Commande de ventilation

Encore un projet Arduino!

Yves Masur (11/2019) - ymasur@microclub.ch

Plan de l'exposé

- Définition du projet
  - Besoins
  - 1) Ambitieux: projet ETML-ES
  - 2) Réaliste, mais fonctionnel
- Composants hardware
- Outils logiciels utilisés
- Modules logiciels réalisés
  - Objets statiques vs dynamiques
  - Gestion des boutons et de l'écran
  - Sauvegarde en EEPROM
- Mise en service

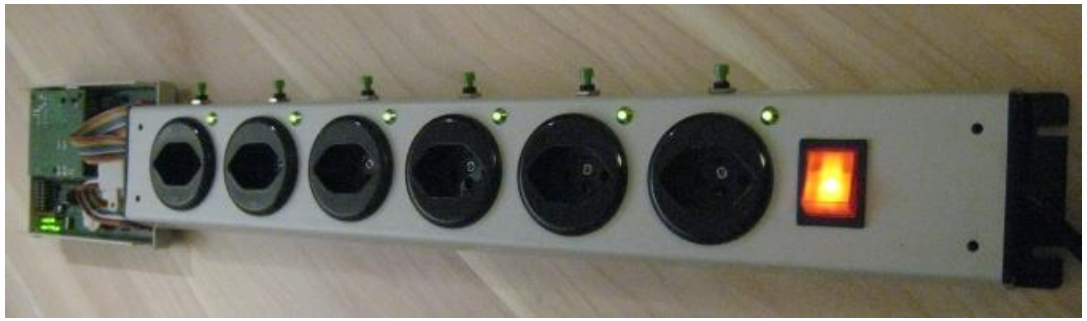
Commande de ventilateur
Besoin

- http://www.fanfarechardonnejongny.ch/
- 30 musiciens, dont 27 souffleurs
- Le local de répétition de la fanfare est un abri PC
- Aménagement: peinture et insonorisation (pour les *FF*)
- La ventilation est assurée par 2 moteurs (ou par manivelle…)
- Le bruit ne permet pas de joue finement (pour les *p*, et *pp*)
- Automatisation de la ventilation avant; après la répétition

Fanfare de
Chardonne - Jongny

Commande de ventilateur
Besoin

Ne tient pas le coup!
Monophasé!



La barrette! 2010 déjà…

1er projet – Ecole Supérieure

- Projet ETML-ES n° 1821
- CDC ambitieux
- Tient compte :
  - du niveau du son
  - de la température
  - de l'humidité
- Pas achevé – non fonctionnel

Réalisation

- Reprise et finalisation du projet ETML-ES ?
  - Sonde pic-kit 2 ?
  - Environnement de développement PIC: 19 Go… Update SSD!
  - Temps réel? Soft à ré-écrire – rentrer dans une architecture
  - Délai?
- Nouveau projet simplifié
  - Base connue: Arduino
  - Mesurer la durée effective de ce qui est demandé, **pour produit fini**
  - Environnement de développement connu, travail modèle et ré-utilisable
  - Délai: reprise en septembre 2019

