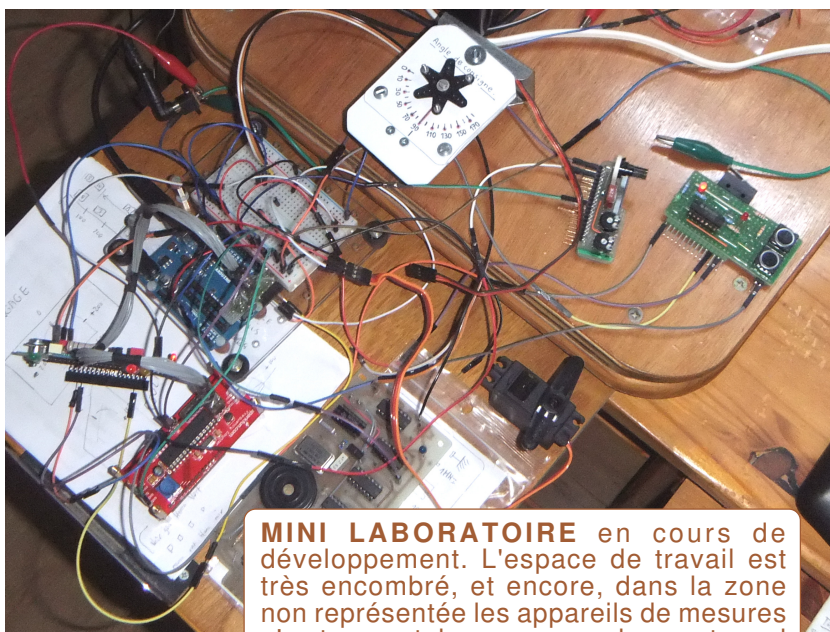


MINI LABORATOIRE "USB" AVEC ARDUINO.

Généralement, tous ceux qui programment sur Arduino se rendent rapidement compte que l'espace disponible au voisinage de l'ordinateur est pratiquement saturé avant d'avoir écrit les premières lignes de code, et ce n'est pas l'illustration donnée ci-dessous qui va contredire cette assertion. Aussi, pouvoir disposer de moyens de mesures très compacts ne peut que séduire le lecteur. Concevoir un MINI LABORATOIRE à base d'Arduino, et au moindre frais, ouvre la piste à la réalisation d'un appareil autonome qui pourrait résider en permanence sur la table de travail. Ce document n'a rien à voir avec un didacticiel pour apprendre le langage C. La toile ne manque pas de références à ce sujet, et prétendre apporter mieux serait assez présomptueux. Dans ces pages, nous allons aborder la conception d'un appareil autonome aux performances très alléchantes. Elles ne constituent qu'un prétexte pour vous faire cheminer un peu dans le monde du mesurage, et surtout vous montrer les fabuleuses possibilités offertes par l'ATmega328, associé au langage C lui même enrichi de procédures prédéfinies qui permettent même à un débutant de réaliser des applications qui, il n'y a encore que quelques années, n'étaient qu'à la portée des Ingénieurs en génie logiciel. Bien que n'étant en rien un modèle de rigueur dans l'approche et l'écriture des programmes qui vous sont proposées dans ce qui suit, la présentation du code est soignée, et les techniques utilisées ont été sélectionnées avec un souci permanent de pédagogie. Nous allons pas à pas approcher quelques concepts élémentaires, certains très généraux, et d'autres plus sophistiquées, pour papillonner dans l'univers de l'IDE d'Arduino. *(I.D.E. est l'environnement de programmation propre aux microcontrôleurs de la famille à laquelle appartient l'ATmega328.)*

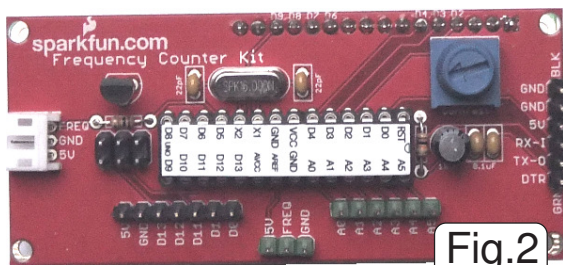
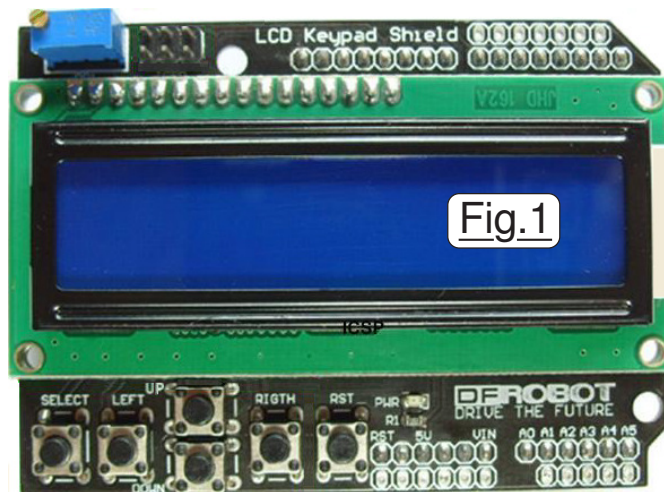


Avant d'intégrer le résultat de toutes nos expériences dans un gros logiciel de plusieurs centaines de lignes, nous allons cheminer par l'entremise de petits programmes élémentaires, chacun apportant une pierre à l'édifice. Vous pourrez ainsi examiner les listages des sources au fur et à mesure de votre apprentissage du langage C et comparer ces petits "démonstrateurs" avec ceux trouvés sur Internet, que vous aurez abordé durant vos loisirs de programmeur.

Mais ne vous y trompez pas, tout en nous amusant, nous allons tracer la route vers un MINI LABORATOIRE à base d'ATmega328 dont les caractéristiques n'auront rien à envier à des produits bien plus onéreux ... sachant au demeurant qu'un appareil qui réunit toutes ces fonctions n'est pas du tout courant sur le marché. Bien que l'aboutissement réside dans un instrument totalement autonome avec visualisation sur écran LCD, vous pourrez vous contenter si vous le désirez, d'un simple Arduino UNO complété avec quelques résistances, un potentiomètre et divers autres composants, le total pour un coût dérisoire, surtout si on le compare à tout ce que ces quelques éléments vont permettre de faire une fois le logiciel téléversé dans la mémoire de programme. Pour alléger ce fichier informatique, seule la version "Ligne série USB" sera abordée dans ces lignes. Le passage à un instrument totalement autonome avec visualisation sur afficheur LCD fait l'objet d'un didacticiel séparé. Par contre, l'optique de compacité préside les choix effectués. Enfin, sachez que dans cette version économique, nous n'avons besoin d'aucune bibliothèque spécifique, l'environnement de base de programmation d'Arduino sera amplement suffisant.

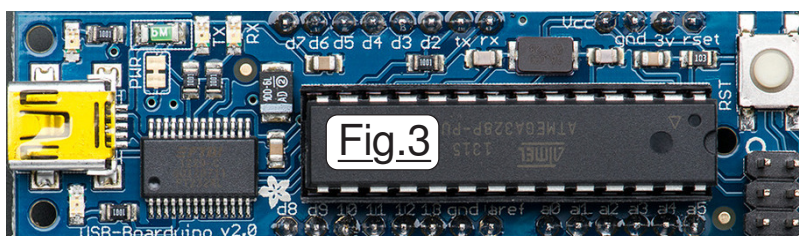
Cher lecteur, comme déjà précisé, il ne s'agit pas ici de vous apprendre à mettre en service l'environnement de programmation d'un Arduino. D'une part parce que "tout" est actuellement disponible sur Internet concernant cette facette du développement sur microcontrôleurs, mais d'autres parts, si vous possédez ou achetez un Arduino UNO c'est que probablement vous êtes déjà autonomes dans ce domaine. Nous allons uniquement effleurer le très vaste domaine du mesurage, et ce de façon ludique. Ce document associé aux listages de nombreux programmes est un bonus gracieusement mis à votre disposition. Une petite friandise informatique en quelques sortes. On suppose ici que vous savez utiliser l'éditeur de texte de l'I.D.E. et que téléverser du code compilé dans Arduino confine pour vous à de "la routine".

Si il est un domaine pour lequel l'ATmega328 s'avère particulièrement convivial, c'est bien celui des mesures analogiques, ou des mesures relatives au déroulement du temps. Avec ses cinq convertisseurs Analogiques Numériques internes indépendants, il devient presque élémentaire avec Arduino d'effectuer des mesures en tous genres, et ce d'autant plus que le compilateur C nous affranchit de l'écriture compliquée en binaire ou en assembleur pour élaborer des procédures de calculs. Un simple Arduino Uno, ou mieux, un petit module plus simple deviennent des outils rêvés pour se concocter des appareils de mesures à notre souhait. Ce petit document résume quelques expériences pour développer divers petits projets, avec pour aboutissement un multimètre "universel" compact et aussi polyvalent que possible. Comme chaque aspect des mesures envisagé est abordé autant au point de vue théorique, que sous l'aspect pratique et celui de la programmation, chacun pourra s'en inspirer pour étudier du "personnalisé". Dans une première étape nous allons graduellement analyser différents types de mesures, la sortie des résultats étant effectuée sur la ligne série USB du module Arduino. Ainsi, mis à part quelques composants élémentaires, on pourra disposer déjà d'un appareil complet ... pratiquement sans bourse déliée. L'aboutissement de ces diverses études élémentaires consistera à étudier un appareil complet totalement indépendant avec affichage sur un écran à cristaux liquides pourvu de deux lignes de 16 caractères.



Les variantes envisagées.

Compte tenu des nombreux produits disponibles en magasin, plusieurs pistes sont envisageables. Si vous disposez d'un Arduino UNO et d'un "Shield" LCD, (Voir la Fig.1) vous pouvez parfaitement transformer ce couple en un mini laboratoire, et vous passer de la version "Ligne série USB". L'assemblage des deux modules alimenté à part de votre P.C. sera déjà totalement autonome, sans pour autant engager des frais supplémentaires. Naturellement, un programme adapté est fourni avec le document relatif à "l'approche LCD". Mais l'intégralité de l'étude a été contrainte par l'optique de finaliser par un appareil très compact utilisant un KIT fréquencesmètre montré sur la Fig.2 dont le microcontrôleur permet d'évaluer la taille. Plus petit qu'un ARDUINO UNO ce module est conçu pour fournir les valeurs sur un afficheur LCD de deux lignes à seize caractères chacune. Sur la Fig.2 l'afficheur n'est pas visible car situé de l'autre côté du petit circuit imprimé. Enfin, pour les fanas de la miniaturisation à outrance, la Fig.3 présente une autre possibilité, car le



minuscule circuit imprimé est à peine plus grand que l'ATmega328 qui permet d'en évaluer l'échelle. Mais ce dernier n'est pas équipé du "boot loader" qui assure le téléversement du programme. Il faut un Arduino UNO pour réaliser cette opération. Ces diverses possibilités qui du reste n'ont rien d'exhaustif nous montrent à l'évidence les innombrables variantes offertes par les microcontrôleurs de la famille AVR® dont le succès mondial n'est plus à démontrer.

Fonctions prévues sur le mini laboratoire.

Véritable couteau Suisse pour électronicien, l'appareil envisagé sera capable d'effectuer des mesures très variées, mais également de générer des signaux "carrés TTL" à la demande. Pour vous mettre l'eau à la bouche, voici la liste des fonctions prévues sur la version "Ligne série USB". Son grand frère de type LCD sera pourvu des mêmes options, mais il sera également complété par la possibilité d'afficher les tensions sous forme d'un "ruban" analogique. Voici le programme :

- VOLTMÈTRE pour tensions continues et ~
- Mesure de Résistances
- COMPTEUR
- IMPULSIOMÈTRE
- GÉNÉRATEUR BF
- Pilotage de SERVOMOTEURS
- AMPÈREMÈTRE
- TESTEUR DE CONTINUITÉ
- PÉRIODEMÈTRE
- FRÉQUENCEMÈTRE
- CHRONOMÈTRE
- GÉNÉRATEUR ÉTALON
- Générateur PWM
- CAPACIMÈTRE

Dans un projet à base de microcontrôleur **la répartition judicieuse des Entrées / Sorties est une phase impérative qui conditionne** directement la difficulté rencontrée lors du **développement logiciel**. Le tableau donné ci-dessous résulte de nombreux compromis. Il est forcément compatible avec les branchements réalisés sur le petit KIT fréquencemètre mais ne doit pas remettre fondamentalement en cause le programme si on utilise un "Shield" LCD. Enfin quelques broches sont imposées par l'usage de certaines spécificités internes à l'ATmega328. Ne croyez surtout pas que c'est celui qui était prévu au début, l'avancement du projet l'a pas mal remanié ...

| AFFECTATION DES ENTRÉES / SORTIES | |
|-----------------------------------|--|
| Broche | Utilisation |
| D0 | Réservée en standard pour la liaison série USB. (RX) |
| D1 | Réservée en standard pour la liaison série USB. (TX) |
| D2 | RS (LCD) |
| D3 | R/W (LCD) |
| * D4 | E (LCD) DB4 |
| * D5 | Entrée pour le Compteur / Fréquencemètre / Périodemètre etc. (TIMER1) DB5 |
| * D6 | DB4 (LCD) DB6 |
| * D7 | DB5 (LCD) DB7 |
| * D8 | DB6 (LCD) RS |
| * D9 | DB7 (LCD) E |
| * D10 | Pilotage de la base du transistor de rétroéclairage. (LCD) |
| D11 | Génération de signaux PWM. |
| D12 | Génération de signaux B.F. et de signaux Étalon de fréquence. |
| * D13 | Visualiser les actions sur les B.P. de commande. (Rétroéclairage du SHIELD.) |
| A0 | Gestion des deux B.P. de changement de fonction et de choix des options. |
| A1 | Entrée du Voltmètre. |
| A2 | Mesure des résistances de 100Ω à 150 KΩ. |
| A3 | Mesure des résistances de 0Ω à 100Ω. |
| A4 | Mesure des intensités de 50mA à 500mA. |
| A5 | Mesure des intensités de 0mA à 50mA. |

En rouge : les impératifs sur le KIT fréquencemètre.
En bleu : ceux imposés par le SHIELD LCD.
(R/W à la masse.)

Définition de **SHIELD** (Bouclier.) : Petite carte électronique spécifique dédiée à une application dont l'agencement permet le branchement "gigogne" direct sur les connecteurs d'une carte Arduino.

LA MESURE DES RÉSISTANCES avec l'ATmega328 :

Mobiliser un Arduino UNO uniquement pour afficher sur la ligne série USB des valeurs de résistances reste très réducteur, et ce d'autant plus injustifié que plusieurs entrées analogiques restent disponibles et que la place occupée par le programme sera loin d'utiliser toute la mémoire du microcontrôleur. Le programme développé et analysé dans ce chapitre ne sera qu'une première étape, une sorte de "mise en bouche informatique" durant laquelle on va aborder des concepts de base incontournables. Évaluer des résistances n'est pas totalement immédiat. Si l'on accepte de mobiliser deux entrées analogiques et que l'on ajoute deux résistances au module microcontrôleur mis à contribution, nous pouvons aisément transformer Arduino en un ohmmètre très honorable. Avant d'engager l'analyse d'un multimètre complet avec affichage sur écran LCD, commençons timidement par cet ohmmètre élémentaire qui dialogue avec l'émulateur de terminal de l'I.D.E.

L'algorithme utilisé.

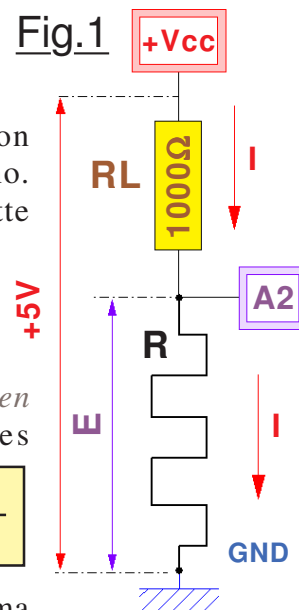
Retrouvons nos fondamentaux, car la mesure d'une résistance consiste à la faire traverser par un courant et à en mesurer la tension aux bornes. Cette bonne loi d'Ohm ($U = R * I$) va nous fournir une solution élémentaire. Le courant sera fourni par l'alimentation +5Vcc réglée d'Arduino. Comme il faut obligatoirement limiter le courant qui sera consommé sur cette dernière, on doit impérativement insérer une résistance en amont du composant à mesurer. Pour simplifier l'analyse, considérons le schéma utilisé montré en Fig.1 dans lequel **R** est la résistance à mesurer et **RL** la résistance de limitation. Les deux résistances placées en série sont parcourues par un courant identique soit $I = 5 / (RL + R) = 5 / (1000 + R)$. (*Tension en Volts, I en Ampères et R en Ohms.*) En effectuant quelques permutations licites mathématiquement on arrive à la formule peu compliquée :

$$R = \frac{E \times 1000}{5 - E}$$

Dans cette équation facile à coder en langage C le symbole **E** représente la tension mesurée par **A2** et exprimée en Volts pour respecter l'homogénéité des dimensions. Avec ce schéma électrique, on peut mesurer des valeurs de résistances comprises entre zéro, soit un court circuit, et 150kΩ environ. Cette plage est suffisante, car avec les systèmes à microcontrôleurs il est assez rare d'employer des composants de fortes valeurs. Cette solution est simple, n'exige qu'une seule résistance à ajouter à Arduino. Sur court-circuit le courant traversant **RL** sera limité à 5mA donc très faible. Toutefois, quand les résistances deviennent inférieures à environ 50Ω, la tension mesurée devient faible et l'imprécision des mesures diverge. Pour pouvoir mesurer des valeurs jusqu'à 1Ω tout en conservant une précision d'au moins deux chiffres significatifs, il faut diminuer la valeur de **RL**. Attention, sur court-circuit le courant doit rester acceptable. Un deuxième calibre est donc prévu en mesurant avec l'entrée analogique **A3** sur laquelle la résistance **RL** est diminuée à 100Ω. Il suffit de remplacer dans la formule **1000** par **100** pour ajouter ce calibre informatiquement.

La programmation.

L'affichage des valeurs par l'entremise de la ligne série USB doit être permanent pour disposer d'un appareil de mesure vraiment opérationnel. Mais un défilement continu de valeurs qui brouillent l'écran vidéo sans pour autant changer notablement ne serait pas du tout convivial. L'idée consiste alors à n'afficher sur l'écran une nouvelle ligne que si la résistance mesurée a changé de valeur. Mais pour aboutir à un résultat satisfaisant, il faut impérativement s'affranchir du "bruit" en procédant à un lissage en utilisant les informations données en page 7 relatives à cette facette de l'informatique. (*Lissage par "intégration".*) Comme la tension **E** mesurée sera sujette à de permanentes petites variations, pour calmer le rafraichissement de l'écran on va réaliser **N** mesures et en faire la moyenne. La valeur de **N** a été choisie à 400. C'est la valeur la plus grande possible qui donne à l'usage l'impression d'un affichage instantané. Il faut impérativement une rapidité de "résolution" quand on désire ajuster à une valeur précise la résistance d'un potentiomètre par exemple. De plus, dès que l'on soumet un composant à une mesure, on désire immédiatement lire le résultat



sur le moniteur vidéo. La Fig.2 présente l'organigramme relatif à cette partie du traitement. Mais comme il y a deux calibres, on va effectuer exactement la même chose sur

Void N_mesures_sur_A2 ()

Mettre la somme à zéro

Effectuer une conversion AN

Somme = Somme + Mesure

N mesures effectuées ? NON

OUI

Realise_la_moyenne(); (1)

SUITE **Fig.2**

Void Realise_la_moyenne()

Valeur E = Somme / N

Transposer cette valeur variant entre 0 et 1023 en une plage comprise entre 0 et 5000

Valeur = 5000 NON

OUI

Valeur = 4999.99

Fig.3

SUITE

l'entrée analogique A3. Pour éviter de calculer la moyenne, de transposer en "volts" et d'exclure "un dénominateur nul dans les calculs" deux fois dans le code de ce programme, la fin du traitement est confiée en (1) à une procédure commune nommée **void Realise_la_moyenne()** dont la Fig.3 ci-contre présente l'organigramme. Un autre petit problème va se produire régulièrement quand aucune résistance n'est en cours de test. La

on aboutit à : $5 - 5 = 0$. (*Facile ... ça tombe juste !*) Hors, mathématiquement, la division par zéro est interdite. Informatiquement elle peut conduire à une boucle de calcul infinie. Dans le cas d'Arduino, après compilation, en exécution on aura un affichage de type "Int" pas très esthétique. C'est la raison pour laquelle toute tension mesurée supérieure à 4,96Vcc écourtera le déroulement du programme et sera ignorée dans la boucle de base qui passera à la suite du traitement cyclique.

Comment cette valeur particulière a-t-elle été déterminée ? Elle résulte d'un compromis. Plus on adopte une valeur proche de +5Vcc, plus la plage des valeurs mesurable augmente vers les valeurs élevées. Rien n'empêche informatiquement de prendre 4,999999Vcc puisque la tension mesurée sera un **float**. Il serait alors possible de mesurer jusqu'à plus de 800kΩ. Mais ce n'est pas idéal pour deux raisons. La précision des mesures se dégrade, comme pour celle des valeurs faibles. Le bénéfice est médiocre, car déjà avec 100kΩ on couvre largement nos besoins courants. Par contre, si proche du +5Vcc les fluctuations de numérisation arrivent à "traverser" le filtre que nous avons mis en place avec une moyenne sur 400 mesures consécutives. Il en résulte des affichages erronés qui viennent régulièrement perturber l'écran par des affichages de résistances fantômes de fortes valeurs. Avec cette valeur de 4,96Vcc déterminée expérimentalement on aboutit à un comportement parfaitement convivial avec une qualité opérationnelle de notre instrument de mesure tout à fait acceptable. L'organigramme de la Fig.4 donné en page 6 résume le déroulement de la boucle de base. C'est le petit programme **P01_Ohmmetre_sur_USB.ino** qui réalise tous ces traitements et ne consomme en tout et pour tout que 5612 octets sur les 32256 disponibles.

Manifestement l'organigramme de la Fig.3 montre que l'on a utilisé une formule de calcul dans laquelle la tension est exprimée en mV avec bien entendu un facteur de correction de 1000 partout où il s'impose. C'est pour obtenir une précision mille fois plus importante dans le traitement numérique effectué par le programme. On bénéficie ainsi des valeurs de résistances affichées avec une seule décimale, mais avec aucune imprécision résultant du traitement proprement dit. Comme il s'agit d'un programme préparatoire en vue d'un projet plus élaboré, pour faciliter la mise au point la valeur de la tension mesurée **E** est également précisée lors des affichages. Si dans un programme "définitif" ce n'était plus utile, un gain de 80 octets serait ainsi facilement possible. Noter au passage que l'affichage de la valeur de la résistance précise si la mesure retenue par le logiciel est issue de A2 ou de A3. Ce n'est pas vital en soi, mais au cours du développement du programme il était commode de repérer quelle entrée se montrait coupable d'un affichage incohérent ce qui facilitait bien la mise au point, et en particulier le choix de certaines valeurs critiques comme le 4,96Vcc.

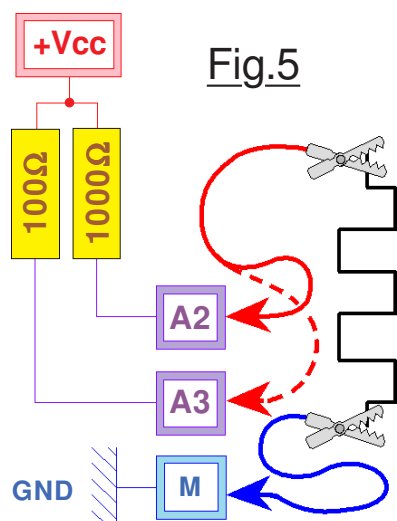


Fig.5

La pratique et les erreurs à éviter.

Utiliser un appareil de mesure aussi simple que celui décrit dans ces lignes pourrait se passer de commentaires. Comme montré sur la Fig.5 on branche les deux fils sur Arduino. L'un va à la masse, et l'autre soit sur **A2** soit sur **A3**. Chaque extrémité est ensuite reliée aux fils du composant à mesurer avec les pinces crocodile. Quoi de plus élémentaire ?

DANGER !

La façon la plus rapide de détruire un potentiomètre consiste à le brancher comme montré sur la Fig.6 c'est à dire entre le plus cinq volts et la masse. Comme la valeur du composant est de 10kΩ on se croit en sécurité. Le problème arrive quand on tourne l'axe de réglage. La valeur diminue et peut descendre jusqu'à zéro, soit un court circuit.

Nous sommes immédiatement renseignés sur cette bêtise, à la fois par l'odeur caractéristique et la fumée qui se dégage du composant. Trop tard !

Naturellement vous ne commettrez jamais une telle sottise ... encore que ! Revenons à notre ohmmètre branché sur le calibre des résistances de faible valeurs. Un courant de 50mA peut traverser le composant en cours de mesure. Si c'est un tout petit potentiomètre ajustable, il ne résistera pas à ce régime. **Conclusion : Ne jamais brancher un élément ajustable comme montré sur la Fig.6 quand vous utilisez l'entrée A3. Toujours commencer par mesurer avec A2 et ne changer de**

calibre que si le logiciel le demande. Vous serez alors sur des composants de faible résistance qui en principe admettent, sauf exception rare, un courant de cette valeur. Revenons au gain de 80 octets possible en hypothèses en bas de la page 5. Une telle économie ne changera pas radicalement la donne. Par contre, la chute de tension aux bornes du composant soumis à la mesure peut s'avérer intéressante. En particulier si l'on soumet des diodes, des transistors etc. C'est la raison pour laquelle sur le schéma de la Fig.5 les fils de liaison pour la mesure sont respectivement rouge et bleu pour satisfaire la polarité durant le mesurage.

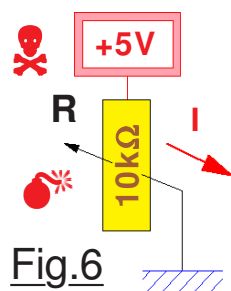


Fig.6

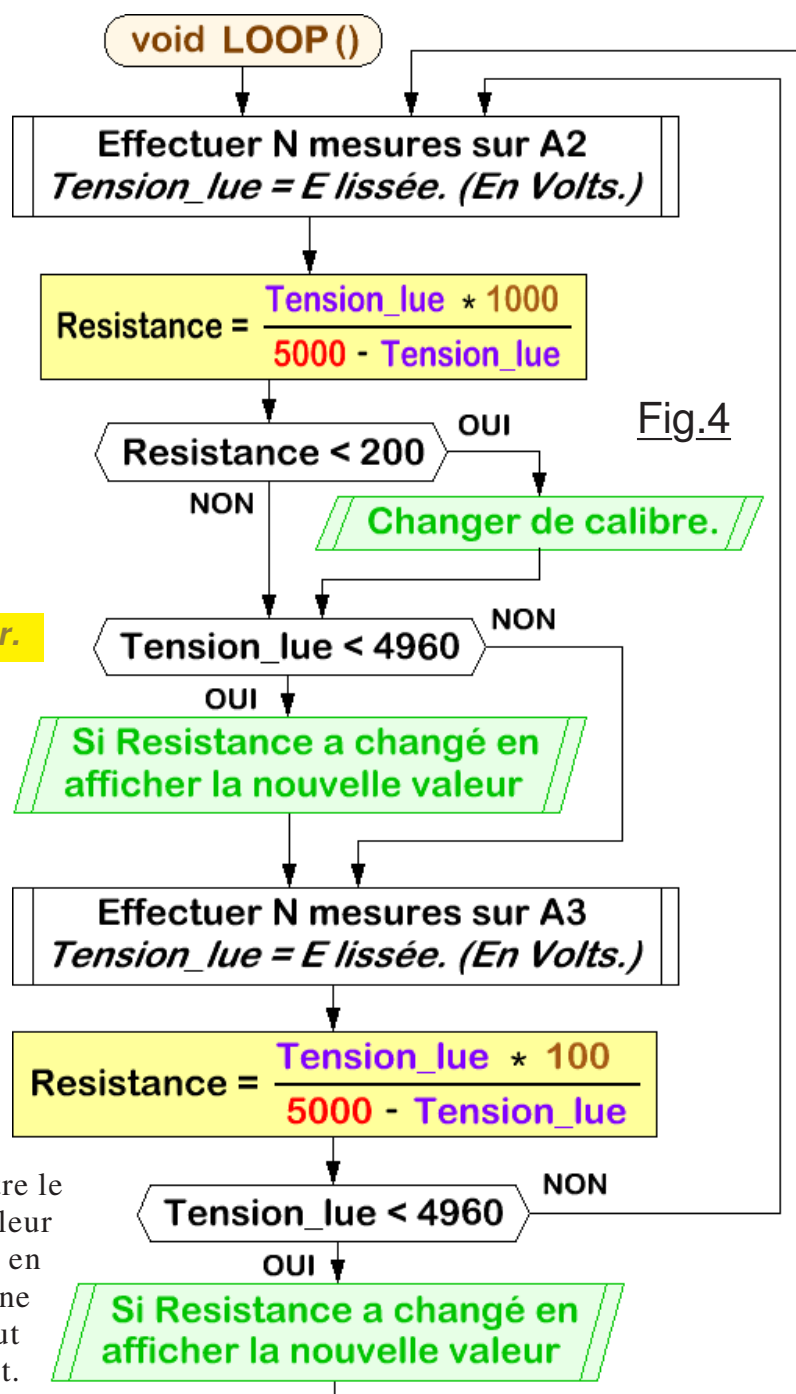
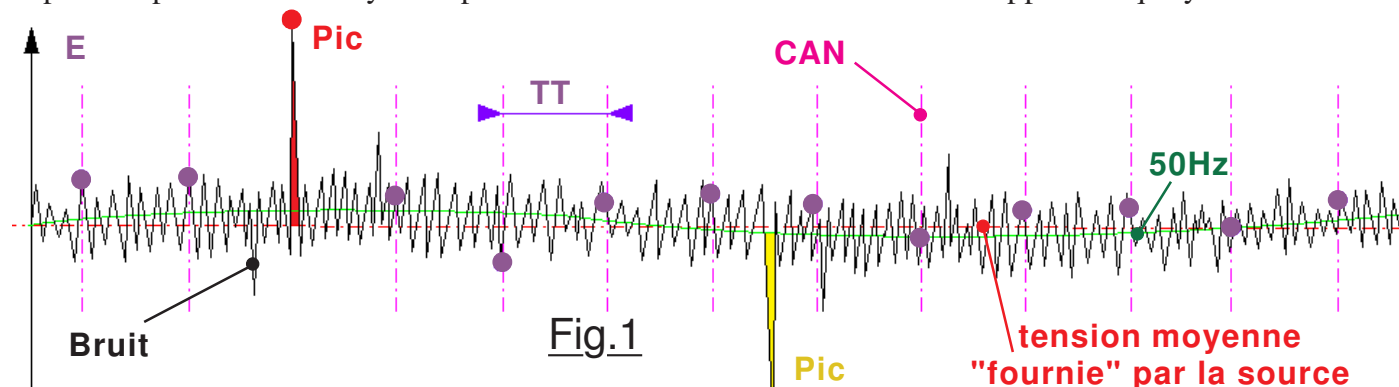


Fig.4

Sans aller jusqu'à développer la théorie du signal ou les séries de Fourier avec mise en œuvre d'une FFT, un minimum d'explications s'imposent si l'on veut comprendre ce que l'on observe quand on utilise un Convertisseur Analogique vers Numérique. (**CAN**) Considérons la Fig.1 qui représente le signal que l'on mesure sur une entrée analogique d'Arduino par exemple. La source mesurée présente une tension moyenne représentée par le trait mixte rouge sur le dessin. Mais cette tension est mesurée au moyen de fils qui vont de la source à l'entrée **E**. Inévitablement cette ligne va capter un peu de 50Hz rayonné par l'environnement du secteur et les appareils qui y sont reliés.



