UNIVERSITE BADJI MOKHTAR ANNABA
FACULTE DE TECHNOLOGIE
DEPARTEMENT D'ELECTRONIQUE
MASTER 1 : AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

Annaba le 19/03/2023

MATIERE : **Systèmes Embarqués et Systèmes Temps Réels**

TP N°3 : Gestion des Entrées Digitales et étude du Phénomène de rebondissement des boutons poussoirs et gestion du clavier matriciel.

Objectifs : les objectifs principaux de ce TP sont :
La gestion des entrées digitales; configuration et lecture des boutons poussoirs des interrupteurs, la gestion des claviers matricielles et l'étude du phénomène du rebondissement des boutons poussoirs ainsi que les solutions anti-rebondissements SOFT et HARD.

Description de la carte EasyMx PRO v7 pour STM32 ARM

EasyMx PRO v7 pour STM32 ARM® est une carte de développement pour les périphériques STM32 ARM® Cortex ™ -M3 et Cortex ™ -M4, M7, M0. Il contient de nombreux modules intégrés nécessaires au développement de périphériques, notamment multimédia, Ethernet, USB, CAN et autres. Le programmateur et le débogueur mikroProg ™ intégrés prennent en

La carte est fournie avec le MC STM32F107VC d'une architecture ARM CORTEX M3

Elle comporte 5 ports GPIO PORTA….PORTE et plusieurs modules, la figure ci-dessous montre les différents bloques de ce MC…….

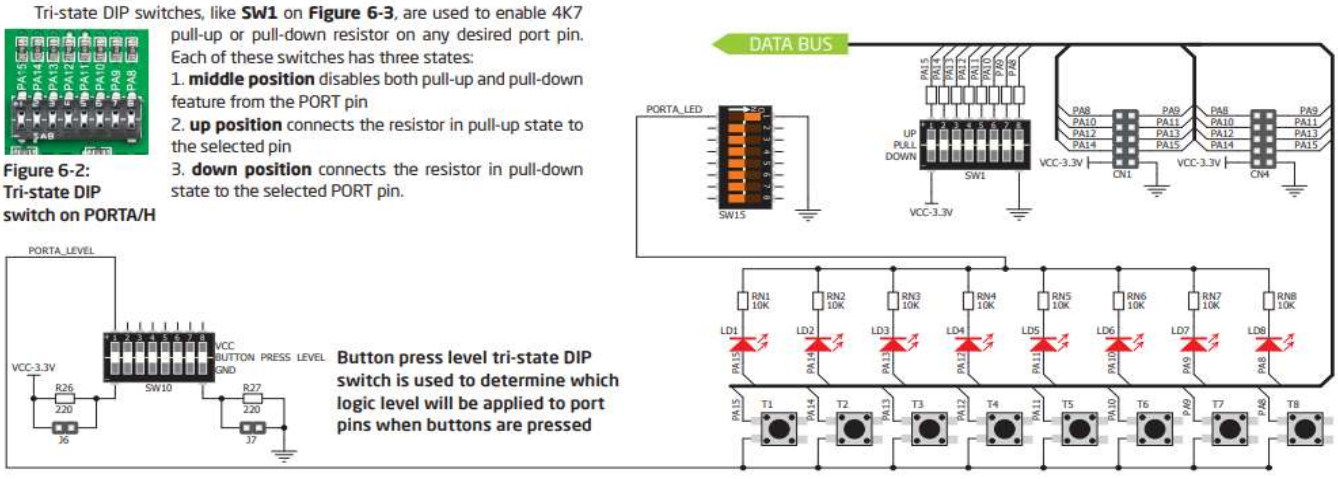
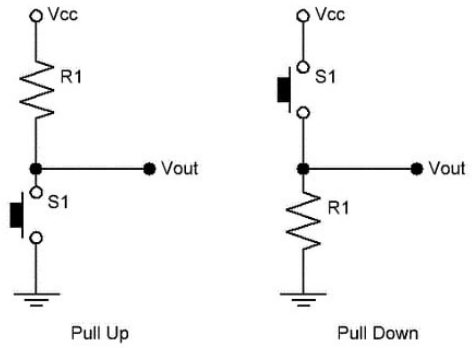


Figure 6-3: Schematic of the single I/O group connected to microcontroller PORTA/H

Le micro-switch SW10 permet de relier les résistances de rappelles au 0v ,3.3 v ou haute i mpédance pour chaque port.