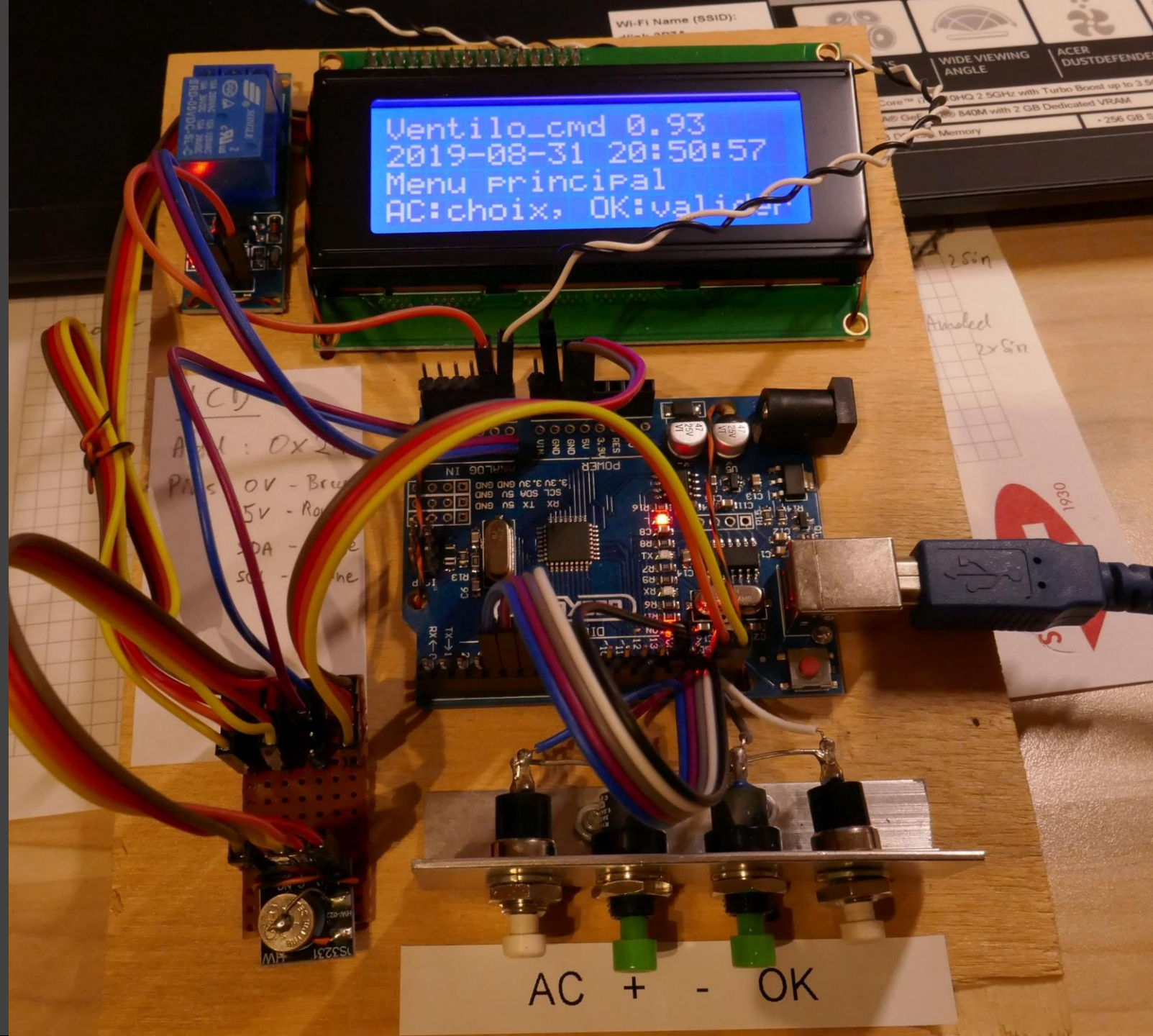
(re)définition du projet

- Cde ventilation à moteur triphasé
- Besoins en commutations
- Interface
  - Ecran
  - Boutons de réglage

Planche à clous

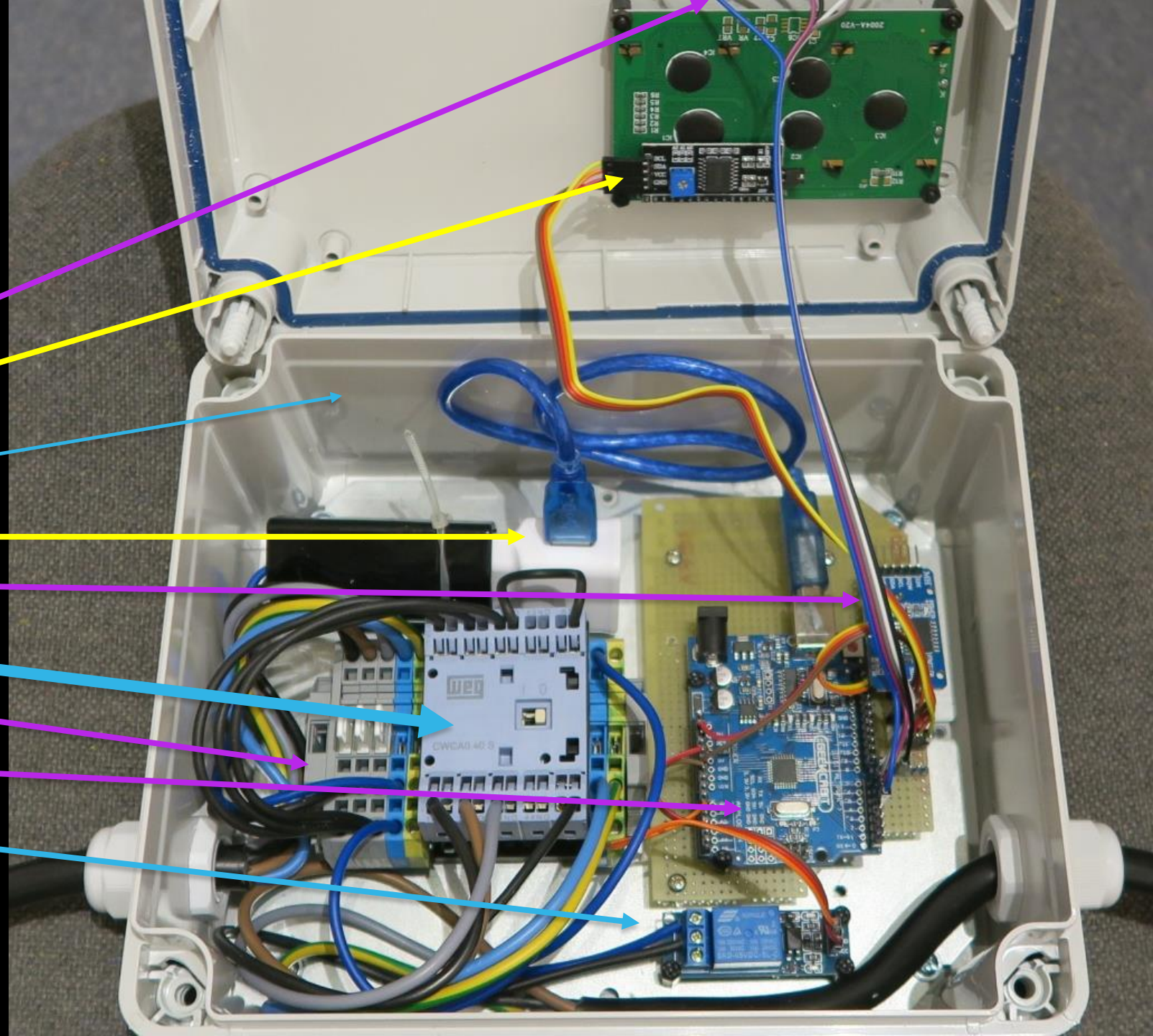
- Permet de tester les versions
- Même hardware connecté à Arduino
- (ou presque)

Schéma



Composants hardware

- 4 boutons <On>
- Ecran LCD I2C
- Boitier simple et solide
- Bloc alim 5V=
- RTC séparée I2C
- Contacteur de puissance
- Bornes sur rail DIN
- Arduino Uno
- Relais 5V / 230 VAC