À ce signal parasite (*Pseudo sinusoïdal et 100Hz ...*) s'ajoute ce que l'on convient de nommer "du bruit". Ce phénomène omniprésent se traduit par des fluctuations rapides et permanentes d'un petit signal qui semble s'ajouter à celui de la source. Les origines en sont nombreuses, et en particulier l'agitation moléculaire issue de la chaleur des composants. Un peu comme les mouvements Browniens dans l'air qui nous entoure. Ce chuintement électronique est nommé "bruit", car il se traduit par du "souffle" plus ou moins important sur les systèmes audio. Enfin, de temps en temps apparaissent des **Pics** très courts en durée, mais d'intensité souvent notable par rapport au "bruit".

Considérons maintenant la phase de numérisation. Sur la Fig.1 **TT** représente le Temps de Traitement, c'est à dire le délai qui s'écoule entre deux affichages sur l'écran. Il inclut le temps de conversion du **CAN**, les routines de calculs, le formatage pour afficher et le délai qu'il faut pour envoyer les données sur la ligne série USB. Sur la Fig.1 les traits verticaux en rose représentent les instants où sont effectués les échantillonnages de valeurs "binaires". Chaque numérisation est séparée de la suivante par la durée **TT** qui représente en fait la durée de la boucle de base **void LOOP**. Chaque petit cercle violet ● représente la valeur instantanée qui a été convertie en "0" et "1" pour être ensuite traitée par le programme. **CATASTROPHE !** On constate que chaque valeur est différente de la précédente. Pas beaucoup, mais différente et l'affichage à l'écran va varier en permanence le rendant pratiquement "illisible". Avec le **Pic** nous avons de la chance, car il est passé inaperçu. Mais **Pic** pour son compte va retourner une valeur aberrante. Et encore, dans ce fatras nous n'avons pas cité les fluctuations qui résultent du convertisseur **CAN** qui pour son propre compte va apporter sa contribution à ce joyeux chamboulement.

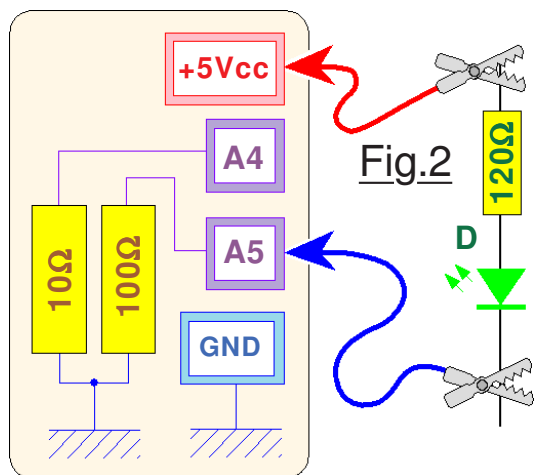
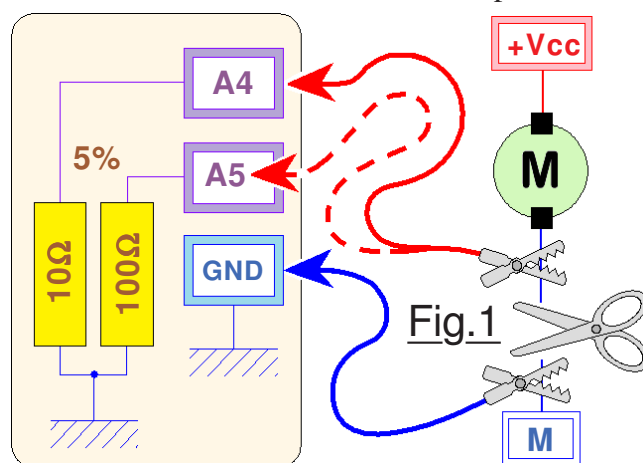
Comment s'affranchir de toutes ces fluctuations ? Heureusement la réponse est à la fois facile à trouver et à mettre en œuvre dans un programme informatique codé en langage C. **Il suffit de LISSER**. Mathématiquement on pourrait parler d'intégration, mais n'entrons pas dans ces détails scabreux. En observant avec attention la Fig.1 on constate que des valeurs numérisées sont plus importantes que celle moyenne que l'on désire "stabiliser". D'autres sont en dessous du trait mixte rouge. Statistiquement, si l'on échantillonne un grand nombre de fois, on finit par avoir autant de dépassements vers le "haut" que "vers" le bas. Il suffit de faire un grand nombre de mesures et d'en faire la moyenne. Plus le nombre de mesures est important, plus la durée d'une **CAN** va augmenter, mais avec les microcontrôleurs à 16MHz actuels on peut avoir des nombres de plusieurs centaines. Autant dire que les **Pics** vont être totalement exclus de notre "paysage herbeux". La solution est donc élémentaire : N mesures en boucle dont on fait la somme puis calcul de leur moyenne en divisant cette somme par N. **On a ainsi effectué un lissage informatique**, méthode utilisable chaque fois qu'un programme mettra en œuvre un **CAN** dont les fluctuations sont préjudiciables.

LA MESURE DES INTENSITÉ avec l'ATmega328 :

A ssez semblable à la mesure des résistances, mesurer une intensité revient à placer une résistance en série dans le circuit en mesurage et à numériser la tension à ses bornes. De la même façon que pour la détermination des résistances, deux calibres se sont montrés nécessaires pour couvrir une plage importante de valeurs mesurables. Ils servent à "étaler" la sensibilité de notre ampèremètre. C'est d'autant plus facile à envisager que le programme précédent laisse les deux entrées analogiques **A4** et **A5** entièrement disponibles. On se doute que ces expériences visent à globaliser le tout dans un programme plus complet qui transformera Arduino en un multimètre numérique à plusieurs "options". Ce chapitre permet de défricher la mesure des courants électriques.

Le schéma électrique.

Incontestablement élémentaire comme on peut le vérifier sur la Fig.1 il se résume à deux résistances SHUNT. L'une de 10Ω pour mesurer entre 50mA et 500mA. L'autre de 100Ω pour une évaluation dix fois plus fine et ainsi numériser entre 0 et 50mA. Envisager des intensités plus importantes serait possible, il suffirait d'utiliser une valeur de shunt dix fois ou cent fois plus faible, mais l'expérience montre qu'avec ces deux plages on couvre largement nos "besoins courants avec Arduino". Rien n'interdit du reste d'utiliser en parallèle sur l'entrée **A4** ou l'entrée **A5** un shunt complémentaire et de multiplier la valeur affichée par 10 ou 100. Pour brancher un ampèremètre, en "standard" on ouvre le circuit à mesurer et l'on insère en série un shunt de précision. Dans notre cas, on se contentera de résistances disponibles facilement et peu onéreuses. Pour ajuster la précision des mesures un simple coefficient multiplicateur sera inséré dans les calculs. C'est l'un



des avantages de l'informatique. **ATTENTION** : L'appareil de mesure étant branché sur le circuit à tester, il se trouve à son potentiel. Il est donc impératif qu'**Arduino soit sur sa propre alimentation avec une bonne isolation galvanique du secteur 220V~**. Ceci dit, rien n'interdit comme montré en Fig.2 d'évaluer la consommation d'un composant utilisé sur Arduino à condition d'accepter une masse commune. Par exemple sur la Fig.2 on désire déterminer la résistance qu'il faudra employer en série avec la diode **D** qui sera branchée sur une sortie binaire d'Arduino. Quand on aboutit à l'intensité désirée conforme aux caractéristiques nominales, le composant mis en série fait dans notre cas 120Ω . La résistance à prévoir dans le montage définitif sera donc de $100\Omega + 120\Omega$ soit 220Ω car il faut tenir compte de la présence du **shunt**.

L'algorithme utilisé.

Largement inspiré de celui relatif à l'ohmmètre il en reprend l'architecture. Représenté en Fig.3 sa structure en est très semblable et reprend les mesures avec lissage. Comme précédemment, l'affichage n'est rafraîchi que si la valeur mesurée a changé. Si la mesure fluctue en permanence, c'est que l'alimentation du circuit en test présente un fort taux de résiduel alternatif. Anticipant l'étude d'un programme complet intégrant les modules précédents, créer une procédure paramétrée pour effectuer toutes les mesures devient une évidence pour minimiser la taille du code. Le rang de l'entrée utilisée sera alors passé en paramètre sous la forme d'un **byte**. La formule de calcul pour déterminer la valeur de l'intensité dérive ici aussi de l'incontournable loi d'Ohm. On mesure une tension aux bornes d'une résistance qui pour la circonstance est nommée SHUNT. L'intensité qui la traverse est donc égale à $I = U / R$, dans laquelle U est la tension mesurée par **A4** ou **A5**.

Utilisation de SHUNT avec "précision logicielle".

Tributaire de la précision des résistances qui sont employées sur les multimètres analogiques à galvanomètres mobiles, les shunts sont obligatoirement des composants à faible dispersion de caractéristiques. Soit ce sont des résistances de précision, soit une résistance courante associée à un mini-potentiomètre pour affiner en usine la déviation pour chaque calibre de l'appareil. Grâce à la facilité de calcul qu'apporte le microcontrôleur, on peut sans aucun problème utiliser des composants ordinaires, et corriger par calcul le résultat de la mesure pour compenser l'écart de précision. Dans `P02_Amperemetre_sur_USB.ino` qui assure la fonction ampèremètre décrite ici, la formule de calcul donnée en bas de la page 8 est légèrement complétée : Les paramètres **K10** et **K100** sont deux coefficients multiplicateurs qui ajustent avec précision les deux calibres. Pour les déterminer, on impose une intensité correspondant au maximum de la plage mesurable, le courant réel étant vérifiée avec un appareil extérieur placé en série avec la charge. Puis, un simple calcul de proportion permet de déterminer la valeur à donner au coefficient correcteur. Prenons l'exemple donné en Fig.4 pour lequel on procède à la correction pour la résistance de **10Ω**. Quand on ajuste l'intensité réelle à exactement 500mA le système Arduino affiche 481mA. La résistance du **shunt** est donc légèrement trop faible. Mais l'erreur qu'elle engendre est proportionnelle à la valeur de l'intensité. En conséquence, un simple coefficient multiplicateur permet de rétablir l'exactitude des mesures qui ne seront plus influencées globalement que par la perturbation du circuit en essai par l'insertion en série du **shunt**, et par l'erreur apportée par la conversion analogique vers numérique. La formule de calcul pour déterminer la valeur du coefficient est :

$$I = \frac{U \times K_i}{R_{SHUNT}}$$

void LOOP ()

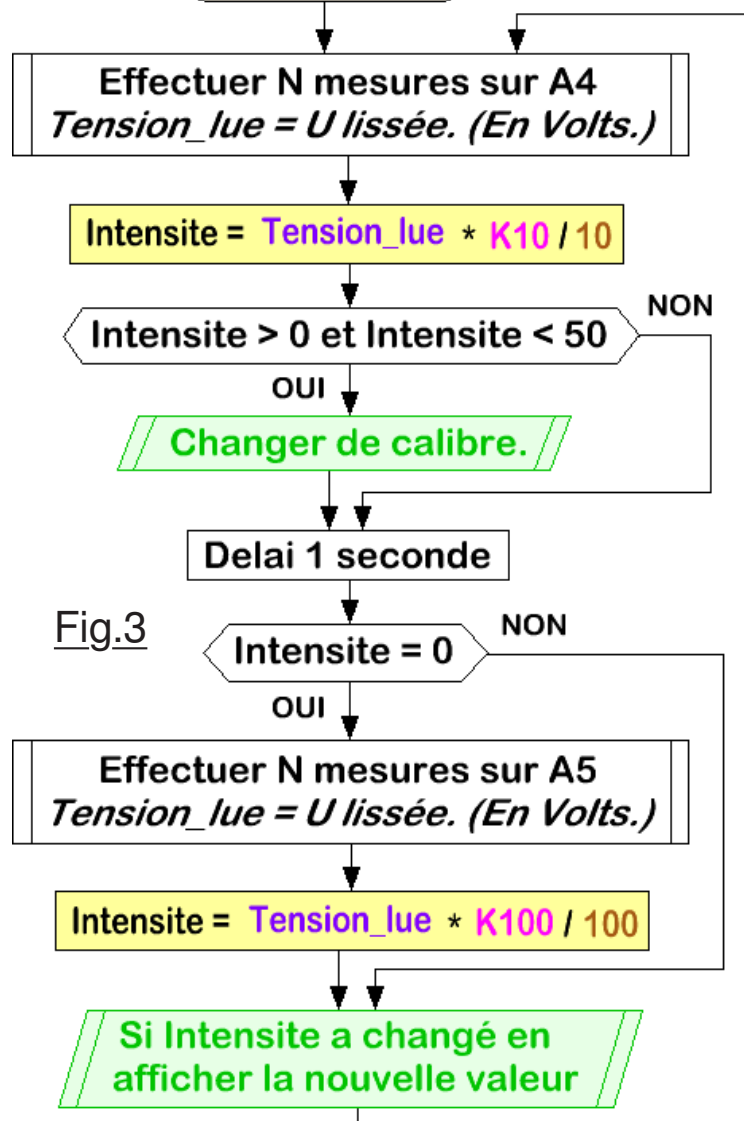


Fig.3

$$K_i = U_{réel} / U_{affiché}$$

$K_{10} = 500 / 481 = 1.04$ valeur qui est introduite dans le programme sous forme d'une définition en tête des déclarations et ainsi permet de faciliter la "transportabilité" du source sur d'autres applications.

`P02_Amperemetre_sur_USB.ino` lors de la compilation conduit à une taille de 5372 octets. On en déduit qu'assembler de multiples fonctions en un programme unique sera largement faisable, et un multimètre complet sera alors émulé par un module Arduino auquel il ne faut ajouter que quelques résistances. L'ATmega328 est un microcontrôleur vraiment convivial.

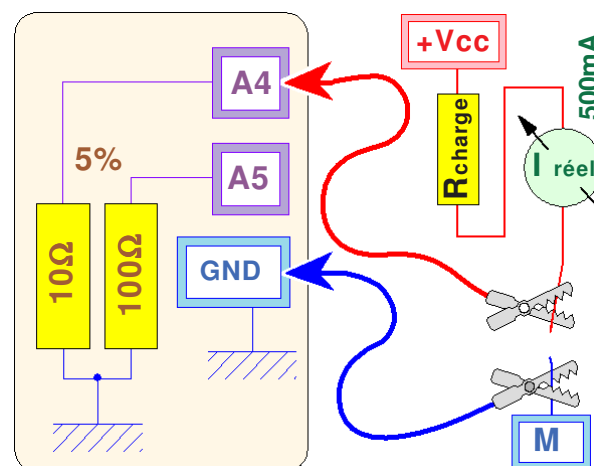


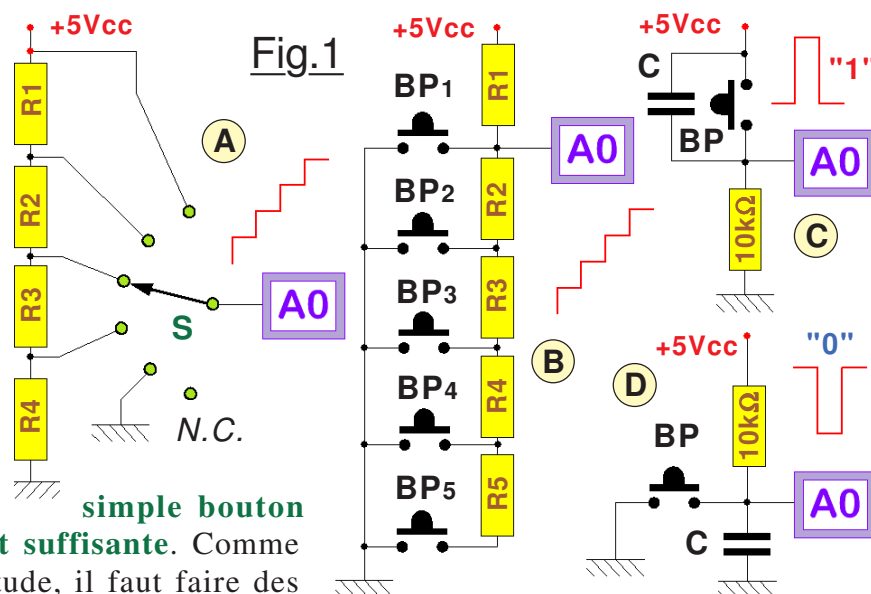
Fig.4

Mesures de base et de natures diverses avec l'ATmega328 :

L'obligation d'exploiter la ligne série USB pour lire les valeurs retournées par les mesures ne peut se justifier que pour une utilisation exceptionnelle. Si l'on désire réaliser un multimètre réellement fonctionnel, il faudra impérativement le munir d'un dispositif d'affichage autonome. L'un des procédés les plus courants et les moins onéreux consiste à associer un afficheur LCD, solution qui s'impose d'autant plus vivement que des SHIELD spécifiques sont prévus pour Arduino UNO. Mais avant d'envisager le développement d'un programme complet qui gère un écran LCD avec un formatage "confortable" des données, il me semble judicieux de passer par l'étape intermédiaire d'une exploitation par la voie série USB. Ce chapitre dégrossit l'analyse de la sélection des fonctions.

Passer de deux programmes indépendants à un logiciel unique (*Qui affecte l'entrée A1 pour mesurer les tensions.*) consiste à ajouter un dispositif quelconque qui permettra à l'opérateur de choisir à la demande la fonction à activer pour effectuer ses manipulations. Dans le but de laisser au lecteur le choix entre les solutions listées sur la Fig.1 pour son projet personnel, c'est A0 qui permettra à l'utilisateur de choisir la fonction qu'il désire mettre en service, bien que pour un simple bouton poussoir une entrée binaire serait suffisante. Comme dans toute application en cours d'étude, il faut faire des choix qui vont conditionner significativement la qualité opérationnelle du produit définitif. Dans notre cas, plusieurs options très courantes représentées en Fig.1 sont possibles. La première en A consiste à ajouter un sélecteur S et des résistances pour créer un diviseur de tension. Le logiciel détermine par la tension mesurée sur A0 sa position et passe à la fonction désirée. Un simple regard sur le bouton flèche de ce commutateur renseigne l'opérateur sur la fonction émulée. C'est idéal sur le plan opérationnel, mais l'encombrement et le coût sont plus importants que l'usage d'un élémentaire bouton poussoir associé à une seule résistance. La solution souvent utilisée montrée en B consiste à employer des boutons poussoir, chacun étant affecté à l'appel de l'une des fonctions spécifiques. Remarquer qu'elle n'est pas optimisée, car si l'on branche comme pour le commutateur S on gagne une résistance. Manifestement plus simples, les solutions C et D n'utilisent qu'un seul bouton poussoir BP en logique positive ou en logique négative. (Éventuellement complété d'un condensateur C "anti rebonds".) Pour choisir la fonction il suffit d'appuyer dessus, les options "défilant" en permutations circulaires. C'est un peu moins convivial que pour les deux solutions précédentes, mais en termes de coût et de volume cette solution technique est de loin la plus séduisante.

L'anti rebonds peut se faire par l'ajout d'un condensateur C (Dans les quatre cas.) ou par logiciel. Personnellement je ne suis pas favorable à la présence d'un condensateur C qui présente deux inconvénients : D'une part c'est un composant de plus à ajouter au montage qui en augmente le coût. Très peu, c'est vrai, mais je préfère "dépenser" quelques octets de plus, solution qui m'apparaît comme plus élégante, surtout si l'on dispose de place dans la mémoire de programme. Par ailleurs, si l'on veut que le condensateur soit efficace, il faut lui attribuer une capacité notable, par exemple 0,1µF. Chaque fois que l'on appui sur un B.P. il se décharge brutalement sur ce dernier qui se comporte comme un court-circuit. Ce n'est bon ni pour les grains du bouton poussoir, ni pour le condensateur. En fin de comptes, c'est la solution C ou la solution D qui sera retenue, et sans la présence du condensateur. Pour opter entre la logique positive et la logique négative, à ce stade de l'analyse aucun critère objectif n'est disponible. Si aucun argument "de programmation" n'émerge lors du développement, ce sera la solution C qui emportera la décision. D'une part raisonner "positif" est plus naturel, d'autre part la masse est généralement moins sujette à la présence de parasites.



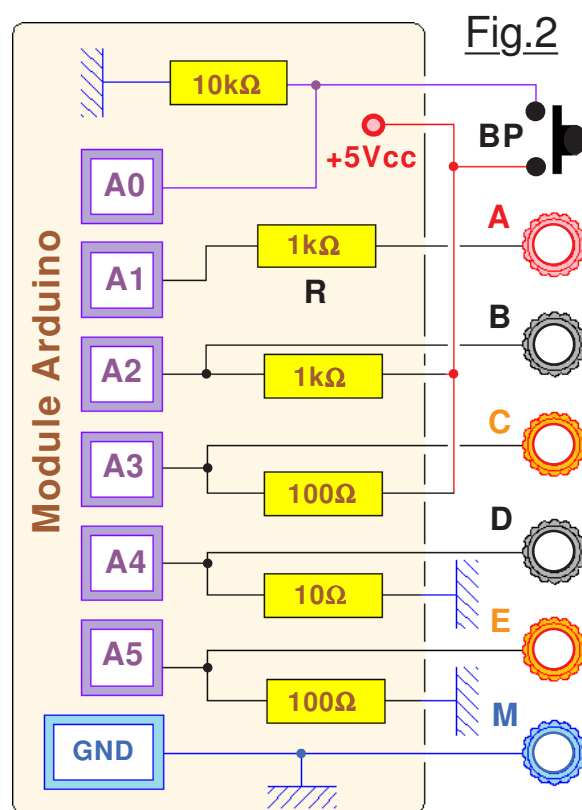
Le schéma électrique.

Réduit à sa plus simple expression il est représenté sur la Fig.2 et se contente de reprendre la présence des divers SHUNTS et se complète avec le bouton poussoir branché sur **A0**. Comme précisé en bas de la page 10 on opte initialement pour un comportement en logique positive avec la classique résistance de 10kΩ qui procure un bon compromis entre faible consommation à l'activation du **BP** et immunité convenable aux parasites hertziens. Le tableau ci-dessous résume les fonctions :

| ENTRÉE | Fonction | Calibre |
|----------|-------------|--------------|
| A | Voltmètre | 0 à 5Vcc |
| B | Ohmmètre | 100Ω à 150kΩ |
| C | Ohmmètre | 0 à 100Ω |
| D | Ampèremètre | 50mA à 500mA |
| E | Ampèremètre | 0 à 50mA |

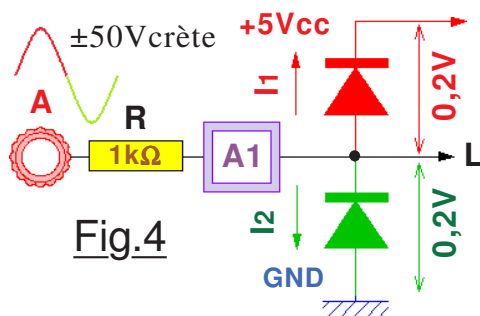
* En couleur orange les entrées de mesures à n'utiliser que sur invitation du logiciel.

* Toutes les mesures se font entre l'entrée dédiée à la fonction désirée et la masse **M**



La protection des entrées.

Seule l'entrée du voltmètre sera protégée sur notre multimètre car c'est celle qui statistiquement est la plus employée. Mais rien ne s'oppose à utiliser cette technique sur toutes les autres fonctions si on le désire. Pour comprendre le principe de la protection examinons la Fig.3 qui décrit très sommairement la conception des entrées sur l'ATmega328. L'entrée **A1** va vers la logique interne **L** qui traite la CAN, le mode sortie etc. Pour éviter que l'électronique en aval de **L** ne soit soumise à des tensions supérieures à +5,2V et -0,2V les diodes **D1** et **D2** drainent le courant vers le +5Vcc ou vers la masse **GND**. **ATTENTION : Cette protection est illusoire** car si l'on applique en **A** une tension exagérée positive ou négative, les courants **I1** et **I2** seront excessifs et détruiront **D1** ou **D2** puis en aval de **L** l'électronique interne du microcontrôleur. Comme montré sur la Fig.4 il suffit de limiter les courants possibles **I1** et **I2** en insérant une résistance **R** entre **A** et l'entrée **A1**. Sa valeur devra être d'autant plus élevée que la tension accidentelle potentielle est de grandeur élevée. Ce composant



en amont de **A1** ne modifie pas les valeurs converties par le CAN compte tenu de l'impédance d'entrée très élevée de ce dernier. On peut facilement l'augmenter jusqu'à 20kΩ sans altérer la précision des mesures. On peut se demander pourquoi les concepteurs du processeur n'ont pas inclus cette résistance dès la fabrication du circuit. Tout simplement parce que **A1** peut fort bien être initialisée en sortie et ainsi pouvoir drainer un courant jusqu'à 40mA. Supposons que l'on applique par accident une tension alternative de 35V~ efficace. La tension sera de ±50Vcrête. C'est la

diode **D2** qui subira le courant le plus élevé puisque les +5Vcc ne sont pas déduits de la surcharge en entrée **A**. Le courant crête avoisinera $50 / 1000 = 50\text{mA}$. Comme seule l'alternance négative va la concerner, il faut diviser le résultat par deux. Mais c'est la valeur efficace qui fait chauffer, il faut donc encore diviser par la racine carrée de deux. Au final cette diode de protection ne sera soumise qu'à un courant efficace de 18mA, pas de quoi l'effrayer. Quand à la résistance **R**, elle devra dissiper une puissance de 1,2W un modèle prévu pour 1/4W sera suffisant car l'avertissement lumineux préviendra l'opérateur

et la surcharge ne sera logiquement que de courte durée.

Avertissement lumineux de surcharge.

C'est un luxe que l'on peut allègrement se permettre car il ne grèvera notre budget que de deux LED et d'une résistance, soit un coût supplémentaire assez dérisoire. Notre multimètre présentera ainsi un aspect très professionnel. L'idée de base est montrée sur la Fig.5 sur laquelle on constate que la surcharge positive sera signalée par une LED rouge alors que l'incident d'inversion de polarité allumera une diode verte.

