et les valeurs retournées par la numérisation dans la variable Mesure_NUM_CTN.

T °C	CAN	U *en v*	CTN kΩ	T °C	CAN	U *en v*	CTN kΩ
10,5	348	1,7	19,4	55	788	3,85	3
14	387	1,89	16,4	60	820	4	2,5
15	398	1,94	15,7	65	847	4,14	2,1
18	432	2,11	13,7	70	872	4,26	1,7
20	455	2,22	12,5	75	892	4,36	1,5
25	512	2,5	10	80	910	4,45	1,2
30	567	2,8	8	85	925	4,52	1
35	619	3	6,5	90	938	4,58	0,9
40	668	3,26	5,3	93,4	946	4,62	0,8
45	713	3,48	4,3	95	949	4,64	0,78
50	753	3,68	3,6	100	959	4,69	0,67

Les lignes de code source présentées ci-avant pour détailler la façon dont sont soumis les traitements à l'ATmega328 pour gérer la température sont extraites du petit programme démonstrateur P03_mesure_la_temperature.ino. On ne peut faire plus compact. Au début les déclarations diverses, l'initialisation du matériel. Puis, la boucle de base effectue immuablement une conversion analogique numérique, soumet le résultat à la procédure de calculs, et enfin affiche le résultat sur l'écran LCD. Enfin, après une temporisation de 0,5S elle recommence indéfiniment.

Envisageons le pire des cas : Vous n'arrivez pas à vous procurer le petit module KY-013 spécifique à Arduino pour mesurer des températures. Approvisionnez une thermistance CTN quelconque, peu importe le modèle, soudez-la sur un petit circuit imprimé avec une résistance de 10kΩ, les deux étant branchées conformément au schéma de la Fig.46 pour respecter le sens de variation des CAN. Puis procédez aux essais qui vous permettront de déterminer les valeurs critiques pour le programme, en l'occurrence celle de la température à l'équilibre thermique. Combien même la température calculée ne serait pas rigoureuse car votre composant n'aurait pas des caractéristiques identiques à celles du module KY-013, l'important consiste à détecter une surchauffe. Peu importe la température annoncée. Pour le seuil qui déclenchera une alerte, adoptez la valeur de ***l'équilibre thermique ventilé*** et ajoutez environ sept à dix degrés Celsius. Problème résolu !

16) Les concepts généraux du programme : Mesure de la capacité :

Justifier et clarifier point par point des séquences particulières conduira inévitablement à une impasse si le fonctionnement global du programme n'est pas parfaitement analysé. Ce chapitre fait un petit tour d'horizon pour mettre en place le scénario global. Ensuite seulement on creusera plus finement les entrailles du logiciel pour en accaparer certaines subtilités.

Principe fondamental de mesure de la capacité d'un accumulateur.

Capacité précise généralement la caractéristique de base d'un Condensateur en électricité et s'exprime en Farads, ou plus exactement des sous multiples du genre µF. Dans le cas moins courant d'une batterie d'accumulateur, le vocable fait référence à ***Capable de fournir de l'énergie***. Ce thème a été abordé dans le chapitre n°01 et fourni avec des exemples numériques. La capacité d'une batterie est à comparer à celle d'un réservoir. Ce dernier est rempli avec des molécules d'eau. Pour l'accumulateur, ce sont des électrons qui sont "entassés". Ouvrez le robinet, le réservoir va se vider plus ou moins rapidement en fonction du débit. Pour un réservoir électrique, il en va strictement de même. Le tuyau devient un circuit filaire conducteur qui va d'une borne à l'autre. Le robinet, une résistance qui freine plus ou moins le débit d'électrons qui pour la circonstance prend le nom de courant ou d'**Intensité** et s'exprime en Ampère. **Pour calculer la capacité** d'un réservoir quand on ne peut mesurer ses dimensions, on multiplie le débit du fluide par le temps qu'il faut pour le vider. *(Simplifions à outrance et supposons ici que ce débit est constant comme le supposait l'instituteur à l'école primaire quand on calculait le temps mis pour se vider par un lavabo qui fuit !)* Dans le cas d'un réservoir électrique, le calcul est totalement analogue :

Capacité de l'accumulateur = Intensité débitée x Temps mis pour la décharge

Notez que si on trouve que vider le réservoir va prendre trop de temps, rien n'interdit en cours de processus, de noter ce qui a été actuellement prélevé, d'augmenter le débit, et de prendre en compte cette nouvelle valeur pour calculer ce qui restait. C'est ici que l'équivalent informatique va se montrer d'une grande souplesse. À chaque seconde, *(Intervalle de temps choisi par le programmeur pour simplifier les calculs.)* le logiciel devra mesurer avec précision l'Intensité. Puis il calculera "le nombre d'électrons" qui sont passés dans le circuit. Il ajoutera ce petit paquet à un total cumulé qui par nature correspondra en fin de décharge à la Capacité de l'accumulateur testé. Chaque petit paquet correspondra à une Ampère Seconde si le courant est exprimé en Ampères. Comme on désire présenter la capacité en Ampères Heures, il suffira de diviser le cumul par 3600 à la fin du processus, ou pour chaque "petit paquet" ce qui permet d'afficher en permanence la capacité actuellement confirmée. Mieux que tout ce verbiage, l'organigramme de la Fig.48 résume le principe de base du processus mis en œuvre dans ce projet. Quand le mesurage débute pour un nouveau test, le nombre d'électrons qui ont été prélevés est nul, donc la capacité est annulée en *(1)*. En *(2)* la précision de la mesure sera directement impactée par le respect rigoureux d'exactement UNE SECONDE. Pas besoin d'insister sur le soin qui devra être apporté à la précision et la finesse en *(3)* pour déterminer la valeur instantanée du courant. Notez que "Mesurer" est mis entre guillemets car nous devrons y revenir. Enfin en *(4)* on ne fait que cumuler à chaque seconde le petit paquet d'électrons qui a traversé le robinet électrique.

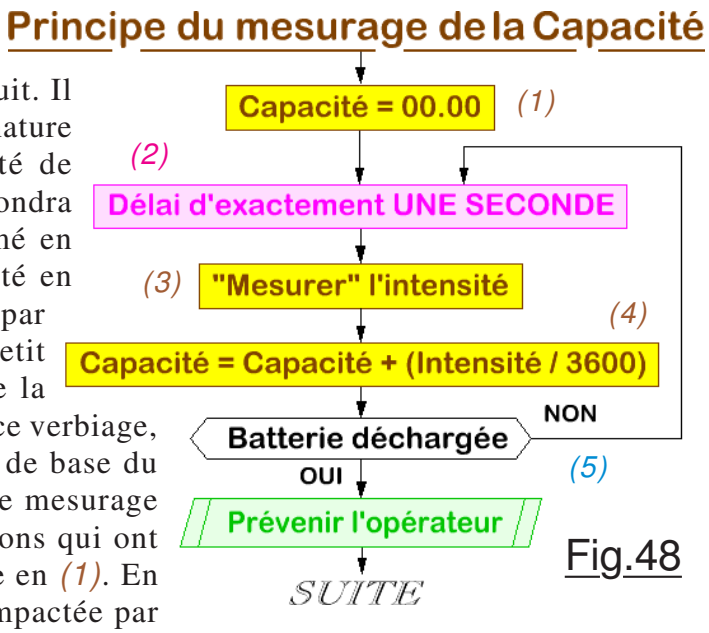


Fig.48

Pour déterminer le moment où l'accumulateur sera considéré comme entièrement déchargé, on mesure à chaque fois la tension qu'il présente à ses bornes. Au nominal, c'est à dire pendant une très grande majorité de son utilisation, la tension avoisine 12V. Quand la batterie est globalement déchargée, assez rapidement la tension s'effondre. À 11V on peut déjà considérer qu'elle est vidée, ***et il importe impérativement de la recharger*** sans trainer. Pour ménager une marge de sécurité, dans le programme la valeur adoptée est de 10,8V sachant que pour passer de 11V à 10,8V sur l'élément de la Fig.1 il ne faut que trois minutes, la résistance interne à ce stade augmentant assez rapidement.

Délai d'exactement UNE SECONDE entre deux mesures.