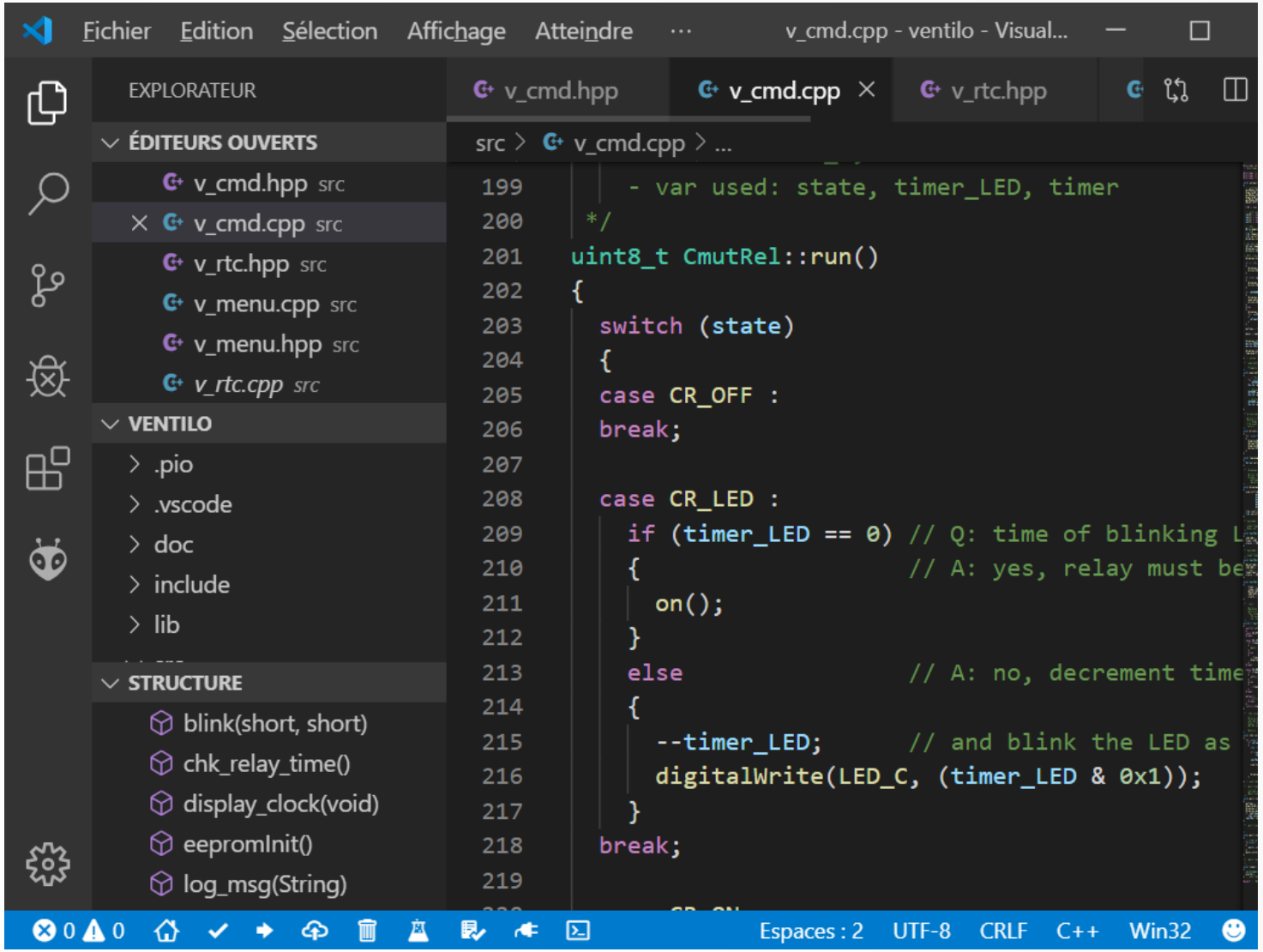


Composants logiciels

- Temps réel: librairie JM_SCHEDULER !
- Objets C++ clairs
- Startup étagé + diagnostics
- Management global
- Gestion des boutons et de l'affichage
- Gestion des tables de commutation
- Horloge RTC réglable

Méthode de travail

- Editeur utilisé: VSCODE
- Environnement Platform IO
- Répertoires “myCloud”
- Versionné sur GitHub
- Création d’objets :
  - Statiques
  - Dynamiques
- Structures & déclarations: fichiers .hpp
- Fonctions: fichiers .cpp



Visual Studio Code – structure de fichiers du projet «ventilo»

- **/doc**: exemples, documentation, listes
- /include: pas utilisé, les .hpp sont dans le source (/src)
- **/lib** : contient les librairies spécifiques, ex. jm_xxxxx
- **/src** : fichiers source (à backuper! À versionner!)
- /test : inutilisé, mais à considérer pour le développement objet
- **/vers** : copie des fichiers src, une fois le/les modules testés

- Publié : GitHub.com: https://github.com/ymasur/ventilo , mis «à la main»

Sortie relais, du hard à l'objet
Macros et états

```
#define REL A1
#define RELAY_ON() digitalWrite(REL, 0) //the relay is active LOW
#define RELAY_OFF() digitalWrite(REL, 1)
#define REL_ON_DURATION 60 // in minutes

uint8_t state; // little state machine, of commutation relay:
#define CR_OFF 0 // relay is Off
#define CR_LED 1 // LED_C blink
#define CR_ON 2 // relay is On
```

Dans l'objet, petite machine d'état: une LED clignote, puis le relais s'enclenche

Sortie relais, du hard à l'objet
Implémenter 'on'

```
/* CmuteRel::on()
--------------
Load timer, and set REL and LED_C to On state
*/
void CmutRel::on()
{
     RELAY_ON();
     LED_ON();
     //timer = REL_ON_DURATION * 60 * POLL_FREQ; // loaded for an hour
     loadTimer();
     state = CR_ON;
}
```

Petite machine d'état

Sortie relais, du hard à l'objet
Objet statique; setup

```
//Global inst. are defined here
CLASS CmutRel cmutRel;

pinMode(LED_C, OUTPUT);
digitalWrite(LED_C, LOW);
pinMode(REL, OUTPUT);
RELAY_OFF();
```

Aucune difficulté!

