

Petit oscilloscope numérique avec Arduino.

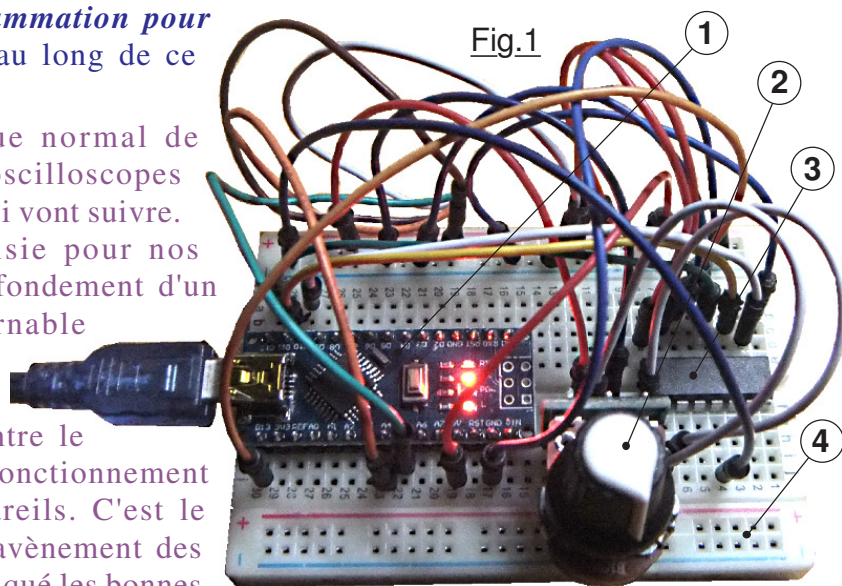
Par Nulentout : Dimanche 22 décembre 2023.

A l'époque où sur la toile d'innombrables oscilloscopes numériques à tous les tarifs sont disponibles, on est en droit de se demander si un tel didacticiel est vraiment utile. *Il s'adresse à toutes celles et ceux qui n'ont jamais été en contact avec un tel appareil* électronique et qui n'ont pas la moindre idée de leur fonctionnement et surtout de la façon de les utiliser. Aussi, je vous propose dans ces lignes une approche très progressive, à la fois théorique et surtout pratique. La meilleure façon d'apprendre, c'est encore de réaliser soi-même son outil. Bien que la réalisation proposée ici est très modeste en performance au regard des "monstres actuels post synchronisés par satellites", elle en contient leur ADN et constitue un tremplin sérieux pour plus tard envisager l'achat d'un appareil plus professionnel. C'est surtout le plaisir d'apprendre dans un contexte ludique qui motive et justifie cette petite réalisation.

Soyons honnête : Ce petit projet n'a initialement été motivé que par le plaisir de programmer en langage C++ pour "ne pas perdre la main". Quand la machine de TURING décrite sur le site de robotique ROBOT MAKER sur <https://www.robot-maker.com/ouvrages/00-amusons-arduino/> a été achevée et le didacticiel mis en ligne, je me suis retrouvé un peu tristounet, comme c'est souvent le cas quand on vient de passer de très agréables vacances et qu'il faut reprendre le joug. Une sorte de "vide", car ce n'est pas les activités ludiques qui me manquent. Mais ... la programmation est une sorte de drogue, une activité qui oblige à faire des efforts intellectuels sans lesquels, à mon âge "de plus de 75 printemps" le cerveau se rouille inexorablement. Bref, je cherchais un prétexte pour rebrancher un Arduino sur le P.C, et des raisons de me creuser les méninges.

Concrètement, nous allons cheminer à notre rythme en associant systématiquement chaque notion abordée à des petites expériences à travers des démonstrateurs simples, qui tous utilisant la même carte Arduino NANO vont permettre une approche méthodique et nous servir non pas de modèles, mais d'exemples possibles de codage en langage C++. *Tous les petits exemples proposés seront optimisés en termes de programmation pour respecter les conseils proposés* tout au long de ce didacticiel *et résumés en Page 68.*

Naturellement il serait presque normal de commencer par l'étude des oscilloscopes puisque c'est le sujet des expériences qui vont suivre. Pourtant ce n'est pas la route choisie pour nos cheminements ludiques. En effet, le fondement d'un oscilloscope réside dans l'incontournable interface Analogique / Numérique qui en conditionne les performances et surtout qui rend possible l'osmose entre le domaine analogique "du réel" et le fonctionnement intrinsèquement binaire de nos appareils. C'est le tournant absolu qui a été initié par l'avènement des circuits binaires qui ont totalement éradiqué les bonnes vieilles technologies cathodiques et envahi définitivement notre quotidien. Quand on aura cerné l'une des façons simples d'interfacer Arduino avec le monde analogique, alors on pourra songer à deux dimensions et représenter de l'électricité sur une surface, la deuxième notion fondamentale étant alors "la base de temps" et le déclenchement. Toutes ces notions seront abordées de façon concrète par des petits programmes démonstrateurs, car seul le concret permet d'établir une relation entre le monde ésotérique de la théorie et celui toujours trop complexe du réel.



01) Première expérience avec le circuit intégré MCP3208.

Convertir une tension analogique en un nombre binaire ou décimal a depuis les débuts de la robotique généré des difficultés considérables et abouti à des techniques aussi variées que bien pensées. Pas question ici d'aborder les méthodes qui se sont multipliées au fur et à mesure des besoins électroniques et informatiques. Inutile d'aborder les techniques de comparaison de tensions, le codage GRAY, les convertisseurs dichotomiques asynchrones, doubles rampes, flash, et bien d'autres encore, tous ces domaines sont hors sujet. *On va se contenter ici de considérer le circuit intégré spécialisé MCP3208 de la Fig2 comme une "boîte noire", sans chercher dans le détail comment il effectue*



Fig.2

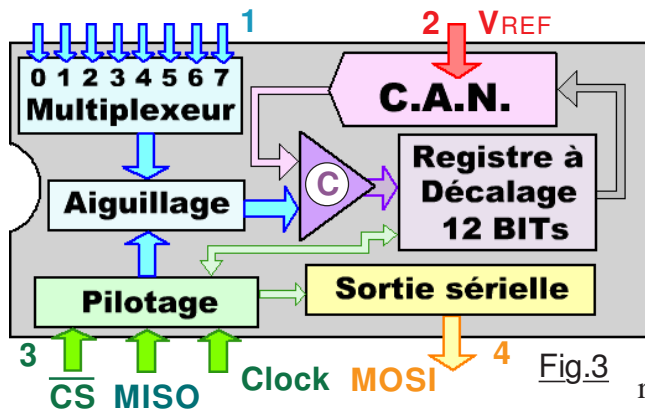


Fig.3

sa mission. Toutefois, nous allons en faire une autopsie minimale sur le synoptique de la Fig.3 pour comprendre les protocoles d'utilisation. Déclencher une numérisation commence par forcer un "0" sur **Chip Select** pour activer les circuits. Puis **MISO** va imposer au circuit de Pilotage l'une des entrées en **1** du **Multiplexeur** qui par l'**Aiguillage** sera orientée vers le comparateur **C**. Par l'entremise du **Convertisseur Analogique Numérique** et de **C** le registre à décalage va se remplir des 12 Bits binaires.

Les valeurs numérisées seront comprises entre 0 et 4096 lorsque la tension analogique de l'entrée **1** aiguillée vers **C** varie entre 0 et **VREF**. Puis à la cadence de **Clock** les 12 Bits par une sortie sérielle **MOSI** en **4** sont injectés dans un registre à décalage pour fournir la donnée binaire de type **int**. Nous en savons assez pour comprendre globalement le démonstrateur **P01_Premier_test.ino** disponible dans le dossier **<Programmes Arduino>**. La valeur de **VREF** peut être comprise entre +0.1V et +5Vcc ce qui autorise une grande sensibilité en entrée.

NOTE : Pour des raisons de simplification, nous n'aborderons pas le **mode différentiel** ou les entrées sont couplées par deux, le signal étant appliqué entre les deux.

➤ **Première expérience avec le convertisseur analogique vers numérique.**

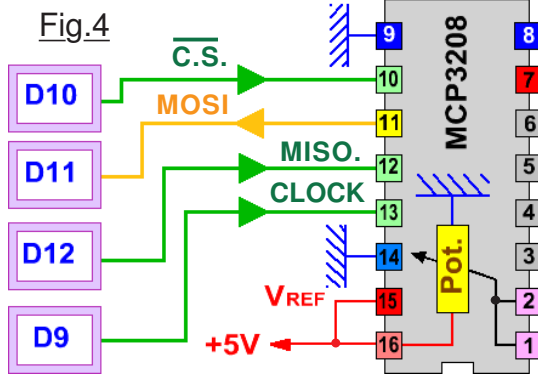
Tout projet Arduino commence comme on peut le voir sur la Fig.1 sur laquelle est concrétisé le schéma de la Fig.4 donné en page 3. On reconnaît en **1** la petite carte Arduino NANO, en **2** un potentiomètre linéaire de valeur quelconque comprise entre **1kΩ** et **47kΩ**. Enfin en **3** le circuit intégré **MCP3208** qui est au cœur de nos préoccupations. L'écart entre les deux lignes de lyres de part et d'autre de la rainure centrale du bloc d'expérimentations **3** fait exactement la distance qui sépare les broches du boîtier DIL du circuit intégré. On peut ainsi l'insérer sans problème sur la plaquette d'expérimentations. Joint à ce tutoriel vous trouverez la notice descriptive **MCP3208.pdf** de ce circuit intégré dans le dossier **<Documents>**.

Compte tenu du fait que fondamentalement pour réaliser un oscilloscope on va utiliser au maximum deux entrées analogiques, vous êtes en droit de vous demander pourquoi ne pas avoir sélectionné un **MCP3204**. Tout simplement parce que c'est le moins coûteux que j'ai trouvé dans le commerce en ligne, sachant que je désirais deux exemplaires. *(C'est indispensable, car lors de la découverte et l'expérimentation nous ne sommes jamais à l'abri d'en détruire un par erreur.)* Pour vous éviter des recherches ils ont été approvisionnés sur :

https://www.amazon.fr/dp/B00M1NFAZY?psc=1&ref=ppx_yo2ov_dt_b_product_details

>>>> Si seule l'expérimentation de l'oscilloscope vous concerne, cet achat est inutile.

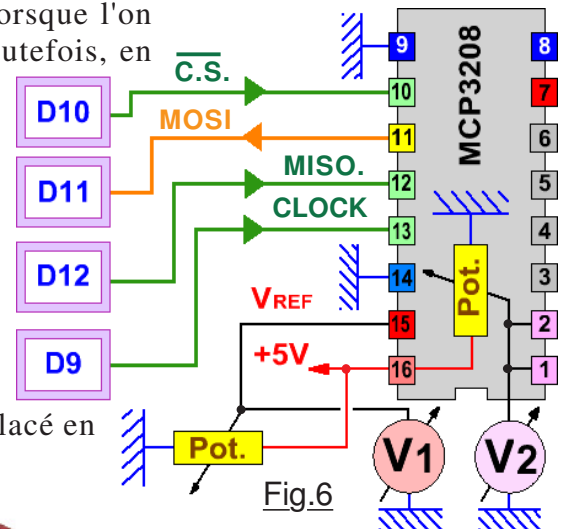
Sachez que je ne suis pas débrouillard sur la toile, vous trouverez certainement mieux. Ceci étant précisé, les deux miens ont fonctionné strictement sans problème, vous pouvez n'en commander qu'un seul. Qui peut le plus peut le moins et cette référence conviendra parfaitement, sachant que vous pouvez opter pour le **MCP3204** qui fonctionnera strictement pareil sans aucune modification du logiciel. Revenons au schéma de branchement pour les liaisons avec Arduino. Vous pouvez remarquer la corrélation des couleurs entre les divers éléments et celles de la Fig.3 et en particulier celles des liaisons filaires. *(Par exemple le rouge pour **VREF** etc.)*



Le canal 8 étant sur GND on devra lire 0. Le canal 7 étant relié au +Vcc indiquera environ 4093. Les entrées entre 3 et 6 étant "laissées en l'air" capteront des tensions électrostatiques quelconques. Enfin, 2 et 3 étant réunies au Potentiomètre afficheront des valeurs couvrant toute la plage de conversion possible en fonction de la position de ce dernier. (*Pot : Linéaire et valeur quelconque comprise entre 1kΩ et 47kΩ.*) Téléverser maintenant le démonstrateur **P01_Premier_test.ino** et activer le **Moniteur de l'IDE**. On obtient des conversions telles que celles de la Fig.5 avec des variations sur le chiffre de poids faible, fluctuations systématiques dans ce domaine. Dans l'exemple visualisé l'ajustement du potentiomètre dépasse un peu la mi-course.

➤ Deuxième expérience : Modification de VREF.

Comme dans la notice descriptive du fournisseur les valeurs possibles pour VREF ne sont pas clairement explicitées, le mieux consiste encore à les déterminer expérimentalement. Faire défiler huit canaux "brouille" le visuel, on va en téléchargeant **P02_Test_de_VREF.ino** n'afficher que la numérisation du Canal n°2. Noter au passage que lorsque l'on ordonne des entités, on commence toujours par la n°1. Toutefois, en informatique une sorte de "tradition" fait commencer par zéro. Du coup, dans les programmes l'ordre des canaux va de 0 à 7. On modifie un peu le circuit de la Fig.1 et montré en Fig.7 on se contente d'ajouter un potentiomètre A imposant une deuxième plaquette d'expérimentation B. Le schéma devient celui de la Fig.6 sur lequel VREF maintenant peut varier car elle aussi étant issue de l'ajustement d'un deuxième Potentiomètre. Pour manipuler dans cette expérience on va devoir utiliser un voltmètre qui sera alternativement placé en V1 ou en V2.



Rien interdit de brancher simultanément deux appareils s'ils sont disponibles.

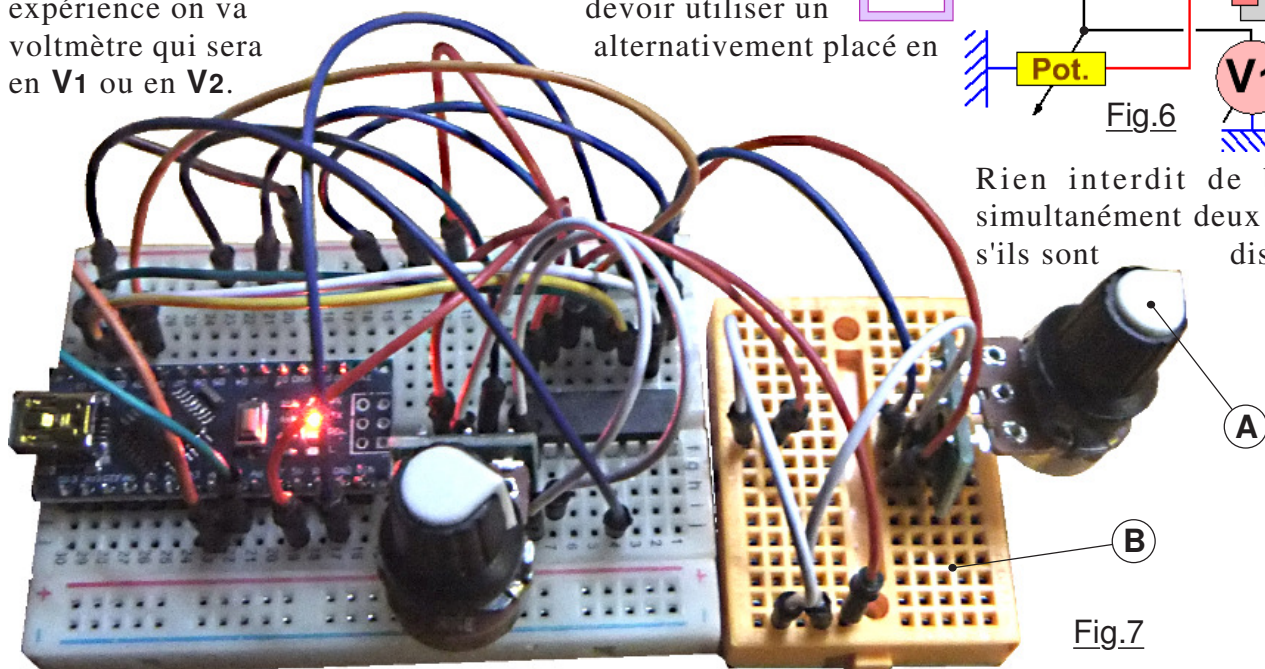


Fig.7

COM3

Fig.5

Programme du 24

Canal 1 :	2252
Canal 2 :	2253
Canal 3 :	40
Canal 4 :	32
Canal 5 :	2
Canal 6 :	21
Canal 7 :	4093
Canal 8 :	0

Canal 1 :	2256
Canal 2 :	2252
Canal 3 :	16
Canal 4 :	35
Canal 5 :	2
Canal 6 :	76

Pour commencer, on ajuste **V1** à +5Vcc. Puis on ajuste la tension **V2** sur la broche **2** du MCP3208 entre 0 et le maximum. On constate dans la fenêtre du **Moniteur de l'IDE** que la numérisation varie comme dans l'expérience précédente entre 0 et environ 4093. Puis, **V1** est ajustée à 2V par exemple. Maintenant, quand on fait varier **V2** entre 0 et +2V la variation de la CAN ne change pas. On obtient toutes les nuances entre 0 et environ 4093. *Puis, si la tension dépasse celle de **VREF**, la valeur numérisée talone à cette butée logicielle maximale de 4093.*

Reprenons cette expérience, mais cette fois **V1** est diminuée à +1V. Faire varier **V2** conduit à un comportement analogue. On retrouve la finesse précédente, avec cette fois une saturation à +1V. Il ne me semble pas très utile de descendre pour **VREF** à moins de 0,1V car j'imagine que les performances du CAN vont se dégrader, et en particulier sa linéarité. Cette manipulation nous enseigne que l'on peut facilement par **VREF** adapter à notre choix la sensibilité du convertisseur.


02) Expérimentation d'un afficheur graphique monochrome.

L'utilisateur n'a que faire d'un nombre compris entre zéro et 4093. C'est la valeur de la tension mesurée qui présente directement "le réel". Aussi, la conversion ayant été effectuée, il faut transposer le résultat en un affichage qui varie entre 0 et **VREF** pour qu'il soit significatif d'une entité palpable. C'est l'objet du démonstrateur **P03_Affichage_sur_ecran.ino**.

Puisque la finalité de ce didacticiel réside dans la réalisation d'un petit oscilloscope, il faudra faire usage d'un écran pour le rendre autonome et qu'il ne dépende plus de la présence d'un P.C.

Autant préparer le terrain et mettre en œuvre un afficheur graphique. Comme le but de ce didacticiel est d'appréhender le fonctionnement des oscilloscopes, que la performance n'est pas le critère prioritaire, on va se contenter d'un afficheur peu onéreux, avec pour pénalité le manque de définition.

Mon choix s'est porté sur l'écran OLED de 1,3 Pouces très facile à se procurer à un tarif raisonnable dans le commerce en ligne. Les références sont kyrielles et celui de la Fig.8 monochrome existe en blanc ou en bleu selon la référence approvisionnée. Il est pilotable par I2C avec seulement deux broches de commande qui sont nécessaires pour le gérer. Il présente une définition 128 x 64 Pixels. Muni d'un contrôleur SSD1306 il est lumineux, donc sans rétro-éclairage. (REMARQUE : Il existe aussi en 0,96 pouces de diagonale. Mais c'est bien celui qui fait 1,3 pouces qui est mis en œuvre ici.)

ATTENTION : Certaines références ne sont pas compatibles avec U8glib.h donc vérifier à la commande. Comme tous les composants populaires, on trouve en ligne une bibliothèque pour les rendre directement et facilement utilisables. Dans notre cas on va utiliser la classique **U8glib.h** qui accompagne ce didacticiel dans le dossier <Bibliothèques>. Également fourni le document  **Bibliothèque U8glib.pdf** à imprimer Recto/Verso pour réaliser un petit livret au format A5 qui résume sur vingt pages les méthodes de **U8glib.h**.



ATTENTION : DANGER ! L'afficheur de la Fig.8 est intégré dans un petit projet de Machine de Turing. Hors, celui que j'utilise pour le l'oscilloscope vient d'une commande passée des années plus tard à une autre source. Bien faire attention car sur ces derniers **les deux broches GND et VCC sont inversées**.



➤ De la numérisation à l'affichage du réel.

Pour illustrer ce propos, le démonstrateur **P03_Affichage_sur_ecran.ino** va transformer le petit montage de la Fig.4 éliminant l'un des potentiomètres en un quintuple voltmètre autorisant la mesure précise et simultanée de huit tensions. S'il s'agissait de notre projet définitif il serait dans la logique de prévoir des calibres différents possibles sur chaque entrée. Dans cette phase de nos expérimentations on va se contenter d'un calibre commun de 0 à +5Vcc et contrairement à l'oscilloscope on ne va pas prévoir des sécurités pour les inversions de polarité ou les dépassements dangereux de tension. Si vous voulez vraiment en faire un appareil pour votre laboratoire, il suffira de transposer le schéma d'entrée de l'oscilloscope et de prévoir des atténuateurs pour mesurer des tensions élevées voir d'ajouter un redresseur pour la mesure des tensions alternatives ...

Le petit logiciel `P03_Affichage_sur_ecran.ino` impose donc au préalable d'installer la bibliothèque `U8glib.h` sur le P.C. et de déclarer cette dernière à l'**IDE**. Puis, le téléversement provoque l'affichage de l'écran Fig.8 pendant quatre secondes avant de passer à celui de la Fig.9 sur lequel on comprend pourquoi seulement cinq canaux sont utilisés sur les huit potentiels. C'est la définition en hauteur de l'afficheur OLED qui limite ce nombre, encore qu'il est vraiment rare dans nos activités ludiques d'avoir à surveiller autant de tensions simultanément. Il serait possible d'afficher les huit canaux avec la police de caractère de 4 x 6 pixels, mais elle est presque illisible et les caractères manquent forcément d'élégance esthétique.

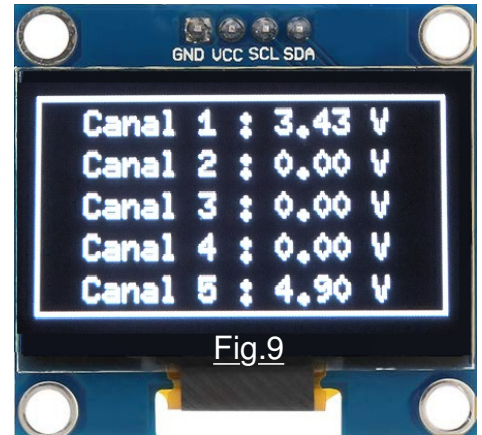


Fig.9

➤ Faire la moyenne de cent mesures.

Bien qu'avec 12 BITS pour la numérisation il serait possible d'afficher avec une précision du milli-volt sur le plan opérationnel ce n'est pas justifiable, et ce d'autant plus que le quatrième chiffre significatif fluctue sans arrêt. Aussi nous allons nous contenter d'une résolution du centième de volt plus que suffisant dans la pratique. Même dans ces conditions l'affichage manque de stabilité. Aussi, pour éviter des changements permanents de la valeur affichée, il suffit de ralentir la fréquence de rafraîchissement. La première idée qui vient à l'esprit consiste à introduire un `delay(100)` dans `void loop()`. Le microcontrôleur passe alors la majorité de son temps "à se tourner les pouces". Aussi il est bien plus astucieux de consommer tu temps pour les numérisations. *Pour chaque canal on va calculer la moyenne de cent mesures qui aura pour effet de "lisser le bruit parasite"*. Comme entre chaque nouvel affichage il faut réaliser 500 conversions analogiques vers numérique, le temps consommé procure à l'affichage une cadence parfaite et une stabilité remarquable. Dans ces conditions on arrive aisément à ajuster la tension variable injectée par un potentiomètre au centième de volt.

➤ Calibration précise des transpositions numériques.

Dans pratiquement tout logiciel on est incité à utiliser des constantes qui fonction du matériel par exemple doivent être calibrées avec précision. *Ces constantes deviennent des paramètres*. Quand on est amené à corriger de nombreux paramètres dans un listage bien plus "étalé" que celui du petit démonstrateur, *il est fortement recommandé de définir tous les paramètres d'initialisation en tête de programme et si possible de les regrouper*. Ainsi, on a une énumération compacte et l'on ne risque pas d'en oublier, et surtout c'est bien plus rapide que d'avoir à chercher de multiples emplacements dans un long listage. Cette façon de faire concerne directement la "programmation avec méthode". Enfin, pour les retrouver immédiatement, les lignes de paramètres sont repérées en les terminant par `@@@@@@@@@@@@@@@@` ce qui naturellement est fait par discipline dans `P03_Affichage_sur_ecran.ino` et dans tous les autres programmes.

Sachant que la valeur retournée par la numérisation dépend directement de celle de `VREF` on doit dans la transposition en tenir compte et faire intervenir un `Coef_de_conversion`. Dans `P03` qu'il faut téléverser, la broche de `VREF` est reliée directement au `+5Vcc` délivré par la ligne USB. En théorie, lorsque l'on injecte directement cette tension sur l'une des entrées, la valeur numérisée est de 4093. Observons les deux lignes de programmation suivantes :

```
① #define Coef_de_conversion 0.001000 //0.001198 @@@@@@@@@@@@@@
② u8g.print(float(TAMPON[Canal] * Coef_de_conversion),2); u8g.print(" V");
```

La ligne ② sert dans une boucle de type `for` à visualiser les cinq lignes, chaque valeur mémorisée dans `TAMPON` étant sélectionnée par le `byte Canal`. Multiplier par 0,001 revient à diviser par 1000. C'est exactement ce que l'on fait avec l'opération `float(TAMPON[Canal] * Coef_de_conversion)`. Étant donné que `Coef_de_conversion` vaut exactement 0.001 il n'effectue aucune transposition de valeur. Si la `CAN` donne 4095, l'affichage sera de `4.095V`. Avec un voltmètre de précision, la tension fournie par la ligne USB mesurée est de 4,9V. L'affichage arrondi à deux chiffres après la virgule donne 4,09V. On en déduit facilement que `Coef_de_conversion` doit être égal au rapport : $4,9 / 4,09 = 1,198$ divisé par 1000 soit 0,001198. C'est la valeur qui est suggérée dans la ligne ①. Vous pouvez facilement effectuer la mesure chez vous pour peaufiner ce paramètre.

► **Un petit dessin pour faire joli !** (SDA sur A4 et SLC sur A5.)

Toujours dans l'optique d'expérimentations ludiques, nous allons exploiter les possibilités graphiques spécifique de l'afficheur OLED utilisé. Vous avez constaté avec le démonstrateur P03 que durant quatre secondes l'écran s'agrément de la petite salamandre, cette dernière étant mon avatar graphique pour toutes mes prestations sur l'Internet. Cette signature s'insinue partout y compris en filigrane dans les pages de textes qui manquent d'images. Ce petit dessin va vous permettre d'observer dans le listage du logiciel la façon dont on réalise une matrice graphique sur OLED. C'est assez particulier. Pour comprendre la technique, reportez-vous en page 12 et page 13 du petit livret [Bibliothèque U8glib.pdf](#) qui accompagne ce tutoriel. (*Je vous conseille fortement de l'imprimer et de l'assembler en consultant le document [Réaliser un petit livret.pdf](#).*) La partie de loin la plus laborieuse consiste à construire la matrice d'octets qui définit la "composition". Comme je suis très paresseux, je me suis contenté de reprendre ici un dessin figurant sur un autre projet, sauf que j'ai utilisé dans cet exemple la technique de la page 13, mais comme le dessin original était petit, *la double grandeur a été appliquée* pour que le petit animal occupe la majeure partie de l'écran. Du



coup la définition est très grossière, car chaque pixel de l'original devient un carré de quatre points ce que la surcharge en rouge met en évidence. Du coup, montré par quelques exemples en violet sur la Fig.10 le dessin est affecté par de nombreuses "marches d'escalier" qui dégradent singulièrement le contour du sympathique batracien. Pour atténuer cette médiocrité de contours, quelques lignes colorées en jaune sur la Fig.110 ont été ajoutées à l'affichage. Une définition deux fois plus grande aurait été plus belle, les contours plus linéaires. *De façon très générale, multiplier par*

deux toutes les dimensions d'un élément augmente sa surface par quatre. Dans notre cas j'aurais été obligé d'étudier quatre fois plus d'octets, tellement indigeste à faire que franchement le courage m'a manqué ! Ceci dit, ce petit amusement montre dans le listage de P03 comment tracer des cadres avec arrondis et des points et des lignes. Largement de quoi donner libre cours à votre imagination d'artiste ...

► **Reprise de P02_Test_de_VREF.ino pour évaluer la rapidité d'échantillonnage.**

C'est précisément la rapidité de mémorisation des échantillons qui conditionne la bande passante d'un oscilloscope. Aussi, pour évaluer la performance possible de notre réalisation, on fait à nouveau appel au démonstrateur P02_Test_de_VREF.ino que l'on doit légèrement modifier. On ouvre l'IDE avec ce programme, et en tête de listage, on transforme la ligne 36 repérée par `//@@@@@@@@` en remarque car c'est la ligne série qui ralentit au maximum la cadence. On constate au passage que dans le résumé de la programmation méthodique on trouve le conseil :

Les lignes qui doivent facilement se retrouver seront terminées par `// @@@@@@@@@@`. Il est respecté dans tous mes logiciels.

Dans ce démonstrateur la fonction `int Lire_valeur(byte Canal)` a été optimisée, en rapidité on ne peut pas faire mieux. La boucle de base simule l'échantillonnage d'un tampon de 1024 échantillons. Cette zone mémoire est enregistrée à la cadence maximale possible. On branche un fréquencemètre précis sur la sortie D13. Dans ces conditions la fréquence mesurée est de 2582 échantillons par seconde. Pour en faire un oscilloscope, c'est dramatiquement bas.

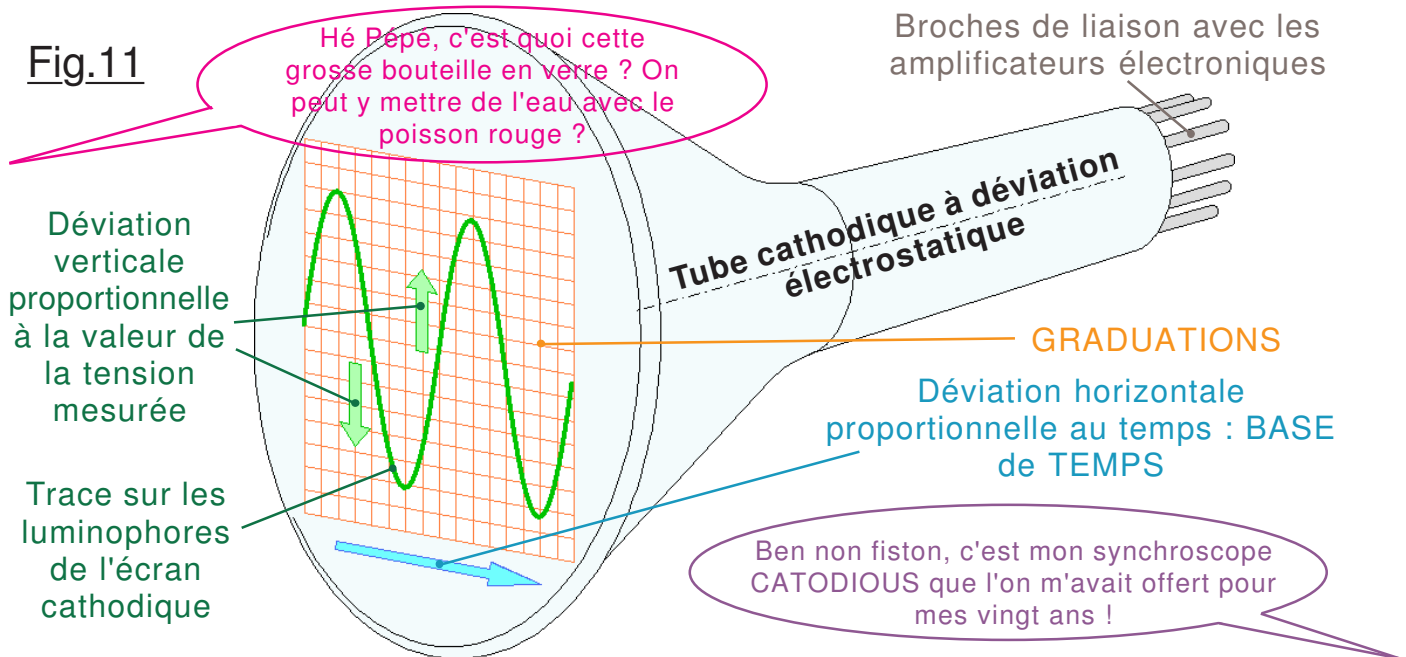
► **Changement de stratégie.**

Face à ce constat décevant, Il nous faut impérativement changer d'approche, et ce d'autant plus que la performance de 12 BITS est complètement dégradée puisqu'en vertical sur notre petit oscilloscope expérimental on dispose au maximum que de 64 PIXELs. C'est à dire que la finesse de 4093 est divisée par 64 pour satisfaire cette limitation. Du coup les CAN d'Arduino peuvent facilement faire le travail. On économise ainsi l'achat du MCP3208, l'appareil sera plus compact et les circuits imprimés largement plus simples. C'est la raison pour laquelle en bas de la page 2 est précisé :

Si seule l'expérimentation de l'oscilloscope vous concerne, l'achat du circuit intégré MCP3308 est inutile.

03) Notion de base : L'oscilloscope analogique de Papi.

Avant de plonger dans la complexité du numérique il nous faut impérativement aborder la notion fondamentale de **BASE de TEMPS**, que nous allons aborder à travers la technologie des tubes à vides qui équipaient tous les ensembles électroniques avant l'avènement du transistor. Gravitant autour d'un tube cathodique servant d'écran graphique, les oscilloscopes de technologie analogique étaient basés sur la présence d'un spot lumineux qui se déplaçait de la gauche vers la droite à une vitesse ajustable pour représenter l'écoulement du temps. En fonction de la tension appliquée en entrée de l'appareil, ce spot était dévié verticalement. Les luminophores allumaient une trace représentative de l'évolution de la tension en fonction du temps. Pour obtenir une trace continue, le balayage était permanent, le spot étant rapidement ramené à gauche pour un nouveau passage. Pour obtenir un tracé exploitable et non un "fouillis" de lignes mélangées, il était impératif de procéder à une synchronisation, c'est à dire un déclenchement précis du balayage pour que les



tracés se superposent. Heureusement pour nous l'échantillonnage mémorisé nous épargne cette facette particulièrement délicate qui exigeait sur les "ancêtres" une foule de boutons et d'ajustements judicieux. Le balayage horizontal devait avoir une rapidité directement liée à la fréquence de l'onde visualisée. Cette donnée de rapidité de balayage est nommée **LA BASE de TEMPS**. Elle est toujours d'actualité, mais sous une autre forme.

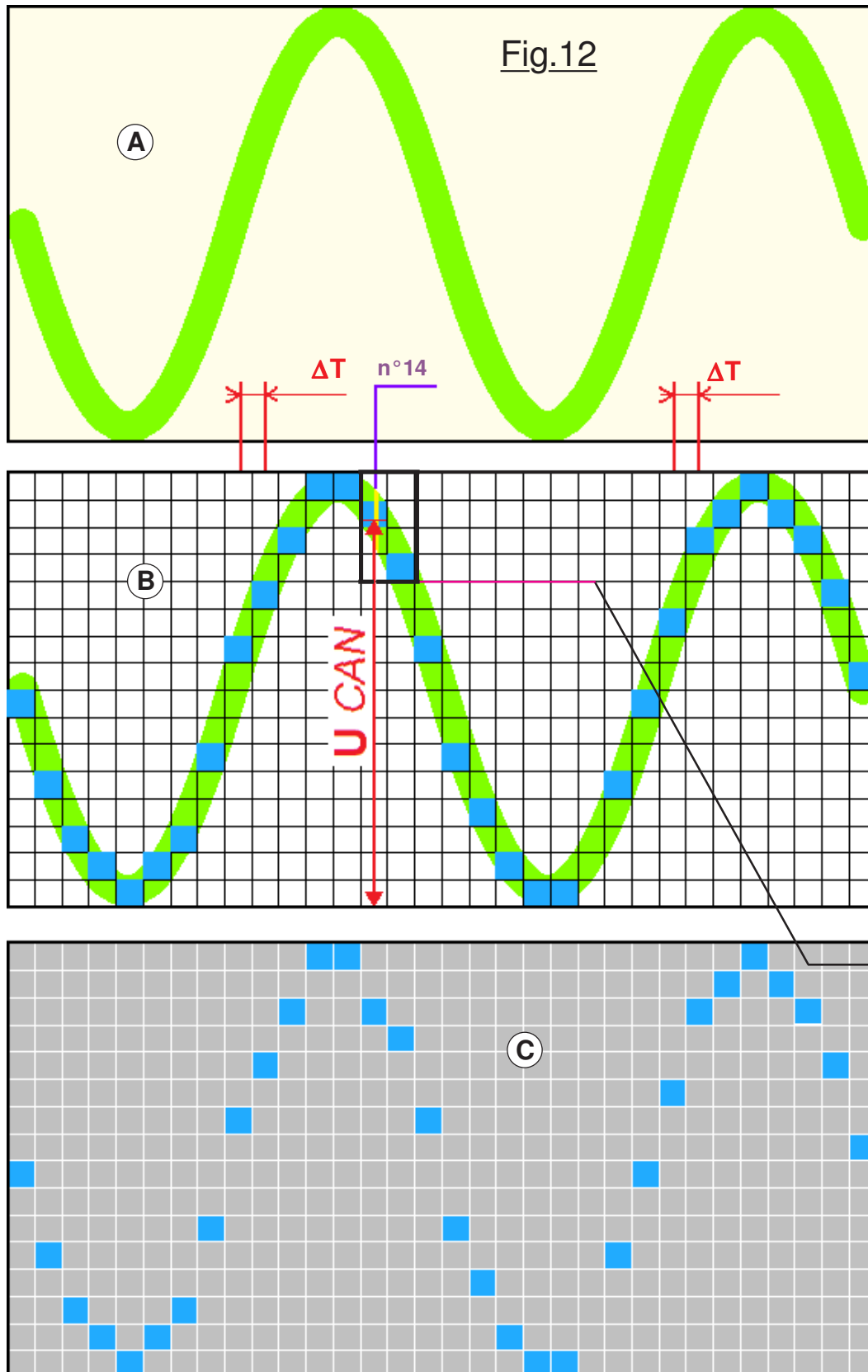
➤ **Passage de l'ère cathodique à celle du silicium.**

Vive l'échantillonnage numérique qui permet de mémoriser les valeurs des tensions mesurées, puis de les tracer sur l'écran en différé. Cette technique évite d'avoir à synchroniser. S'il est indispensable de pouvoir ajuster finement le seuil de déclenchement d'un échantillonnage sur un oscilloscope professionnel, dans un premier temps on peut s'en passer royalement pour une application ludique telle que celle envisagée ici.

Le principe de base est enfantin : À intervalles de temps réguliers et calibrés ΔT on lit la valeur du convertisseur **CAN** comprise entre 0 et 1023 sur Arduino. Chaque échantillon est stocké rapidement dans une mémoire TAMPON. La taille de cette mémoire conditionne le nombre de points "en largeur" qui seront enregistrés. Pour simplifier on se contentera de 128 échantillons, c'est à dire la définition en largeur de l'écran OLED utilisé. Quand la mémoire est pleine, on traite l'affichage en différé. Ces valeurs sont alors translatées entre 1 et 64 la définition verticale de la matrice de PIXELs pour visualiser graphiquement le signal. En vertical nous avons la tension en fonction de l'écoulement du temps en horizontal. Grosse différence avec les tubes cathodiques, l'affichage des PIXELs est mémorisé dans l'écran OLED. Il est figé, stable, et n'a pas à être rafraîchi comme sur un tube cathodique pour éclairer en continu. Du coup, l'échantillonnage étant "une photographie instantanée" avec un temps de pose de 128 échantillons, on peut la déclencher sans restriction.

➤ **Inconvénient n°1 de la technique numérique.**

P our que nous puissions comparer ce qui est comparable, sur la Fig.12 sont représentés les deux technologies. Sur le dessin **A** le tube cathodique et sur **C** l'équivalent actuel d'une dalle de pavés électroluminescents. On suppose ici que les deux technologies ont des écrans de dimensions identiques. Sur l'ancienne technologie le spot de la trace lumineuse en vert est de diamètre analogue au côté d'un pavé électroluminescent bleu pour que dans les deux cas la finesse d'affichage soit comparable. En **B** le dessin montre comment est effectuée la numérisation. Pour en comprendre le principe nous allons examiner le cas de l'échantillon n°14 en partant de la gauche. L'analyse sera plus facile sur la Fig.13 qui constitue un agrandissement de l'encadré rose.



Le milieu des PIXELs se situe latéralement à l'instant précis de la saisie de la tension par la **CAN**. Sur le tube cathodique c'est le milieu de la trace qui correspond à la tension instantanée. C'est sur celui-ci que se fait la **CAN**. Mais verticalement la tension numérisée **U_{CAN}** ne se trouve pas forcément au centre des dalles

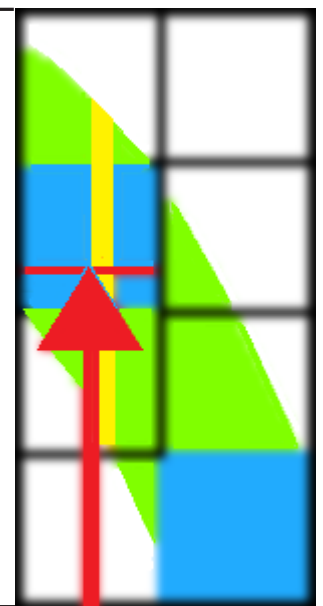


Fig.13

En numérique la trace n'est plus du tout continue et ne peut afficher des lignes verticales si les échantillons sont au nombre de la définition horizontale.

lumineuses. C'est donc le pixel le plus proche "verticalement" qui sera allumé pour visualiser la trace. Contrairement à la visualisation sur tube cathodique, la trace n'est plus continue. *On constate donc que l'ancienne technologie assurait un visuel bien plus élégant.* De plus, sauf à les reconstituer par traitement, des transitions parfaitement verticales comme celles d'un signal carré ne peuvent pas être représentées alors que sur un tube cathodique le tracé étant forcément continu elles sont présentes. (Toutefois en bien moins lumineux car le temps de montée étant très court les luminophores sont peu excités.) Noter que c'est la durée ΔT entre la saisie de deux échantillons qui conditionne la représentation horizontale du temps. C'est donc ΔT qui définit la **BASE de TEMPS**.

➤ **Inconvénient n°2 de la technique numérique.**

Premier cas typique d'une aberration de représentation, celui représenté en rouge sur la Fig.14 lorsque la fréquence d'échantillonnage est proche de celle du signal analysé. (Proche, ou le pire des cas égale à cette dernière.) Sur ce dessin les instants de numérisation sont représentés par les verticales épaisses rouges. Pour en faciliter le repérage un intervalle ΔT sur deux est colorié en jaune. On observe alors une représentation qui ressemble à celle d'une tension continue. Son amplitude sera fonction de l'instant de déclenchement de l'enregistrement par rapport à la phase du signal. Deuxième cas également typique, une fréquence d'échantillonnage inférieure à celle du signal

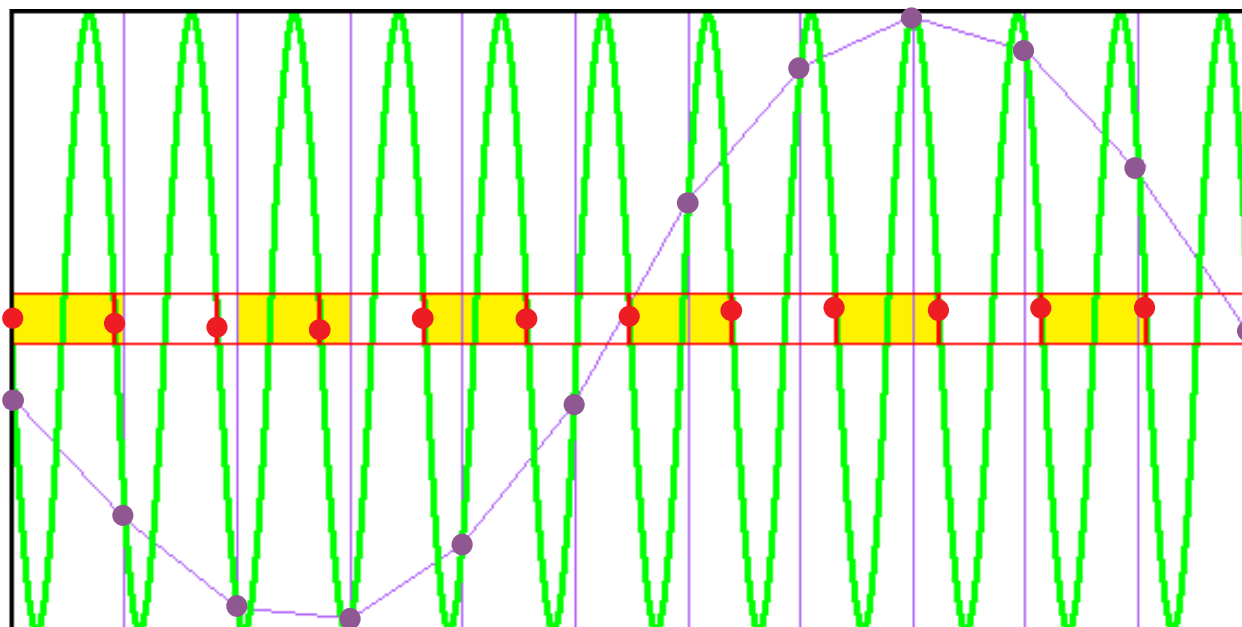


Fig.14

enregistré. C'est le cas représenté en violet, les traits verticaux représentant les instants précis de capture des **CAN**. On observe, que les points visualisés peuvent faire croire à un signal sinusoïdal de fréquence très inférieure à celle de celui qui a été enregistré. Si la forme d'onde n'est pas sinusoïdale, la représentation devient "quelconque" et ne représente absolument pas la réalité.

➤ **La bande passante.**

C'est la caractéristique la plus importante d'un oscilloscope. Elle définit la fréquence la plus grande des signaux qui pourront être visualisés correctement sur l'appareil. Pour un oscilloscope analogique, c'est le gain vertical des amplificateurs d'entrée qui définit la bande passante. Pour simplifier, un exemplaire qui indique 20MHz signifie qu'à partir de cette limite l'amplitude va chuter car le gain en tension va s'effondrer. *Pour un appareil numérique, on estime que la bande passante correspond au dixième de sa fréquence d'échantillonnage.* Pour un appareil numérique

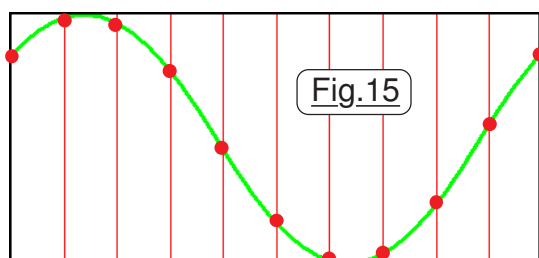


Fig.15

on estime généralement que la limite en fréquence visualisée correctement est celle de l'échantillonnage divisée par dix. Un signal sinusoïdal de fréquence maximale n'est alors représenté par dix points comme sur la Fig.15 sur laquelle le début de la mémorisation des échantillons a été déclenché lors du front montant du signal à environ 2/3 de sa tension crête. On a seulement dix PIXELs par période ce qui est peu.

