C’est vrais que nous n’avons pas encore fait de tuto pour le custom port. C’est prévu, ce topic va en être l’ébauche.
Donc, pour une fonctionnalité personnalisée du portA ou B, on devra éditer le fichier *Custom_port.cpp*. Ne touchez à aucun autre fichier pour ne pas risquer de perturber le fonctionnement général du firmware.
Rappelons qu’il est nécessaire d’installer Vs-code sur votre ordinateur, puis le plugin platformIO dans VS-code.
La suite ne sera pas très différente de ce qu’on ferait avec un code arduino standard.

A suivre !!
Dis moi déjà si tout est clair pour toi.

Custom port :

Le custom port de l’arpschuino32 permet de personnaliser les fonctionnalités d’un ou des deux ports de l’arpschuino32 ou de la wilulu32. On peut, sans toucher à l’architecture globale du firmware, coder le comportement d’un port comme on le ferait pour un code Arduino simple.
Nous n’utiliserons par contre pas l’IDE Arduino qui ne conviens pas pour un code comme notre firmware qui contiens plusieurs langages (C++, HTML, Javascript, CSS).

Il faudra donc installer Vscode et platformIO. La bonne nouvelle, c’est que vous n’aurez ensuite pas à vous soucier d’installer des librairies ou autre, platformIO s’occupera de tout, automatiquement.

Commençons donc par Vscode. On télécharge la dernière version ici
https://code.visualstudio.com/Download
On choisi bien sur la version adapté à son système et on l’installe normalement, on peut laisser les paramètres par défaut.
Une foi Vscode installé, on l’ouvre et on clique sur extension.
Image
Dans la boite de recherche d’extension, on tape **platformIO IDE**. Une icône à tête de fourmis devrait apparaître, il ne reste plus qu’à cliquer sur **install** et attendre… C’est prêt !

Si vous ne l’avez pas encore fait, téléchargez la dernière version du firmware arpschuino32 ou Wilulu32.
https://arpschuino.fr/telechargements.php#codes
Une foi le dossier dézippé, si vous cliquez droit dessus vous devriez avoir l’option **ouvrir avec code**. Si ce n’est pas le cas, ce n’est pas grave, ouvrez Vscode, aller dans file/open folder (ou fichier/ouvrir un dossier si vous êtes en français). On va vous demander si vous faite confiance aux auteurs, c’est nous donc cliquez oui si vous nous faites confiance !

Bienvenue dans le code source d'arpschuino32 ! Comme vous pouvez le voir, il y a beaucoup de choses, on a pas chômé ! On va s'intéresser à un fichier en particulier : **custom port.cpp** (ne touchez pas au autres fichiers sauf si vous savez vraiment ce que vous faites!).

Pour cet exemple, nous allons travailler sur le port A.
Dans la section consacré au port A, on peut voir 3 fonctions :
- setupA : cette boucle est exécuté une seule fois au démarrage de la carte. C'est une boucle d'initialisation, équivalente au setup d'un programma Arduino.
- actionA : Cette boucle est appelé à chaque réception d'une trame DMX ou artnet.
-at_each_loopA : la boucle principale qui s'exécute en boucle comme la loop d'un programme Arduino.

Prenons un exemple simple : on veut les 4 premières sorties du portA en on/off et les 4 autres en PWM.

```
//////////// port A ///////////////
```
Mettre ici (avant le setup):
- Le #includes.
- Les bibliothèques (librarys).
- Les variables globales.
- les prototypes de fonction.

Dans notre exemple, ne mettons rien pour l'instant.

Setup :

```
void setupA()   //this loop is executed once at startup
{
 portA.setNbDmxChannels(8);
```
Il est dans tous les cas nécessaire de renseigner ici le nombre de circuits utilisés (ici 8).
Ensuite, on vas faire tous ce qu'on ferait dans la boucle setup() d'un code arduino standard. Dans notre exemple, on doit mettre le 8 broches en mode OUTPUT :
```
pinMode(Arp[0], OUTPUT);
```
Pour initialiser Arp0 en sortie. Ou :
```
 for(int i=0;i<8;i++){
  pinMode(Arp[i], OUTPUT);
 }
```
Pour initialiser les broches Arp0 à Arp7 en sortie.
```
}
```

Action :

Le plus simple consiste à aller voir dans le code source comment ça se passe en mode ON/OFF et PWM. Ça se trouve dans le fichier Port.cpp, dans la fonction servo_action. Les fichiers sont relativement volumineux, pensez à utiliser ctrl-F pour trouver facilement.
Voyons de quoi la fonction on_off_action se compose :
```
for (int i = 0; i < m_nbDmxChannels; ++i)
```

On entre dans une boucle for. Comme on veux qu'elle s'exécute 4 fois (pour les 4 sorties on/off), on va remplacer la variable *m_nbDmxChanels* par 4.
```
  for (int i = 0; i < 4; ++i)
  {
```

```
if (DMXslice[i] > m_trigger - 1)
```
La variable *m_trigger* représente la valeur DMX à laquelle la broche passera de off à on.
On remarque que le compilateur la souligne en rouge parce qu'elle n'est pas connue dans cette partie du code. On peux la remplacer par exemple par 127 ce qui représente 50 % DMX.

```
digitalWrite(Arp[m_startPin + i], 1 - m_inverted);
```
Comme on veut affecter les sorties Arp0 à Arp3, on va simplement mettre :
```
digitalWrite(Arp[i], HIGH);
```
Ce qui signifie : si le niveau DMX est > 50 % : met la broche au niveau haut !

Voici la fonction complète :

```
void action_A(const std::vector<uint8_t> & DMXslice)  //this loop is executed at each signal reception
{
  for (int i = 0; i < 4; ++i)
  {
    if (DMXslice[i] > 127)
    {
      digitalWrite(Arp[i], HIGH);
    }
    else
    {
      digitalWrite(Arp[i], LOW);
    }
  }

}
```

at_each-loop :

Il n'y à rien à faire dans cette loop pour le on/off. On la laisse vide et c'est terminé !
```
void at_each_loop_A()   //this loop is executed continuously
{
        Equivalent à la boucle loop() d'un code arduino standard.
}
```

Il va maintenant falloir faire la même chose pour le PWM...