(Personnellement j'ai utilisé une jaune qui était disponible.) Les modèles utilisés sont à très bon rendement pour s'éclairer dès qu'un petit courant les traverse. Ainsi nous serons également prévenus pour les faibles surcharges qui ne mettent pas en péril l'électronique, mais qui engendrent un résultat

de mesure saturé à +5V ou faussé à zéro. Si on branche **A** sur une source de courant alternatif les deux LED seront allumées simultanément, sauf si la tension du pic positif ne dépasse pas 5,2V en crête. Bien que séduisant de simplicité, ce schéma ne fonctionne pas et les LED restent désespérément éteintes. Pour bien montrer que dans la réalité **D1** est en parallèle sur **LED1** et **D2** en parallèle sur **LED2** la "borne" d'entrée **A1** est, sur la Fig.5, traversée par la liaison électrique qu'il ne faut pas oublier. Hors **D1**

et **D2** ont des tensions de seuil faibles de l'ordre de 0,2V. Une LED pour pouvoir s'illuminer doit avoir à ses bornes un minimum de 0,3V et de ce fait nos deux témoins lumineux restent inertes. La solution est simple. La Fig.6 montre qu'il suffit d'ajouter une résistance **R2** de 4,7kΩ avant **A1** pour ne plus que **D1** et **D2** ne viennent perturber les LED. Comme la résistance totale placée en amont du convertisseur analogique numérique ne fait que 5,7kΩ, la présence de ces deux composants n'aura aucune influence sur la précision des mesures. On peut maintenant concevoir le "circuit imprimé".

Le petit circuit imprimé qui supporte les divers shunts.

Compte tenu du faible nombre de composants à insérer dans notre appareil de mesure, la plaquette d'essai cuivrée qui va leur servir de support présente de faibles dimensions, et ce d'autant plus que les LED d'avertissement de surcharge et le bouton poussoir sont placés en face avant. Représenté sur la Fig.7 le petit circuit imprimé pré percé au pas de 2,54 mm de quelques centimètres carrés est représenté avec un connecteur femelle pour son raccordement avec Arduino. Ce n'est pas du tout impératif, on peut parfaitement utiliser une limande à 8 fils du type nappe de connecteurs d'ordinateur. La Fig.7 représente en **A** la plaquette vue côté composants ainsi qu'en **B** les liaisons à établir avec la face avant du coffret qui enferme le tout. La Fig.8 en **C** est identique à la Fig.7 mais les pistes de cuivre sont vues comme si la plaquette

Vue côté composants.

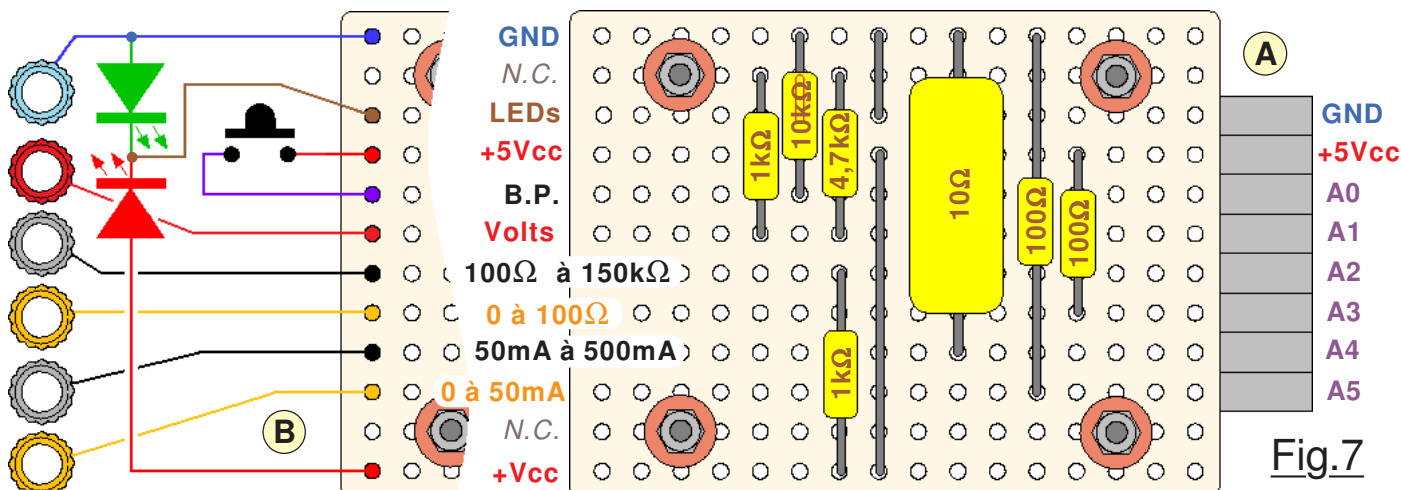
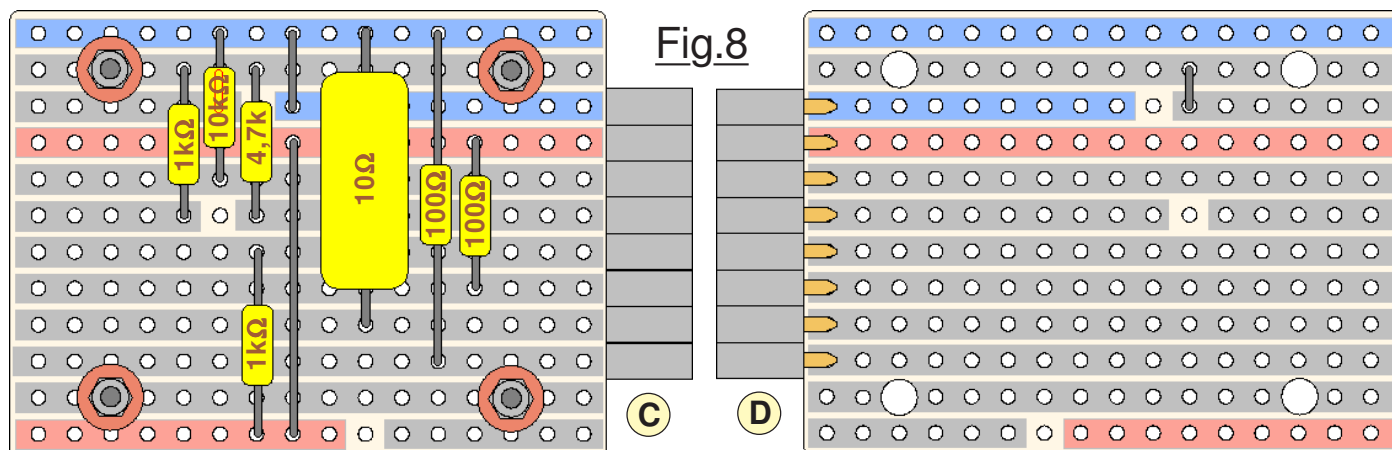


Fig.7



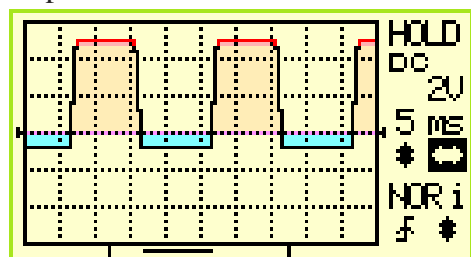
était transparente pour mieux comparer le dessin avec le schéma prévu. Le dessin de la figure **D** représente le circuit imprimé vu coté cuivre avec les trous de perçage et les trois coupures de pistes. Notez la présence du petit fil qui sera soudé sur le dessous pour réunir les deux pistes voisines.

L'aspect logiciel.

Conçu initialement comme un démonstrateur en vue d'élaborer un multimètre autonome avec son propre système d'affichage des menus et des données, il n'en constitue pas moins un logiciel complet parfaitement exploitable si l'on accepte de mobiliser un ordinateur et une prise USB associée au terminal série de l'IDE d'Arduino. C'est `P03_Multimetre_sur_USB.ino` qui se charge d'animer le fonctionnement de notre appareil de mesure. Pour une utilisation agréable il faut impérativement que le bouton poussoir d'appel des fonctions réagisse immédiatement et à tout moment. Pour atteindre cet objectif la boucle de base y fait appel par "sécurité", mais le programme passe une majorité du temps dans la procédure de mesure `N_mesures_sur_Ai()`. C'est la raison pour laquelle est inséré directement dans la boucle de "*Lissage*" la procédure `Tester_le_BP_A0()`. Cette dernière n'est pas pénalisante en temps de traitement car il y a sortie immédiate si le **B.P.** n'est pas activé. Cette procédure réalise l'anti rebonds, attend la stabilisation du relâcher et se contente de signaler qu'il y a eu un "clic" sur le bouton poussoir. Comme montré sur la Fig.9, c'est ensuite dans la boucle de base que la requête est prise en compte, que la fonction indexée est incrémentée et que le titre de la fonction en cours est affiché. La boucle de base traite ensuite la fonction active.

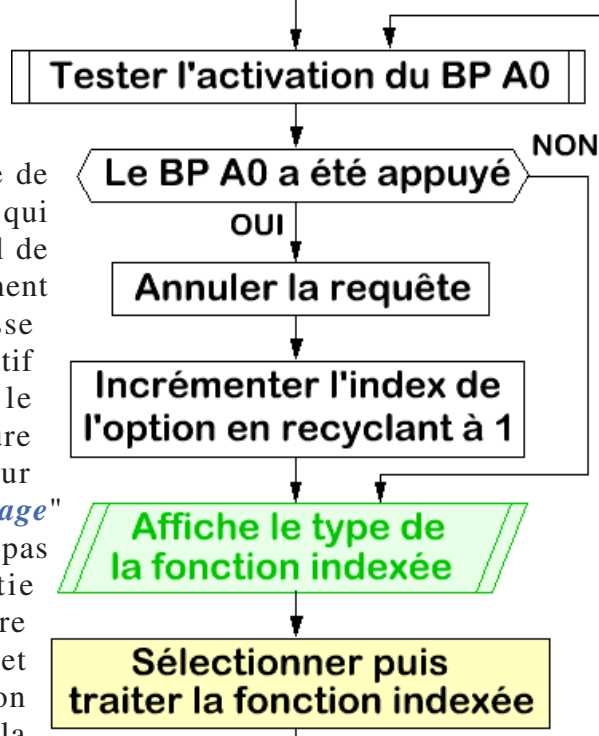
Vérification de la protection avec témoins lumineux.

Lorsque le programme a présenté le comportement attendu sur toutes les fonctions, que le bouton poussoir s'est avéré d'une utilisation tout à fait conviviale, que les résultats affichés pour les mesures se sont montrés conformes à ceux des appareils de contrôle indépendants, il importait de vérifier la fiabilité du système de protection présenté en Fig.6 et d'en valider le concept.



Dans ce but, l'entrée **Volts** à été branchée sur la sortie d'un transformateur fournissant 15V~ efficace soit $\pm 21V$ crête pendant une très longue période. La copie d'écran d'un petit oscilloscope numérique ci-contre montre que les tensions sur le nœud entre les deux résistances et les deux diodes variaient entre -0,3V et +5.6V valeurs strictement sans danger pour **A1** protégée par la 4,7kΩ. La résistance de 1kΩ ne dissipe que 0,25W sans problème.

void LOOP() Fig.9



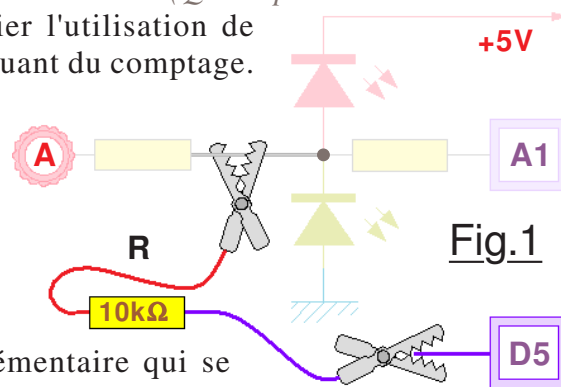
Comptage évènementiel et chronométrage avec l'ATmega328 :

Vocabulaire accrocheur qui masque une grande simplicité, car compter est l'une des fonctions les plus simples à mettre en œuvre sur un microcontrôleur quel qu'il soit. On peut presque affirmer que ces circuits intégrés sont fondamentalement conçus pour ça ! C'est une fonction tellement simple à mettre en œuvre que ce chapitre pourrait facilement être éludé. Il n'est toutefois pas judicieux d'en faire l'impasse, car il va nous permettre d'introduire les concepts de base qui permettent de lier une action de comptage au "temps qui s'écoule". On se doute que l'aboutissement de ces études permettra plus avant d'agencer la fonction Fréquence-mètre / Période-mètre. Commençons par le plus élémentaire :

Composants à ajouter pour la fonction comptage d'événements.

Sachant que les signaux que nous allons utiliser pour effectuer du comptage évènementiel seront issus de capteurs actuellement indéterminés, la logique incite à continuer à utiliser l'entrée **A** qui s'accommode sans risque de tensions pouvant être élevées et alternatives. Par ailleurs, l'étude du fréquence-mètre, avec anticipation de la présence d'un afficheur LCD (*Qui imposera la réservation de plusieurs E/S pour le piloter*) nous amène à privilégier l'utilisation de l'entrée binaire **D5** pour servir les diverses fonctions effectuant du comptage.

Comme ce chapitre n'est qu'une étape de transition servant d'études préliminaires, inutile de réaliser un circuit imprimé spécifique. Nous envisagerons une nouvelle version quand l'intégralité du schéma électronique sera déterminée. Pour cette manipulation on va se contenter, comme montré sur la Fig.1, de brancher "en provisoire" une résistance de 10kΩ entre le "commun" de l'entrée **A** et la broche **D5** du microcontrôleur. Cette liaison supplémentaire qui se résume à une seule résistance de plus dans notre appareil de mesure sera suffisante pour pouvoir agencer plusieurs fonctions nouvelles dans notre projet. Économique non ?

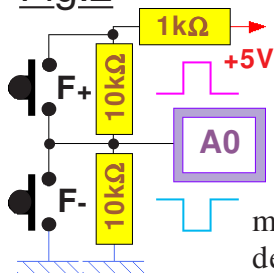


Deux boutons poussoir ... c'est tellement plus agréable !

Compter se résume à ajouter une unité à une variable entière, et à en afficher la grandeur que si cette dernière a changé de valeur. Informatiquement nous n'allons pas en attraper une migraine. Il importe simplement de pouvoir remettre à zéro ce compteur à convenance, et pouvoir sortir de la fonction quand on le désire ... sauf que pour le moment notre appareil de mesure est sensé ne disposer que d'un seul bouton poussoir pour tout faire. C'est faisable. Par exemple toute action sur **BP** force la valeur à zéro. Tout enfoncement de plus de deux ou trois secondes ramène à la fonction première de mesure des tensions. Rien de bien compliqué. Toutefois, on sent confusément que le nombre de fonctions possibles sur le multimètre va devenir significatif. Un appui de trop sur **BP** et l'on doit refaire un cycle complet pour "recirculer" jusqu'à l'option désirée. Avec l'agacement d'avoir été trop rapide la première tentative, on devient fébrile, et la mésaventure va se reproduire.

CE N'EST PAS CONVIVIAL ! Ceux qui vont choisir un sélecteur adopteront un modèle à douze positions facile à approvisionner dans le commerce et la difficulté est contournée. Mais si l'on désire privilégier la compacité de notre appareil de mesures, on doit dès maintenant orienter la solution future vers l'usage de deux boutons poussoir. L'un utilisé pour faire "avancer" dans la liste des fonctions, l'autre qui permet de recirculer en "rétrograde". On pourra ainsi agréablement "se déplacer" librement dans les deux sens de la liste des fonctions, avec parfois la facilité d'alterner rapidement entre deux options d'une même famille de mesurages. La solution de la Fig.1

Fig.2



en **B** sur la page 10 s'impose alors naturellement pour ne pas avoir à "consommer" une entrée de plus sur l'ATmega328. Après réflexion on arrive au schéma de la Fig.2 qui oblige à ajouter deux résistances. Si vous n'entrevoiez pas l'utilité de celle de 1kΩ il suffit par la pensée de l'éliminer. Que se passerait-il alors si l'on enfonçait les deux boutons poussoir simultanément ?

Le logiciel va devoir effectuer de l'anti rebonds non plus sur des "0" et des "1", mais sur des seuils de plages de valeurs tout en filtrant simultanément les problèmes de parasites électriques se superposant à la tension continue mesurée sur **A0**.

La gestion analogique de plusieurs B.P.

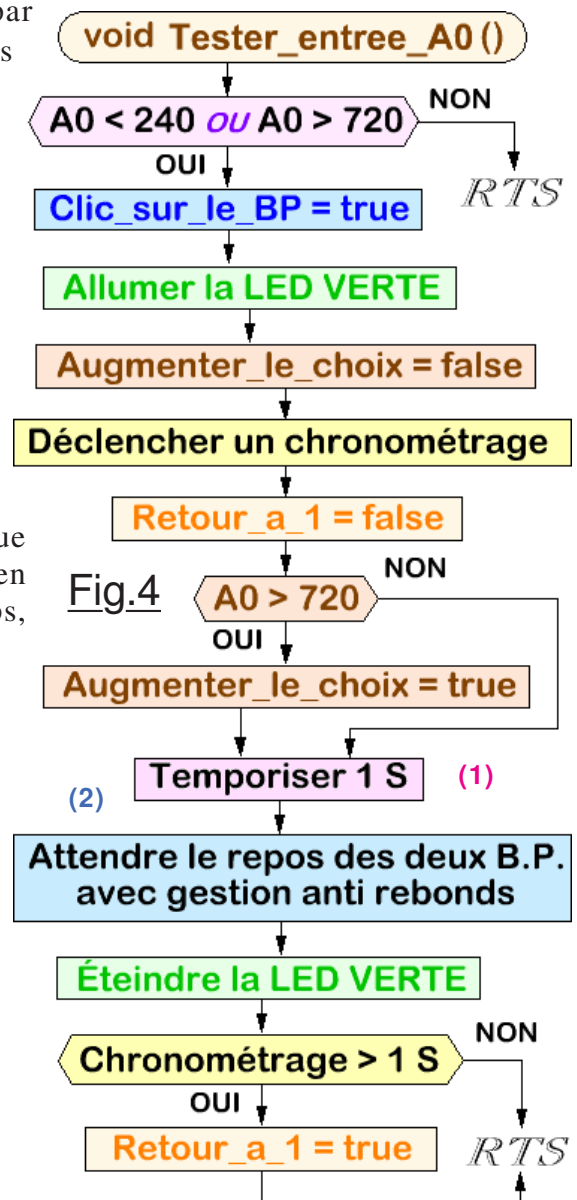
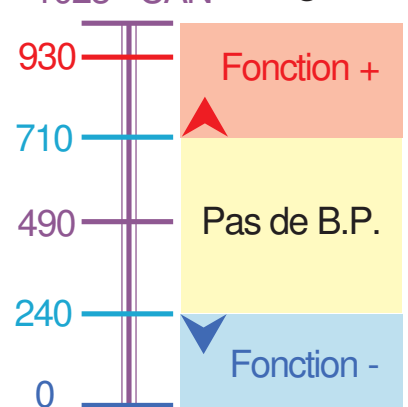
Bien que traité sur deux boutons poussoirs dans le cadre de notre projet en cours d'études, cette technique permet sans problème de gérer jusqu'à six ou huit B.P. avec une seule entrée analogique. Il suffit d'adapter les plages de tolérance et les seuils de détection. Les explications seront relatives au schéma de la Fig.2 mais il sera très facile "d'interpoler" pour le cas d'éléments multiples. Considérons la Fig.3 sur laquelle figurent les valeurs de numérisations issues de la CAN. Pour des raisons "esthétiques" ces dernières sont arrondies à la dizaine la plus proche. Quand aucun B.P. n'est enfoncé, la numérisation fluctue autour de 490 à quelques unités près. (*Présence de bruit comme déjà abordé en page 7.*) Quand on appuie sur le B.P. de Fonction - la valeur binaire retournée par le CAN est pratiquement un 0 constant. La plus grande valeur possible issue d'une CAN serait de 1023, mais la présence de la résistance de protection de 1kΩ dégrade à 930 la valeur numérisée quand Fonction + est activé. Pour lutter efficacement contre le bruit, mais surtout contre les Pics jamais totalement exclus, il suffit de calculer les moyennes entre les valeurs numérisées pour définir les seuils de décision. Tout ce qui sera supérieur à 710 sera considéré comme une action sur Fonction +, toute CAN inférieure à 240 dénoncera une action sur Fonction -.

Sans pour autant pénaliser de façon significative la taille du programme, il est aisé d'améliorer encore la commodité de sélection des fonctions par bouton poussoir. Si le nombre des options et des calibres est important, il serait agréable de pouvoir à tout moment réinitialiser la fonction de base n°1. Pour ce faire, il suffira d'appuyer sur l'un des boutons poussoir durant plus d'une seconde. Dans notre cas le menu reviendra à la fonction voltmètre sur le calibre +5Vcc. La Fig.4 présente l'organigramme de la procédure qui traite la surveillance de l'entrée A0. Outre son appel et ses variables locales, elle totalise exactement 400 octets soit 1,24% de la mémoire de programme. Ce n'est pas du tout exagéré si l'on prend en compte le fait que la qualité opérationnelle d'un appareil de mesure est directement fonction de la facilité qu'il y aura à naviguer entre les nombreuses options qui le caractérisent. La temporisation en (1) plus économique qu'une gestion anti rebonds la remplace parfaitement tout en assurant à la LED verte de s'allumer un minimum de temps, y compris si l'action sur l'un des deux B.P. est très courte. Par contre pour minimiser le temps passé dans cette procédure, la gestion anti rebonds en (2) pour l'attente du relâcher des B.P. est effectuée par comptage d'au moins cinquante lectures stables sur A0 avant de décider que le repos est réellement stabilisé. Dans cette version du logiciel, cette procédure positionne trois booléens qui seront pris en compte dans la boucle de base :

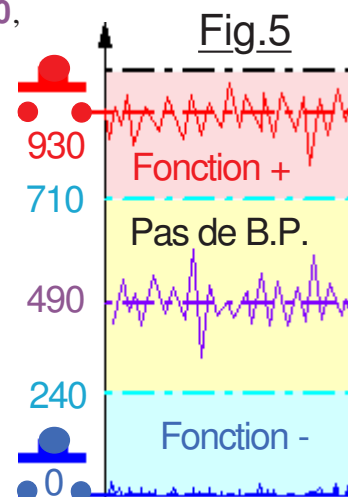
- Clic_sur_le_BP qui signale une requête,
- Augmenter_le_choix qui détermine le B.P. actionné,
- Retour_a_1 représentatif d'une action longue.

Quand Clic_sur_le_BP est à true, la boucle de base examine alors Augmenter_le_choix pour augmenter ou diminuer l'indexation dans le menu, avec priorité à Retour_a_1 qui force le premier choix de la liste du menu.

1023 CAN Fig.3



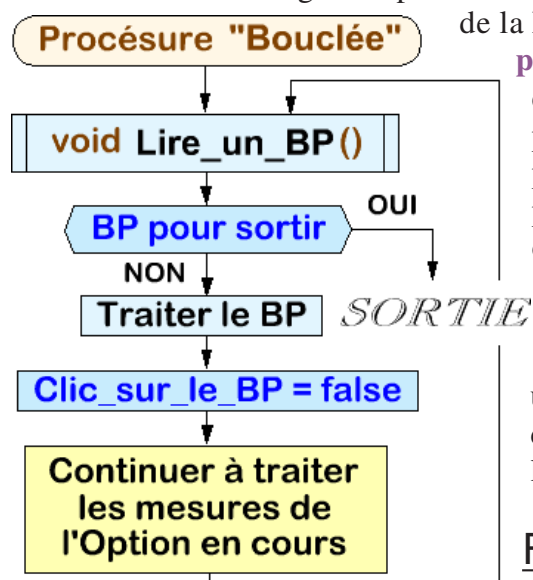
Caricaturant l'amplitude du bruit électronique présent sur l'entrée A0, la Fig.5 met en évidence l'importante immunité aux parasites que l'on peut obtenir quand l'étendue de numérisation ne comporte que trois plages. Ce dessin exagère vraiment la grandeur des PICs et "de l'herbe". Aussi, dans la réalité, on peut facilement diviser par six ou par huit la plage totale de numérisation sans pour autant diminuer significativement l'immunité aux parasites. Revenons à la structure du programme. Ce que ne montre pas l'organigramme de la Fig.4, c'est la faculté du microcontrôleur à effectuer simultanément plusieurs actions, et ce tout particulièrement quand il s'agit de compter. L'ATmega328 met à notre disposition plusieurs compteurs indépendants qui fonctionnent en autonomie totale une fois que le microcontrôleur a été correctement configuré. L'un d'eux reçoit en permanence un signal de 1000 Hz issu de l'horloge 16MHz pilotée par le quartz. Ce compteur est mis à zéro lors du RESET et "mouline en tâche de fond" sans autre forme de procès. On peut en ignorer la présence, mais pour effectuer un (*Ou plusieurs simultanés.*) chronométrage précis à la milliseconde près, il suffit d'en noter la grandeur au début de la mesure, et d'en soustraire cette donnée de la valeur de ce compteur à la fin de la période à mesurer. Le compilateur C de l'IDE d'Arduino met à notre disposition la fonction `millis()` qui retourne sous un **unsigned long** la valeur instantanée de ce compteur particulier. La grandeur exprimée est en millisecondes ce qui permet des chronométrages précis, et la taille de ce compteur est suffisante pour ne pas recycler à zéro avant ... 49,71 Jours ! La procédure `Tester_entree_A0` fait appel à ce compteur entre les deux instructions colorées en jaune sur le diagramme de la page 15. On convient de dire qu'il s'agit d'un **travail de fond masqué**.



Menu avec fonctions "bouclées".

L'informatique nous démontre souvent qu'un concept initial simple cache souvent des pièges inattendus. Le développement d'une routine qui ne devait nous prendre que quelques minutes ruine notre optimisme et la procédure en question se goinfre de plusieurs heures de mise au point. Rien de bien nouveau ... il faut faire avec. Le chapitre précédent rentre dans le cadre de ces petites mésaventures de programmeur, si l'une ou plusieurs des options dans le menu est une procédure "bouclée". De quoi s'agit-il ?

Considérons l'organigramme de la Fig.9 donné en page 13. Dans ce dernier, [**Sélectionner puis traiter la fonction indexée**] se résume à effectuer une mesure, et à en afficher la valeur si cette dernière a changé. On peut réduire cet organigramme à celui

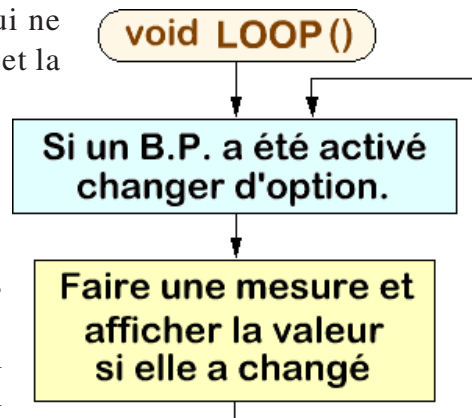


de la Fig.6 qui montre que [**MESURER**] est une

procédure ouverte dont la sortie est systématique après avoir effectué une mesure. Considérons maintenant la Fig.7 qui présente le cas d'une **procédure bouclée**. Contrairement au cas précédent dont la CAN était très courte et la sortie de la procédure pratiquement immédiate, dans une Option telle que du COMPTAGE, le mesurage est "continu". On est donc "enfermé" dans la procédure qui persistera dans une boucle fermée (*Pléonasme !*) tant qu'une action spécifique sur l'un des deux B.P. ne sera pas effectuée pour imposer une sortie et un retour à la boucle de base. Cette situation pose un problème de "convivialité" dans la gestion des menus. Pour comprendre la difficulté dans une telle situation pour agencer un appareil de mesures agréable à utiliser, il faut au préalable établir une stratégie d'utilisation des deux B.P.

Fig.7

Fig.6



Stratégie d'utilisation des deux boutons poussoir.

Puisque c'est la fonction COMPTAGE d'événements qui la première nous place dans une situation de procédure bouclée, c'est dans ce cas de figure que nous allons effectuer l'analyse. Contrairement à afficher la valeur instantanée d'une tension, d'un courant etc, **COMPTEUR** revient à afficher un nombre chaque fois que ce dernier change, et ce **JUSQU'À CE QUE L'OPÉRATEUR DÉCIDE D'ARRÊTER**. C'est donc par une action particulière sur l'un des deux B.P. que sera provoquée la sortie. Par ailleurs, la fonction en cours peut présenter plusieurs options particulières, dont la sélection ne peut se faire que par le truchement des deux boutons poussoir. Il faut alors définir le protocole d'utilisation des deux B.P. pour que leur utilisation soit la plus "naturelle" possible. De plus, si l'une des options locales est "dangereuse", comme la sortie inopinée de la procédure, la sécurité incite à privilégier une action longue sur l'un des deux B.P. C'est ce que l'on fait sur le bouton de RESET d'un ordinateur par exemple. Dans notre cas, un clic sera considéré comme long si il dépasse 0,5 seconde, cette valeur opérationnelle est déterminée expérimentalement.