Fondamental en électronique, en informatique, en robotique, calibrer le temps qui s'écoule est tellement impératif que tous les microcontrôleurs actuels sont pourvus de ressources internes dédiées à cette préoccupation. Exemple : un servomoteur est piloté en envoyant à son asservissement des impulsions de fréquence et de durées calibrées. Nous n'aurons vraiment aucune difficulté à gérer ce paramètre temporel. Dérivé directement de la Fig.48, l'organigramme de la Fig.49 présente en *(5)* la valeur du test et surtout montre en *(7)* qu'à chaque mesure l'écran LCD sera rafraichi pour présenter les nouvelles valeurs du moment. En fonction des options imposées par l'opérateur, il sera possible d'indiquer la température du radiateur, la tension actuelle aux bornes de la batterie et bien d'autres paramètres encore. En fonction de ce qui sera écrit sur l'écran LCD, l'ATmega328 prendra des durées différentes. Les calculs pour afficher la valeur de la température ainsi que ceux pour déterminer la grandeur du courant débité également. Pourtant, toute la boucle située entre *(2)* et son retour en *(6)* devra présenter une durée d'exactement UNE SECONDE quel que soit le temps passé dans les innombrables instructions situées entre les deux. Avec tout ce que cache le microcontrôleur de la carte Arduino, nous avons à notre disposition plusieurs moyens pour gérer le temps. C'est la fonction millis() du langage C++ compilé de l'**IDE** qui va faciliter considérablement le chronométrage de la boucle "rose" sans avoir à tenir compte des variations de temps consommés par les instructions situées dans la boucle. Elle fait appel à l'une des particularités de l'ATmega328. Sur sa puce électronique est implanté un compteur binaire totalement indépendant qui est incrémenté par l'horloge interne du circuit intégré mille fois par seconde. Sur un RESET, ce compteur est remis à zéro. L'électronique de ce compteur intègre "32 Bits", il peut aller jusqu'à 4294967295 sans repasser à zéro. Un rapide calcul montre que suite à un RESET il faut 40 jours pour arriver à le "remplir". Ce compteur contient le temps qui s'est écoulé depuis un RESET en mS, sa précision est considérable puisque c'est l'horloge à quartz de la carte électronique qui le pilote. La fonction millis() retourne sous la forme d'un unsigned long la valeur lue dans le compteur électronique. L'utilisation de cette facilité informatique est résumée sur la Fig.50 qui présente la séquence dédiée que l'on trouvera dans la boucle de base. Le reste de la boucle se charge vérifier si l'opérateur a sollicité le clavier, s'il faut déclencher une alerte etc. Considérons l'organigramme. L'instruction *(2)* prend la valeur actuelle du compteur et ajoute 1000. Sans trop se creuser les méninges, 1000mS fait bien UNE SECONDE. Lorsque le compteur dépasse cette valeur de Référence, le test en *(1)* devient positif et tout ce qui se trouve en *(3)* sera réalisé. À partir du moment où ces instructions consomment moins d'une seconde, le processus sera déclenché avec une précision d'une milli seconde. Ces explications ne sont valides que si la séquence *(3)* **ET** le reste de la boucle totalisent une durée inférieure à la seconde. Vu que l'ATmega328 cavale à 16MHz, bien que le programme complet semble "meu meu", l'ensemble des instructions déroulées quelle que soit la combinatoire logicielle reste de très loin inférieur à cette valeur critique et exactement UNE SECONDE sera parfaitement satisfait.

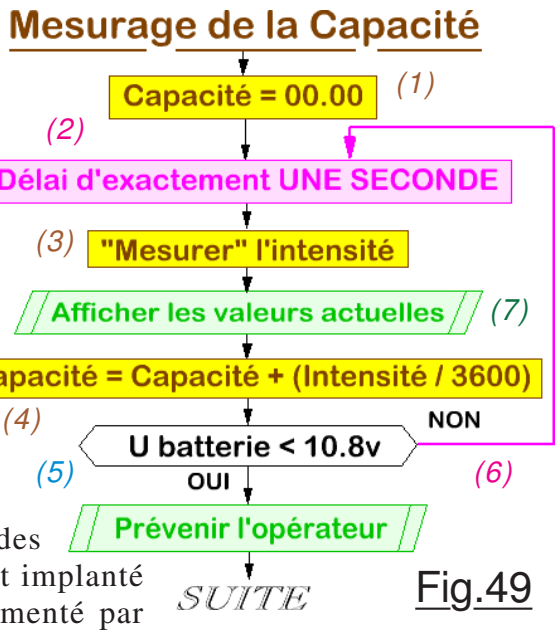


Fig.49

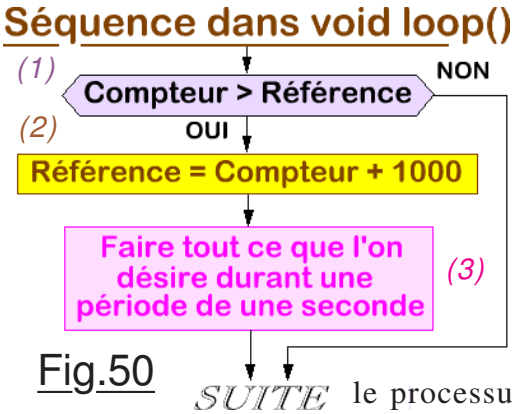


Fig.50

Ajustement précis de la durée de la boucle de base.

Comme nous allons le constater dans ce chapitre, le programmeur ne maîtrise pas toujours le comportement du microcontrôleur, et tout particulièrement quand son programme fait appel à des séquences dont il ignore complètement l'écriture et le déroulement. Ceux qui refusent de se prendre la tête pour des considérations techniques peuvent royalement ignorer ce chapitre, il ne s'adresse qu'à ceux qui aiment bien se creuser les méninges sans justification objective.

Donnant dans la facilité, les explications du chapitre précédent ont fait l'impasse sur des petits détails insignifiants. Vous avez noté sur la Fig.5 que la sortie binaire **D3** peut allumer une LED verte à notre convenance. Dans la séquence qui est déclenchée une fois par seconde, on inverse l'état de ce témoin lumineux. Ainsi on voit battre le cœur de la carte électronique, confirmant que le programme ne s'est pas perdu dans une quelconque boucle infinie. Logiquement, cette LED doit changer d'état exactement une fois par seconde. *(Sauf quand le programme attend que l'opérateur clique sur le clavier. Dans ce cas pour l'avertir elle clignote très rapidement.)* La sortie **D3** et **GND** sont disponibles sur un connecteur HE14. Possédant un périodemètre d'une précision à 10^{-7} il m'est possible de mesurer avec rigueur la cadence du clignotement. Dans le programme, au lieu d'ajouter exactement 1000 à Référence, on utilise une constante #define T_base_millisecondes.

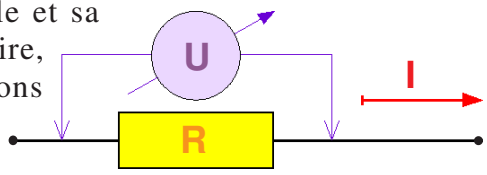
En ajustant finement sa valeur, on doit pouvoir aboutir à un temps de boucle principal à une milliseconde près donc compris entre 0,999S et 1,001S. Dans ce but, après avoir mesuré la durée exacte de la boucle quand l'équilibre thermique est atteint, *(Autant pour l'appareil de mesure que pour la carte Arduino.)* on corrige cette constante jusqu'à satisfaction. Par exemple si le temps mesuré pour la période est de 2,008215S, on en déduit que le comptage avec millis() prend 4mS de trop. En affectant la valeur 0.996 à la constante T_base_millisecondes on doit aboutir à une durée de cycle strictement égale à 2,000NNNS. Les décimales NNN sont aléatoires puisque l'on calibre la boucle à une ±1mS. Testant diverses valeurs, le tableau de la Fig.51 présente les résultats obtenus, les µS n'étant pas indiquées car non significatives. On constate qu'au mieux nous serons à ±3mS. Cette différence s'explique par divers phénomènes qui interviennent de façon discrète. Sans que le programmeur n'en soit forcément conscient, plusieurs instructions de l'**IDE** utilisent les interruptions. C'est en particulier le cas de millis() qui n'est pas forcément prioritaire. Par ailleurs il suffit que le test soit effectué entre deux comptages pour ajouter ou retrancher une unité. Il s'agit d'une relation de phase entre l'horloge quartz et la boucle de base au moment du test. Bref, l'imprécision sera de 3 pour 1000. Dans le pire des cas, c'est à dire pour une grosse batterie, le résultat affiché sera de 49,85Ah au lieu de 50,00Ah. Pas de quoi en faire une crise d'urticaire, d'autant plus que la valeur étant minimisée, nous sommes certains que l'élément testé fait au moins ce qui est affiché. Pour une batterie de 22Ah l'imprécision ne sera plus que de 0,07Ah car il y a proportionnalité.

T_base_millisecondes	T	ΔT
994	0.997	-3mS
995	1.003	+3mS
996	1.003	+3mS
997	1.003	+3mS

Fig.51

Mesurer l'intensité avec précision.

Physiquement, mesurer directement la grandeur d'un courant n'est pas élémentaire. On sait faire, en mesurant le champ magnétique qu'il engendre autour du conducteur électrique. Toutefois, cette technique est assez complexe et n'est pas très efficace pour les faibles intensités. Aussi, depuis les débuts de la "fée électricité", on se contente d'intercaler une résistance dans le circuit. La tension aux bornes de cette dernière est mesurée. La loi d'Ohm par le simple calcul **I = U / R** déduit le courant qui circule. Habituellement, la chute de tension **U** mesurée aux bornes du shunt **R** inséré dans le circuit perturbe ce dernier. Pour minimiser cette influence néfaste, on choisit pour **R** une résistance aussi réduite que possible. **U** est alors très faible et sa mesure impose un voltmètre particulièrement sensible. Au contraire, dans notre cas, **R** est de forte valeur et à ses bornes nous aurons globalement 12V. Les conditions sont idéales. N'importe quel technicien déduit de la formule utilisée, que la précision du calcul dépendra directement de celle de la valeur de **R** et de celle du CAN qui sur **A1** sera chargé d'évaluer **U**. Passons en revue les méthodes qui permettent d'affiner la détermination des paramètres pertinents. Les trois composants mis en parallèle sont réputés à 5%. *(La résistance théorique sera de 2,666...Ω)* Il faut également tenir compte de la chute de tension en ligne et sur le gros bornier. Pour déduire avec précision la valeur réelle de la résistance du circuit complet, on branche une alimentation de laboratoire capable de débiter 5A sous 12V. On mesure à l'extérieur avec un ampèremètre et un voltmètre précis I et U, et la loi d'Ohm permet de calculer la résistance réelle du circuit quand l'ensemble est entièrement câblé. Sur le prototype la valeur déduite fait 2,7Ω. Conclusion : Conduire ces essais n'était pas indispensable, les composants approvisionnés sont très précis et la perte en ligne