04) Créer un oscilloscope de caractéristiques modestes avec Arduino.

Avec un investissement en matériel presque dérisoire, nous allons pas à pas aborder toutes les notions précédentes pour qu'elle deviennent des évidences. Puis on va enrichir notre expérience en ajoutant à notre démonstrateur de base des fonctions présentes sur tous les oscilloscopes numériques digne de ce nom. Bref, nous allons nous amuser avec Arduino, et nous construire une culture technique qui ne pourra que nous servir ultérieurement dans nos loisirs informatiques.

➤ L'optimisation logicielle.

Avant de définir le matériel et *le cahier des charge fonctionnel* du futur prototype, il nous faut évaluer la faisabilité et en particulier "dégrossir" la saisie des enregistrements pour évaluer la fréquence d'échantillonnage possible, donc la **BANDE PASSANTE** que l'on pourra espérer. La façon dont vont être saisies les **CAN** influencera également la quantité des échantillons qui seront mémorisés. Bref, il va falloir donner dans l'optimisation et les compromis. C'est l'objet de ce chapitre. Philosophiquement, je considère qu'un petit démonstrateur comme **P02** par exemple n'est pas vraiment un programme. C'est avec ses 3078 Octets un petit "bricolage". (*Même si pour le développer je me suis s'est cogné à des séquences pouvant se montrer réticentes de façon agressive.*) À mon sens, je parle de vrai programme quand il va falloir aligner un nombre considérable d'instructions, et que le code finira par saturer la mémoire disponible et qu'il faudra chercher désespérément de la place. Ce n'est pas quand le bateau coule qu'il faut apprendre à nager. Aussi, y compris pour des tout petits démonstrateurs, il faut optimiser l'optimisation des séquences déjà ultra condensées. En résumé, *optimiser le code est plus qu'un réflexe de programmeur et cette facette doit virer à l'obsession*. En fonction de la séquence en cours de rédaction, le programmeur peut privilégier :

- *Une taille minimale du code* objet qui encombrera la mémoire,
- *Un temps d'exécution minimal* d'une séquence car c'est un critère prioritaire dans le contexte.

➤ Optimiser l'enregistrement des échantillons.

Sachant que les **CAN** des entrées analogiques de l'ATmega328 fonctionnent sur 10 BITS, les valeurs échantillonnées vont s'échelonner entre 0 et 1023. Pour les stocker on va devoir utiliser des **int** consommant deux OCTETS par valeur. C'est un peu du gaspillage, car l'écran graphique ne présente que 64 PIXELs en vertical. Aussi, pour ranger les valeurs sous forme de **byte** il suffit de diviser la valeur de la **CAN** par quatre avant de la stocker. Les valeurs seront ainsi comprises entre 0 et 255. La façon la plus rapide pour réaliser cette division consiste à effectuer deux décalages à gauche. (*Opérateur SHIFT left codé par << 2 en langage C++.*) Toutefois il est légitime de se demander si cette opération de deux décalages à gauche ne dégradera pas la rapidité d'échantillonnage. C'est ici qu'il faut expérimenter pour trouver le meilleur *compromis entre taille et rapidité*.

T éléver le démonstrateur nommé **Teste_Echantillonnage.ino** pour effectuer les essais qui s'imposent. Une partie du listage en Fig.16 présente la boucle de base ② qui effectue l'enregistrement des échantillons. Plus leur nombre sera important, plus la trace contiendra de l'information. Toutefois, avec une largeur d'écran de 128 PIXELs, si l'étendue de la mémoire dépasse cette valeur, il faudra "fenêtrer" la visualisation. Comme à ce stade du tutoriel cette option est

```
① unsigned int TAMPON[512];  
② void loop() {  
③   for (int Octet = 0; Octet <= 512; Octet++) {  
④     digitalWrite(LED_Arduino,HIGH); digitalWrite(LED_Arduino,LOW);  
⑤     TAMPON[Octet] = analogRead(15) << 2;}  
⑥     TAMPON[Octet] = analogRead(15);}
```

Fig.16

envisagée, dans la déclaration en ① on réserve 512 cellules. Inutile de tenter plus, car avec 1024 par exemple, le compilateur refuse une saturation de la mémoire dynamique. Lorsque le démonstrateur est chargé, la ligne ⑤ est en remarque et n'intervient pas. La boucle **for** en ③ se contente de lire la **CAN** et de la stocker les 512 valeurs le plus rapidement possible dans le **TAMPON**. En ligne ④ on crée une impulsion très courte sur la sortie **D13**, signal qui sert à mesurer la fréquence d'échantillonnage avec un appareil très précis. On trouve une cadence de 8301Hz.

C'est tout de même bien meilleur qu'avec le circuit intégré MCP3208 dont la cadence talonnait à 2582 échantillons par seconde bien qu'il soit susceptible de fonctionner à 100 ksps. La lenteur constatée vient de la séquence Arduino qui enchaîne une foule d'instructions qui ralentissent le processus. En conclusion on élimine la présence de ce circuit intégré et on utilisera une entrée analogique de l'ATmega328. Enfin nous n'avons plus besoin de la fonction Lire_valeur(byte Canal) ce qui fait économiser 212 octet qui seront certainement les bienvenus à la fin du développement.

Deuxième essai, on va tenter la division par quatre pour voir si cette dernière est pénalisante en terme de rapidité de traitement. Le but étant de ne manipuler que des données sur huit BITS pour diminuer la place occupée par le TAMPON, il faut au préalable changer la déclaration de la ligne ① par unsigned byte TAMPON[512]. Il importe de déclarer les données en "non signées" pour obtenir un SHIFT pur. Si on oublie ce détail, le décalage ne remplit pas les deux BITS de poids faible par un "0", mais par le BIT de poids faible initial. Puis, on fait passer la ligne ⑥ en remarque et l'on valide la ligne ⑤. À partir d'ici le croquis téléversé va diviser par quatre chaque valeur fournie par le CAN avant de la ranger dans TAMPON. On réactive le logiciel pour mesurer une fréquence d'échantillonnage de la même valeur de 8301Hz.

EXPLICATION : Dans la boucle for de la ligne ③, le traitement ④ prend une durée dérisoire. Par ailleurs, les deux décalages à gauche de la ligne ⑤ ne prennent que quelques microsecondes. C'est le traitement de la boucle for en ③ qui pénalise un peu le processus car il intègre la gestion d'un compteur, ainsi que le stockage de la valeur dans TAMPON qui impose la gestion d'un pointeur en mémoire dynamique. Et puis c'est surtout la numérisation qui se taille la part du lion, elle prend 100µs durée indiquée dans la documentation sur Arduino.

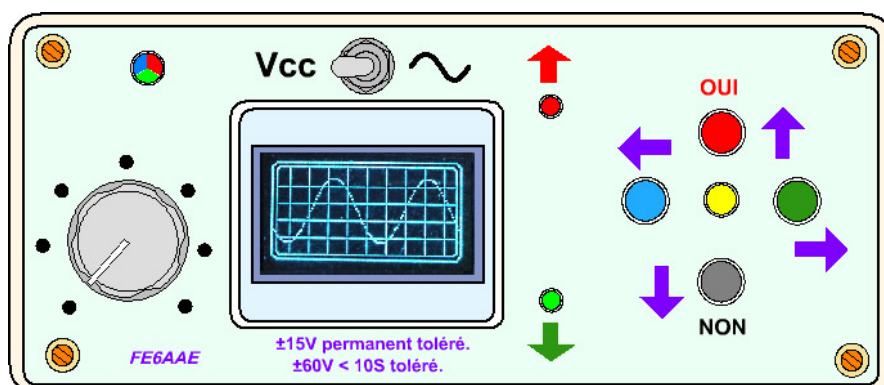
CONCLUSION : Comme traduire les valeurs en les limitant à des byte ne pénalise rigoureusement pas la rapidité d'échantillonnage on pourra se permettre un TAMPON[512] qui autorisera de sauvegarder "quatre écrans de large". De plus, la sauvegarde en mémoire non volatile est envisagée et se justifie totalement dans l'optique expérimentale de ce petit projet. Il restera de ce fait encore 512 autres emplacements en EEPROM, probablement pour y loger des textes du dialogue Homme/Machine affichés sur l'écran OLED. Nous pouvons Sereinement envisager l'avenir ...

05) Définir le cahier des charges fonctionnel : Une étape incontournable.

Foncer tête baissée dans le développement d'un programme est la technique la plus mauvaise que l'on peut adopter. On commence par un schéma électronique initial. Puis on développe les premières séquences. Les essais remettent en cause les choix initiaux ou font émerger de nouvelles idées. Il faut entièrement revoir l'affectation des broches d'E/S. (Entrées/Sorties) Puis on poursuit sur la lancée pour à nouveau penser à de nouvelles fonctionnalités qui encore obligent à tout remanier. Au final, en écrivant rapidement les premières routines on croit aller rapidement vers notre but, et ne pas avoir suffisamment défini le projet on s'engage dans des pertes de temps considérables. Aussi, avant de programmer *il faut absolument consacrer du temps pour déterminer relativement finement ce que l'on désire exactement obtenir* et éventuellement en prouver la faisabilité.

➤ Un cahier des charges fonctionnel "ambitieux".

Par "ambitieux" il faut comprendre que l'on va se faire plaisir au maximum, quitte à réduire les exigences en fonction de la faisabilité et des compromis qu'il faudra consentir au cours des études de faisabilité et des contraintes rencontrées lors du développement. Si la place en mémoire



de programme n'était pas suffisante, on abandonnerait certaines fonctions. La Fig.17 présente une idée générale de ce que pourrait être le coffret de l'appareil qui reste relativement dépouillé. En particulier on opte pour une sobriété dans le nombre de boutons de commande en façade.

Fig.17

Liste des fonctions et options envisagées :

- 01) Limiter à quatre boutons poussoir et un potentiomètre l'encombrement de la façade de pilotage.
- 02) Présence d'une LED triple pour afficher l'état de l'appareil.
- 03) Présence d'une LED jaune entre les boutons poussoir pour attester d'un **Clic long**.
- 04) Protection de l'entrée si surcharge en tension ou inversion de polarité.
- 05) Présence de deux LEDs pour indiquer un débordement de trace vers le haut ou vers le bas.
- 06) Pouvoir passer en pause librement.
- 07) Envisager la présence d'un petit bruiteur passif pour la gestion du clavier et des incidents.
- 08) Envisager deux calibres de sensibilité en entrée.
- 09) Base de temps avec large plage de vitesses d'échantillonnage.
- 10) Synchronisation du déclenchement en option.
- 11) Synchronisation avec **choix du seuil et de la transition**.
- 12) Fournir en permanence un signal étalon.
- 13) Pouvoir sauvegarder une trace en EEPROM et la réafficher à convenance.
- 14) Option d'effacement de la trace présente en EEPROM. (*Si place possible en zone programme.*)
- 15) Pouvoir exporter un enregistrement sur la ligne série USB vers le **Moniteur de l'IDE**.
- 16) Option d'affichage des états des commandes actuelles. (*Synchronisation, base de temps etc.*)
- 17) Enregistrement de "quatre écrans de large" avec affichage fenêtré.
- 18) Envisager l'indication de la valeur efficace du signal affiché dans la fenêtre.
- 19) Envisager la sauvegarde en EEPROM de quatre fenêtres indépendantes.
- 20) Option d'affichage des caractéristiques du signal.
- 21) Pouvoir postsynchroniser le déclenchement de l'enregistrement par satellite. (*Usage de la 4G.*)

Manifestement cette liste donne dans la démesure. Le programme aura beau être optimisé à outrance, **il est peu vraisemblable que nous puissions satisfaire toutes ce prévisionnel**. Toutefois, c'est au tout début d'un projet que l'on doit balayer large quitte par la suite à restreindre les ambitions. Autant il n'est pas dramatique de faire l'impasse sur certains "petits luxes", autant élaborer dès le début un schéma complet pouvant satisfaire tous les critères fait par la suite gagner un temps considérable. Dans cette optique optimiste, nous pouvons passer aux études préliminaires.

➤ Doubler par logiciel le nombre de touches d'un clavier.

Parmi les techniques de programmation que je vous propose dans ces lignes, c'est l'une des plus séduisante à retenir à mon avis, et j'en abuse souvent. L'idée initiale est simple et fait appel à la notion de **clic court** et de **clic long**. (*En fait je n'ai rien inventé, puisque sur les claviers d'ordinateur, avec un clic court on frappe une lettre, avec un clic long la touche passe en répétition.*) Cette technique est très avantageuse et s'applique aussi bien à un B.P. isolé qu'à un clavier multiplexé.

- Si la touche ou le B.P. est cliqué un court instant on génère l'effet n°1 : **Clic court**.
- Si l'enfoncement dépasse un délai calibré on déclenche l'effet n°2 : **Clic long**.

C'est expérimentalement que l'on définit la durée de 0,8S à partir de laquelle l'analyse transite de **Clic court** vers **Clic long**. Dans ce cas l'appareil aura huit commandes par usage de son clavier.

➤ Multiplier par sept (Ou plus.) les commandes de chaque touche.

C'est l'artifice standard omniprésent sur nos appareils lorsque l'on doit gérer des menus conduisant à de nombreuses possibilités. On prévoit dans le projet un codeur rotatif. **Généralement on fait usage à un codeur incrémental à encliquetage** sans limite de rotation angulaire. Dans notre cas d'une réalisation ludique et modeste, on va minimiser l'investissement en composants. (*Clause n°01 de notre cahier des charges.*) Comme il faudra impérativement intégrer un potentiomètre pour le traitement des tensions alternatives, autant se servir de la position de ce dernier lorsque l'on veut invoquer une fonction ou une option. Par exemple un **clic long** sur une touche génère un effet qui sera fonction de la tension potentiométrique. Comme on va le voir par la suite sept "indexages" sont facilement possibles et fiables sur la plage angulaire du potentiomètre. Potentiellement avec quatre touches c'est comme si nous avions un clavier à 28 touches, sans compter que l'on peut faire pareil avec un **clic court** ce qui double les possibilités. En principe nous en aurons assez. Il aurait été possible de se contenter de seulement deux touches, mais avec quatre l'usage sera plus convivial. **Page 12**

➤ Affectation des Entrées/Sorties de la carte Arduino NANO.

L'attribution des E/S d'une carte microcontrôleur est une phase cruciale qui va induire des conséquences importante pour toute la suite du projet. Si les choix initiaux sont judicieux, nous allons gagner un temps considérable lors de l'élaboration du programme d'exploitation. Aussi, sans être exhaustif, on peut déjà citer plusieurs critères de choix :

- **Commencer par les impératifs incontournables.** Par exemple si on peut éviter d'utiliser **D0** et **D1** qui servent au téléchargement du code c'est préférable, car on évite ainsi d'avoir à isoler du matériel ces deux broches à chaque téléversement du code objet. **On sera également souvent tributaire des affectations de certaines "library" qui ne nous laissent pas le choix.**
- Affecter ensuite certaines broches qui permettent la programmation facile de certains périphériques. Par exemple si on désire ajouter un bruiteur passif, les sorties telles que **D3**, **D5**, **D6** etc sont capables de générer de la PWM.
- Chaque fois que c'est possible réutiliser les affectations d'un autre projet facilitant ainsi l'étude du circuit imprimé et l'implantation des composants.
- Si toutes les broches d'E/S ne sont pas utilisées, choisir les moins souples pour laisser le maximum de possibilité en "future amélioration". Par exemple sur une NANO, utiliser en priorité **A6** et **A7** comme entrées analogiques, car ces deux broches ne peuvent pas être initialisées en sorties.
- Si vous envisagez de faire fonctionner ce programme sur une carte UNO, ne pas utiliser **A6** et **A7** d'une carte NANO servant par exemple au développement initial.

➤ Programmer avec méthode : Commencer par vérifier le matériel.

Lorsque l'on va développer le programme d'exploitation de cette petite application on ne veut pas risquer des pertes de temps à rechercher une erreur logicielle, alors que l'un (Ou plusieurs !) des composants à été mal branché. Aussi, **concevoir un programme qui permet de vérifier rapidement l'ensemble du câblage est à mon sens un incontournable.** "avec méthode" commence par choisir des identificateurs "évidents", **la première qualité d'un logiciel quel qu'il soit est sa LISIBILITÉ.** Envisager que votre "production" doit être compréhensible par "tous", et surtout par vous si vous devez reprendre l'ouvrage quelques semaines ou quelques années plus tard. **Un skech comportera inévitablement des calculs non évidents, ou des astuces d'optimisation. Dans ce cas il sera primordial de les expliciter avec des commentaires.** Éviter autant que possible les séquences très longues imposant des défilements du listage sur l'écran de l'ordinateur. **Idéalement, une procédure ou une fonction ne devrait "jamais" dépasser en longueur la possibilité d'affichage de l'écran pour avoir toutes les instructions simultanément visibles.** Pour illustrer ces conseils nous allons utiliser le démonstrateur **P05_Test_materiel.ino** qui ne sera pas commenté informatiquement.

Bien que ce ne soit pas une obligation du tout, **il est fortement recommandé de placer toute fonction ou procédure avant celle qui y fait appel.** Ainsi, en déverminage, on sait que si une séquence est invoquée, il faut la rechercher en amont. Cette recommandation qui est une obligation en langage PASCAL par exemple fait gagner un temps fou quand on dévermine ou quand on améliore un programme "long" c'est à dire incluant des centaines d'instructions.

Pour structurer "proprement" les constantes et les variables utilisées par le logiciel, elles sont classées par type et globalement par ordre alphabétique. Elles sont de plus énoncées par tailles croissantes : byte, int, long, float, tableaux ... Toutes les déclarations sont placées en tête de programme.



Franchement, cette blague sur la post synchronisation par satellite en 4G elle est débile, personne ne va se faire avoir avec une idiotie pareille !

06) Schéma électrique et affectation des Entrées / Sorties sur Arduino.

C'est presque dans l'ordre inverses qu'il faut procéder. Toutefois, pour tester la faisabilité de certaines options du cahier des charges il faut tester certains circuits électroniques au préalable. Ayant conduit ces études préliminaires, je peux vous détailler l'ensemble sur la Fig.18 et en justifier certains choix. Par exemple pour **A4** et **A5** nous sommes tributaires de la bibliothèque **U8glib.h** qui utilise une ligne de dialogue de type I2C. (*Two Wire Interface.*) Pour gérer la LED triple, **D8**, **D9** et **D10** ainsi que **D2** pour la LED jaune et **D4** pour le bruiteur et **A0** pour **Pot** ont été choisies dans le but de faciliter l'implantation des composants sur le circuit imprimé. Par ailleurs, **D4** peut générer de la PWM ce qui éventuellement pourra servir pour gérer le bruitage. Pour la génération du signal étalon **D3** facilite également l'étude du circuit imprimé. Pour la gestion des

Globalement la valeur des résistances n'est pas du tout critique. On constate qu'il y a beaucoup de composants de 10kΩ. Je privilégie cette valeur car elle conduit à des impédances raisonnables, du coup j'en ai approvisionné un lot de cent !

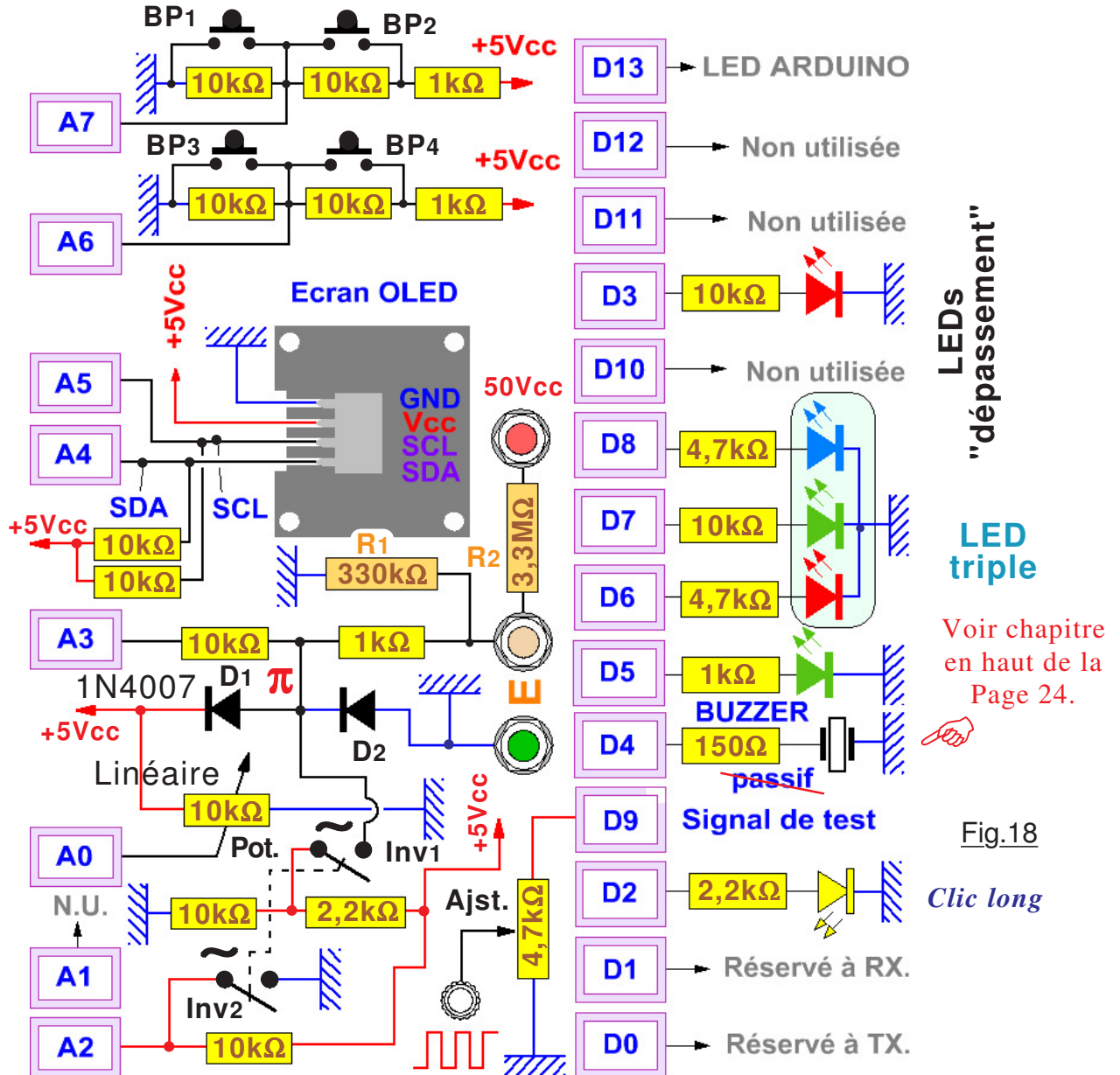


Fig.18

Clic long

B.P. une seule entrée aurait été possible. Comme nous avons des broches de disponible "à revendre", il est préférable de "doubler la mise" car ainsi les seuils de décision sont plus éloignés les uns des autres ce qui favorise à l'élimination "du bruit" parasite. Noter que toutes les résistances de limitation de courant des LEDs sont choisies pour obtenir des éclairagements équivalents, sachant qu'elles n'ont pas des rendements lumineux identiques. (*C'est expérimentalement qu'il faudra adapter chez vous.*)

Considérons l'entrée **E**. C'est elle qui recevra les signaux à analyser pour une plage de 5V crête à crête. Toutefois, on peut désirer mesurer des tensions plus élevées. C'est la raison de la présence des deux résistances **R1** de 330kΩ et de **R2** de 3,3MΩ qui constituent un diviseur potentiométrique par dix. Naturellement il n'est pas obligatoire de prévoir ce calibre. Dans ce cas ces deux composants **R1** et **R2** peuvent être omis.

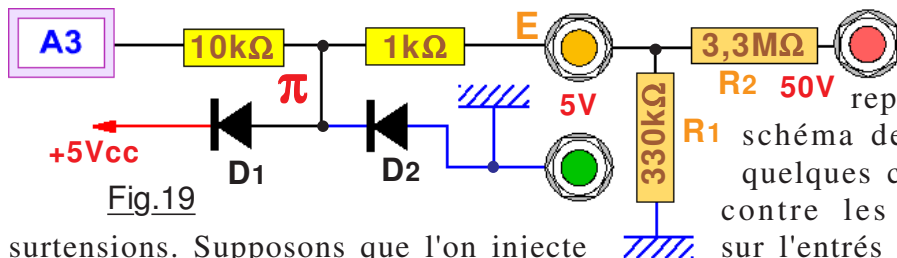


Fig.19

sur tensions. Supposons que l'on injecte une tension négative de -15V. (Ou 15V~ efficace) La diode **D2** est conductrice et court-circuite le signal sur **GND**. La tension en π est alors de -0,7V. L'entrée analogique de l'ATmega328 possède sa propre protection interne. Toutefois la documentation précise qu'il ne faut pas dépasser de $\pm 0,3V$ la tension nulle et celle du +Vcc. C'est la raison pour laquelle une résistance de 10kΩ de limitation de courant supplémentaire est insérée entre π et **A3**. Dans ces conditions c'est la résistance de 1kΩ qui est en danger. En effet, soumise à 15V elle est traversée par un courant de 15mA. Elle est soumise à une puissance de $15 \times 0,015 = 0,225W$. Prévue pour dissiper 1/4W elle peut supporter ce régime en permanence. Si l'on surcharge à $\pm 40V$ il ne faut pas dépasser dix secondes, car ensuite elle surchaufferait et serait en danger car soumise à 1,6W. Sur le calibre 50V on peut en principe aller jusqu'à 800V. L'intensité devient 242μA. **R2** est alors soumise à une puissance dissipée en chaleur de 0,19W qu'elle peut supporter en permanence. (Attention toutefois à la tension d'isolement entre les broches du connecteur HE14 en entrée.) Si la surcharge est positive on retrouve des puissances dissipées équivalentes. La Fig.20 présente un relevé oscilloscopique de la tension présente en π lorsque l'on injecte en entrée **E** une tension alternative de $\pm 15V$ crête auquel l'appareil a été soumis durant plusieurs heures sans subir le moindre dommage. L'index à gauche correspond à 0V. La tension écrêtée par les diodes **D1** et **D2** varie donc entre -0,7V et +5,8V. En résumé :

- Surcharge permanente tolérée sur **E** : $\pm 15V$. (15V~ efficace)
- Surcharge inférieure à 10 secondes sur **E** : $\pm 40V$. (40V~ efficace)
- Impédance d'entrée 330KΩ. (Calibre 5V en entrée **E**.)

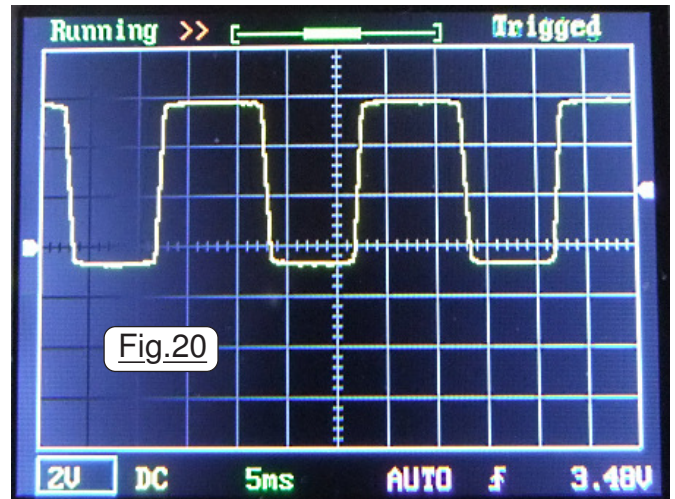


Fig.20

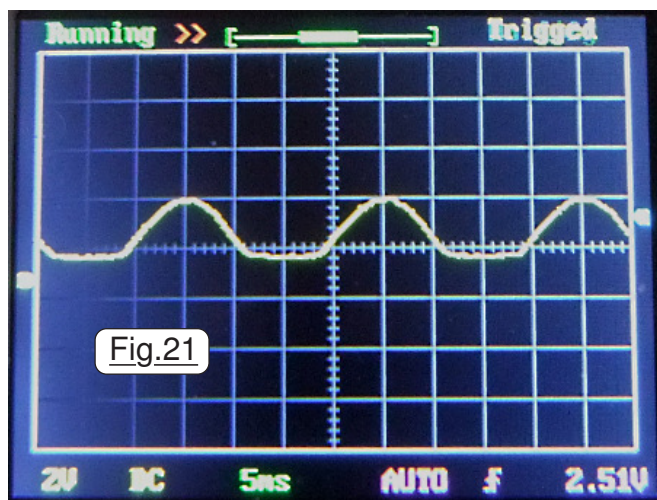


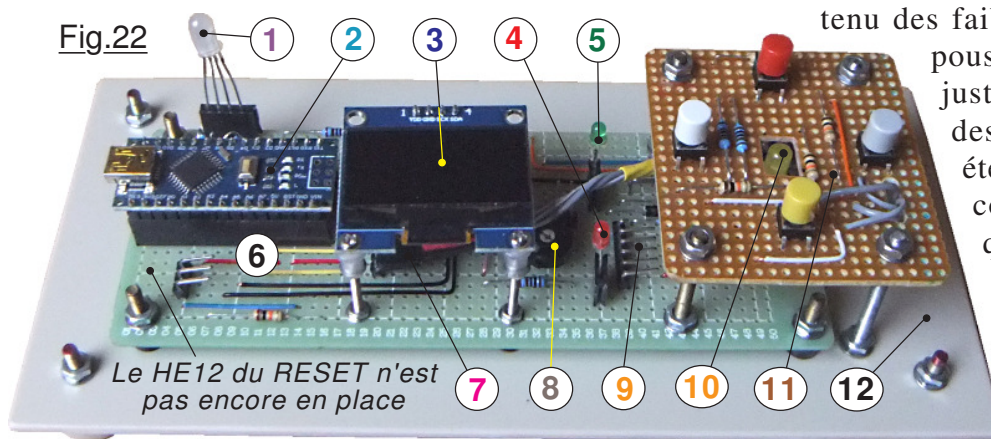
Fig.21

ATTENTION : La diode de protection **D1** n'aura pas vraiment d'influence sur la représentation de la trace car elle "rabote" le signal à partir de +5,8V c'est à dire "au dessus" de +5V qui correspond au calibre d'entrées. La partie déformée sera donc hors de la fenêtre d'écran. Le dépassement du signal vers le haut sera simplement indiqué par l'allumage de la LED rouge de diamètre 3mm. Par contre, la diode **D2** écrête toute tension négative à partir de -0,7V environ. Une tension alternative ne sera donc pas représentée correctement, *sauf si par un module externe on lui superpose "algébriquement" une tension positive*. Pour bien montrer ce phénomène

l'oscillogramme de la Fig.21 a été réalisé en injectant en entrée **E** une tension alternative de 4V crête à crête issue d'un transformateur basse tension branché sur le secteur. On constate effectivement que les alternances positives sont représentées correctement mais les négatives sont tronquées et distordues. La trace sur notre appareil ne représentera pas les tensions négatives et tout débordement par le bas sera signalé par l'éclairement de la LED verte.

07) La concrétisation matérielle.

L'ensemble du schéma électronique est défini sur la Fig.18 et doit conduire à l'étude et à la création de deux circuit imprimés. Considérons le montage provisoire de la Fig.22 pour lequel le coffret ultérieur a été défini avec une bonne précision. De ce fait le support **12** constituera le dessous du boîtier et est déjà muni dans les quatre coins de ses pieds en caoutchouc. Au cours de la réalisation, diverses images commentées sont disponibles dans le dossier joint nommé **<Photographies>**. En particulier **Image 01.JPG** montre la semelle du coffret vu de dessous. Compte



tenu des faibles dimensions des boutons poussoir, le clavier **11** sera situé juste en dessous de la façade du dessus. Le petit écran OLED **3** a été surélevé au maximum par un connecteur à broches longues que l'on peut voir sur la photographie **Image 02.JPG**. Sur **Image 03.JPG** on voit bien comment le petit écran graphique est soutenu par deux boulons ϕ M2 servant

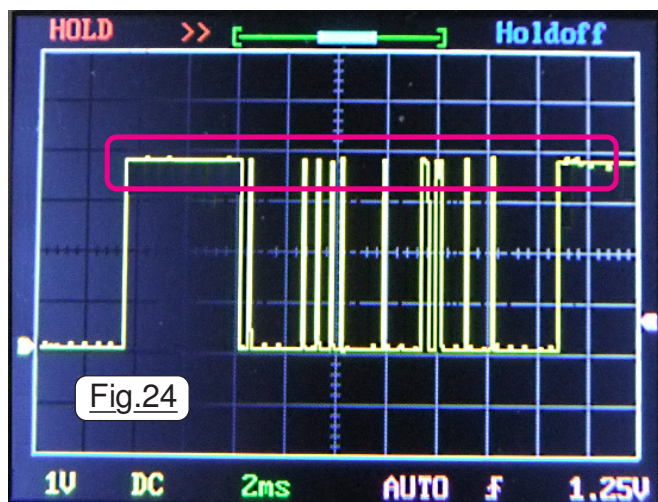
d'entretoises de l'autre côté de son connecteur. Toutes les LEDs **1**, **4**, **5** et **10** sont insérées dans des supports HE14 femelle pour les surélever. La LED tricolore **1** sera assez haute, mais pas les trois autres LEDs dont les broches seront prolongées par des HE14 de type mâle. Comme ce n'est pas encore le cas les deux flèches jaunes symbolisent cette future modification. Comme on peut l'observer sur la Fig.23 le clavier est ajouré en son centre par une lumière à travers laquelle passera la LED jaune **10** de ϕ 5mm qui sera rehaussée à la demande pour présenter en débordement de façade une hauteur identique à celle des quatre touches.

En **2** la carte Arduino NANO est aussi sur deux support NE14 femelle et peut se déposer à la demande pour les

de vérification ou également visible en **7** le connecteur

HE14 sur lequel se branche la ligne du clavier. Noter que tous les connecteurs coudés mâles tels que **6** et **9** sont orientés vers le haut au soudage pour faciliter l'introduction des fiches femelles conjuguées. C'est particulièrement

bien visible sur **Image 04.JPG**. Sur **Image 05.JPG** on comprend facilement comment est soutenu le clavier **11** par de la visserie ϕ M3. On a vu sur la Fig.8 en Page 4 qu'en fonction du fournisseur, les deux broches d'alimentation **GND** et **+Vcc** peuvent être inversées. Pour s'accommoder de l'importe quelle référence et vous éviter tout problème, le circuit imprimé principal est muni de deux connecteurs HE14 mâles verticaux sur lesquels on pourra librement choisir la polarité d'alimentation du petit écran en disposant convenablement deux "Straps". Ces éléments sont bien visibles sur la photographie **Image 06.JPG**.



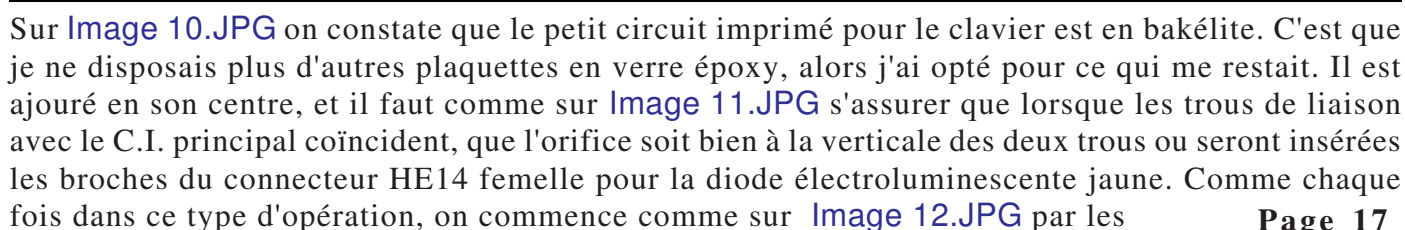
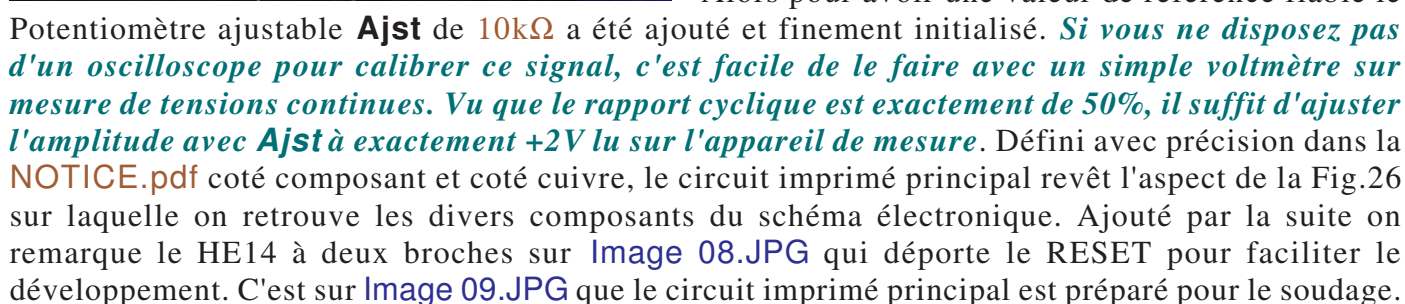
opérations de tests, de maintenance. Peu

HE14 sur lequel se branche la ligne du clavier. Noter que tous les connecteurs coudés mâles tels que **6** et **9** sont orientés vers le haut au soudage pour faciliter l'introduction des fiches femelles conjuguées. C'est particulièrement

bien visible sur **Image 04.JPG**. Sur **Image 05.JPG** on comprend facilement comment est soutenu le clavier **11** par de la visserie ϕ M3. On a vu sur la Fig.8 en Page 4 qu'en fonction du fournisseur, les deux broches d'alimentation **GND** et **+Vcc** peuvent être inversées. Pour s'accommoder de l'importe quelle référence et vous éviter tout problème, le circuit imprimé principal est muni de deux connecteurs HE14 mâles verticaux sur lesquels on pourra librement choisir la polarité d'alimentation du petit écran en disposant convenablement deux "Straps". Ces éléments sont bien visibles sur la photographie **Image 06.JPG**.

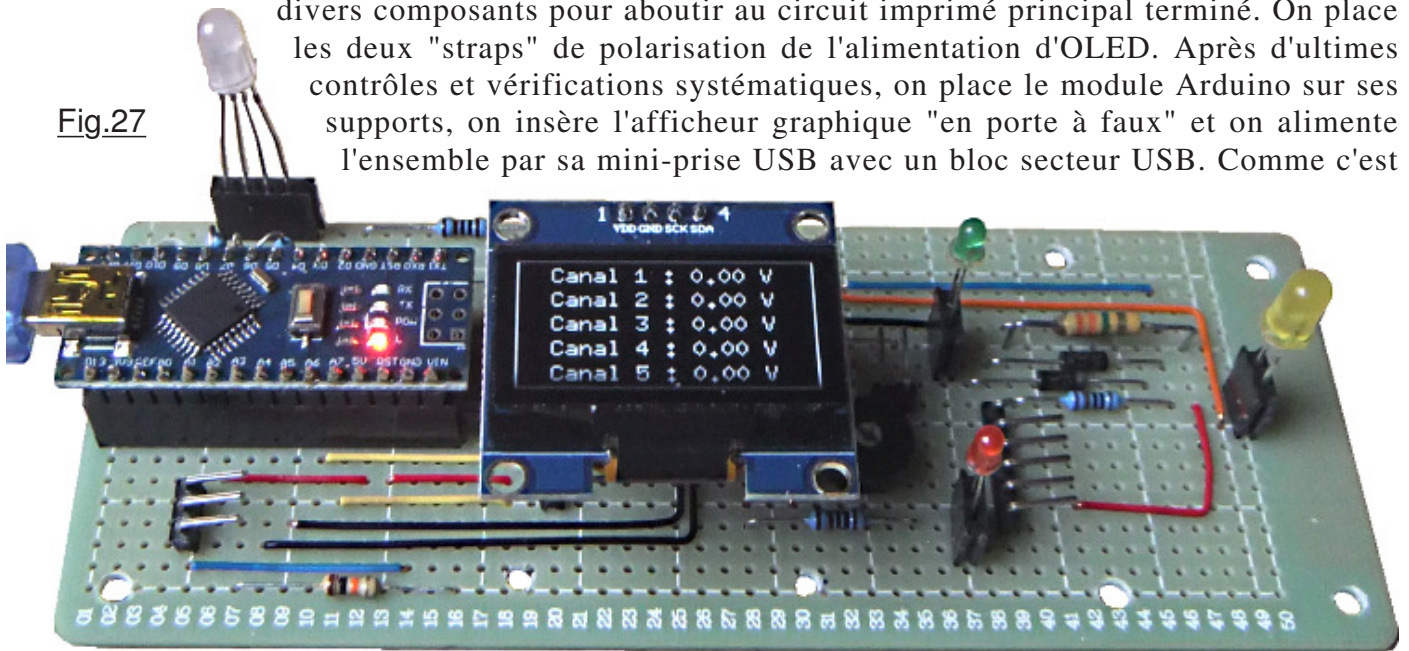
On documenté en Fig.18, on peut y observer que **SDA** et **SCL** sont reliées au **+5Vcc** par des résistances de **10k Ω** . Intrinsèquement une ligne de **protocole I2C** peut dialoguer avec plusieurs périphériques. Pour "tirer" vers le "haut" chaque fil **SDA** et **SCL** doivent être reliés au **+5Vcc** par une résistance. Pour éviter que tous les périphériques amènent leur polarisation qui ajoutée aux autre diminuerait trop l'impédance de la ligne, donc l'intensité à drainer vers **GND**,

CONCLUSION : Sur le circuit imprimé les deux résistances de 10kΩ sont prévues. *Dans un premier temps pas la peine de les souder.* Ce n'est que si les démonstrateurs se bloquent qu'il faudra soupçonner ce détail et insérer les deux composants pour compléter le circuit électronique.



composants "les moins hauts". On remarque au passage que la résistance de $3,3M\Omega$ est un peu surélevée pour augmenter son isolement par rapport aux autres composants, car elle est susceptible d'être soumise à des tensions élevées. C'est un composant de 1/2W pouvant supporter entre broches des tensions très élevées sans risquer d'amorcer. Stratégie identique pour le clavier en [Image 13.JPG](#). Présenté en [Image 14.JPG](#) et sur [Image 15.JPG](#) le clavier est achevé. Soudés sur le dessus les fils de liaison passent en dessous par la lumière oblongue pour protéger mécaniquement cette ligne torsadée. (*Torsadée pour augmenter la souplesse du toron.*) Noter sur [Image 16.JPG](#) le repérage au feutre à alcool indélébile le coloriage des lignes +Vcc et GND au fur et à mesure du soudage. C'est facile à faire et ça facilite grandement le travail. Remarquer également que les lignes sur le dessous qui se croisent sont en fil isolé. C'est le cas pour le déport du RESET qui n'est pas encore câblé sur cette photographie. Montrant une foule de détails, d'[Image 17.JPG](#) à [Image 20.JPG](#) on soude les divers composants pour aboutir au circuit imprimé principal terminé. On place les deux "straps" de polarisation de l'alimentation d'OLED. Après d'ultimes contrôles et vérifications systématiques, on place le module Arduino sur ses supports, on insère l'afficheur graphique "en porte à faux" et on alimente l'ensemble par sa mini-prise USB avec un bloc secteur USB. Comme c'est

Fig.27



le démonstrateur **P03** qui est présent dans la mémoire de programme d'Arduino, immédiatement l'afficheur doit présenter la salamandre puis comme sur la Fig.27 les tensions sur les cinq canaux qui toutes sont nulles. OUF, quand OLED affiche normalement, le plus difficile est vaincu, il faut maintenant valider toutes les autres entrées et sorties de l'ATmega328.

08) La vérification matérielle.

Permettre un test complet de l'ensemble du matériel qui sera utilisé est un impératif suggéré en page 13 avant de développer le programme d'application. On peut aussi en profiter pour préparer l'avenir et intégrer des routines qui seront utiles plus tard. Il est également avantageux de pouvoir évaluer certains paramètres qui seront indispensables pour créer les autres démonstrateurs. C'est exactement ce que fait **P05_Test_materiel.ino** en introduisant la routine d'exploitation du petit clavier et en mesurant certaines valeurs critiques. Toutefois, avant de téléverser le programme de

Optimisation de la taille des variables.

Dans le cadre de la **programmation méthodique**, nous avons vu en Page 9 qu'il est très conseillé lors de l'écriture du programme d'ordonner les variable par type, par taille et par ordre alphabétique. Par ailleurs, **il est absolument indispensable de choisir le type de chaque variable pour en minimiser la place occupée en mémoire dynamique et en accélérer le traitement.** C'est un impératif absolu. **Du bon choix du type des données dépend considérablement la taille occupée par le programme et le temps d'exécution.** ATTENTION, les tableaux sont les données les plus voraces en espace utilisé **et les textes sont intrinsèquement des tableaux.** Aussi, **si la mémoire non volatile EEPROM n'est pas utilisée pour mémoriser des données** à conserver sur coupure alimentation, **y logger un maximum de textes** affichés. Enfin, **pour une lisibilité maximale du listage, toujours choisir avec beaucoup d'attention les noms des identificateurs des constantes et des variables** avec des noms qui "parlent".