Avant de définir une stratégie d'exploitation, examinons la combinatoire possible d'utilisation de deux B.P. sachant qu'appuyer sur les deux simultanément n'est pas une solution car elle engendre un 0 non différent de l'utilisation du B.P. **Fonction -**. Le petit tableau ci-dessous résume les différentes possibilités et montre que la combinatoire ne présente que quatre variantes. Comme l'une d'elle, celle **coloriée en rouge**, est réservée pour la **sortie de la procédure** bouclée, toutes les fonctions de ce type ne pourront **exploiter au maximum que trois "options locales"**. Notez au passage que la sortie étant "à risque", c'est donc une action longue qui sera systématiquement utilisée pour ça, et le B.P. de **Fonction +** étant prévu au dessus de celui de **Fonction -** sur notre maquette, cette position sera un rappel qui symbolise le "risque maximum". Nous avons tous les éléments en main pour choisir un protocole d'exploitation de la fonction COMPTEUR d'événements dont il nous faut lister les options envisageables.

| B.P. | Action courte | Action longue |
|---------------|---------------|---------------|
| Fonc + | 0 | 1 |
| Fonc + | 1 | 0 |
| Fonc - | 0 | 1 |
| Fonc - | 1 | 0 |

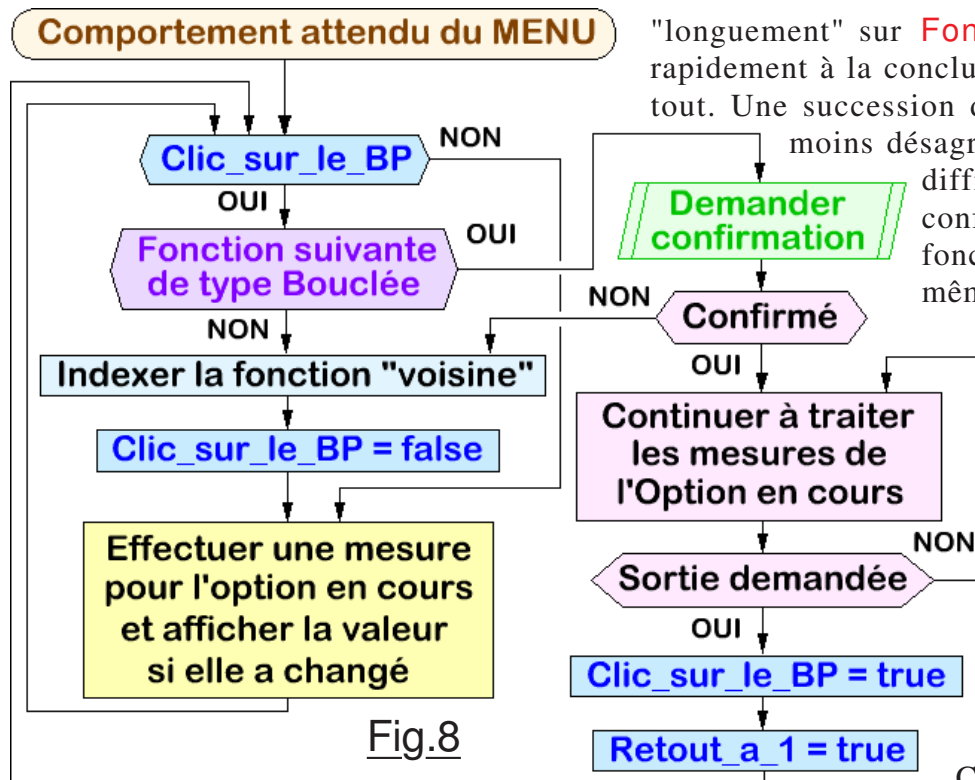
Comme nous ne savons pas à l'avance si l'opérateur désirera surveiller des fronts montants ou descendants, (*Transitions électriques issues du capteur utilisé.*) il faudra prévoir ces deux options dans la fonction de comptage. L'utilisateur pourra aussi désirer remettre à zéro à tout moment. Nous aboutissons pour la fonction de comptage à deux options locales qui ne dépassent pas la combinatoire possible avec la limite de deux B.P. sur l'appareil de mesure. Quand la fonction COMPTEUR sera validée, on a choisi le protocole suivant :

- Action courte sur **Fonction -** : Forcera le résultat à zéro sans pour autant suspendre le comptage. (*Facile à retenir car zéro est généralement en bas dans les graphes, comme pour le B.P. F-*)
- Action longue sur **Fonction -** : Même effet que celui de l'action courte.
- Action courte sur **Fonction +** : Inversera le sens des transitions dénombrées.
- Action Longue sur **Fonction +** : En "standard" sortie de la fonction en cours.

Naturellement d'autres protocoles pourraient être préférés dans cette combinatoire, et éventuellement chaque programmeur pourra personnaliser à sa guise en adaptant le code source.

Problème posé par les fonctions "bouclées".

Informatiquement pas de quoi alerter le ministère des programmeurs : Provoquer la sortie d'une procédure sur examen d'un ou deux booléens n'a pas de quoi faire pâlir de frayeur. Mais c'est le comportement du programme ensuite qui peut rendre l'usage de notre appareil de mesure "indigeste". Envisageons le cas banal d'exploration du menu de base. Chaque action sur l'un des deux B.P. fait passer à l'option voisine et en affiche la nature. Si on chemine dans le menu et que l'on transite par l'une des fonctions bouclées, on entre forcément dans cette dernière qui affiche son titre. Comme mentalement on désire passer à la fonction suivante ou précédente, on va cliquer un court instant sur l'un des deux B.P. et inverser une option locale. Bzzzzz ... agassif car il faut maintenant cliquer



"longuement" sur **Fonction +**. À l'usage on arrive rapidement à la conclusion que ce n'est pas idéal du tout. Une succession d'actions courtes s'avère bien moins désagréable. L'idée pour parer cette difficulté consiste à demander une confirmation avant d'engager une fonction bouclée. Toujours dans le même ordre d'idées, le B.P. du haut sera attribué à l'option "risquée", c'est à dire pour le **OUI** et celui du bas pour le **NON**, autrement dit affecté au passage à la fonction "voisine" dans le MENU principal. Voisine est mise entre guillemets, car le cheminement actuel peut se faire aussi bien en mode prograde qu'en "déplacements" rétrogrades. Comme nous sommes dans une

situation où l'on veut "sauter" l'option, il est logique de continuer dans la progression en cours.

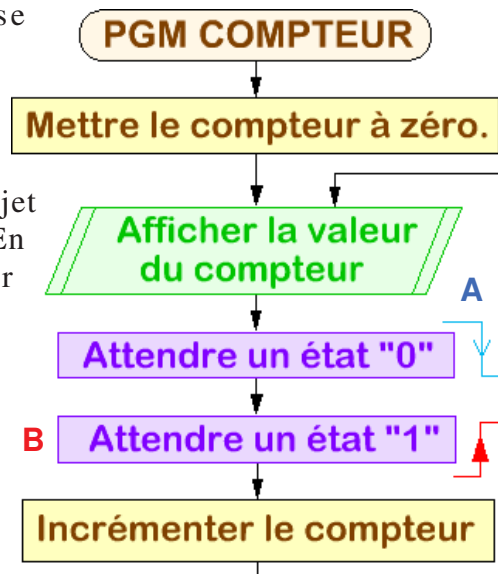
L'organigramme de la Fig.8 montre que finalement gérer informatiquement un MENU avec seulement deux boutons poussoir n'est tout compte faits pas si immédiat que l'on pouvait l'imaginer au départ. C'est le prix à payer pour la qualité opérationnelle. Comme ce type d'architecture peut se rencontrer souvent dans la vie d'un programmeur, autant l'agencer dans un **programme** de type **outils** dont on réutilisera à loisir la structure fonctionnelle. Pour mettre au point le logiciel de la Fig.8 sans s'encombrer de la programmation des options du multimètre, je vous propose de tester **P04_Deux_BP_et_Fonctions_Fermées.ino** qui n'exige de brancher que les deux B.P. et la ligne série USB pour en vérifier le bienfondé. Certains vont peut être estimer qu'avoir consacré plus de quatre pages entière à la gestion de deux boutons est un peu exagéré. Force est de constater que des appareils commercialisés en grande série souffrent sur ce point de faiblesses manifeste, et tout particulièrement dans la gestion de l'anti rebonds quand le temps affecte le fonctionnement de leurs B.P. souvent de qualités médiocres, et qui se dégradent significativement au cours du temps ...

L'approche triviale du comptage.

Riches de deux boutons poussoirs et d'un protocole réputé convivial, nous pouvons maintenant développer une procédure spécifique pour ajouter à notre multimètre une fonction de comptage événementiel. Pour l'amplitude du dénombrement, on va limiter à 9.999.999 la capacité maximale. C'est une valeur qui se justifie par la limite envisagée pour le fréquencemètre, et par les problèmes de formatage de l'affichage qui seront abordés plus avant. Ceci étant précisé, avec une telle capacité il sera rare de se trouver confronté à une insuffisance. Mais avant de passer à du "vrai comptage" notre cheminement va transiter par une petite expérience dans laquelle nous allons procéder de façon archi banale, c'est à dire utiliser une procédure classique avec une boucle "standard" qui surveille une entrée, puis procède à l'incréméntation chaque fois qu'une transition est détectée. Comme le traitement envisagé n'est que très provisoire, seul un comptage sur front montant sera analysé. La technique présentée dans cette approche "simplifiée" est très courante, car elle élude l'apprentissage des INTERRUPTIONS et reste utilisable chaque fois que la fréquence des événements reste "modeste". Par exemple dans notre démonstrateur nommé **P05_Compteur_SIMPLIFIE.ino** la fréquence limite reste inférieure à 1000Hz, principalement ralentie par l'affichage sur la ligne USB. Par ailleurs, le traitement est pratiquement nul, mais dans la boucle réalisée il faudrait aussi tester les boutons poussoir. Bref, cette technique élémentaire ne permet pas des cadences élevées.

Comptage évènementiel avec l'ATmega328 :

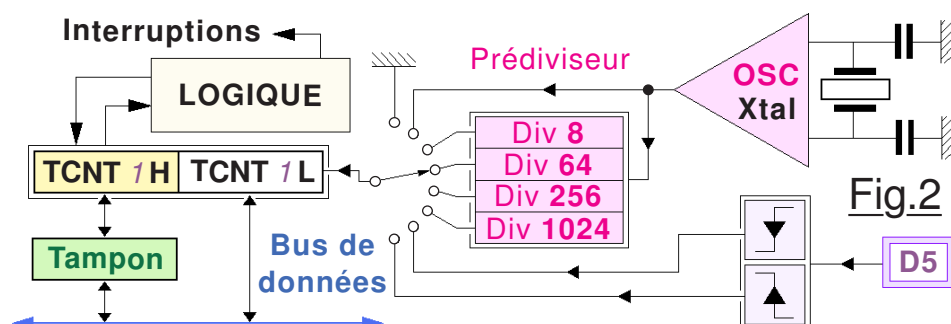
A vant de "sortir l'artillerie lourde", c'est à dire l'utilisation des ressources internes spécifiques au microcontrôleur, et en particulier les INTERRUPTIONS, examinons un minimum le programme de base **P05_Compteur_SIMPLIFIE.ino**. Comme montré sur la Fig.1 l'organigramme de la boucle assurant le comptage est réellement élémentaire. Nous pourrions utiliser n'importe quelle broche configurable en entrée, mais nous avons choisi l'entrée binaire **D5** car c'est elle qui sera utilisée dans le projet final pour mesurer des fréquences. Ce choix n'est pas libre. En effet, comme nous le verrons plus avant, c'est **D5** qui sur l'ATmega328 est reliée au TIMER1. En **A** on prépare l'attente du front montant qui sera détecté en **B** dès que l'entrée surveillée passera de "0" à "1". Tester en permanence une entrée binaire est assez rapide, et incrémenter un entier ne prend pas beaucoup de temps. Par contre, l'affichage de la valeur sur la ligne série va ralentir considérablement la boucle de mesure, surtout si le débit est faible. De toute façon, même si l'affichage était plus rapide, on n'atteindra jamais avec cette technique la cadence d'un compteur électronique. Le "silicium" sera toujours plus performant que le logiciel. C'est la raison pour laquelle pratiquement tous les microprocesseurs disposent de compteurs internes purement électronique pouvant être "lus" sur les bus de l'unité centrale. L'ATmega328 dispose d'au moins quatre compteurs rapides, dont celui déjà mentionné en page 16. (Fonction *millis()*;) Fig.1



LE COMPTEUR TIMER1 de l'ATmega328.

D isposant de trois COMPTEURS/TIMEURS principaux dont les modes d'utilisation sont assez complexes, il n'est pas question dans cet exposé d'en passer en revue toutes les possibilités. Nous allons limiter dans ces pages les explications au plus élémentaire, uniquement pour que vous puissiez comprendre les programmes proposés. **Désolé, mais un minimum de théorie s'impose !**

Globalement, TIMER0 et TIMER1 présentent des morphologies identiques pour la sélection de la source provoquant le comptage. La Fig.2 exagérément simplifiée met en évidence les diverses sources d'incrémentations possibles. Ce sont les **trois bits** de poids faible du registre de contrôle nommé **TCCRnB** qui **imposent l'une des huit options** disponibles. C'est en aiguillant vers le compteur des signaux d'horloge périodiques que ce dernier devient un chronomètre capable de gérer des événements liés au temps. La Fig.2 représente le TIMER1 qui fonctionnant sur 16 bits est en réalité composé de deux registres de huit bits **TCNT1H** et **TCNT1L**. Lorsque le CPU fait un accès à l'emplacement **TCNT1H**, l'unité centrale accède au registre **Tampon** pour l'octet de poids fort. Ce registre temporaire est mis à jour avec la valeur de **TCNT1H** lorsque **TCNT1L** est lue. **TCNT1H** est mis à jour en écriture avec la valeur du registre **Tampon** lorsque **TCNT1L** est écrit. **Le registre de poids forts TCNT1H ne peut donc être accessible directement par l'unité centrale du microcontrôleur.** La **LOGIQUE** permet de remettre à zéro les compteurs, de valider les interruptions sur débordement, de précharger des valeurs initiales etc. La fréquence du signal d'horloge de l'**OSCillateur** interne



piloté par le quartz extérieur **Xtal** (Généralement de 16Mhz) peut être envoyée au compteur via un **Prédiviseur** qui augmente la période des signaux comptés. On dispose ainsi d'un chronomètre qui peut de ce fait fonctionner à des cadences variées. Fig.2

La fonction compteur d'événements.

A l'instar de tous les compteurs 16 bits, TIMER1 recyclera naturellement à zéro quand il aura atteint 65535 et qu'une impulsion de plus lui parviendra. Bien que cette valeur soit inférieure aux 9.999.999 que nous nous sommes fixés pour objectif final, dans un premier temps nous allons nous en contenter. On va juste apprendre à compter, sans se préoccuper du dépassement de capacité de ce double registre. C'est le programme `P06_Compteur_TIMER1_brut.ino` qui va illustrer cette approche initiale. Avant de décortiquer quelques lignes de ce programme, examinons la Fig.3 qui montre le programme. L'organigramme présente un lien de parenté évident avec celui de la Fig.1 mais présente toutefois une différence fondamentale. Le processeur peut passer autant de temps qu'il le désire dans la **routine d'affichage**, car en (1) le TIMER1 indépendant compte inexorablement chaque fois qu'un front montant arrive sur **D5**. De plus, taillé dans le silicium ce compteur peut incrémenter jusqu'à 8MHz d'où l'avantage incontestable d'une logique câblée assistant l'unité centrale.

Tout au plus, entre deux affichages, la valeur aura changé d'autant plus que l'affichage est long et que la fréquence du signal compté est élevée.

TCCR1A (*Identificateur reconnu de l'I.D.E.*) est l'un des deux registres de conditionnement du TIMER1. L'instruction `TCCR1A = 0;` configure TIMER1 en simple compteur. La ligne

`TCCR1B = (1<<CS12) | (1<<CS11) | (1<<CS10);` aboutit à placer **111** dans les trois bits de poids faible du registre TCCR1B (*Identificateur reconnu de l'I.D.E.*) qui est le deuxième registre de configuration du TIMER1. Cette forme d'écriture est particulière à l'environnement d'Arduino, en détailler ici les raisons sortirait du cadre de cet exposé. Cette combinaison binaire **111** impose comme **source**

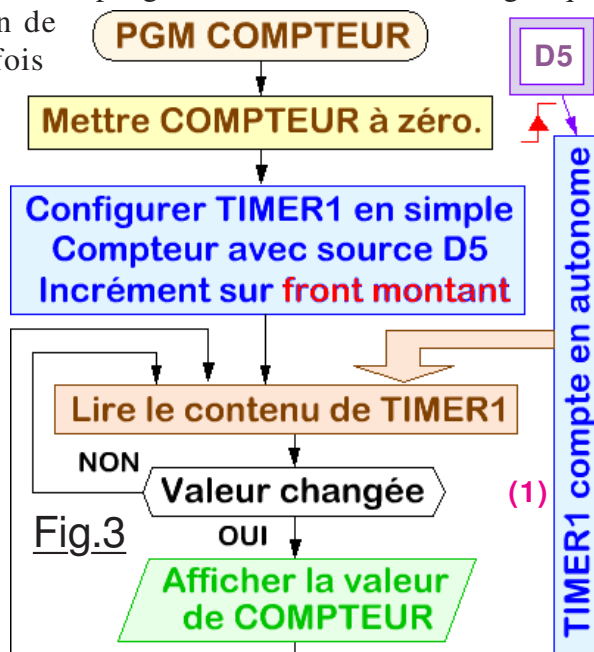
l'entrée D5 et sélectionne le **front montant** pour provoquer l'incrémentation.

```
//===== Traiter le COMPTEUR =====
// Ci-dessous une première lecture fait passer TCNT1H dans le TAMPON.
COMPTEUR = TCNT1L; (2)
// Ci-dessous lecture des 16 bits dans la variable COMPTEUR.
COMPTEUR = TCNT1; (3)
//=====
```

Conformément à ce qui était précisé en bas de la page 19 on ne peut pas lire directement les deux registres du TIMER1 car l'ATmega328 est un microcontrôleur huit bits avec un bus de données à ce format. La première lecture de TCNT1L en (2) fait passer la valeur de TCNT1H dans le registre dédié **Tampon**. Notez qu'il serait possible de coder `COMPTEUR = TCNT1;` qui codifie la lecture des deux registres de TIMER1. La valeur de TCNT1H passerait exactement de la même façon dans le **Tampon**. Maintenant que les poids forts sont disponibles dans le **Tampon**, la lecture du TIMER1 complet en (3) transfère la valeur attendue dans la variable 16 bits **COMPTEUR**.

COMPTEUR est l'identificateur que nous avons donné à notre variable. C'est un nom "personnalisé" qui contrairement à TCCR1A, TCCR1B, TCNT1L ou TCNT1H n'est pas standard dans l'IDE. Dans ce programme nous aurions tout aussi bien placé directement TCNT1 dans l'instruction `Serial.println(COMPTEUR);` mais par la suite, augmenter la capacité à 9.999.999 va obliger à du traitement intermédiaire, et cette donnée devra être mémorisée. (*On prépare le terrain !*)

Dans la procédure de service `void RAZ_COMPTEUR()` on a écrit `TCNT1H = 0; TCNT1L = 0;`. C'est une façon de faire qui respecte l'ordre qui en écriture doit commencer par TCNT1H et finir par TCNT1L, l'usage du **Tampon** étant inversé en lecture et en écriture. Rien n'interdit d'écrire directement `TCNT1 = 0;` car le compilateur procède dans l'ordre correct en écriture et en lecture.



Rompre régulièrement la routine.

Contrairement aux humains qui détestent être interrompus sans arrêt quand ils cherchent à dénombrer de quelconques événements, les microcontrôleurs deviennent particulièrement réactifs quand on vient perturber le confort routinier de la boucle de base ou de ses procédures de servitude. Le petit programme expérimental **P07_Compteur_TIMER1_avec_IRQ.ino** va nous permettre d'illustrer ce propos. Vous avez déjà compris que nous allons nous aventurer dans les sentiers redoutés, à tort, des INTERRUPTIONS. Rassurez-vous, c'est infiniment plus simple que l'on croit trop souvent. Du reste, vous pratiquez des interruptions régulièrement sans y avoir prêté attention. Envisageons par exemple le fonctionnement très routinier de la boucle de base. Dans **void LOOP()** on "tourne" régulièrement d'une instruction à la suivante avec une régularité lancinante. Mais si votre programme est bien écrit, au lieu de comporter 500 instructions élémentaires, notre ronde n'est composée que d'appels à des fonctions de service assurant chacune une tâche bien spécifique. Lors de l'appel, il y a interruption de la boucle de base pour sauter à un traitement "ponctuel". Quand ce dernier est achevé, on revient au train train familier de **void LOOP()**. Dans les propos qui précèdent, on quitte le traitement en cours dans un ordre déterminé à l'avance. Les INTERRUPTIONS, c'est exactement analogue, sauf que l'appel de la routine particulière peut survenir à tout moment. Le traitement en cours est suspendu, la routine d'interruption est traitée, puis on reprend le travail de fond là où il avait été suspendu. C'est aussi simple que ça.

Fonctionnement des INTERRUPTIONS.

Choisir en exemple d'école dans la vie de tous les jours va nous faire comprendre facilement le mécanisme des INTERRUPTIONS. Envisageons une infirmière dévouée, (*Toutes le sont !*) qui travaille de nuit dans la surveillance des patients d'un Hôpital. Dans le calme nocturne, sa boucle de base routinière consiste à passer de chambre en chambre pour vérifier que tout va bien. Discrètement, elle pénètre dans la chambre 38, consulte le moniteur, et si tout va bien elle revient dans le couloir et continue courageusement vers la chambre 39. Tout d'un coup, à l'improviste, une alerte sonore se déclenche. Que fait-elle alors ?

- Elle sort immédiatement dans le couloir, cherche la lampe rouge est allumée au dessus des portes.
- Elle se rend sans perdre de temps à la chambre 12 dont le moniteur a déclenché l'alerte.
- Avec maîtrise et calme elle traite l'incident. Quand c'est fait, elle désactive et réarme l'alerte.
- Puis elle retourne à la chambre 39 pour y terminer ses vérifications avant de poursuivre en 40.

Le fonctionnement en INTERRUPTIONS des microcontrôleurs est strictement analogue. Plusieurs électroniques périphériques à l'U.C. peuvent au souhait du programmeur assister cette dernière. Chacune est l'équivalent de nos chambres d'hôpital. Quand l'une d'elles a besoin d'être prise en compte, elle le signale au processeur qui mémorise l'endroit où il "se trouve" dans le programme. Il y a alors saut à une procédure spécifique qui commence par interdire toute nouvelle interruption si elle est prioritaire. Quand elle s'achève, le "drapeau" qui la concerne est remis à "0", éventuellement les interruptions réactivées, puis le processeur retourne à l'adresse du code qui avait été mémorisée.

TOUT CE PROCESSUS EST AUTOMATIQUE ... OUFFFFFFFFFFFFFFFFFFF !

Concrètement, chaque module électronique interne au processeur pouvant déclencher une interruption sera organisé pour que l'on puisse facilement travailler (*Programmer !*) comme le fait l'infirmière mentionnée ci-avant. L'ATmega328 possède plusieurs sources internes pouvant déclencher des interruptions particulières. Pour chacune de ces sources on dispose :

- D'un "vecteur" dans lequel on indiquera quelle est la procédure de traitement associée,
 - D'un "drapeau" qui indique à l'U.C. qui a déclenché l'interruption,
 - D'un masque d'interruption qui permet d'activer les déclenchements ou de bloquer le demandeur.
- Pour compter, mesurer des fréquences, des périodes etc, on va utiliser les interruptions.

EN QUOI UNE INTERRUPTION EST-ELLE PLUS DYNAMIQUE qu'un programme routinier ?

Il suffit de reprendre l'exemple d'école déjà envisagé. Dans la chambre 12, le patient est victime d'un arrêt cardiaque. S'il n'y avait pas les alertes, l'infirmière continuerait professionnellement à passer les 28 autres chambres en revue avant de revenir à la 12 ... trop tard une fois de plus !

L'immense avantage des interruptions, c'est que des phénomènes transitoires parfois de très courte

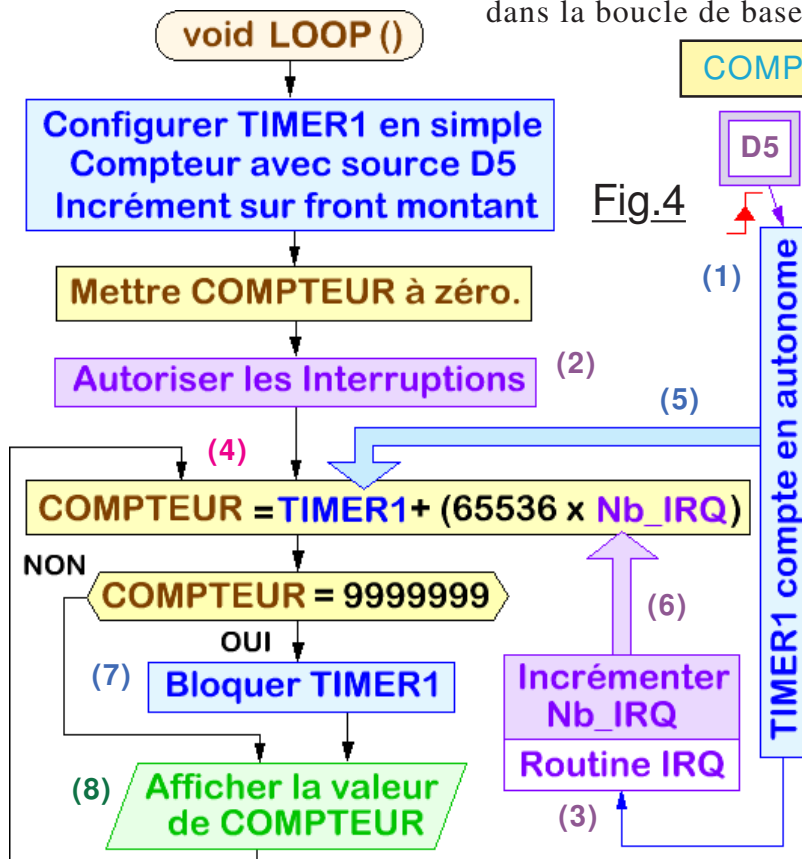
durée sont immédiatement pris en compte et ne passent pas inaperçus. Interrompre la routine de base qui ne s'occupe que d'éléments non urgents n'est absolument pas pénalisant. Nous avons tous les éléments en main pour réaliser un vrai compteur rapide. En sois, il est assez rare de désirer compter des phénomènes à cadence rapide. Mais plus avant, on va concevoir un fréquencemètre. Ce n'est pas autre chose qu'un compteur que l'on valide durant une seconde, mais on désire alors pouvoir mesurer à des fréquences de plusieurs MHz ... et il n'y aura pas de temps à perdre !

Compteur rapide avec TIMER1.

Avec le programme simplifié **P06_Compteur_TIMER1_brut.ino** nous nous sommes contentés d'utiliser TIMER1 sans prendre en compte son débordement, c'est à dire le recyclage de la valeur maximale (*Sur 16 bits.*) de 65535 à 0. S'il s'agissait de compter les automobiles qui passent sur une route ou les vaches qui rentrent dans une étable, cette capacité couvrirait largement nos besoins et l'on pourrait en rester là. Mais lorsque nous allons chercher à concevoir un fréquencemètre, le limiter à 65KHz serait bien tristounet. Limitation d'autant plus "déraisonnable" que théoriquement TIMER1 peut mesurer jusqu'à 8MHz. Nous allons donc nous imposer un compteur capable de couvrir cette plage. Pour des raisons diverses, la limite sera fixée à 9.999.999 comme déjà annoncé dans les pages précédentes. C'est **P07_Compteur_TIMER1_avec_IRQ.ino** qui va nous servir de banc d'essai, mais avant, un précepte fondamental doit contraindre en permanence notre enthousiasme :

Idéalement une routine d'interruption doit être la plus courte possible pour libérer les "urgences" sans perdre de temps. Elle doit se contenter du traitement minimal, laissant à la boucle de base le loisir d'effectuer les opérations "laborieuses" en toute sérénité.

Pour cerner le bienfondé de ce principe qu'il ne faudra jamais oublier, envisageons un comptage à cadence maximale, soit à 8MHz. À cette vitesse, les 65536 comptages sont effectués en moins de 9 mS. Si à chaque interruption on lambine et l'on prend le temps d'effectuer un long traitement, on risque de laisser passer un recyclage et la valeur mesurée sera erronée. Hors, pour avoir un résultat correct, il suffit de savoir avec certitude combien de dépassements ont été recensés. Le programme respectant ces préambules est résumé sur la Fig.4 qui met en évidence le traitement "dérisoire" (*Mais impératif.*) réalisé dans la procédure d'interruption, et le travail assuré ensuite dans la boucle de base. Vous avez déjà compris que :



Formule dans laquelle l'identificateur **Nb_IRQ** représente le "Nombre d'Interrupt ReQuest".

Détaillons l'architecture et le fonctionnement du programme :

En (1) le TIMER1 étant conditionné en simple compteur fonctionne en autonome et s'incrémentera chaque fois que sur D5 une transition montante sera détectée. Comme en (2) on a validé des déclenchements d'interruptions chaque fois qu'il passe de 65535 à 0, il y aura des branchements à la Routine IRQ à sa demande. Dans notre logiciel, la routine d'interruption spécifique à TIMER1 ne fait qu'incrémenter en (3) la variable Nb_IRQ ce qui ne monopolise l'U.C. que durant quelques μS .