dérisoire. E***n tête de programme sont réunies les constantes que vous pourrez modifier à votre convenance***. Pour repérer facilement la liste, la première ligne de, celle qui indique la version du logiciel est complétée par //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ et dépasse les autres. Elle est ainsi facilement repérable. Pour pouvoir procéder à des essais en situation sans pour autant consommer 4,5A car vous ne disposez certainement pas d'une alimentation de laboratoire aussi musclée, *(Sans compter le gaspillage d'énergie)* il suffit d'enlever le fusible. Pour ne pas déclencher une ALERTE de "Pas de FUSIBLE !" il faut dans le programme désactiver le test. Une deuxième ligne mise en évidence par //@@@@@@@... peut donc être validée. Vous effacez les deux "/" pour valider goto Sauter_test_fusible; et le programme acceptera une tension élevée sur **A2**. Indiquer la valeur précise pour **R** se trouve dans la directive : **#define R_Bloc_thermique 2.7**.

Déterminer U avec précision.

Caractéristique fondamentale des convertisseurs analogiques numériques de l'ATmega328 : Il ne faut pas dépasser +5Vcc sur leurs entrées. Comme certaines batteries très chargées peuvent aller jusqu'à +17Vcc, on divise la tension mesurée par quatre. Sur la Fig.5 c'est le pont diviseur constitué de **R2**, **R3**, **R4** et **R5** qui est chargé de cette adaptation. Naturellement, dans le logiciel on multipliera par 4 la valeur numérisée pour obtenir la tension réelle. Le facteur de division sera directement influencé par l'imprécision des résistances. Si celles que vous utilisez font partie d'un même lot de fabrication, la dispersion de caractéristiques sera insignifiante. Par contre, si les composants sont un peu "disparates", la division s'en écartera. Pour assurer une précision logicielle, comme pour tous les autres paramètres critiques, une constante #define Correction_U_Batterie 1.015 sur le prototype corrige finement la valeur du coefficient. Dans le programme, deux lignes de code assez élémentaires assurent la transposition de la valeur CAN_U_Batterie issue de **A1** :

```
① U_Batterie = CAN_U_Batterie; // L'entier de numérisation passe dans le réel U_Batterie.
② U_Batterie = (U_Batterie * 20 / 1023) * Correction_U_Batterie;
```

Le CAN retourne un entier. La tension U_Batterie sera codée dans un réel. La ligne ① assure la compatibilité du passage de la valeur numérisée. Une entrée analogique numérise entre 0 et 1023 lorsque la tension qui lui est soumise évolue entre 0 et 5Vcc. Pour obtenir la valeur en volts à partir de la CAN il suffit d'appliquer la proportionnalité : U = CAN / 1023 * 5. Dans notre cas, la tension réelle est quatre fois plus grande. Donc U = (CAN / 1023 * 5) * 4. C'est la ligne ② qui se charge de ce petit calcul, dans lequel on a "regroupé" 5 * 4 = 20. La valeur pour Correction_U_Batterie devrait être 1.000 exactement. Bien que les quatre composants soudés sur le circuit imprimé font partie d'un même lot, on constate qu'il faut corriger de 13/1000. C'est que ce coefficient corrige simultanément l'intégralité des dispersions matérielles, et notamment la non linéarité du CAN.
Franchement, si vous ne disposez pas d'un voltmètre de précision pour effectuer les mesures, ne changez pas cette valeur. En revanche, commandez un lot de dix résistances de 10kΩ, elles seront toutes identiques, car issues d'une même fabrication, comme sur le prototype.

Brainstorming ... stérile.

Encore un chapitre dans lequel on va jouer au Sudoku. Traduisez : On va se prendre la tête juste pour le plaisir de cogiter. Dans deux chapitres ci-avant on a calculé le courant dans la résistance de décharge **R** en fonction de la tension à ses bornes. Toutefois, **A1** mesure la tension entre **GND** et **+Vcc**. En toute rigueur **R** n'est pas soumise à la totalité de la grandeur mesurée car il y a une chute de tension aux bornes du fusible. Doit-on la prendre en compte ?
Pour répondre à cette question, alimentation de laboratoire encore sollicitée, le fusible a été soumis à divers courants et la tension aux bornes de son support mesurée avec précision. Le tableau de la Fig.52 résume les valeurs générée par ce test. Si on n'en tient pas compte, le débit dépassant légèrement 4A, l'erreur avoisinerait 0,10 pour 12 soit plus de 1%. Étant donné l'application, ce n'est pas très important, il serait parfaitement raisonnable d'oublier ce petit détail. Il se trouve qu'**A2** mesure cette tension pour détecter l'absence de fusible et avertir le manipulateur. Comme l'ATmega328 passe son temps "à se tourner les pouces", autant lui faire plaisir et lui proposer un petit zeste de code en plus. Quand le convertisseur analogique

Intensité	ΔU
1A	0.025v
2A	0.056v
3A	0.08v
4A	0.10v
5A	0.13v

Fig.52

numérique est soumis à des tensions aussi faibles, sa linéarisation n'est pas géniale. Aussi, on observe ce qu'affiche l'écran LCD en fonction de ΔU et l'on en déduit un autre coefficient correcteur. Sur le prototype, la directive **#define Correction_U_Fusible 1.018** se charge de l'ajustement précis. *(Dans les options d'affichage, on peut faire indiquer la valeur mesurée par **A2** ce qui permet déterminer la valeur du coefficient de correction.)* La séquence de code qui se charge du "traitement fusible" devient :

```
U_sur_Fusible = analogRead(Entree_Fusible); // Mesuré directement.
U_sur_Fusible = (U_sur_Fusible * 5 / 1023) * Correction_U_Fusible;
```

Dans laquelle U_sur_Fusible est déclaré de type float et 5 / 1023 transpose la CAN en volts.

Tatillonnons la pinaillerie !

Uniquement pour vous proposer une approche la plus rigoureuse possible, ce chapitre frise le ridicule. Autrement dit, on va dans ce dernier pousser l'analyse bien au delà du nécessaire, simplement pour la beauté du geste. Également pour vous servir d'exemple sur la façon de s'y prendre, le jour où une telle finesse sera vraiment nécessaire. Pour cette application, c'est un luxe manifeste. C'est parti pour une boulimie cérébrale. Ceux qui n'ont pas de temps à perdre ... passez au chapitre suivant. ***Circulez, ya rien à voir !***

Considérons le schéma de la Fig.53 qui résume les trois consommateurs de courant. **I1** est celui que l'on a évalué avec précision en mesurant la chute de tension aux bornes de **R**. La carte Arduino Uno consomme intrinsèquement et au maximum 80mA quand le rétroéclairage est allumé. Comme elle est alimentée simultanément par le secteur sur sa prise USB, elle ne dérive pas de courant depuis la batterie et **I3** ***est supposé nul***. On peut alors se demander quelle est l'utilité de brancher son alimentation +Vcc sur la batterie. Et bien imaginez que vous avez lancé un processus de mesurage. Environ dix minutes avant que ce dernier ne soit achevé :

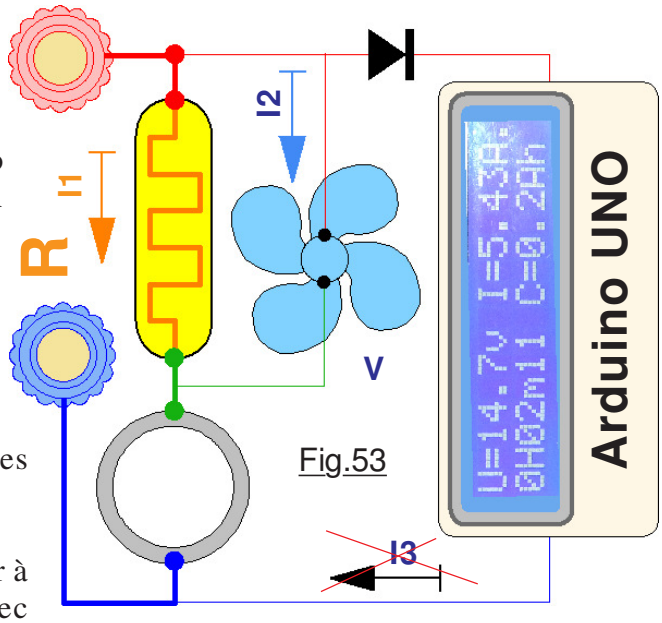


Fig.53

PAFFFFFFFFF coupure de secteur !

Les mesures sont perdues. Il n'y a plus qu'à recharger à nouveau l'accumulateur et tout recommencer. Avec cette précaution, consommant alors de façon dérisoire un **I3** de 80mA non pris en compte, le résultat sera faussé de 8 pour 450 soit 1,7% et uniquement pendant la coupure secteur. Nous sommes sauvés.

Mais alors pourquoi brancher la prise USB sur un bloc secteur puisque la batterie alimente sans vergogne Arduino qui ne prélève au maximum que trois fois rien ?