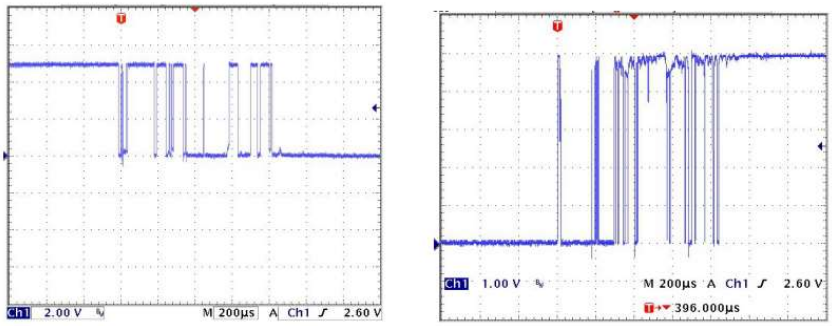
(Attention !! il faut suivre les instructions de configuration des différents micro-switches pour chaque exemple).

Pour la configuration en pull-up ou pull-down un seul jumper j6 ou j7 est positionné est non pas les deux sinon risque de court-circuit.



REBONDISSEMENT

Lorsqu'un interrupteur est basculé, un bouton-poussoir pressé ou relâché, on s'attend à ce que le signal bascule aussi d'état de façon instantanée. Dans les faits, les signaux visualisés à l'oscilloscope ci-dessous montrent qu'il n'en est rien



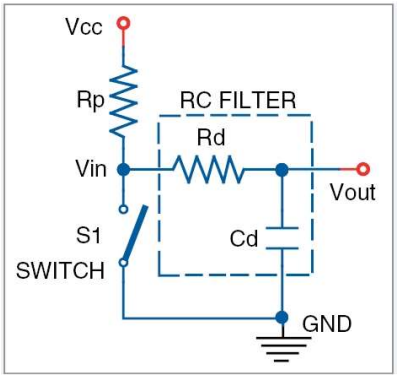
Fermer un circuit avec un interrupteur mécanique revient à mettre en contact des parties métalliques de l'interrupteur entre elles. Or, l'établissement du contact n'est pas instantané, et des rebonds qui se traduisent par des signaux parasités peuvent durer parfois quelques millisecondes avant la stabilisation du signal. Ce phénomène de rebonds s'observe souvent aussi bien à la fermeture qu'à l'ouverture du circuit.

Dans de nombreux cas, la durée des rebonds n'a aucune influence sur le fonctionnement du système. Mais quand un interrupteur est relié à une entrée d'un microcontrôleur suffisamment performant, ces rebonds peuvent être perçus comme autant de cycles de fermeture/ouverture très rapides et mettre en défaut le fonctionnement prévu.

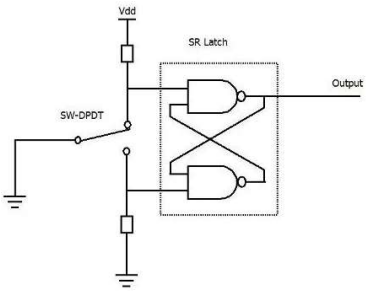
C'est pourquoi vous aurez parfois besoin de mettre au point un dispositif « antirebonds », géré par logiciel ou par du matériel spécifique.

Les solutions (hardware) est l'utilisation d'un filtre RC ou un circuit séquentiel à base de bascule de Schmitt ou un circuit séquentiel de comptage des front de rebondissement.