Ici, non plus!

Entrées, du hard à l'objet
Les switchs

```
#define SW_ACT 4 // Input hard definition
#define SW_PLUS 5
#define SW_MINUS 6
#define SW_OK 7
#define SW_NB 4 // nb of used switches
```

Entrées, du hard à l'objet
Switchs - Instanciation dynamique

```
// init an array of button control
sw[0] = new Sw(SW_ACT, "ACT");
sw[1] = new Sw(SW_PLUS, "[+]");
sw[2] = new Sw(SW_MINUS, "[-]");
sw[3] = new Sw(SW_OK, "OK");
```

Cool, pour afficher le nom

Référence hard

Mise en tableau

Entrées, du hard à l'objet
Définitions logiques!

```
#define B_ACT 0 // define logical definition
#define B_PLUS 1
#define B_MINUS 2
#define B_OK 3

if (sw[B_ACT]->getActivated()) // [AC] -> menu choice
{
    menu ++; // 0..3 menus
    smenu = 0;
    if (menu > 3) menu = 0;
}
```

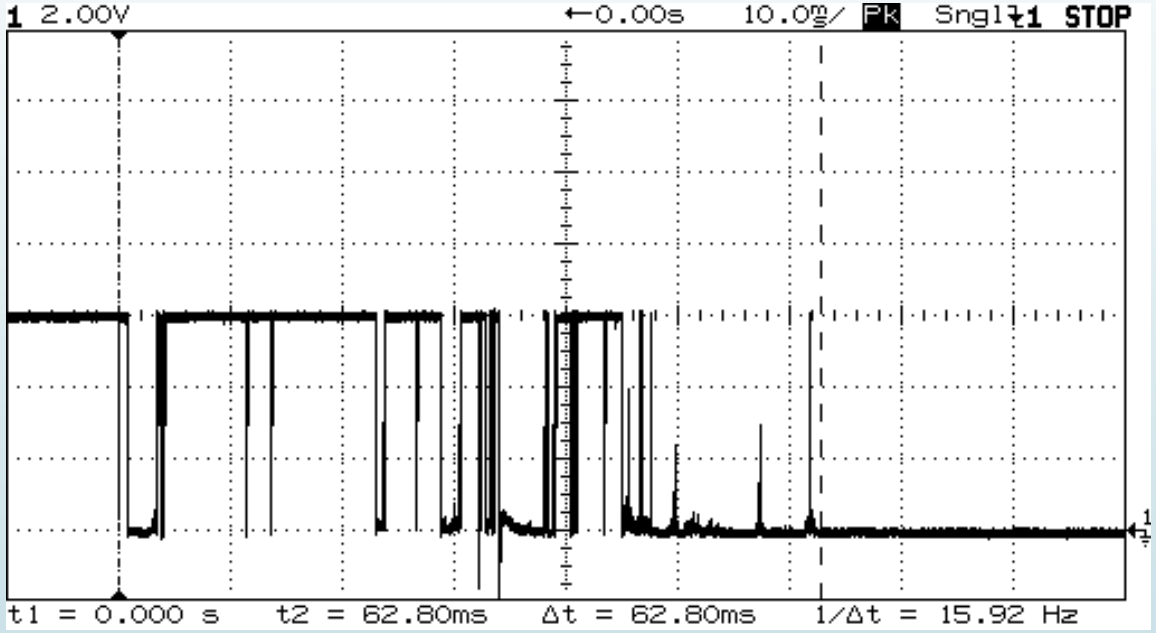
Référence soft, logique au tableau

Utilisation, fct membre

Entrées : les switchs
Le rebond de contact mécanique

- Dépend de
  - Poids de la matière
  - la qualité du contact
  - La pression mécanique
- Contacteur: 100 ms
- Relais “reed” : 2 ms