Réponse : Dès que l'on branche la batterie, elle débite au maximum. Hors le début du processus propose à l'opérateur diverses options. Pendant ce temps laissé libre, le perte d'énergie n'est pas mesurée, le résultat final serait alors imprécis.

Reste que nous n'avons pas pris en compte le courant **I2** qui alimente et traverse le ventilateur **V**. L'intensité dans son petit moteur à courant continu est variable et directement proportionnelle à la tension à laquelle il est soumis. Celui adopté sur l'appareil réalisé est prévu pour ventiler certains P.C. et se branche sur du 12V. C'est reparti pour des mesures sur site d'autant plus aisées que sa ligne peut s'isoler car il est branché sur un connecteur HE14. Le tableau de la Fig.54 présente les valeurs des consommations quand **V** est soumis à des tensions variables appliquées directement à ce dernier. Un ampèremètre précis mesure le courant qu'il absorbe. Sachant que globalement la batterie va présenter une tension nominale durant la quasi totalité du mesurage, il serait parfaitement logique de simplifier le programme et de supposer constante et de 78mA l'intensité **I2**. Du reste, il est probable que le ventilateur intégré dans votre coffret sera différent. Mesurez son courant à 12V, puis contentez-vous dans le programme de remplacer dans la procedure void Calcule_le_courant_fourni_par_la_batterie() la première ligne de code par :

Tension	I2
8v	50mA
10v	65mA
12v	78mA
14v	93mA
15v	100mA

Fig.54

```
Courant_de_decharge = Courant_ventilateur_mesuré_a_12V;
```

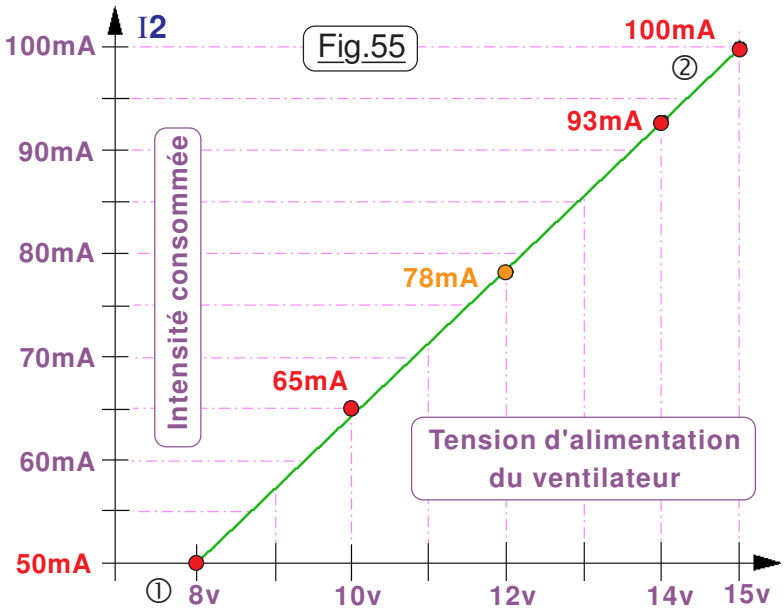
Y = A * X + B

Sauf dans la mesure ou vous avez envie de voir comment traiter un tel cas, si tenir compte des variations de courant était impératif, je vous suggère de passer à la suite et de sauter ce délire d'intello. Pour évelopper le programme qui tiendra compte des caractéristiques particulières du ventilateur, on commence par tracer la courbe de variation de **I2** en fonction de **U**. On observe manifestement que sur la Fig.55 les points représentatifs des cinq mesures consignées sont alignés. Du coup, mathématiquement, pouvoir calculer la valeur de **I2** en fonction de la tension devient facile. Tout bachelier sait qu'une fonction dont la courbe représentative est une droite est de la forme Y = A * X + B. Dans notre cas la fonction Y représente **I2**, et la variable X la tension aux bornes de la batterie d'identificateur U_Batterie. La formule de calcul devient :

I2 = A * U_Batterie + B

Pour pouvoir soumettre ce calcul au compilateur C++, il nous faut trouver les valeurs des deux inconnues A et B.

Bon, je vais frimer en vous faisant croire que je suis un dieu en mathématiques : Pour tenter d'obtenir ces deux valeurs avec le plus de précision possible, nous allons faire appel aux deux points extrêmes de la Fig.55 issus des mesures, donc traduisant une réalité physique :



① ⟹ A * 8 + B = 0.05 Car pour 8v on a 50mA soit 0.05A.
② ⟹ A * 15 + B = 0.1 Car pour 15v on a 100mA soit 0.1A.

Un bachelier qui ne galère pas trop en mathématiques déduirait que nous disposons d'un ***système lié de deux équations à deux inconnues*** A et B. Pour le résoudre, on va "tripatouiller" un peu ces équations :

① ⟹ B = 0.05 - (A * 8) Remplaçons B par cette valeur dans ②. L'équation devient :

② ⟹ A * 15 + (0.05 - (A * 8)) = 0.1 On développe cette équation pour aboutir à :

(A * 15) + 0.05 - (A * 8) = 0.1 Ou si vous préférez : (15 * A) - (8 * A) = 0.1 - 0.05

Courage, nous progressons ! En faisant les opérations on peut écrire que 7 * A = 0.05
Génial car de cette équation on déduit que A = 0.05 / 7 = 0.0071428
Reportons cette grandeur dans ① pour trouver la valeur de B :

① ⟹ B = 0.05 - (0.0071428 * 8) = 0.05 - 0.0571424 = -0.0071424

Chic chic chic nous avons enfin notre belle équation :

I2 = 0.0071428 * U_Batterie - 0.0071424

Coder cette "chose" en langage C++ confine à de l'enfantillage :

```
   void Calcule_le_courant_fourni_par_la_batterie() {
①   Courant_de_decharge = (0.0071428 * U_Batterie) - 0.0071424;
②   Courant_de_decharge = Courant_de_decharge + ((U_Batterie - U_sur_Fusible) /
③ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . R_Bloc_thermique) ; }
```

En ligne ① de la procédure on calcule dans Courant_de_decharge la valeur du courant **I2** qui traverse le ventilateur **V**. Cette ligne de calcul ne consomme que 68 octets de plus que si nous avion imposé au compilateur la constante moyenne de 0.078A. Ce serait bien dommage finalement de ne pas soumettre cette ligne au microcontrôleur qui n'attend que ça pour se faire plaisir. À ce courant **I2** la ligne ② ajoute celui qui traverse **R**. Attention, la mise en page de ce didacticiel coupe la ligne ② dont la fin est en ③.

CONCLUSION : Bien que dans cette application le calcul du courant circulant dans le ventilateur n'était pas impératif, pour montrer qu'avec un peu de mathématiques il est souvent très élégant d'utiliser des concepts relativement simples. On a déterminé l'équation représentant la réalité. Puis, comme le processeur n'est pas surchargé, loin s'en faut, au final on intègre ce calcul dans le programme.

17) Mémoriser l'évolution de la tension de décharge en EEPROM:

Lorsque le logiciel était entièrement développé, que la capacité était mesurée correctement et le résultat affiché à convenance sur l'écran LCD, le programme complet restait bien discret, très loin de saturer la place possible dans la SDRAM qui lui est réservée. Pourtant la détection de fin de charge était en place, ainsi que le système d'alerte sur absence de fusible. N'utiliser que le quart de l'espace programme disponible n'est pas très rentable, aussi, l'idée de mémoriser des données traduisant l'évolution du processus devenait particulièrement tentante. ***De plus, tant qu'à utiliser la mémoire EEPROM non volatile, autant s'en servir pour sauvegarder le contexte. Ainsi, il devient possible d'interrompre le processus à tout moment, puis de le reprendre plus tard***. Par exemple vous avez commencé à tester une batterie, et vous réalisez que la fin du mesurage va survenir vers trois heures du matin ! Soit le laboratoire est loin de votre chambre. Vous n'entendrez pas le bruiteur qui vous alerte. Hors il faut stopper la décharge sans tarder. Soit vous avez caché le bazar sous votre lit pour vous chauffer l'hiver, et ne désirez surtout pas être réveillé durant la nuit. Dans cette hypothèse, bien que la décharge totalise déjà plusieurs heures, vous suspendez le processus et débranchez la batterie. Le lendemain matin, après un bon petit déjeuner, vous rechargez le contexte depuis l'EEPROM, relancez les mesures et branchez à nouveau la batterie. Pour savoir ce que désigne "le contexte", voyons auparavant comment on va mémoriser l'historique du processus. Ces sauvegardes de données pourront librement être consultées par la suite. On en déduira si on le souhaite la tension nominale de la batterie, la courbe de décroissance de cette tension en fonction du temps. Bien entendu, seront également mémorisés le nombre d'échantillons consultables, et la capacité mesurée pour le bloc énergétique évalué. Chaque échantillon contiendra l'instant de capture, et la tension batterie à ce moment. L'instant correspond au temps écoulé depuis de début de la décharge. Il faut un octet pour coder le nombre d'heures, et un octet pour les minutes. Les saisies d'échantillons se faisant pour des secondes nulles, deux octets sont donc suffisants. Pour la tension batterie, on mémorisera la valeur issue du CAN codée sur un entier. Donc deux octets seront suffisants. À la restitution, on transposera cet entier en volts, la routine est déjà écrite pour l'exploitation en cours de mesurage.