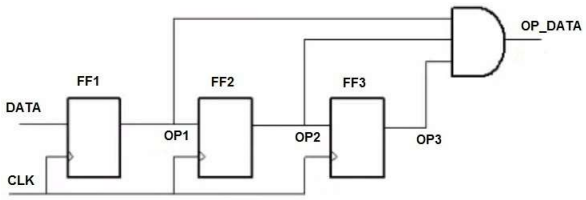
Hardware :



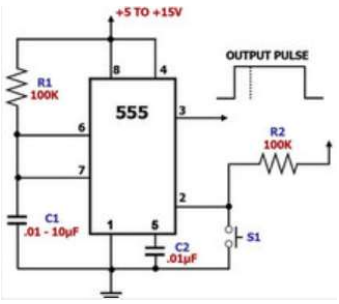
Filtre PB RC



Bascule RS



Pour des cartes électroniques doté d'une horloge basse fréquence.



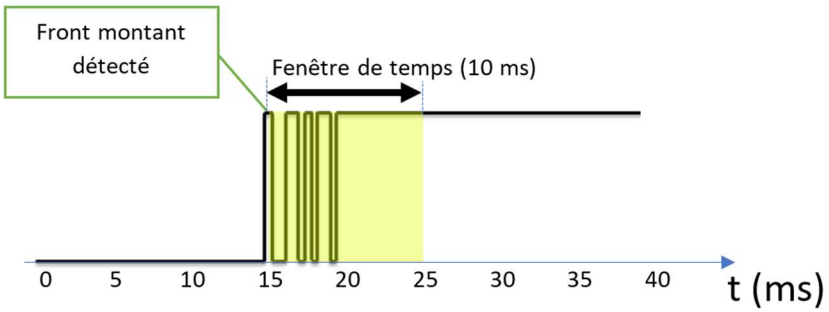
NE555 configuré en monostable

Software

Les solutions logicielles peuvent détecter les rebonds et les filtrer, ou plus simplement les ignorer.

Mise en œuvre d'une pause « antirebonds »

Avec des variantes, la stratégie antirebonds s'appuie principalement sur une fenêtre de temps après la détection d'un premier front où les rebonds doivent être ignorés.

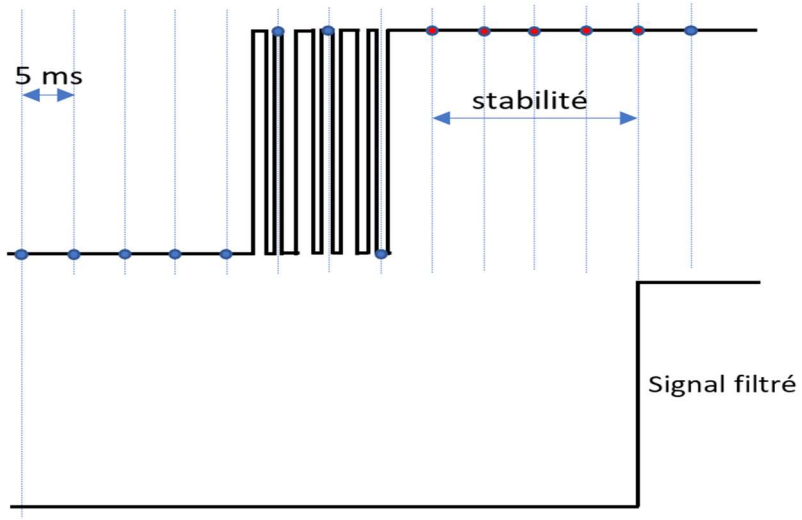


Le plus simple reste alors de mettre en pause la lecture du bouton pendant ce laps de temps, en espérant qu'après cette courte pause, la phase des rebonds soit terminée.

FILTRAGE

Un filtre sert en priorité à protéger votre système du « bruit » qui engendre de « fausses » impulsions sur un signal. On peut utiliser le principe des filtres pour supprimer les rebonds.