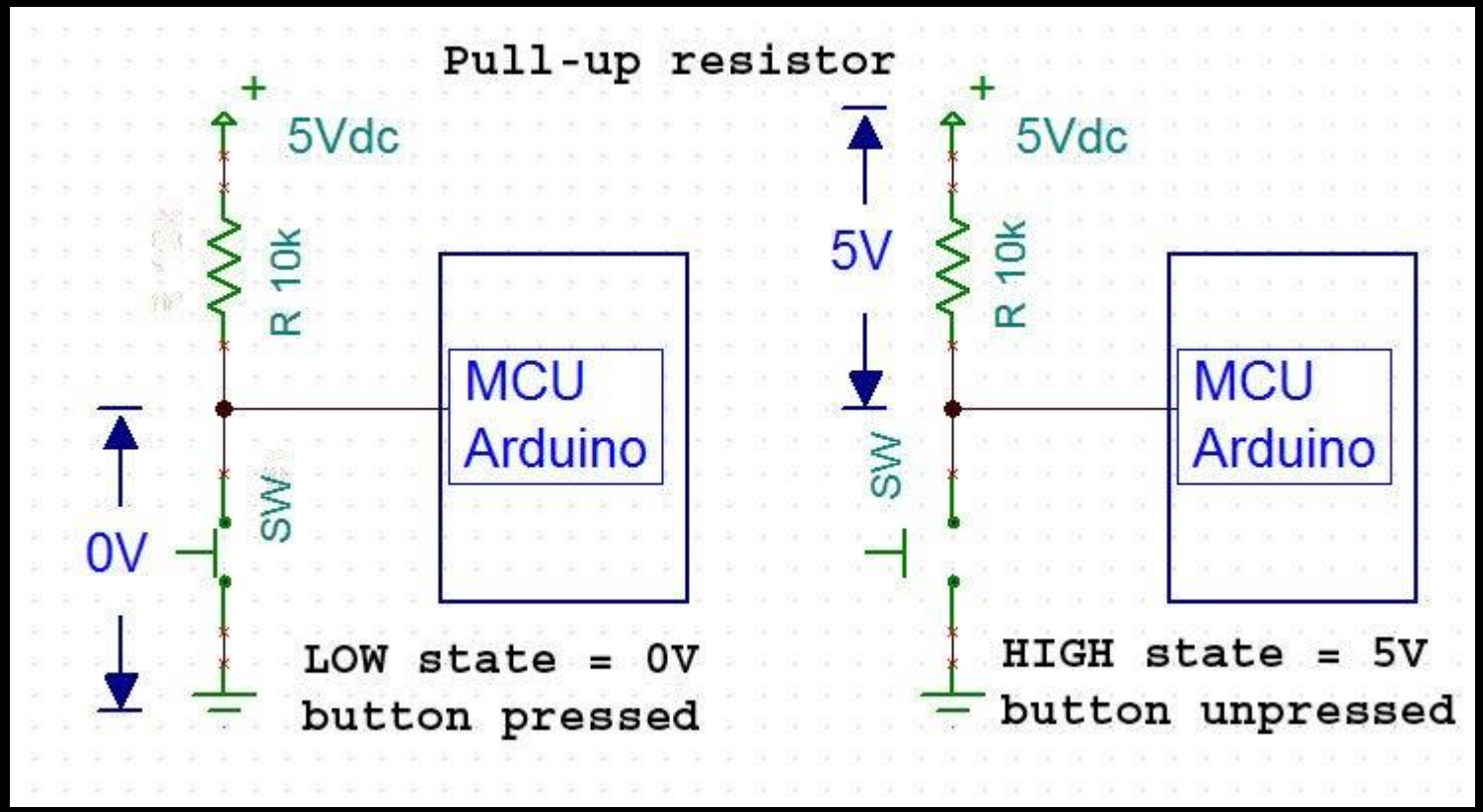
C’est inévitable!!



Recherche Google

Entrées : les switchs
La fonction scan()

- Appelée souvent pour une utilisation optimale
  - Trop lent: risque de rater une pression
  - Trop rapide, utilisation CPU et risque de doublon
  - 10 à 50 ms semble OK pour l'utilisateur
  - Appel routine, via jm_scheduler
- Comporte l'anti-rebond
- Maintient un timer
- Permet la répétition
- Logique vraie (OUI logique): 0 volt => pressé => 1 => true



Entrées : les switchs
La fonction scan()

```
void scan()
{
    if (timer<30000) timer++;
    bool in = !digitalRead(pin); // get the reversed value
    if (in != state) // Q: pin state changed?
    { // A: yes,
        state = in; // Store value & reset counter
        timer = 0;
    }
}
```

Entrées : les switchs
collection de fonctions ‘inline’

```
public:
inline bool getSt(){ return state; }
inline char getStChr(){ return (state ? '1':'0'); }
inline short getTm(){ return timer; }
inline String getName(){ return name; }
inline bool getPressed(){ if (state==true && timer>=1) return true; return false; }
inline bool getRepeted(){ if (state==true && timer%B_REP(4)==0) return true; return false; }
inline bool getChanged(){ if (timer<=1) return true; return false; }
inline bool getActivated(){ if (getChanged() && getPressed()) return true; return false; }
```

Répétition: expliqué plus loin

Entrées : les switchs focus sur la répétition…

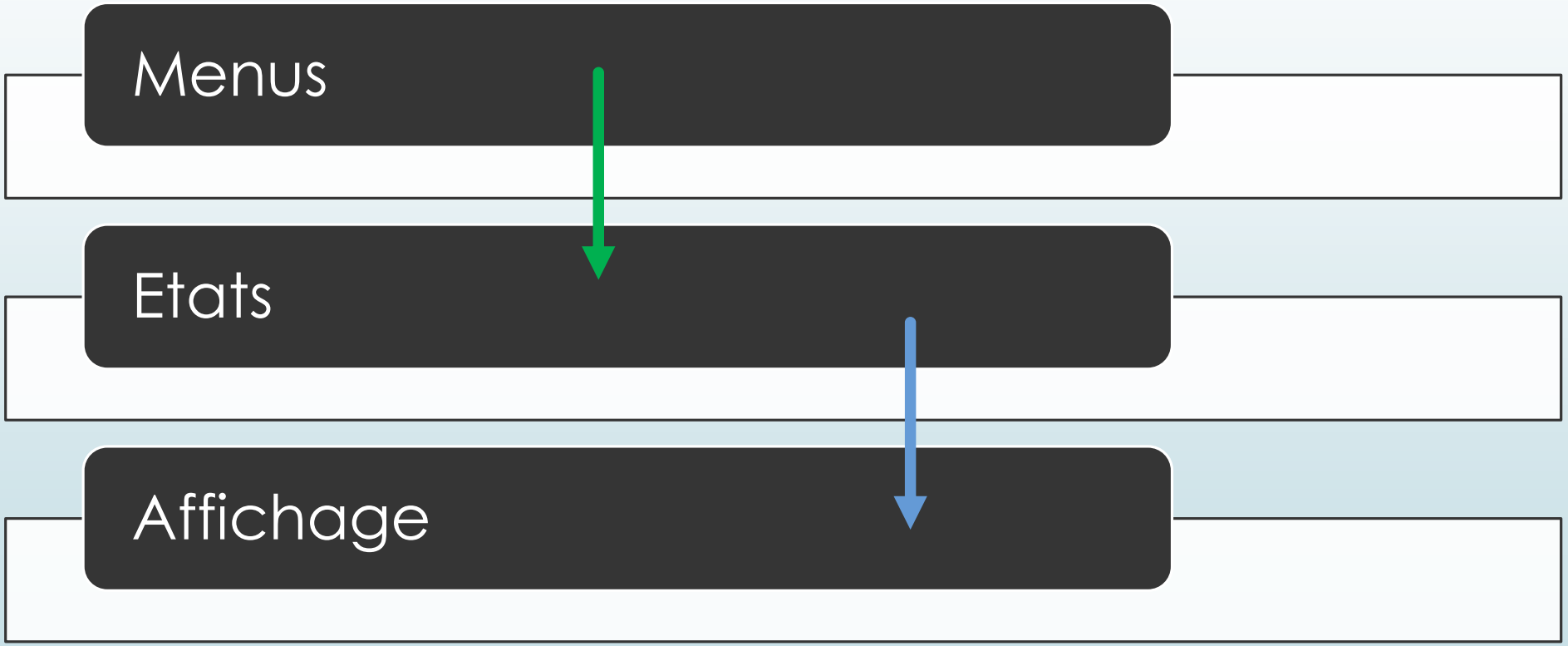
```
// compute the repetition delay for a button continuously pressed
// ex. 4x/sec: 1000/(10*4) = 25 ; gives -> 250 ms
#define B_REP(n) (1000/((B_SCAN_PERIOD) * n))

bool getRepeted()
{
    if (state==true && timer%B_REP(4)==0)
        return true;
    return false;
}
```