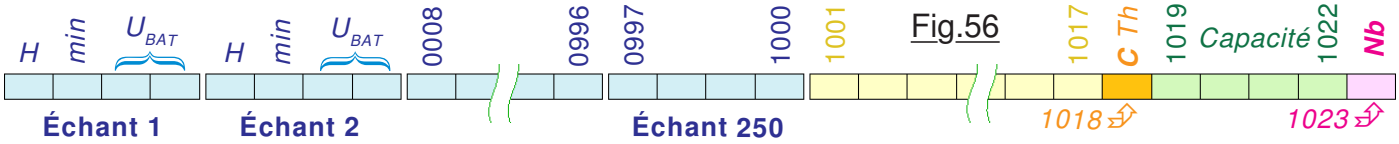


Fig.56

CONCLUSION : Il faut quatre octets par échantillon. On s'impose de ne pas dépasser 1000 octets réservés aux données, pour en avoir 24 pour sauvegarder le contexte. Comme on peut aller jusqu'à 50Ah avec un courant moyen de 4A, la décharge exigera 10,5 Heures au maximum soit 750 minutes. Sachant que le nombre d'échantillons maximal sera de 1000 / 4 = 250, on en déduit facilement l'intervalle de temps entre deux échantillons : **ΔT = 750 / 250 = 3 minutes**.
Pour simplifier la gestion de sauvegarde, les échantillons seront inscrits depuis le bas de la mémoire. Tout en haut, à l'adresse *1023* on logera le nombre d'échantillons mémorisés qui sera fonction de la capacité de la batterie, ou du fait que l'on a interrompu le mesurage. Maximum 250 se code sur un octet. Juste en dessous, sur quatre octets aux adresses comprises entre *1019* et *1022* sera sauvegardée la valeur mesurée de la capacité. Codée sur un réel, donc un float, quatre octets sont suffisants. Pour restituer le contexte, *(Explications données plus avant.)* il faut également sauvegarder la valeur de la **C**apacité **Th**éorique de la batterie testée. Codée sur un octet en cellule *1018*. Les octets de l'EEPROM compris entre les adresses *1001* et *1017* inclus restent disponibles pour d'éventuelles modifications futures. La Fig.56 résume l'implantation des données en EEPROM.

Gestion de la mémoire EEPROM.

Contrairement à la mémoire vive, c'est à dire la RAM qui contient les données, l'EEPROM est une mémoire qui ne s'efface pas quand on coupe l'énergie. Elle ne peut être écrite qu'environ 100000 fois, comme la SD RAM de programme. Ensuite, on peut la considérer comme n'étant plus fiable. Il ne faut pas y inscrire des données à des cadences considérables, comme le programme le fait pour ses données par exemple. Ceci étant dit, quand vous aurez testé 100000 fois des batteries … vous pourrez vous payer un autre ATmega 328 pour changer celui qui est sur la carte Arduino UNO ! La mémoire EEPROM est une ressource interne au microcontrôleur, sa taille est fonction du modèle utilisé. Sur l'ATmega328 on dispose d'un kilo octets soit 1024 emplacements. C'est

déjà pas mal du tout et ouvre une foule d'applications possibles.
Comme toutes les ressources spécifiques à un microcontrôleur, le langage C++ qui lui est dédié peut offrir au programmeur des instructions spéciales. Pour lire et écrire des octets élémentaires en EEPROM, c'est assez élémentaire. Par contre, si on veut y loger des entiers, des réels, il faut aller chercher les valeurs des variables en mémoire RAM. Hors, intrinsèquement nous ne savons pas où elles se trouvent, car c'est le compilateur qui se charge de leur réserver des emplacements.

Résoudre ces difficultés passe par ce que les initiés nomment **les pointeurs**. BERKKKKKkkk ! Heureusement, des courageux ont écrit des bibliothèques, nous n'avons qu'à en utiliser les procédures adaptées. Si nous n'en comprenons pas tout à fait la syntaxe, avec ces "*" qui se baladent un peu partout, ce n'est pas fondamental. On respecte l'écriture imposée, on remplace les paramètres par les identificateurs de nos variables et roule boule tout rond ...

On commence par déclarer la bibliothèque : #include <avr/eeprom.h>

Pas besoin d'aller la chercher sur Internet. Elle est tellement commode qu'actuellement elle fait partie intégrante de l'**IDE**. Donc rien de particulier à faire, c'est méga génial non ?
Pour "naviguer dans l'espace EEPROM on va utiliser un indicateur d'adresse nommé INDEX. C'est là, ou les cellules mémoire pointées par INDEX, qui seront lue ou écrites. Une cellule si on travaille sur un byte, deux si c'est un int et quatre pour un float. Nous avons à notre disposition six procédures et fonction de service qui permettent des échanges faciles avec la mémoire EEPROM :

```
Fonction Lire_un_OCTET_en_EEPROM(ADRESSE)
Fonction Lire_un_Entier_en_EEPROM(ADRESSE)
Fonction Lire_un_Float_en_EEPROM(ADRESSE)
```
} (1) Laisse inchangé INDEX.

```
Procedure Ecrire_un_OCTET_en_EEPROM(ADRESSE, Octet)
Procedure Ecrire_un_Entier_en_EEPROM(ADRESSE, Entier)
Procedure Ecrire_un_Float_en_EEPROM(ADRESSE, Réel)
```
} (2) INDEX pointe la prochaine cellule libre.

(1) Par principe personnel, et vous êtes parfaitement libres de ne pas le partager, pour une lecture en mémoire je ne modifie pas le pointeur d'adresse. C'est avant d'aller chercher la donnée que j'affecte une valeur à ce dernier. (2) Toujours par systématisme, lorsqu'une l'écriture est réalisée, quel que soit la taille de ce qui est inscrit en mémoire, le pointeur d'adresse est laissé sur la prochaine cellule mémoire. Ainsi, on peut effectuer une succession d'écritures sans avoir à se préoccuper de l'adresse. Les six subroutines dont la liste précède respectent cette façon d'élaborer un logiciel.

Notez au passage qu'il y a une différence fondamentale entre lecture et écriture. Pour lire, on utilise des ***fonctions qui retournent un résultat***. Donc, en préambule on impose une valeur à INDEX, puis on affecte la valeur retournée à une variable de type identique à celui de la fonction.
Par exemple : INDEX = 1019; Capacite = Lire_un_Float_en_EEPROM(INDEX);

En revanche, les écritures sont des ***procédures***, c'est à dire qu'elles ***réalisent des instructions sans retourner de résultats***. Dans mes procédures, INDEX est mis à jour à chaque écriture. C'est au tout début du programme que le pointeur est forcé à l'adresse 0000. Ensuite, écrite *H* et *min* l'incrémentera de 1, sauvegarder U_{BAT} l'augmentera de deux. Ainsi INDEX pointe en permanence une cellule libre "à la suite" pour le prochain échantillonnage.
Par exemple au tout début d'un mesurage : INDEX = 0000;
Puis toutes les trois minutes :

```
Ecrire_un_OCTET_en_EEPROM(INDEX, Heures);
Ecrire_un_OCTET_en_EEPROM(INDEX, Minutes);
Ecrire_un_Entier_en_EEPROM(INDEX, CAN_U_Batterie);
```

Lorsque l'on amorce le mesurage d'une batterie, on se doute que le nombre d'échantillons mémorisés est forcé à zéro, et incrémenté toutes les trois minutes. Quand la détection de fin de décharge est déclenchée, automatiquement le nombre d'échantillons ainsi que la capacité mesurée sont alors sauvegardés respectivement en *1023* et en *1019*. On peut alors couper la batterie, car le programme nous en informe par un "tirlituit" sonore très particulier dont vous aurez rapidement compris la signification. L'organigramme de la Fig.57 résume le déroulement du programme complet dans sa boucle de base pour que nous ayons une vue d'ensemble. Quelques explications