Un principe très simple pour filtrer le signal consiste à scruter l'état de l'interrupteur à intervalles réguliers, par exemple toutes les 5 ms. L'état est mis à jour seulement s'il est conservé à l'identique cinq fois de suite (fenêtre de stabilité = 20 ms) :

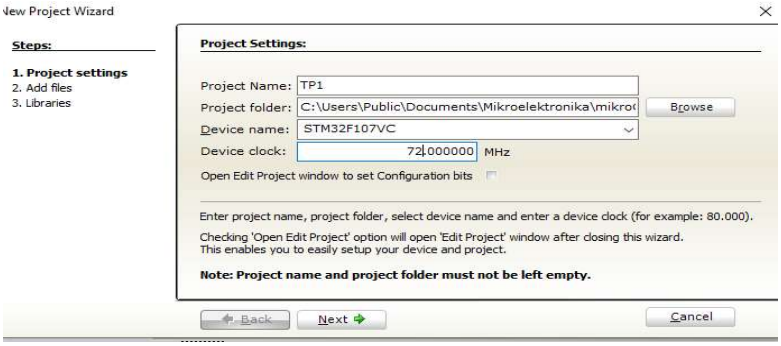


CREATION D'UN NOUVEAU PROJET

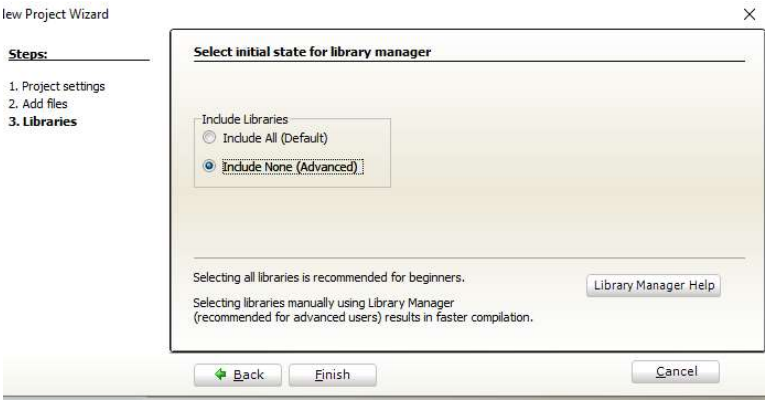
Le projet est défini comme à chaque appui sur le bouton poussoir relié à PA0 incrémente le PORTD par '1'

Utilisation du menu

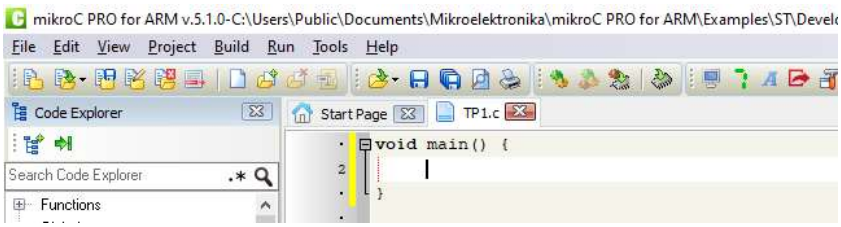
Configuration du processeur et la fréquence de fonctionnement.



Next pour n'ajouter aucun fichier au projet, puis ne cocher aucune bibliothèque à ajouter.



Finish le projet TP1 est créé et un fichier TP2.C vide



Taper le code suivant et sauvegarder TP2.c

```
unsigned int oldstate;

void main() {

  GPIO_Digital_Input(&GPIOA_IDR, _GPIO_PINMASK_0); // Set PA0 as  input

  GPIO_Digital_Output(&GPIOD_ODR, _GPIO_PINMASK_ALL); // Set PORTD as O

  oldstate = 0;

      GPIOD_ODR =0;

  do {

  /*  if (Button(&GPIOA_IDR, 0, 1, 1))     // detect logical one on PA0 pin

      oldstate = 1;

    if (oldstate && Button(&GPIOA_IDR, 0, 1, 0)       // detect one-to-zero

      GPIOD_ODR = ~GPIOD_ODR;              // invert PORTD value

      oldstate = 0;

   */

      if (GPIOA_IDRbits.IDR0)                        // detect logical one on
PA0 pin

      oldstate = 1;

    if (oldstate && (~GPIOA_IDRbits.IDR0))

{

GPIOD_ODR = GPIOD_ODR+1;

                  oldstate = 0;

                  delay_ms(50);

    }

  } while(1);

}
```