Selon ex., la période est 10 ms

Modulo du timer, 4 fois par seconde

Menus et affichage

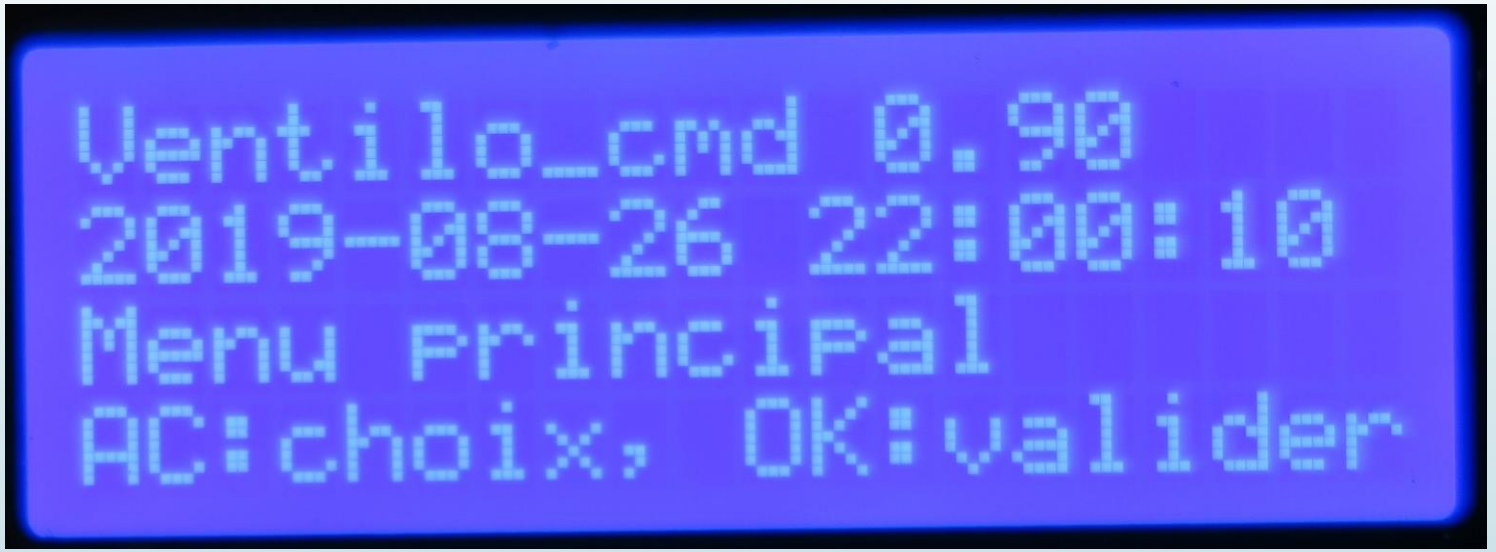


Menu et sélection

- Gestion en 2 parties indépendantes
  - Choix de menus, par boutons
    - Scan des boutons à 10..50 ms
  - Affichage, selon menu
    - Rafraîchi à 500 ms
- Menu, sous-menu, champs
  - Etat avec 2 variables globales, menu et smenu
  - Le champ est calculé dans smenu