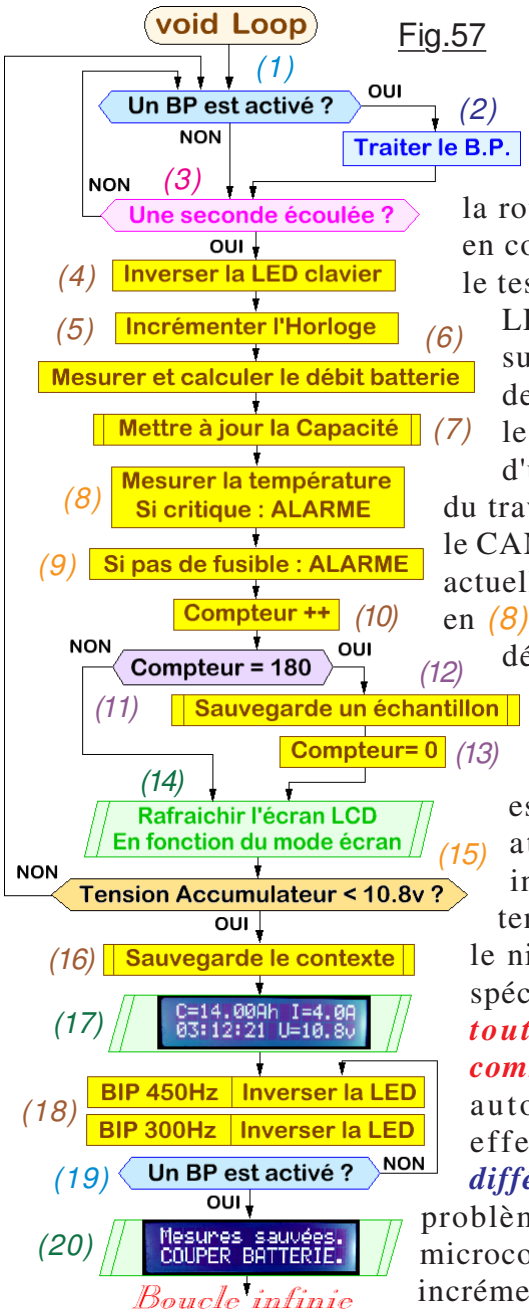


Fig.57

s'imposent. Examinons ensembles le fonctionnement de la ***boucle de base***. Si en *(1)* un bouton poussoir n'est pas activé, le programme passe immédiatement à la suite en *(3)*. Puis, si une seconde n'est pas écoulée, le logiciel reboucle en *(1)*. Comme ces deux tests ne consomment que très peu de temps, la rotation est rapide, et une sollicitation de l'opérateur sera prise en compte sans attendre. Si exactement une seconde s'est écoulée, le test *(3)* est positif et le logiciel commence par inverser en *(4)* la LED verte qui atteste du bon fonctionnement du programme. Et surtout sur le connecteur HE14 nous avons le signal qui permet de vérifier la période avec précision. Puis l'horloge qui mesure le temps écoulé depuis le début du mesurage est incrémentée d'une seconde en *(5)* dans le système sexagésimal. En *(6)* le gros du travail est effectué. La tension aux bornes de **R** est mesurée par le CAN. Puis l'intensité débitée est calculée. Enfin, en *(7)* la capacité actuellement vérifiée est mise à jour. Pour des raisons de sécurité, en *(8)* la température du dissipateur de chaleur est mesurée. Si elle dépasse le seuil critique, l'alarme en boucle fermée de la Fig.58 est déclenchée incitant l'opérateur à intervenir. La tonalité des différentes alarmes est "agressive", car il faut attirer l'attention d'un intervenant éventuel présent à proximité des lieux. Puis, la tension aux bornes du fusible est mesurée en *(9)*. Si elle dépasse le niveau de détection d'absence, une autre alarme avec un texte spécifique comme montré sur la Fig.59 est déclenchée. ***Notez que toutes les alertes, avant de prévenir l'opérateur commencent par sauvegarder le contexte pour sécurité***, autorisant s'il le désire effectuer une ***reprise en différé*** après avoir traité le problème. Si tout va bien, le microcontrôleur continue son petit bonhomme de chemin. En *(10)* il incrémente un compteur. Si en *(11)* ce compteur arrive à 180, comme il est modifié exactement une fois par seconde, c'est qu'il s'est écoulé trois minutes. Il est temps en *(12)* de sauvegarder un échantillon contenant l'Heure et la valeur de la tension batterie issue du CAN. En *(13)* on réarme le comptage. Dans tous les cas, en *(14)* on affiche les valeurs actuelles. La nature des informations visualisée dépendra de l'option sélectionnée par l'opérateur lors du traitement en *(2)* d'une action sur un bouton poussoir. La Fig.60 présente les quatre types d'informations disponibles. En *(15)* le logiciel vérifie si la batterie n'est pas complètement déchargée. Si ce n'est pas le cas, il y a reprise en *(1)* pour un nouveau tourbillon jusqu'à la prochaine seconde écoulée. Si la batterie est épuisée, immédiatement le contexte est sauvegardé en *(16)*. Puis en *(17)* l'écran **A** est forcé à l'affichage. Les deux notes *(18)* émises en boucle génèrent le "tirlituit" particulier informant l'utilisateur que le mesurage s'achève. Dès qu'en *(19)* on clique sur l'un des B.P, l'écran en *(20)* incite l'intervenant à débrancher la batterie et précise que les données ont été sauvegardées en EEPROM pour consultation ultérieure.

Fig.58

Fig.59

L'écran **A** est le plus immédiat. Il indique la valeur de la capacité actuellement évaluée, le courant de décharge, le temps écoulé depuis le début du mesurage, et la tension

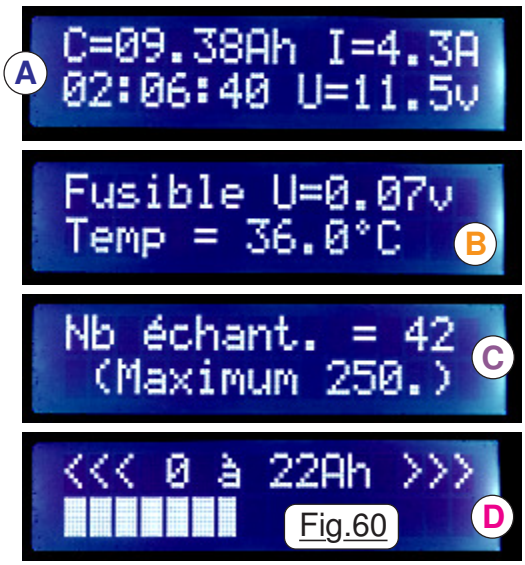


Fig.60

actuelle aux bornes de la batterie en cours de test. L'écran **B** permet de surveiller si on le désire la température actuelle du dissipateur thermique. Accessoirement, sur la ligne du haut est indiquée la tension aux bornes du fusible qui sera toujours très faible. *(Sauf si il grille ou s'il est enlevé de son support.)* Quand on clique sur le bouton **LEFT**, on fait alterner l'affichage entre les deux dernières options. L'affichage **C** précise le nombre actuel d'échantillons. La ligne du bas étant disponible, pourquoi ne pas indiquer le maximum possible ? C'est un peu du luxe, mais la mémoire de programme n'est vraiment pas encombrée, on peut gaspiller des octets à profusion. Et comme le processeur n'était pas du tout rentabilisé, en **D** nous avons un "sablier" dont le fonctionnement est précisé dans le chapitre qui suit.

Affichage d'un "sablier".

Difficile de gaver la mémoire disponible pour loger un programme dans l'ATmega328. Il faut un logiciel "meumeu" pour la consommer presque entièrement. Du coup, quand tout fonctionnait, que les échantillons étaient sauvegardés en EEPROM ainsi que les séquences de visualisation de ces derniers écrites, il restait encore plein plein de place non utilisé. C'est assez agassif de se dire que l'on a financé 32256 octets et que l'on en consomme que 10424. Aussi, histoire de rentabiliser notre investissement, l'idée du sablier s'est imposée comme étant absolumentment INDISPENSABLE.

De quoi s'agit-t'il ?

Ben ... quand vous téléchargez sur Internet un "gros truc" et que votre BOX est d'une lenteur désespérante, pour vous faire patienter WINDOWS affiche une rampe analogique qui simule un sablier. Il évalue le temps qu'il faudra, puis vous montre la proportion déjà effectuée.

Le sablier **D** fonctionne exactement pareil. Toute la largeur de la ligne du bas correspond au temps qu'il faudra pour "ratatiner" entièrement la batterie. Comme sous la tension nominale de 12V le courant est connu, il est facile de déterminer la durée nécessaire pour la décharge :

T = Capacité théorique / Courant sous 12V

Calculer pour obtenir une largeur d'affichage correspondant à la durée maximale estimée exige de connaitre la Capacité théorique de la batterie testée, c'est à dire la capacité annoncée par le fournisseur quand elle est neuve. Cette information ne peut être devinée par le programme, aussi elle sera initialisée à la mise en service de l'appareil de mesures.

Le ruban analogique exploite la possibilité de pouvoir faire afficher jusqu'à huit caractères spéciaux à notre module LCD. En particulier le "é"," le "à" et "°" n'existent pas d'origine, pas plus que ↑ et ↓. Par ailleurs, le "p", le "g" et le "j" minuscules ne sont pas très beaux. Aussi, tous ces caractères sont générés dans des matrices spécifiques. Le ruban analogique qui est affiché sur la ligne du bas de l'afficheur exige également pour son compte des caractères spéciaux. Deux programmes de démonstration sont disponibles. P04_Test_de_simultaneite.ino et P05_Rampe analogique.ino servent à dégrossir le problème.

Pour afficher la rampe analogique il faut des pavés complets à cinq colonnes comme sur la Fig.61 celui représenté en **1**. Mais l'étalement latéral ne comportera pas forcément un multiple de cinq colonnes élémentaires. Pour assurer la proportionnalité qui résultera du calcul de pourcentage, les caractères de **2** à **5** sont également indispensables. Au total, nous avons besoin de 13 caractères spéciaux. Hors l'afficheur ne peut ***en loger simultanément*** que huit. On peut les modifier librement à tout moment. En revanche, l'affichage des deux lignes ne se fait que sur un seul jeu de matrices car les pixels visualisés sur l'écran sont branchés "directement" sur les matrices personnalisées. Pour vous permettre de mieux comprendre ce qui se passe, téléchargez P04_Test_de_simultaneite.ino dans l'ATmega328, observez le déroulement du programme et surtout décortiquez son écriture. Enfin, il faut transposer le rapport calculé entre Capacité théorique et Capacité actuelle pour que 100% corresponde à 16 caractères, c'est à dire 16 x 6 = 80 colonnes élémentaires. Le démonstrateur P05_Rampe analogique.ino facilite l'analyse logicielle. Initialement la ligne du haut devait contenir le texte "Capacité", donc les caractères spéciaux "p" et "é" devaient être disponibles simultanément avec les pavés géométriques. Le texte définitif adopté montre sur la Fig.60 **D** qu'il faut le "à". Pour obtenir la simultanéité, l'ordre décrit sur la Fig.62 permet un affichage correct, car changer la police de caractères spéciaux n'écrase dans les cellules de matrices de l'afficheur que les cinq premiers.