En (4) la boucle de base procède

"sereinement" et sans se presser aux divers calculs. Elle commence par récupérer en (5) la valeur actuelle de **TIMER1** qu'elle place dans **COMPTEUR**. Puis en (6) elle lit la valeur de la variable globale **Nb_IRQ** qu'elle multiplie par 65536, résultat qu'elle ajoute à **COMPTEUR**. En (7), si **COMPTEUR** est arrivé à la valeur maximale autorisée, **TIMER1** est bloqué et s'arrête définitivement de compter. Notez qu'en (8) la valeur de **COMPTEUR** n'est affichée que si elle a changé pour éviter un défilement vertical permanent des informations sur le moniteur vidéo. Dans tout ce mécanisme, il importe de remarquer que le fonctionnement de **TIMER1** est totalement indépendant du déroulement de la boucle de base **void LOOP()**. Par ailleurs, les interruptions qu'il déclenche en (3) ne font que dérouter le programme dans **Routine IRQ** durant à peine quelques μS . Ces détournements arrivent à n'importe quel "endroit" de **void LOOP()**. C'est le principe de base des mécanismes d'INTERRUPTION. Le reste n'est plus qu'une question de codage avec des écritures et des conventions propres à chaque compilateur. En langage C adapté à l'environnement d'Arduino nous disposons de divers protocoles et identificateurs réservés à cette fin. Passons en revue les lignes de programmes relatives à cette facette de la programmation de l'ATmega328 dans **P07_Compteur_TIMER1_avec_IRQ.ino** servant à défricher ce chapitre de la "programmation" :

- Vous avez reconnu l'instruction :

TCCR1A = 0; qui configure **TIMER1** en simple compteur et la ligne :

TCCR1B = (1<<CS12) | (1<<CS11) | (1<<CS10); qui impose comme **source l'entrée D5** et sélectionne le **front montant** provoquant l'incréméntation.

- Ce qui est nouveau, c'est la déclaration **ISR(TIMER1_OVF_vect) {Nb_IRQ++;}** **placée en tête de programme**, avant **void SETUP()** qui définit la routine d'interruption du **TIMER1** sur débordement. Dans cette forme d'écriture, **ISR** informe le compilateur que le vecteur (*Adresse d'interruption.*) relatif à **TIMER1_OVF** devra pointer sur **la routine dédiée placée entre accolades**. (*Vector for Interrupt Service Routine on OVerFlow TIMER1*)

Cette écriture n'est finalement pas compliquée du tout. On se contente de prévenir que la ligne de code concerne **les interruptions**, puis **entre parenthèses** on précise quel **module fonctionnel** du microcontrôleur est **concerné**, puis **entre accolades** on place toutes **les instructions** qui seront déroulées lors de cette interruption. Avant de poursuivre, ouvrons une petite parenthèse : (*Vocabulaire*)

LES DRAPEAUX en informatique.

Bien que très connu dans l'univers de la programmation, je ne crois pas déraisonnable d'en préciser la signification à l'attention des débutants. Historiquement, les drapeaux sont utilisés depuis toujours pour donner des informations. Le drapeau blanc par exemple, les fanions dans la marine signalant une maladie à bord, le drapeau national symbolisant une identité etc.

En informatique, on nomme **drapeau un BIT spécifique** qui mis à "0" ou à "1" précisera au microcontrôleur une information particulière. Par exemple le bit de signe dans un nombre peut être considéré comme un drapeau. Des REGISTRES particuliers (*De 8 BIT pour l'ATmega328.*) sont réservés à des regroupements de drapeaux. Par exemple, en ce qui nous concerne le REGISTRE désigné par **TIMSK1** dans l'IDE désigne un octet particulier dans lequel chaque BIT est "attaché" à une interruption. **TOIE1** est le drapeau relatif aux interruptions déclenchées par **TIMER1** sur débordement. Si ce drapeau est mis à "1", **TIMER1** pourra Interrompre. S'il est mis à "0" ses requêtes seront ignorées. Chaque "électronique spécifique" pouvant générer des interruptions dispose de son drapeau individuel dans **TIMSK1**. Des mots tels que **TIMSK1** et **TOIE1** sont des mots réservés pour l'I.D.E. d'Arduino.

- Autre particularité : La ligne **TIMSK1 |= (1<<TOIE1);** qui sous une forme d'écriture propre à l'I.D.E. place à "1" le drapeau **TOIE1** dans le registre **TIMSK1**. Les explications développées dans l'encadré doivent vous permettre d'en déduire que **TIMER1** pourra interrompre sur débordement.

Au final, quand on analyse le programme expérimental, mis à part des écritures un peu particulières comme **TIMSK1 |= (1<<TOIE1);** qui utilisent du décalage logique avec du OU dont il suffit de comprendre la structure, utiliser un **TIMER** en interruption sur débordement est en conclusion très simple. La structure de la séquence globale se résume à :

- Préciser au compilateur le code de la routine d'interruption et la fonction électronique concernée,
- Configurer le TIMER/COMPTEUR en fonction de nos choix, (*Compteur, chronomètre etc.*)
- Mettre à zéro le TIMER et les variables associées dans notre programme,
- Valider le drapeau du TIMER dans le REGISTRE des INTERRUPTIONS,
- Utiliser les valeurs des variables associées dans la boucle de base ou ses servitudes,
- Éventuellement arrêter le comptage ou suspendre les interruptions si le logiciel l'exige.

Notez que dans **P07_Compteur_TIMER1_avec_IRQ.ino** c'est la procédure **RAZ_COMPTEUR()**; qui se charge de mettre à zéro les variables du programme et de configurer TIMER1. Dans cet exemple expérimental ce n'est pas très rentable, car l'appel à ce code est unique, mais on anticipe ici le développement d'un programme plus global qui devra effectuer cette opération plusieurs fois. Remarquez également que pour agrémenter l'affichage, des "zéros en tête" sont ajoutés pour rendre la visualisation plus agréable à l'écran, stabilisant sur les lignes affichées la position latérale des nombres qui défilent à la rapidité du comptage ou du rafraichissement de la boucle de base.

Certains vont avoir l'impression que les trois pages qui précèdent présentent par moment quelques aspects "indigestes", car encombré de beaucoup de texte. C'est probablement assez justifié, vu que l'on a foncé "bille en tête" dans le domaine des Interruptions qui généralement se trouve plutôt vers la fin des livres de vulgarisation. **Ne vous prenez pas la tête, si tous ce charabias vous a un peu gavé, oubliez ce sentier, on peut s'en passer. Contentez-vous de tous les autres chapitres où nous n'abordons que des "fondamentaux de base".** Si certaines lignes de code ne sont pas des évidences, aucune raison d'en faire un complexe. Appropriiez-vous ce qui vous plait et ... oubliez le reste. Il vous sera toujours possible d'y revenir quand l'envie s'en fera sentir. Maintenant que la notion de compteur est globalement assimilée, tout au moins dans les grandes lignes, on va pouvoir explorer des domaines très intéressants dans le monde du mesurage.

Pour achever ce chapitre sur le comptage rapide, vous pouvez si vous en avez l'envie, charger le petit démonstrateur **P08_Compteur_USB_par_TIMER1.ino** qui propose une version "à deux B.P." en vue de son intégration dans le logiciel définitif. Le bouton poussoir **FC +** en utilisation courte inverse le sens de la transition qui provoque le comptage, l'affichage précisant en début de chaque ligne l'option en cours. Le bouton **FC +** en utilisation longue fait sortir de la fonction compteur et retourne(ra) au menu de base. Enfin **FC -** provoque une remise à zéro du compteur sans interrompre ses incrémentations. Pour tester la fonction COMPTEUR de notre mini laboratoire, il suffit d'injecter sur **D5** un signal périodique quelconque. S'il fait 1MHz, le septième chiffre affiché changera environ toutes les secondes. À 1000 Hz c'est le quatrième chiffre qui s'incrémentera à chaque seconde. À 4MHz le compteur sera saturé en moins de trois secondes et à 8MHz ... devinez !

L'artillerie lourde !

Entre survoler de très loin en "cachant tout" et détailler la moindre molécule informatique constituant nos programmes, il a fallu accepter des compromis pas forcément facile à choisir. Dans un document tel que celui-ci, il n'est pas possible d'expliciter chaque ligne de programme. Parfois un encadré tel que celui de la page 22 ou celui de la page 23 peuvent nous aider à clarifier certains points incontournables. Mais multiplier ce genre de parenthèses alourdirait inexorablement notre cheminement réputé "amusement de loisir". D'un autre côté, quand on affirme que des instructions telles que $TCCR1B = (1 \ll CS12) | (1 \ll CS11) | (1 \ll CS10)$; ne sont que des écritures particulières pour imposer une **source l'entrée D5** et sélectionner le **front montant** provoquant l'incrément, (*Voir page 20 par exemples.*) certains aimeraient bien disposer d'informations supplémentaires sans avoir à chercher des développements ou des résumés sur Internet. C'est la raison pour laquelle le petit fichier **Fiches Arduino.pdf** est mis à votre disposition. Vous y trouverez en vrac un certain nombre de petites fiches gavées à saturation d'informations "disparates" mais plus ou moins directement liées au contenu des "bavardages" qui encombrant **Fiches Arduino.pdf**. À votre guise vous pouvez ignorer sans vergogne ces fiches, mais aussi en faire une consommation sans modération.

Hé bé, elle est pas marrante cette page. Que du texte, pas d'image. C'est vraiment pas rigolo l'informatique !!!

Après le long chapitre, juste pour pouvoir s'octroyer une petite récréation et d'oublier interruptions. Dans cette page nous allons aborder la "gestion du temps qui s'écoule". (*Expression qui confine à de la débilité profonde ... comme s'il était possible d'influencer le temps qui inexorablement égraine ses éléments dans la relativité $E = MC^2$!*) Depuis toujours, l'humain a été dans l'obligation de calibrer ses activités, que ce soit pour synchroniser une vie sociale ou techniquement servir des impératifs temporels de plus en plus exigeants en précision. Du simple sablier jusqu'à l'horloge atomique, notre vie de tous les jours est intimement cadencée par ces machines à

temps qui passe. Cette page va munir notre laboratoire d'un "sablier" à quartz dont l'imprécision sera essentiellement due au manque de rapidité de l'opérateur. Dans cette application, c'est le B.P. **Fonction +** qui servira à déclencher ou suspendre le chronométrage. Mais rien n'interdit de se bricoler une poignée ergonomique comme celle de la Fig.1 qui permet à un opérateur averti d'espérer

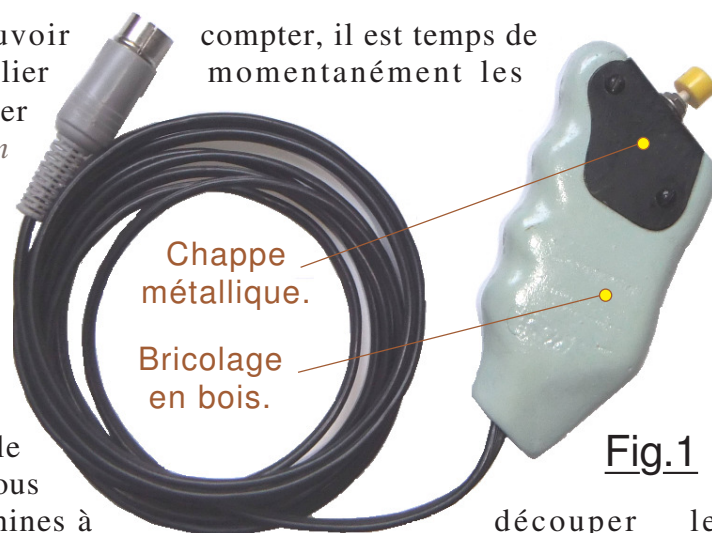


Fig.1

| B.P. | Action en chronomètre. |
|----------------------|--|
| Fonc + longue | Sortie en standard de la fonction bouclée. |
| Fonc + courte | Marche / Arrêt. |
| Fonc - longue | Rapide / Lent. |
| Fonc - courte | Remise à zéro sans modifier MA / AR. |

Outre la possibilité de pouvoir le déclencher ou le stopper à convenance, un chronomètre n'est vraiment opérationnel que si de plus on peut à tout moment le remettre à zéro sans en changer le mode d'activité. Comme les deux boutons de configuration de notre mini laboratoire permettent trois options, on a prévu de toutes les utiliser. Le tableau donné ci-dessus précise les options retenues. En mode **Rapide** le chronomètre affiche les dixièmes de seconde, la visualisation est donc dix fois plus prompte qu'en mode réputé **Lent**. Le programme **P09_CHRONOMETRE_sur_USB.ino** qui permet de mettre au point le CHRONOMETRE est en réalité d'une simplicité déconcertante et ne fait appel à rien de neuf. Pour gérer le temps nous écarterons les interruptions et l'utilisation de l'un des TIMERS. Nous avons déjà abordé en page 16 la fonction **millis()** qui effectue un **travail de fond masqué**. Nous savons que c'est un compteur qui est incrémenté inexorablement 1000 fois par seconde. Pour gérer le temps de façon simple il suffit d'en interroger la valeur régulièrement. En mode **Rapide** on attend que le compteur ait évolué de 100 unités, soit 0,1 seconde écoulée avec une précision de 1/1000^{ième} de seconde. En mode **Lent** le moniteur vidéo ne sera rafraîchi qu'une fois par seconde et les 1/10^{ième} de secondes seront ignorés à l'affichage. C'est la procédure de servitude **void Incrémenter_le_CHRONOMETRE()** qui effectue tout le travail de chronométrage. À chaque intervalle de temps programmé écoulé, elle procède à l'incrémentation du COMPTEUR HEXAGÉSIMAL constitué des quatre variables **HEURES**, **MINUTES**, **SECONDES** et **DIXIEMES**.

Comme vous pouvez le constater, un tel compteur logiciel est particulièrement facile à mettre en œuvre et se passe de tout commentaire. Dans cette version du programme, nous avons estimé qu'une capacité de 24 heures serait suffisante, mais rien ne vous empêche d'ajouter des variables du genre **JOURS**, **ANNEES** à votre convenance et en faire pourquoi pas une horloge séculaire ? Vous pouvez observer au passage que **millis()** est un "électron libre imperturbable". (*Sauf sur un RESET bien entendu !*) De ce fait, on peut en lire le contenu à tout moment et de façon totalement aléatoire. (*J'ai failli écrire "intemporelle" !*) Du coup, nous pouvons réaliser autant de chronomètres logiciels que nous le désirons. Du reste, pendant que la fonction CHRONOMETRE "tourne", chaque appui sur un B.P. déclenche un deuxième chronométrage en parallèle pour savoir si c'est un clic long ou un clic court. Avouez que ce compteur interne au microcontrôleur est d'un usage bien commode, surtout avec la fonction dédiée **millis()** intégrée dans l'I.D.E. d'Arduino.

compter, il est temps de momentanément les

découper le

approcher le dixième de seconde en chronométrage manuel. Sur notre réalisation pratique, le cordon étant long nous avons utilisé du câble B.F. blindé pour mettre à couvert la ligne **A0** des parasites environnants.

La mesure des Fréquences avec l'ATmega328 :

Quand nous savons comment réaliser un compteur logiciel très rapide et à forte capacité, mesurer une fréquence ou une période deviennent élémentaires. Par définition, la fréquence d'un événement quelconque, et par voie de conséquences celle d'un signal électrique périodique, est le nombre de fois par seconde que ce produit le phénomène étudié. Quand à la période, c'est le temps qu'il faut entre deux reours aux "conditions initiales" créant un comptages.

Fréquence = NB événements / seconde.

Période = Durée d'un événement = 1(seconde) / Fréquence.

Matériellement, c'est encore l'entrée **D5** qui sera utilisée pour recevoir le signal électrique à analyser. Cette nouvelle fonction du mini laboratoire est donc totalement gratuite puisqu'elle ne fait appel à aucun composant nouveau. Elle ne consommera que de l'espace dans la mémoire de programme, mais nous en avons à revendre. Du reste, quand l'intégralité du source sera intégrée en un seul logiciel avec toutes les fonctions et tous les perfectionnements possibles, il restera encore pratiquement la moitié de la place encore disponible. Ce n'est donc pas la peine de chercher à faire des économies "bout de chandelle".

REMARQUE : Le fréquencemètre est la seule fonction qui était disponible sur le petit KIT visible sur la Fig.2 en page 2. Et encore, avec un affichage tristounet sans les points décimaux et ne "comptant" qu'une fois par seconde. Autant dire que le microcontrôleur n'y est pas vraiment rentabilisé. C'est bien dommage, car matériellement ce KIT est un petit bijou très bien agencé.

Comme il en a été question pour le CHRONOMÈTRE, nous allons pour le FREQUENCEMÈTRE exploiter au mieux les trois options possibles de la fonction bouclée. Pour conserver un maximum de possibilités, PÉRIODE et FREQUENCE seront affichées "simultanément" puisque l'une se déduit de l'autre. Par contre, la recherche de convivialité de comportement a aboutit à une combinatoire qui s'écarte un peu de la philosophie qui se retrouve pour les autres fonctions bouclées. Pour comprendre cette petite entorse à "la routine", envisageons avant le "cahier des charges" (*Ouaououou, ça fait sérieux ça !*) de la FONCTION FREQUENCEMÈTRE / PÉRIODEMÈTRE :

- Affichage de la fréquence et de la période simultanément. (*Sur une même ligne.*)
- Précision d'au moins $\pm 0.000001\%$ pour les fréquences $> 1\text{MHz}$. (*Cinq chiffres significatifs.*)
- Faculté de choisir entre une mesure par seconde (*Précision maximale.*) et dix mesures par seconde. Dans le mode **Rapide** la précision n'est que de $\pm 0.00001\%$ mais le rafraichissement de l'écran pratiquement instantané. Cette possibilité est particulièrement agréable quand on désire ajuster un générateur, la durée de fonctionnement d'un monostable etc. Une fois l'ajustement dégrossi on peut alors affiner le réglage en cours avec le mode **Lent**.
- Possibilité de mettre en PAUSE et mémoriser la valeur actuelle mesurée. (*Option HOLD.*)

REMARQUE : Encore une précision de vocabulaire. Désolé d'alourdir cette page avec un encadré de plus, mais créer une fiche supplémentaire pour ça ne serait pas très optimisé.

Dans ce document, j'utilise le mot FONCTION et le mot PROCÉDURE. Je crois indispensable de clarifier un peu leur portée car j'en fais ici un usage pas totalement "standard".

Quand le mot FONCTION concerne le mini laboratoire, la terminologie utilisée est "libre", il définit dans ce cas l'une des diverses possibilités de mesurage ou de génération de signaux disponibles. À l'intérieur d'une fonction "bouclée", les possibilités sont alors nommées OPTIONS.

Quand il s'agit de commentaires relatifs aux programmes, je fais une petite nuance personnelle.

En langage C, tous les sous-programmes sont nommés FONCTIONS et dénoncés par le code source `void NOM_de_la_fonction()`. Personnellement, j'étais programmeur en langage PASCAL avant de venir papillonner en langage C. En PASCAL, si un sous-programme ne fait que "dérouler du code", il est nommé PROCÉDURE. Ce n'est que s'il RETOURNE UNE VALEUR TYPÉE qu'il est alors désigné par FONCTION. Étant très attaché à cette nuance, dans mes propos j'utilise ici l'une ou l'autre des écritures, car je crois important dans le texte de faire la différence. Je vous remercie bien vivement d'accepter avec bienveillance ce coté un peu "rigide" de ma part.

Protocole d'usage du FREQUENCEMÈTRE / PÉRIODEMÈTRE.

Imaginons notre mini laboratoire en action. L'entrée principale reçoit le signal à analyser, que ce soit un créneau de type TTL ou du sinusoïdal ; peu importe. À l'activation de la fonction, par défaut les mesures se font en mode **Lent**. Si l'on veut respecter la précision de six chiffres significatifs, il faut impérativement que la procédure `void Mesurer_la_frequence()` respecte rigoureusement le délai de 1S ou de 0,1S. Cet impératif **interdit de tester les boutons poussoir**

| B.P. | Action en fréquencemètre. |
|----------------------|--|
| Fonc + longue | Sortie en standard de la fonction bouclée. |
| Fonc + courte | Ignorée. |
| Fonc - longue | Si Rapide fait passer en PAUSE. (HOLD) |
| Fonc - longue | Si Lent fait passer en mode Rapide. |
| Fonc - courte | Si Rapide fait passer en mode Lent. |

dans ce sous-programme. Du coup, en mode **Lent** il faudra appuyer au moins durant une seconde ou légèrement plus sur le B.P. pour qu'il soit pris en compte. Comme **Fonction + long** fait sortir, seul **Fonction -** sera possible. Du coup,

il faut pouvoir sélectionner **Lent / Rapide** et la **PAUSE** avec le même bouton ! La seule façon d'y parvenir réside dans l'utilisation particulière qui est faite des B.P. et résumée dans le tableau ci-dessus. Le B.P. **Fonction + court** n'est pas exploité ce qui ressemble à du gaspillage. Il aurait été possible de s'en servir pour inverser la nature du front qui déclenche les comptages par exemple. Mais dans le cas d'un FRÉQUENCEMÈTRE ou d'un PÉRIODEMÈTRE c'est strictement sans utilité, alors autant simplifier au maximum les protocoles adoptés pour ce type de mesurage.

ATTENTION : Nous savons que tous les systèmes d'affichages numériques, par principe même de conception fluctuent toujours à ± 1 sur le chiffre des unités. Quand on indique **Précision d'au moins $\pm 0.000001\%$** dans le cahier des charges, on précise en fait qu'en dessous de 1MHz cette précision relative ne sera plus possible. C'est une évidence dans la mesure où l'affichage se fait à l'Hertz près, donc en dessous de 1MHz le nombre de chiffres affichés devient forcément plus faible. De plus, c'est la fréquence dont on donne la précision, pas la Période. (Voir plus avant ...)

L'organigramme de la Fig.1 présente l'architecture du programme de démonstration `P10_Frequencemetre_sur_USB.ino` dont la boucle principale ne fait pas grand chose si ce n'est de tester une activation d'un bouton poussoir, de son traitement éventuel puis elle passe la main à `void Mesurer_la_frequence()` qui effectue tout le travail. C'est dans cette procédure que l'on boucle durant 1/10^{ième} ou une seconde sans pouvoir tester les B.P. Ensuite, en sortie de cette

procédure, il y a transmission des valeurs vers la ligne série USB si elles ont changé. Quand on prétend que c'est `void Mesurer_la_frequence()` qui fait tout, c'est largement exagéré. Dans la pratique c'est **TIMER1** et sa routine d'interruption qui cavale à toute vitesse pour compter les transitions à une cadence infernale sur l'entrée **D5** de notre appareil de mesures. Concrètement `void Mesurer_la_frequence()` "se tourne royalement les pouces". Elle ne fait qu'attendre une seconde ou dix fois moins, par contre avec une grande précision puisque cette durée est ajustée à la microseconde près. C'est de cette temporisation rigoureuse que le mini laboratoire est redevable de la précision annoncée.

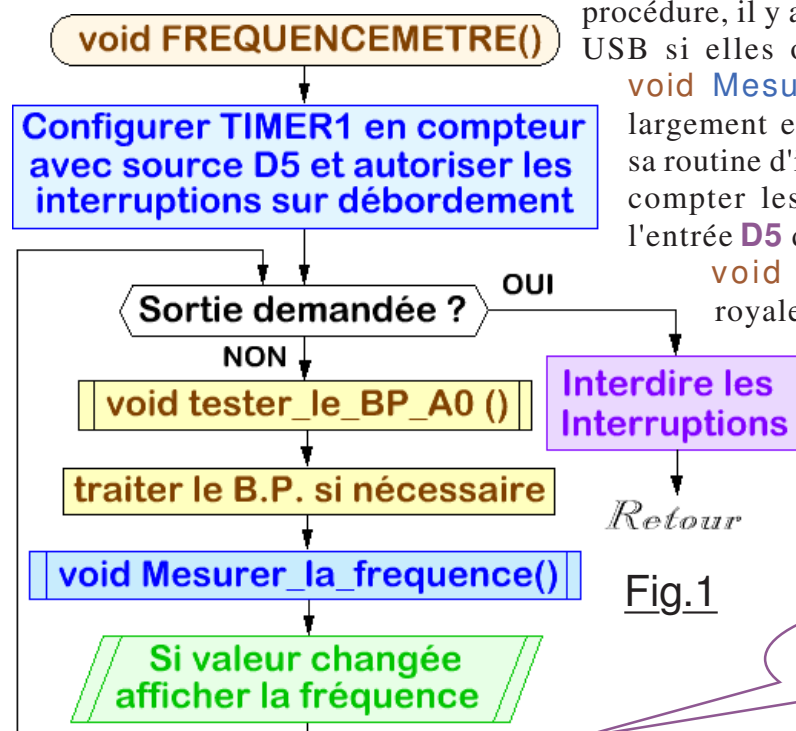


Fig.1

Hé... il exagère l'auteur, il y a déjà deux encadrés dans cette page, et encore, il en a planqué un en page 26 !

Affichage formaté pour de grands nombres.

Quand une mesure commence à donner des valeurs importantes telles que 1732651 qui de plus fluctuent sur quelques chiffres de poids faibles, on a bien du mal à repérer où se situent les "Kilo" et les "Méga". En revanche, si on formatait la valeur affichée sous l'aspect 1.732.651, on serait convaincus immédiatement de l'amélioration de la lisibilité. Ce constat est évidemment le même que les informations soient listées sur l'écran vidéo ou sur un module LCD. Si vous consultez la fiche intitulée [Affichage numérique formaté avec des points](#), il y est précisé, qu'une immobilité latérale pour ce type d'information est très importante au point de vue de l'aisance de lecture. Force est de constater que souvent les programmes que l'on rencontre sur la toile, ou certains produits du commerce ne prennent pas en compte cet aspect relativement important pour la qualité opérationnelle. Citons par exemple le cas du petit fréquencemètre représenté en Fig.2 de la page 2 pourtant un produit commercial dont la qualité matérielle est soignée. C'est peut être parce que développer un traitement qui réalise un tel formatage est bien plus indigeste qu'il n'y paraît avant analyse. J'avoue que pour ma part, de très nombreux essais infructueux ont jalonné le sentier qui m'a fait aboutir à du code fonctionnant correctement. Peu importe la difficulté, le temps passé, les routines qui assurent cette mise en forme des données font l'objet maintenant d'un "OUTIL" pouvant être inséré dans tout programme après une adaptation généralement facile.

Pour vous rendre compte de l'amélioration considérable apportée par ce type d'affichage amélioré, il vous suffit de charger [P11_Frequencemetre_sur_USB_avec_formatage.ino](#) et de soumettre à l'entrée D5 des signaux de fréquence relativement élevée et si possible instables sur deux ou trois chiffres significatifs. Utilisez ensuite un quelconque générateur et faites varier "rapidement" la fréquence de ses signaux entre quelques dizaines d'Hertz et plusieurs Mégahertz. On constate immédiatement que compléter les affichages par des zéros en tête pour maintenir stable la position latérale des nombres affichés est un plus incontestable. Une expérience aussi simple sera largement suffisante pour justifier amplement l'augmentation de taille du programme engendrée par le formatage dans la présentation des données. Mais comme de toute façon dans notre projet la place ne manque pas, une telle amélioration des visualisation s'impose sans discussion.

Accentués et gaspillage de place en mémoire.

L'avènement des "textos" et des forums sur Internet incite à aller au plus rapide, quitte à bafouer ici et là les règles élémentaires de la syntaxe. C'est dans l'ordre des choses et il faut faire avec. Mais une écriture sans mettre les accents où ils seraient les bienvenus est visuellement choquante pour les amoureux de la grammaire, ou tout simplement les "anciens" pour qui les accentués sont aussi importants que la présence d'un point final en fin de ligne ou l'utilisation d'une lettre majuscule à son début. J'appartiens à cette catégorie, aussi, visualiser des textes non accentués m'est frustrant au possible. Malheureusement l'I.D.E. d'Arduino "prend le contrepied". Les accentués sont interdits dans les noms de programme, dans les identificateurs de variable.

Bref, plus d'accentués = Norme. *SI si si, plein d'accentués ... PAPY fait de la résistance !*

La ligne série USB dans l'environnement d'Arduino est allergique aux accentués mais on peut aisément contourner cette difficulté. Il suffit comme vous le rencontrez dans les divers programmes mis à votre disposition, d'utiliser le type `char` suivi d'une valeur ASCII. Une fiche, du côté intitulé [TABRE DES CARACTÈRES ASCII affichables sur LCD](#), nous donne les principaux codes utilisables. Facile non ?

45 octets de gaspillés !

ATTENTION : Le perfectionnisme a un coût ! Pour chaque accentué inséré dans le programme, la taille de ce dernier augmente entre 22 et 23 octets car il y a plusieurs appels à procédures. Chaque appel ajoute de code ainsi que chaque retour. Il n'est donc raisonnable de soigner de cette façon les affichages que si l'on dispose de beaucoup de place, ce qui suppose un programme qui soit loin de saturer les 32256 octets disponibles. Chic chic chic : Une raison de plus pour vivre avec son temps ...

Hé bé, encore une page tristounette sans dessin et avec que des trucs pas rigolo du tout ! C'est un essecandale !