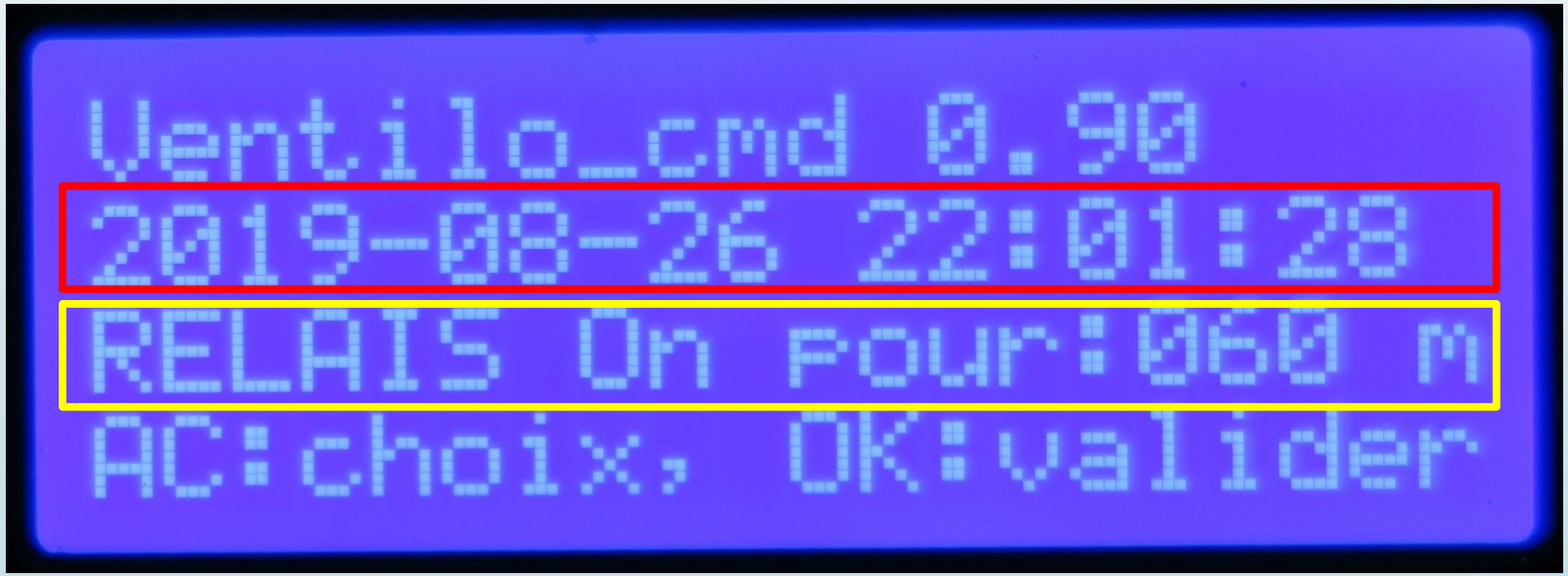
Menu principal

- Menu 0 (au repos)



Menu principal

- Menu 0, avec 2 lignes dynamiques
  - 2) Horloge
  - 3) Si le relais est actif (manuel ou table de commutation)



Traitement des menus

Ordre des menus, parcourus en boucle par [AC]:

0 - Menu principal

1 – Relais

2 – Commutations

3 – Horloge

- Et on boucle.
- Choix, par [OK]

Traitement des sous-menus

- Relais → un seul élément, choix par + ou –
  - Facile: soit 'on', soit 'off'
- Commutation → 4 éléments
  - La table: n°, heure+minute, et jour(s) de semaine
  - Modification directe et contrôlée des éléments
  - **Pointeur d'affichage** nécessaire pour visibilité à l'utilisateur
- Horloge → 5 éléments
  - Année, mois, jour, heure, minute – calcul du jour de sem., est automatique
  - Il faut une **copie** modifiable de la structure **DateTime**
  - Il faut calculer la position du **pointeur d'affichage**…

Traitement des sous-menus Exemple de l'horloge

- Le n° sous-menu est le **champ modifié**, à chaque [OK]
  - 0 : début (copie de la DateTime, pour proposer la date actuelle)
  - 1: année
  - 2: mois
  - … etc …
  - 6 : confirmation
  - 7 : enregistrement et fin de séquence

Traitement des sous-menus: Pointeur '*' du réglage de l'horloge

Le sous-menu indique le champ modifié

```
lcd->set_cursor(4+(smenu*3), 3);
lcd->print("*");
```

Astuce:
Tous les champs utilisent 3 chrs
Et sont à la ligne 3 (curseur à la 4)

Traitement des sous-menus: Répétition, ou non

- Pour un choix sans répétition, seule une pulse est prise en compte:

```
if (sw[B_PLUS]->getActivated()) smenu++;
```

- Pour incrémenter rapidement, le maintient fait la répétition:

```
if (sw[B_PLUS]->getRepeted()) hh++;
if (sw[B_MINUS]->getRepeted()) hh--;
```

Limiteur pour variable
Macro BOUND

```
#define BOUND(val, min, max)    \
{                               \
  if (val > max) val = max;     \
  if (val < min) val = min;     \
}
```

- ✓ Fonctionne pour entier, long, flottant…
- ✓ Bornes positives et négatives OK

Limitation: sélection de table

```
else if (smenu <10)        // Table selection (1..4)
    {
      if (sw[B_PLUS]->getActivated()) smenu++;
      if (sw[B_MINUS]->getActivated()) smenu--;
      BOUND(smenu, 1, NB_TABLES);
      commute_nb = smenu-1;
    }
```

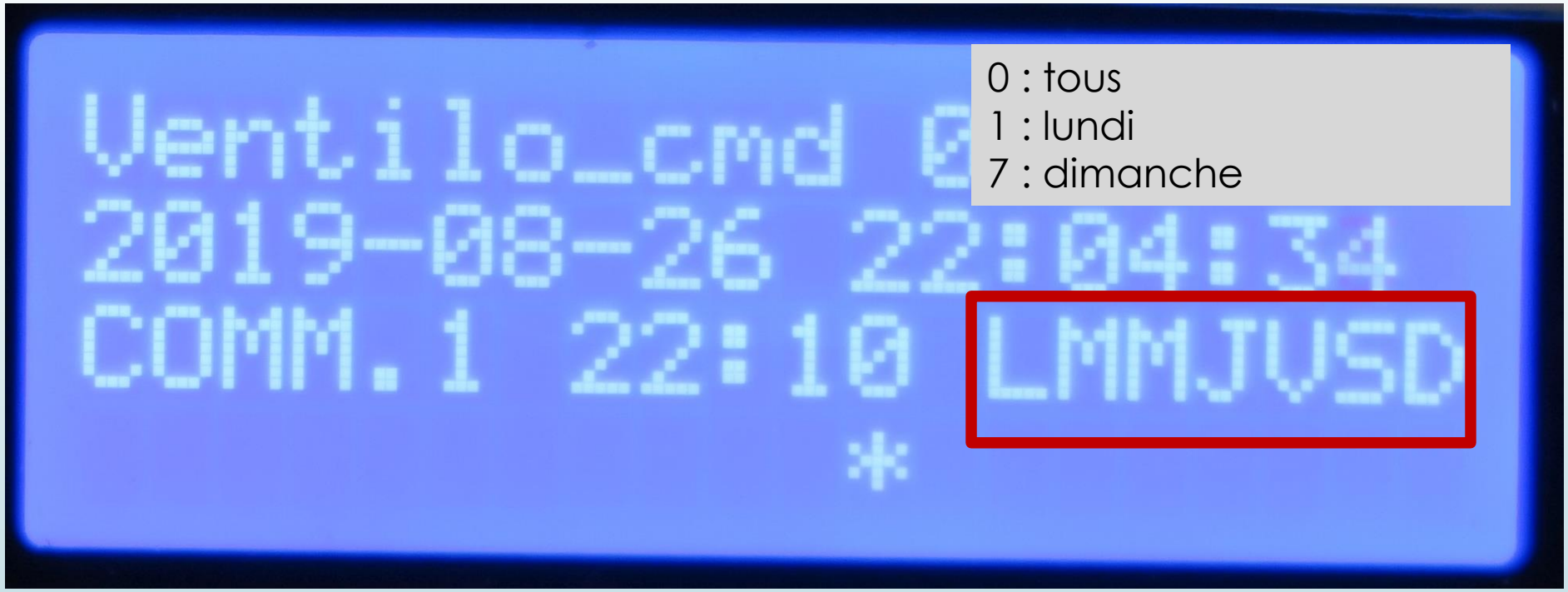
Pour mémoire, les tableaux C/CPP vont de 0 à n-1!