Fig.61

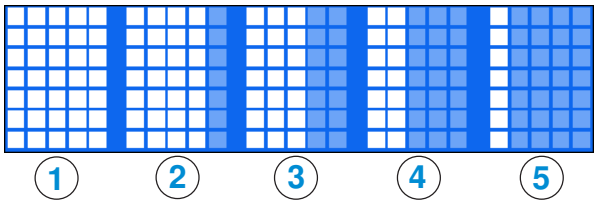
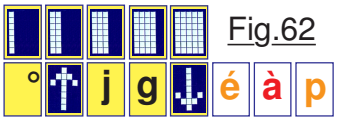


Fig.62



18) Les premières actions du programme complet :

Maintenant que nous avons analysé les diverses options du programme, définir en détail "la sauvegarde du contexte" et présenter finement les premières actions du logiciel, celle déroulées par void **setup()**, devient possible. La Fig.63 déroule à sa façon les premières actions réalisées par le programme suite à un RESET. Commentons cette représentation simplifiée. La procédure représentée n'est exécutée qu'une seule fois, puis void **loop**() prend la relève. Comme dans tout programme quel qu'il soit, on commence en *(1)* par configurer le matériel. *(Les E/S, l'afficheur LCD etc.)* Puis en *(2)* le logiciel se présente, suivi en *(3)* de l'attente d'un BP pour laisser à l'utilisateur le temps de lire l'écran. Notez que chaque invitation clavier s'accompagne d'un clignotement rapide de la LED verte. En *(4)* on propose d'aller consulter l'historique éventuellement sauvegardé en mémoire non volatile. Pour faciliter la vie de l'opérateur, pour chaque questionnement le comportement du programme sera identique : Le bouton **UP** valide l'option. Bien que l'écran laisse à penser que seul le bouton situé en dessous **DOWN** veut dire NON, ce sont tous les BP autres que **UP** qui sont considérés comme des réponses négatives. En *(5)* on laisse du temps pour lire et décider. Si en *(6)* la réponse est positive, on passe à un sous-programme en *(7)* dont la sortie ne peut se faire que par un RESET. Dans le cas contraire, c'est que l'on désire procéder à un mesurage. L'écran propose alors en *(8)* de reprendre un test qui pour une quelconque raison aurait été suspendu. Si la réponse est OUI le programme vérifie en *(11)* qu'il y a bien des données en EEPROM. Si ce n'est pas le cas, l'information *(12)* est affichée et la LED verte clignote rapidement. Seul un RESET fait sortir de cette situation. S'il y a bien des échantillons en EEPROM, c'est qu'un mesurage a bien été commencé puis suspendu. Dans ce cas les données pertinentes sont restituées en *(13)* et le programme poursuit en *(18)*. L'opérateur désirant en *(8)* débuter un nouveau mesurage, le contexte est alors effacé en *(10)*. Le programme ne peut savoir quelle est la valeur de la capacité théorique de la batterie qui va se voir évaluée. Donc en *(14)* il présente la valeur par défaut que vous aurez défini dans le code source avec la directive :

#define Capacite_indiquee_sur_la_batterie 33.

Puis attente en *(15)* et analyse de la réponse en *(16)*. Statistiquement vous aurez presque toujours à tester le même candidat, la valeur par défaut vous conviendra. Toutefois, en *(17)* il sera aisé de changer la valeur. UP et DOWN augmentent ou diminuent la valeur de 10, RIGTH et LEFT d'une unité, SELECT valide la valeur saisie. La fourchette autorisée est comprise entre 0 et 50Ah, tout débordement provoquera un BIP d'erreur. Enfin, en *(18)* le programme vous invite à brancher la batterie avec l'attende banale en *(19)* pour vous laisser lire. Dès que la batterie est branchée elle commence à se décharger. Hors le programme attend un BP pour passer à la boucle de base qui effectue les mesures. Aussi, il vous incite fortement à ne pas trainer sur la ligne du bas. Il aurait été envisageable de ne laisser cet écran que durant trois à quatre secondes puis de passer en mode mesurage. Ce n'est pas possible, car si vous prenez votre temps pour brancher, une tension nulle est détectée et le logiciel l'interprètera comme étant le résultat d'une décharge complète. La représentation de cet organigramme ne respecte pas totalement l'architecture exacte du programme. *(Pour obtenir un dessin le plus aéré possible.)* Dans la réalité, toutes les actions *(10)*, *(14)*, *(15)*, *(16)* et *(17)* sont incluses dans une procédure unique nommée void Efface_le_contexte_et_INIT_BAT(). Dans mes programmes je regroupe souvent des "paquets" d'instructions dans des subroutines pour rendre le logiciel global plus lisible.

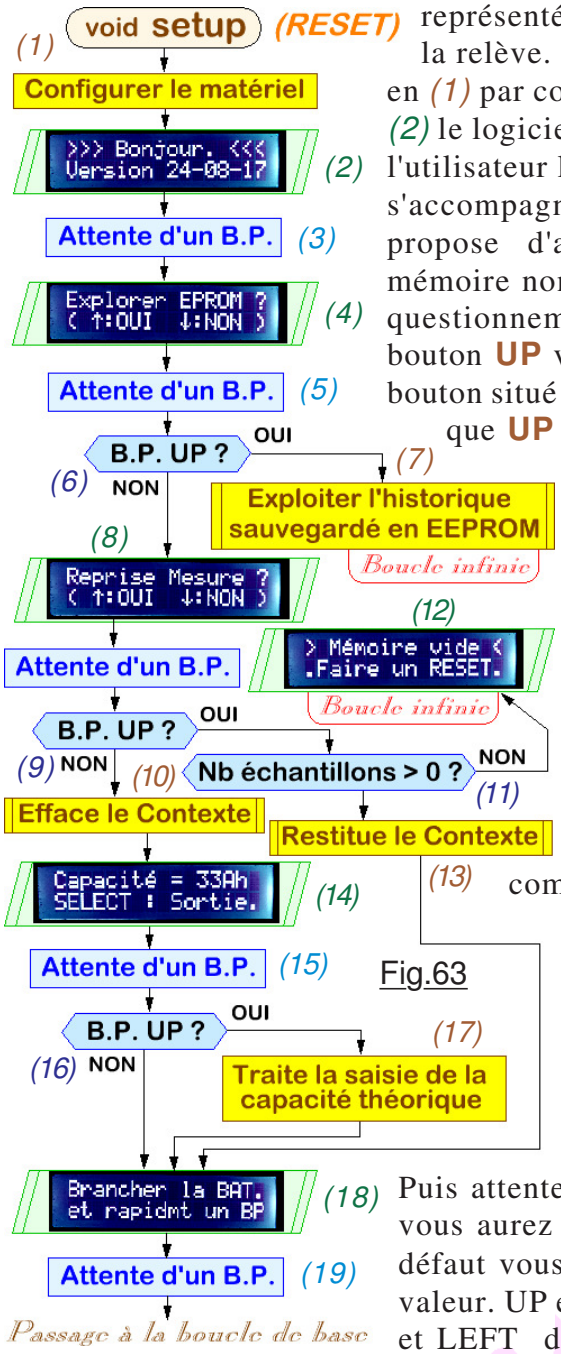


Fig.63

Sauvegarder / Restituer le CONTEXTE.

Cité à diverses reprises, CONTEXTE est une notion qui mérite de se voir expliciter pour en saisir le sens exact. Pour faire court, CONTEXTE désigne le nombre minimal de données qui permettent de reprendre une activité qui a été suspendue. Pour minimiser la place occupée par les données qui sont logées en EEPROM, on n'y loge que celles qui sont intrinsèques, c'est à dire qui ne peuvent pas se déduire des autres. Par exemple, il sera **inutile** d'écrire la valeur du pointeur de mémoire INDEX. En effet, par nature il se trouve juste à la suite du dernier échantillon. Comme le premier échantillon commence à l'adresse zéro, pour rétablir INDEX il suffit globalement de multiplier le nombre des échantillons par quatre. L'instruction exacte est :

```
INDEX = (4 * NB_echantillons) - 4; // Pointe le début du dernier échantillon.
```

Cette instruction n'est invoquée qu'après avoir restitué dans NB_echantillons la valeur de la cellule d'adresse *1023*. Évidemment il importe de restituer la valeur actuellement mesurée de Capacite, mais vous vous en doutiez. Moins évident, Capacite_theorique doit également être restituée pour que l'affichage analogique du sablier puisse se gérer correctement. Enfin, avant de calculer la valeur d'INDEX pour qu'il pointe la première cellule libre dans l'historique, il faut avoir cherché le temps qui s'est écoulé depuis le début des mesures effectuées. Cette durée est sauvegardée dans le dernier échantillon de l'historique. Naturellement, la sauvegarde doit inscrire ces mêmes valeurs pertinentes.