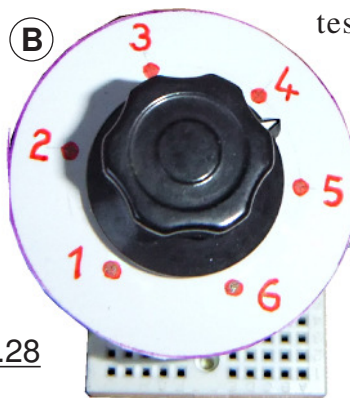
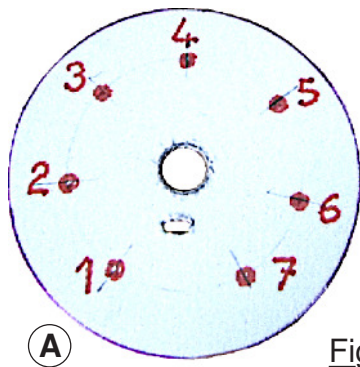


Fig.28

test et d'en décrire l'usage et le fonctionnement, il n'est pas raisonnable de laisser OLED en "porte à faux". C'est la raison pour laquelle on prépare la semelle d'[Image 01.JPG](#) sur laquelle on immobilise les deux circuits imprimés ainsi que le petit écran sur la plaquette principale. On peut voir sur la Fig.28 ainsi que sur [Image 21.JPG](#) la présence provisoire d'un potentiomètre de $10k\Omega$ inséré sur une plaquette expérimentale 8. Observons la Fig.29

qui présente en coupe médiane le petit module potentiométrique. On a découpé dans du carton rigide "culinaire" deux disques 2. Celui en B de la Fig.28 est prévu pour six positions. (Voir en page 12 le cahier des charges.) Il n'est pas interdit que le besoin fasse émerger la nécessité de sept indexages raison pour laquelle un deuxième disque "gradué" est disponible en A. Si l'on regarde la Fig.30 qui correspond



Fig.30

au potentiomètre commandé qui sera sur la façade, on constate que les broches à souder étant orientées vers la droite comme prévu pour l'insertion dans le coffret, l'ergot 3 anti-rotation E est alors vers nous. C'est la raison pour laquelle sur les disques "gradués" A et B il est vers le bas par rapport aux index. Le potentiomètre est soudé perpendiculairement au circuit imprimé 4 qui s'insère par les picots

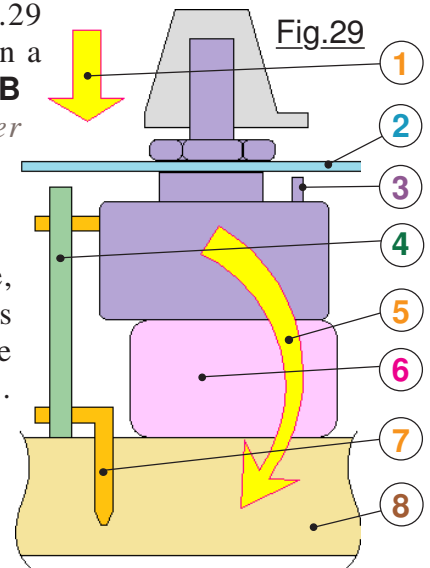


Fig.29

coudés 7 sur la plaquette 8. Du coup il est en porte à faux et toute pression verticale en 1 s'accompagne d'une torsion inévitable 5. À ce régime les picots de sortie du potentiomètre ne résisteront pas longtemps. Pour les soulager un petit bloc de mousse synthétique 6 est calé entre le corps du potentiomètre et la plaquette expérimentale 8. Ce bloc souple se distingue très bien sur [Image 21.JPG](#). Le décor est en place, on va pouvoir tester le matériel en toute sérénité et analyser [P05_Test_materiel.ino](#).

➤ La vérification des LEDs.

À ce stade, la configuration d'[Image 21.JPG](#) est en place et l'entrée E est reliée à la sortie du Signal étalon sur D9. Conformément au protocole de la Page 17 le signal de référence a été ajusté à exactement 4V crête. À la mise sous tension ou sur un RESET l'écran OLED affiche la page de la Fig.31 avec sur la ligne du haut la version du démonstrateur P05. Étant donné que l'entrée E est soumise au Signal étalon la valeur en ligne 2 alterne régulièrement entre 0.00 et environ 4.00 volts la valeur crête. Comme nous n'avons encore cliqué sur aucune touche du clavier, les données le concernant en 3, 4 et 5 sont "neutres". En ligne 1 la valeur actuelle retournée par le CAN de l'entrée A0 fait ici 215 car le Potentiomètre est indexé sur la position n°2. (Analyse plus

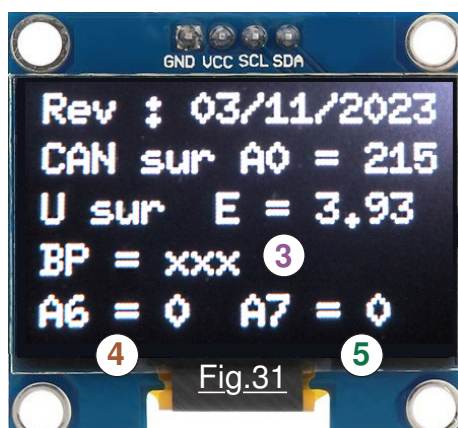


Fig.31

avant.) Tant que l'on n'agit pas sur le clavier, la LED jaune reste éteinte. Si le circuit imprimé principal a été correctement câblé, sans rien avoir à faire, en tâche de fond void loop allume tour à tour tous les témoins lumineux durant exactement une seconde, puis extinction de tout ce petit monde durant également une seconde. C'est la LED triple qui s'active, avec dans l'ordre le rouge, le vert et le bleu. Puis c'est le tour des deux petites LEDs de 3mm de diamètre qui s'éclairent, la rouge en premier puis la verte. Notons au passage que les délais de pause et d'illumination sont longs, puisque d'une seconde. Et pourtant ils ne ralentissent pas la boucle de base. Une explication s'impose.

➤ Des chronomètres à gogo.

Chaque usage de la fonction **millis** permet de créer un chronomètre indépendant qui ne bloque pas le déroulement du programme dans un **delay()** qui immobiliserait le microcontrôleur, et on peut en créer à volonté. Toutefois, chaque chronomètre devra se servir d'un "Top chrono" qui lui est propre. La Fig.32 présente un exemple avec deux chronomètres indépendants. Lorsque le microcontrôleur est mis sous tension ou lors d'un RESET, un compteur interne démarre à zéro et se voit incrémenté mille fois par seconde. (Il contient la durée écoulée en mS.) À tout moment **millis** peut en lire la valeur et la retourner sous forme d'un **unsigned long** en sortie de cette fonction. Pour mesurer l'intervalle de temps **T1** la valeur retournée par **millis** est enregistrée dans **Top_1**. Puis dans le programme le chronomètre **A** va lire en permanence la valeur de **millis**. Dès que la valeur de **[millis - Top_1]** sera égale ou plus grande que la valeur désirée **T1** c'est que la durée calibrée sera atteinte et l'on déclenchera l'action pour **A**, puis immédiatement **Top_1** sera rechargé avec **millis** pour démarrer un nouveau chronométrage. Pour réaliser un deuxième chronomètre **B** afin de mesurer une temporisation **T2** il suffit de dupliquer cette séquence, avec comme origine de chronométrage une mémoire **TOP_2** et effectuer la comparaison **[millis - Top_2]**. On peut ainsi en créer autant que nécessaire. Ces séquences peuvent se situer n'importe où dans le programme avec des durées **T1** et **T2** quelconques. *Noter que la durée de la temporisation ne sera pas strictement égale à celle calibrée, car il faut tenir compte du fait que l'écart de temps étant arrivé, le programme peut être occupé ailleurs.* (Par exemple tant qu'un B.P. est cliqué ...)

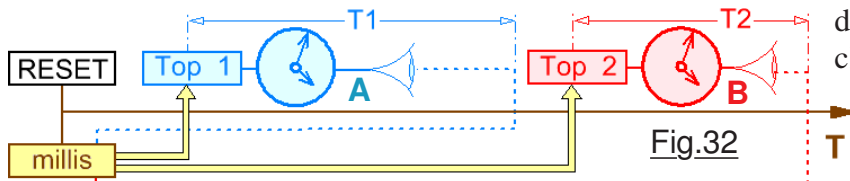


Fig.32

que la durée calibrée sera atteinte et l'on déclenchera l'action pour **A**, puis immédiatement **Top_1** sera rechargé avec **millis** pour démarrer un nouveau chronométrage. Pour réaliser un deuxième chronomètre **B** afin de mesurer une temporisation **T2** il suffit de dupliquer cette séquence, avec comme origine de chronométrage une mémoire **TOP_2** et effectuer la comparaison **[millis - Top_2]**. On peut ainsi en créer autant que nécessaire. Ces séquences peuvent se situer n'importe où dans le programme avec des durées **T1** et **T2** quelconques. *Noter que la durée de la temporisation ne sera pas strictement égale à celle calibrée, car il faut tenir compte du fait que l'écart de temps étant arrivé, le programme peut être occupé ailleurs.* (Par exemple tant qu'un B.P. est cliqué ...)

➤ Analyse du chronométrage pour les LEDs.

Exemple typique d'un chronométrage sans pénalité pour la rapidité du logiciel, la Fig.33 présente le listage de la séquence qui gère les LEDs dans le démonstrateur **P05** et qui est invoquée à chaque cycle de la boucle de base infinie **void Loop**. C'est dans la zone **1** colorée en rose pastel que sont effectués les tests pour le chronométrage. Cette comparaison est très rapide et la sortie est immédiate tant que la durée attendue n'est pas atteinte. Puis, dès que l'on dépasse la temporisation consignée dans la valeur mise en évidence en jaune, on déclenche toutes les actions de la zone vert

```
void Traiter_une_fois_par_seconde() { // Le délai est de une seconde.
//===== Chronométrage pour une seconde =====
1  if (millis() - Ancien_Temps_Ecoule_depuis_RESET > 1000) { Début
2  Ancien_Temps_Ecoule_depuis_RESET = millis(); // réaeme pour une seconde.
//----- Actions à traiter pour chaque seconde écoulée. -----
    Num_LED++; if (Num_LED == 10) Num_LED = 0;
    switch(Num_LED) { case 1 : {digitalWrite(Trp_R,HIGH);break;}
                      case 3 : {digitalWrite(Trp_V,HIGH);break;}
                      case 5 : {digitalWrite(Trp_B,HIGH);break;}
                      case 7 : {digitalWrite(Rouge,HIGH);break;}
                      case 9 : digitalWrite(Verte,HIGH);}
    Allumer = !Allumer; if (!Allumer) {EteindreLED_Toutes_les_LEDs;} }
//===== Fin ===== }
```

Fig.33

pastel **3**. La première action urgente pour obtenir un chronométrage précis, c'est en **2** le **Top_1** de la Fig.32 sauf que dans notre cas il n'y a pas besoin de mémoriser cette valeur puisqu'il n'y a qu'un seul chronométrage à gérer. Généralement les actions **3** n'exigent que peu de cycles microcontrôleur et ensuite la boucle de base reprend sa routine de fond généralement à un rythme élevé.

NOTE : J'ai oublié de vous préciser que dans mes listages, les mots du langage C++ sont en marron, les identificateurs sont en vert et souvent les valeurs particulières sont en violet.

RÉSUMÉ : Pour réaliser un chronométrage on peut à convenance :

- Utiliser l'instruction **delay()** qui "fige" le microcontrôleur durant tout le chronométrage,
- Utiliser la fonction **millis** qui permet de créer des chronomètres indépendants,
- De créer un chronométrage géré par le programmeur au moyen d'un simple compteur.

(On peut aussi faire par "interruptions", mais c'est un autre sujet !)

➤ Rapidité de "rotation" de la boucle de base.

Dans les nombreux conseils prodigués péremptoirement dans ce didacticiel, il est précisé que l'on doit mesurer la rapidité d'exécution `void loop()` pour vérifier que le taux d'affichage est important, que le délai de prise en compte des potentiomètres, B.P. et autres périphériques de type clavier est faible. Bref, que le *dispositif est réactif*. Seule limite, il faut disposer d'un fréquencemètre. Pour ceux qui le désirent, sur

<https://www.robot-maker.com/ouvrages/apprendre-a-programmer-arduino-en-samusant/> est décrit ce type d'appareil réalisé avec une carte Arduino.

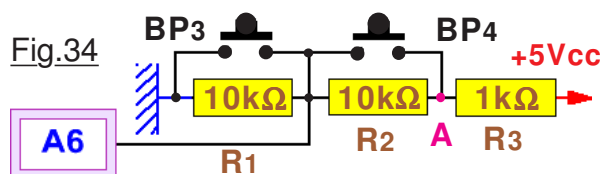
On obtient pour le démonstrateur **P05** une fréquence de 7Hz. La prise en compte du bouton poussoir est donc vive sans être toutefois exceptionnelle. En effet, elle est ralentie considérablement par la boucle d'affichage sur le petit écran OLED car toutes les autres séquences sont très rapides.

NOTE IMPORTANTE : Chaque fois que c'est possible dans un programme, je réserve la sortie **D13** pour gérer la LED Arduino dans la boucle de base. C'est généralement cette boucle de base qui "tourne" en tâche de fond et qui s'occupe du MENU de base du programme d'exploitation. Il importe que `void loop()` soit la plus rapide possible pour effectuer la prise en compte des divers claviers et codeurs rotatifs du système. Aussi il faut l'optimiser et en vérifier sa cadence. *Mais le clignotement rapide de la LED Arduino atteste également du fonctionnement de void loop(). Aussi, si cette LED ne clignote plus, c'est que le programme s'est enlisé dans une séquence très longue, ou pire, dans une boucle infinie dont il ne sortira plus.*

➤ La gestion d'un bouton poussoir et l'antiparasitage.

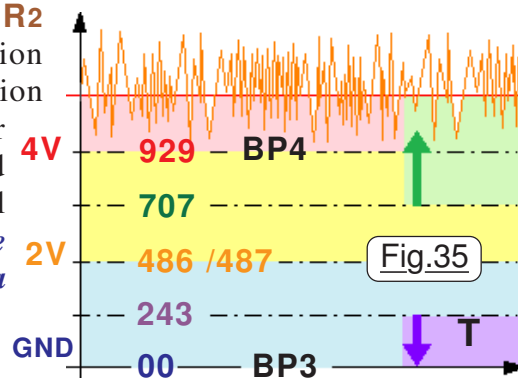
Dans ce chapitre nous allons mêler théorie et pratique et prendre en exemple la ligne clavier gérée par **A6**. Pour mémoire la Fig.34 reproduit son schéma électrique. Bien que dans ce démonstrateur on ne va prendre en compte que quatre boutons poussoir, ce thème reste directement utilisable pour un clavier plus conséquent, voir multiplexé. La première source de parasitage dans la lecture d'une entrée analogique vient avec le "bruit alimentation". Considérons la Fig.35 qui montre

Fig.34



l'évolution de la tension alimentation au cours de temps. En théorie l'adaptateur secteur fournit un +5V bien filtré et continu représenté par le trait rouge horizontal. Dans la réalité, à cette tension continue se superposent des variations très rapides que les techniciens nomment "herbe" ou "bruit blanc". Sur la Fig.35 cette perturbation est considérablement exagérée. (Ou alors l'adaptateur secteur doit être mis à la poubelle !) Même si l'alimentation est parfaite, le +5V du bloc USB fournit la polarisation par la ligne électrique dans laquelle se trouve la résistance **R3** de protection de 1kΩ. Elle est indispensable, car sans elle si l'opérateur clique simultanément sur les deux boutons **BP3** et **BP4** on réalise un court-circuit franc entre le +5V et le **GND**. Cette ligne dans un environnement électromagnétique très encombré capte les parasites hertziens et sera forcément affectée par moment de petites impulsions parasites. Plus la résistance sera de valeur faible, moins ces phénomènes seront présents. À vide, **R1** et **R2** forment un diviseur de tension par deux. Comme en **A** la tension à vide est de l'ordre de 4V, sur **A6** elle est de 2V et la numérisation par le **CAN** donne la valeur de 486. Cliquer sur **BP3** a pour effet de faire un court-circuit entre **A6** et **GND**. Donc quand on cliquera sur le bouton poussoir, **A6** se retrouvera au potentiel nul. *Pour minimiser l'influence des parasites captés par la ligne de polarisation, il suffit de prendre comme seuil de décision la moitié de la tension présente entre celle du repos et celle de l'enfoncement de la touche concernée.* Ce qui compte pour le programme, ce n'est pas la tension mesurée, mais le résultat

de la numérisation du **CAN**. Les valeurs seront directement concernées par celles des résistances **R1**, **R2** et **R3** qui ont une tolérance de fabrication. (Probablement dans les 5%.) Du coup sur votre prototype elles ne seront pas forcément les mêmes. Pour optimiser le code, **P05** est



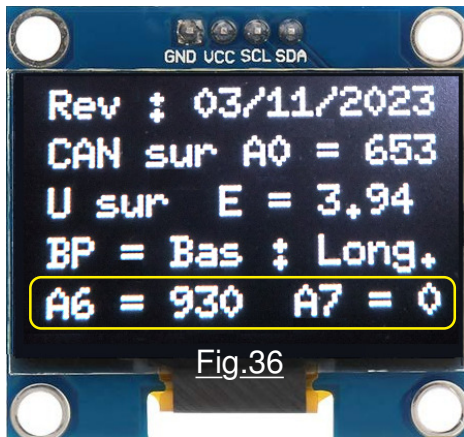
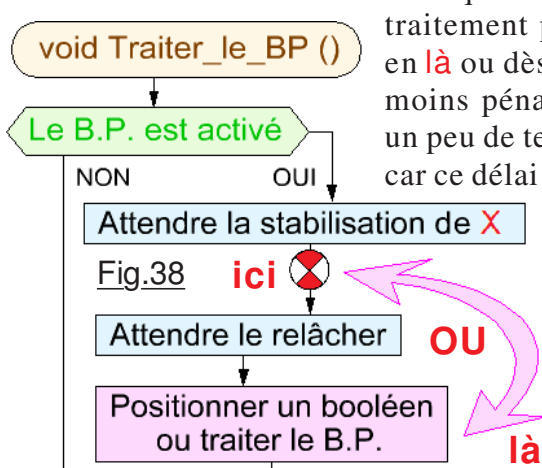
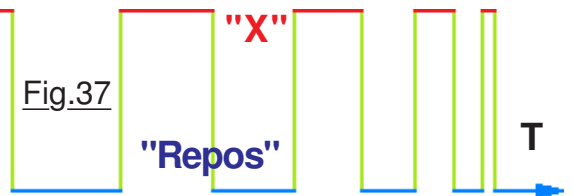


Fig.36

conçu pour mesurer ces valeurs numérisées et les afficher. Cliquer un *court instant* sur le B.P. du haut puis *deux secondes sur celui du bas*. Sur la ligne "CAN sur A0" la valeur de 653 résulte de l'orientation de **Pot** sur l'index 4. Sur la ligne de l'encadré jaune on trouve les valeurs des numérisations effectuées lors du dernier clic réalisé sur les deux lignes du clavier. Donc A6 = 930 est relatif au B.P. du haut, soit BP4, alors que A7 = 0 concerne BP3. Juste au dessus du cadre, l'information du type BP = Bas : Long concernera toujours les attributs de la dernière touche cliquée. Il est donc facile d'obtenir les quatre valeurs des **CAN** pour toutes les touches activées. Notez au passage que lorsque l'on clique sur le B.P. du haut, on génère un BIP d'alerte. On teste donc au

passage une routine qui sera utile aux futurs programmes. Sur la Fig.35 ont été reportées les valeurs numérisées sur le prototype. *Le seuil de décision situé entre les données limites* est également porté en *vert* et en *violet*. Si vous consultez le listage de la procédure `void Tester_le_BP()`, ce sont ces valeurs qui sont consignées. Toute valeur plus grande que 707 désignera BP4, toute valeur inférieure à 243 dénoncera BP3. Il reste à déterminer les valeurs des **CAN** au repos des deux lignes A6 et A7. Dans ce but il suffit dans le logiciel de P05 de passer en remarques les deux lignes repérées par des @@@@@@@@@@@@@@@@@@ et de valider les deux lignes du dessous. On a réglé le problème de "l'herbe" mais il reste à éliminer les rebonds du contact électrique. Quand on bascule la mécanique d'un inverseur ou celle d'un bouton poussoir, on engendre le mouvement de pièces métalliques qui viennent en contact. Électriquement on génère un état "X" sur une entrée analogique ou binaire. Et bien cet état est inutilisable directement. En effet, *lorsque deux pièces mécaniques sont "bousculées" l'une contre l'autre, il y a des rebonds mécaniques* et l'évolution de l'état logique sur l'entrée concernée ressemble à ce à celle de la Fig.37 qui correspond à un élément de très bonne qualité, car ici le contact ne rebondit que quatre fois lors de l'activation. Les transitoires verts sont représentés par des "durées nulles" car ils se font très rapidement. *Si on ne tient pas compte de ce phénomène, alors le logiciel va croire que l'action sur le composant a été déclenchée quatre fois* et les traiter en conséquence. Dans la pratique, un interrupteur de qualité va présenter entre dix et vingt "ricochets". Un composant médiocre peut en générer jusqu'à deux cents voir plus ! Aussi, il faut systématiquement faire de l'*anti-rebonds*. Parmi les techniques ordinaires, on peut adopter celle de la Fig.38 dans laquelle le



traitement peut être effectué juste après le relâcher du bouton poussoir en *là* ou dès que le clic est stabilisé en *ici*. L'approche "ici" est en général moins pénalisante en taille de programme, et peut surtout faire gagner un peu de temps en exécution si le composant met du temps à se stabiliser, car ce délai sera en partie "masqué" par la durée du traitement. La stratégie "là" impose la gestion d'un booléen mais autorise l'appel multiple à la séquence de surveillance d'une utilisation du clavier. Cet organigramme n'explique pas comment se fait l'anti-rebonds. En analysant le code source, on constate que l'on utilise un **COMPTEUR**. Chaque fois qu'une mesure signale un état "Repos" il est incrémenté. Si durant le comptage on constate un état "X", c'est qu'il y a rebond et le **COMPTEUR** est remis à zéro. Il faut atteindre un nombre

de lecture stables NB_tests_antirebonds fois pour considérer que le contact électrique est stabilisé et que le relâcher est stabilisé. On ne se préoccupe pas de la stabilisation de l'état travail, puisque seule celle du relâcher est importante. La valeur de NB_tests_antirebonds est trouvée expérimentalement. Plus le composant est médiocre, plus il faut l'augmenter au détriment naturellement du temps pris par le traitement des "ricochets". On peut "juguler" un peu ces faux contacts en plaçant un condensateur entre l'entrée logique et GND mais je ne le fais que dans des cas particuliers car ce sont des composants en plus à ajouter à la circuiterie électronique.

► Utilisation de la PWM ou de l'instruction tone ?

Lors du chapitre précédent on a modifié deux lignes pour tester les **CAN** au repos des touches du clavier. On a repéré immédiatement les longues lignes de remarques du genre `//@@@@@@@@@@@@@@@@`. C'est un artifice pour attirer immédiatement le regard dans un listage. J'utilise systématiquement cette sorte d'étiquette dans mes programmes pour des *lignes de code dont on est amené à changer "régulièrement" le contenu*. Par exemple en tête de programme on trouve un texte qui précise la date de validation du démonstrateur. Si plus tard on le modifie, pour mettre à jour cette référence il faut la retrouver facilement. Dans **P05** on utilise la PWM pour générer le signal étalon. C'est la raison pour laquelle sa *fréquence de répétition* est de **490 Hz** avec une *période* $T \approx 2042 \mu S$ pour PWM **D3**, **D9**, **D10** et **D11**. Pour les sorties PWM **D5** et **D6** la *fréquence de répétition serait de 976 Hz* avec une *période* $T \approx 1025 \mu S$. (*Pulse Width Modulation*) Dans une instruction telle que `analogWrite(Sortie_PWM, N)` le *rapport cyclique* est directement proportionnel à la valeur **N** passée en paramètre et `Sortie_PWM` fait référence à la broche de sortie utilisée. **N** impose la durée à l'état "1", donc influence directement le rapport cyclique.

- Quand **N** = 0 la sortie reste en permanence à 0 V.
- Quand **N** = 128 le rapport cyclique est de 50%.
- Quand **N** = 255 la sortie reste constamment à 5 VCC.

Lorsque l'on déclenche une PWM, elle continue "définitivement" sauf si par la suite on utilise `analogWrite(Sortie_PWM, N)` avec **N** = 0 ou **N** = 255 sur la broche concernée. C'est de cette façon qu'avec la seule instruction `analogWrite(Signal_Etalon, 128)` dans `void setup()` on génère le signal de test pour notre oscilloscope sur la broche **D9** de l'ATmega328.

Étant limité à deux tonalités au maximum, il est naturel de chercher à enrichir les possibilités musicales du bruiteur installé. C'est précisément la finalité de l'instruction **tone** dont la mise en œuvre est particulièrement conviviale. Pour s'en rendre facilement compte, dans la procédure `void Gere_stabilisation_et_duree()` on utilise la **PWM** pour générer avec `tone(BUZZER, 200)` la tonalité grave sur **D4** durant l'enfoncement d'une touche. Dès que son relâchement est stabilisé, avec l'instruction `noTone(BUZZER)` on stoppe le bruitage. Cette instruction laisse la broche au potentiel de **GND** le courant étant alors nul. Pour le **BIP()** sonore, la fréquence générée de 4000Hz a été choisie car elle n'est ni trop grave ni trop aigüe, toutefois amusez-vous à proposer d'autres tonalités. On ne peut rêver plus simple pour "faire de la musique". *Toutefois, la taille du programme augmente de 1312 Octets d'un coup et 23 Octets de plus dans les données dynamiques.*

Conclusion : Générer une tonalité musicale peut se faire de plusieurs façons. Utiliser de la **PWM** est parfait pour gérer la luminosité d'une LED par exemple en modifiant le rapport cyclique. Par contre, pour créer une note musicale on est limité à une seule fréquence relativement basse. Si on désire n'utiliser qu'un seul BIP sonore, on peut se contenter d'un **BUZZER actif** et d'une procédure élémentaire telle que : (*Un BUZZER actif génère du 3000 Hz quand il est soumis à +5Vcc*)

```
void BIP() {digitalWrite(BUZZER, HIGH); Delay(100); digitalWrite(BUZZER, LOW);}
```

Si le bruiteur est passif comme dans notre application, on peut créer notre propre procédure :

```
void BIP () {  
    for(int N = 0; N < MAX; N++)  
        {digitalWrite(BUZZER, HIGH); delay(T); digitalWrite(BUZZER, LOW); delay(T);}
```

Fig.39

Dans cette procédure qui réalise un BIP sonore sur un bruiteur passif la fréquence sera d'autant plus élevée que **T** est de faible valeur. La durée du BIP est directement proportionnelle à **MAX**. Cette procédure ne consomme que 52 Octets de programme et rien de plus dans la mémoire dynamique.

Résumé : Pour générer des tonalités par un bruiteur il existe un nombre infini de possibilités qui n'est limité que par notre manque d'imagination. Si l'on désire se faire plaisir est obtenir facilement des tonalités différentes, l'emploi d'un BUZZER passif s'impose. Si de plus on dispose de largement trop de place en mémoire de programme pour y loger notre code, ne pas hésiter à privilégier **tone** et **notone**. Si au contraire on cherche désespérément à loger le code objet qui dépasse l'espace disponible, passer éventuellement à un BUZZER actif et se contenter d'une seule tonalité, d'environ 3000Hz qui est celle de ce type de composant. Reste à aborder la caractéristique du bruiteur adopté initialement sur le prototype car on va en changer pour le remplacer par un actif.

ATTENTION : Ces éléments, qu'ils soient actifs ou passifs sont polarisés et la Fig.40 en broche **B** précise la sortie positive, l'autre devant aller sur **GND**. Corrigé sur le schéma de la Fig.18 en page P14 le dispositif initial était branché directement sur la sortie **D4**. Lorsque l'on observe l'oscillogramme de la Fig.41 on constate que le niveau logique "1" talonne à 1,5V avec des petits "pics" qui montent jusqu'à 4,4V et des impulsions négatives jusqu'à -0,7V. Ce constat étant réalisé, j'ai mesuré la résistance du composant : **11Ω** ! Suite à un démontage du BUZZER initial, son étude montre que c'est un simple bobinage avec un noyau central et un aimant permanent annulaire autour. C'est "le sens" de l'aimant qui impose une polarisation. Du coup les **11Ω** font presque un court-circuit sur la sortie **D4** et à ce régime l'ATmega328 ne va pas supporter très longtemps. Il serait possible de persister dans cette voie, en intercalant un petit transistor amplificateur de courant. Outre qu'il ne reste pas assez de place sur le circuit imprimé, des essais ont montré que ce n'est pas vraiment la solution. *Aussi, avec le prochain démonstrateur on va radicalement changer de solution et utiliser un bruiteur ACTIF.* Ce dernier présente une impédance très élevée et ne consomme pratiquement rien. Toutefois, une résistance de **150Ω** est insérée en série, car il est un peu trop bruyant. C'est la raison pour laquelle le schéma de la Fig.18 a le mot **passif** rayé. C'est

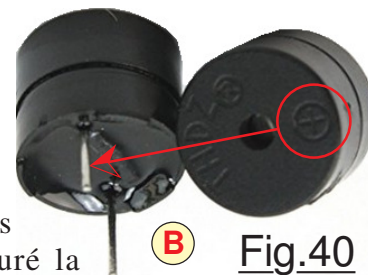


Fig.40

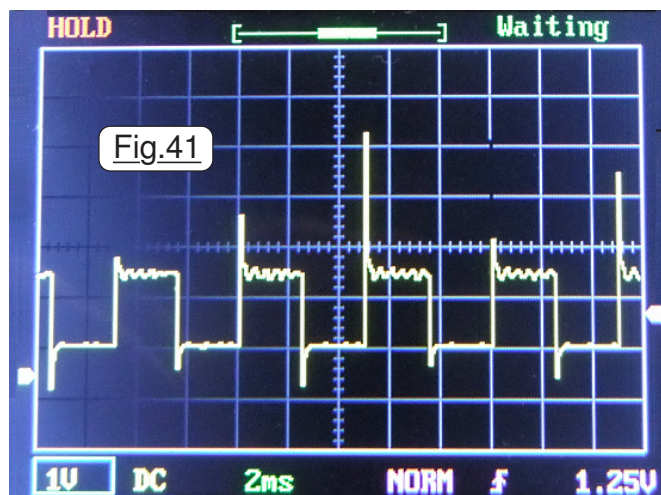


Fig.41

avec le prochain démonstrateur **P07_Gestion_MENUs.ino** que sera mis en service ce composant qui a été changé sur le circuit imprimé, les dessins étant mis à jour sur la Fig.26 et dans la notice.

Donner dans la facilité n'est jamais une bonne chose. Aussi, il suffit de réécrire **void BIP()** et de changer quelques instructions dans **void Gere_stabilisation_et_duree()** pour passer de **P05** au démonstrateur **P06_Test_optimal.ino** qui effectue strictement le même travail avec toutefois une réduction de 1312 octets pour la taille du programme ce qui est considérable et une diminution de 23 octets dans la mémoire dynamique. (*tone est simple à utiliser, mais gloutonne 1312 octets !*)

STRATÉGIE ADOPTÉE : Il suffit d'abandonner la facilité apportée par **tone** et utiliser PWM pour générer la tonalité grave qui signale l'activation d'une touche du clavier. Quand à **void BIP()** elle est entièrement construite pour générer la fréquence de 4000Hz dans le BUZZER. La période est de 250μs d'où l'utilisation de **delayMicroseconds(125)** pour temporiser de la moitié qui dans la Fig.39 correspond à **T**. Comme l'on désire une durée du BIP de 0.15 seconde, le nombre de périodes **MAX** est de $0.15 / 0.00025 = 600$ valeur adoptée dans la séquence sonore.

CONCLUSION : Souvent, quand on programme sous la houlette d'un langage évolué comme le C++ l'approche la plus naturelle consiste à faire usage de toutes ses instructions et en particulier des plus évoluées. Ce n'est pas forcément la bonne approche au point de vue de l'optimisation du code. On en a un exemple typique avec **tone**. Aussi, le programmeur devrait toujours envisager d'autres écritures qui permettent parfois une économie en octets substantielle. À méditer ...

➤ **Préparer les menus par gestion du Potentiomètre branché sur A0.**

Fig.42

N°	CAN
1	0
2	190
3	426
4	645
5	850
6	1023

Envisagé dans le chapitre du bas de la Page 12, les divers menus d'exploitation vont utiliser le positionnement du **Potentiomètre** branché sur **A0**. Pour en effectuer la gestion nous aurons besoin des valeurs que retourne la **CAN**. Pour les noter les démonstrateurs **P05** ou **P06** affichent ces dernières dans la ligne **1** de la Fig.31 en Page 19. Dans le tableau de la Fig.42 sont résumées les valeurs relevées sur le prototype. Elles sont directement fonction des caractéristiques de **Pot** mais également de la position des points tracés pour l'indexation, et bien entendu de l'ouverture du potentiomètre. Il suffit pour votre exemplaire de faire ces relevés, et surtout ne vous formalisez pas si les valeurs affichées fluctuent. Leur saisie à ± 10 unités **Page 24**

sera largement assez précise comme on va le constater dans le chapitre qui suit. Si par la suite nous envisagions sept indexations au lieu de six, il suffirait de remplacer le disque en carton par celui en **A** sur la Fig.28 et de téléverser à nouveau **P05** ou **P06** pour refaire la manipulation. Avant d'en terminer avec la validation complète des deux cartes électroniques, on débranche l'entrée **E** du **Signal de test** et on la relie comme suggéré sur la Fig.43 à un quelconque potentiomètre à variation linéaire compris entre **1kΩ** et **47kΩ**. C'est d'autant plus facile à faire que le **+5Vcc** et **GND** sont disponibles sur les picots non utilisés par les straps de polarisation du petit écran OLED. Quand on balaye l'intégralité de son ouverture, l'information de la ligne **U sur E** en **2** de la Fig.31 doit varier entre **0.00** et **5.00** volts.

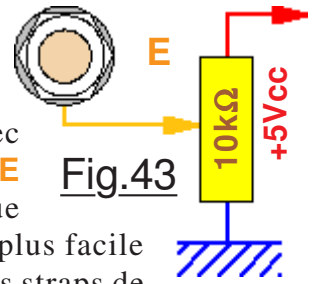


Fig.43

➤ Principe de la sélection de l'une des six fonctions envisagées

Quel que soit le MENU ou le sous-menu traité, l'idée consiste à déclencher l'une des six fonctions lorsque l'on active un **Clic long**. (Possible aussi sur un clic court.) **C'est la position calée sur le bouton du potentiomètre qui détermine laquelle est invoquée**, raison pour laquelle on utilise un bouton flèche et que sur l'étiquette six points d'indexage ont été tracés en divisant l'amplitude de la rotation en cinq secteurs d'ouvertures angulaires équivalentes. Pour fiabiliser et faciliter au maximum le travail de l'opérateur, la technique (Voir la Fig.44) consiste à attribuer à chaque index un secteur angulaire dont les limites sont éloignées au maximum des positions voisines. Il suffit de placer la frontière du secteur à la moyenne des deux positions mitoyennes. Avec cette approche, même souffrant d'une très mauvaise coordination gestuelle, plaçant vaguement la flèche du bouton pas trop loin du point vert, on déclenche sans aucun problème l'activation de la fonction désirée. Sur le prototype les valeurs à retenir sont les suivantes :

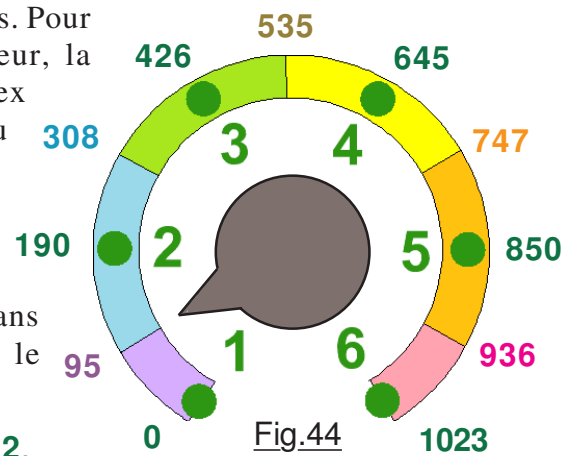


Fig.44

0 à 95 : Fonction 1.	96 à 308 : Fonction 2.
309 à 535 : Fonction 3.	536 à 747 : Fonction 4.
748 à 936 : Fonction 5.	939 à 1023 : Fonction 6.

Pour tester la routine **Lire_INDEX** qui gèrera les menus et qui consomme 136 octets téléverser **P07_Gestion_MENUUs.ino** qui affichera en ligne du bas la position de l'INDEX déterminé lors de l'utilisation de n'importe quelle touche du clavier, que le clic soit long ou court. Comme maintenant avec le bruiteur ACTIF on ne dispose que de la tonalité à 3000Hz, la routine **BIP(byte Duree)** est paramétrée. Les BIPs d'alerte seront longs, alors que chaque clic sur une touche du clavier ne déclenchera qu'un BIP très court mais suffisant pour informer l'opérateur de sa prise en compte.

Faire un petit bilan avant de commencer l'étude de l'oscilloscope, me semble utile à ce stade. Si l'on compare le travail de **P07** au cahier des charges de la Page 12 on a traité les items suivants :


- 01) Limiter à quatre boutons poussoir et un potentiomètre l'encombrement de la façade de pilotage.
- 03) Présence d'une LED jaune entre les boutons poussoir pour attester d'un **Clic long**.
- 04) Protection de l'entrée si surcharge en tension ou inversion de polarité.
- 07) Envisager la présence d'un petit bruiteur ACTIF pour la gestion du clavier et des incidents.
- 10) Fournir en permanence un **Signal de test**.

Pourtant, vous êtes en droit de vous inquiéter pour la suite du projet. En effet, strictement aucune fonction relative à l'oscilloscope n'est écrite et le démonstrateur se gloutonne déjà 34% de l'espace mémoire. Rassurez-vous, l'expérience sur un très grand nombre de projets m'a montré que chaque fois les premières ébauches prennent beaucoup de place. En réalité on fait appel à des bibliothèques qui engendrent beaucoup de code. Puis, par la suite on ne va utiliser que des "instructions élémentaires" et la boulimie en octets va rapidement s'étioler. Il n'y a pas de raison que pour ce projet ce soit différent, personnellement je reste confiant. Par ailleurs, on a déjà mis en place la gestion de l'afficheur qui incorpore dans le code sa police de caractères et ses routines. C'est l'un des deux gros consommateurs actuels. Le deuxième "convive affamé" réside dans les bavardages, c'est à dire les textes affichés à l'écran. Le chapitre suivant va traiter de ce sujet fondamental.

09) Gestion de la mémoire non volatile EEPROM.

Dans l'encadré rose en bas de la page 18 était précisé avec insistance : **ATTENTION**, les tableaux sont les données les plus voraces en espace utilisé *et les textes sont intrinsèquement des tableaux*. Aussi, *si la mémoire non volatile EEPROM n'est pas utilisée pour mémoriser des données à conserver sur coupure alimentation, y logger un maximum de textes affichés*. C'est par le respect de ce conseil que l'on peut réaliser le plus facilement une optimisation de la taille du code objet. À ce stade du développement, seule une trace oscilloscopique est prévue pour la sauvegarde en mémoire non volatile. Aussi, l'idée de base consiste à placer les échantillons en haut de la mémoire, et d'y "entasser" les textes affichés au fur et à mesure que l'on va développer les menus et les écrans de données. L'occupation envisagée est représentée sur la Fig.45, la répartition précise des zone restant à définir.

Envisageons la clause n°15 du cahier des charges qui stipule : Enregistrement de "quatre écrans de large" avec affichage fenêtré. Sachant que la largeur possible de la page graphique fait 128 PIXELs, pour sauvegarder quatre fois cette dernière il faudra réserver 512 octets en **Zone des échantillons**, soit exactement la moitié de l'espace disponible en EEPROM. L'étendue des **Données diverses** ne devrait pas dépasser les 30 octets au maximum, car elle ne contiendra que les attributs de l'enregistrement : Base de temps, sensibilité verticale, front de déclenchement etc. Il nous reste donc au moins 482 octets pour les divers textes affichés.

Pour compléter les "briques" qui assemblées plus tard constitueront le programme d'application, on va avec **P09_Tester_EEPROM.ino** développer les routines qui servent à gérer la mémoire non volatile de l'ATmega328. Pour les futurs démonstrateurs il faudra pouvoir sauvegarder une trace oscilloscopique, la récupérer pour l'afficher, *effacer la zone des échantillons, encore que ça ne présente pas un gros intérêt*. Si par la suite il faut "faire de la place" on pourra toujours enlever cette fonction du programme. Il faudra en outre pouvoir lire les textes figés en EEPROM pour les afficher. Naturellement, pour que **P09** puisse effectuer ce travail, il faut impérativement que des textes soient déjà présent en EEPROM. C'est la raison pour laquelle avant de téléverser **P09** il faut en préambule transférer **P08_Textes_provisoires.ino** et en **déclencher l'exécution** en activant le **Moniteur de l'IDE** avec  initialisé à **115200 baud**. Outre quelques textes pour tester les procédures d'affichages, la **Zone des échantillons** est entièrement remplie par une trace oscilloscopique fictive. On pourra ainsi disposer d'un enregistrement de début pour tester l'effacement de la zone. Ce n'est qu'une fois l'EEPROM en partie initialisée que l'on pourra manipuler avec **P09_Tester_EEPROM.ino** les séquences qui seront intégrées dans les prochains logiciels de validation. On notera au passage qu'avec un OCTET on dispose de 256 valeurs possibles, alors que la hauteur de l'écran ne fait que 64 PIXELs. Donc il faudra diviser par quatre les valeurs pour les afficher sur l'écran graphique.





Durée de vie d'une EEPROM.

Autant en lecture une mémoire EEPROM est inusable, autant en écriture elle présente une limite de fiabilité. Les fournisseurs de ce type de composants précisent qu'une EEPROM n'est garantie et fiable que pour 100.000 écritures. Ensuite elle va commencer à présenter aléatoirement des mauvaises inscriptions. Évidemment ce n'est qu'un ordre de grandeur.

CONCLUSION : ne jamais utiliser l'EEPROM comme de la RAM dans laquelle on passe son temps durant le déroulement d'un programme à y écrire des variables qui changent rapidement de valeurs, la mémoire non volatile arriverait rapidement à sa destruction.

Quand on stocke des valeurs qui ne doivent pas être perdues si se présente une coupure secteur, il importe pour le programmeur de s'assurer de la durée de vie du composant, ce que nous allons faire immédiatement. Les textes figés en EEPROM ne seront modifiés que durant le développement du programme, donc très peu de réécriture seront effectuées. Supposons que l'on sauvegarde dix fois par jours, tous les jours. Avant que l'EEPROM ne soit usée il va s'écouler $100000 / 10 = 10000$ jours soit 27 années complètes. Je doute fort que ce petit appareil ne soit utilisé de cette façon. Aussi, coté "usure de l'EEPROM" nous n'avons rien à craindre.

➤ Le remplissage provisoire de la mémoire non volatile.

Sachant que la hauteur de l'écran graphique est limitée à 64 PIXELs, il est totalement inutile de conserver la définition de 1024 pour la visualisation. *En divisant par quatre la valeur issue du CAN* elle sera comprise entre 0 et 256 et sera stockable dans des OCTETs, tenant ainsi deux fois moins de place. C'est la raison pour laquelle la *Zone des échantillons* n'occupe que la moitié de l'EEPROM. On commence par activer le **Moniteur** de l'**IDE** avec l'idéogramme  et on initialise la vitesse de transfert à **115200** baud. Puis on transfère le démonstrateur **P08_Textes_provisoires.ino** et on l'active en cliquant une deuxième fois sur . La fenêtre du moniteur se remplit et ressemble à la copie d'écran de la Fig.46 avec en violet des commentaires au fur et à mesure que le programme se déroule. En particulier dans la zone **1** les textes inscrits sont listés. Puis il y a affichage sous forme de tableau de l'intégralité du contenu de l'EEPROM avec l'indication des *adresses* des OCTETs dans la zone jaune. (*Adresse du premier octet de la ligne à gauche à laquelle on doit ajouter la "valeur" de la colonne affichée ici en hexadécimal.*) Noter qu'à tout moment dans la liste des logiciels fournis vous disposez à convenance d'un outil nommé **Lister_EEPROM_en_HEXAdecimal.ino** qui vous permet de lister le contenu de l'EEPROM de n'importe quelle carte Arduino. Dans la zone **2** sont donc listés les textes, pour ainsi déterminer l'adresse de leur début indispensable aux routines d'affichage qui viendront puiser les caractères. Pour pouvoir facilement lors de l'élaboration du programme qui viendra inscrire les textes du logiciel d'utilisation final, il est utile pour le programmeur de savoir en permanence la zone de **2** qui reste encore disponible à chaque stade du développement. Les OCTETs non utilisés sont repérés par **".."** et détectés par la remise à zéro initiale de l'intégralité de l'EEPROM avec des \$FF. Dans la zone des données colorisée en bleu on observe en adresse 511 le chiffre **05**. Il indique la façon dont sont sauvegardés les échantillons. Il est à ce stade envisagé deux protocoles possibles si la taille occupée par le programme utilisateur le permet :

- Les 512 OCTETs sont remplis par une **Trace longue** contenant quatre écrans de large.
- La zone **3** pourra contenir *quatre enregistrements différents* contenant une largeur d'écran.



Fig.47

Contenu EEPROM du 7 Novembre 2023.

Ecriture en EEPROM des textes :

Rev :

Trace actuelle : Haut

Trace en EEPROM : Bas

RAZ EEPROM : Gauche

Un B.P. pour sortir.

Trace longue.

Inscription des Echantillons en EEPROM.

ADRS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	R	e	v	:		T	r	a	c	e	a	c	t	u		
0016	e	l	l	e	:		H	a	u	t	T	r	a	c	e	
0032	e	n		E	E	P	R	O	M	:		B	a	s		
0048	R	A	Z		E	E	P	R	O	M	:		G	a	u	
0064	c	h	e	U	n		B	.	P	.		p	o	u	r	
0080	s	o	r	t	i	r	.	T	r	a	c	e		l	o	n
0096	g	u	e	.	O	U	I	:		B	P		H	a	u	
0112	t	.	N	O	N	:		B	P		B	a	s	..		
0400
0496	05
0512	80	88	91	9A	A3	AB	B3	BB	C3	CA	D1	D8	DE	E4	E9	ED
0528	F2	F5	F8	FB	FD	FE	FE	FE	FE	FD	FB	F8	F5	F2	ED	F9
0960	6E	77	7F	88	91	9A	A3	AB	B3	BB	C3	CA	D1	D8	DE	E4
0976	E9	ED	F2	F5	F8	FB	F5	FE	FE	FE	FE	FD	FB	F8	F5	F2
0992	ED	E9	E4	DE	D8	D1	CA	C3	BB	B3	AB	A3	9A	91	88	80
1008	77	6E	65	5C	54	4C	44	3C	35	2E	27	21	1B	16	12	0D

Fig.46

Fig.46

La *Zone des échantillons* est listée en HEXADÉCIMAL car les valeurs peuvent aller jusqu'à 254 qui en décimal impose trois caractères, alors que les colonnes sont formatées à deux.

➤ Utilisation de P09_Tester_EEPROM.ino.