La mesure des Périodes avec l'ATmega328 :

Glups, mais on prend les mêmes et on recommence ma parole ! En réalité, nous allons retourner la "chaussette" informatique ! lorsque nous avons expérimenté le programme de démonstration, `P11_Frequencemetre_sur_USB_avec_formatage.ino` par exemple, une alerte nous invitait à passer en périodemètre dès que la fréquence descendait en dessous de 1000 Hz. On est en droit de se demander pourquoi le programme insiste aussi lourdement. La réponse réside dans un souci maniaque de vouloir à tout prix la plus grande précision possible. Analysons le problème :

Fréquence / Période calculée :

| | | |
|------------|---------|--|
| 10.000Hz / | 100µS | > La fréquence présente plus de chiffres significatifs que la période. |
| 1.000Hz / | 1000µS | > Fréquence et période de précisions similaires. (Quatre chiffres.) |
| 100Hz / | 10000µS | > La période présente plus de chiffres significatifs que la fréquence. |

CONCLUSION : Si l'on recherche la précision maximale dans le mesurage, la fréquence de 1000 Hz constitue un seuil de décision :

- Fréquence > 1000Hz privilégier la mesure de Fréquence et en déduire la valeur de la Période.
- Fréquence < 1000Hz privilégier la mesure de Période et en déduire la valeur de la Fréquence.

Mesure de Période avec affichage formaté.

Persistant à employer le compteur rapide avec la prise en compte des débordements, nous allons "inverser" le processus. Autrement dit, **on va effectuer la mesure de la Période et l'on en déduira par calcul la valeur de la fréquence**. Globalement la structure du programme `P12_Periodometre_sur_USB_avec_formatage.ino` s'avère pratiquement identique à celle de `P11_Frequencemetre_sur_USB_avec_formatage.ino`, autant la "copier", puis en adapter quelques séquences. On va développer ce démonstrateur directement pourvu d'un affichage formaté. C'est d'autant plus raisonnable que pour le logiciel définitif la procédure de présentation des mesures ne sera écrite qu'une seule fois et utilisée à profusion. C'est ... du recyclable !

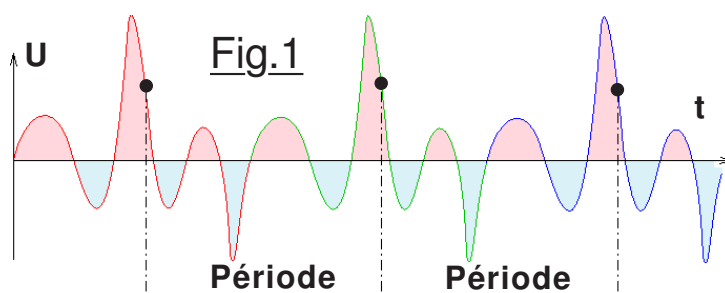
Période = Durée d'un événement = Temps pour partir d'une condition et y revenir.

Fréquence = 1.000.000 / Période. (1.000.000 car la période est désirée en µS)

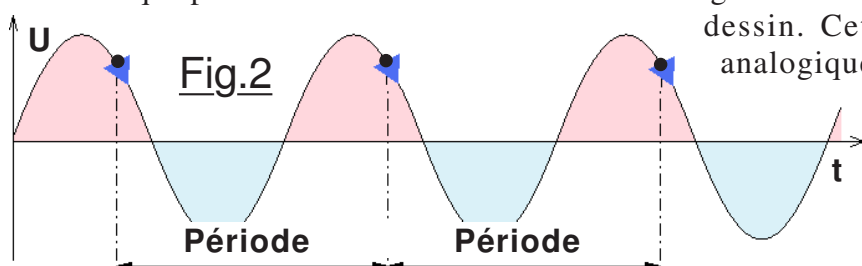
Que peut bien signifier "Partir d'une condition et y revenir" ?

Comme montré sur la Fig.1, un signal périodique peut évoluer de façon quelconque. Mais régulièrement la trame recommence. C'est la **définition intrinsèque des phénomènes périodiques**.

Sur la Fig.1 nous avons un signal électrique complexe. Partir d'une situation initiale et attendre qu'elle ne se reproduise serait très complexe informatiquement. Nous allons limiter nos exigences à des signaux pseudo sinusoïdaux (Avec pour ces derniers une précision moindre.) et surtout des signaux "rectangulaires". Cette restriction n'est pas pénalisante, car dans le monde des microprocesseurs ce



sont les formes électriques de très loin les plus rencontrées. Dans ces conditions, sélectionner une condition initiale, puis être capable de trouver la suivante revient à surveiller à notre choix un front montant ou un front descendant. Par exemple sur la Fig.2 on a choisi le front descendant. La tension de seuil qui permet de surveiller la transition négative est repérée par des triangles bleus sur ce



dessin. Cette tension résulte d'une conversion analogique vers numérique, et nous savons que si l'on ne fait pas du lissage, les mesures sont "floutées" par du bruit électrique. De ce fait, comme le programme que nous utilisons travaille avec une entrée binaire,

dans la mesure où le signal ne présente pas des transitions "verticales" il faut s'attendre à du scintillement. C'est à dire que les mesures successives seront relativement instables. Pour éviter une trop importante fluctuation de l'affichage, l'évaluation des périodes est effectuée en réalité par une séquence de dix mesures consécutives dont on fait la moyenne. (*Lissage rudimentaire.*)

Un caractère "carré" n'est pas forcément mauvais.

L'exemple de signal de type TTL représenté sur la Fig.3 est bien plus significatif de la famille des signaux que l'on rencontre quand on chemine dans le monde des micro automatismes. On peut déjà citer la PWM que génère `P12_Periodemetre_sur_USB_avec_formatage.ino` en tâche de fond et que l'on pourra injecter en `D5` pour vérifier le comportement du programme. Dans ce cas de figure, ce sont les fronts montants qui sont surveillés et représentés par les triangles rouges. Il serait du reste tout à fait équivalent de réaliser les mesures en prenant en compte les fronts descendants, (*Comme montré en bleu.*) le résultat serait strictement le même. Quand le front

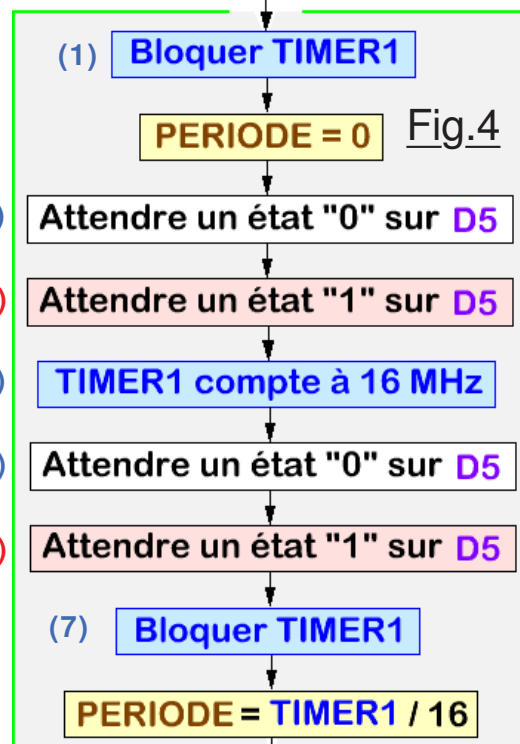
est surveillé par une entrée binaire, les fluctuations dans les mesures sont bien moindres lorsque le signal en épreuve présente des transitions courtes. Ceci étant précisé, la mesure des périodes ainsi que celles des impulsions sera bien moins "calme" que celle des fréquences quelle que soit la stabilité des signaux sur `D5`.

Principe du mesurage des PÉRIODES.

L'organigramme de la Fig.4 montre dans le détail comment on procède pour effectuer la mesure d'une période. En

(1) le `TIMER1` est bloqué, on peut le mettre à zéro en vue d'effectuer une mesure. En (2) on se place en attente de la transition montante initiale. Cette boucle d'attente peut se produire à un moment quelconque lorsque le signal est à "1" ou à "0" : Peu importe. En (3) le passage à l'état "1" annonce que la transition montante s'est produite, donc immédiatement en (4) le `TIMER1` est configuré en chronomètre. Il "cavale" à sa cadence maximale possible, c'est à dire à la fréquence de l'horloge d'Arduino qui frétille allègrement à 16MHz. En (5) on anticipe à nouveau l'arrivée du prochain front montant qui marquera la fin de la période analysée. Enfin en (6) la deuxième transition montante est détectée. Rapidement en (7) on bloque `TIMER1`. À partir d'ici la mesure est complète, et l'on n'est plus dans l'urgence pour extraire la grandeur de la `PERIODE` qui sera issue de la valeur résiduelle de `TIMER1` et de `Nb_IRQ`. On divise par 16 pour avoir la période exprimée en μS alors que pour obtenir la meilleure définition possible on a laissé `TIMER1` galoper à 16MHz, c'est à dire 16 fois "trop rapidement". Concrètement, pour diminuer par dix les fluctuations de mesures, tout ce qui se trouve encadré en vert et grisé est effectué dix fois et les valeurs des mesures totalisées. Il suffit ensuite de diviser par dix pour en faire la moyenne. On retrouve la technique de lissage déjà utilisée lors des CAN. Quand nous mesurons des fréquences, il suffisait de libérer `TIMER1` pendant rigoureusement une seconde. Cette durée était calibrée à la μS près. Pour mesurer des périodes, Passer de (4) à (5), et de (6) à (7) n'est pas instantané. Par ailleurs, même si tester l'état d'une entrée binaire ne prend que quelques cycles par le microcontrôleur, ces instructions très rapides consomment quelques microsecondes. Il est impossible à l'avance de savoir combien, donc impossible de calibrer à 1 μS . Mesurer des périodes sera toujours plus fluctuant que mesurer des fréquences.

Mesurer une PÉRIODE



SUITE

mesures

La mesure des Impulsions et des rapports cycliques avec l'ATmega328 :

Encore une fonction gratuite à ajouter au mini laboratoire, car à l'instar des fréquences et des périodes nous allons encore utiliser l'entrée **D5**. Mesurer des impulsions ne présente pas beaucoup de sens avec des signaux autres que "rectangulaires". Tous ceux qui doivent ajuster la durée d'un temporisateur, la constante de temps d'un monostable qu'il soit redéclenchable ou non comprendront immédiatement le parti que l'on peut tirer d'un appareil de mesure qui permet de mesurer la durée d'une impulsion à l'état "1" ou celle d'un créneau à l'état "0". Mais dans l'univers des petits automatismes, ce sont les signaux rectangulaires périodiques tels que ceux montrés sur la Fig.1 qui envahissent le devant de la scène. Sur ce dessin **Pls +** représente une impulsion dite positive. Qu'elle soit en niveaux TTL entre 0 et +5Vcc, en RS232 comprise entre -12Vcc et +12Vcc n'est pas significatif. Ce qui importe c'est sa durée à l'état "1". De manière analogue **Pls -** est une impulsion réputée négative. Elle se caractérise par sa durée à l'état "0". Comme le signal est répétitif, vous avez déjà compris que la somme des deux durées donne la période **T**.

Statistiquement, dans le territoire d'Arduino, les durées **Pls +** et **Pls -** ne sont pas les informations les plus significatives. Le plus souvent, on désire gérer une sortie nommée "analogique" pour illuminer plus ou moins une LED par exemple. Dans ce cas, l'information pertinente n'est plus du tout la fréquence, la durée **Pls +** ou la durée **Pls -**. L'effet lumineux ne dépendra que de la **valeur efficace** pour peu que la fréquence soit suffisamment élevée pour éliminer tout phénomène de clignotement ou de scintillation. Cette valeur efficace est **caractérisée par le rapport cyclique**.

Valeur efficace proportionnelle au **Rapport cyclique**.

Rapport cyclique = **Pls +** / **Période T**. (C'est sans dimension)

Avec **T** = **Pls +** + **Pls -**

La faculté de générer des signaux rectangulaires à fréquence fixe mais de Rapport cyclique pouvant varier de 0 à 100 est tellement commode que l'ATmega328 peut configurer six de ses sorties pour générer de tels signaux. **Les sorties fonctionnent alors en mode PWM.** (*Pulse Width Modulation*) En bon Français on utilise le vocable d'impulsions à modulation de largeur. Sans préciser tous les détails, la PWM sur l'ATmega328 est générée à 490Hz sur quatre des sorties dédiées, dont celle qui sera sollicitée dans notre projet. (*Soit une période de 2040µs*) Comme ce type de génération met en service l'un des TIMERS, donc il est possible d'obtenir un tel signal en tâche de fond. Ainsi disponible en parallèle de certaines fonctions de notre mini laboratoire, on pourra de la sorte tester le fréquencemètre, le périodemètre et naturellement cette nouvelle fonction d'impulsiomètre.

Bien que l'on pourrait générer un signal de rapport cyclique quelconque, le choix s'est porté sur des durées égales pour **Pls +** et **Pls -** soit un rapport cyclique de 0,5. Le petit programme expérimental **P13_Impulsiometre_sur_USB.ino** est encore une copie honteuse du démonstrateur **P12_Periodometre_sur_USB_avec_formatage.ino** avec très peu de modifications. La Fig.2 permet de détailler le principe des mesures. En (1) on attend un état "0" pour se préparer à l'arrivée d'un front montant. Quand en (2) il est détecté (*État "1"*.) on déclenche un chronométrage rapide. Puis on surveille l'arrivée de l'état "0" qui en (3) dénonce le front descendant suivant. On stoppe le compteur et l'on en extrait la valeur de **Pls +**. On recommence quelques part en (5) où un état "1" anticipe la prochaine mesure qui débute en (6) quand **D5** repasse à "0". On attend alors la "remontée" du signal qui en (7) détermine le front montant. Le TIMER1 est alors stoppé et l'on calcule **Pls -**. En réalité pour chaque impulsion on effectue la moyenne sur dix mesures. Mais comme **Pls +** et **Pls -** sont mesurées à des moments différents, **la mesure des impulsions est plus fluctuante que celle des périodes**. Des valeurs de **Pls +** et **Pls -** sont déduites celles de la **période et de la fréquence, du rapport cyclique** mais ce ne sont que des **évaluations approximatives** relativement instables.

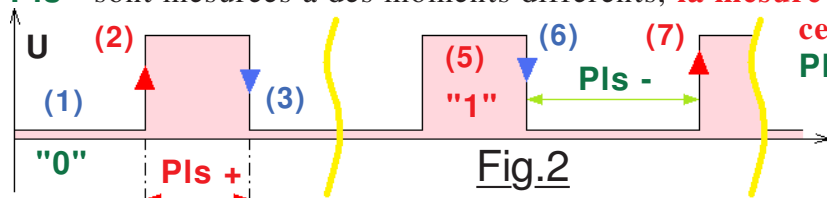


Fig.1

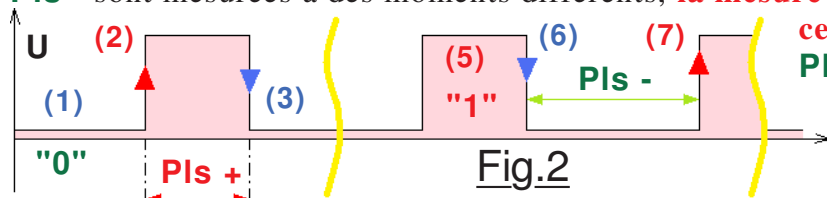


Fig.2

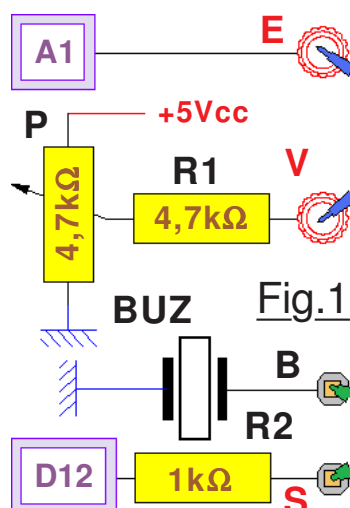
La génération de signaux "carrés" périodiques avec l'ATmega328 :

Jusqu'à cette page, toutes les applications étudiées étaient relatives à du mesurage. Autrement dit, on "injectait" un signal quelconque dans notre système "électronicoinformatique" et l'on déduisait des caractéristiques relatives à ces derniers. Nous ne possédions qu'un multimètre très polyvalent. L'un des domaines dans lesquels les microcontrôleurs sont particulièrement à l'aise, c'est celui de la génération de créneaux binaires calibrés dans le temps. Avec ses sorties PWM et ses divers TIMERS, l'ATmega328 est tout particulièrement adapté à la génération de signaux périodiques, et l'on ne va pas se priver de cette faculté. Le projet va donc pouvoir générer une foule de signaux plus utiles les uns que les autres. Du coup, le multimètre polyvalent est génétiquement modifié pour devenir un vrai petit laboratoire très complet pour électronicien informaticien.

Un générateur B.F. de signaux carrés niveaux TTL.

Quand on annonce "BASSES FRÉQUENCES", il faut relativiser. Autrefois, il y a très longtemps dans une autre galaxie, l'électronique utilisait des tubes cathodiques que l'on nommait des lampes car un filament interne chauffait une cathode. Ils étaient légèrement rougeoyants. Haaaaa ... quelle époque fantastique. Par B.F. on faisait référence aux vibrations audibles par un humain. Quand un générateur commençait à atteindre les 20kHz, nous étions déjà largement dans le domaine des ultrasons. Les tubes cathodiques ont été chassés par les transistors, qui ont été balayés par les circuits intégrés qui à leur tour disparaissent au profit des microcontrôleurs qui à leur tour seront définitivement supplantés par ... Chépas ! Ce que l'on peut affirmer, c'est que dans cette évolution les électroniques sont devenues de plus en plus rapides. Les ondes courtes ont laissé leur place aux VHF, puis UHF et l'on a glissé du mégahertz au gigahertz et plus si affinité. La notion de signal B.F. a subi une inflation analogue, ce sigle empiète largement dans le domaine des ultrasons. Le petit générateur émulé par le démonstrateur [P14_Generation_SIGNAL_sur_D12.ino](#) va couvrir entre 31Hz et 50kHz en quatre gammes pour étaler à convenance les plages d'ajustement.

Matériellement on va devoir investir, car il faudra sur notre mini laboratoire avoir un quelconque bouton pour ajuster la fréquence générée en fonction des besoins. Le changement de gamme utilisera les deux B.P. mais pour la variation linéaire de fréquence rien ne vaut un "truc qui tourne". On va donc investir dans un potentiomètre et un beau bouton à fixer sur son axe. Ce choix reste raisonnable, car ajouter ce composant n'augmentera que faiblement la taille de notre mini laboratoire, et ce d'autant plus que le bouton de commande n'a pas besoin d'avoir un disque gradué. La valeur de la fréquence sera indiquée soit sur l'afficheur LCD, soit sur la ligne série USB. Comme un générateur B.F. délivre des signaux audibles, on peut aussi casser la tirelire et se financer un BUZZER piezoélectrique passif. C'est un composant peu onéreux, et pouvoir entendre le son généré est souvent très utiles ... sans compter de dans la foulée il va permettre d'ajouter au menu des fonctions un TESTEUR DE CONTINUITÉ. C'est décidé, on peut passer au schéma électrique. La Fig.1 montre que le potentiomètre **P** branché en interne fournit sur une borne **V** une tension comprise



entre 0 et +5Vcc. Si l'on ponte **V** avec l'entrée **E** du voltmètre, on pourra déjà réaliser un "auto test" de bon fonctionnement. Mais surtout, ce potentiomètre devient un bouton rotatif de commande pour plusieurs fonctions de notre projet. Le signal sera généré par le microcontrôleur sur la sortie binaire **D12**. Comme il n'est jamais exclus de ponter par erreur la sortie **S** à la masse, la résistance **R2** de 1kΩ pare tout incident. Quand à **R1** le courant qu'elle limite à 1mA maximum met à l'abri **P** des fausses manipulations toujours possibles. Le potentiomètre **P** est choisi avec une valeur faible de 4,7kΩ pour présenter une impédance réduite et ainsi minimiser le "bruit" sur la tension continue qu'il fournit en **V**. L'antiparasitage logiciel se chargera d'éliminer totalement les parasites collectés par la liaison de **V** vers **E** procurant une très bonne stabilité à la fréquence du signal généré. Le potentiomètre **P** est constamment branché sur le +5V interne de notre instrument polyvalent.

Mais il consomme moins de 2mA et dissipe chichement 6mW. Cette puissance reste dérisoire, rien à craindre pour l'alimentation régulée ni pour le potentiomètre **P**.

Générer une tonalité avec le langage C propre à Arduino.

Produire un signal "carré" (*Le rapport cyclique est exactement de 0.5*) précis et stable en fréquence n'a rien d'élémentaire si on désire pouvoir en changer à convenance la fréquence. La méthode consiste, vous l'avez deviné, à calculer la Période en fonction de la fréquence désirée par l'utilisateur, puis de mettre à contribution un TIMER cadencé par l'horloge interne du microcontrôleur. En parallèle à la tâche de fond qui fait alterner **D12** entre "0" et "1" à une cadence très précise, il faut surveiller les boutons poussoir sur **A0**. Indépendamment de ces missions, il importe également de surveiller la tension en **A1** par l'entremise du Convertisseur Analogique Numérique. Visiblement l'ATmega328 ne va pas lambiner, quand à nous on va devoir encore solliciter un TIMER, les interruptions sur débordement et tout le tralala. **ENCORE !**

Bonne nouvelle, on va utiliser toutes ces ressources, mais à travers une instruction spécifique à l'environnement de développement d'Arduino qui fait tout le travail en une seule ligne de code. Ouvrez **P14_Generation_SIGNAL_sur_D12.ino** dans lequel on utilise l'instruction :

```
tone(Broche,Fréquence,Durée);
```

Cette ligne de code très facile à comprendre réalise un travail considérable. Elle conditionne l'ATmega328 de façon à ce qu'il génère sur une broche binaire déclarée en sortie nommée ici **Broche** un signal périodique. Ce signal est généré en continu, (*Des créneaux carrés compris entre 0 et +5V de rapport cyclique 0,5.*) et présente une **Fréquence** précise calibrée à un hertz près. Fabuleux non ? On peut aussi ajouter le paramètre optionnel **Durée** qui précise en millisecondes la durée pendant laquelle la note sera délivrée. Pour faire du Morse ou un BIP sonore c'est très bien, mais dans notre application ce paramètre sera ignoré. Notez que l'instruction **tone** monopolise le TIMER2 qui devient indisponible. Pour le libérer, il suffit d'utiliser l'instruction **notone(Broche)**;

Pour comprendre le reste du programme, il suffit de savoir qu'une boucle mesure en permanence la tension analogique appliquée sur l'entrée **A1**. Le CAN délivre un entier compris entre 0 et 1023.

Les instructions de type **map(NOMBRE, Min, MAX, A, B)**; effectuent la transposition des plages de valeurs de la variable **NOMBRE** allant de **Min** à **MAX** en une gamme allant de **A** à **B**.

Dans notre programme, par exemple **QRG = map(Tension_lue, 11, 1005, 150, 2000)**; est une instruction qui lit le contenu de la variable **Tension_lue** issue de la conversion analogique numérique sur **A1** qui varie en théorie entre 0 et 1023. Mais ici on a resserré les bornes pour obtenir au minimum la gamme annoncée dans les caractéristiques sur **D12**. Quand la tension injectée sur l'entrée **E** variera entre 0 et +5Vcc, l'instruction fournira un entier compris entre **150** et **2000**. Il suffit alors de soumettre au compilateur l'instruction **tone(D12,QRG)**; et la sortie générera une tonalité comprise entre 150Hz et 2000Hz en fonction de l'ajustement du potentiomètre **P**. Avec quatre instructions **map** on obtient avec une facilité déconcertante quatre gammes à notre convenance pour le générateur.

La génération de signaux ÉTALON avec l'ATmega328 :

Abusant des deux instructions décrites dans le chapitre précédent, nous pouvons créer très facilement des signaux de fréquence quelconque. Hors, très souvent ce sont des fréquences précises et "qui tombent juste" dont on a besoin. Dans ce cas on utilise le vocable Étalon de fréquence. Si vous chargez et observez le programme **P15_Generateur_ETALON.ino**, vous allez constater que c'est une variante très proche du programme précédent. Au lieu de calculer **une valeur entière** à partir de la tension issue du potentiomètre, et d'en générer "la fréquence", on soumet directement à l'instruction **tone** la fréquence désirée. Dans notre démonstrateur on a limité à quatorze le nombre de valeurs possibles, mais vous pouvez en ajouter autant que vous voulez. Ces dernières sont échantillonnées globalement dans des séries de type 1, 2, 5. La dernière fréquence de 440Hz est présente pour bien vous convaincre que **tone** est conçue pour faire de la musique. Du 440Hz correspond au LA indice 3 bien connu des musiciens, c'est à dire la note que fait entendre un diapason. Comme pour le générateur B.F. la sortie de cette fonction utilise **notone** qui libère le TIMER2. C'est important car ce dernier participe à la génération PWM dont il est question plus avant.

REMARQUE : *QRG signifie FREQUENCE dans le code Q toujours en vigueur dans la marine, en aviation etc.*

Tester les continuités avec l'ATmega328 :

Souvent nos applications impliquent de réunir les liaisons multifilaires à des entités diverses. Dès que le câblage dépasse six ou huit fils, le repérage au moment de souder se mute en une galère sans nom. C'est alors que le testeur de continuité fait miracle. On branche la première sonde sur la broche du connecteur, puis fil à fil, à l'autre extrémité du toron, on teste les conducteurs avec l'autre pointe de touche. Quand le buzzer couine, on a trouvé le bon fil. Pour prendre un raccourci, un testeur de continuité est un ohmmètre qui fait BIP quand les deux pointes de touche sont mises en court-circuit ... généralement aux extrémités d'un conducteur à trier parmi d'autres.

Un testeur de continuité "intelligent".

Intelligent est un bien grand mot, presque une publicité mensongère. D'une façon générale, les modèles courants se contentent de générer une tonalité bien définie quand une liaison est détectée. La faculté d'effectuer un traitement plus subtil permet de faire mieux. L'idée consiste à différencier la qualité du contact analysé par le truchement de tonalités d'autant plus graves que la résistance est élevée. (*Donc que le contact est mauvais.*) Cinq sons ont été assignés à cinq "niveaux" de résistance. On pourrait en adopter bien plus, mais franchement ce n'est pas utile. Celles adoptées sur le prototype sont indiquées dans le tableau donné ci-contre. Si la résistance est supérieure à 10Ω ou 100Ω on ne peut plus parler de continuité. Si c'est un inverseur ou un relais qui est testé, vous pouvez le mettre à la poubelle, les "grains" des contacts ont charbonné. Donc, pour une résistance plus grande que ces valeurs définies en fonction du calibre, il n'y a plus de tonalité car on considère que les touches de test sont non branchées.

| Plage de Resistance | Tonalité générée |
|---------------------|------------------|
| 0 à 10Ω | 2000 Hz |
| 10Ω à 20Ω | 800 Hz |
| 20Ω à 40Ω | 300 Hz |
| 40Ω à 70Ω | 75 Hz |
| 70Ω à 100Ω | 50 Hz |
| Supérieur à 100Ω | Silence. |

Gamme 5mA

| Plage de Resistance | Tonalité générée |
|---------------------|------------------|
| 0 à 1Ω | 2000 Hz |
| 1Ω à 2Ω | 800 Hz |
| 2Ω à 3,5Ω | 300 Hz |
| 3,5Ω à 6,4Ω | 75 Hz |
| 6,4Ω à 10Ω | 50 Hz |
| Supérieur à 10Ω | Silence. |

Gamme 50mA

L'expérience montre qu'en fonction de ce que l'on désire tester, il faut pouvoir maîtriser le courant qui sera véhiculé par l'élément en cours de vérification. Par exemple pour repérer des petits fils, tester des petits inverseurs et autres minuscules boutons poussoir, il est fortement conseillé de limiter le courant à une intensité dérisoire. Sur notre testeur de continuité on adopte 5mA. En revanche, pour vérifier la piste d'un circuit imprimé, les contacts d'un gros relais de commutation, il vaut mieux au contraire faire passer un courant relativement important. Dans ces conditions on adopte la plage "forte intensité" de 50mA qui suite aux essais semble raisonnable. l'usage montre que des tonalités qui se succèdent sont mieux identifiées auditivement si elles ne sont pas en harmoniques. (*Mêmes notes entendues mais à des gammes différentes.*) C'est la raison pour laquelle les fréquences adoptées ne sont pas des "multiples directs" les unes des autres.

Aspect matériel et aspect logiciel.

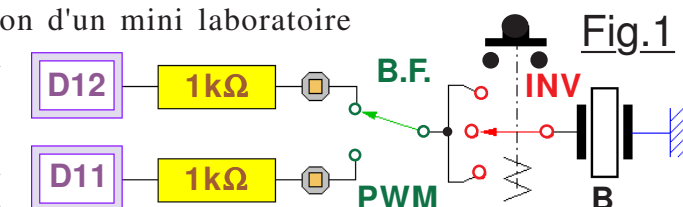
Encore du 100% gratuit, car notre appareil de mesures est déjà équipé de ce qu'il faut pour mesurer des résistances et générer des tonalités. Il n'y a rien à ajouter, mis à part de la consommation d'octets dans la mémoire de programme. Comme nous avons l'intention de "la saturer", ça tombe bien. Ce n'est pas que l'on désire dilapider, mais le microcontrôleur disposant de 32256 emplacements pour le code, autant en utiliser la majorité, les autres ne serviront jamais. Le gaspillage, serait de livrer un ATmega328 presque vide. Le programme servant à la recherche des tonalités [P16_Testeur_de_continuite.ino](#) s'inspire considérablement de celui de l'ohmmètre. Générer des tonalités nous savons faire, les instructions **tone** et **notone(Broche)** prendront en charge la broche **D12** reliée au buzzer. Quand à détecter les continuités, il suffit de réutiliser les entrées **A2** et **A3** munies de leurs résistances de limitation de courant. Ce sont précisément ces deux composants qui déterminent le courant maximal débité lors d'un test de continuité sur un contact électrique qui

globalement se comporte comme un court-circuit. Nous devons prendre les mêmes précautions d'utilisation que celles déjà évoquées pour la fonction ohmmètre.