Exploiter un historique enregistré en EEPROM.

Fig.64

Mis à part que cette séquence boucle à l'infini, son traitement est assez banal. Comme pour la saisie des caractéristiques d'une batterie, chaque échantillon affiché comme sur la Fig.64 pourra se voir exposé à l'écran LCD, la procédure restant pratiquement identique. **UP** et **DOWN** font avancer ou reculer de 10 échantillons, **RIGTH** et **LEFT** d'un seul, **SELECT** pour son compte affiche l'écran de la Fig.65 qui indique la valeur de la capacité mesurée ainsi que le nombre d'échantillons disponibles. Toute tentative d'accéder à des échantillons inférieurs à 1 ou supérieurs au nombre disponible engendrera un BIP d'erreur. Comme tous les BP sont exploités, il n'en reste plus pour sortir de cette subroutine sauf ... un petit clic sur **RESET** !

Fig.65

Fig.66

Encore faut-il qu'un historique soit présent dans l'EEPROM. On peut très bien déclencher un nouveau processus, et le quitter précisément au début, en *(19)* de la Fig.63 juste avant de valider le branchement de la batterie. Dans ce cas le contexte a été effacé. Le nombre d'échantillons a été remis à 000. Pour nous prévenir qu'il n'y a rien dans l'historique, le message d'erreur de la Fig.66 est affiché avec un BIP d'alerte. Le programme boucle alors sans fin jusqu'à un RESET comme pour toutes les ALERTES.

Utilisation du clavier dans la boucle de base.

Traitée directement dans vood **loop**() et précisé en *(2)* sur l'organigramme de la Fig.57 la séquence qui s'occupe du traitement des touches durant un mesurage n'a pas été entièrement détaillée. **UP** présente l'écran **A** de la Fig.60 alors que **DOWN** celui en **B**. Nous savons que **LEFT** permet d'alterner entre les informations **C** et **D**. Il reste à décrire l'effet des deux autres touches. Le bouton **SELECT** pour sa part fait alterner entre un écran lumineux et un écran très sombre, c'est alors le mode VEILLE qui permet d'économiser l'écran durant les longues périodes où l'on ne vient pas voir l'appareil. Bien que sombres, les informations restent toutefois visibles. Notez que toute action sur une touche fait immédiatement repasser l'écran en arrière-plan lumineux pour en faciliter la lecture. **RIGTH** est utilisé comme "bouton potentiellement à risque" car il est placé juste à coté de **RESET**. Cliquer sur ce dernier fait afficher l'écran de la Fig.67 pour proposer de figer le mesurage en cours. Seule **UP** validera cette proposition. Les données sont alors sauvegardées. Puis le programme passe à la séquence infinie qui marque la fin d'un mesurage par l'affichage de la Fig.68 déjà mentionnée tout en bas de l'organigramme de la boucle de base vood **loop**().

Fig.67

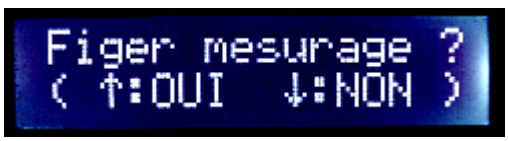


Fig.68

19) Jamais assez d'octets consommés :

Tout irait pour le mieux dans le meilleur des mondes si votre narrateur n'était pas d'un radinisme patent quand il s'agit de rentabiliser un microcontrôleur. Le programme se comporte comme un charme. Sauf qu'à la compilation, le couperet tombe, impitoyable : À peine 11458 octets de consommés dans la mémoire réservée au programme. Une misère, et il reste 65% de place disponible. Un gaspillage scandaleux. Il en résulte des nuits blanches à broyer du noir, *(Ouafff le jeu de mots !)* un moral au Nadir, une perte de poids et une humeur exécrable. ***Il faut absolument trouver un moyen de consommer quelques octets.***

OUFFFFFFFFFFFF, enfin une idée : Lister la totalité de l'historique sur l'écran de l'ordinateur en utilisant le Moniteur de l'**IDE**. Le tout formaté pour une lecture confortable comme montré sur la Fig.69 avec une vue de l'ensemble bien plus commode, qu'échantillon par échantillons sur l'afficheur LCD.

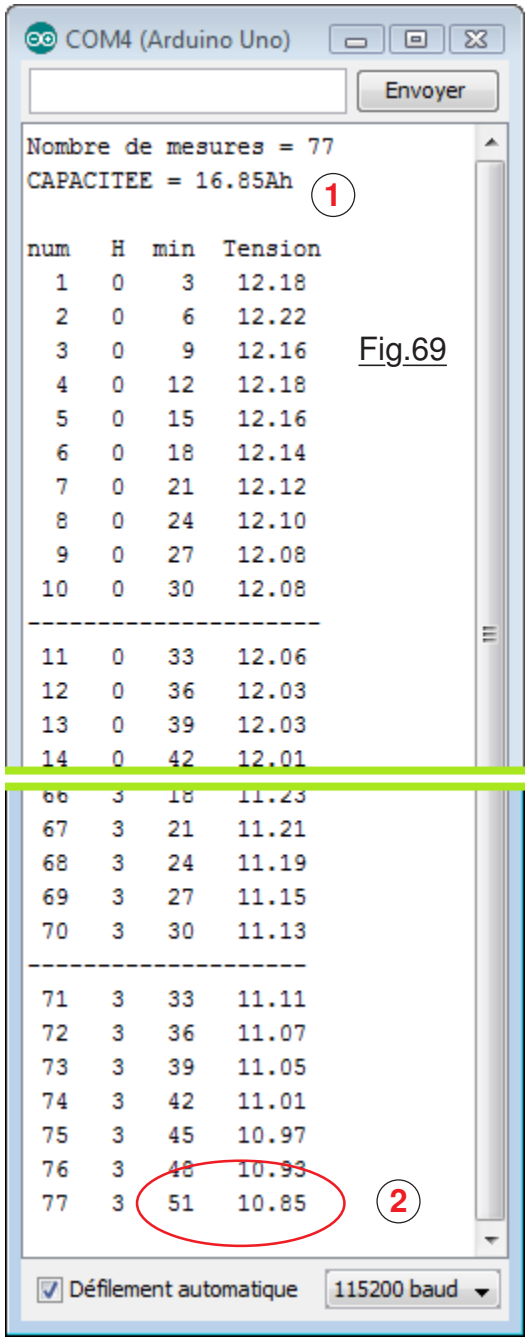


Fig.69

NOTE : Les mesures de la Fig.69 sont issues de la batterie neuve de la Fig.1 pourtant réputée de 22Ah. C'était le deuxième cycle d'utilisation. Le premier cycle avait servi à l'alimentation d'un petit télescope, et montrait manifestement un manque d'autonomie. C'est ce doute qui a motivé la réalisation de cet appareil de mesures. On constate en **2**, mais ce serait également vrai avec une batterie en bon état, qu'une fois avoir restitué la plus grande partie de son énergie potentielle, la tension aux bornes d'un accumulateur au plomb s'effondre relativement rapidement. Ce bloc d'alimentation 12V a été changé dans le cadre de la garantie ...

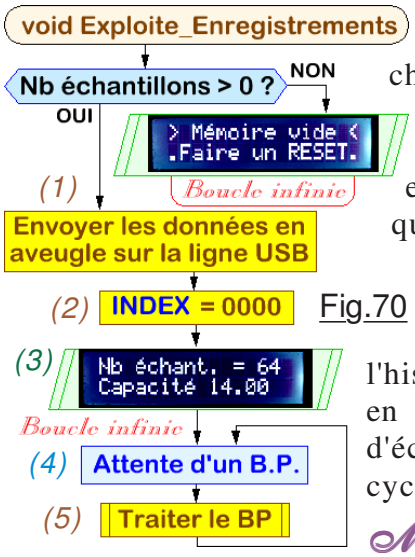


Fig.70

L'organigramme de la Fig.70 développe la séquence qui se charge d'exploiter un historique. Si la mémoire n'est pas vide, en *(1)* les données sont envoyées sur la ligne série USB que vous ayez mis en service le **Moniteur série** de l'**IDE** ou non. Libre à vous d'en tenir compte ou de l'ignorer. En *(2)* le pointeur en EEPROM INDEX est placé sur le début des données du premier échantillon. En *(3)* un résumé de l'historique est proposé. Puis on arrive dans une **boucle sans fin** qui attend en *(4)* l'activation d'une touche du clavier. Cette dernière change d'échantillon et en affiche les données, ou réaffiche de résumé *(3)*. Puis le cycle recommence indéfiniment en *(4)* jusqu'à provoquer un **RESET**.

Mazette, seuls 13158 octets sont consommés en mémoire de programme, soit à peine 40% ... et je ne sais plus quoi faire. Il faut savoir s'arrêter, et le moment semble venu de nous séparer. Le but de ce tutoriel était autant de vous présenter des méthodes d'analyse et de programmation que la réalisation d'un appareil dont l'usage est, il faut bien l'avouer, relativement peu fréquent.

Je souhaite que la lecture de ce didacticiel vous apportera beaucoup de plaisir, et que vous en tirerez des informations qui vous serviront souvent. Reste que ce n'est pas un cours magistral, mais seulement les bavardages d'un Arduinaute amateur en informatique avec forcément pas mal de faiblesses et de lacunes en programmation.

Amicalement : Nulentout.