Après avoir inscrit les textes et une simulation de **Trace longue** en EEPROM, on téléverse le logiciel **P09** qui affiche l'écran de la Fig.47 à son démarrage. On est ainsi directement renseigné sur la façon d'utiliser le petit clavier. Cliquer sur le bouton du **Haut** déclenche la visualisation de la Trace actuellement en mémoire. Comme l'oscilloscope n'est pas encore émulé, cette dernière est générée artificiellement au démarrage. Elle est



Fig.48

MENU de BASE, cliquer sur le bouton de **Gauche** génère un BIP sonore pour avertir l'opérateur que s'il valide il y aura "perte" définitive d'informations et l'écran **B** de la Fig.50 précise l'action en cours. Confirmer avec le BP **Haut**. Provoquer un RESET, puis

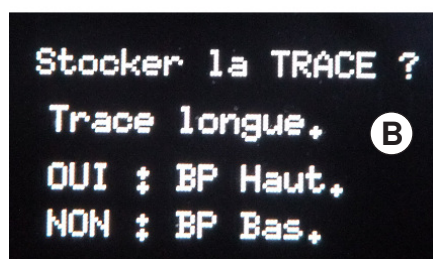
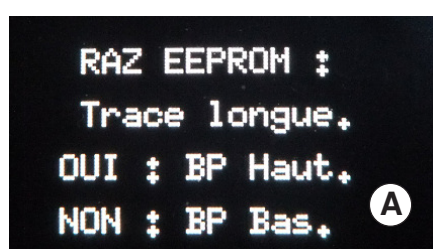


Fig.50

comme sur la Fig.48 constituée d'une dents de scie classique sur les générateurs basses fréquences pour tester la linéarité d'un amplificateur par exemple. Superposé en haut de la trace est indiquée la consigne pour revenir "au MENU de BASE". Cliquer maintenant sur un bouton quelconque pour sortir et sur le bouton du **Bas** pour faire afficher la **Trace longue** actuellement figée en EEPROM. L'écran affiche la belle sinusoïde de la Fig.49 qui pourrait être celle de la sortie d'un transformateur secteur. Comme pour la fonction précédente toute touche du clavier fait sortir de cette page écran. Étant dans le

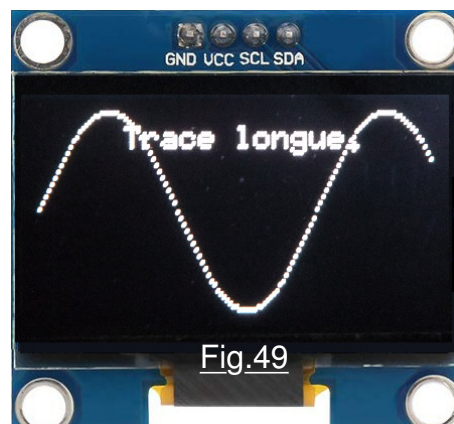


Fig.49

cliquer sur le bouton du **Haut** suivi de deux clics sur celui du **Bas**. La page écran confirme l'effacement définitif de la trace qui était en EEPROM. Déclencher encore un RESET suivi du bouton poussoir de **Droite**. Avertissement sonore d'une perte potentielle définitive d'informations. Confirmer avec "**OUI**". Cliquer enfin sur la touche du **Bas** pour vérifier que "la dent de scie" a bien été inscrite en mémoire non volatile. La belle sinusoïde est définitivement perdue sauf à faire usage du démonstrateur **P08**. Vous avez observé que pour ces deux fonctions, durant l'écriture en EEPROM la composante rouge de la LED triple s'allume confirmant l'opération. À ce stade le programme consomme 8534 Octets et 948 emplacements en mémoire dynamique.

➤ "Passage" des textes en EEPROM.

Nous allons par les manipulations qui vont suivre vérifier le bénéfice patent à inscrire les textes de dialogue opérateur dans la mémoire EEPROM. Dans ce but on va éliminer toutes les procédures qui affichent actuellement du texte et les remplacer par des procédures *totalemt identiques mis à part que les textes sont puisés en mémoire non volatile*. Pour ce faire commencer par transformer le bloc [===== Routines pour les textes dans le programme =====] qui contient l'intégralité de ces procédures. Il suffit de le faire précéder par **/*** et de le terminer par ***/**. Puis enlever ces deux types de délimiteurs dans le bloc [===== Routines pour les textes dans l'EEPROM =====] situé juste au dessus. Téléverser cette version et vérifier par les manipulations précédentes qu'il n'y a pour l'opérateur strictement aucune différence. Pourtant, le programme fait 142 OCTETs de moins alors que l'on a ajouté la procédure **Aff_TEXTE_EEPROM(Paramètres)**. En mémoire dynamique on économise 184 OCTETs, avantage substantiel pour diminuer le risque de collision de PILE. (*Problème abordé en détails plus avant.*) On imagine le gain de place que l'on va réaliser quand l'intégralité des dialogues viendront "gonfler les bavardages". De plus, le bénéfice est encore plus avantageux si le texte contient des accentués, ce qui n'est pas le cas dans ce démonstrateur. La conclusion s'impose : **Un maximum de textes sera logé en mémoire EEPROM.**

➤ La lisibilité d'un programme.

Finalité de ce petit projet : S'amuser avec Arduino tout en apprenant à programmer avec méthodes. Dans cette optique il me semble incontournable d'aborder le thème de la lisibilité d'un logiciel. En Page 13 ce thème a déjà été abordé et je vous invite avec insistance à relire le chapitre qui commence par la vérité biblique *la première qualité d'un logiciel quel qu'il soit est sa LISIBILITÉ*. Nous allons observer dans le démonstrateur **P09** quelques lignes de programme où la lisibilité a été pris en compte. Considérons une instruction du type **Aff_TEXTE_EEPROM(48,12)** utilisée

pour le dialogue Homme/Machine. Lors du développement chaque fois que l'on désire modifier un texte, il faut le changer en EEPROM. Puis modifier les paramètres le concernant dans le programme. Pour retrouver la bonne instruction dans le fatras des **Aff_TEXTE_EEPROM** c'est une galère, car 48,12 n'est vraiment pas lisible. C'est la raison pour laquelle il est important de faire suivre ces instructions par des remarques de type // "RAZ EEPROM :" précisant le texte affiché par l'instruction.

Autre point important qui facilite indubitablement la lecture du logiciel. Considérons la Fig.51 qui précise le numéro arbitraire affecté à chaque touche du petit clavier. Si l'on analyse la séquence de la Fig.52 on est obligé de se reporter au dessin du clavier pour comprendre à chaque fois de quel touche il s'agit, alors que la séquence Fig.53 est bien plus parlante.

Fig.52

```
switch (Touche) {
  case 1 : {Affiche_trace_EEPROM(); break;}
  case 2 : {Affiche_trace_Actuelle(); break;}
  case 3 : {BIP(150); Efface_trace_EEPROM(); break;}
  case 4 : {BIP(150); Enregistrer_la_TRACE();}}
```

Fig.53

```
switch (Touche) {
  case BAS : {Affiche_trace_EEPROM(); break;}
  case HAUT : {Affiche_trace_Actuelle(); break;}
  case GAUCHE : {BIP(150); Efface_trace_EEPROM(); break;}
  case DROITE : {BIP(150); Enregistrer_la_TRACE();}}
```

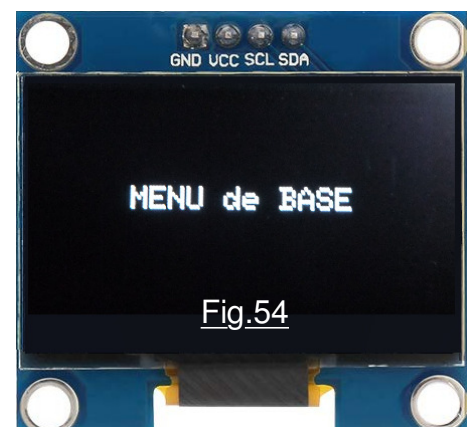
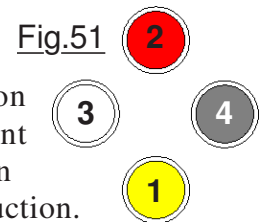
Dans le même ordre d'idée, un `if (Touche == OUI)` sera plus immédiat qu'un `if (Touche == Haut)` ou un `if (Touche == 2)`. On insiste ici sur l'importance dans un programme de choisir des identificateurs parlants que ce soit pour les noms de procédures, les variables ou les paramètres.

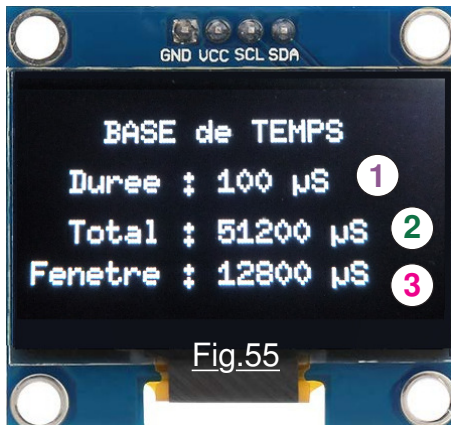
10) La BASE de TEMPS.

Avec les démonstrateurs précédents nous avons bien préparé le terrain. Les briques logicielles qui vont créer le programme d'utilisation sont en place. On va dans ce chapitre vraiment commencer à émuler les fonctions indispensables à tout oscilloscope. En particulier la première fonction fondamentale consiste à enregistrer une trace pour pouvoir ensuite l'afficher graphiquement. En fonction de la fréquence de variation du signal injecté en entrée de l'appareil on doit pouvoir choisir la rapidité (*Ou la lenteur*) de l'échantillonnage. C'est le rôle de la **BASE de TEMPS**. Pour choisir la temporisation entre deux échantillons il existe une infinité de protocoles possibles. Utilisant les particularités et prenant en compte les limites de l'écran graphique, le programmeur a la charge de trouver des affichages et des manipulations simples et conviviales. Dans ce chapitre nous allons développer cette facette de l'utilisation de notre petit appareil. Pour éviter d'avoir à modifier les textes en EEPROM à chaque évolution des démonstrateurs, dans ces derniers les chaînes de caractères seront incluses dans le programme. Ce n'est qu'avec le logiciel final d'utilisation que l'on placera tous les "dialogues" en EEPROM.

➤ La fonction d'initialisation de la BASE de TEMPS.

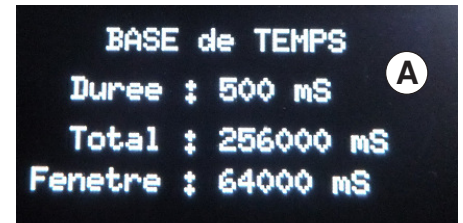
Invquée dans un **MENU de BASE** par l'opérateur quand il désirera sélectionner la rapidité de la BASE de TEMPS elle est émulée comme une fonction indépendante dans le démonstrateur **P10_La_BASE_de_TEMPS.ino**. Sur RESET ce croquis commence par l'écran de la Fig.54 qui simule un **MENU de BASE** qui dans la pratique contiendra la liste des fonctions principales de l'oscilloscope. N'importe quelle touche du clavier, avec *clic court* ou *long* conduit au sous-menu de la fonction d'initialisation de la base de temps qui à l'ouverture affiche l'écran de la Fig.55 correspondant à la plus grande vitesse d'échantillonnage possible. Pour mémoire la **CAN** exige précisément cette durée de 100µS. Aussi la plus grande cadence sera obtenue quand les échantillons seront capturés sans générer de délai entre chaque **CAN**. Cette fonction ignore la durée d'enfoncement des touches, donc l'opérateur peut utiliser le clavier rapidement. C'est le bouton





de DROITE qui fait sortir de la fonction en cours et ramène au **MENU de BASE**. En **1** est précisée la durée de temporisation entre la capture de deux échantillons. En **2** la durée totalisée par les 512 mesures. Enfin **3** représente le temps écoulé pour les 128 échantillons qui seront représentés sur la fenêtre de l'écran graphique. Les trois entités seront suivies de l'unité temporelle utilisée qui peut être la microseconde, la milliseconde ou la seconde. La touche du HAUT fait augmenter la valeur de la **BdT** alors que celle du BAS la diminue. Les variations sont en permutations circulaires dans la

progression 1, 2, 5. En **A** de l'écran de la Fig.56 sont obtenues les valeurs les plus grandes qui seront affichées. En **B** on trouve la lenteur la plus élevée de la base de temps. Avec cinq secondes entre la saisies de deux échantillons on effectue la totalité de l'enregistrement sur un délai de presque 43 minutes. J'ai pensé que ce serait largement assez. Les durées de la **BdT** possibles sont : 100µS, 200µS, 500µS, 1mS, 2mS, 5mS, 10mS, 20mS, 50mS, 100mS, 200mS, 500mS, 1S, 2S et 5S. On respecte je pense la



clause **09) Base de temps avec large plage de vitesses d'échantillonnage** du cahier des chages. Les textes affichés étant supposés rangés en EEPROM cette fonction consomme environ 3300 Octets de programme soit environ 10% de l'espace disponible et environ 25 Octets dans la mémoire dynamique. Cette taille peut effrayer. C'est une facette relativement complexe. Au final, à ce régime on peut en loger dix aussi développées, alors au final on peut rester "zen", pour le moment notre projet n'est pas compromis. Alors passons à la suite logique du développement.

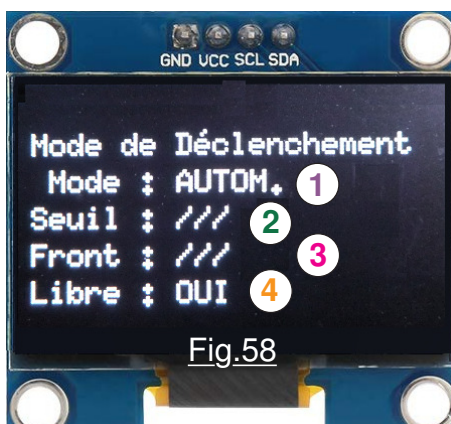
11) Les options de déclenchement.

A vant de pouvoir afficher une trace, il faut effectuer la saisie des échantillons. Pour assurer cette deuxième fonction fondamentale de l'oscilloscope, il faut tenir compte des options de déclenchement retenues par l'opérateur qui peut désirer :

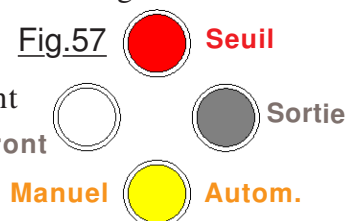
- Une trace répétitive. (*Le déclenchement est automatique et permanent.*)
- Un déclenchement manuel. (**CAN** suivie d'une **PAUSE** jusqu'à ce que l'opérateur intervienne.)
- Un déclenchement avec attente du seuil de tension, ou du choix de la transition.
- Qu'il soit manuel ou répétitif, dans tous les cas le déclenchement pourra se faire *sans contrainte* ou *en respectant le seuil* de tension, et un *choix du flanc de transition*.

➤ Le sous-menu des choix du mode de déclenchement.

C omme dans le cas du croquis précédent, cette fonction sera invoquée dans un **MENU de BASE** par l'opérateur. Du coup sur RESET on retrouve l'écran de la Fig.54 avec un comportement identique du clavier qui ignore *clic court* ou *clic long* laissant à l'opérateur le choix d'utiliser rapidement le clavier. C'est **P11_Mode_de_DECLENCHEMENT.ino** qui se charge de tester cette facette de l'utilisation d'un oscilloscope. La Fig.57 décrit les choix effectués



pour distribuer les commandes. Pour des raisons d'homogénéité de comportement le retour au **MENU de BASE** se fait avec le BP de DROITE. Comme il y a quatre paramètres à initialiser, la touche du BAS aura deux effets. Un *clic court* va faire en **1** de la Fig.58 alterner le mode entre **MANUEL** et **AUTOMATIQUE** sans influencer les diverses contraintes de déclenchement. Un *clic long* impose la numérisation **LIBRE** en **4** et annule toutes les contraintes de déclenchement. Le bouton poussoir de GAUCHE alterne entre **Front Montant**



et Front **Descendant**. Il annule l'option **LIBRE** en **4** mais n'a pas d'effet sur **Seuil** en **2** si cette option est active. La touche du HAUT se comporte comme une bascule de type OUI/NON. Elle valide ou suspend l'option de **Seuil** en **2**. La validation annule **LIBRE** et impose un **Front** de type **Montant** car il faut impérativement une sélection de transition avec le mode **Seuil** qui surveillera le flanc de transition pour le déclenchement de la **CAN**. **Seuil** étant activé on peut librement inverser de type de **Front**. Son annulation impose **LIBRE** et annule **Front**. La Fig.58 correspond à la configuration par défaut. Sur la Fig.59 en **A** on est passé en mode **MANUEL**. L'option **Seuil** a été imposée puis on a inversé avec le bouton de GAUCHE le sens de la transition. En **B** on s'est contenté de revenir au **Front Montant** et on a choisi le déclenchement **AUTOM**. En cliquant encore sur le bouton du HAUT on a annulé le mode **Seuil** et on a retrouvé la configuration par défaut **AUTOM** et **LIBRE**. Enfin un clic sur la touche de GAUCHE a engendré la configuration **C** où le **Seuil** n'est plus défini, seul le Front sera pris en compte. Le tableau de la Fig.60 résume les options et leurs effets sur la configuration du déclenchement. Cette fonction que l'on peut expérimenter à souhait avec le démonstrateur **P11** consomme environ 3562 Octets de programme soit presque 12% de l'espace disponible. Elle réalise toutefois les deux clauses du cahier des charges :

11) Synchronisation du déclenchement en option.

10) Synchronisation avec **choix du seuil et de la transition**.

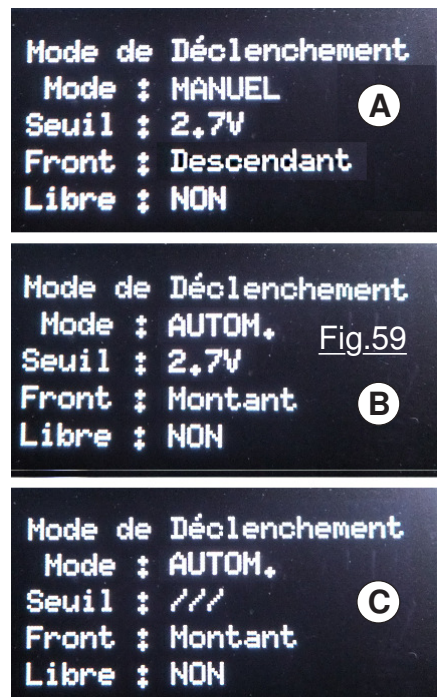


Fig.60

Option	État	Effet
Mode	AUTOM	Aucun effet sur les autres options.
Mode	MANUEL	Aucun effet sur les autres options.
Seuil	OUI	Impose Front Montant et annule LIBRE .
Seuil	NON	Annule Front et active LIBRE .
Front	OUI / NON	Si Seuil = /// annule LIBRE et change d'état. Si Seuil validé change l'état de Front .
Libre	OUI	Annule Front et annule Seuil .

➤ Quelques détails sur le mode de déclenchement.

Électroniciens rompus à l'usage d'un oscilloscope, passer votre chemin, il n'y a rien à voir, circuler ! Ce chapitre s'adresse aux béotiennes et au béotiens. Nous allons détailler un peu les divers **modes de synchronisation**, ou si vous préférez de déclenchement. Premier cas, l'oscilloscope est en **AUTOM**atique et le déclenchement est **Libre**. Automatique signifie que le programme enchaîne en permanence des enregistrements et les visualise sur l'afficheur graphique. Par exemple sur la Fig.61 qui va nous servir d'exemple, le premier enregistrement est montré en vert. Comme le temps

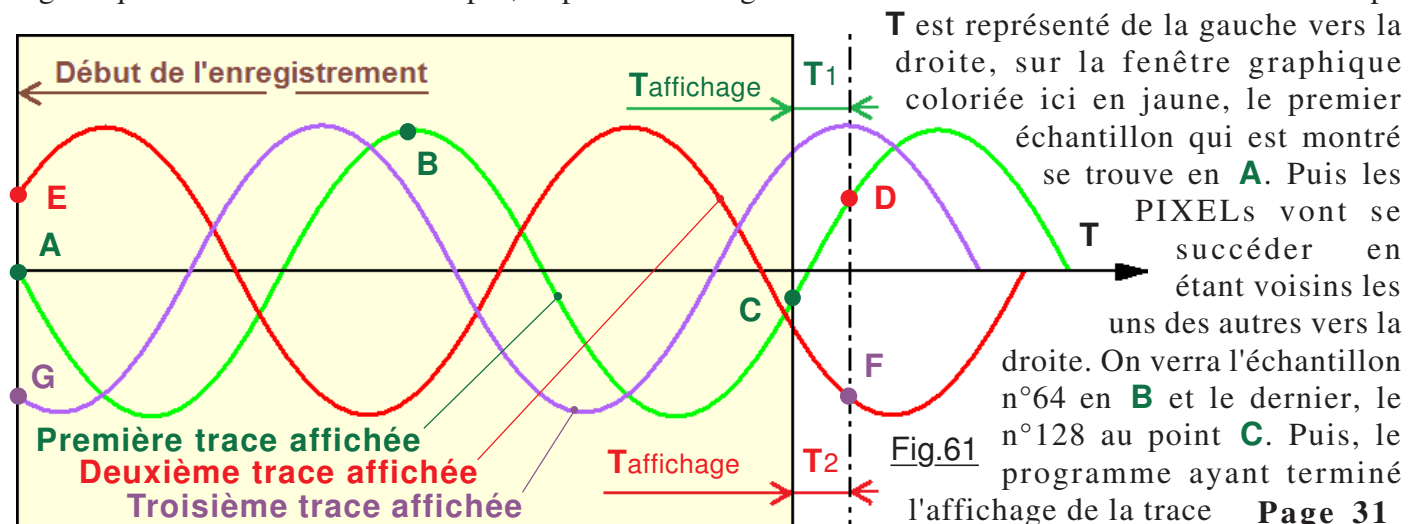


Fig.61

sur la largeur de la mosaïque de l'écran, il va immédiatement enchaîner l'enregistrement suivant car avec **Libre** il n'y a aucune contrainte de déclenchement. Pour afficher sur OLED la trace verte il faut l'intervalle de temps **T1**. La nouvelle séquence d'enregistrement commence en **D**. Elle est alors immédiatement affichée (*Repérée ici en rouge.*) à partir du point **E**. Le signal continue à évoluer. Dès que l'écran a affiché la trace rouge, ce qui prend le temps **T2**, au point **F** le logiciel amorce un troisième enregistrement. Il l'affiche ensuite en commençant en **G**. On constate que la courbe à l'écran change constamment de morphologie. Cette configuration est inexploitable pour analyser le signal. Elle n'est utile que pour vérifier "qu'il se passe quelque chose" ou pour aider quand la trace est mal centrée et "hors de l'écran graphique". Si l'opérateur ne s'intéresse pas particulièrement au moment où sera déclenché l'enregistrement, il suffit de passer en mode **MANUEL**. Chaque clic sur la touche prévue à cet effet déclenchera un enregistrement qui sera visualisé aussi longtemps que l'opérateur n'en provoquera pas un autre. Indispensable en mode **AUTO**matique pour avoir des images "stables", ou en **MANUEL** pour choisir les conditions de déclenchement, il faut imposer à la capture des échantillons une synchronisation.

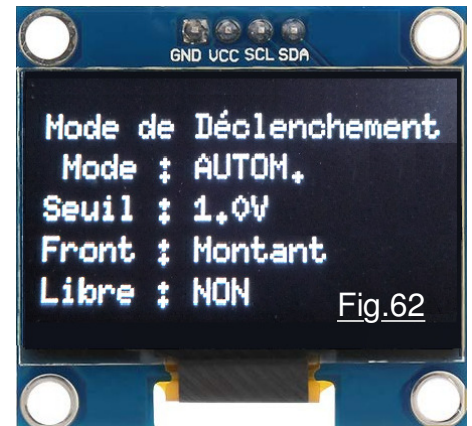
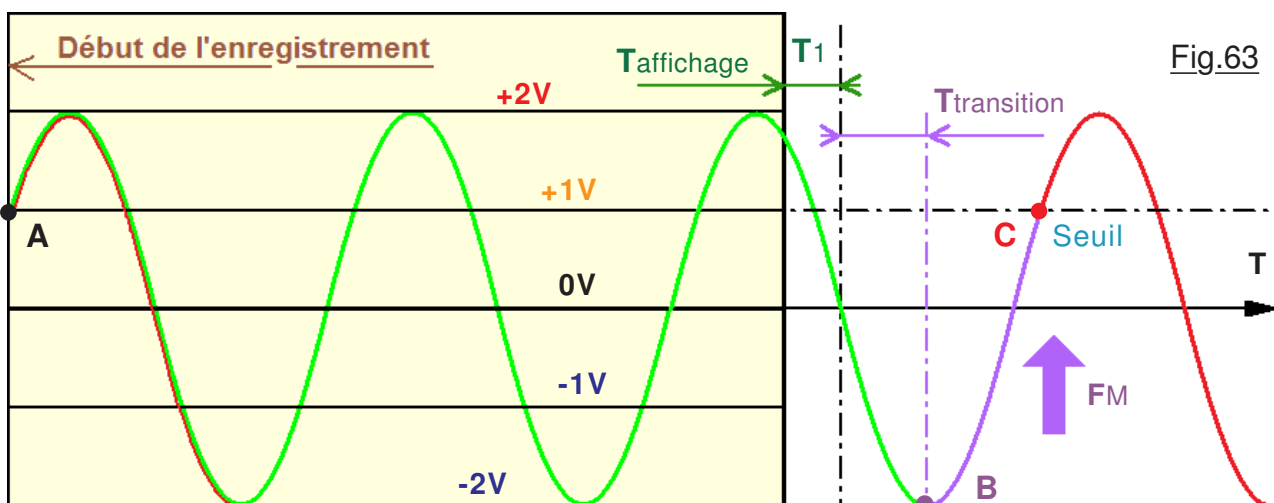


Fig.62

Supposons comme illustré sur la Fig.63 que le mode de déclenchement choisi **Seuil** soit de **+1V** et pour une transition montante. (*Voir la page écran de la Fig.62*) La courbe verte correspond à la première trace qui au point **A** commence à +1V alors que la tension augmente. Le logiciel consomme **T1** pour l'afficher. Puis il attend **Ttransition** jusqu'à détecter en **B** le début de l'augmentation de la tension, c'est à dire le Front Montant **FM**. Le programme surveille alors la valeur de la tension



et attend qu'elle arrive à +1Vcc. Il déclenche alors en **C** la numérisation. Si le signal est périodique, la courbe rouge numérisée est exactement analogue à la verte. Son affichage à partir de **A** se "superposera" à la courbe verte et l'opérateur aura l'impression que l'affichage est figé. L'utilisateur peut alors calmement observer la courbe. Toutefois, s'il veut l'analyser entièrement sur les "quatre fenêtres de large" il faudra passer en **PAUSE** et travailler en **MANUEL**.

La configuration de la Fig.59 **C** est à privilégier si le signal est de type binaire. Ainsi, dès que le logiciel a détecté le **Front** souhaité il déclenche la numérisation sans avoir à se préoccuper d'un seuil. La trace commence alors par une transition "verticale" montante ou descendante.

Dans tous les cas **la synchronisation ne donne à l'écran une image stable que si le signal est vraiment périodique**. Si la forme du signal change en permanence, les traces à l'écran seront différentes à chaque rafraîchissement de l'image et inutilisables pour leur analyses. C'est aussi vrai pour un signal analogique qu'une suite d'impulsions binaires. (*Par exemple les signaux sur une voie série comme la RS232 ou une ligne I2C.*) Naturellement pour le moment vous ne pouvez pas expérimenter ces divers phénomènes, mais dès que l'oscilloscope sera entièrement émulé, on "s'amusera" à expérimenter les nombreux aspects de ce thème. Le menu de la synchronisation est en place, il reste maintenant à lui donner vie dans le logiciel, objet du chapitre qui suit.

12) Représentation de la trace sur l'écran graphique.

Conditionnant directement le nombre d'échantillons mémorisés et l'occupation de l'espace RAM ainsi que celui de l'EEPROM pour les stocker, l'affichage graphique prend le pas sur le développement de la capture des enregistrements. C'est la raison pour laquelle, avant de développer les séquences d'enregistrement et de stockage des échantillons nous allons en préambule définir les représentations possibles des traces sur OLED ce qui va remettre un peu en cause les décisions prises dans le chapitre n°9 et surtout la gestion de la **BdT**. En première évaluation on désire :

- L'affichage ou non de la grille de repérage "des unités".
- L'affichage ou non des conditions de déclenchement. (*Pas forcément pertinent à l'usage.*)
- L'affichage ou non des unités imposées sur la grille.
- La précision de la valeur efficace du signal représenté dans la fenêtre visualisée.

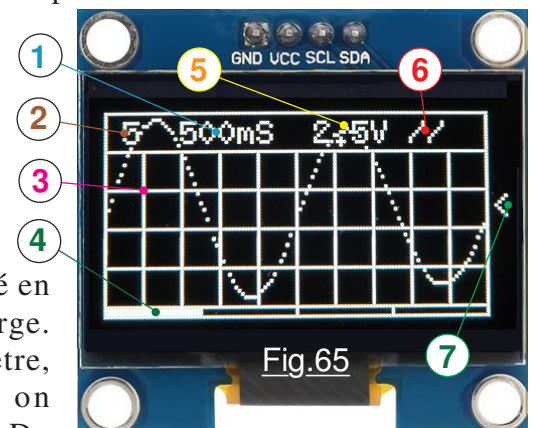
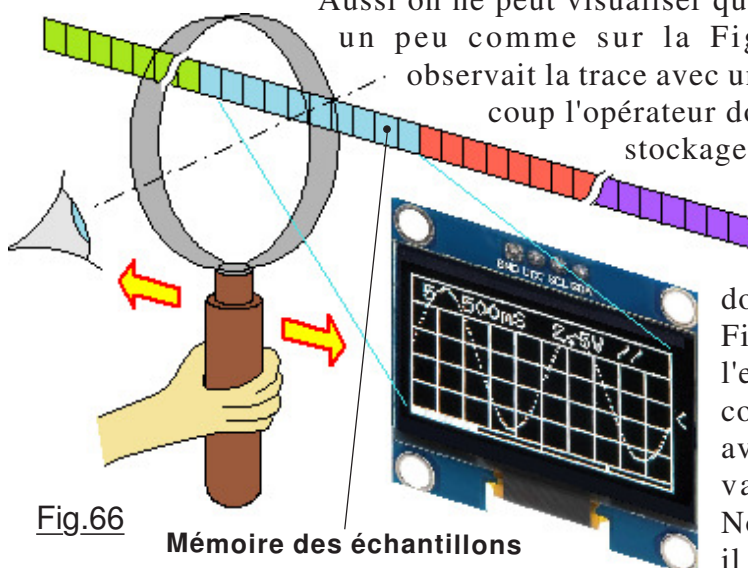
➤ Un écran avec graticule.

Lorsqu'un électronicien analyse une trace oscilloscopique, il doit impérativement pouvoir facilement évaluer les tensions pour les déviations verticales et les intervalles de temps pour l'étalement horizontal du graphe. Il serait possible d'afficher deux lignes graduées, l'une verticale, l'autre horizontale, toutefois il est bien plus convivial de représenter à l'écran une grille en précisant la grandeur correspondante à chaque division. *Une telle grille* est généralisée sur

tous les oscilloscopes et *porte le nom de graticule*. C'est précisément la concrétisation de ce graticule sur OLED qui va engendrer un effet "boule de neige" nous confrontant à de nombreuses difficultés. Pour pouvoir examiner facilement les choix qui vont s'imposer, une trace sera directement

disponible sur RESET. Représentée sur la Fig.64 elle est constituée d'une sinusoïde amortie dont le décroissement est linéaire. C'est à dire que la diminution de l'amplitude est proportionnelle au temps qui s'écoule comme mis en évidence en rose sur le dessin. La trace montrée sur la Fig.64 représente la numérisation complète pour quatre fenêtres de large repérées par les différentes couleurs sur la courbe du graphe. Comme montré sur la Fig.65 qui correspond à l'écran affiché *suite à un RESET*, sur OLED la courbe n'est plus continue, mais est construite avec une succession de points. En 1 est indiquée la durée correspondant à une "graduation" en largeur. En 2 c'est le calibre de tension correspondant à toute la hauteur. Dans notre cas chaque graduation verticale représente une différence de tension de 1V. Les graduations sont donc constituées du *graticule* repéré en 3. L'enregistrement complet correspond à quatre écrans de large.

Aussi on ne peut visualiser qu'une fenêtre, un peu comme sur la Fig.66, si on observait la trace avec une loupe. Du coup l'opérateur doit être informé de la "section" en mémoire de stockage qui est montrée sur l'écran. C'est le rôle de la section 4 qui symbolise sur la largeur du cadre les quatre sections possibles. Sur la Fig.65 le curseur est complètement à gauche. C'est donc le premier quart de la courbe en vert sur la Fig.64 qui est affiché correspondant au début de l'enregistrement. En 5 et en 6 sont résumées les contraintes de déclenchement de la numérisation avec en 5 le *Seuil* du déclenchement s'il est validé, et en 6 la nature de la transition. Normalement, si comme ici le *Seuil* est validé, il devrait y avoir obligatoirement

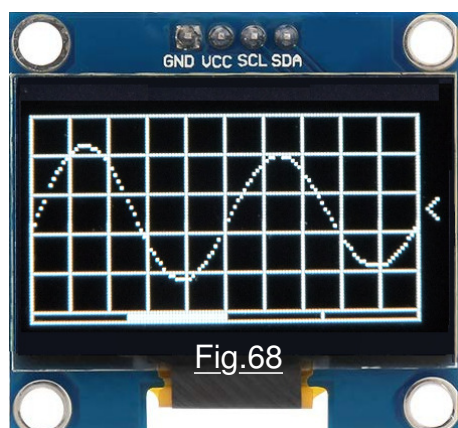


Page 33

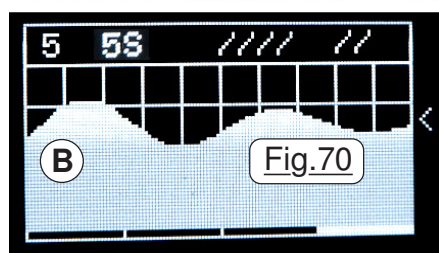
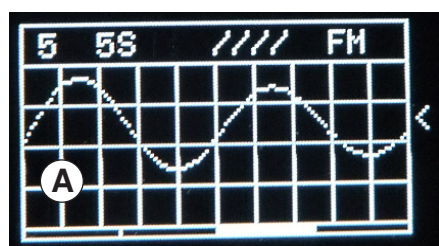
l'indication en **6** d'une transition. Comme il s'agit d'une simple simulation d'affichage, le démonstrateur **P12** a été simplifié. En **7** le curseur est positionné à la valeur de la *tension efficace globalisée par tous les échantillons situés dans la fenêtre* actuellement visualisée.

➤ Les commandes provisoires.

Provisaires car dans les choix actuels de la Fig.67 il n'est pas prévu de sortir de la fonction pour revenir au **MENU de BASE**. Par ailleurs pour pouvoir tester les options d'affichage et les choix visuels il faut disposer de commandes qui ne seront plus du tout pertinentes sur le programme d'utilisation de l'appareil. Ces commandes "hors propos" sont surlignées en jaune sur la Fig.67 et s'obtiennent avec un *clac long*. La touche du haut ignore la durée du clic. Passons à l'inventaire des divers modes retenus pour l'affichage. Comme on s'en doute, GAUCHE et DROITE déplacent "la loupe de la Fig.66" sur la "largeur" de l'enregistrement en permutation circulaire.



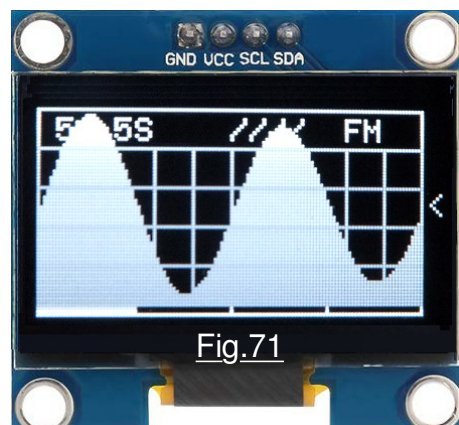
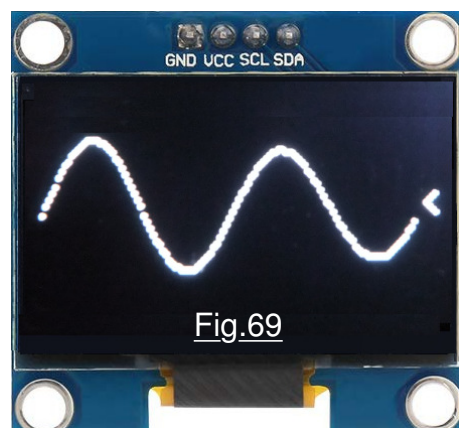
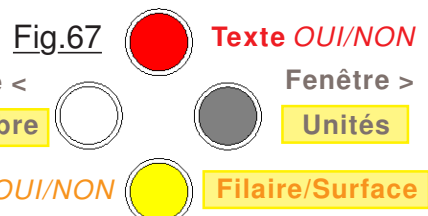
L'index en **4** de la Fig.65 précise laquelle des quatre fenêtres est actuellement visualisée. Par exemple sur la Fig.68 on a cliqué sur DROITE et on montre la courbe bleue de la Fig.64 avec une amplitude plus faible. Ayant cliqué sur HAUT le texte n'est plus affiché. Du coup le graticule va jusqu'en haut de la page écran. On peut remarquer que comme il s'agit d'un signal alternatif avec une composante continue de 2,5V, la valeur efficace est à cette valeur. Un clic court sur BAS et le graticule est effacé pour obtenir sur la Fig.69 (*Un peu surexposée !*) l'écran le moins encombré



d'informations. Toutes les combinaisons sont possibles, par exemple cliquer sur la touche HAUT et les textes seront affichée, mais sans la grille. Sur la Fig.70 en **A** la grille et les textes sont rétablis et on visualise la fenêtre n°3. (*L'amplitude de la sinusoïde a diminué.*) Avec DROIT on a changé les unités qui sont passées de la **mS** à la **Seconde**. Avec un *clac long* sur GAUCHE on a imposé le mode "déclenchement sur **Front Montant**". Avec un *clac court* sur DROITE on a imposé la fenêtre n°3. Enfin sur le cas **B** c'est la fenêtre n°4 qui est visualisée. L'amplitude de la sinusoïde est encore plus faible. En haut on a l'affichage qui précisera un déclenchement Libre. Surtout, avec un *clac long* sur BAS le visuel est passé du mode filaire au mode surface. C'est la surface blanche qui associée à une moyenne sur les 119 échantillons permet de calculer la valeur efficace. Ce mode de représentation est particulièrement utile quand on visualise des signaux "carrés" les pixels représentatifs du signal pouvant se trouver masqués par l'une des lignes horizontale du graticule. Pour terminer ces manipulations avec le démonstrateur **P12_AFFICHAGES.ino** sur la Fig.71 on a rétablit les textes et l'affichage de la grille. Toujours en mode "surface" on est revenu sur la fenêtre n°1 de l'enregistrement dans laquelle la sinusoïde présente l'amplitude la plus élevée.



Ben Môa môa, quand le dos me démange je me graticule avec énergie !



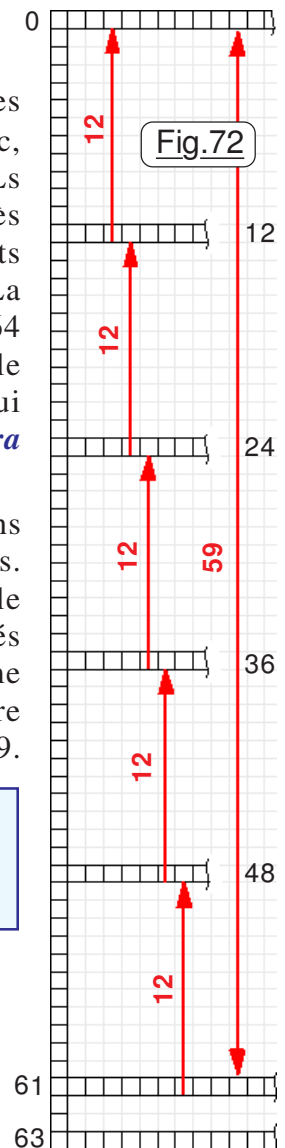
13) Contraintes imposées par les limites de l'écran graphique.

L adage populaire "*Le diable se cache dans les détails*" prend tout son sens dans ce chapitre. En effet, prendre des valeurs de **BdT** simples dans la progression classique 1, 2, 5 semblait couler de source. L'affichage adopté dans le démonstrateur **P12** va pourtant tout remettre en cause !

➤ Les limites imposées par le cadrage de la grille.

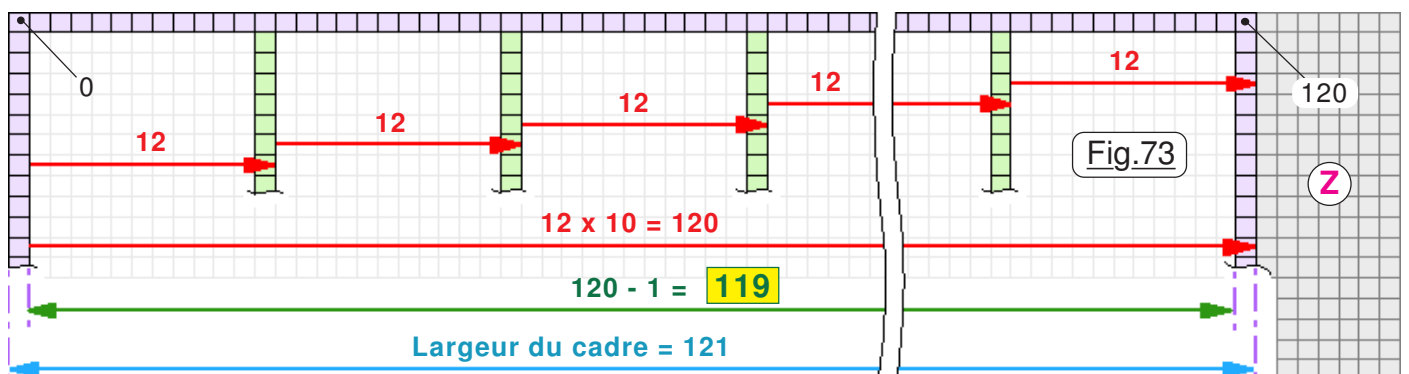
P our des raisons évidentes de simplification du logiciel et de facilité d'interprétation des graphes, le graticule doit afficher un nombre de cases entier aussi-bien en largeur qu'en hauteur. Comme le calibre en entrée est de 5Vcc, il va de soit qu'en hauteur on va privilégier cinq niveaux. Le nombre de PIXELs par graduations sera donc de $64 / 5 = 12,8$ qui sera arrondi à 12. En largeur, après divers essais, ce qui semblait le mieux correspondait à une grille dont les éléments sont carrés. En largeur on peut donc en placer $128 / 12 = 10,6$ arrondi à 10. La Fig.72 représente le coté gauche de la mosaïque de l'écran OLED avec ses 64 PIXELs de hauteur numérotés 0 à 63. La plus grande hauteur qui sera disponible pour représenter la courbe du signal sans avoir des points cachés par le cadre qui lui est réservé sera de 12×5 moins un pour la ligne du bas du cadre. *On aura donc une trace qui sera limitée à 59 PIXELs en hauteur.*

P our la largeur on va retrouver une disposition analogue. Les dix graduations de large repérées en verts clair sur la Fig.73 totalisent $12 \times 10 = 120$ PIXELs. Toutefois, la dernière "colonne graduée" voit son coté droit coïncider avec le cadre repéré en violet pastel. La largeur totale du cadre violet fait donc 121 pavés de largeur et il s'étend des verticales de mosaïques numérotées de 0 à 120. Comme il ne servirait à rien de tracer la courbe du signal "derrière" le cadre, cette dernière en largeur n'occupera que 119 luminophores de la colonne n°1 à la colonne n°119.



CONCLUSION : La zone de visualisation de la courbe représentant le tracé sera comprise dans une matrice de 119 PIXELs de large et de 59 PIXELs de haut. Il restera la zone **Z** pour placer l'indication de la tension efficace.

OPTIMISATION de l'ENREGISTREMENT : Chaque colonne de la matrice réservée à la représentation du signal enregistré correspondra à un échantillon. Sur une fenêtre d'écran on ne peut présenter au maximum que 119 points en largeur. Il est donc inutile d'en stocker plus en mémoire. *Un enregistrement en zone RAM ou en mémoire non volatile EEPROM sera composé de 4×119 numérisations soit 476 échantillons.*

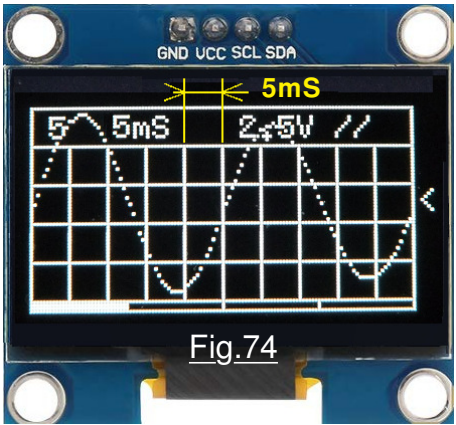


A vec des enregistrements qui ne font plus que 476 échantillons on réduit de 36 OCTETS l'encombrement du tableau de stockage. C'est particulièrement avantageux en mémoire dynamique car on réduit d'autant le risque de collision entre la PILE et le TAS sans compter que les temps de traitements à tous les niveaux seront diminués globalement d'environ 7% ce qui n'est pas totalement négligeable, en particulier pour les écritures en EEPROM. Enfin, ces 36 OCTETS en mémoire non volatile seront les bienvenus pour y loger du texte. C'est tout bénéfice !

Nous pouvons constater en tête du listage de **P12_AFFICHAGES.ino** que l'estimation de la taille des séquences utiles pour l'affichage titille les 4380 Octets de programme et gloutonne environ 14% de l'espace programme disponible. Cette boulimie est assez normale et classique. La génération d'une page écran comme celle de la Fig.65 accompagnée de diverses options impose toujours pas mal de traitements consommateurs de place. Il n'y a donc pas lieu de se formaliser.

➤ **Remise en cause des valeurs de la BASE de TEMPS.**

Lorsque le démonstrateur **P10** a été développé, nous avons choisi des valeurs pour la BASE de TEMPS qui semblaient parfaitement logiques. Hors elles étaient supposées définir la durée entre la saisie de deux échantillons. Il se trouve que l'affichage de la courbe avec une belle grille de repérage des valeurs va chambouler la donne. En effet, *ce qui intéresse l'utilisateur c'est l'intervalle de temps correspondant à un "carré en largeur de la grille"*. Si par exemple comme sur la Fig.74 la valeur de la **BdT affichée** est de 5mS, l'opérateur va en déduire que ce temps représente la durée pour un carré de la grille. Comme dans cet exemple une période du signal occupe 5 carrés, l'usager en déduira que la période de la sinusoïde fait 25mS et que la fréquence de ce signal fait 40Hz. Une **CAN** avec amplification associée à la sauvegarde de la valeur consomme environ 112µS. De ce fait, le plus court délai pour une graduation de la grille sera de 112µS x 12 échantillons entre deux graduation soit environ = 1,34mS. *Les BdT exprimées en µS ne sont plus réalistes.* La cadence d'échantillonnage la plus rapide conduira à une **BdT** de 1.34mS par graduation valeur indiquée par le logiciel. Pour calculer la valeur de la base de temps il faut raisonner sur la largeur complète de la grille soit 121 PIXELs. *La valeur du délai pour respecter les BdT exprimées en mS mais utilisant l'instruction delayMicroseconds() se calcule avec la formule :*



Valeurs_BT = (BdT x 10 graduations x 1000 car µS) / 121 Echantillons sous la grille

Dans ces conditions les intervalles de temps entre la saisie de deux échantillons doivent être recalculées et sont consignées dans le tableau de la Fig.75 qui établit la relation entre la **BdT affichée** et le délai réel à respecter. *La BdT affichée est relative à la durée entre deux graduations horizontales.*

Temporisation entre deux échantillon en fonction de la BdT affichée.								
1.35mS	2mS	5mS	10mS	20mS	50mS	100mS	200mS	500mS
0	165µS	413µS	826µS	1652µS	4132µS	8264µS	16528µS	41322µS
0	1	2	3	4	5	6	7	8

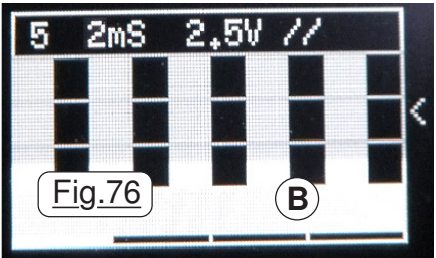
Fig.75

➡ I_BdT

Les **BdT** en secondes ne présentant pas un réel intérêt. Du coup les seules unités qui subsistent sont les **mS**, ce qui va simplifier le logiciel d'application et en particulier la routine d'affichage.