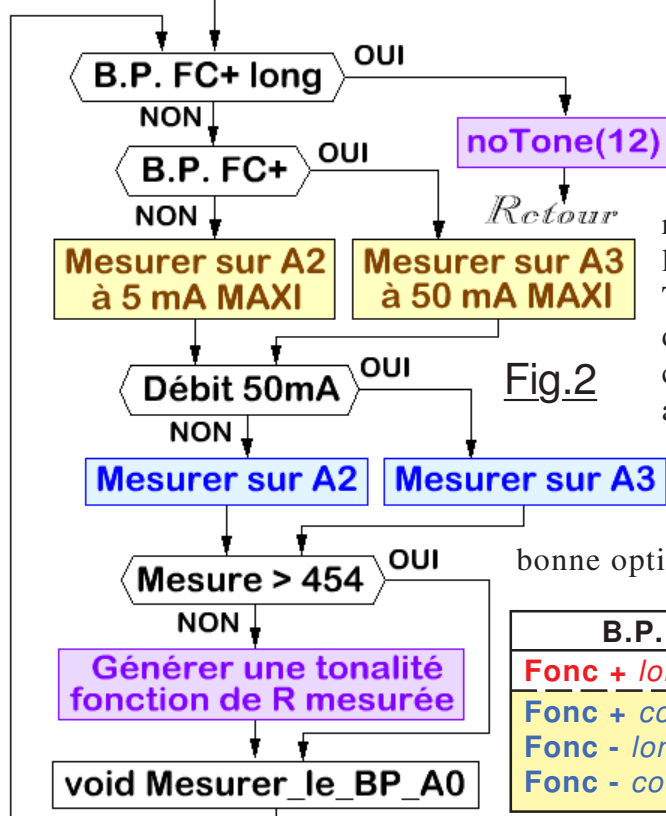
Des goûts de luxe.

Manifestement la solution simple représentée en Fig.1 de la page 32 est très économique, il suffit sur le connecteur de ponter la sortie **D12** au buzzer **B**. Toutefois, si l'on désire améliorer substantiellement l'agrément d'utilisation d'un mini laboratoire totalement autonome, il faut investir dans deux inverseurs, ce qui n'est tout de même pas tragique. La Fig.1 ci-contre présente la solution "confortable" adoptée sur notre prototype. Nous allons faire connaissance plus avant avec la notion de

génération **PWM**. C'est la sortie **D11** (Également munie d'une résistance de protection de $1k\Omega$.) qui sera dédiée à ce type de signaux fournis par notre appareil. Comme ces signaux sont audibles, il peut s'avérer utile de les entendre à titre de vérification. C'est l'inverseur colorié en vert qui nous permet de choisir quelle est la source sonore écoutée. Le signal transite ensuite vers le buzzer **B** à travers l'**INV**erseur rouge. Ce modèle est assez particulier. La position basse est instable, il se comporte comme un bouton poussoir et revient naturellement au centre quand on le relâche. C'est bien commode lorsque l'on désire tester rapidement l'une des deux sorties audible. Enfin sa position vers le haut est stable et autorise une écoute permanente. Rassurez-vous, ces deux éléments n'altère



Fonction TESTEUR DE CONTINUITÉ



strictement pas la compacité de l'appareil réalisé. Ce petit perfectionnement reste donc sans inconvénient mis à part le coût des deux inverseurs ajoutés à la nomenclature.

Pour clore ce chapitre qui sera le dernier permettant de polluer notre espace de travail par des sifflements énervants, un petit regard sur l'organigramme de la Fig.2 démontre que la séquence de programme relative à cette fonction TESTEUR de CONTINUITÉ n'est pas compliquée du tout. On peut en déduire que le bouton **FC +** court fait passer au calibre 50 mA mesuré sur **A3** alors que **FC -** (*Court ou long.*) impose les mesures à 5 mA sur l'entrée **A2**. Contrairement à l'ohmmètre il n'y a pas d'invitation à changer de calibre. C'est à l'opérateur de sélectionner la bonne option en fonction des impératifs du mesurage.

| B.P. | TESTEUR DE CONTINUITÉ. |
|---------------|--|
| Fonc + longue | Sortie en standard de la fonction bouclée. |
| Fonc + courte | Mesures sur A3 à 50 mA en court circuit. |
| Fonc - longue | } Mesures sur A2 à 5 mA en court circuit. |
| Fonc - courte | |

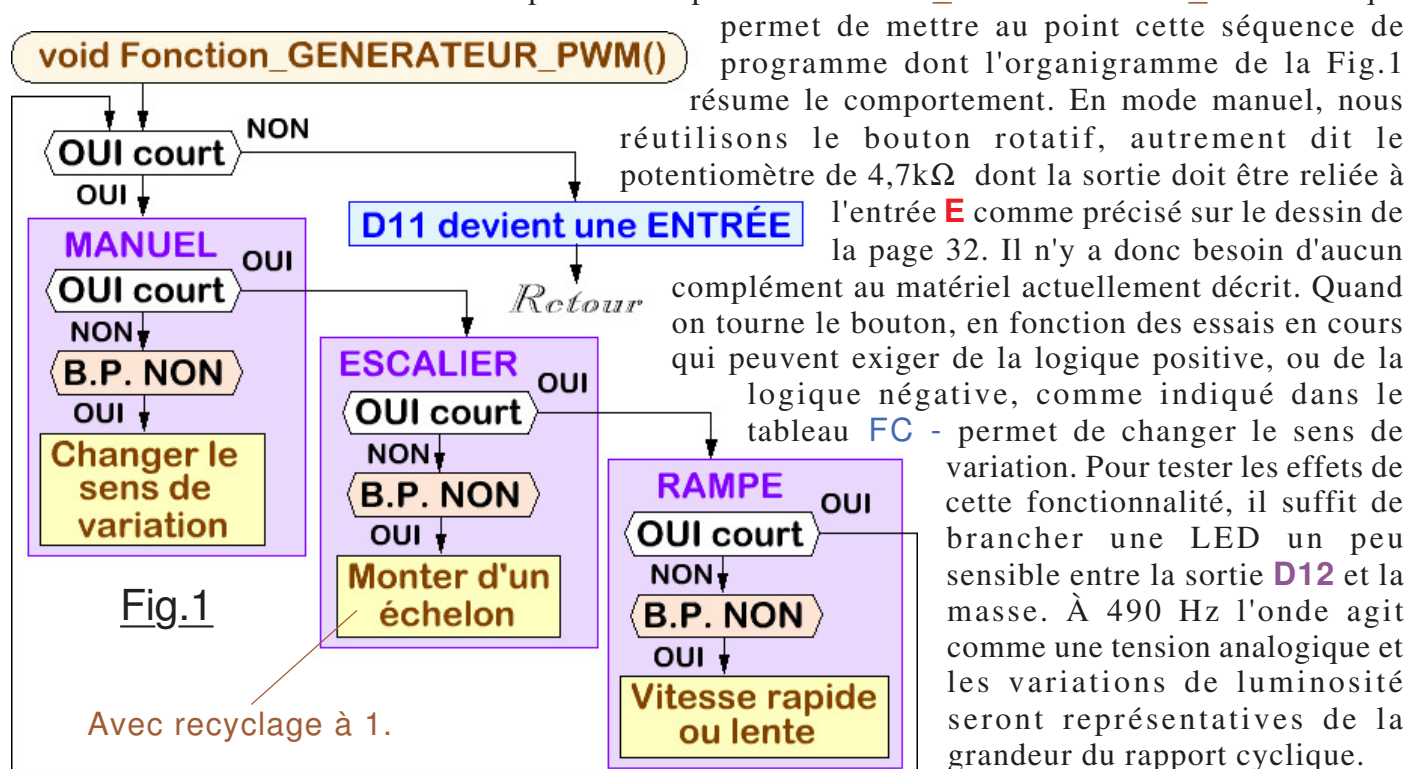
NOTE sur les BUZZER : Attention, tous les buzzers passifs ne présentent pas une sensibilité analogue. Initialement un modèle avait été sélectionné pour ses faibles dimensions. Si l'on reprend le schéma de la Fig.1 située en page 32, on vérifie que la sortie **D12** est protégée par une résistance de $1k\Omega$ qui se trouve en série avec **BUZ**. De ce fait l'élément testé initialement était à peine audible. Il faudra en conséquence choisir une référence assez sensible. Ceci étant précisé, beaucoup de modèles peuvent convenir. Par exemple, le **HPE-227** donne pleinement satisfaction. La référence retenue est un **KBT-33RB-20N** légèrement plus grand en diamètre mais très plat.

Génération d'impulsions à modulation de largeur :

Nombreuses sont les applications faisant appel à des impulsions à largeur modulées, alias de la **P.W.M** : **Pulse Width Modulation**. Se sont des signaux électriques "rectangulaires" dont la fréquence est fixe mais le rapport cyclique peut varier de 0 à 100 en fonction des besoins. Ce type d'onde est tellement employé dans le monde des micro-automatismes que l'ATmega328 peut configurer six de ses sorties pour engendrer de tels signaux. **Les sorties fonctionnent alors en mode PWM**. La sortie **D11** en mode PWM sur l'ATmega328 oscille à un 490Hz précis et stable. (Soit une période de 2040µS) Comme ce type de fonctionnement met en service l'un des TIMERS, il est donc possible d'obtenir un tel **signal en tâche de fond alors que le microcontrôleur est occupé à d'autres travaux**. C'est ainsi que dans notre mini laboratoire, on pourra de la sorte mettre en application cette faculté et pouvoir en "interne" tester le fréquencemètre, le périodemètre, l'impulsiomètre etc. Pour avoir plus de détails concernant cette fonction de service de l'ATmega328 consultez la fiche nommée **Les sorties analogiques PWM**. Bien que ces formes d'ondes soient périodiques et varient entre tout ou rien, elles sont souvent réputées sorties ANALOGIQUES et surtout utilisées comme tel. Par exemple avec ce type de signaux on peut faire varier graduellement le chauffage d'une résistance, la vitesse de rotation d'un moteur à courant continu, la luminosité d'une LED.

Générer de la PWM manuellement ou automatiquement.

Facilitant considérablement le travail des concepteurs, le langage C de l'environnement d'Arduino nous gratifie d'une instruction spécifique avec laquelle "modulo pulser" devient un jeu d'enfant. Par exemple **analogWrite(11, 128)** va fournir un signal carré, (*Donc de rapport cyclique 0,5*) sur la sortie **D11** à une fréquence stable de 489,7Hz. En faisant varier le paramètre **bleu clair** entre 1 et 254, le rapport cyclique va varier entre 0 et 1. On ne peut rêver plus simple. L'encadré en bas de cette page précise les divers modes de fonctionnement ainsi que les manipulations à faire sur les deux B.P. de service pour les imposer. C'est **P17_GENERATEUR_PWM.ino** qui



| B.P. | Génération de signaux PWM. |
|----------------------|---|
| Fonc + longue | Sortie en standard de la fonction bouclée. |
| Fonc + courte | Permuter les modes ESCALIER > RAMPE > MANUEL (Graduel). |
| Fonc - longue | } Mode manuel : Variation Prograde ou Rétrograde. ESCALIER : Incrémenter les échelons. (11 échelons compris entre 3 et 253.) RAMPE : Variation Rapide ou Lente. |
| Fonc - courte | |

Tester un servomoteur avec le mini laboratoire :

Visiblement le servo motorisant les ailerons de roulis sur notre maquette volante ne présente pas un comportement normal. Est-il en cause, est-ce le récepteur ou l'émetteur de télécommandes qui sont en cause ? Autre cas de figure : Vous venez de recevoir le dernier modèle de ServotronicMachin et vous désirez en déterminer les caractéristiques en vue de le mettre en service, tant sur le plan mécanique que celui micro-informatique. Un dispositif fournissant à la demande des signaux permettant de répondre à ce type de besoin sera le bienvenu, d'autant plus qu'il ne nous coutera strictement rien à par un bon paquet d'octets ajoutés au programme global.

Diverses possibilités de pilotage.

Pour en savoir plus sur la technologie des servomoteurs couramment utilisés en modélisme et dans les petits automatismes, consultez la fiche nommée [Pilotage des servomoteurs](#). Seulement trois fils à brancher, on ne peut rêver plus simple. Les technologies utilisées sont anciennes et éprouvées. En revanche, d'un modèle à l'autre on trouve de fortes dispersions de caractéristiques relatives à leur comportement en fonction des signaux reçus. La Fig.1 donne une échelle des durées d'impulsions que l'on pourra générer avec le mini laboratoire. En vert se situent les fréquences correspondant à la norme. Mais force est de constater que sur certains éléments il faut injecter des



impulsions de durées inférieures et supérieures pour faire balayer à leur arbre toute la plage angulaire pour laquelle le moteur est prévu. Les zones jaunes correspondent à des valeurs couramment rencontrées, il est donc impératif que notre instrument de mesure puisse couvrir ces plages. Les valeurs repérées en orange ne sont pas rares non plus, quoique moins fréquentes. Les fourchettes rouges constituent une marge de sécurité pour nous assurer de pouvoir évaluer tout servomoteur du commerce, y compris si ses caractéristiques sont relativement marginales.

Tester un moteur est d'une grande simplicité. Réunir la masse et le +5Vcc pour alimenter l'échantillon. Relier sa ligne de commande (*Le troisième fil.*) à la sortie **D12** déjà sollicitée pour la génération de tonalités B.F. Quand on active la fonction PILOTAGE de SERVOMOTEURS, l'élément branché doit immédiatement tourner pour se placer en position neutre, c'est à dire son orientation centrale par rapport à l'angle total qu'il peut balayer. Le tableau proposé en bas de cette page résume les options présentées par les deux boutons poussoir. Chaque clic sur l'un des deux boutons fait changer de plus ou de moins 100µs l'impulsion "positive". Le moteur doit immédiatement réagir dans un sens ou dans l'autre. Cliquer consécutivement sur le même bouton jusqu'à pousser le moteur dans ses retranchements. Arrivé en limite de sa plage angulaire, il ne tourne plus et se met à "grenailler", une sorte de bruit de grillon. On a trouvé sa limite. Il suffit de lire sur l'écran de visualisation de la ligne série USB la valeur de "Travail" et l'on a une première borne connue pour la programmation. On recommence pour l'autre butée. Si le moteur présente un fonctionnement erratique, c'est soit qu'il consomme trop sur le +5Vcc du mini laboratoire et en perturbe le fonctionnement, soit qu'il est en mauvais état et qu'il doit être changé ou réparé. Si un tel cas se présente, pour lever le doute, alimenter le moteur avec un bloc secteur séparé. C'est d'autant plus concevable que les servomoteurs ne sont pas forcément prévus pour fonctionner sous 5V. Certains exigent bien plus.

ATTENTION : Les servomoteurs sans limitation angulaire présentent un comportement différent. Le signal ne sert plus à les positionner, mais il conditionne leur vitesse de rotation. 1500µs correspond à l'arrêt. Plus la durée s'écarte de cette valeur, plus le moteur tourne rapidement, avec bien entendu une limite. L'écart vers 200µs fait tourner dans un sens, celui vers 3000µs dans l'autre.

| B.P. | Pilotage de SERVOMOTEURS. |
|----------------------|--|
| Fonc + longue | Sortie en standard de la fonction bouclée. |
| Fonc + courte | Augmente la durée du créneau positif de 100µs avec butée à 3000µs. |
| Fonc - longue | } Diminue la durée du créneau positif de 100µs avec butée à 200µs. |
| Fonc - courte | |

Le logiciel associé au testeur de servomoteurs.

L'idée qui vient immédiatement à l'esprit consiste à se servir des sorties spécialisées en PWM de l'ATmega328 puisque c'est précisément ce type d'onde électrique que l'on désire générer. Malheureusement pour nous, comme ces broches spécialisées sont destinées à créer des tensions "ANALOGIQUES", pour privilégier "la rémanence" leur fréquence est relativement élevée et ne convient pas puisque la répétition des impulsions de pilotage doivent se faire à 50 Hz. C'est bien dommage, car il aurait été possible de reléguer la génération PWM en tâche de fond et disposer ainsi du travail en parallèle du microcontrôleur. Tant pis, on doit faire avec ...

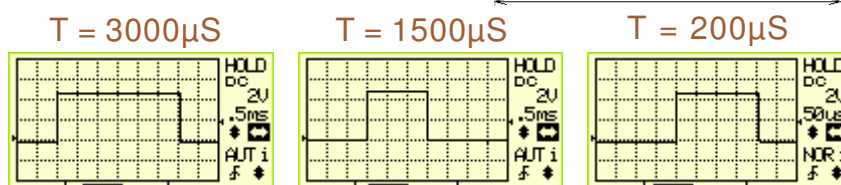
La deuxième piste qui tombe sous le sens, tout au moins si vous avez consulté la fiche relative au [Pilotage des servomoteurs](#), consiste à se faciliter le travail en utilisant les instructions de la bibliothèque spécialisée [Servo.h](#) qui propose des fonctionnalités très alléchantes. Et bien là encore on va oublier les bienfaits de la civilisation, car cette approche présente deux inconvénients :

- Dès que l'on importe cette bibliothèque, la taille du programme augmente de 380 octets, avant d'avoir fait appel à l'une quelconque de ses procédures. Surtout, il puise 92 octets dans la mémoire réservée aux données dynamiques. Hors, pour le programme complet, on commence à friser la correctionnelle dans ce domaine. Du reste le compilateur génère un avertissement à ce sujet.
- La bibliothèque fait appel aux ressources du microcontrôleur et nous ne sommes plus maîtres de la fonction `millis()` qui est employée dans notre procédure `void Tester_le_BP_A0()`. De ce fait la lecture des boutons poussoir n'est plus garantie et se produisent des effets pervers.

Le programme "cousu main".

Globalement la structure de la fonction en cours d'étude est identique à toutes celles déjà créées. C'est celle d'une fonction bouclée dont on sort de façon standard par une utilisation longue du bouton `FC +` et c'est `P18_Pilotage_SERVOMOTEUR_sur_USB.ino` qui nous sert de démonstrateur. Rien de bien nouveau. La difficulté réside dans le fait que le microcontrôleur est en permanence bloqué dans la génération du signal, alors qu'il doit surveiller les deux B.P. Pour cerner le fonctionnement de la fonction bouclée, aidons-nous de l'organigramme très simplifié représenté sur la Fig.2 dans lequel les instructions en jaune appartiennent en réalité à une procédure de service. En (1) on fait monter le signal à l'état "1". Puis, avant de le ramener à "0" en (3) on effectue un délai (2) correspondant à la durée T désirée pour l'impulsion positive. (T comprise entre $200\mu S$ et $3000\mu S$.)

La durée du repos doit être égale à $20ms - T$ puisque la fréquence est imposée à 50Hz. On commence à générer en (4) un premier délai "important" de 15mS. Enfin en (5) on complète ce délai pour que la période complète du "1" plus le "0" fasse exactement $20000\mu S$. Deux instructions en cascade sont prévues. En (4) on temporise à une milliseconde près. En (5) on peut affiner à une microseconde près. Ainsi, moyennant d'utiliser un périodemètre indépendant on peut peaufiner la durée, et ainsi tenir compte du temps passé dans les instructions bleues. C'est franchement du luxe, il faut bien l'avouer. Les trois dessins proposés ci-dessous représentent des copies d'écran effectuées avec un petit oscilloscope numérique DSO 062 et sont mémorisés lorsque le signal est généré pour le neutre, ainsi que pour les deux limites de la plage possible des valeurs prévues pour T .



Pilotage de SERVOMOTEURS

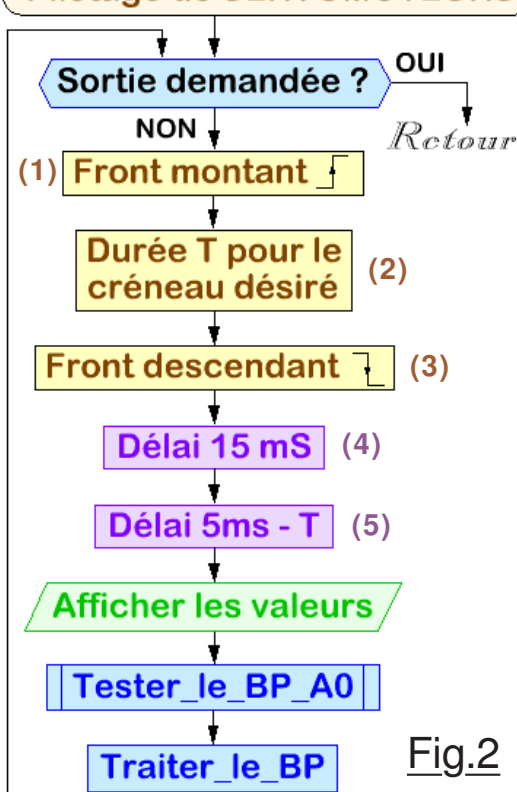


Fig.2

Mesure des condensateurs "de forte capacité" :

Avec cette fonctionnalité on renoue avec le mesurage. Historiquement cette possibilité avait été développée bien avant la génération de signaux en tous genres, le projet initial ne visait que le domaine des mesures, pas celui d'un mini laboratoire polyvalent. Puis l'avancement de la programmation a clairement montré que l'espace disponible en mémoire de programme de l'ATmega328 offrait largement l'opportunité d'augmenter significativement les possibilités. Comme la mesure des condensateurs se montre à l'usage bien moins fréquente que le besoin de disposer de signaux électriques calibrés, cette option a été provisoirement "oubliée". Comme un assemblage du total s'avère possible, au prix de quelques optimisations, autant ajouter le CAPACIMÈTRE dans le menu de base. Seules les capacités supérieures à 9 μ F sont envisagées, car des valeurs inférieures imposent une complication électronique qui sortirait du cadre de ce projet. C'est d'autant plus injustifié qu'il est assez rare d'avoir vraiment à mesurer un condensateur. En revanche, les composants de forte capacité ont tendance au vieillissement qui se traduit par une diminution de leur résistance interne et de leur capacité. Pouvoir vérifier des composants jusqu'à plus de 10000 μ F peut s'avérer utile, et d'autant plus raisonnable que l'investissement cette fois encore ne se limite qu'à du programme.

Retour à l'école primaire !

L'auteur de ces propos serait-il en train de se faire un petit coup de nostalgie ? Ressortir ici ce bon vieux concept des vases communiquant est pour le moins incongru. Patiente, acceptons ce petit moment de déprime et retournons à l'école primaire. La Fig.1 représente deux réservoir **A** et **B** reliés par une canalisation entièrement obstruée par le robinet **R**. (Non, pas de fuite ! J'ai toujours détesté ces lavabos qui fuient et dont on calculait le temps mis pour qu'ils se vident. Il aurait été préférable de faire venir le plombier !) Comme la hauteur **HA** est supérieure à la hauteur **HB**, la

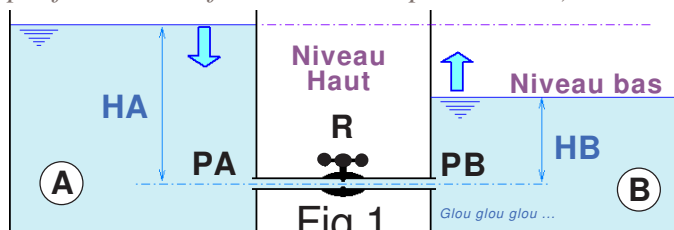


Fig.1

pression qui correspond au poids de l'eau située au dessus en **PA** est plus grande que celle en **PB**. Ouvrons un peu le robinet. (Beaucoup si vous êtes pressés.) **PA** poussant plus fort que ne retient **PB**, l'eau est chassée de **A** vers **B**. Le Niveau haut descend, le niveau bas monte. **Notre instituteur nous apprenait qu'au bout d'un certain temps**

T les deux niveaux arrivaient exactement à une hauteur identique et annonçait péremptoirement ce bon vieux principe universel DES VASES COMMUNICANTS.

Sauriez-vous évaluer le temps **T** ... c'est facile, c'est du niveau primaire ! Et bien la réponse est facile : Il faut un temps infini, soit beaucoup, beaucoup, beaucoup. En effet, au début la différence entre **HA** et **HB** est grande, donc celle entre **PA** et **PB** aussi. Le débit est important. Mais au fur et à mesure que les deux hauteurs se ressemblent, la différence de pression diminue, et la "force" qui pousse les molécules est de plus en plus anémique. Quand les niveaux sont presque identiques, cette différence de pression est tellement faible qu'elle n'arrive plus à vaincre les frottements fluides et le transfert s'achève. Donc, en théorie le réservoir **B** ne sera jamais exactement au même niveau que le réservoir **A**. (Zavez vu que le principe c'est pour des vases, pas pour des réservoirs. Et bien Môamôa je n'ai encore jamais vu deux vases communiquer avec un tuyau !)

Circuit RC et constante de temps.

Ouvrons une fois de plus, et avec bonne humeur, notre manuel d'électricité fondamentale pour réviser cette notion pas vraiment compliquée de **constante de temps**. Rassurez-vous ... il n'y aura pas d'interro ! (Cette fois c'est la nostalgie du collègue qui turlupine l'auteur, ma parole tout le cycle scolaire va y passer si ça continue !) La Fig.2 présente le circuit RC en "configuration intégrateur", c'est à dire que le condensateur **C** est placé en récepteur et alimenté par une tension continue à travers une résistance **R** qui limite le courant. En **G** nous observons un générateur qui délivre un signal "carré" dont l'amplitude varie entre 0 et **+U**. Sur ce dessin le condensateur **C** est représenté en tant que composant polarisé, c'est en effet le cas de la plupart des modèles situés dans la

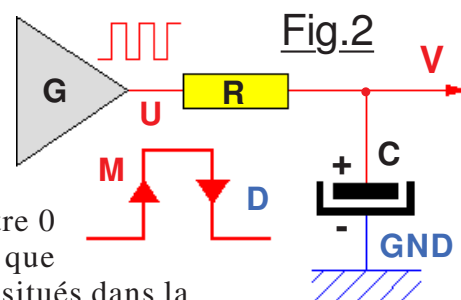


Fig.2

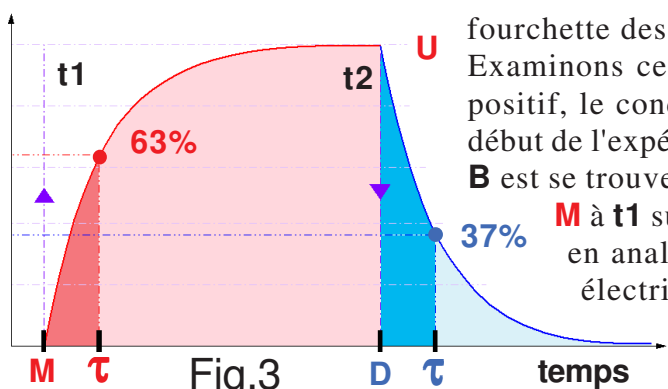


Fig.3

fourchette des éléments mesurables.

Examinons ce qui va se passer lors de l'application d'un créneau positif, le condensateur **C** étant supposé entièrement déchargé au début de l'expérience. Il se comporte exactement comme le réservoir **B** est se trouve totalement vide quand on applique le front montant **M** à **t1** sur la résistance **R** qui ici freine le courant, remplaçant en analogue électrique le robinet hydraulique. La pression électrique en **U** est plus importante que celle en **V** qui augmente progressivement en partant de zéro. Au fur et à mesure que le condensateur **C** se charge, La tension **V** à ses bornes augmente et s'approche

de celle de la source **U**. Comme la différence de pression électrique (*Différence de potentiel pour les spécialistes.*) diminue, le débit s'étioule et la croissance de **V** tend vers du dérisoire ... comme c'était le cas pour les débits hydrauliques. Il faudra donc un temps infini pour que la tension **V** aux bornes du condensateur soit la même que celle **U** du générateur.

Par définition, la constante de temps τ est égale au temps qu'il faut pour charger **C en partant de zéro jusqu'à 63% de la tension **U**.**

De même, on peut également observer la décharge du condensateur **C** quand "entièrement chargé" on applique en amont de la résistance **R** une tension nulle par le front descendant **D** à l'instant **t2**. Les deux pourcentages ne sont strictement pas arbitraires. Ils ont été choisis de façon à privilégier dans le trio **R**, **C** et **τ** la relation simple :

Par définition, la constante de temps τ est égale au temps qu'il faut pour décharger **C en partant de **U** jusqu'à 37% de la tension **U**.**

$$\tau = R \times C.$$

τ exprimé en Secondes.
R exprimé en Ohms.
C donné en Farads.

Technique utilisée pour le mesurage.

