Code Principale

> Documentation développeur : https://usini.github.io/m1d1_36/docs/index.html

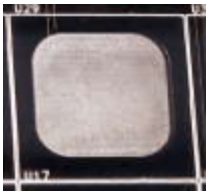
Afin de simplifier la modification du fonctionnement de l'instrument, tout le code est basé sur un [système événementielle]
Chaque fonctionnalité est rangé dans un fichier.h

- OLED.h (Afficheur)
- buttons.h (Boutons)
- midi.h (Midi)
- mpr121.h (Module capacitif)
- pots.h (Potentiomètre)
- settings.h (Paramètres)

Voici les 3 événements disponibles dans ces 3 fonctions

- MPREvent
- potsEvent
- buttonsEvent

MPREvent (PAD)



Lorsqu'un pad est appuyé ou relâché, cette fonction devient active.

Voici les paramètres de cette fonction:

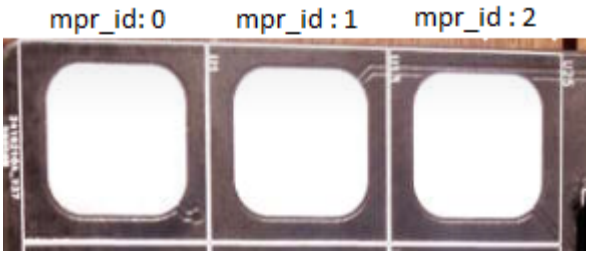
capPos : Position du l'ensemble des pads De 0 à 36 (noté sur le PCB comme U16, U17, U18 par ex)

note : La note correspondant à la position du pad comme noté dans settings.h (voir capNote[36])

```
uint8_t capNote[36] = {28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
                       40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
```

```
    52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63
}; ///< Note from pad (when transpose = 0)
```

state : **true** si appuyé et **false** si relaché
mpr_id : id du module capacitif, chaque ligne correspond à un module et commence par la ligne du bas



cap_id : id du pad par rapport au module (de 0 à 12)

Exemple, jouer une note du tableau capNote[36]

```
if (state) { //Pad appuyé
 noteOn(note, 120, 1) //Note midion (équivalent à capNote[capPos]), velocité, canal)
} else { // Pad Relaché
 noteOff(note, 0, 1) //Note midioff (équivalent à capNote[capPos]), velocité, canal)
}
```

potsEvent (Potentiomètre)

Lorsqu'un potentiomètre est tourné, cette fonction s'active

Voici les paramètres de cette fonction:
pot_id : Le numéro correspondant au potentiomètre **(noté A0 à A3 sur le PCB)**

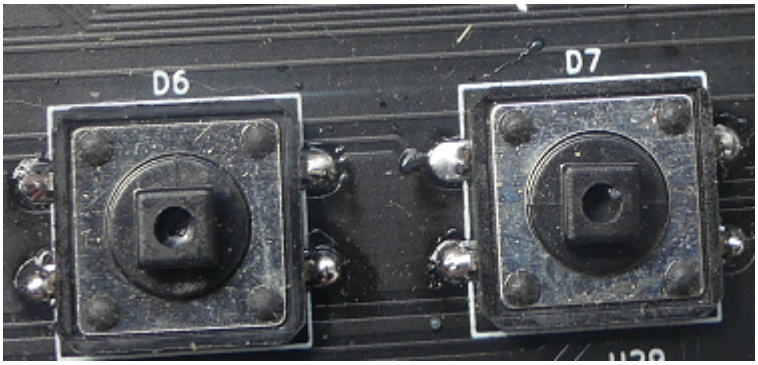
pots_cc[] : Le paramètre lié au potentiomètre **sous la forme de 0x01 à 0x7F (voir settings.h)**
pots_val[] : La valeur du potentiomètre de 0 à 127

> pots_cc et pots_val sont des variables globales

Exemple

```
switch (pot_id) {
    case 0:
      controlChange(0x01, pots_val[pot_id], 1);
      break;
    case 1:
      controlChange(0x02, pots_val[pot_id], 1);
      break;
    case 2:
      controlChange(0x03, pots_val[pot_id], 1);
      break;
    case 3:
      controlChange(0x04, pots_val[pot_id], 1);
      break;
}
```

buttonsEvent (Boutons)



Lorsqu'un bouton est **appuyé**, **relaché**, **appuyé deux fois**, **laissé appuyer** et **appuyé plusieurs fois**

Pour gérer les boutons, je suis passé par la bibliothèque AceButton qui permet de facilement pouvoir gérer plusieurs évènements.

Voici les paramètres de cette fonction:
button : Objet bouton, ici nous ne l'utilisons uniquement pour récupérez son identifiant
button->getid() : Identifiant du bouton
eventType : L'évènement qui a déclenché cette fonction
buttonState : État numérique du bouton (LOW/HIGH)
btn[] : État sauvegardé du bouton
btn_cc[] : Le paramètre lié au potentiomètre **sous la forme de 0x01 à 0x7F (voir settings.h)**

> btn[] et btn_cc sont des variables globales

Exemple

```
uint8_t id = button->getId();
switch (eventType) {
case AceButton::kEventPressed:
  btn[id] = !btn[id]; //Inversement de l'état du bouton
  if(btn_id){
    controlChange(btn_cc[id], 0, 1);
  } else {
    controlChange(btn_cc[id], 127,1);
  }
  break;
}
```

Revision #6
Created 2020-06-20 12:21:54 UTC
Updated 2021-02-24 09:09:12 UTC