➤ **La refonte des séquences de la BASE de TEMPS.**

Avant de consacrer du temps à la procédure de saisie des échantillons qui sera certainement assez complexe pour la mise en œuvre de la synchronisation, il me semble préférable en préambule de réaliser un autre démonstrateur qui intégrera la procédure simplifiée de l'affichage graphique et la nouvelle séquence du menu de la BASE de TEMPS qui tient compte du tableau de la Fig.75 en optimisant au maximum, on s'en doute, le code de l'ensemble. C'est le démonstrateur nommé **P13_BdT_optimisee.ino** qui se charge de cette mission. Comme il intègre deux protocoles



d'utilisation de l'oscilloscope, il émule un petit **MENU** de BASE et comme **P12**, il génère *une trace fictive qui dans cet exemple simule un signal binaire* qui permet lors des manipulations de mieux cerner

l'avantage du mode "surface". En effet, bien qu'affiché sur la Fig.76 en **A** le signal est totalement masqué par le graticule alors qu'en **B** il est parfaitement "présent". Naturellement, cette fois *les valeurs de la BdT sélectionnées par l'utilisateur sont celles relatives à l'intervalle d'un carré sur le graticule*. Pour optimiser le code, on se contente lors de la sélection de la **BdT** d'en représenter la valeur par un **Indice** nommé **I_BdT**. Cet indice simplifie la visualisation de l'écran graphique et la saisie des valeurs temporelles. Il permettra en outre de sélectionner dans une liste de constantes les valeurs des délais découlant des calculs de la Fig.75 rangés dans un tableau de constantes. Par raison d'homogénéité des commandes dans ce démonstrateur, le retour au **MENU de BASE** se fait avec la touche DROITE en *clic long* pour les deux sous-menus. La simplification de la gestion du temps n'ayant que des **BdT** indiquées en **mS** et l'abandon des durées supérieures à 500mS engendre un gain de place d'environ 1760 octets ce qui est considérable. Évolution du logiciel adoptée !

14) La capture des échantillons.

Fondamentalement, si la synchronisation est activée, la saisie d'un enregistrement complet revient à attendre que les conditions du déclenchement soient satisfaites, puis à enchaîner 476 mesures avec stockage des valeurs dans un tableau dynamique. *Entre chaque CAN on insèrera un délai défini dans la base de temps*. Si l'on a choisi la vitesse la plus grande avec une **BdT** affichée de **1.34mS** les numérisations s'enchaîneront sans intercaler de temporisation.

➤ Table des délais pour la BASE de TEMPS.

L'opérateur doit pouvoir faire confiance aux données temporelles affichées à l'écran c'est à dire que la durée entre deux graduations sur l'écran doit correspondre à la base de temps sélectionnée. La théorie annoncée ci-avant précise que l'on doit intercaler *un délai défini dans la base de temps*. Ce n'est pas si immédiat que ça, car il faut tenir compte du temps consommé par la **CAN** et celui nécessaire au stockage des données. En s'appuyant sur la Fig.77 on va pouvoir facilement déterminer le délai précis à intercaler entre chaque saisie d'échantillon. Construire la base de temps revient à confier dans un tableau la valeur des délais qui seront imposés au microcontrôleur entre deux stockages de données numérisées. Le principe du calcul des valeurs de temporisation est relativement simple et résumé dans l'encadré ci-contre.

$$\text{Valeur_BT} = T1 + T2 + T3 + T4 + T5 + T6$$

Les diverses valeurs sont résumées dans le tableau de la Fig.75 et le programme doit calculer à partir de la base de temps désirée, le délai **T4** en μS qui ajustera avec précision le temps passé entre deux saisies d'échantillons. La formule de calcul est simple :

$$\text{TBT} = T4 = \text{Valeurs_BT} - (T1 + T2 + T3 + T5 + T6)$$

Les divers essais destinés à mesurer le temps de traitement pour (**T1** + **T2** + **T3** + **T5** + **T6**) ont donné comme résultat **112 μS** . Comme les constantes dans **Valeurs_BT** sont consignées en μS , le calcul de **TBT** devient :

$$\text{TBT} = (\text{Valeur_BT} * 1000 / 13) - 112;$$

Naturellement, comme il s'agit de constantes, nous allons les calculer une fois pour toutes et les consigner dans un tableau d'entiers. Avant d'activer une numérisation il suffira de faire appel à l'instruction **TBT = Valeurs_BT[I_BdT]**; et dans la boucle de saisie des 476 échantillons se contenter de la temporisation **delayMicroseconds(TBT)**. On aboutit au tableau de la Fig.78 qui résume la liste des valeurs des délais.

$$\text{TBT} = \text{Valeurs_BT}[\text{I_BdT}];$$

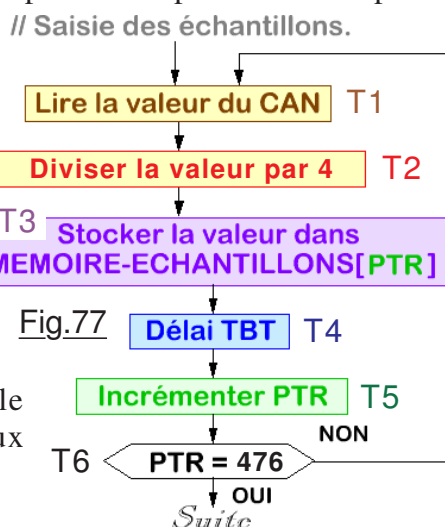


Fig.78

Temporisation entre deux échantillon en fonction de la BdT affichée.								
1mS	2mS	5mS	10mS	20mS	50mS	100mS	200mS	500mS
1 μS	53 μS	301 μS	714 μS	1540 μS	4020 μS	8152 μS	16316 μS	41210 μS
0	1	2	3	4	5	6	7	8

16383 μS

I_BdT

Une limite logicielle à considérer : Lorsque l'on consulte la documentation d'utilisation de l'ATmega328 programmé en C++, il est clairement stipulé que l'instruction `delayMicroseconds()` est limitée à une valeur maximale de **16383** car elle doit probablement utiliser un compteur codé sur 14 BITS. Du coup la **BDT** de **500mS** pose un problème. Pour toutes les autres durées on peut utiliser directement l'instruction `delayMicroseconds()`, alors que pour **500mS** il faudrait la combiner avec `delay()`. Comme une durée d'échantillonnage aussi importante ne présente que très peu d'intérêt en électronique, autant l'abandonner pour simplifier le logiciel, raison pour laquelle elle est barrée sur les Fig.75 et Fig.78.

➤ Une sérieuse refonte du MENU de BASE.

Ayant bénéficié d'un nombre d'essais opérationnels considérable, c'est le démonstrateur **P14_Enregistrer les Echantillons.ino** qui vient chambouler les décisions et les choix initiaux qui ont été faits durant l'étude des démonstrateurs précédents. En effet, à l'usage les très nombreuses manipulations ont démontré que le menu pour la BASE de TEMPS n'était pas du tout pertinent, car *c'est durant la phase de numérisation des échantillons qu'il faut pouvoir en temps réel modifier la valeur de la BdT*. Les expériences ont également fait apparaître les actions primordiales et celles qui sont moins importantes. On peut résumer avec les constats suivants :

- Augmenter ou diminuer la valeur de la BASE de temps est prioritaire.
- Passer en mode AUTOMATIQUE ou MANUEL est prioritaire.
- Déclencher des numérisations en mode MANUEL est prioritaire.
- Modifier le gain en Entrée est secondaire.
- Afficher ou non le graticule est secondaire.
- Synchroniser ou déclencher sans contrainte est prioritaire.
- Revenir au MENU de BASE est secondaire.

En Fig.79 les options secondaires sont surlignées en jaune.

Comme nous avons huit options à assurer et seulement quatre touches sur le petit clavier, nous allons déclencher les options prioritaires par un *clac court* pour des raisons de convivialité, et les directives secondaires avec un *clac long*. La répartition des options est montrée sur la Fig.79 sur laquelle on notera que les touches "verticales" HAUT et BAS s'utilisent pour augmenter ou diminuer la **BdT** ou gérer le gain en tension. Les touches horizontales GAUCHE et DROITE pour leur compte servent à gérer le déclenchement. Lorsque le mode MANUEL est validé, toutes les touches peuvent déclencher une numérisation sauf DROITE réservée pour pouvoir revenir en mode AUTOMATIQUE. Du coup les trois touches "du côté gauche" située dans l'encadré bleu perdent leur spécificité. Pour en avertir l'opérateur la LED triple s'illumine en violet. Dès que l'on retrouve le mode AUTOMATIQUE elle clignote au rythme des **CAN** et les trois touches servent à nouveau à initialiser les contraintes de déclenchement.

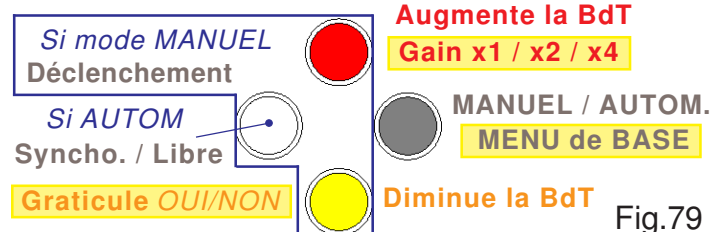


Fig.79

➤ Le "sapin de Noël".

Festival lumineux, il était temps de commencer à utiliser les diverses LEDs installée sur le prototype. C'est le démonstrateur **P14** qui inaugure l'utilisation de cette optoélectronique. Les deux premières LEDs à mentionner sont les deux petits témoins de diamètre 3mm. La petite LED rouge s'allume dès que la trace arrive tout en haut du graticule car il y a soupçon de débordement par saturation. La petite LED verte s'illumine dès que la **TRACE** vient contre le bas du graticule car il peut y avoir saturation par débordement par tension négative.

Dans le **MENU de BASE** la LED triple éclaire en bleu signalant qu'il y a attente sur l'une des quatre touches pour déclencher une fonction. Dès que l'on active la fonction **Afficher TRACE** la LED triple passe du bleu au vert, toujours pour indiquer qu'il y a attente sur l'une des quatre touches pour modifier l'affichage. Quand on active **Numériser**, la composante verte s'illumine durant chaque numérisation. Quand on active le mode MANUEL la LED triple s'illumine en violet pour préciser à l'opérateur que les trois touches encadrées en bleu sur la Fig.79 déclenchent une Numérisation et n'influencent plus la **BdT** et le gain en tension. (*Passe en vert/rouge durant la CAN.*)

15) Comportement du démonstrateur P14.

C hamboulant complètement les expérimentations précédentes, il présente déjà un **MENU de BASE** qui pourrait bien être définitif et qui anticipe sur les possibilités futures de l'appareil. Les deux fonctions **Menu EEPROM** et **Sortie USB** n'étant que des prévisions potentielles si la place dans le programme le permet, actuellement les touches concernées provoquent un BIT sonore d'erreur. Passons en revue le comportement du programme pour les fonctions de base.

BASE de TEMPS
Durée : 1.35mS
Total : 0mS **(A)**
Fenêtre : 0mS Fig.80

BASE de TEMPS
Durée : 200mS
Total : 8S **(B)**
Fenêtre : 2S

➤ **RESET avec un B.P. activé.**

A ctuellement une seule option sur RESET est envisagée. Du coup n'importe laquelle des quatre touches provoquera l'effet qui **RESET avec un BP activé** invoque le menu d'affichage des durées d'échantillonnage en fonction de la BASE de TEMPS sélectionnée. On modifie librement la **BdT** pour faire afficher les durées totalisées par la mémorisation d'une **TRACE** complète. Mais en sortie la cadence de 112µS la plus rapide est initialisée soit une **BdT** de 1.34mS. Les Fig.79 **A** et **B** présentent les pages-écran des durées minimales et maximales. Durant cette fonction la touche du HAUT augmente la valeur, celle du BAS la diminue en permutation circulaire. GAUCHE génère un BIP sonore d'erreur, alors que DROITE ouvre le **MENU de BASE**.

➤ **L'amplification durant l'échantillonnage.**

A ugmenter par deux ou par quatre le gain en tension en entrée sans avoir à intercaler un amplificateur quelconque est particulièrement avantageux. Nous avons vu Fig.16 en Page 10 lors du chapitre sur la mémorisation des échantillons, que pour pouvoir enregistrer les données sur un OCTET il fallait diviser la valeur par quatre ce que l'on obtenait avec deux décalages (**SHIFT**) à droite. Toutefois, si la tension en entrée ne dépasse pas 2,5V en ne divisant que par deux, la "portée" restera codée sur un **byte**. Par ailleurs, si la trace ne dépasse pas 1,25V crête, on peut même ne plus diviser, tout en ayant une valeur qui reste inférieure à 255. Il suffit tout simplement de réserver une touche pour sélectionner x1, x2 et x4 et **fonction du gain choisi faire deux décalages à droite, un seul ou aucun.**

Au lieu d'indiquer la valeur de l'amplification en **1**, il est bien plus facile pour l'interprétation de préciser la tension à pleine échelle en hauteur sur le graticule. Ensuite en **2** c'est la valeur de la **BdT** par graduation qui est visualisée. Pour finir, en **3** est affichée la valeur de la tension efficace évaluée sur la "largeur de la fenêtre" avec en **4** le curseur représentatif de cette dernière.

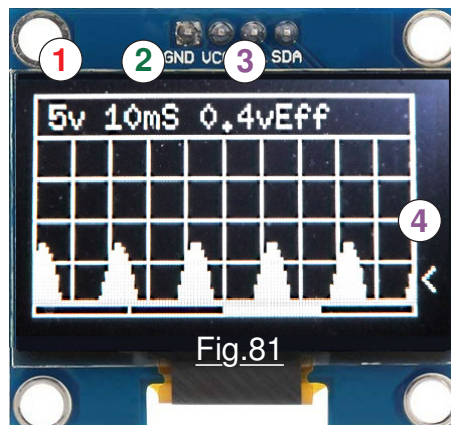


Fig.81

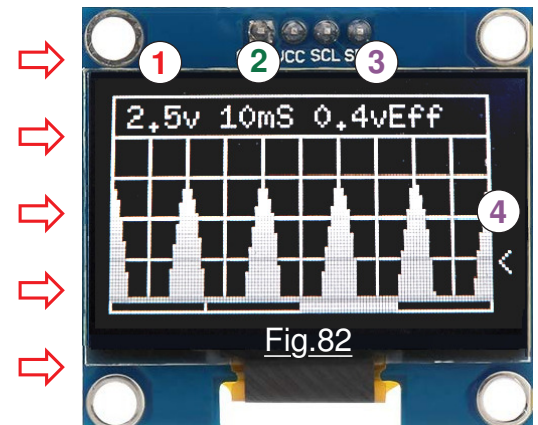


Fig.82

M éfiance, cette technique simple à mettre en œuvre présente toutefois un piège potentiel à l'utilisation. En effet, on divise par deux et par quatre la valeur numérisée sur dix BITS pour la stocker sur un **byte**. Considérons la Fig.83 pour raisonner sur un exemple concret. En **A** est symbolisée la numérisation sur 10 BITS avec **les poids faibles à droite et les poids fort à gauche**. La tension présente fait 2V qui numérisée donne 1023 / 5 x 2 soit 409 en décimal. La conversion analogique sur 10 BITS va donc fournir cette valeur 409 en BINAIRE. Puis, pour stocker sur un **byte** on effectue deux décalages à droite soit la configuration sur la Fig.83 représentée en **A**. La valeur numérisée est de 102. L'instruction **map()** conduira pour le calibre **5v** à une déviation verticale de 59 / 255 x 102 soit ≈ 24 PIXELs. Avec l'amplification par deux on ne réalise qu'un seul décalage. La valeur qui sera mémorisée est donc celle en **B** avec

A 0 0 0 1 1 0 0 1 1 0
B 0 0 1 1 0 0 1 1 0 0

Fig.83

Page 39

une valeur binaire qui vaut maintenant 204 toujours compatible avec un **byte**. Le bit de poids faible "0" pouvant être un "1" ne changera pas le résultat final car il sera "éliminé" lors de la transposition. L'instruction **map()** conduira pour le calibre **2.5v** à une déviation verticale de $59 / 255 \times 204 = 47$ PIXELs. Pour établir la relation qui existe entre toutes ces valeurs numériques, considérons la Fig.84 qui résume le cas d'une mesure sans amplification en **1** et **2**, et celui d'une **CAN** avec un seul décalage

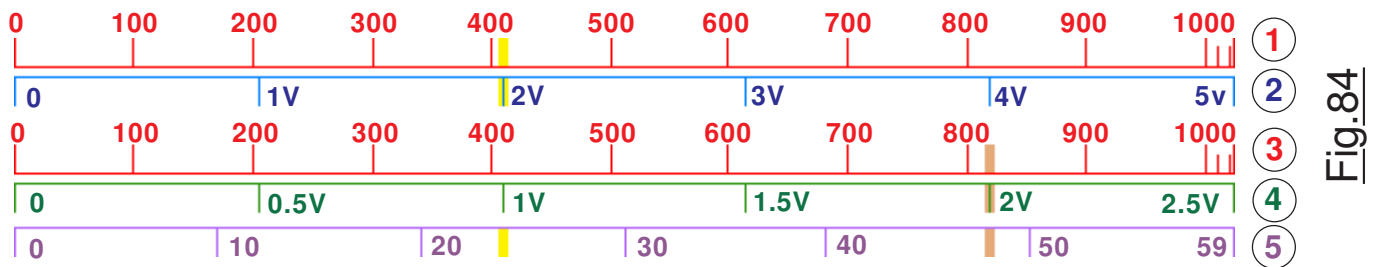


Fig.84

à droite en **3** et **4**. Sur le calibre **5v** en **2**, la tension de 2Vcc repérée en jaune donne la **CAN** = 409. Cette dernière transposée par **map()** engendrera sur la **TRACE** verticale **5** une déviation de 24 PIXELs. La déviation est bien de deux graduations sur le graticule. Amplification par deux avec un seul décalage, la **CAN** donne la valeur double de 818 qui est stockée avec une grandeur de 204. Elle conduit à une transposition de $59 / 255 \times 204$ soit ≈ 47 PIXELs. *Et alors ?*

Supposons maintenant que l'on reste en amplification de deux, mais que la tension soit de 3v au lieu de deux, c'est à dire qu'elle dépasse le calibre maximal de **2,5v**. La **CAN** va fournir une valeur de $1023 / 5 \times 3$ soit 613 en décimal qui sur la Fig.85 est représentée en **A**. Le décalage à droite se traduit par la configuration **B**. C'est ici que se produit le problème. En effet comme le stockage ne s'effectue que sur un OCTET, on ne prend que les BITS situés dans les cellules coloriées en bleu.

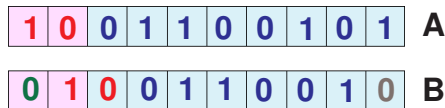


Fig.85

La valeur mémorisée sera alors égale à 50 ce qui revient à ne prendre que ce qui dépasse de la grille ! Cet effet est illustré sur la Fig.86 pour laquelle l'amplification x2 a été volontairement appliquée alors que la tension crête dépassait le seuil de **2,5v**. Il s'agit d'un montage graphique "artificiel" du côté gauche en **A**, proposant une simulation de ce qui devrait s'afficher si la

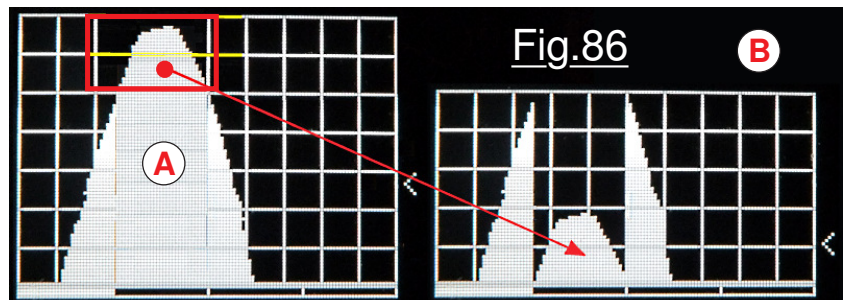


Fig.86

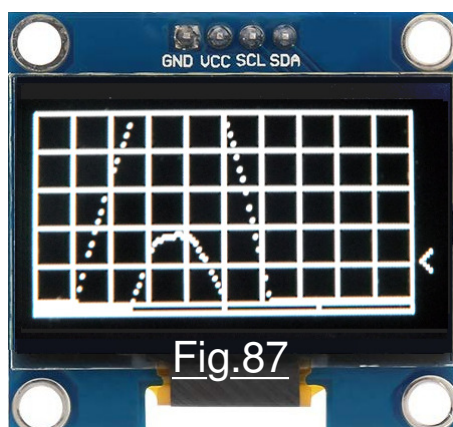


Fig.87

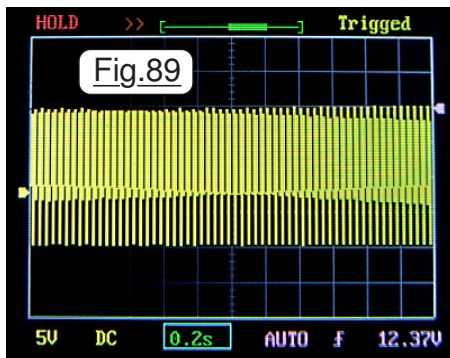
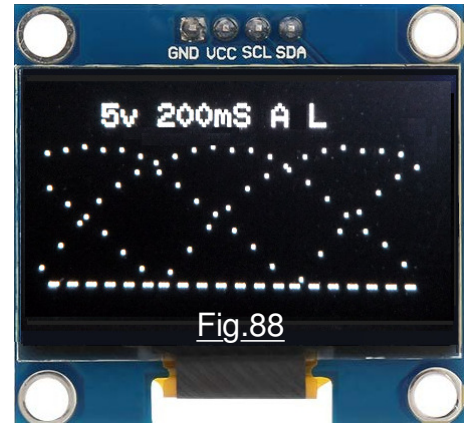
hauteur de l'écran était plus importante et que l'incident ne se produisait pas. En **B** est reproduite la réalité ou tout ce qui se trouve dans le graticule est remplacé par ce qui se trouverait au dessus. L'arche de sinusoïde n'est plus du tout affichée correctement. Sur la Fig.87 l'aléa est strictement analogue, sauf que sur cette copie d'écran pour la visualisation de la TRACE l'option "filaire" remplace la représentation de la courbe en mode surface. Une conclusion importante doit être tirée de ces manipulations et s'impose relative à l'utilisation de l'appareil :

ATTENTION : *On ne doit utiliser l'amplification* par deux, c'est à dire le calibre 2,5V *que si le signal pour toute la durée mémorisée ne dépasse pas la valeur du calibre*. On ne doit utiliser l'amplification x4 que si le signal pour toute la durée mémorisée ne dépasse pas 1,25v. C'est pour faciliter l'observation des dépassements de ces deux seuils que par défaut c'est le calibre **5v** qui est initialisé dans l'ouverture de la fonction **Numériser**.

Au passage, pour clore ce chapitre ... j'ai KKkkrrrraké ! Dans la galerie d'images, comparer **Image 21.JPG** avec **Image 22.JPG** car la solution de la Fig.29 n'était vraiment pas commode. Outre que l'assemblage manquait de "raideur", pour déplacer l'ensemble au laboratoire de mesures (*Qui abrite des appareils comme le fréquencemètre.*) c'était un peu une galère.

➤ Une base de temps mal sélectionnée.

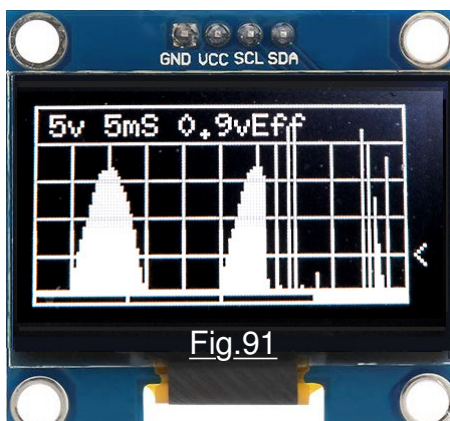
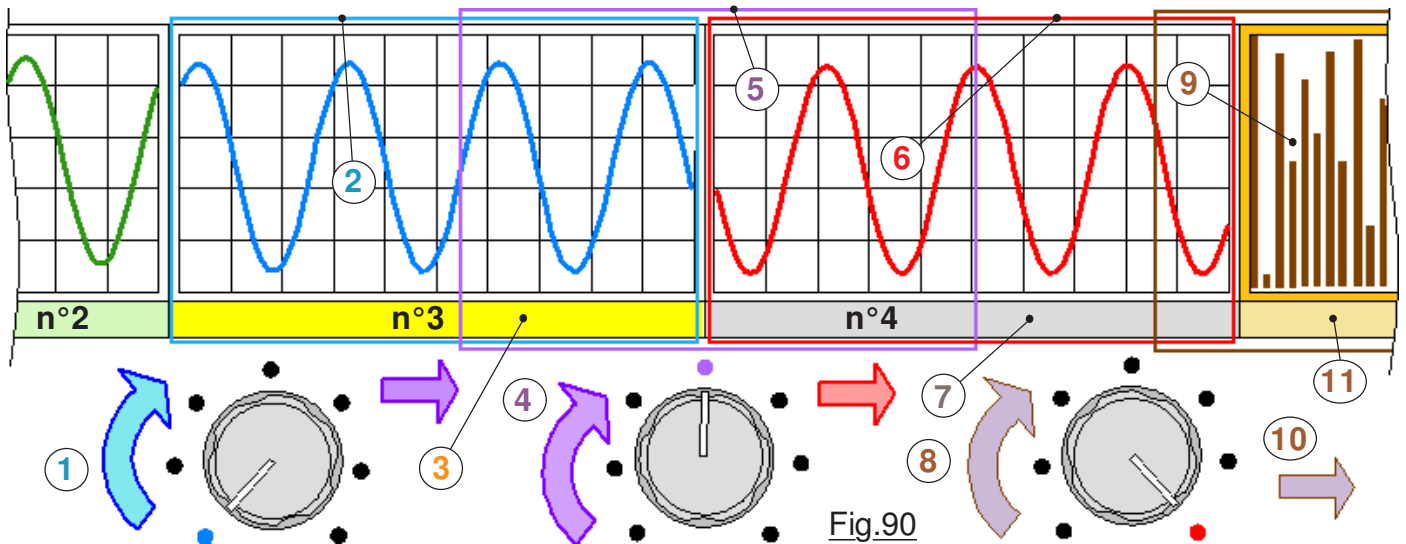
Considérons la Fig.88 qui présente une TRACE pour le moins étrange. Elle donne l'impression que trois sinusoïdes s'imbriquent sur la largeur de l'écran. L'affichage serait encore plus troublant en mode surface et présence du graticule. Ce que l'on observe n'est pas du tout un incident inattendu ou un problème dans le logiciel. Ce phénomène qui a été décrit Fig.14 en Page 9 se produirait sur tout autre oscilloscope numérique. La preuve en est apportée sur la Fig.89 qui est l'image obtenue sur un petit appareil du commerce. Les conditions sont similaires, mis à part le fait que la sinusoïde est complète et que le graticule fait 12 graduations sur la largeur.



CONCLUSION : Pour éviter des représentations du signal erronées, toujours partir d'une base de temps de valeurs faible et augmenter la durée entre deux échantillonnages progressivement durant la fonction Numériser.

➤ Décalage latéral de la TRACE.

Permettre à l'utilisateur de décaler latéralement le signal affiché est un impératif. En effet, lorsque l'on veut évaluer la durée d'une portion du signal pour en déduire la période et la fréquence, il faut pouvoir librement aligner l'une des transitions sur le graticule, ce qui est indispensable quand la fonction Afficher TRACE est activée. La technique utilisée est très simple, il suffit de faire tourner le potentiomètre pour décaler la trace vers la gauche. En position 0 la fenêtre n'est pas décalée. (Voir la Fig.90) Supposons que l'on a validé avec le curseur 3 la fenêtre



n°3 et que le potentiomètre soit en butée antihoraire. On visualise la courbe contenue dans la fenêtre repérée par le cadre 2. On tourne dans le sens horaire 1 la fenêtre se déplace vers la droite. Quand le curseur est vers le haut, on a balayé la moitié de l'amplitude possible de rotation. La fenêtre de visualisation repérée par le cadre 5 est à moitié sur n°3 et à moitié sur n°4. On continue en 4 de tourner le bouton flèche. Lorsque la pleine rotation est appliquée, le décalage vers la droite est équivalent à la largeur d'une fenêtre, ce qui revient à afficher la section voisine n°4 repérée dans le cadre rouge 6. Un aléa se produit si on a indexé avec le curseur en 7 la fenêtre n°4. Quand on tourne

en 8 le potentiomètre, le logiciel va aveuglément chercher à droite 10 dans la mémoire dynamique les OCTETS en 9 qui suivent, comme s'il y avait une autre fenêtre colorisée ici en orange avec le curseur virtuel 11. Hors ce sont des données du programme qui contiennent des valeurs quelconques comprises entre 0 et 225. Du coup, on voit sur la grille arriver comme sur la Fig.91 des affichages "quelconques". *Ce n'est pas du tout un problème majeur, il suffit d'y penser.* Noter que la saisie de l'orientation du potentiomètre donne une CAN qui peut fluctuer à ± 1 comme pour toute valeur numérique issue d'une mesure. Il en résulte parfois une instabilité latérale qui peut agacer. Dans ce cas, il suffit d'utiliser la touche HAUT avec *clic long* pour activer ou suspendre le décalage latéral.

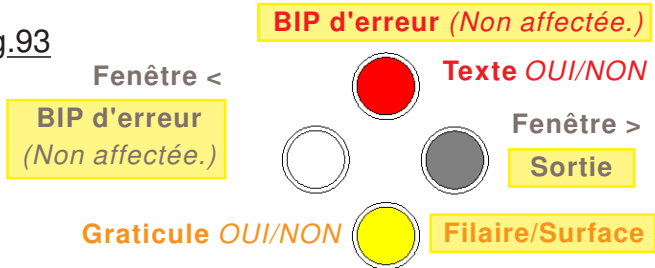
➤ La fonction Afficher TRACE.

Participant indubitablement aux performances de l'appareil, on a vu en Page 35 qu'elle influence directement le nombre d'échantillons mémorisés. Lors des études initiales en Page 34 on se doutait que les choix effectués étaient provisoires, et effectivement l'utilisation du clavier est un peu différente tout en reprenant la notion de *clic court* et de *clic long*. Annoncé en début de ce chapitre, sur RESET c'est la page écran de la Fig.92 qui est visualisée. C'est le MENU de BASE provisoire qui est réactivé chaque fois que l'on sortira des fonctions invoquées et en particulier pour Afficher TRACE. Pour cette fonction la nouvelle répartition des actions est résumée sur la Fig.93 avec surligné en jaune le actions déclenchées par un *clic long*.



Fig.92

Fig.93



➤ Le respect des durées de la BASE de TEMPS.

C'est une caractéristique importante de tout oscilloscope. Il faut impérativement que l'on puisse faire confiance aux BdT affichées en 1 et à la correspondance en latéral sur le graticule. La théorie énoncée en Page 37 c'est bien. Elle justifie les algorithmes utilisés et les constantes référencées dans le programme. *Il reste incontournable de vérifier avec précision le comportement logiciel pour valider avec fiabilité le bienfondé du codage.* Dans ce but un générateur de signaux "carrés"

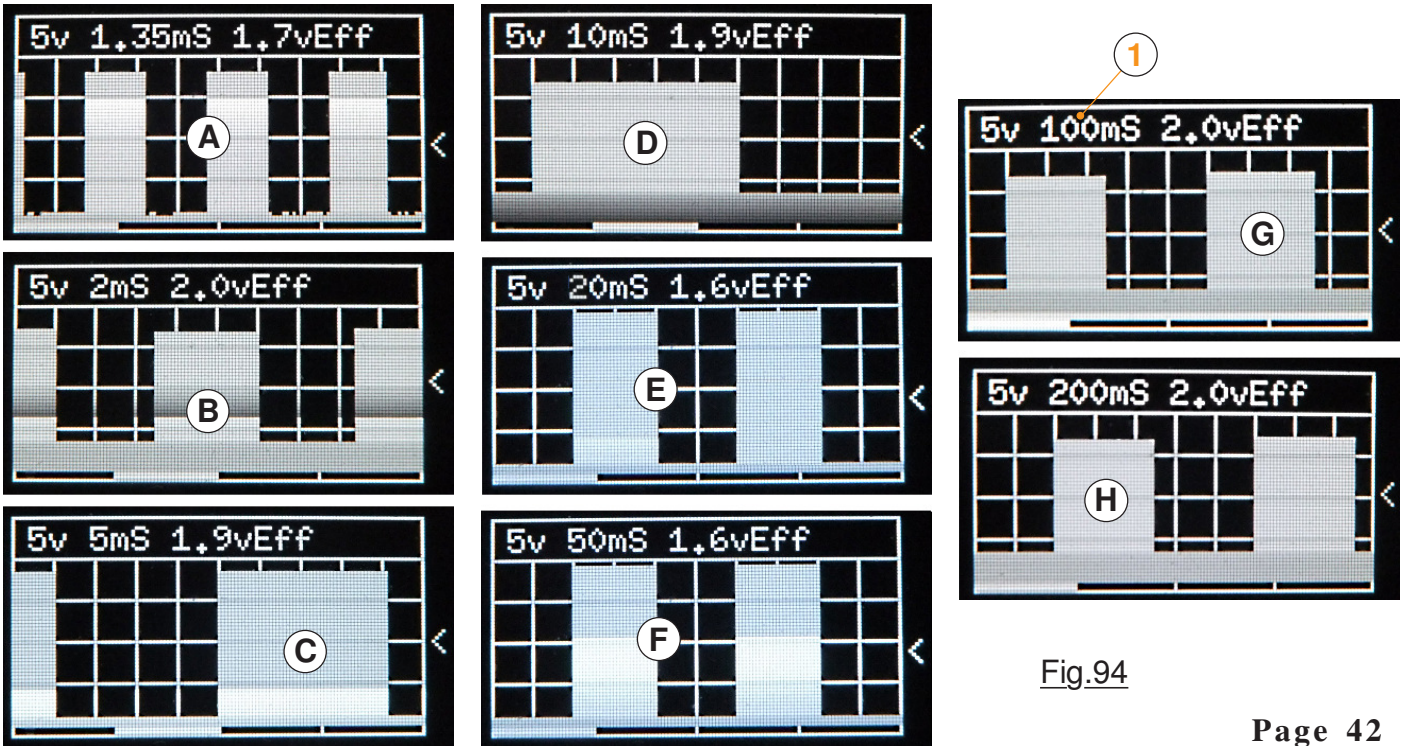


Fig.94

extrêmement précis piloté par un oscillateur quartz thermostaté a été mis en œuvre. Les huit calibres de la base de temps ont été vérifiés avec les valeurs consignées sur le tableau de la Fig.95 avec les divers essais présentés sur la Fig.94 en **A, B, C, D, E, F, G** et **H**.

Photographie	Calibre	Période	Fréquence
A	1.35mS	4mS	250Hz
B	2mS	10mS	100Hz
C	5mS	40mS	25Hz
D	10mS	100mS	10Hz
E	20mS	80mS	12.5Hz
F	50mS	200mS	5Hz
G	100mS	1S	1Hz
H	200mS	1S	1Hz

Les fréquences générées pour le signal analysé sont précises à 10^{-7}

Fig.95

Remarquez au passage sur la Fig.93 que la fonction d'affichage de la trace a été simplifiée. En effet, indiquer les conditions de synchronisation n'est pas pertinent, cette information n'apportant aucun renseignement particulier sur le signal électrique enregistré. Du coup, maintenant la touche GAUCHE n'est plus affectée sur *clac long*.

➤ Les options de Déclenchement.

Contrairement à ce qui avait été envisagé initialement lors de l'élaboration du démonstrateur **P11**, le traitement des options de déclenchement ne fait plus partie du **MENU de BASE** car l'expérimentation montre que c'est durant la saisie des échantillons qu'*il faut pouvoir modifier les conditions "en temps réel"* sans avoir à sortir de la fonction. C'est donc, comme montré sur la Fig.99 avec la touche de GAUCHE en *clac long* que l'on ouvre ce sous-menu lorsque la fonction d'échantillonnage est en cours. L'affectation des touches de la Fig.96

ignore la notion de *clac court* ou de *clac long* et la sortie se fait de

façon "standard" en utilisant le bouton poussoir DROIT. L'entrée dans cette fonction allume dans la LED tricolore la composante

bleue pour inviter à utiliser l'une des touches du mini clavier.

Lorsque l'on clique en **1** sur la touche du HAUT ou celle du BAS

on modifie la nature du **Front** qui engendrera le déclenchement. Haut valide une variation **Montante**

affichée **FM** en **2** alors que BAS valide le front **Descendant** qui est indiqué par **FD**. Cette transition

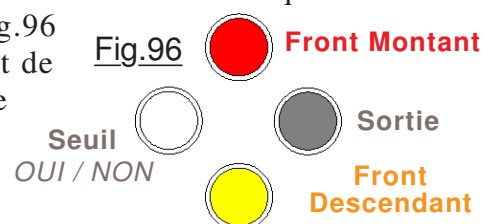


Fig.96

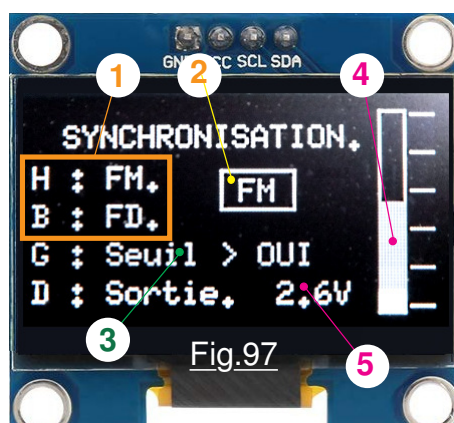


Fig.97

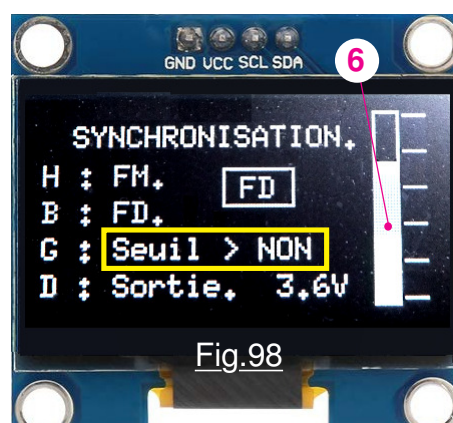


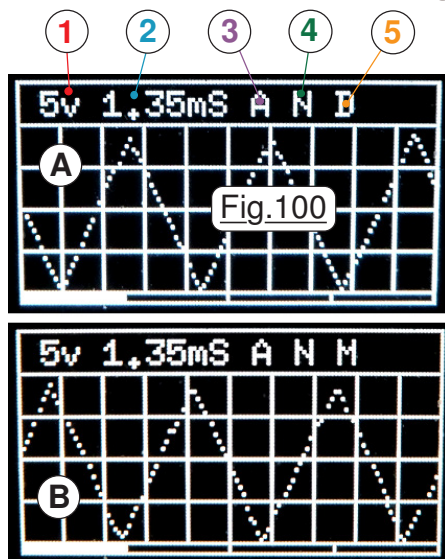
Fig.98

suspendre le niveau de **Seuil** de déclenchement. Quand on suspend cette option comme sur la Fig.98 par exemple, la synchronisation ne tient plus compte de la valeur de la tension mais déclenche la numérisation dès que la tension est au dessus ou en dessous de +2.5V en fonction du Front sélectionné.

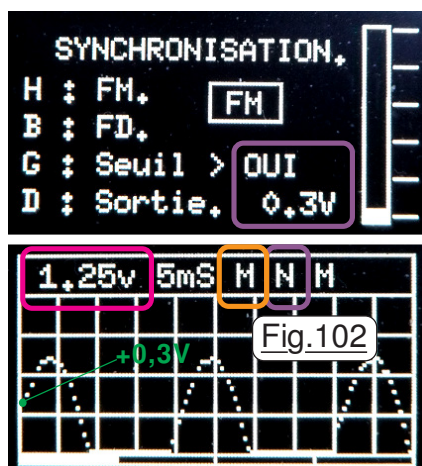
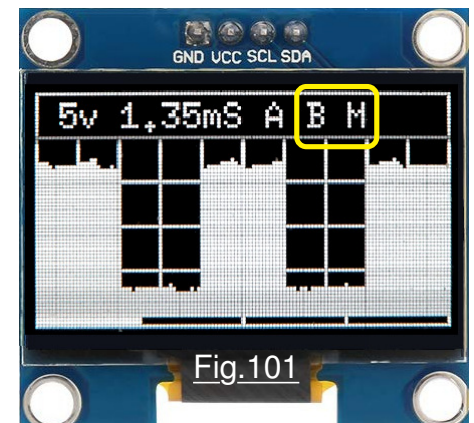
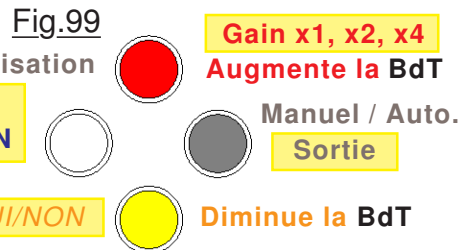
Ignorer le seuil de déclenchement est à utiliser quand le signal est de type binaire et de type TTL pour pouvoir déclencher sur une transition "verticale". Remarquer sur la Fig.98 que le niveau en **6** a été modifié et que le déclenchement se réalisera sur une variation descendante. Dès la sortie de ce menu les contraintes sont immédiatement prises en compte saut en mode **Libre**.

➤ La fonction Numériser.

C'est la plus "charnue" car elle intègre à la fois le sous-menu pour choisir les contraintes de déclenchement, permet d'ajuster le gain en entrée en x1, x2 et x4, autorise le choix entre le mode MANUEL ou AUTOMATIQUE, et octroie l'ajustement de la base de temps. Autant dire que le petit clavier est Utilisé à saturation, la Fig.99 résumant l'affectation des touches. En standard c'est DROITE avec un **clik long** qui fait sortir de la fonction et ramène au MENU de BASE. Déjà précisé dans un chapitre précédent, quand on active Numériser, la composante verte de la LED tricolore s'illumine durant chaque numérisation. Quand on active le mode MANUEL la LED triple s'illumine en violet pour préciser à l'opérateur que les trois touches encadrées en bleu sur la Fig.79 déclenchent un échantillonnage et ne peuvent plus influencer la BdT ni le gain en tension. (Passe en vert/rouge durant la



numérisation des 256 échantillons.) Quand on active le sous-menu de choix des options de synchronisation, la composante bleue s'illumine incitant à utiliser le petit clavier. *Le mode "Filaire" ou Surface défini lors de l'affichage de la TRACE est conservé durant la fonction de saisie des valeurs et d'affichage de la courbe.* Considérons la Fig.100 qui montre deux cas de saisies d'échantillons. Un signal triangulaire est injecté en entrée de l'appareil. On remarque que *c'est la fenêtre de gauche n°1 qui est imposée en entré de la fonction échantillonnage* et que l'on retrouvera en sortie. En 1 est indiqué la valeur de l'échelle verticale. En 2 on retrouve la BdT. En 3 la lettre 'A' signifie Automatique, alors que 'M' précise le mode Manuel. En 4 la lettre 'L' indique le mode Libre. Le caractère 'N' signifie qu'un Niveau de Seuil est validé. Enfin, un 'B' implique une technologie Binaire avec par convention un niveau TTL. (TTL pour Transistor/Transistor/Logic fonctionnant entre 0 et +5Vcc.) Enfin en 5 est affiché la nature du front de déclenchement soit Descendant comme en A ou Montant comme en B. Dans les deux cas le Seuil de synchronisation a été ajusté à +2,5V. Sur la Fig.101 le signal injecté est de type Binaire "accroché" en Automatique sur un front Montant. Sa période a été ajustée à exactement $4 \times 1.35 = 5.4\text{mS}$. On en déduit que la fréquence est d'environ 185 Hz. Abordé dans le chapitre intitulé *L'amplification durant l'échantillonnage* on a vu qu'il était possible avec HAUT en **clik long** de modifier le gain en entrée en permutation circulaire x1, x2 et x4. À titre d'exemple



c'est le cas sur la Fig.102 qui montre un signal alternatif dont les "arcs négatifs" sont éliminés par la protection électronique. Les échantillons sont saisis ici en mode Manuel pour changer. Le déclenchement reste synchronisé à un Seuil de 0,3V sur un front Montant. Sur cette photographie on peut vérifier que l'amplitude verticale à pleine échelle est choisie à 1,25V, c'est à dire avec l'amplification x4. Chaque graduation est équivalente à 0,25v et l'on déduit que la tension crête avoisine 0,6v. Le point de déclenchement à gauche est un peu au dessus de la graduation du bas, la tension de déclenchement correspond bien à celle qui a été consignée.

ATTENTION : Si la tension de seuil est supérieure au maximum du signal le système reste figé jusqu'à injection d'un signal de tension suffisante ou activation d'un RESET.

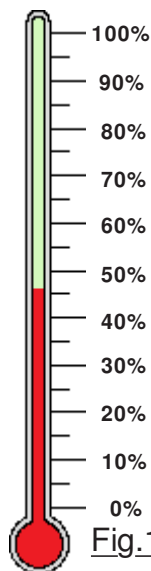


Fig.103

➤ Un petit "bilan financier".

L'observation du listage du démonstrateur **P14** à la compilation nous informe que le code objet pèse lourd, car il gloutonne 14304 Octets soit **46%** de l'espace réservé au programme. La Fig.103 représente un thermomètre dont la colonne rouge représente l'espace mémoire dédiée au programme actuellement consommée. C'est parfaitement normal, voir très raisonnable, car ce croquis est bien plus qu'un simple démonstrateur car il émule complètement l'Oscilloscope et intègre toutes les fonctions fondamentales. De plus le **MENU de BASE** est déjà entièrement en place, sans compter que sur RESET on peut faire afficher les durées d'échantillonnage en fonction de la **BdT**. Par ailleurs les textes sont ici en mémoire de programme et non en EEPROM. Alors nous pouvons nous réjouir, car il reste une place considérable pour améliorer l'ordinaire et ajouter à profusion de petits perfectionnements et des fonctions secondaires. Nous n'allons pas nous en priver.