Trouver un générateur de créneaux positifs suivi d'une résistance dans notre appareil de mesures n'est pas bien compliqué, il suffit de reprendre la sortie **B.F. (D12)** qui, comme montré sur la Fig.4 est suivie d'une résistance **R** de 220 Ω . On ponte cette sortie sur **E (A1)** qui mesurera la tension. Le condensateur est placé entre l'entrée **E (Côté +)** et la masse. (*Côté -*) Le reste n'est plus qu'une question de programme dont se charge de démonstrateur **P19_Capacimetre_sur_USB.ino**. Pour mesurer un condensateur on fait passer **D12** à +5Vcc puis immédiatement on déclenche un chronométrage avec le **TIMER1** qui provoquera des interruptions sur débordement. L'horloge est utilisée sans prédiviseur, chaque comptage représente 1/16^{ième} de μ S. Pendant que le **TIMEUR1** chronomètre, on mesure la tension sur **A1**. Comme 63% de 1023 est égal à 644, c'est la valeur mesurée sur **A1** qui correspond à la constante de temps **τ** . Pour déterminer la valeur du condensateur

il suffit d'appliquer la formule **$C = \tau / R$** . Le chronomètre travaillant en μ S nous aurons la valeur de la capacité en μ F. Comme chaque comptage représente 0,0625 μ S et que **R** vaut 220 Ω , on corrige le calcul par une division par 672. Avant de recommencer une mesure il faut attendre la pleine décharge du condensateur. Dans ce but on attend que la tension mesurée sur **E** une fois numérisée descende en dessous de la valeur **CAN = 10**. (*Et oui, on ne va pas attendre un temps infini !*) Vous avez compris que pour 10 μ F la mesure sera rapide, mais pour 10000 μ F ou plus la durée entre deux échantillonnages dépasse la seconde. Donc : **La fréquence des mesures diminue quand la capacité mesurée augmente.**

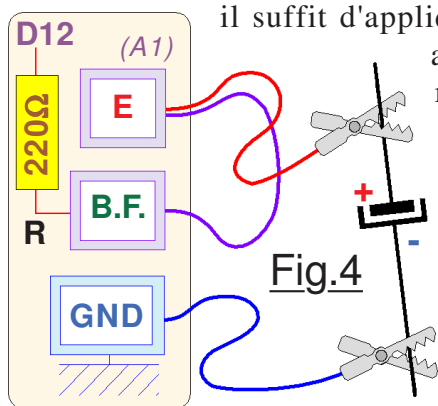


Fig.4

| B.P. | CAPACIMÈTRE. |
|---------------|--|
| Fonc + longue | Sortie en standard de la fonction bouclée. |
| Fonc + courte | Ignoré. |
| Fonc - longue | } PAUSE pour mémoriser la valeur mesurée. (HOLD) |
| Fonc - courte | |

Voltmètre à quatre calibres.

Avec ce chapitre nous faisons une dernière incursion dans le domaine du mesurage. Bien que mis au point dès les débuts du développement de ce logiciel, le calibre 50Vcc avait été éludé car il n'apportait pas grand chose au point de vue "nouveau logiciel". Mais avant de s'engager dans la version "définitive", on ne peut en faire l'impasse, ne serait-ce que pour évoquer l'aspect électronique de complément. Le démonstrateur [P20_Voltmètre_aux_calibres_etendus_sur_USB.ino](#) apporte trois calibres supplémentaires à notre mini laboratoire mais impose un complément matériel.

Voltmètre de calibre 50Vcc. >>> ATTENTION DANGER !

Bien que la plage comprise entre zéro et 5Vcc couvre une majorité de nos besoins, force est de constater que de temps à autre nous sommes amenés à désirer pouvoir mesurer des valeurs de tension plus élevées. Par exemple la tension continue qui est fournie au régulateur 5Vcc d'Arduino par un petit bloc secteur, la tension qui alimentera un petit moteur à courant continu, le chargeur de batterie de notre véhicule automobile etc. Comme le chiffre cinq s'impose par une anticipation à la version LCD de notre instrument à tout faire, on va ajouter à notre appareil de mesures une plage dix fois plus élevée. Le prototype version LCD visualisant sur un petit écran à deux lignes de caractère intégrera un affichage analogique des tensions sous forme d'une rampe lumineuse. Sa petite fenêtre est graduée de 0 à 5, donc les trois calibres du voltmètre seront dans des multiples de ce nombre particulier. La Fig.2 en bas de cette page présente l'aspect de cette fonctionnalité potentielle. Naturellement l'entrée doit toujours être protégée contre les surcharges accidentelles. L'idée de base consiste à placer en amont de l'entrée 5Vcc un diviseur de tension par dix. On va persister à utiliser l'entrée 5Vcc car elle est protégée des surtensions et des tensions inverses. Par ailleurs, comme nous l'avons déjà abordé, cette entrée est également reliée à l'entrée binaire **D5** qui permet la mesure des fréquences et des périodes sur des signaux répétitifs. Le schéma retenu Fig.1 résulte de certains compromis et du respect de certaines contraintes. Si on le compare au montage précédent (*Page 12*) donné en Fig.6 on retrouve en couleurs pastel les composants du calibre 5Vcc et en couleurs vives ceux qui sont ajoutés. La fonction **Fréquence** / **Période** utilisant **D5** ne coûte que la résistance **R3**, autant dire pas grand chose. La résistance **R2** a été augmentée à 10kΩ pour augmenter un peu l'impédance d'entrée sur **A1**. Pour créer le nouveau calibre on a ajouté le diviseur de tension par dix constitué de **R4** et de **R5**. La résistance **R5** constitue avec l'ancienne résistance **R** un diviseur de tension et fausse très légèrement la mesure sur le calibre de 5Vcc en **A1**. Pour en minimiser l'influence on serait tenté de la sélectionner la plus grande possible. Mais il ne faut pas omettre le fait que **R4** devra être dix fois plus élevée. Hors n'oublions pas que **R4** voisine le microcontrôleur qui rayonne allègrement "des signaux carrés" à fréquence élevée. Si la résistance est de trop forte valeur elle sera le siège de signaux parasites qui vont entacher les mesures. Par ailleurs, il importe de choisir des valeurs de composants normalisés et très courants pour en faciliter l'approvisionnement, donc des valeurs ordinaires.

De nombreux essais ont abouti aux valeurs du schéma Fig.1 qui donne entièrement satisfaction. L'incidence de la présence de **R5** est facile à compenser par du code pour retrouver la précision normale sur l'entrée **A1**. C'est toujours **A1** qui sera analysée par le programme pour les deux calibres en voltmètre. On constate que pour les deux calibres **E** et **0 à +50Vcc** les entrées **A1** et **D5** sont entièrement protégées par la LED jaunes pour les inversions de polarité et la LED rouge pour les surtensions. Toutes ces petites modifications ajoutées au cours de notre cheminement dans l'univers du mesurage finissent par rendre confuse la vision globale de ce que l'on devra cumuler sur le circuit imprimé complet. Un schéma électrique de l'ensemble est maintenant le bienvenu, il est proposé

sur la Fig.1 donnée en page 44.



Fig.2

Mesure des tensions alternatives avec l'ATmega328.

C'est volontairement que ce chapitre a été relégué à la fin de nos expérimentations relatives au mesurage et ce pour deux raisons : La première, c'est que la mesure des tensions alternative est loin d'être aussi évidente que celle des tensions continues. La deuxième résulte dans le choix de tensions mesurables relativement élevées générant des risques intrinsèques pour ce type de manipulation, et ce autant pour le matériel que pour l'opérateur. Comme avoir à mesurer des tensions alternatives reste "marginal", l'électronique nécessaire n'est pas intégrée dans le mini laboratoire mais fait l'objet d'un petit module à part. Un accessoire que tout le monde ne sera pas obligé de réaliser. Par contre, si l'on désire bénéficier de cette possibilité, il est forcément incontournable de compléter le logiciel en conséquence. C'est encore le petit programme d'expérimentation [P20_Voltmètre_aux_calibres_etendus_sur_USB.ino](#) qui permet de tester cette nouvelle fonction.

Électronique complémentaire pour mesurer les tensions alternatives.

Uniquement capable de numériser des tensions continues avec son CAN, Arduino nous impose de transformer la tension alternative en cours de mesure en une tension redressée et filtrée. Ces deux impératifs ne sont pas sans conséquence technique. Examinons le schéma retenu donné en Fig.1 qui propose l'électronique chargée de cette adaptation. En premier on remarque les deux résistances **R1** et **R2** de $100\text{k}\Omega$ qui sont en série avec l'entrée. Habituellement sur les appareils de

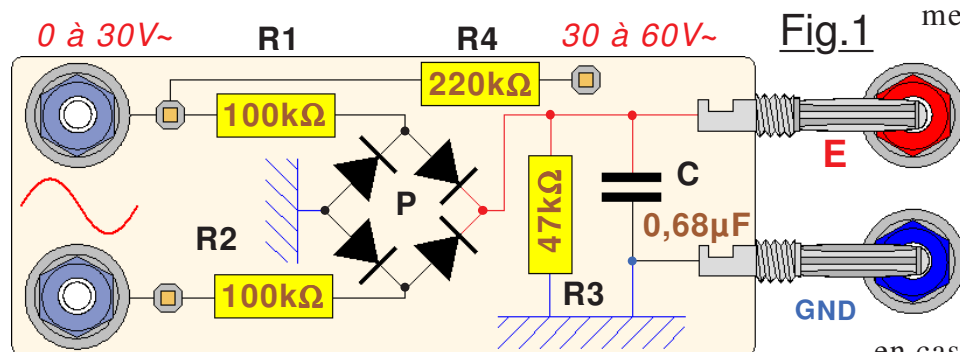


Fig.1

mesure classiques on trouve une seule résistance. Il serait tout à fait possible de n'en mettre qu'une seule de $220\text{k}\Omega$. Mais répartir l'atténuation en deux éléments présente l'avantage sécuritaire de placer les deux lignes en haute impédance par rapport à la masse **GND**. Ainsi, en cas de fausse manipulation le mini

laboratoire sera parfaitement protégé. Par exemple **GND** étant à la terre, votre pointe de touche vient malencontreusement toucher le secteur $220\text{V}\sim$. Les diodes du pont **P** ne seraient alors traversées que par 2mA crête et 1mA moyen. La résistance pour son propre compte ne dissiperait que $0,25\text{W}$ donc une puissance assez dérisoire. Notre matériel est ainsi bien protégé.

La Fig.2 représente par la courbe verte la tension appliquée en entrée de notre adaptateur. Le courant

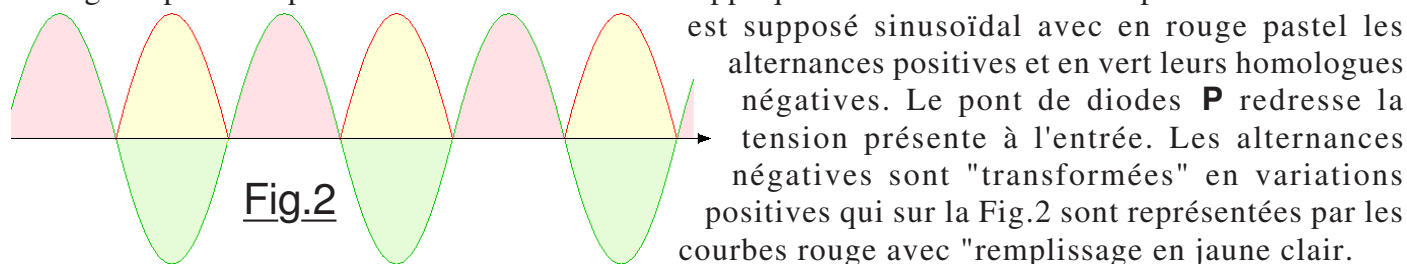


Fig.2

est supposé sinusoïdal avec en rouge pastel les alternances positives et en vert leurs homologues négatives. Le pont de diodes **P** redresse la tension présente à l'entrée. Les alternances négatives sont "transformées" en variations positives qui sur la Fig.2 sont représentées par les courbes rouge avec "remplissage en jaune clair". Le signal ainsi modifié pourrait être soumis au convertisseur analogique vers numériques. Cette belle courbe redressée ne correspond pas à la réalité. En effet, pour que les diodes du pont redresseur puissent conduire, il faut dépasser environ $0,3$ à $0,6$ volts à leurs bornes. Du coup l'onde redressée ne descend plus à zéro, mais une tension permanente d'environ $1,2$ volts (*Car il y a deux diodes traversées à chaque alternance.*) représentée en jaune sur la Fig.3 va générer une tension continue résiduelle. Comme nous allons le voir plus avant, ce détail perturbe considérablement la qualité des mesures pour des tensions faibles. La tension "pulsée" n'est pas directement soumise à l'entrée du

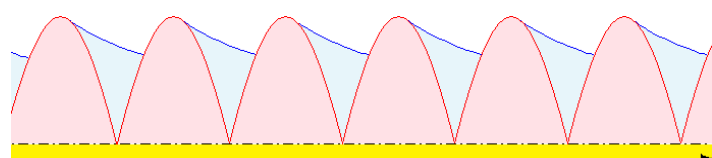


Fig.3

un condensateur **C** vient filtrer cette onde. S'il était seul, il se chargerait, puis la tension aux bornes ayant approché infinitésimalement la tension crête, il ne se passerait plus rien. Le CAN serait soumis à une tension parfaitement continue et égale à la tension crête du

signal analysé. Mais dans ces conditions, l'appareil de mesure serait inutilisable, car une fois débranché, la tension indiquée resterait présente, et tenter de mesurer une tension plus faible deviendrait illusoire. Il faut donc décharger **C** avec **R3**, et dans un délai suffisamment court pour permettre à l'opérateur d'enchaîner rapidement ses manipulations. La présence de cette résistance **R3** dégrade notre belle tension continue, la décharge étant sur la Fig.3 représentée en bleu. Le CAN va donc recevoir une tension "ondulée". On calmera la frénésie des variations par 400 mesures dont on fait la moyenne, technique qui maintenant vous est devenue presque naturelle.

Précision sur ce que l'on affiche.

Fig.4

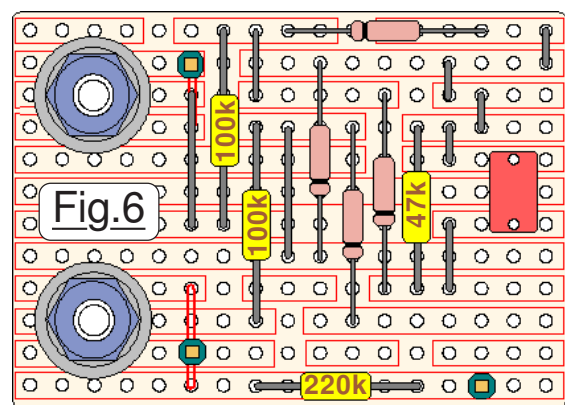
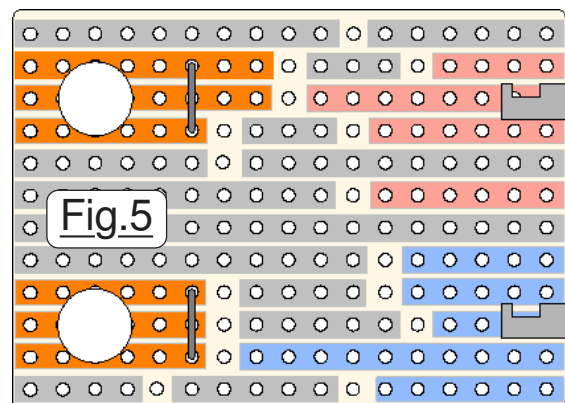
Sachant que chaque mesure résulte d'un lissage de 400 échantillonnages, la moyenne issue du CAN correspondra à la valeur efficace, ou si vous préférez à la "somme des surfaces" coloriées en bleu clair, en rouge pastel et en jaune sur la Fig.3 exprimée en volts et "divisé par le temps". Il suffit d'appliquer un coefficient multiplicateur pour tenir compte de la valeur des résistances **R1** et **R2**, de l'atténuation apportée par **R3** et du lissage plus ou moins arbitraire qu'entraîne **C**. Seule faille, la tension résiduelle jaune qui ne fait pas partie du signal alternatif mais résulte de l'imperfection matérielle des diodes de redressement. Si la tension présentée en entrée est

de forte valeur, le petit résiduel de $2V_{cc}$ n'est pas tragique. Par contre, si comme montré sur la Fig.4 la tension alternative est de faible valeur, ce résiduel fausse complètement les mesures.

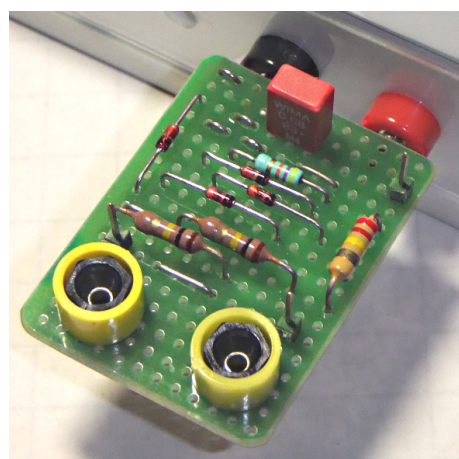
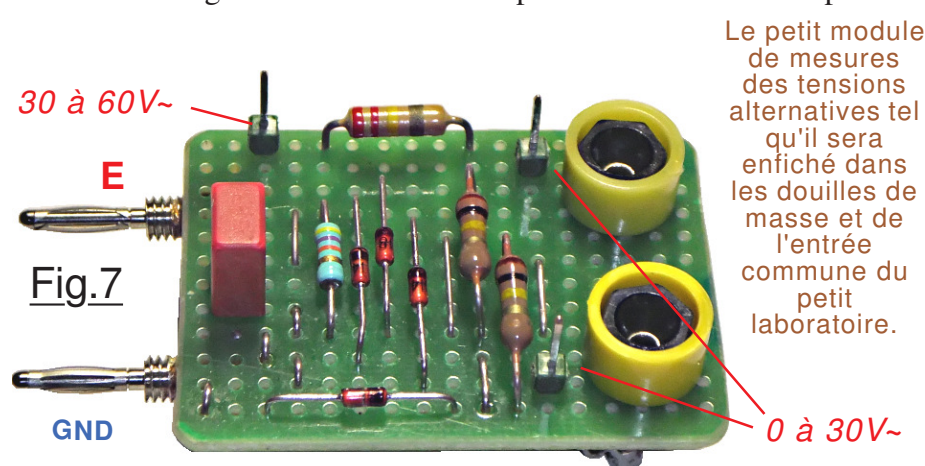
C'est la raison pour laquelle sur les anciens galvanomètres l'échelle du voltmètre alternatif pour les tensions faibles n'était pas linéaire. Dans notre cas, la gamme de portée $30V_{\sim}$ ne commence qu'à partir de $3V_{\sim}$ et celle de $60V_{\sim}$ ne sera pas utilisée en dessous de $30V_{\sim}$. Un graphe de corrections est donné sur la fiche spécifique à ce module électronique nommée

Adaptateur pour mesurer des tensions \sim .

La Fig.5 montre le dessin du très petit circuit imprimé vu coté cuivre, alors que la Fig.6 le décrit coté composants. Les deux petites douilles pour fiche banane diamètre 2mm sont doublées par des picots toujours utiles pour multiplier les tests sur un prototype. Notez que la plage de mesures pour $60V_{\sim}$ ayant été ajoutée par la suite n'est disponible que sur picot. La photographie de la Fig.7 présente ce tout petit module. Les deux fourreaux jaunes sont de récupération et haussent légèrement les douilles pour diminuer leur dépassement sur le dessous.



et haussent légèrement les douilles pour diminuer leur dépassement sur le dessous.



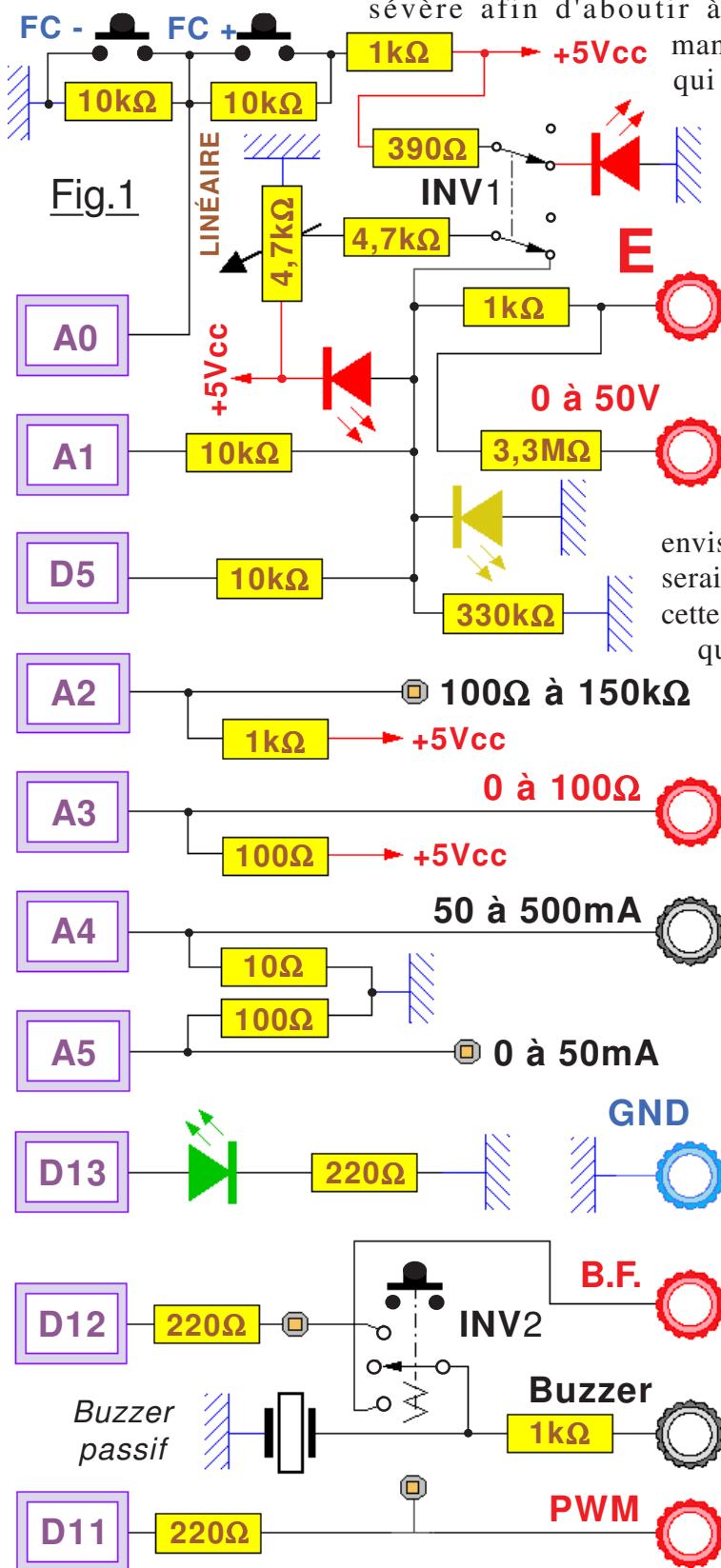
Autant le schéma complet donné sur la Fig.1 (*Sans celui de l'adaptateur pour mesurer les tensions alternatives*) reste simple puisqu'il se limite à dix-sept résistances, deux inverseurs, trois LED, un buzzer passif et deux inverseurs, autant l'intégration en un seul programme des vingt démonstrateurs n'est pas immédiate du tout. Avant de réunir tous ces codes en un seul paquet que peut digérer le compilateur, un petit regard sur le schéma s'impose car quelques modifications de dernière minute ont été effectuées. Vous avez compris qu'elles sont le fruit d'une campagne de tests

sévère afin d'aboutir à une solution pérenne. Les nombreuses manipulations effectuées et surtout une expérience qui se compte en décennies montrent qu'il est bien

plus utile d'utiliser l'inverseur **INV1** pour relier le **+5Vcc variable du potentiomètre** sur l'entrée commune **E** que de basculer le buzzer sur la sortie PWM. En effet, vouloir tester auditivement le 490Hz de la PWM restera marginal, alors qu'utiliser le bouton rotatif du potentiomètre fait partie de l'usage courant de notre laboratoire. Si l'on veut entendre la PWM, il suffit de ponter sa sortie sur l'entré directe du buzzer. L'inverseur **INV2** va directement sur la sortie **D12** via la résistance de **220Ω**. Il serait tout à fait envisageable d'ajouter un inverseur, mais comme ce serait au détriment de la compacité du "produit fini", cette éventualité n'a pas été retenue. Vous pouvez noter que les connections qui véhiculent des courants importants (**A3 et A4**) vont sur des douilles pour fiches bananes, car il n'est pas question d'utiliser pour ces calibres le petit connecteur au standard HE14 du circuit imprimé. La masse ainsi que ces deux liaisons seront réalisées avec un fil électrique de diamètre suffisant.

P21_MINI_LABORATOIRE_sur_USB.ino

Maintenant que les démonstrateurs ont validé chaque séquence de programme, il suffit de les chaîner avec attention en évitant les doublons, tant pour les procédures que pour les variables. Une fois avoir tout réuni et optimisé le code, le compilateur annonce 874 lignes de programme, 20076 octets pour le logiciel et 1734 octets pour la mémoire dynamique. Il prévient qu'il ne reste que 314 octets de mémoire dynamique et que des instabilités peuvent survenir. Rassurez-vous, ce n'est qu'un avertissement. Mais notre programme n'a pas de procédures récursives et ne plonge pas dans des grandes séries d'appels à fonctions effectués sans retour. Il n'y a donc pas de risque de collision entre la PILE" et la zone "HEAP" ... OUF !





ATTENTION DANGER : LISEZ IMPÉRATIVEMENT CE PARAGRAPHE.



Comme n'importe quel appareil de mesure, aussi simple soit-il, tout outil impose à son utilisateur des compétences pour éviter des incidents tant matériels qu'humains. C'est particulièrement vrai dans le domaine de l'électricité. Branchez le mini laboratoire en mesure d'intensité sur une alimentation quelconque et c'est le court-circuit franc. Alimentation ou Arduino peuvent en subir des dommages. Si durant les expérimentations vous réunissez incorrectement les composants sur une platine d'essai, certains d'entre eux peuvent être détruits. Il importe donc à chacun d'estimer ses limites avant de s'engager dans un chemin sans certitude d'atteindre sans perte la destination. En principe, dans l'environnement d'Arduino on ne rencontre que des tensions faibles et non dangereuses pour les opérateurs. Tout au plus un 24Vcc musclé pour alimenter de gros moteurs par exemple. Mais votre parcours pourra vous amener à ouvrir des coffrets dans lesquels sont présentes des tensions dangereuses, ne serait-ce que l'arrivée 220V du secteur. **Ne jamais ouvrir le boîtier d'un dispositif électrique ou électronique quelconque si votre niveau technique dans les domaines impliqués n'est pas suffisant pour vous préserver de tout risque d'accident, tant matériel qu'humain.** Ce document n'est surtout pas un cours d'électronique ni d'informatique. Ce n'est qu'une porte ouverte vers le mesurage en se promenant dans l'univers fabuleux d'Arduino. Une ballade tout à fait ludique pour nous faire découvrir les possibilités extraordinaires du langage C complété par des spécificités remarquables pour l'ATmega328. De ce fait, si nous garantissons le bon fonctionnement et globalement les performances annoncées dès le dernier branchement effectué, ainsi que le téléportage du programme sur Arduino UNO, nous ne saurions être rendus responsables des erreurs commises par les lecteurs. Ceci étant précisé, compte tenu de la simplicité des schémas et de la robustesse des composants utilisés, en principe tout ne peut que bien se passer.

Je vous souhaite autant de plaisir à découvrir tous ces aspects pas forcément très connus, tout au moins pour des débutants, que j'en ai eu à créer mon mini laboratoire dont je tire pleinement satisfaction. C'est d'autant plus vrai qu'il correspond exactement à ce dont je rêvais. L'avantage avec Arduino, c'est qu'une fois que vous serez capable d'écrire du code sans galérer, vous pourrez adapter le programme à votre guise. C'est tout l'intérêt des systèmes programmables. Se faire du "sur mesure" sera toujours très gratifiant. Le résultat c'est bien, mais c'est surtout le chemin parcouru qui présente de la valeur ...

TABLE DES MATIÈRES :

| | |
|---|-----|
| Préliminaires, mise en bouche | P02 |
| LA MESURE DES RÉSISTANCES avec l'ATmega328. | P04 |
| Le bruit et les mesures en électronique..... | P07 |
| LA MESURE DES INTENSITÉ avec l'ATmega328. | P08 |
| Mesures de base et de natures diverses avec l'ATmega328. | P10 |
| Tout faire avec seulement deux B.P..... | P14 |
| Comptage évènementiel avec l'ATmega328..... | P19 |
| Chronométrage "sur USB". | P25 |
| La mesure des Fréquences avec l'ATmega328. | P26 |
| La mesure des Périodes avec l'ATmega328..... | P29 |
| La mesure des Impulsions et des rapports cycliques avec l'ATmega328. | P31 |
| Tester les continuités avec l'ATmega328..... | P33 |
| La génération de signaux "carrés" périodiques avec l'ATmega328..... | P32 |
| La génération de signaux ÉTALON avec l'ATmega328..... | P33 |
| Tester les continuités avec l'ATmega328..... | P34 |
| Génération d'impulsions à modulation de largeur : | P36 |
| Tester un servomoteur avec le mini laboratoire | P37 |
| CAPACIMÈTRE sur ligne USB | P39 |
| Voltmètre à quatre calibres. | P41 |
| Mesure des tensions alternatives avec l'ATmega328. | P42 |
| L'ASSEMBLAGE FINAL | P44 |