Table: sous-menu et pointeur

- Le sous-menu 1..9 (en fait:6) sélectionne la table
- La dizaine les champs:
  - 1X : heure
  - 2X : minute
  - 3X : le jour de semaine
  - 4X : confirmation de sauvegarde en EEPROM
- Les limitations sont incluses dans les fonctions de l'objet TimeCommute

Difficulté pour le display: le pointeur doit être calculé spécifiquement.

Table, affichage du pointeur
Spécial : jour de semaine



Données en EEPROM

Microcontroller	EEPROM
ATmega328 (Arduino Uno, Nano, Mini)	1024 bytes
ATmega168 (Arduino Nano)	512 bytes
ATmega2560 (Arduino Mega)	4096 bytes

- **The EEPROM finite life**
- The EEPROM has a finite life. In Arduino, the EEPROM is specified to handle 100 000 write/erase cycles for each position. However, reads are unlimited. This means you can read from the EEPROM as many times as you want without compromising its life expectancy.

Données en EEPROM

- Stockage de table: 3 valeurs
  - Heure
  - Minute
  - Jour de semaine

Facile: *toutes tiennent en 1 octet*!

- Pointeur de la bonne table
- Lire/écrire les 3 valeurs

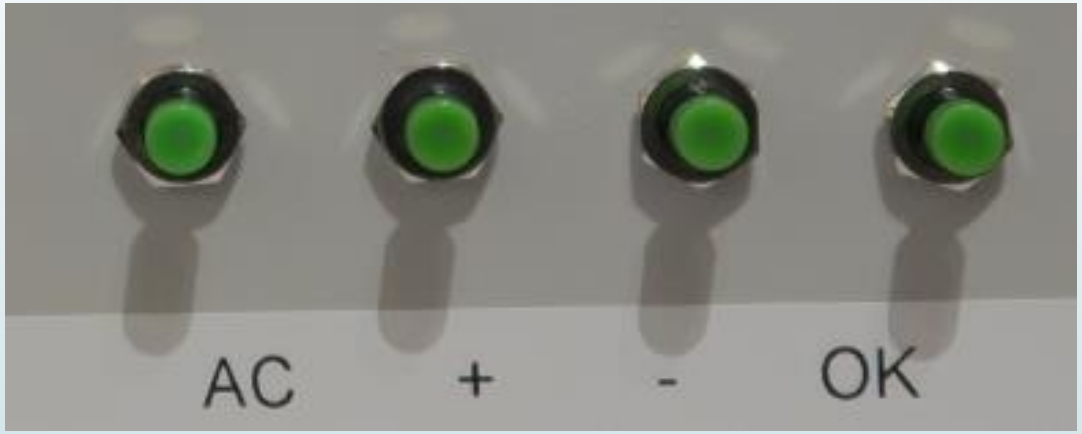
- Garde fou
  - octet 0, signature
  - octet 1: version de stockage

- La fonction objet doit connaître son ordre, pour écrire dans le bon slot
- Calcul du pointeur de la table n:

pos = n * 3 + EEPROM_DATA_OFFSET

Spécial : effacer l'EEPROM

- Maintient de [AC] et [OK] pendant 5 secondes



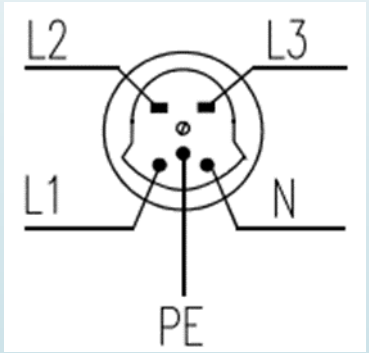
- Macro B_WAIT(x), temps indépendant du scan
- Gag : ne pas oublier les valeurs en mémoire…

Tests, mise en service

- Seule la prise de la machine à laver de la maison est triphasée…
- Sueur au premier enclenchement !
- Nécessité de filtrer la bobine du contacteur
- Commutations temporelles OK

Tests, mise en service

- Info projet:
  - 65 heures pour le codage et montage, documentation utilisateur
  - Coût du matériel: environ 150.-



Mise en service
5 septembre 2019

Améliorations

Faites

- Tables augmentées à 6
- Durée active redémarrable

A faire

- Durée réglable –OU-
- Composant dans la table de commutation

Merci de votre attention

Questions?