➤ Stratégie d'utilisation de la bibliothèque U8glib.

Livret Bibliothèque **U8glib** en main, on notera en bas de **P4** que chaque police de caractère encombrera la RAM dynamique. Il ne faut donc pas s'étonner qu'actuellement le programme **P14** consomme 46% de l'espace vital. Ce n'est pas dramatique. D'un autre coté, si l'on change plusieurs fois de police de caractères, ce sera chaque fois du code qui s'ajoute. Aussi, comme l'on affichera que des textes "ordinaires", *on n'utilisera qu'une seule police de caractères pour tout le programme*. Par ailleurs, il aurait été tentant d'exploiter les beaux curseurs de la page **P9** du petit livret. Toutefois, il suffit de déclarer la "fonte réduite" et de faire afficher une fois le curseur pour que le programme enfle de 100 octets. *On se passera de la possibilité d'afficher des curseurs graphiques*. Par ailleurs, si possible *on minimisera le nombre des outils graphiques utilisés*. (Dans cette application les triangles ne s'imposent pas par exemple.) Le plan budgétaire étant approuvé, on peut passer à la suite du développement du logiciel. L'exploitation de l'appareil sera nettement plus agréable si l'on peut sauvegarder une TRACE en EEPROM. On s'y colle !

16) Sauvegarde et récupération des enregistrements en EEPROM.

L'argument défriché avec le démonstrateur **P09_Tester_EEPROM.ino** on va de façon totalement analogue commencer par inscrire les textes en EEPROM car maintenant ces derniers commencent à se stabiliser et les divers menus y compris le **MENU de BASE** sont définis. Du coup on va se servir du programme **P00_Textes_en_EEPROM.ino** qui de plus comme son prédécesseur place en haut de mémoire une TRACE cohérente à utiliser pour tester les routines de récupération. Cette fois comme exemple on reprend la sinusoïde amortie de **P12**. Vous avez parfaitement compris qu'avant de tester **P15_SAV_en_EEPROM.ino** il faudra téléverser **P00** et l'activer avec le Moniteur de l'**IDE**, la vitesse sur ce dernier étant initialisée en 57600 baud.

Quand **P12** a été développé, dans un premier temps la seule modification apportée au code a consisté à ajouter les routines d'exploitation de l'EEPROM et d'utiliser les textes inscrits par **P00** dans cette dernière. Comme précisé en tête de listage le programme qui "pesait" 14304 octets soit 46% de l'espace dédié à diminué de 278 octets passant de 46% à 45% sur le thermomètre. Les données dynamiques quand à elles chutent de 234 octets. Ne consommant que 37% de la place prévue autant dire que coté collision de PILE il n'y a rien à craindre. (Le problème de la "collision de PILE est abordé plus avant dans le tutoriel.)

Lorsque dans le **MENU de BASE** on clique sur la touche de GAUCHE *clic court* ou *clic long* on ouvre le **MENU EEPROM** de la Fig.104, la LED triple passant alors du bleu au vert pour mettre en évidence ce changement de page de commande et inciter l'opérateur à utiliser le clavier. Un changement important a été porté à l'utilisation de la mémoire non volatile dont la Fig.105 décrit la nouvelle organisation. Noter que cette copie de la fenêtre d'écran du Moniteur de l'**IDE** a été effectuée lors de l'utilisation de **P00** du 27 Novembre 2023 en 1. La suite du développement engendrant de nouveaux textes à afficher, dans la

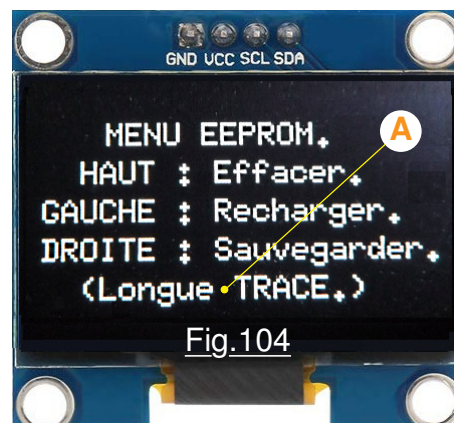

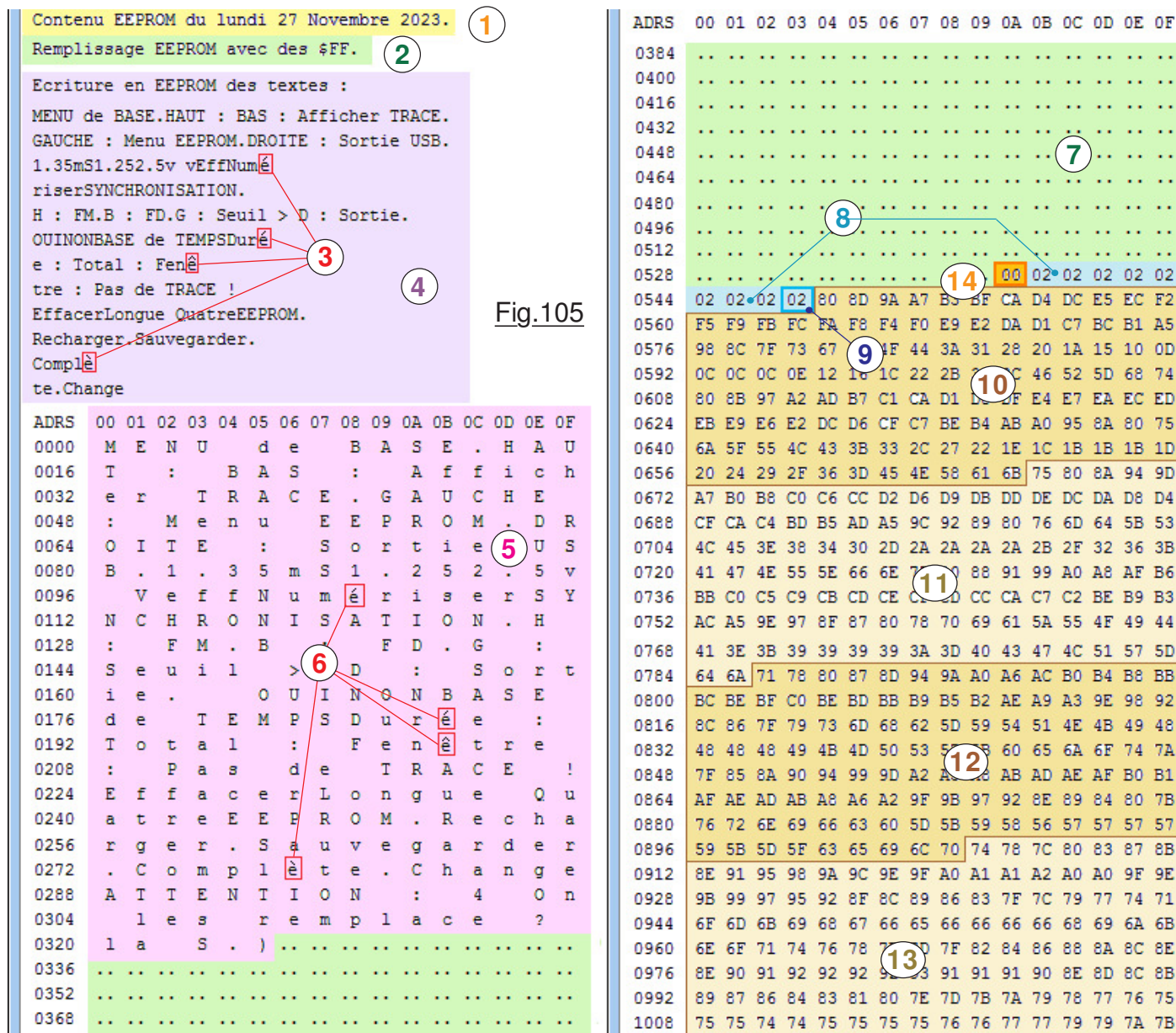


Fig.104

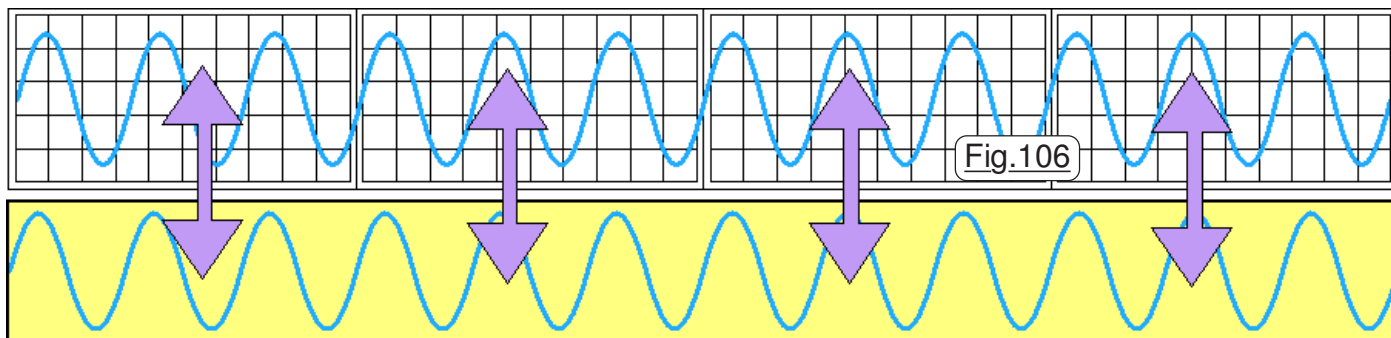
dernière version la zone de texte **4** sera plus étendue, alors que celle qui reste libre en **7** sera réduite d'autant. Lorsque **P00** est activé avec  il commence par "effacer" toute l'EEPROM avec des \$FF et le signale en **2**. Il se trouve que les accentués ne sont pas affichés dans la fenêtre du Moniteur. Assi ils ont été surchargés manuellement dans les zones **3** et **6** pour que vous puissiez en déterminer leur position. La zone **4** liste les textes inscrits en EEPROM alors que la zone rose **5** en situe les adresses dans la mémoire non volatile. En **8** on trouve huit emplacements qui préserveront les attributs des traces sauvegardées. (*Amplitude verticale et Base de Temps.*) L'emplacement d'adresse 547 en **9**



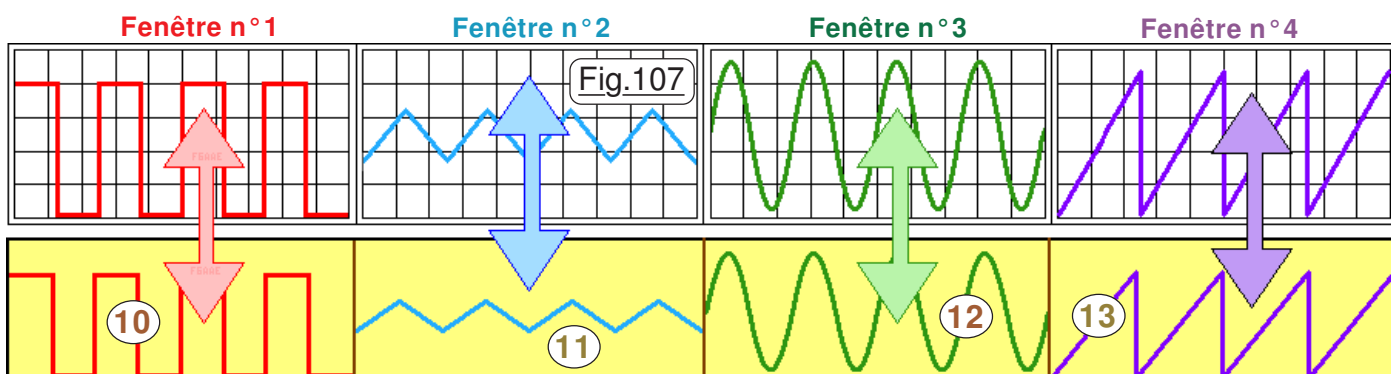
précisera si l'EEPROM est vide, si elle contient une trace **Longue** ou **Quatre TRACES** individuelles, on va détailler cet aspect dans un chapitre qui suit. Si c'est une trace **Longue** qui est stockée en mémoire, les 476 emplacements allant de la cellule 548 à la cellule 1023 contiennent cette dernière. Au contraire, si l'on a opté à la sauvegarde pour **Quatre TRACES** le haut de la mémoire est alors divisé en quatre zones indépendantes de 119 emplacements **10**, **11**, **12** et **13**.

➤ La sauvegarde de plusieurs fenêtres.

Généralement, le signal que l'on a échantillonné est périodique, et l'intégralité de l'enregistrement est une suite continue de "motifs" qui se répètent. Dans ce cas, déplacer le curseur de fenêtre en fenêtre ne fait que montrer des traces qui sont analogues, la manipulation n'apportant pas d'information supplémentaire. La Fig.106 résume ce cas très fréquent. Sauvegarder l'intégralité des 476 échantillons n'est alors plus pertinent, on "gaspille" la place disponible dans les partitions **10**, **11**, **12** et **13**. Il est dans ce cas bien plus utile, comme représenté sur la Fig.107 de ne sauvegarder que l'une des fenêtres dont on précisera l'ordre. Dans ce cas elle ne prend que 119



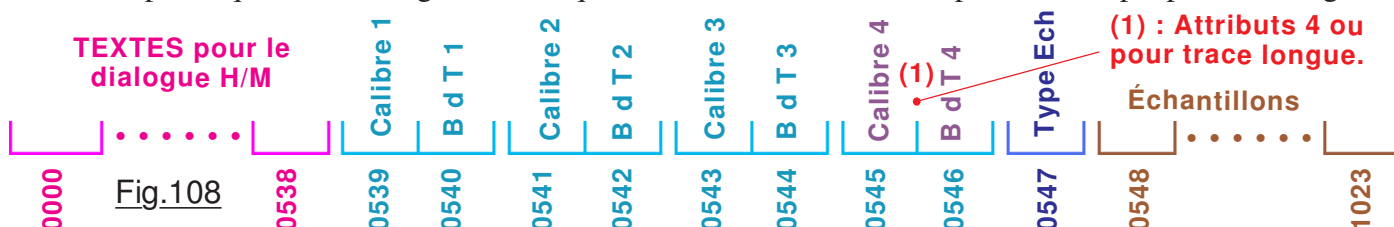
cellules et l'on peut organiser la partition mise en évidence par les limites de zones marrons de la Fig.105 réservant les quatre zones ordonnées **10**, **11**, **12** et **13** pour des fenêtres indépendantes. Ce n'est que pour un échantillonnage dont la forme d'onde varie sur l'intégralité de "la largeur" de l'enregistrement qu'il est intéressant de ne sauvegarder qu'une seule trace étendue. Par exemple pour la sinusoïde amortie ou un train d'onde carrées relatives à un signal binaire du genre CII ou



RS232. (*Signaux binaires échangés sur des lignes de type série.*) Sauvegarder une trace, pour comparaison ultérieure avec de nouvelles mesures, doit préserver la forme d'onde, ainsi que ses attributs dont l'information reste vitale. Les attributs sauvegardés sont le **Calibre vertical** et la **Base de Temps** qui étaient sélectionnés lors de la saisie des échantillons. Ces paramètres sont systématiquement enregistrés en EEPROM que ce soit pour une trace **Longue** pour **Quatre TRACES**. Dans ce deuxième cas c'est la fenêtre présente sur la page écran OLED qui est sauvegardée, c'est à dire avec son décalage potentiométrique. C'est un point important, car ainsi on conserve le cadrage latéral que l'on a initié en vue de faire correspondre des transitions avec les graduations du graticule.

➤ Nouvelle organisation de l'espace EEPROM.

Outre la ou les traces qui vont des adresses de **0548** à **1023**, on sauvegarde le **Type** de l'enregistrement ainsi que quatre **couples d'octets** représentant les attributs des sauvegardes en EEPROM. Ces données directement liées aux **échantillons préservés dans le "haut" de la mémoire** non volatile, sont placées juste avant la grande **zone marron**. Le reste partant de **0000** jusqu'à **0538** sera réservé aux textes du dialogue Homme/Machine ou éventuellement à d'autres données spécifiques. Cette organisation qui à ce stade devrait rester pérenne est proposée en Fig.108



avec précision des adresses relatives en mémoire EEPROM. On notera que les deux octets d'adresses **0545** et **0546** contiendront en fonction de l'information **Type** d'Echantillonnage soit les attributs de la fenêtre n°4 soit ceux de la **Longue TRACE**. L'octet du **Type** d'Echantillonnage sera codé comme indiqué dans l'encadré de la Fig.109.

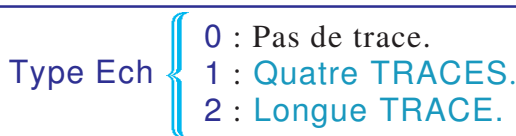


Fig.109

➤ Menu EEPROM et comportement de P15.

Déjà présenté sur la Fig.104 en page 45, le **MENU EEPROM** propose trois actions de base. La Fig.110 montre une variante car la mémoire EEPROM est dans cet exemple occupée par **Quatre TRACES**. Enfin,



Fig.111



Fig.113

ce sous-menu peut nous informer comme sur la Fig.112 que ce n'est pas la peine d'insister avec les commandes possibles car la zone des échantillons est actuellement entièrement effacée et ne contient pas d'échantillons. Si on désire effacer entièrement la zone réservée en EEPROM en la remplaçant par des valeurs zéro, on clique sur le bouton HAUT qui en Fig.111 montre la page-écran qui demande confirmation. En effet, on pourrait avoir cliqué sur la touche par mégarde et vouloir ne pas engager l'effacement. Cette demande de confirmation engendre également l'allumage en rouge de la LED tricolore. Les touches BAS ou DROITE ramènent directement au **MENU de BASE** avec retour au bleu de la LED triple. Même effet si on accepte l'effacement avec toutefois un délai d'environ deux secondes correspondant à l'écriture des 476 cellules EEPROM. Étant

dans le **MENU EEPROM**, cliquer sur GAUCHE fait passer la LED triple en cyan, génère un BIP sonore d'erreur et affiche l'écran de la Fig.113 avec attente d'une touche au clavier. N'importe lequel des quatre boutons poussoir fait revenir au **MENU de BASE** LED triple en bleu. Si l'EEPROM contient une trace **Longue** ou de type Fenêtre, cliquer sur GAUCHE allume la LED triple en cyan et affiche la page écran de la Fig.114 avec attente d'une confirmation. La touche de GAUCHE est sans effet mis à part qu'elle déclenche un BIP d'erreur. Les trois boutons poussoir en **clic court** ou **clic long** font revenir au **MENU de BASE**. Seul celui du HAUT recharge l'intégralité de la zone d'échantillons en "écrasant" ce qui était présent en mémoire vive. Étant en **MENU EEPROM**, quand on clique sur la touche de DROITE la LED triple passe en rouge avertissant que les commandes de la Fig.115 peuvent faire perdre des données. (Si cette page écran s'affiche, c'est qu'il y a bien présence d'une TRACE en EEPROM.) Si on clique sur HAUT et que la mémoire contient actuellement des traces individuelles, la page écran de la Fig.116 s'ouvre et le logiciel attend confirmation. Le refus ramène au **MENU de BASE**. Si l'on persiste, un texte précise "Sauvegarde la Longue." durant environ deux seconde puis il y a retour au **MENU de BASE**. Si l'on désire sauvegarder une Fenêtre, c'est dans ce cas qu'au préalable il devient pertinent

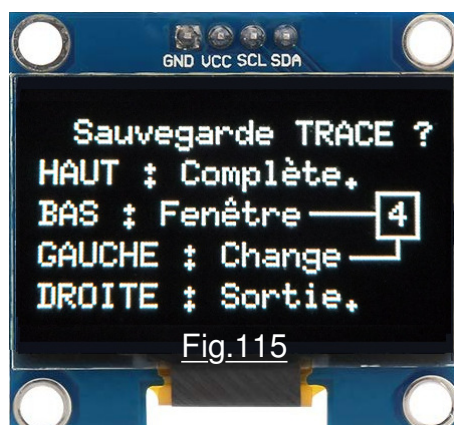


Fig.115

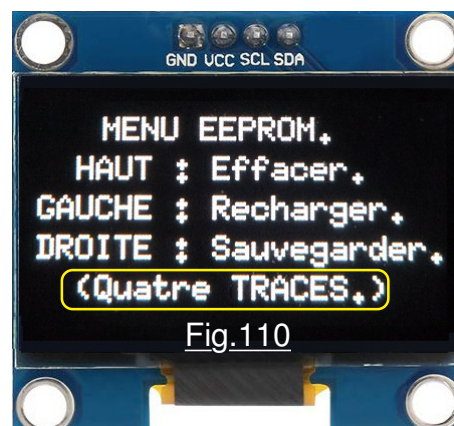


Fig.110

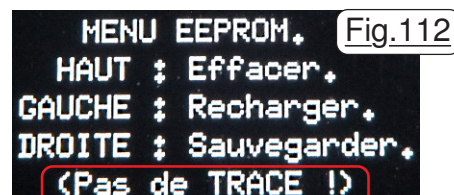


Fig.112



Fig.114

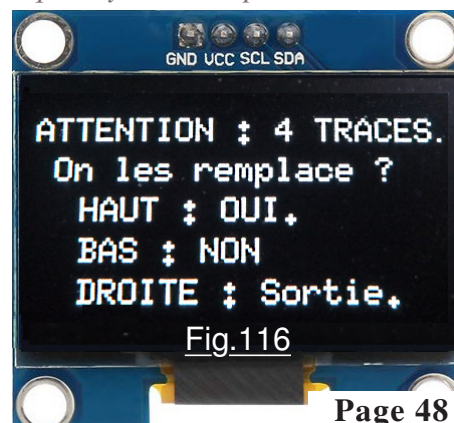
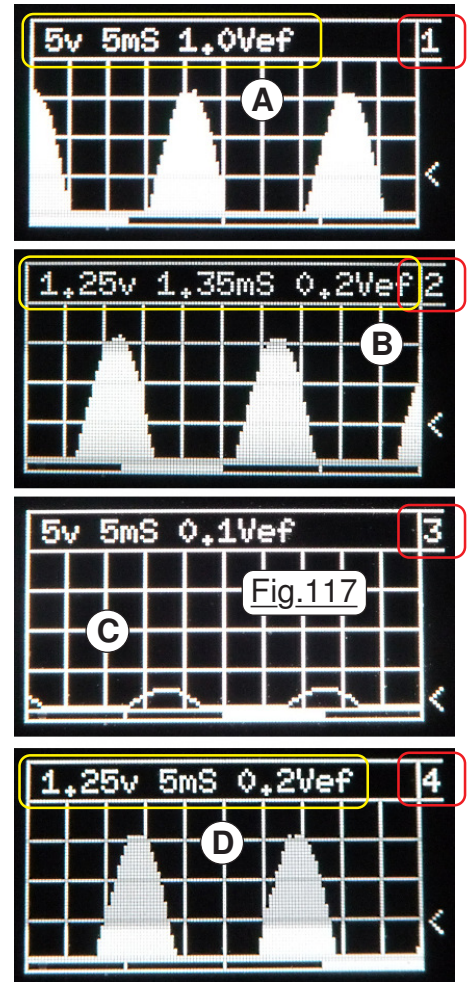


Fig.116

d'effacer l'EEPROM. Avec la touche de GAUCHE on peut changer à convenance le numéro de la fenêtre qui pourra être sauvegardée. Chaque clic sur GAUCHE change l'ordre en permutation circulaire et le précise dans le petit encadré situé à hauteur de la ligne de texte **BAS : Fenêtre**. On se doute que cliquer sur cette touche GAUCHE aura pour effet de sauvegarder les 119 valeurs des échantillons relatifs à la **Fenêtre** concernée par l'index de sélection du bas de la page écran d'affichage des traces. C'est la zone affichée de la courbe qui est mémorisée, c'est à dire que le décalage latéral de la visualisation avec le potentiomètre est pris en compte. Seule l'une des zone **10, 11 12** et **13** de la Fig.105 désignée par la touche GAUCHE sera écrasée par les nouvelles données.

➤ Affichage des Fenêtres individuelles.

Visualiser une **Fenêtre** élémentaire ou une **Longue TRACE**, la Fig.117 en donne des exemples, ne modifie en rien l'aspect de l'affichage, et il n'est pas évident pour l'opérateur de s'en rendre compte au premier coup d'œil surtout si la dernière trace a été sauvegardée il y a longtemps. Aussi, pour attirer son attention le numéro de la fenêtre est ajouté dans l'encadré rouge sur les quatre photographies. Cet avertissement prévient l'opérateur qu'il s'agit bien d'une fenêtre élémentaire, et que par conséquent ses attributs situés dans l'encadré jaune ne sont relatifs que pour la page écran affichée. Quand on change de fenêtre les valeurs sont mises à jour. Par exemple en **A** le calibre était de 5V et la base de temps de 5mS par graduation horizontale. Le cas en **B** est celui qui conduit à l'affichage des données le plus large. C'est pour ne pas que le texte de sorte de son encadrement que le **vEff** de la Fig.94 est devenu **Vef** comptant un caractère de moins. (*Et changement de Majuscule et Minuscule.*) En **C** le calibre est toujours de 5v alors qu'en **B** le calibre est de 1.25v pour la pleine hauteur et la **BdT** n'est que de 1.35mS par graduation. Noter que pour chaque fenêtre la valeur de la tension efficace est recalculée. Bien entendu, à tout moment on peut choisir le mode surface comme en **A, B** et **D** ou le mode ligne comme en **C**. Dès que l'on procède à un nouvel échantillonnage, l'intégralité de la "largeur" lui est attribuée et automatiquement le logiciel revient au mode **Longue TRACE**.



➤ Un nouveau "bilan financier".

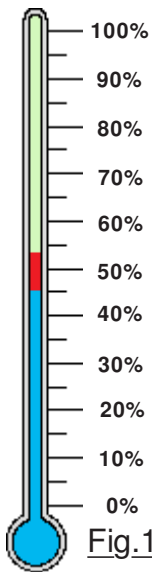


Fig.118

A vouons que représenté sur le graphe de la Fig.118 il est assez préoccupant et alerte les actionnaires de l'entreprise. Ce dessin fait apparaître en bleu l'occupation mémoire du programme lorsque tous les textes ont été logés en EEPROM. L'occupation est alors passé de 46% à **53%**. Puis les fonctions EEPROM on été ajoutées. Elles contiennent pas mal de "bavardages" et des séquences de programme complexes. En dépit des ces particularités, le programme n'a augmenté que de 1760 octets soit à peine 5% de la zone programme. Cette augmentation est montrée en rouge sur le thermomètre virtuel. Hors les actionnaires ne pensent qu'à la rentabilité. (*En réalité, "les actionnaires" c'est ma pomme ! Personnellement je suis ultra radin. Du coup, quand je m'amuse à développer un petit projet, j'ai à cœur d'utiliser au maximum les ressources du microcontrôleur. C'est à dire que je ne suis satisfait que si j'arrive à saturer la zone programme et à rempli l'EEPROM. Et ici on est loin du compte ... Hiiiippppsssss !*)

Pour rentabiliser l'entreprise, il va falloir trouver des idées grosses consommatrices d'octets. Et dire qu'il reste assez de place pour loger neuf fonctions aussi complexes que celles de l'exploitation de l'EEPROM. On n'y arrivera jamais, c'est désespérant ...

Quoi qu'il en coûte, on va continuer à optimiser le code car il n'est pas question de gaspiller. N'oublions pas que le but de l'entreprise consiste à s'amuser en programmant, **et avec méthode**. Donc, sachant dès maintenant que la place disponible ne sera pas toute consommée, et de loin, on va continuer à soigner à l'octet prêt le code C++ ... notre réputation en dépend !

17) Un choix que la morale ne va pas cautionner !

Sachant que l'on va disposer de mémoire à revendre et qu'il y aura largement de la place pour intégrer les routines de sortie des enregistrements sur la ligne série du **Moniteur**, j'ai décidé d'ajouter à l'actuel logiciel **P15** un menu déclenché sur RESET si l'une des touches est cliquée durant le redémarrage du programme. Cette décision semble un peu prématurée, et il serait plus pertinent d'avancer sur des fonctions plus importantes. Elle est toutefois motivée par le fait que je désire dès ce stade incorporer un outil qui permet à tout moment de lister le contenu de l'EEPROM. *C'est une facilité qui ne concerne que les programmeurs*, mais autant en profiter durant le reste du développement si le besoin s'en fait sentir. C'est une aide incontestable qui permet de voir exactement le contenu de la mémoire non volatile et de tirer des conclusions sur le comportement du programme quand il a été modifié. Le **MENU du RESET** pourra prévoir des fonctions vraiment secondaires sans pour autant pénaliser la convivialité d'utilisation de l'appareil puisque ces dernières n'encombrent absolument pas le **MENU de BASE**. Prévoir à ce stade du "luxe" est assez scandaleux, mais comme la morale reste muette et ne peut me raisonner, je vais lâchement en profiter. À nous la débauche de gadgets ... tout en optimisant à outrance le code ça va sans dire !

➤ Un Menu du RESET bien nourri.

L'actuelle page écran de la Fig.119 obtenue sur RESET avec l'un des B.P. clavier cliqué est déjà prévu avec quatre fonctions. Toutefois, il n'est pas du tout prouvé qu'il ne soit par la suite complété par de "nouvelles idées futiles". Aussi, actuellement il ignore *clac court* ou *clac long* sauf pour la touche DROITE qui permet de sortir et de faire transiter le programme vers le **Menu de BASE**. L'ouverture de ce sous-menu allume la LED tricolore en vert. Si on clique sur la touche du BAS durant environ une seconde la LED triple passe en couleur

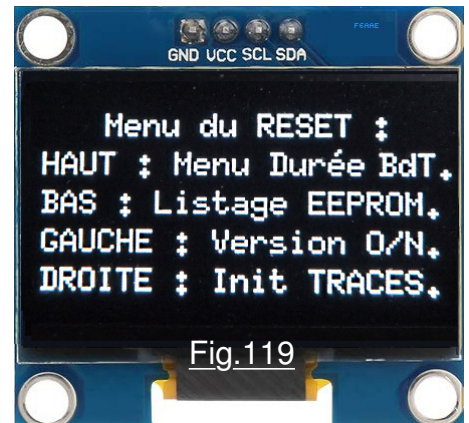


Fig.119

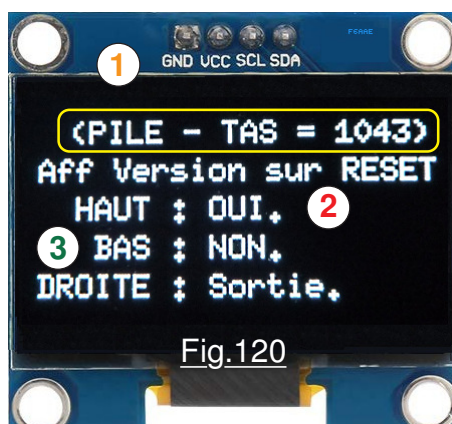



Fig.120

cyan et OLED affiche "**Listage EEPROM**". Puis il y a retour au **MENU du RESET** avec la couleur verte. Pour bénéficier de cette fonction il importe au préalable d'activer le **Moniteur de l'IDE** avec  tout en maintenant actif l'un des boutons poussoir du clavier. Si au préalable la vitesse d'échange des données était bien à 57600baud la fenêtre dédiée affiche les données de la Fig.105 complétée dans la zone violette des nouveaux textes et surtout en 14 d'un octet qui vaut soit 00 pour false, soit 01 pour true. C'est un booléen qui est mis à jour lorsque l'on ouvre le sous-menu de la Fig.120 avec un clic sur GAUCHE. Par défaut lorsque l'on initialise l'EEPROM avec **P00** l'octet est forcé à false. Mais par la suite les actions sur 2 et 3 de la Fig.120 modifieront l'état de la cellule d'adresse 0538. À chaque redémarrage, la procédure **void setup()** se termine par l'examen de ce booléen. S'il est à false le programme enchaîne directement sur **void loop()**. Si c'est 01 qui est inscrit dans l'EEPROM l'écran OLED affiche la version du logiciel de la Fig.121 et attend que l'on clique sur une touche pour passer la main à la routine **void loop()**. Pas de quoi bouleverser notre émotionnel ! Toutefois, il est à ce stade du développement assez probable qu'il sera possible d'agrémenter cette page d'un petit LOGO. Ce sera radical pour gloutonner des octets et améliorer la rentabilité ... encore qu'avec les "délires" ajoutés au démonstrateur **P15** on commence à être pas mal sur ce critère. Dans cette situation la LED triple s'illumine en "orange" incitant l'opérateur à cliquer sur le clavier ce qui ramène à la configuration banale du **MENU de BASE**. L'information située dans encadré jaune en 1 de la Fig.120 dont la valeur numérique va probablement diminuer pour les programmes futur sera commenté plus avant dans le didacticiel.

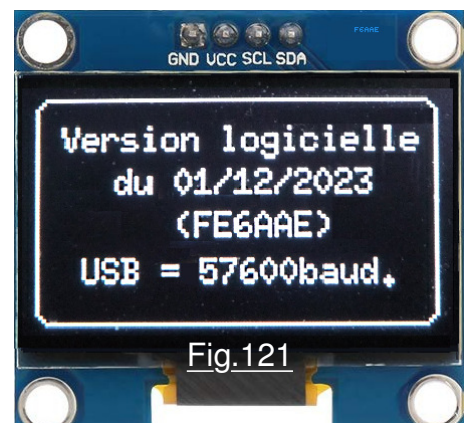


Fig.121

Observons au passage que le démonstrateur **P16_Avec_Menu_RESET.ino** donne un rappel de la vitesse de transmission qui doit être imposée au **Moniteur** de l'**IDE** pour que les affichages de la Fig.105 soient corrects. Normalement il n'y a pas de raison de modifier ce paramètre. Toutefois, si vous observez à l'écran un listage avec des décalages de caractères, il sera alors impératif de diminuer cette cadence. Initialement elle était de 115200baud, mais parfois il y avait des "petits loupés". Avec 57600baud le rafraichissement écran reste très rapide et le listage est parfait.

➤ Un petit correctif.

C'est avec la touche HAUT durant le **MENU du RESET** que l'on va remarquer une petite correction qui concerne la fonction invoquée par le B.P. HAUT. Vous ne l'avez peut être pas remarqué, mais avant **P16_Avec_Menu_RESET.ino** quand on faisait afficher les durées d'échantillonnage, sur la Fig.80 en A de la page 39 les durées indiquées étaient nulles ce qui manifestement n'est pas crédible. Ayant largement de la place disponible pour logger des octets de code, comme le confirme la Fig.122 maintenant ce, petit détail est totalement corrigé.

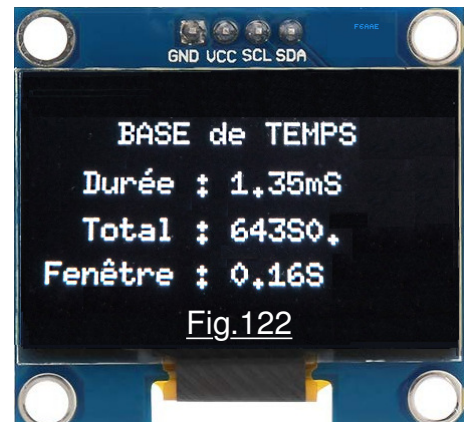


Fig.122

➤ Conseil d'administration.

Encore un bilan financier en Fig.123 pour évaluer la rentabilité de **P16** en termes "d'occupation des sols". Si utilisant le **MENU du RESET** avec la commande du B.P. BAS vous provoquez un listage du contenu de l'EEPROM, vous pouvez vérifier qu'il ne reste plus que 41 emplacements de disponibles pour logger du texte. Autant dire que la "rentabilité" d'utilisation de la mémoire non volatile va friser l'idéal. Pour les fonctions futures on risque de rapidement saturer cette ressource de l'ATmega328. Ensuite s'il y a de nouveaux bavardages on sera obligé de logger les textes dans le programme, avec diminution de l'espace dynamique ce qui est moins intéressant. (On augmente le risque de collision de PILE ... voir le chapitre suivant.) Donc : Affaire à suivre ! En revanche, la zone occupée par ces changements pourtant boulimiques en octet reste désespérément "faible" et l'augmentation de température en rouge n'est pas démentielle. Nous allons vraiment avoir du mal à saturer la zone dédiée au programme. Il faut trouver des idées ...

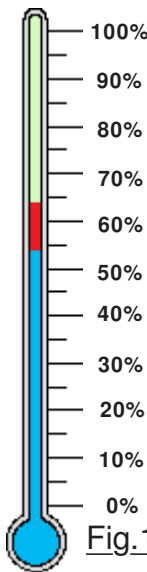


Fig.123

➤ Paramétrer en tête de programme.

Dans le démonstrateur **P16**, par exemple dans la procédure **Affiche_la_version_du_logiciel()** on doit modifier la date de validation du logiciel d'exploitation, ou dans cette même routine modifier l'adresse en EEPROM où commencent les données à lister en hexadécimal. Tout au long du développement il peut s'avérer d'avoir ainsi à modifier parfois des constantes ou de nombreux paramètres et on doit pouvoir facilement repérer les lignes d'instruction concernées. Il suffit de les faire terminer par des remarque de type `//@@@@@@@@@@@@@@@@@@@@`. Toutefois, quand on est amené à corriger des valeurs dans un listage bien plus "étalé" que celui du démonstrateur **P16**, **il est fortement recommandé de définir tous les paramètres d'initialisation en tête de programme et si possible de les regrouper**. Ainsi, on a une énumération compacte et l'on ne risque pas d'en oublier, et surtout c'est bien plus rapide que d'avoir à chercher de multiples emplacements dans un long listage. Cette façon de faire concerne directement la "programmation avec méthode". Aussi, dans le

```
//----- Constantes du programme -----
#define Version "01/12/2023" //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#define Vitesse_USB 57600 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#define Debut_Hexadecimal 538 // Adresse de fin des textes pour du listage en HEXADECIMAL
#define NB_tests_antirebonds 50 // Nombre de tests de stabilisation du B.P.
```

Fig.124

programme de développement **P16_Avec_Menu_RESET.ino** la date de version est définie en tête de listage par la constantes **version** et le taux de transfert sur la ligne de dialogue USB est défini par la constante **Vitesse_USB**. Du reste, comme on peut l'observer sur la Fig.124 on a regroupé quatre paramètres sous forme de déclarations **#define**. Dans ces dernière il est possible de définir des chaînes de caractères, des entiers, des réels, des booléens etc. Par exemple la valeur 50 pour **Nb_tests_antirebonds** a été optimisée expérimentalement en fonction de la qualité des B.P. utilisés.

- **Brusquement un programme présente un comportement anormal,**
- **La séquence qui diverge n'a rien à voir avec les dernières modifications effectuées.**
- **Quand un tel comportement étrange se produit avec la certitude qu'il n'y a aucune relation entre la séquence anormale et les derniers correctifs apportés au logiciel, il faut diagnostiquer une collision de **PILE**, c'est la cause la plus probable du problème.**

18) Collision entre la PILE et le TAS. (Chapitre qui ne concerne que les programmeurs.)

Bien que non précisé dans la liste des conseils pour programmer avec méthode, car la page est bien assez remplie, ce serait dommage dans notre cheminement expérimental de ne pas aborder ce sujet, car forcément un jour ou l'autre vous risquerez d'en être la victime. Explications :

Phénomène particulièrement sournois, la collision de **PILE** survient brusquement sans qu'aucun signe avant-coureur ne nous prévienne. Pour comprendre de quoi il s'agit, il faut entrer dans la vie intime du fonctionnement des microcontrôleurs. Il serait hors propos dans ces lignes d'étudier à la loupe l'agencement matériel de l'ATmega328 et d'en détailler finement le fonctionnement interne. Nous allons dans ce chapitre nous en tenir au strict minimum vital.

Fonctionnement de la mémoire vive SRAM.

La mémoire vive (256 + 2Ko) est généralement divisée en quatre zones :

- Les 256 premiers octets pour les registres généraux du microcontrôleur (Représentée en jaune sur la Fig.85) occupent "le bas" de la SRAM. (En "assembleur" c'était la Page zéro".)
- La zone nommée **BSS** qui contient toutes les variables globales, allouées statiquement au moment de l'édition de lien lors de la compilation. La **BSS** est utilisée par de nombreux compilateurs pour désigner une zone de données contenant les variables statiques définies dans les initialisations, et les déclarations avant **void loop()**.
- Le **TAS** sur lequel on entasse du bas vers le haut est destiné aux **allocations dynamiques** dans lequel on peut attribuer et libérer des blocs de mémoire. (Nommé **HEAP**) Le **TAS** se fragmente généralement au cours de l'évolution du programme, (Car une variable locale libérant de la place laisse "un trou" libre.) avec un risque notable de le rendre inutilisable.

*Défragmenter **HEAP** par une séquence de code de type "Ramasse miettes" est faisable mais relativement dangereux, car si l'on déplace une variable en cours d'utilisation, les conséquences peuvent s'avérer ingérables.*

- La **PILE** nommée **STACK** mémorise temporairement :
 - * Les paramètres associés à l'appel des fonctions et procédures,
 - * **Les adresses de retour des fonctions et procédures,**
 - * Les variables locales aux fonctions et procédures.

La **PILE** est une zone de mémoire commençant en haut de la SRAM qui se charge vers le bas de façon linéaire et continue lors des appels des fonctions ou des procédures. Elle se réduit vers le haut lors des retours. Chaque appel à une procédure empile l'adresse, chaque retour la dépile et libère la place. Au cours du programme, si un grand nombre de données sont sur le **TAS** qui est "très haut", et que l'on enchaîne un grand nombre d'appels à procédures sans retour, (Cas des *subroutines récursives par exemple*) il peut arriver que l'espace entre **PILE** et **TAS** devienne nul. C'est la collision et l'écrasement mutuel des OCTETS engendre un fonctionnement totalement imprévisible du microcontrôleur. Aussi, **avant de considérer que notre programme est fiable, il faut impérativement vérifier que le risque de collision de PILE est dérisoire.** L'expérience montre que lorsque toutes les initialisations de **void setup()** sont terminées, il est recommandé de ne pas avoir **moins de 100 octets**, car le risque de collision par fragmentation de la zone devient exagéré.

(@) : Sur la Fig.125 les noms des divers pointeurs sont imposés par le compilateur de l'IDE.

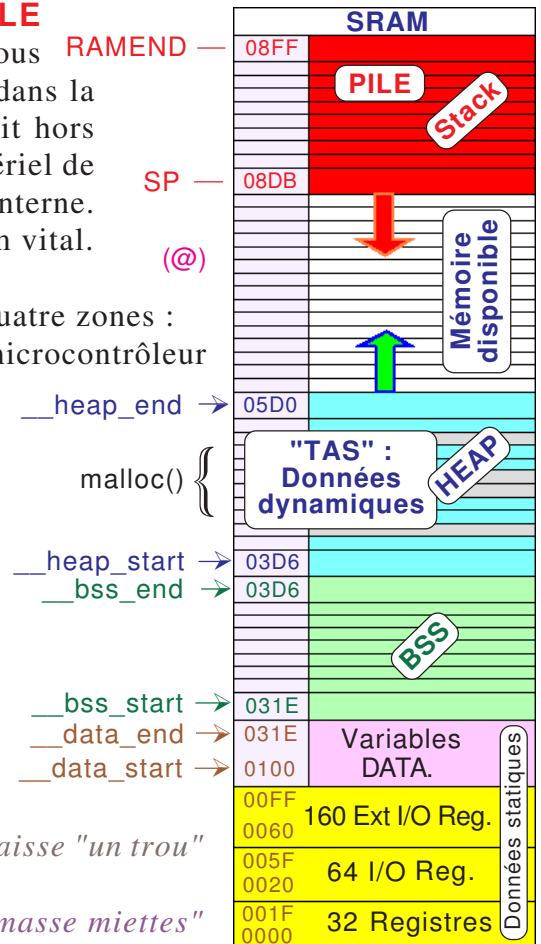


Fig.125

➤ Prendre une assurance contre les collisions de PILE.

Invoker du préventif dans ce domaine n'est pas une arme absolue. Observez le programme d'exploitation provisoire **P16_Avec_Menu_RESET.ino** dans lequel, dans la procédure nommée **void Menu_Version()** se trouve la séquence suivante qui affiche l'espace entre la **PILE** et le **TAS** à ce stade de l'exécution où l'ensemble du contexte est en place :

```
Aff_TEXTE_EEPROM(444,13); // "PILE - TAS = "  
u8g.print(SRAM_LIBRE())
```

Juste au dessus on trouve la fonction qui calcule la valeur de l'espace disponible :

```
int SRAM_LIBRE() { // Fonction qui retourne la taille de SRAM disponible.  
    extern int __heap_start, *__brkval; // Déclaration des deux pointeurs dédiés.  
    byte BIDON; // Dernière variable allouée, donc occupe le "haut" du TAS.  
    if (__brkval == 0) {return (int) &BIDON -(int) &__heap_start;}  
    else {return (int) &BIDON -(int) __brkval;}}
```

Le résultat sur la Fig.126 (*Qui n'est qu'un clone de la Fig.120 pour nous éviter d'avoir à tourner les pages.*) pour notre programme d'exploitation montre qu'avec une telle marge de sécurité, si notre logiciel se met à faire des choses étranges quand on l'a modifié, c'est que l'on a commis une erreur de logique, car ici la collision de **PILE** n'est pas vraisemblable. Notez au passage que le compilateur indique une place disponible pour les variables locales de **1043** octets ce qui n'est pas directement lié avec la RAM réservée aux données dynamiques. Enfin, je vous recommande très très fortement de **toujours effectuer ce test quand vous venez de terminer un programme**. Pour un sketch qui



consomme la presque totalité des 30720 OCTETS disponibles, on peut manquer de place pour logger ce test. Il importe alors de supprimer un ou deux appels à procédures (*Ce qui ne modifie pas l'évaluation*) pour faire provisoirement de la place. Une fois la marge de sécurité vérifiée, vous rétablissez les lignes provisoirement passées en remarques et l'affaire est définitivement classée ! Avant de clore définitivement ce chapitre il me semble utile de passer en revue les éléments de programmation qui influencent le plus le risque de collision de **PILE** de façon à ce que vous puissiez aborder sereinement le problème si d'infortune votre programme en était victime :

➤ Configuration qui produit la collision de PILE avec le TAS.

Concrètement on oublie royalement qu'un nombre important d'interruptions se produisent en tâche de fond. Quelquefois un codeur rotatif génère des interruptions, sans compter les procédures **delay()**, les fonctions telles que **millis()**, la PWM ... une foule de ressources internes déclenche des interruptions. C'est transparent pour l'utilisateur car c'est le compilateur C++ qui sur ces instructions fait sa cuisine interne. Arrive un moment, ou trop de données sont empilées sur le **TAS** et viennent écraser les adresses empilées. Puis le délimiteur '}' de fin d'une procédure ou d'une fonction demande au processeur de dépiler une adresse de retour. Comme cette dernière contient les résidus de la variable qui a "écrasé" les octets, le programme se "branche" strictement n'importe où. Étant alors sur du code objet incohérent, le comportement du logiciel devient totalement aléatoire. Par exemple vous avez changé un texte **"Salut"** en **"BONJOUR les amis"**. Suite à cette broutille le programme diverge complètement ou "se fige". Ce n'est manifestement pas un problème de logique. Il ne reste plus dans un tel cas qu'à diagnostiquer l'éventualité d'une collision de **PILE**.

PRÉVENTIF : Aucun programmeur n'est à l'abri d'une telle "catastrophe". Aussi, pour minimiser les risques il faut placer le minimum de chaînes de caractères dans le programme car en réalité elles sont placées sur le **TAS**. Il faut également (*Et surtout.*) minimiser les tableaux.

CURATIF : Quand se produit le **Scratchhhh prouitchhhh bom bring protchhhh !** c'est qu'il est trop tard. Nous avons placé plein plein plein de bavardages, alors que nous savons que ces "bla bla bla" sont entassés dans la mémoire dynamique. Notre démonstrateur comporte une foule de procédures et de fonctions qui passent des paramètres, sans compter les **for (byte l=1, ...)** qui ne sont pas gratuits. En effet, les variables locales des boucles **for** doivent aussi être logées en RAM.

REMÈDE : Dégager impérativement de la place sur le **TAS**.

O bnuilé par la rédaction du didacticiel et surtout par la mise en page, j'ai totalement oublié dans le **MENU du RESET** de décrire l'effet obtenu lorsque l'on active avec un *clac court* la touche de DROITE. En apparence il ne se passe rien. Dans la pratique, le programme remplit l'espace mémoire dédié aux enregistrements par une trace *Longue* simulant une sinusoïde amortie. On ne se rendra compte de sa présence que si dans le **MENU de BASE** on fait afficher la *TRACE* avant de déclencher une quelconque numérisation ou la récupération d'un enregistrement EEPROM. Ce correctif au tutoriel étant effectué, nous pouvons poursuivre notre chemin.

19) Exporter l'enregistrement courant vers le Moniteur de l'IDE

P ouvoir envoyer la totalité d'un enregistrement vers un ordinateur ou une tablette est une fonction que tout oscilloscope actuel doit mettre à la disposition de l'utilisateur. Cette possibilité n'a rien à voir avec un stockage provisoire pour comparer des signaux électriques lors de la mise au point d'ensembles électroniques. Pour satisfaire cette facette de l'utilisation de l'appareil, la sauvegarde en EEPROM est généralement suffisante. En revanche, pour documenter un historique, rédiger un article quelconque, l'illustrer avec des graphes adaptés devient un impératif. Par exemple observez la Fig.41 page 24 ou la Fig.89 en page 41. Ces dessins ont été obtenus en photographiant l'écran d'un petit appareil du commerce avec tous les problèmes engendrés par des poussières, les reflets parasites, la distorsion trapézoïdale etc. S'il m'avait permis de récupérer les courbes sur l'ordinateur, le résultat serait bien plus probant. Pour le plaisir de la programmation on va doter notre prototype de la possibilité de déverser les valeurs des échantillons vers le **Moniteur de l'IDE**, *et voir comment les récupérer sur l'ordinateur pour s'en servir dans un tableur par exemple*. Pour mémoire, c'est dans le **MENU de BASE** de la Fig.92 redonnée ici pour ne pas avoir à la rechercher dans le didacticiel, qu'avec la touche de DROITE on invoque cette fonction.

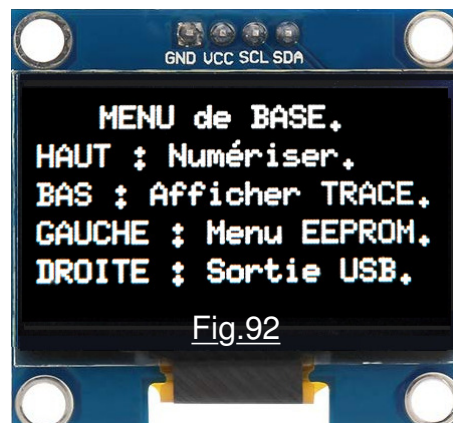


Fig.92

➤ **"Téléverser" les valeurs des échantillons dans la fenêtre du Moniteur de l'IDE.**

T héoriquement il suffit d'ajouter à **P16** dans un autre démonstrateur nommé **P17_Avec_televersement_de_la_TRACE.ino** les routines qui vont émuler cette nouvelle fonction. C'est d'autant plus faisable que l'on dispose encore de la place à revendre. Le plus difficile est déjà réalisé : On a trouvé un nom pour le prochain démonstrateur. À nous le codage en C++ ! Lorsque ce démonstrateur est téléversé et que l'on clique sur la touche de DROITE, durant une seconde environ la LED triple devient rouge et l'écran OLED affiche "*Exporte sur USB :*". Puis il y a retour au **MENU de BASE** la LED tricolore étant à nouveau bleu azur. En première apparence il ne s'est pas passé grand chose et on n'a rien vu. Il fallait, avant, activer le **Moniteur de l'IDE** configuré pour une vitesse de transfert de 57600 baud. Trop tard, il n'y a plus rien à voir ... circulez !

On recommence mais cette fois on anticipe avec . Du coup on obtient un listage qui ressemble à celui de la Fig.128 à condition toutefois que nous ayons au préalable généré dans la mémoire dynamique des échantillons un échantillonnage fictif de **4 TRACES** individuelles objet du chapitre qui suit. Dans l'exemple de la Fig.127 on a imposé la trace d'un signal BINAIRE. Cette simulation est relative à une *Longue TRACE* ce qu'indique la

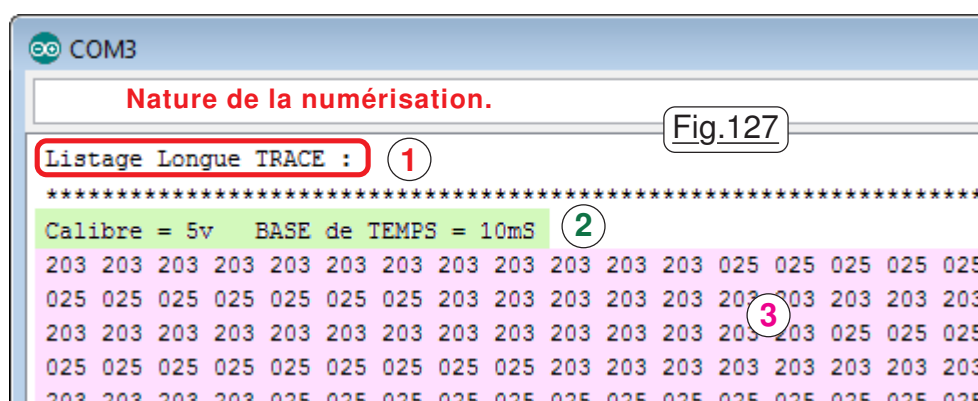
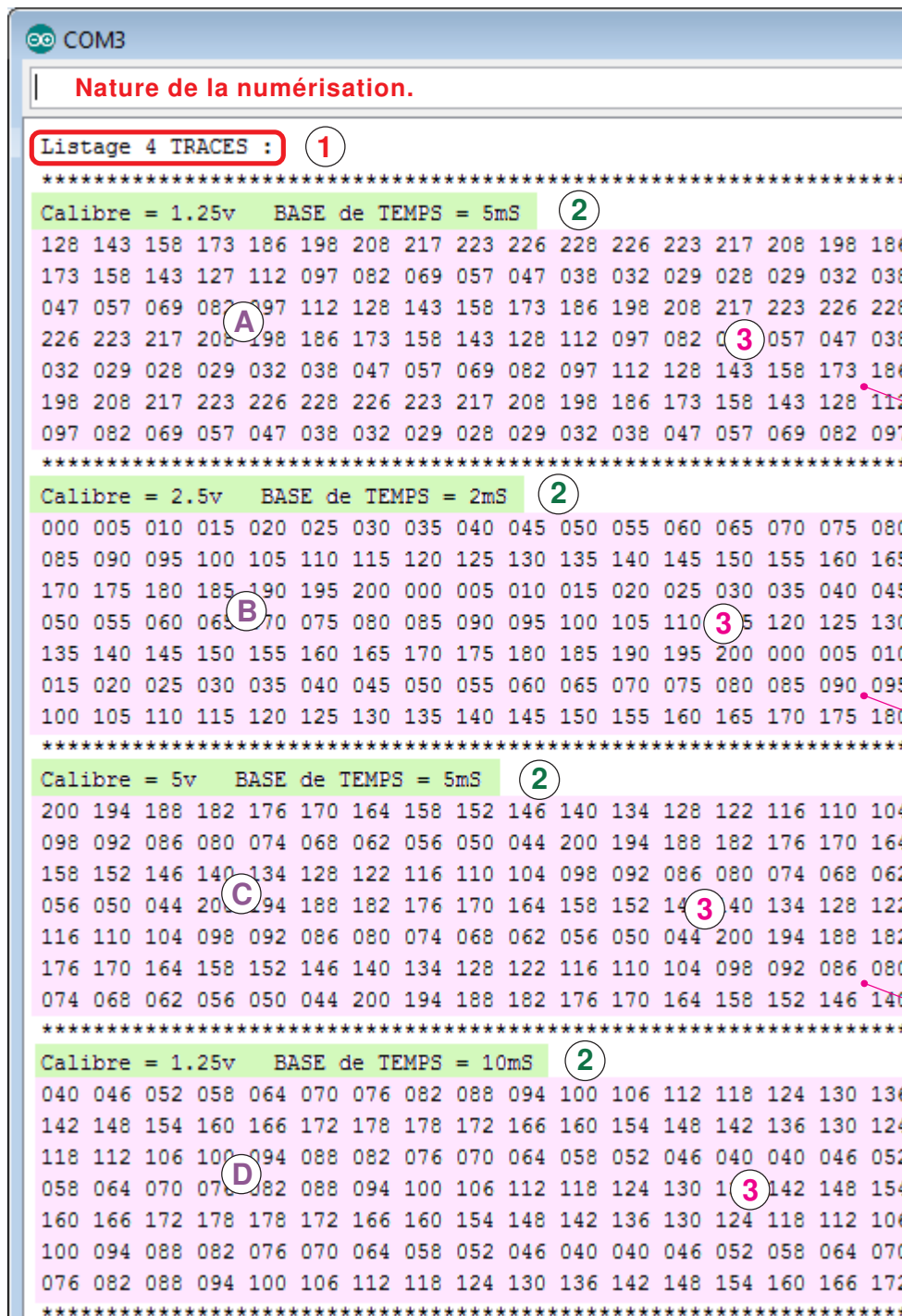


Fig.127

fonction de téléversement au début du listage en **1**. Dans ce cas, l'indication des conditions d'enregistrement **2** se retrouvent pour les quatre "blocs" **3** listant chacun les 109 échantillons enregistrés d'une fenêtre. Au contraire, l'exemple de la Fig 128 est relatif à la



Listage de quatre TRACES indépendantes avec pour chacune des conditions de capture différentes.

Fig.128

"Bloc" A : Sinusoïde périodique.

7 lignes de 17 échantillons qui totalisent les 119 mesures.

"Bloc" B : Dents de scie montantes sans composante de tension continue.

7 lignes de 17 échantillons qui totalisent les 119 mesures.

"Bloc" C : Dents de scie descendantes avec composante de tension continue.

7 lignes de 17 échantillons qui totalisent les 119 mesures.

"Bloc" D : Signal triangulaire avec composante de tension continue.

numérisation de quatre fenêtres indépendantes. Du coup les informations de saisie des échantillons en 2 sont différentes pour chaque bloc de 119 valeurs 3.

➤ Dilapider sa richesse, un vrai calvaire !

Compte tenu des ressources d'un microcontrôleur ATmega328 et du manque d'idées pour arriver à "rentabiliser" notre programme, nous nous trouvons dans la situation de ces ultra-riches qui n'arrivent plus à dépenser leur fortune tellement elle les étouffent. Même en achetant un JET privé et une ROLEX chaque jour, pas moyen de désengorger les placements optimisés effectués dans les paradis fiscaux ! Placer les revenus boursiers ne fait qu'empirer leur calvaire, car la fortune augmente encore plus vite. Heureusement pour nous, quand on dépense nos OCTETS à profusion, la zone verte du thermomètre diminue ... mais avec une lenteur désespérante. Dans ce démonstrateur a été ajoutée une dépense aussi importante que celle pour transférer les enregistrements vers la ligne USB, cette consommation démesurée étant consentie pour une fonction totalement dérisoire. En effet, l'ajout consiste en un sous-menu pour pouvoir générer plusieurs sortes de simulations d'échantillonnages. Ce groupe de quatre fenêtres indépendantes s'avère bien utile pour tester le listage sur USB ou la Sauvegarde / Rechargement en mémoire EEPROM.

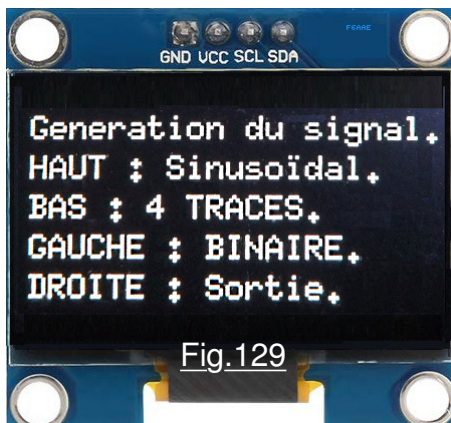


Fig.129

L'écran de la Fig.129 montre le sous-menu qui a été mis en place pour cette dépense scandaleuse et inconsidérée pour générer six sortes de signaux que l'on rencontre de façon courante lorsque l'on s'amuse en électronique dans des domaines variés. Ces formes d'ondes sont représentées sur les Fig.130, Fig.131 et Fig.132 qui sont des photomontages où les quatre secteurs **A**, **B**, **C** et **D** sont présentés en continuité pour reconstituer la totalité de l'échantillonnage que ce soit en **Longue TRACE** pour les exemples des deux représentations Fig.130 et Fig.131 ou une trace multiple en Fig.132 incluant quatre formes d'ondes différentes. Sur tous ces arrangements les secteurs sont séparés par un trait vertical de couleur verte et les curseurs correspondants aux fenêtres concernées sont coloriés en jaune. Considérons le cas de la Fig.130 qui affiche une trace longue. On retrouve la Sinusoïde amortie linéairement déjà rencontrée lorsque les textes ont été sauvegardés en EEPROM. Le signal de la Fig.131 a été surchargé par un contour rouge. C'est celui

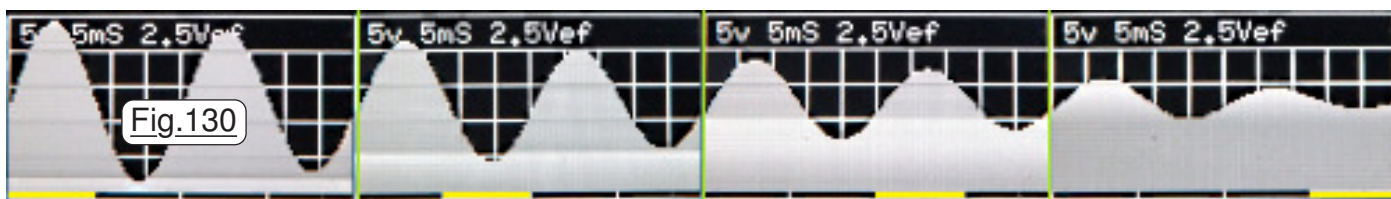


Fig.130

de la Fig.127 qui présente donc un **Calibre** de 5 volts et une base de temps de 10mS par graduation. Cette trame est relative à une donnée sérielle de 28 impulsions avec de surcroît un BIT de START à l'état "1" en tête, de quatre caractères codés en ASCII sur 7 BITS, de six BITS **Correcteurs** par **Redondance**, d'un BIT de Parité et d'un BIT de STOP à l'état "0". (*Oubliez tout ça, c'est pour frimer !*) On peut facilement déterminer sur cette trame que la vitesse de transmission est de 10mS par BIT soit un taux de transfert de $1 / 0.01 = 100\text{baud}$ ce qui est peu fréquent. (*110baud serait plus courant*)

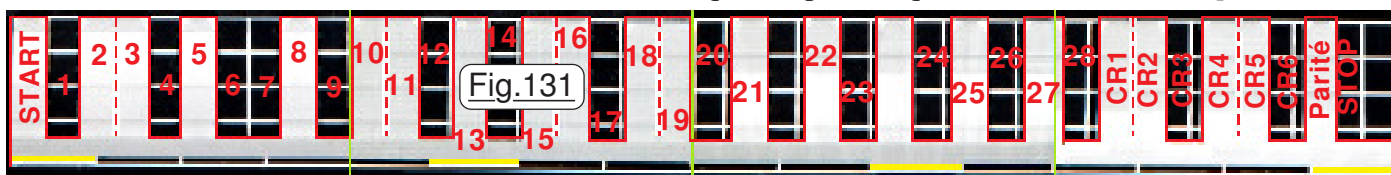


Fig.131

Le groupe de quatre signaux indépendants en Fig.132 est très représentatif de formes d'ondes très classiques en électronique. En **A** une sinusoïde d'environ 17mS de période soit 60Hz, signal assez caractéristique du réseau électrique des USA par exemple. (*Attention, ici le signal n'est pas alternatif et présente une composante continue et positive de 0,5v environ.*) En **B** un signal triangulaire sans composante continue qui sert généralement à vérifier la linéarité d'un amplificateur. En **C** un autre signal de type triangulaire avec composante continue qui permet de vérifier la linéarité d'un amplificateur et de voir comment il élimine cette composante continue. Cas **B** on analyse le

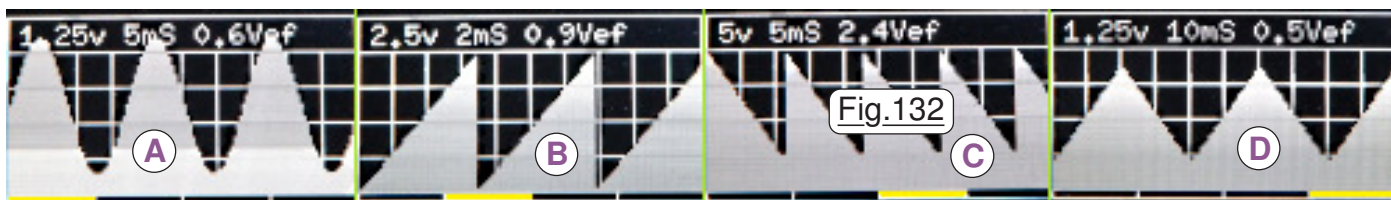


Fig.132

comportement sur un front descendant brusque, alors qu'en **C** c'est la transition montante qui sera étudiée. Le signal triangulaire **D** quand à lui est une variante des deux précédents, toujours avec présence d'une composante continue mais sans les fronts brusques de transition. Ces simulations sont appréciables pour manipuler et expérimenter le transfert sur USB ou la Sauvegarde/Restitution en EEPROM alors qu'un générateur BF n'est pas disponible dans notre modeste laboratoire d'amateur.



Ben Môa môa j'affirme que "gaspillem trébeaucoupyem des octetsyem c'est maouzeyem perseverare diabolicum !

➤ Saturation de la mémoire EEPROM.

Avec le démonstrateur `P17_Avec_televersement_de_la_TRACE.ino` mission accomplie. En effet, lorsque l'on exporte le contenu de l'EEPROM sur le Moniteur de l'IDE on constate sur l'extrait de la Fig.133 que sur les 1023 OCTETS de la mémoire non volatile on en a utilisé 1022, autant dire que pour la rentabilité on est au sommet. Il ne reste plus que l'emplacement d'adresse

0464 i o n l o g i c i e l l e (F
0480 E 6 A A E) U S B = b a u d
0496 . C a l i b r e v Fig.133 E x p
0512 o r t e s u r U S B : S i
0528 n u s o i d a l . . 00 02 02 02 02 02
0544 02 02 02 02 80 8D 9A A7 B3 BF CA D4 DC E5 EC F2
0560 F5 F9 FB FC FA F8 F4 F0 E9 E2 DA D1 C7 BC B1 A5
0576 98 8C 7F 73 67 5B 4F 44 3A 31 28 20 1A 15 10 0D
0592 0C 0C 0C 0E 12 16 1C 22 2B 33 3C 46 52 5D 68 74
0608 80 8B 97 A2 AD B7 C1 CA D1 D8 DE E4 E7 FA FC FD

537 repéré en jaune. C'est une sécurité, car par la suite si l'on trouve une idée géniale qui impose de sauvegarder une donnée de type `byte` ce sera possible. *Si c'est un entier ou deux OCTETs dont on aura besoin, on pourra libérer l'emplacement 536 et utiliser la procédure `Point()` pour compléter l'affichage.* Attention, car à partir de

maintenant tous les textes devront résider dans le programme et chaque nouveau "bavardage" empiètera l'espace réservé au croquis, mais surtout diminuera d'autant la place disponible entre la **PILE** et le **TAS**. Il faudra donc tempérer notre ardeur si l'on ajoute des fonctions avec texte.

➤ Encore une réunion du conseil d'administration.

Présenter le bilan EEPROM se solde par des applaudissements. En revanche l'inventaire des consommations pour la fonction d'exportation USB et celui de génération des signaux fictifs est lamentable comme le montre le thermomètre de la Fig.135 qui met en évidence la diminution pour le moins timide de la zone verte. Bien que nous ayons ajouté deux fonctions grosses consommatrices d'OCTETs, force est de constater qu'il reste encore 29% d'emplacements non rentabilisés ! C'est purement dramatique, car il faudrait encore une flopée de fonctions "Octivores" pour combler ce vide dramatique. Il faut absolument trouver des idées ... sauf que notre appareil bénéficie de tous les perfectionnements imaginés. Nous sommes dramatiquement riches, le moral est au plus bas. Et encore, le démonstrateur est compilé avec la version 1.7.9 de l'IDE. Si l'on utilise

Le croquis utilise 19614 octets (63%) de l'espace de stockage de programmes. Le maximum est de 30720 octets.
Les variables globales utilisent 1020 octets (49%) de mémoire dynamique, ce qui laisse 1028 octets de mémoire libre.
889

Fig.134

la version 1.8 la taille programme dégringole à 19614 octets car elle plus optimisée. La réduction de taille du code objet est de 2288 octets soit -8% ce qui est considérable. (*Personnellement je ne*

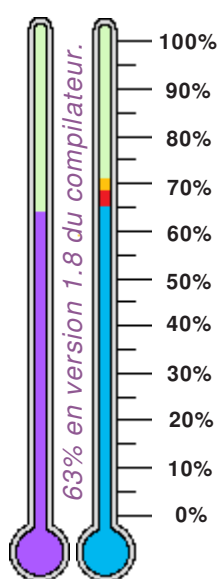


Fig.135

passer à cette dernière que lorsque l'espace programme commence à manquer. En effet, je préfère les couleurs de la version 1.7.9 dans l'éditeur de texte de l'IDE, c'est purement personnel ... car je déteste le changement.)

Pour consommer des octets, je pourrai vous refaire le coup de la post synchronisation par satellite et géo-localisation par déphasage différé, mais vous n'allez plus me croire pour cette blague idiote. Il faut absolument trouver une idée ...

- YOUOUOUOUYYYYYYYYY, j'ai une idée !

En relisant ce didacticiel j'ai retrouvé Fig.121 en bas de la page 50 une remarque que ... je n'avais pas vraiment oubliée : Agrémenter la page de garde d'un LOGO.

CONCLUSION : Pour ce projet l'ATmega328 présente des ressources plus que suffisantes. Nous sommes dans une stratégie ludique où une *approche de la programmation méthodique et rigoureuse est prioritaire*, et naturellement c'est toujours dans cette optique qu'il faut "travailler". Ceci dit, vous n'avez aucune raison d'en rester à la version 1.7.9 même si le compilateur v1.8 est un râleur. En effet, il génère un message un peu abscond relatif à des pointeurs, mais rassurez-vous le code OBJET est correct et tourne à 100%. C'est la séquence de détermination de **PILE** - **TAS** qui doit être en cause car elle

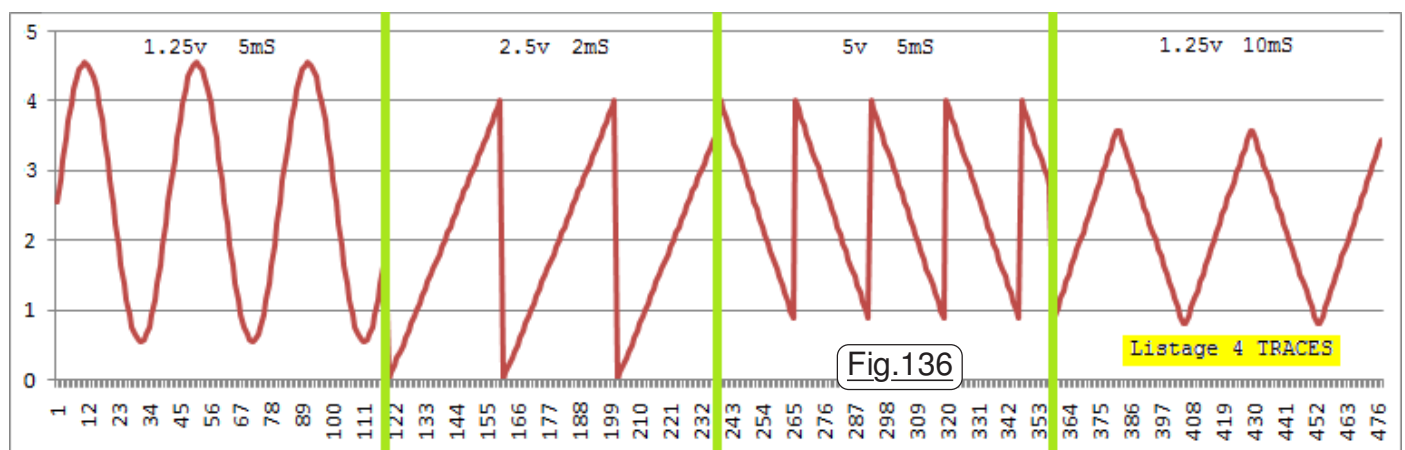
utilise des pointeurs qui ont certainement été modifiés. Ce n'est pas fondamental ... et pour le moment je ne vais pas creuser le sujet. N'oublions pas que ces histoires de consommations à outrance d'octets pour rentabiliser l'utilisation des ressources du microcontrôleur ne sont que des plaisanteries. Toutefois, la représentation de l'évolution de la taille du programme avec des thermomètres est très sérieuse. Elle montre à quel point les premières procédures sont hautement consommatrices d'espace programme, car avec une simple instruction comme `Serial.begin(57600)` d'un coup on amène dans le corps du programme plusieurs routines "octivores". C'est pire avec l'utilisation d'une bibliothèque comme "`U8glib.h`". Ensuite, les fondations étant en place, l'inflation, y compris pour des séquences compliquées, devient considérablement plus faible.

RESUMÉ : (Quel que soit le projet on constatera les effets mentionnés ci-dessous.)

- Quand on débute la rédaction d'un programme, au début la taille augmente de façon inquiétante, mais ***il ne faut pas s'en préoccuper***, car les routines de base sont insérées dans le logiciel.
- Ensuite la taille grandit de façon bien moins rapide et l'on est étonné de constater que plus on ajoute du code, moins rapide est l'évolution du code OBJET.

➤ Réaliser simplement un graphe pour archivage.

A vant d'ajouter un LOGO à la "page de garde", comme c'était promis et écrit en vert juste sous la Fig.92, on va voir comment récupérer facilement les données fournies numériquement par la voie de dialogue série USB. On va examiner un protocole possible visant à transformer la TRACE multiple exportée en un graphe analogue à celui de la Fig.136 par exemple. Avouez que ce



graphe est autrement plus propre que le montage de la Fig.130 pourtant bien plus compliqué à créer. La première étape consiste à copier les valeurs dans la fenêtre du **Moniteur**. Passer ces valeurs en utilisant une présentation telle que celle de la Fig.128 s'est avéré particulièrement indigeste, car pour créer un graphe comme celui de la Fig.136 il faut recopier les 476 valeurs sur une seule colonne du tableur. Aussi, "au pied levé" le programme `P17_Avec_televersement_de_la_TRACE.ino` à été modifié. Il totalise actuellement 21948 octets pour le code objet et 1019 octets de mémoire dynamique. Les valeurs énoncées dans les chapitres précédents ne sont plus tout à fait exactes, mais pour quelques octets de plus je ne vais pas reprendre les textes et les images.

S uite à cette petite modification en apparence le programme se comporte exactement comme avant. Donc, étant dans le **MENU de BASE** on effectue un ***clic court*** sur la touche de DROITE et l'on obtient comme avant la présentation des données sous forme d'un tableau. Par contre, si DROITE est assortie d'un ***clic long***, l'affichage des 476 valeurs se fait sur une seule colonne. C'est "presque illisible" mais c'est exactement ce qu'il nous faut.

Protocole pour réaliser le graphe :

La première action à conduire consiste à ouvrir un tableur quelconque comme `Exel.com` par exemple ou tout clone disponible actuellement. Puis, action un peu indigeste, on sélectionne dans la colonne de gauche, comme montré sur la Fig.137 en **B** les cellules sur 476 lignes. C'est un peu long, mais je vous assure que ce serait infiniment plus agassif d'avoir à recopier les 476 valeurs numériques. C'est lorsque toutes ces cellules sont indexées en gris clair que l'on va copier dans la fenêtre du **Moniteur** de l'**IDE** les valeurs des 476 échantillons. La technique est très simple.

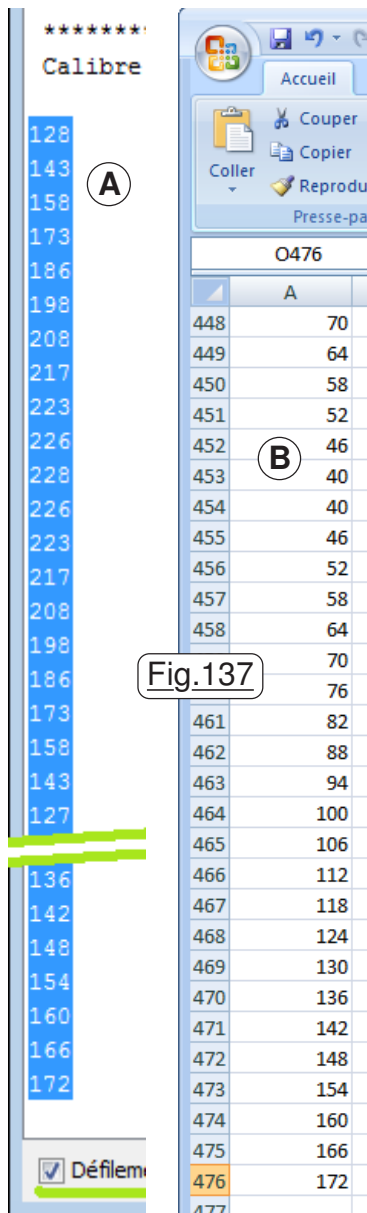
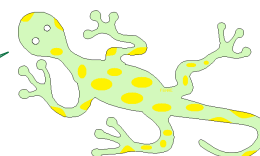


Fig.137

Dans la pratique, pour observer les 476 échantillons listés sur une seule colonne verticale, il faut utiliser "l'ascenseur latéral" pour en explorer toute la hauteur. L'une des caractéristiques vraiment très utile de la fenêtre du **Moniteur de l'IDE** réside dans la possibilité de copier sous forme de textes n'importe quelle zone que l'on sélectionne par balayage *avec le bouton gauche de la souris*. Quand la zone sélectionnée montrée en bleu sur la Fig.137 **A** convient, **[CTRL c]** pour la copier. Puis revenir vers le tableur qui conserve la sélection des 476 cellules dans la colonne de gauche. Utiliser **[CTRL v]** pour y coller toutes ces données ce qui donne le résultat **B**. C'était un tantinet laborieux, tout en restant facile et surtout fiable. Imaginez ce que serait ce travail "à la main" ! Maintenant, présenter ces données sous forme graphique n'est plus qu'une question de connaissance pratique du tableur. Comme je ne sais pas dans **Excel** traiter des tableaux à deux dimensions, j'ai modifié **P17** et adopté le protocole décrit ici. Cliquer avec le bouton gauche de la souris sur le **A** en haut de la colonne. Puis cliquer sur **Insertion > Colonne** ou toute autre présentation possible, et avec les techniques du logiciel **Excel** ou d'un équivalent on réalise le graphe. Le but de ce chapitre était surtout de vous proposer une piste relativement aisée pour extraire les données de notre appareil et de les formater de façon simple pour les fournir à un tableur. La récupération du cadre contenant la représentation graphique est effectuée avec utilisation de la touche **[Impr Écran]** puis d'un **[CTRL v]** dans un outil de dessin aussi simple que **Paint.com** largement suffisant pour ajouter les caractéristiques de saisie des échantillons et séparer les fenêtres par les tracés verticaux vert. Bref, pour la présentation on persévère avec les outils que l'on utilise de façon ordinaire. On peut passer au prochain démonstrateur et aller gloutonner des octets en ajoutant un LOGO "de présentation de l'entreprise".

Môamôa j'interdit que l'on me traite de LOGO car mal prononcé ça ressemble trop à LOLO ou GOGO !



20) Le bénéfice d'une programmation optimisée à outrance.

C'est un phénomène bien connu, la richesse pousse l'individu à dilapider. Plus cette dernière est importante, plus l'on gaspille. C'est vrai dans la vie de tous les jours, mais aussi en programmation et ce projet va dans ce domaine montrer un exemple scandaleux de plus !

➤ **L'exemple à ne surtout pas suivre !**

Créer un dessin sous forme d'une matrice de nombres que l'on intègre dans un programme est à proscrire car la consommation en ressources du microcontrôleur est "catastrophique". Dans ce projet on peut se le permettre car le programme d'utilisation est terminé et qu'il reste encore 30%



Fig.138

d'espace disponible pour le programme et la moitié de la RAM dynamique pour les données. Donc on peut dilapider, ce qui dans ce contexte ludique va nous permettre d'aborder la facette graphique de l'utilisation d'OLED qui dans ce domaine est très particulière. Pour ce petit cheminement complémentaire, on va développer **P18_Page_de_garde_avec_LOGO.ino** qui ajoute le petit dessin de la salamandre quand on affiche la version du logiciel. Cette dernière étant mon avatar graphique pour toutes mes prestations sur l'Internet. Cette signature s'insinue partout y compris en filigrane dans les pages de textes qui manquent d'images ou en bas de page pour proposer un amusement et alléger tous ces propos trop sérieux pour une activité de loisir. **Page 59**

► La toute petite salamandre.



Fig.139

Contrairement à ce qui avait été envisagé dans le chapitre *Un petit dessin pour faire joli !* avec la Fig.10 donné en page 6, ici on ne double pas la taille de l'image. Pourtant, comme le prouve la Fig.138 elle agrémente singulièrement la page de présentation de la version du logiciel. Bien qu'elle soit discrète, elle empêche d'afficher "USB = " du coup six caractères redeviennent disponibles en EEPROM. Pour ne pas

les perdre, ils ont été récupérés et remplacés par le texte "signal" qui va faire économiser autant de caractères dans le programme. **Attention, maintenant dans les démonstrateurs précédents le texte de la page de garde est faussé et affiche "signal57600baud".** Comme c'était le cas pour la version de taille double, pour générer le dessin on s'en tient à un nombre d'octets entiers. Pour économiser des données dans la matrice, la queue du petit animal est complétée par le tracé repéré en rouge sur la Fig.139 obtenu par deux lignes verticales.

0464	i	o	n	l	o	g	i	c	i	e	l	l	e	(F
0480	E	6	A	A	E)	s	i	g	n	a	l	b	a	u
0496	.	C	a	l	i	b	r	e	v	E	x	p			
0512	o	r	t	e	s	u	r	U	S	B	:	S	i		
0528	n	u	s	o	i	d	a	l	.	..	00	02	02	02	02
0544	02	02	02	02	80	8D	9A	A7	B3	BF	CA	D4	DC	E5	EC
0560	F5	F9	FB	FC	FA	F8	F4	F				A	D1	C7	BC
0576	98	8C	7F	73	67	5B	4F	4				8	20	1A	15
0592	0C	0C	0C	0E	12	16	1C	22	2B	33	3C	46	52	5D	68
0608	80	8B	87	A2	AD	B7	C1	CA	D1	D8	DE	E4	E7	FA	FC

Fig.140

► Coder une image pour l'afficheur OLED.

Procédure spécifique au composant utilisé, l'instruction `u8g.firstPage()` commence par remplir la matrice d'affichage de "0" c'est à dire de PIXELs noirs. Ensuite, par les instructions graphiques on va y loger des "1" qui allumeront les pavés relatifs. Par exemple utiliser l'instruction `u8g.drawLine(XA,YA,XB,YB)` va inscrire des "1" pour les PIXELs de la mosaïque qui sont sur la droite qui joint A et B. Pour écrire un dessin sur cette mosaïque la première opération consiste à transformer le portrait de **Bubule** (*Qui a accidentellement été mangé par le chat mais vous vous en fichez.*) en une matrice de carrés noir et blancs comme représenté sur la Fig.141 en supposant sur cet exemple qu'OLED de faible définition ne comporte que huit PIXELs de large et quatre lignes de haut. Notre dessin construit sur une grille dont l'étendue représente la grandeur du dessin occupant l'afficheur doit maintenant être transformé en une matrice binaire comme celle de la Fig.142 où chaque point allumé contiendra un "1" et chaque point éteint sera codé par un "0". Jusqu'ici c'est assez évident,

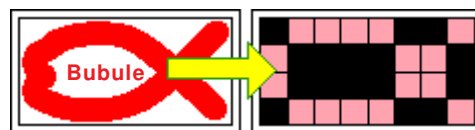


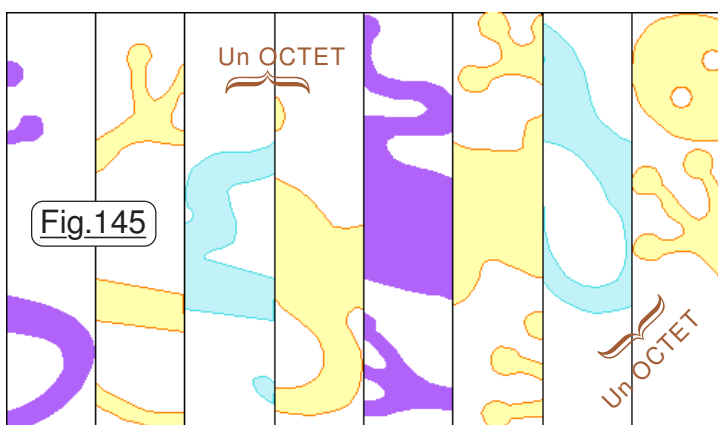
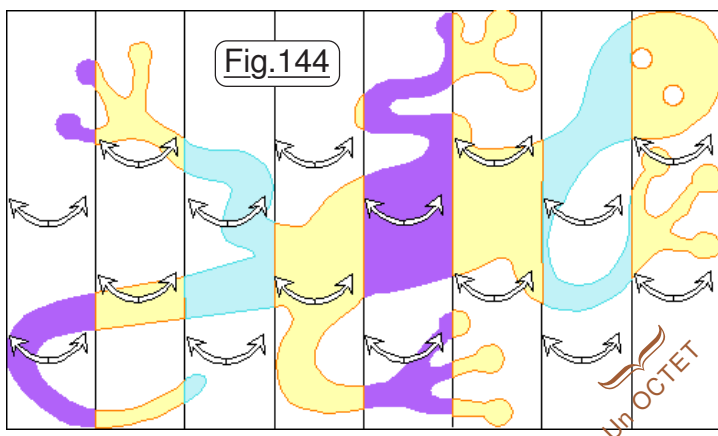
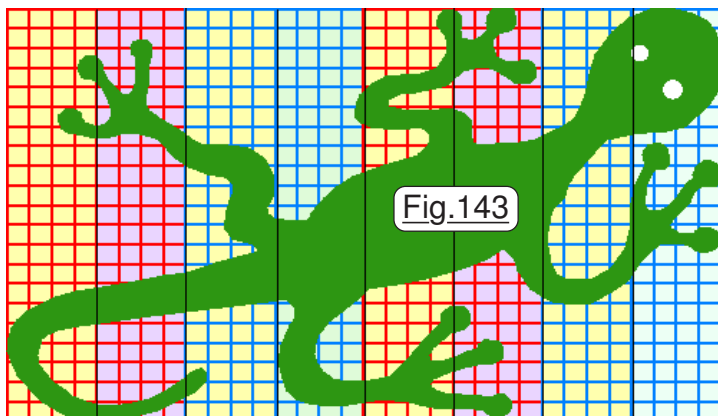
Fig.141

Fig.142

0	1	1	1	1	0	0	1
1	0	0	0	0	1	1	0
1	0	0	0	0	1	1	0
0	1	1	1	1	0	0	1

facile à comprendre `u8g.drawXBM(11, 32, Largeur_du_dessin, Hauteur_du_dessin, Salamandre)`. Les valeurs 11 et 32 correspondent aux coordonnées de l'angle supérieur gauche de la matrice dans la mosaïque complète de l'afficheur. La **Largeur_du_dessin doit correspondre à un nombre entier d'OCTETS**. Cette largeur sera donc un nombre multiple de huit. La **Hauteur_du_dessin** représente le nombre de "lignes de PIXELs". Enfin, **Salamandre** désigne un tableau de byte où les OCTETS sont codés avec les "0" et les "1". C'est ici que se complique singulièrement le codage qui doit "découper en tranches de demi-octets" le dessin avec symétrisation verticale. Glups !

Pas de panique, le principe reste assez élémentaire. On commence comme sur la Fig.143 par tracer une grille qui correspond exactement à celle du dessin que l'on désire construire. Normalement, sur la Fig.143 il devrait y avoir cinq octets de largeur. J'en ai représenté que quatre pour que le dessin ne soit pas trop confus. Il devrait y avoir également 24 lignes alors qu'il n'y en a que 21 pour que les carreaux de la mosaïque soient presque carrés. Lorsque la grille est réalisée, on l'imprime en grand sur un format A4 et on trace le contour du LOGO sur cette dernière. Puis, c'est un travail d'artiste, on noirci les cases dans lesquelles passe le contour en minimisant l'effet d'escalier. Il faudra coder par des "1" les PIXELs qui sont sur ce contour, ainsi que ceux qui



```
byte Salamandre[120] = {
B00000000, B00000000, B00000000, B00001001, B01110000,
B00000000, B00000000, B10000000, B00000100, B11111100,
B01000000, B00000000, B10000000, B00000010, B11110110,
B01001000, B00000000, B10000000, B00011111, B11111110,
B01010000, B00000000, B11100000, B00000001, B11011111,
B01100000, B00000000, B00110000, B00000000, B01111111,
B11100100, B00000000, B11111000, B10000001, B00111111,
B10011000, B00000011, B10000000, B11110001, B00000011,
B00000000, B00011110, B11100000, B11111111, B00010001,
B00000000, B00111100, B11111100, B11111111, B10001000,
B00000000, B00110000, B11111111, B01111111, B01001000,
B00000000, B10111000, B11111111, B00111111, B00101000,
B00000000, B11111000, B11111111, B00111111, B00011000,
B00000000, B11110000, B11111111, B01111111, B11111000,
B00000000, B11111000, B11111111, B01111111, B00000100,
B00000000, B11111110, B11111111, B01100001, B00000110,
B11110000, B11111111, B00111111, B11000000, B00000011,
B11111111, B00111111, B00000111, B11000000, B00000001,
B11111111, B00001111, B00000111, B00000001, B00000000,
B00000001, B10000000, B10000011, B00000000, B00000000,
B00000000, B11000000, B01000001, B00000100, B00000000,
B00000001, B11000010, B11100000, B00000011, B00000000,
B00000011, B11000001, B01111111, B00000000, B00000000,
B11111110, B10000000, B11000111, B00000111, B00000000};
```

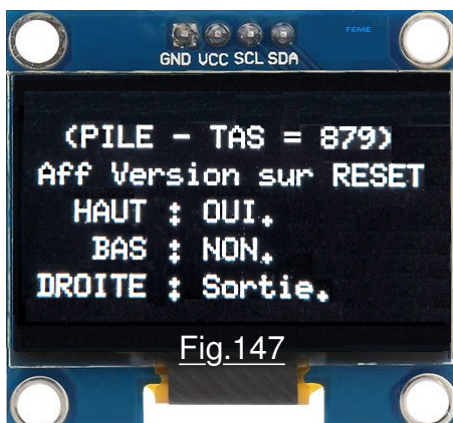
sont à l'intérieur mis à part les yeux du petit animal. On se retrouve avec un carrelage binaire de 40 x 24 dalles l'extérieur en non rempli, l'intérieur noirci au crayon. C'est un peu laborieux car ça représente tout de même 960 carreaux. *(Personnellement je ne noirci que le contour extérieur et les mirettes.)*

Riche de ce modèle on le cloisonne en huit tranches verticales. C'est maintenant que l'opération devient franchement rébarbative. Il faut coder les "0" et les "1" en les regroupant par OCTETS sachant que chaque tranche doit être symétrisée verticalement. C'est un travail long et qui exige beaucoup d'attention. Il faut donc l'aborder avec calme et sérénité. Si on désire doubler les dimensions de notre œuvre


d'art, c'est 3840 BITS qu'il faudra traduire. On comprend assez bien pourquoi on va éviter de se "cogner" une fresque de 3m sur 2m ! Une fois que l'on a transcodé l'ensemble, on aboutit au tableau de **byte** de la Fig.146 qui est déverminé car vous allez rapidement vous rendre compte que lorsque vous désirerez le remplacer par un LOGO personnel, il va y avoir une foule de petites erreurs dont on a un mal fou à retrouver la position dans ce fatras. Ceci dit, une fois que la matrice est parfaite et que l'on a abouti, la satisfaction est à la hauteur des difficultés rencontrées. C'est à vous maintenant de prendre la palette et le pinceau informatique, et de développer votre propre avatar.

➤ Le prix à payer.

Intégrer un dessin dans un programme n'est pas du tout gratuit et d'autant plus pénalisant qu'il n'est pas logé en EEPROM. À l'instar des chaînes de caractère, il est traduit sous forme d'un tableau. Dans ce projet ce dernier contient 120 **bytes**, alors la taille du programme enflé d'autant. *(Plus les routines de traitement mais l'on ne peut pas s'en passer.)* Surtout, comme vous pouvez le vérifier sur la Fig.147 l'espace disponible entre la **PILE** et le **TAS** diminue d'autant. Conclusion : Autant que faire ce peut il faut impérativement loger la matrice du dessin dans l'EEPROM pour éliminer ces deux conséquences.



➤ Une grille spécifique à l'afficheur OLED.

Avec les outils graphiques disponibles sur l'ordinateur, il est particulièrement facile de créer une grille de 128 colonnes et de 64 lignes. On ajoute à cette dernière des indications des coordonnées tous les huit PIXELs et l'on obtient le fichier  GRILLE pour OLED.pdf à imprimer que je vous livre dans le dossier <Documents>. Il ne faut pas croire que l'on développe des écrans graphiques tel que celui de la Fig.65 par exemple, d'un simple claquement des doigts. Dès que la police de caractères est connue et les dimensions de ces derniers notés, on trace sur la grille de référence les divers cadres dont on optimise les surfaces. Puis on ajoute les textes en vérifiant qu'ils ne débordent ni de leurs encadrements ni des limites de l'afficheur. *C'est un travail préalable absolument indispensable* sans lequel on va consommer en pure perte un temps considérable lors de la programmation. Vous avez l'outil, à vous de vous en servir ...

21) Une dernière petite surprise avant de se quitter.

Observant le thermomètre de la Fig.148 on constate une fois de plus qu'ajouter une fonction pourtant "cossue" avec gaspillage de place en mémoire, l'augmentation rouge reste dérisoire. À peine 1%. À ce taux d'inflation on pourrait encore en ajouter 28 de consistance analogue. (*Sauf qu'il ne faudrait pas trop de tableaux ou le ratio PILE moins le TAS deviendrait négatif.*) On constate à chaque ajout de fonctions dans le programme que l'on s'approche de moins en moins rapidement de la saturation. Il faut vraiment un projet très complexe pour arriver à consommer les 100% de la zone programme. Avec P19_Version_UTILISATION.ino on termine notre cheminement dont le but était d'explorer avec méthode diverses facettes du développement en C++. Il y a toutefois un chapitre pourtant fondamental qui n'a pas été abordé : L'utilisation des opérations avec masques LOGIQUES. Aussi je vous propose un exemple ludique, car une petite surprise vous attend si dans le Menu du RESET vous activez la touche HAUT avec un *clic long*. (*Ben allez-y, faites-le !*)

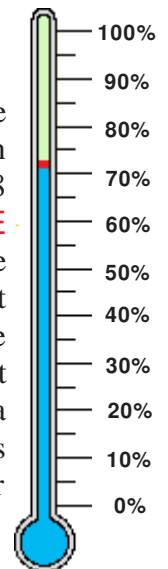


Fig.148

➤ Le domaine des MASQUES LOGIQUES.

Fréquentant depuis de longues années des programmeurs amateurs, force est de constater que les si les opérateurs booléens tel que le OU et le ET sont bien assimilés, il en est autrement pour l'usage des masques logiques. C'est particulièrement vrai quand on programme en langage évolué comme BASIC, PASCAL ou C++ etc. Généralement des expressions comme :

- Si (Condition 1) OU (Condition 2) faire Action 1 Sinon faire Action 2. } sont bien
- Si (Condition 1) ET (Condition 2) faire Action 1 Sinon faire Action 2. } assimilées.

Il en est tout autrement de l'usage du OU Exclusif. Par exemple que signifie ?

- Si (Condition 1) OU Exclusif (Condition 2) faire Action 1 Sinon faire Action 2 ? ? ?

C'est la pratique de la programmation "au raz de la machine" notamment en ASSEMBLEUR qui généralise l'usage des opérateurs LOGIQUES BIT à BIT et tout particulièrement l'intervention des masques logiques. Pourtant, comme on va le voir dans l'exemple de P19 ce type de programmation peut rendre de signalés services également dans les langages évolués. (*On oubliera ici les opérateurs de décalage dont on a fait appel plusieurs fois dans ce petit projet. Ils permettent facilement de multiplier ou diviser rapidement par des multiples de deux.*) Nous allons dans ce chapitre passer en revue le domaine d'application des masques logiques et des opérateurs associés. On peut déjà noter que :

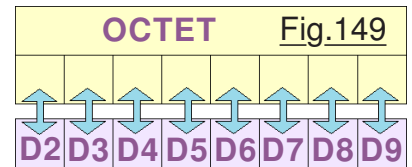
Les Masques et les opérateurs LOGIQUES servent à agir individuellement sur des BITS dans des données globales telles que les byte et les int, les long etc.



Je suis offuscationnée. Que l'on me découpe en tranche à la rigueur c'est acceptable surtout si c'est pour la bonne cause. Mais que l'on me transforme en cette chose de la Fig.145 c'est TOTALEMENT INACCEPTABLE !

➤ La modification d'un OCTET BIT à BIT.

Bien que ce chapitre prend en exemple des données de huit BITS, on peut généraliser sans autre forme de procès à des variables entières de taille quelconque. Par exemple on va raisonner sur une variable **OCTET** qui dans une procédure adaptée sert à lire ou écrire directement les huit BITS des broches binaires allant de **D2** à **D9** comme montré sur la Fig.149 donnée ci-contre. L'instruction suivante **OCTET = DATA** recopiera dans les huit BITS de sortie d'**OCTET** ceux de la variable **DATA**. Réciproquement **DATA = OCTET** recopiera dans **DATA** les huit BITS des entrées BINAIRES d'**OCTET**. Ici l'opérateur "=" agit d'un seul coup sur l'intégralité du groupement nommé **OCTET**. On va maintenant voir comment à l'aide des opérateurs et des **Masques LOGIQUES** on peut agir sur des BITS individuels.



Masque_LOGIQUE								A	Masque_LOGIQUE									Masque_LOGIQUE							
0	1	0	1	0	1	0	0		0	1	0	1	0	1	0	0		0	1	0	1	0	1	0	0
ET								B	OU								Fig.150	OU Exclusif							
1	1	1	0	0	0	1	1		1	1	1	0	0	0	1	1		1	1	1	0	0	0	1	1
0	1	0	0	0	0	0	0		1	1	1	1	0	1	0	0		1	0	1	1	0	1	1	1
0	1	0	0	0	0	0	0		1	1	1	1	0	1	0	0		1	1	1	0	0	1	1	1

Le tableau de la Fig.150 propose trois exemples, mais pour bien saisir les transformations qui résultent de ces opérateurs reportez-vous sur la Fig.151 au résumé de l'emploi des **Masque LOGIQUE** :

Sur la Fig.150 la ligne **A** correspond à un **Masque LOGIQUE** commun affecté aux trois exemples. Sur la ligne **B** trois fois la même données sur laquelle est appliqué l'opérateur. En **C** est affiché le résultat de l'opération. Dans l'exemple avec le **ET**, supposons que les huit sorties BINAIRES de **OCTET** pilotent des LEDs. Et bien cette instruction va éteindre **D2**, **D4**, **D8** et **D9** sans modifier les autres sorties. Si on exécute une deuxième fois l'opération sur cette donnée **OCTET** modifiée en **C**, avec le même **Masque LOGIQUE** on constate sur la ligne **D** que l'on ne retrouve pas l'état initial du BIT concerné. La donnée initiale est définitivement perdue car il est impossible d'en retrouver la

- Un **ET** force un "0" sur la donnée lorsque dans le **Masque_LOGIQUE** il y a un "0".
- Un **OU** force un "1" sur la donnée lorsque dans le **Masque_LOGIQUE** il y a un "1".
- Un **OU Exclusif inverse l'état du BIT** qui dans le **Masque_LOGIQUE** est a un "1".
- Un **OU Exclusif** effectué deux fois avec le même **Masque_LOGIQUE** restitue l'état initial.

☞ Ces diverses opérations se font BIT à BIT sur la totalité de la donnée modifiée.

combinaison de départ. Le **ET** associé à un **Masque LOGIQUE** sert également à tester un BIT individuel sur une donnée globale. Supposons par exemple qu'**OCTET** a été initialisé en huit entrées binaires reliées à des capteurs. On désire savoir dans quel état se trouve celui qui est branché sur **D5** par exemple. Il suffit dans la donnée lue de forcer tous les autres BITS à "0". L'instruction devient : **Entrees = OCTET; if (Entrees && B00010000) then ...**

Vous avez certainement compris que l'opérateur **OU** qui force des "1" individuellement va servir à piloter des sorties binaires individuellement. Si on désire mettre en marche le moteur (*Ou la LED etc.*) branché sur **D7** il suffit d'un **OU** avec **00000100**. Pour le stopper c'est **ET** associé à **11111011** qui servira de masque. Il nous reste encore à passer en revue le domaine d'exploitation classique du **OU Exclusif**. On a vu en Fig.151, ce que confirme l'exemple de droite de la Fig.150 que cet opérateur sert à inverser les BITS correspondant lorsque dans le **Masque LOGIQUE** on a placé des "1". Du coup on peut inverser l'état d'un système sans se préoccuper de son état initial. C'est exactement ce que font sans que nous le sachions des instructions de type :

```
digitalWrite(LED_Arduino, !digitalRead(LED_Arduino)); delay(1000));
if(!Clic_long) Action ...
```

La première instruction inverse l'état de la LED de **D13** une fois par seconde. Concrètement, dans ces deux instructions, l'opérateur d'inversion d'état qui en C++ est symbolisé par "!" génère pour le microcontrôleur une opération élémentaire avec un **OU Exclusif** et un masque logique. **Page 63**

Il me semble important de vous faire remarquer que la ligne relative au clignotement de la LED d'Arduino ne fonctionne correctement que par le fait qu'un OU **Exclusif** appliqué deux fois sur la même donnée avec un **Masque LOGIQUE** identique restitue cette dernière. Du coup on obtient bien une alternance d'état sur une paire d'exécution puis le cycle se poursuit.

A vant de passer à la suite, examinons ensembles une application banale en informatique de l'opérateur **ET** pour transformer un texte saisi en minuscules en majuscules. Par exemple vous développez un projet pour lequel les directives sont envoyées au dispositif par l'usage du **Moniteur de l'IDE**. Chaque commande est obtenue par un caractère. Par exemple si l'utilisateur frappe 'E' il effacera ses données. Vous désirez que cet usagé ne soit pas forcément obligé d'enfoncer la touche [MAJ]. Si il frappe 'e', votre programme devra le transformer en 'E'. En codage ASCII c'est le BIT n°5 qui à "1" correspond à une minuscule, et à "0" donne son équivalent majuscule. L'instruction du genre **Caractere = Caractere & 32;** transformera toute minuscule en majuscule. ATTENTION : cette transformation n'est correcte que si le clavier utilisé retourne des codes ASCII et non de l'EBCDIC ou des codes clavier "IBM" ...

➤ Un tout petit "chouilla de cryptographie."

D omaine tellement vaste que des dizaines de didacticiens ne feraient qu'à peine l'aborder, on va ici cacher dans le programme du texte qui sera crypté. Le but de cette approche était de justifier ce chapitre sur les opérateurs logiques. N'oublions pas que ce n'est pas tellement la possession de ce petit oscilloscope aux performances modestes qui constitue le facteur déclenchant de ce projet, mais le plaisir de la programmation sur Arduino en C++. Aussi, lorsque vous allez déclencher la petite surprise avec la touche HAUT **clic long** du **Menu du RESET**, la page écran va afficher un texte comme pour tous les autres menus. Pour ces derniers, les textes affichés sont directement écrits dans le programme ou en EEPROM. par contre, pour cette page écran inutile de chercher les chaînes de caractère dans le croquis. Les lettres ont été cryptées de façon simple. Pour la substitution j'ai utilisé un simple opérateur OU **Exclusif** associé au **Masque_LOGIQUE** suivant :

```
#define Masque_LOGIQUE B00111110 // Inversera cinq BITS.
char Texte_CRYPTÉ[98] = {'s','[','P','K',' ','Z','[',' ','R','_',' ','m','k','l','n','l',
    'w','m','{',' ','v','A','k','j',' ',' ','n','Q','M','J','M','G','P',' ','V','L',
    'Q','P','W',' ','-',' ','M','_','J','W','Q','P',' ','N','_','L',' ','M','_','J','[','R',
    'R','W','J','[',' ','A','m',' ',' ',' ','N','J','K','L','[',' ','n','v','A',
    'm','{',' ','z','l','q','w','j','{',' ',' ','l','[','J','Q','K','L',' '};
```

Fig.153

P our pouvoir appliquer sur les caractères affichés le masque logique caractère par caractère, les textes sont fournis sous forme du tableau de **char** de la Fig.153 nommé **Texte_CRYPTÉ**. Ainsi pour générer la page OLED on va extraire caractère par caractère et appliquer l'opération :

Caractere = char(Caractere ^ Masque_LOGIQUE);

En C++ le OU **Exclusif** se code '^'. Profitant du fait que cet opérateur fonctionne "dans les deux sens", dans un premier temps j'ai rempli le tableau **Texte_CRYPTÉ** avec le texte en clair. L'écran OLED a alors affiché un texte correspondant à celui de la

s[PK Z[R_ mklnl
wm{.vAkj : nQMJMGPJVL
QPW--M_JWQP N_L M_J[R
RWJ[Am : }_NJKL[nvA
m{.zlqwj{ : l[JQKL. Fig.154

Fig.154 avec certains comme l'espace ou le point qui étaient non affichés. C'est la raison pour laquelle l'opérateur '^' n'est pas appliqué sur cinq caractères **Particuliers**. Il m'a suffi de remplacer dans le tableau les textes en clair par ceux de la Fig.154 affichés sur OLED et le cryptage était effectué sans se tordre les méninges. Du reste, pour que vous puissiez voir ce que donne l'opérateur sur n'importe quel caractère, avec le **Moniteur de l'IDE** je vous propose **Transcodage_par_masque_logique.ino** ajouté aux autres démonstrateurs. **Finalement, nous ne sommes pas encore au terme de ce petit cheminement ludique** dans les sentiers du C++ et le thermomètre montre que cette version n'arrive pas à 80% d'espace programme utilisé. La zone verte disponible est encore scandaleusement grande. Il se trouve qu'**en rédigeant ce chapitre, une idée intéressante a émergé** avec l'espoir d'améliorer encore la "rentabilité du projet". Alors sur le métier on replace l'ouvrage et l'on reprend le joug. C'est d'autant plus "émoustillant" que **cet ajout va vraiment améliorer les performances** de notre petit joujou.

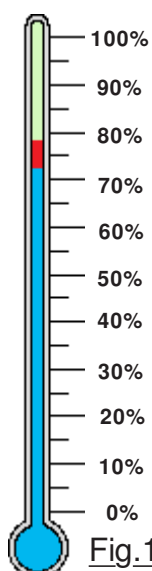


Fig.155

22) Accepter un signal alternatif sans le déformer.

Contrairement à ce que laisserait à penser la Fig.49 en page 28 ou la Fig.65 en page 33 notre appareil tel qu'il a été décrit jusqu'ici n'est pas apte à visionner un signal alternatif car la sécurité d'inversion de polarité rabote toute tension négative en entrée. Les signaux obtenus sur les exemples cités ou la belle sinusoïde de la Fig.132 A en page 36 sont obtenus avec un générateur capable de superposer à l'onde

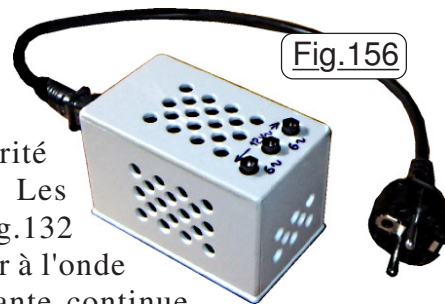


Fig.156

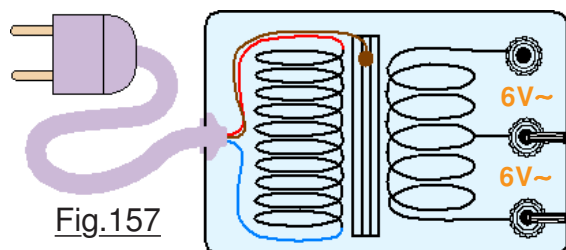


Fig.157

alternative une composante continue

"d'offset". En revanche, si à partir du secteur on enregistre la sortie d'un petit transformateur basse tension comme celui de la Fig.156 avec isolement, le schéma étant donné en Fig.157 on visualise des signaux rabotés tels que ceux montrés en Fig.117

page 49. Aussi, la seule façon d'éviter cet inconvénient consiste à ajouter une composante continue positive au signal examiné, exactement comme le fait le générateur utilisé dans ce qui précède.

➤ La modification du schéma électronique d'entrée.

Changer l'électronique pour satisfaire cette amélioration ne devait pas conduire à un sous-système compliqué. Au contraire, l'appareil n'étant qu'un amusement informatique, envisager un mélangeur à haute impédance sophistiqué était hors propos. Tout au plus trois à cinq composants peu onéreux qui puissent trouver leur place sur le circuit imprimé actuel constituait la base de cette étude. Suite à ces modifications, le schéma général donné en Fig.18 page 14 ainsi que le dessin de

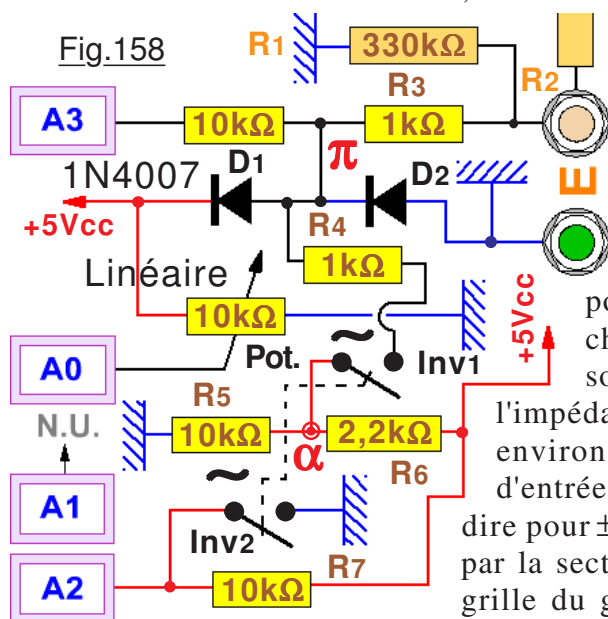


Fig.158

la Fig.26 en page 17 ont été mis à jours et correspondent au circuit actuel. Pour décortiquer la modification effectuée, on va se reporter sur la Fig.158 qui est un extrait du schéma dans la zone concernée.

Les deux résistances **R5** et **R6** constituent un diviseur de tension pour amener à environ $+2V_{cc}$

le point π . Quand on bascule l'inverseur double en position travail, la tension en α , donc également en π chute à $+2V_{cc}$ car maintenant **R5** et l'impédance de la source en **E** sont en parallèle via **R3** et **R4**. Du coup,

l'impédance d'entrée de l'oscilloscope passe d'environ $330K\Omega$ à environ $12k\Omega$. Ce n'est pas sans conséquence sur le Calibre d'entrée. Maintenant la pleine déviation se fait pour 4V, c'est à dire pour $\pm 2V$ en entrée **E** quand la tension "d'offset" est appliquée par la section **Inv1**. Du coup, il devient impératif de modifier la grille du graticule. Pour que ce

soit automatique, il faut que le logiciel soit informé de la configuration "Alternatif". C'est la raison d'utilisation de l'entrée analogique **A2** et de la section **Inv2**. En configuration banale **A2** est forcée à $+5V_{cc}$ par **R7** et sera lue comme un état "1". Quand on fait passer l'inverseur en position travail, le mode "Alternatif" est configuré et **A2** se retrouve sur **GND** qui engendre un état "0". Le logiciel affiche alors le graticule particulier de la Fig.159 et précise par rapport à la ligne zéro en **A** la tension présente en entrée **E** avec sa polarité positive en **B** et négative en **D**. Ce nouveau graticule ne permet plus de visualiser l'index qui indique laquelle des quatre zones en largeur est affichée. Aussi, en **C** dans le petit cadre est précisée cette information. La modification du circuit imprimée reste modeste mais a imposé de placer la résistance **R4** sur le dessous du circuit imprimé principal car sur le dessus il n'y avait pas la place. Allez observer Image 23.JPG à Image 28.JPG. En particulier sur la photographie d'Image 28.JPG on situe bien la position de **R4** sur le dessous coté cuivre.

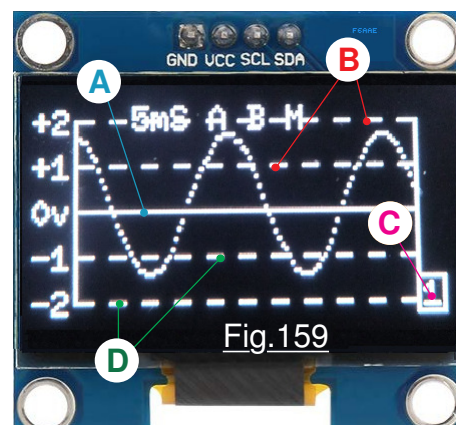


Fig.159

► **Difficulté supplémentaire que présente ce nouveau schéma.**

L'impédance interne de la source de tension analysée en entrée **E** de l'oscilloscope vient charger le diviseur de tension constitué de **R5** et **R6**. Du coup la tension en **A**, diminue au fur et à mesure que le signal analysé est de faible amplitude. Il en résulte une baisse de la valeur de la tension continue d'offset. Plus la tension alternative est de faible amplitude, plus le décalage de la trace vers le bas montré sur la Fig.160 augmente et fausse l'observation. Dans cet exemple la

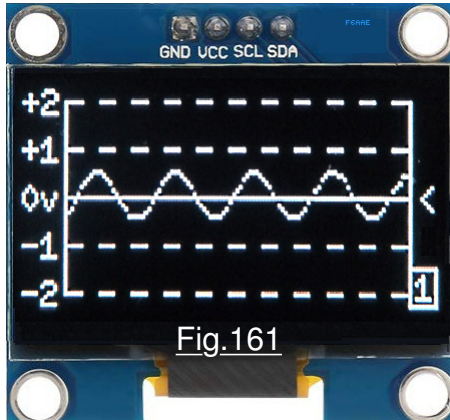


Fig.161

tension efficace semble valoir -0,5V alors que dans la réalité elle est nulle. Pour corriger ce problème, il faut ajouter à la tension continue d'offset générée par le pont diviseur de **R5** et de **R6** une **tension de compensation** directement fonction de l'amplitude du

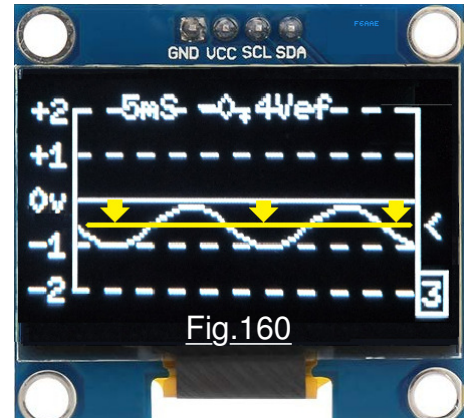


Fig.160

signal alternatif introduit en entrée **E**. Cette tension de compensation est nommée CV (Pour **C**ompensation **V**erticale.) dans la procédure **Calcule_la_compensation_verticale()**. On peut vérifier sur la Fig.161 et sur la Fig.164 que le problème est correctement évacué. La photographie de la Fig.163 a été saisie en mode normal avec un signal alternatif de faible amplitude. Pour améliorer la capture des échantillons l'amplification maximale a été appliquée. La Fig.164 en atteste. La base

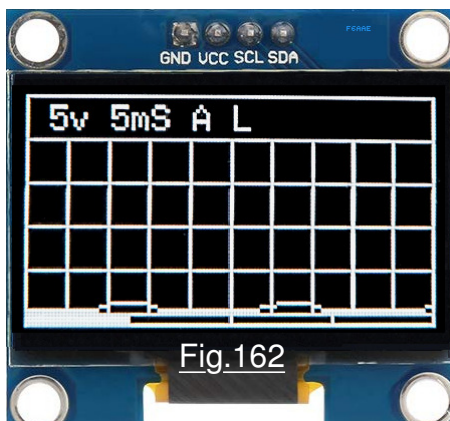


Fig.162

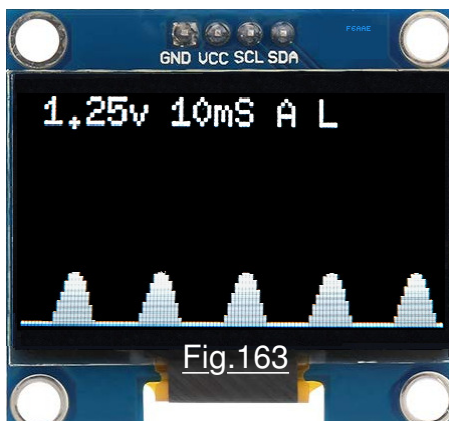


Fig.163

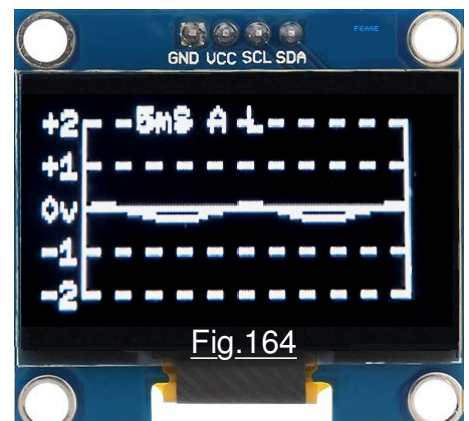


Fig.164

temps a été modifiée, le graticule n'est plus affiché et le mode "surface" est privilégié. Sur l'image de la Fig.164 (*Désolé pour le bougé durant la prise de vue effectuée en "macro".*) le calibre vertical est de $\pm 2v$ car on est en mode Alternatif avec composante continue superposée. Bien que de faible amplitude le signal est correctement représenté sans le "raboitage" visible sur les deux images précédentes. Deux autres exemples sont présentés sur la Fig.165 et sur la Fig.166 avec une tension alternative sinusoïdale le plus forte amplitude. Noter au passage que les images de Fig.162 à Fig.165 sont prise en mode capture d'échantillons alors que sur la

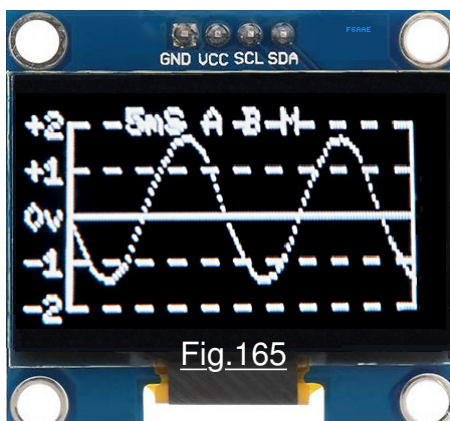


Fig.165

Fig.166 c'est en mode affichage de la trace raison pour laquelle figure sur cette dernière la valeur de la tension efficace et la position de l'index de celle-ci. **La valeur efficace est bien celle du signal, la composante continue d'offset étant déduite.**

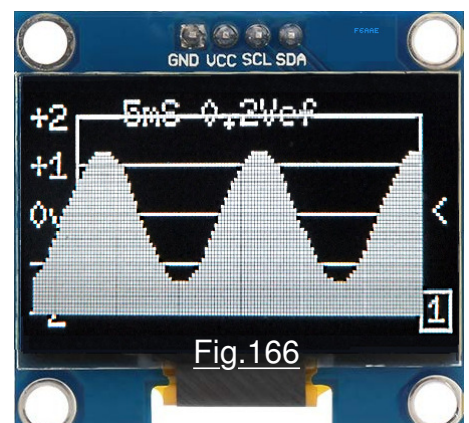


Fig.166

Enfin dans toutes ces vues les paramètres d'affichage initialisés lors de l'affichage de la trace sont mémorisés et se retrouvent en capture d'échantillons. (*Mode filaire ou mode surface ...*)

➤ **Calibre vertical constant.**

L'offset étant généré par un pont intégrant deux résistances de valeurs fixes, les deux volts superposés sont par conséquent constants. Hors l'amplification logicielle se fait sur les valeurs mesurées qui sont donc influencées par cette tension positive. Le gain en tension par programme s'appliquerait sur cette composante continue et conduirait à une saturation des signaux positifs. L'amplification logicielle est donc prohibée en mode alternatif. Du coup, si durant une saisie d'échantillons on génère un **clac long** sur la touche de DROITE l'écran de la Fig.167 s'affiche sur OLED et un BIP sonore d'alerte se fait entendre. La LED triple s'illumine en rouge et le logiciel attend que l'on frappe une touche du petit clavier pour sortir et revenir au mode échantillonnage.

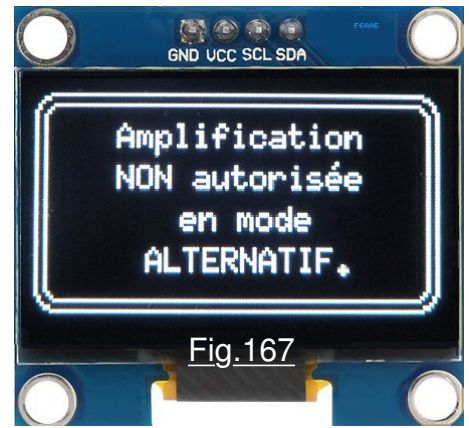


Fig.167

NOTE : Dans la version ultime du logiciel lorsque le programme attend la frappe d'une touche pour sortir d'une fonction, la LED jaune clignote rapidement. (Environ 10Hz.)

➤ **Une autre option d'affichage.**

Adopter des lignes pointillées comme celles **B** et **D** de la Fig.159 permet au premier coup d'œil de repérer le mode **ALTERNATIF** tant en capture d'échantillons qu'en visualisation de la TRACE. Toutefois, si ce mode d'affichage du graticule est parfait pour des signaux sinusoïdaux, en dents de scie, en ondes triangulaires, il devient particulièrement malcommode à analyser quand le signal injecté en entrée **E** de l'oscilloscope est binaire comme en témoigne la Fig.168 en **A**. on a du mal à faire la distinction entre les lignes pointillées et les créneaux haut et bas des impulsions électriques. (Sur ces exemples c'est le signal étalon fourni par notre appareil qui est utilisé.) Et encore, le cas devient pire si la largeur des "pointillés électriques" devient équivalente à celles des lignes du graticule. Aussi, si en visualisation de la trace on effectue un **clac long** sur la touche du HAUT, les lignes horizontales du graticule deviennent continues comme en **B** ce qui est nettement mieux. Avec un **clac court** sur HAUT on enlève les textes et en **C** la trace est encore plus facile à observer. Avec un **clac long** sur BAS la présentation en mode surface en **D** n'est ensuite qu'une question de choix personnel. Enfin **clac court** sur BAS en configuration **C** il ne reste alors plus que le tracé du signal, c'est à dire la forme la plus épurée disponible sur notre appareil.

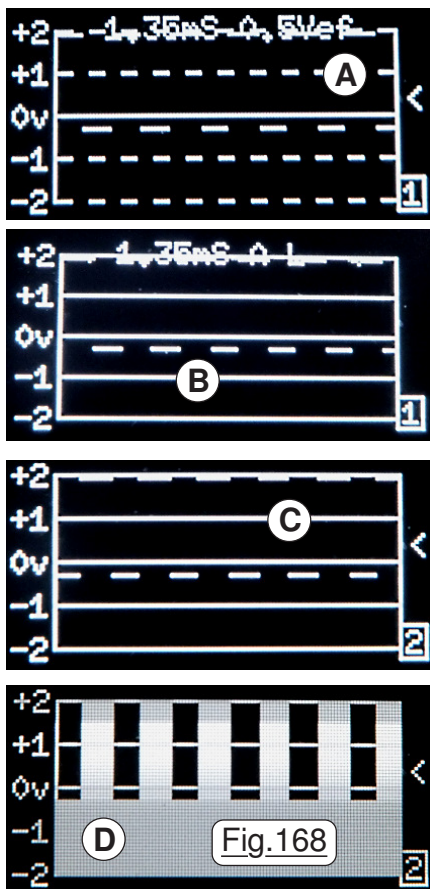


Fig.168

➤ **Une amélioration douteuse.**

Histoire de distinguer immédiatement le **MENU de BASE** des autres pages de sélection des fonctions, le titre a été agrémenté d'un cadre ce que montre la Fig.169 une modification vraiment mineure. Les deux disques de part et d'autre de ce cadre ne font que renforcer la différence avec les autres pages-écran. Ce n'est pas forcément très esthétique, le but étant de démontrer l'influence d'une aussi modeste modification du programme. En effet, on peut constater en tête du listage de **P20** que ces deux disques font augmenter la taille du logiciel de 436 octets ce qui est considérable. Le but de ce petit détail ajouté en dernière minute au croquis est d'illustrer sur un exemple concret le bienfondé de la stratégie **Stratégie d'utilisation de la bibliothèque U8glib** proposée en haut de la page 45.



Fig.169

23) Résumé de quelques conseil pour programmer de façon méthodique.

Probablement trop long aura été cette saga informatique. Le but fondamental reste dans la pratique du langage C++ et dans les techniques et les conseils particuliers proposés dans cette narration. Ces points singuliers sont du coup trop éparpillés, et il m'a semblé très utile d'en élaborer ici une liste relativement complète avant de nous quitter :

- Avant de programmer **il faut absolument consacrer du temps pour déterminer relativement finement ce que l'on désire exactement obtenir** et en prouver la faisabilité.
 - **La première qualité d'un logiciel quel qu'il soit est sa LISIBILITÉ.**
 - Les lignes qui doivent facilement se retrouver seront terminées par // @ @ @ @ @ @ @ @ @ @ @ @.
 - Placer un maximum de commentaires pour expliciter certains calculs, certains choix etc.
 - Pour une lisibilité maximale du listage, **toujours choisir avec beaucoup d'attention les noms des identificateurs des constantes et des variables** avec des noms qui "parlent".
 - Toutes les déclarations sont placées en tête de programme.
 - Éviter autant que possible les séquences très longues imposant des défilements du listage sur l'écran de l'ordinateur. Une procédure ne devrait jamais dépasser "la taille verticale" de l'affichage.
 - Préciser en tête de listage le but précis du programme, ainsi que son comportement attendu.
 - Ajouter en tête de listage les branchements à effectuer.
 - Préciser en tête de listage la date de dernière modification du programme, la taille du code et celle des données en mémoire dynamise. (*Fonction de la version à préciser du compilateur.*)
 - Il est fortement recommandé de placer toute fonction ou procédure avant celle qui y fait appel.
 - Pour structurer proprement les constantes et les variables utilisées par le logiciel :
 - * Les classer par type et globalement par ordre alphabétique.
 - * Les lister par tailles croissante : **boolean, char, byte, int, long, float, double, tableaux ...**
 - **Affectation judicieuse des Entrées / Sorties** : Voir le cadre de la Page 9.
 - Dans **void setup** initialiser en premier les broches d'E/S. Vers la fin initialiser alors les variables en classant les instructions par l'ordre alphabétique des identificateurs.
 - **Optimiser le code doit virer à l'obsession** :
 - * Optimiser le code c'est **Minimiser la taille du code** objet qui encombrera la mémoire,
 - * Optimiser le code c'est **Minimiser le temps d'exécution** d'une séquence critique.
 - **Optimisation de la taille des variables** : **il est absolument indispensable de choisir le type de chaque variable pour en minimiser la place occupée en mémoire dynamique et en accélérer le traitement.** C'est un impératif absolu. Du bon choix du type des données dépend considérablement la taille occupée par le programme et le temps d'exécution. **ATTENTION**, les tableaux sont les données les plus voraces en espace utilisé et les textes sont intrinsèquement des tableaux. Aussi, **si la mémoire non volatile EEPROM n'est pas utilisée pour mémoriser des données** à conserver sur coupure alimentation, **y logger un maximum de textes** à afficher sur les écrans.
 - Doubler par logiciel le nombre de touches d'un clavier :

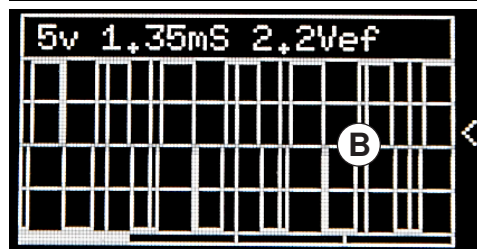
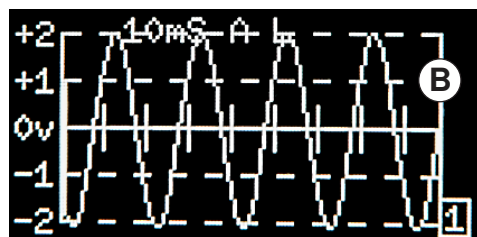
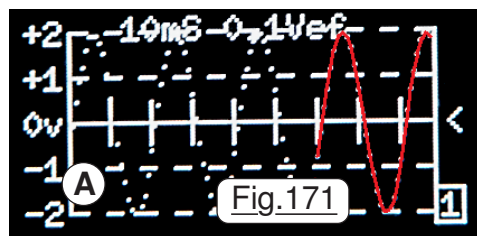
- * Si la touche ou le B.P. est cliqué un court instant on génère l'effet n°1 : **Clic court.**
 - * Si l'enfoncement dépasse un délai calibré on déclenche l'effet n°2 : **Clic long.**
 - Tripler par logiciel le nombre de touches d'un clavier :

- * **Si sur RESET une touche est active on génère un effet particulier.**
 - Pour diverses raisons **la boucle de base doit "tourner" à grande vitesse.** (*Rapidité de prise en compte d'une consigne opérateur, rapidité des affichages etc.*) Le programme étant "terminé", il est fortement conseillé d'en mesurer la fréquence d'exécution.
 - Un moyen incontournable pour diminuer les temps de développement consiste à **SIMULER LES DONNÉES** au lieu de les mesurer, et à diminuer certains délais de comportement initialisés.
 - Il est conseillé en développement de placer dans la boucle de base un "stéthoscope" pour s'assurer à tout moment que le logiciel n'est pas enlisé dans une séquence très longue ou infinie.
 - **Si des paramètres dans des procédures sont susceptibles d'être modifiés** à la mise au point, il est de loin conseillé de **les définir dans des constantes en tête de listage.**
- Page 68

24) Un chapitre qui bouscule l'espace temps !

Numéroter 24 ce chapitre est une tromperie, car dans la réalité cette page a été rédigée bien après le suivant qui est ordonné n°25. S'il est placé avant, c'est pour éviter de chambouler la mise en page et continuer à terminer sur les conclusions. C'est en rédigeant le document **Prise en main initiale.PDF** que je me suis rendu compte d'une grande faiblesse dans **P20**. En effet, j'ai réalisé un peu tard, que le graticule en mode alternatif n'avait pas de graduations "en largeur" pour pouvoir mesurer les durées. Par exemple sur la Fig.170 **A** il est impossible d'évaluer la valeur de la période. C'est la raison pour laquelle des graduations verticales ont été ajoutées sur la ligne médiane du graticule comme montré sur la Fig.170 en **B**. Bien que cette photographie a été saisie alors que l'on n'a pas assuré une coïncidence entre les pics des alternances et le graticule, on peut observer que la période occupe approximativement 4 graduations et présente une valeur de $4 \times 5 = 20\text{ms}$. On en déduit une fréquence de 50Hz.

Autre tromperie qui est passée inaperçue : Oser nommer de filaire la représentation avec des PIXELs telle que celle de la Fig.171 en **A** ici numérisée en mode alternatif. On peut constater que si l'on rejoint



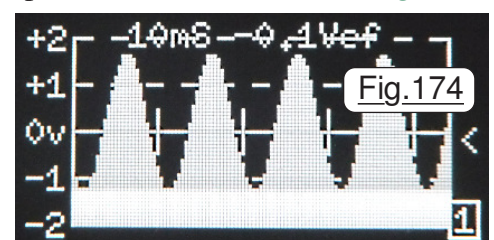
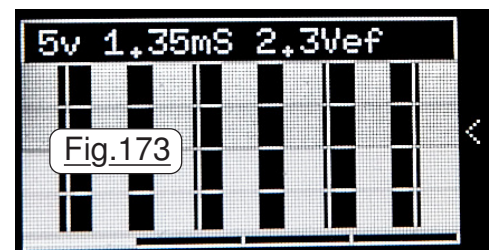
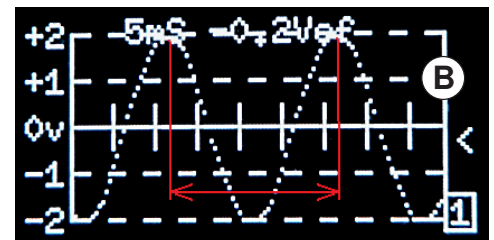
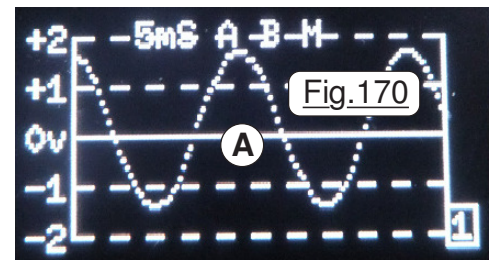
les points par des segments de droite, ce que fait la surcharge en rouge, le visuel est bien plus évocateur. C'est l'idée avantageuse qui a émergé lors de la modification logicielle pour ajouter les graduations verticales. Aussi une autre correction a été apportée au programme **P20** qui conduit aux représentations des exemples **B** et **C**. Ce tracé en vrai filaire est tellement plus efficace qu'il a remplacé par défaut le mode surface sur RESET, car les courbes sont parfaitement observables sans masquer le graticule. Naturellement ce nouveau mode filaire restera de mise

pour des enregistrements en mode "tensions non alternatives" comme sur l'exemple **D**.

Rien n'est parfait, et ce nouveau procédé de visualisation engendre un petit inconvénient. Si le signal est de type créniaux BINAIRES, les transitions "verticales" se faisant obligatoirement entre deux colonnes de PIXELs voisines, apparaît comme sur la Fig.172 en **A** ou en **B** un effet d'escalier. Du coup, la représentation en **mode surface** qui n'est pas affectée par cet inconvénient reste comme sur l'exemple de la Fig.173 une option conseillée **sur des signaux de nature BINAIRE**.

Que le signal soit de type BINAIRE ou alternatif comme sur la Fig.174 par exemple, dans tous les cas l'opérateur pourra choisir entre les deux modes visuels. En définitive, aucune option de représentation des traces

n'a été modifiée. **Ce n'est que le mode filaire qui a été amélioré en ne traçant plus des PIXELs, mais en joignant les points figuratifs par des segments de droite.**



La Base de temps par défaut qui était de **1.35mS** a été portée à **5mS** par graduation. Étant en enregistrement d'un signal, il suffisait de cliquer par inadvertance une seule fois sur BAS pour imposer 200mS par graduation. Du coup la capture des 119 échantillons exige 8S. Quand on constate l'erreur, il faut attendre huit secondes avec la touche HAUT ou BAS cliquée pour revenir à des durées de captures plus raisonnables. Si au bout de ces huit secondes le clic est pris en compte mais que l'on ne relâche pas la touche assez rapidement, c'est un **clic long** qui change le calibre mais pas la **BdT**. Et c'est reparti pour huit secondes ! Alors avec **5mS** de **BdT** il faut cliquer au moins trois fois sur BAS pour engendrer cette situation. Ce ne sera plus par inadvertance ...

25) Fin d'un périple ludique dans le domaine des oscilloscopes numériques.

Compte tenu de la modestie des performances du petit oscilloscope décrit dans ces lignes, il est évident que des facettes importantes de l'utilisation de ces outils ont été laissées de côté. Par exemple j'ai écarté l'idée d'une unité à deux entrées avec enregistrement simultané de deux signaux. La rapidité d'échantillonnage de notre appareil serait trop faible. Par ailleurs, le faible

nombre d'échantillons mémorisés rend illusoire la faisabilité d'ajouter un analyseur des séries de Fourier. Du reste, on peut se poser la question de la pertinence d'un tel chapitre concernant "la théorie du signal" dans une activité ludique. Une telle étude serait probablement hors propos. Avec les dernières modifications apportées au programme, on a réussi à dépasser les 80% d'occupation de la zone réservée au programme et la marge entre la **PILE** et le **TAS** reste plus que confortable. La "rentabilité" est au final tout à fait acceptable. Comme depuis plusieurs chapitres on ajoute du code "pour s'amuser" et explorer divers aspect logiciel, on constate à quel point avec un processeur tel que celui utilisé ici on peut élaborer des programmes vraiment très développés ... surtout si on ne dilapide pas les ressources de l'ATmega328 comme ça a été le cas durant notre cheminement de loisir.

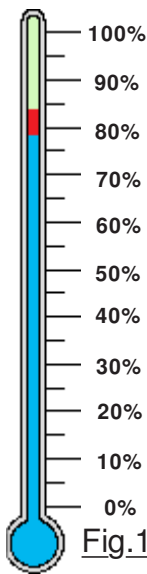


Fig.176

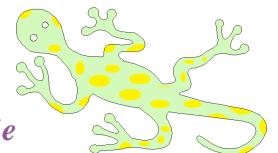
```
(PILE - TAS = 585)
Aff Version sur RESET
HAUT : OUI.
BAS : NON.
DROITE : Sortie.
```

Fig.175

➤ La mise en service et la prise en main.

Traditionnellement, je termine presque toujours mes didacticiels par un chapitre relatif aux "regrets" dans lequel je fais un bilan des points négatifs qui affectent le prototype. Un petit chapitre du genre "Si c'était à refaire ...". Et bien pour cette réalisation, comme le but n'est pas de posséder un oscilloscope performant, je n'ai vraiment rien à critiquer. Le but était de cheminer dans le monde de la programmation d'Arduino, et dans ce domaine on a balayé pas mal de facettes pas toujours explorées sur le nombre colossal de programmes qui sont mis en ligne. Mission accomplie. (Par exemple les menus par potentiomètre de la Fig.44 en page 25 n'ont pas été retenus.) Peu importe, il en restera forcément quelque chose. Aussi, on va pouvoir se quitter avec un moral au beau fixe. Toutefois, compte tenu du nombre de fonctions intégrées dans ce petit appareil, il me semble presque incontournable de vous proposer une "prise en main" où pas à pas je vous promène dans toutes les fonctions et les options. **Ce sera particulièrement utile pour celles et ceux qui séduits par l'idée vont vouloir réaliser ce "joujou" sans pour autant avoir le temps de lire l'intégralité du didacticiel.** Pour ne pas encombrer plus ce tutoriel, un petit livret **à imprimer recto/verso** accompagne l'ensemble des documents fournis. Ce fichier préservé dans le dossier <Documents> est nommé **Prise en main initiale.PDF** qui sera à imprimer et à assembler.

Chères lectrices, chers lecteurs, cette longue présentation arrive à son terme. Tout à une fin, mis à part l'Univers, et arrive forcément un moment où il faut raisonnablement considérer que "le voyage d'agrément" est terminé. Je souhaite intensément que certaines et certains oseront s'engager dans la réalisation d'un clone, je ne doute pas de leur réussite. Surtout, je vous souhaite à toutes et à tous de trouver dans ces lignes le plaisir de la découverte. Si d'aventure vous engagez vos heures de liberté dans une telle réalisation et que vous rencontriez une difficulté, vous pouvez me contacter sur : michel.droui@laposte.net et dans les limites de mon temps de libre, c'est avec grand plaisir que je tenterai de vous dépanner. Je vous souhaite à toutes et à tous agréable lecture et que l'envie d'en savoir plus sur les oscilloscopes puisse titiller votre sagacité ...



Chaleureusement : Nulentout.