Avec EDIT > EDIT PROJECT > sélectionner avec load schem le fichier
TM32F107VC_PLL_25_to_72MHz.cfgsc

Vérifier que

- les LEDS reliées aux **PORT D sont 'ON' SW15 et**
- **SW10 PORT A 0v**
- Compiler pour corriger les erreurs
- Programmer le MC

1. Tester l'exemple 1

2. Pourquoi le fonctionnement du compteur n'évolué pas comme attendu.

3. Pour éliminer le problème de utiliser des temporisations après la détection du niveau 1 sur PA0
VALEURS à tester 1ms, 20ms, 200ms

4. Quelle la plus petite valeur de la temporisation qui corrige le fonctionnement du programme

Tester l'exemple 3 expliquer le programme

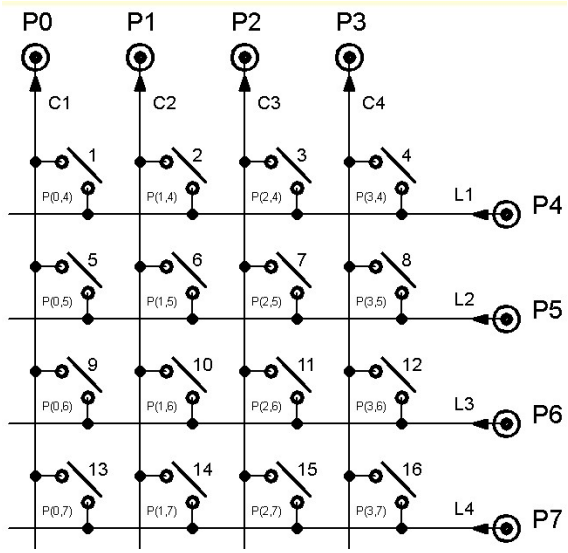
Tester l'exemple 3

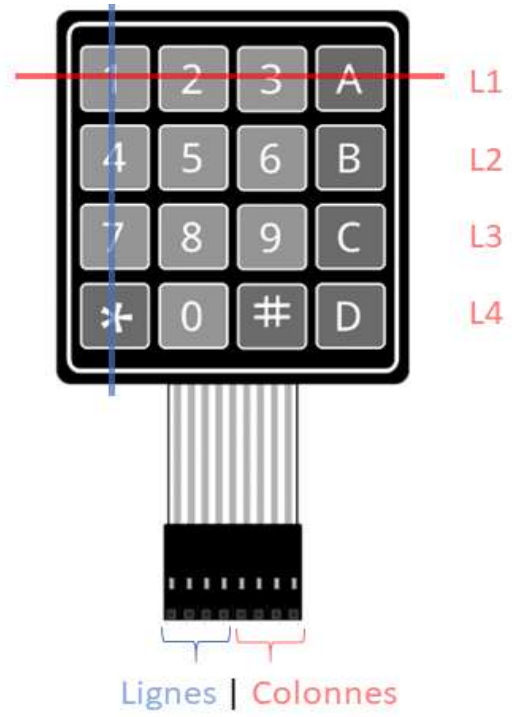
Tester l'exemple 4 et verifier le fonctionnement des résistances interne de rappelle .

6. Travail à rendre :

Le clavier est un élément fondamental d'un système de microcontrôleur, afin d'affecter les performances des fonctions à exécuter, de saisir des données et de pouvoir interagir avec la machine. Cela a une disposition matricielle des touches qui vous permet d'exploiter efficacement les broches des microcontrôleurs. Les 16 touches sont mappables en utilisant seulement 8 broches de données de microcontrôleur, la programmation est très rationalisée et simple à mettre en œuvre.

Ecrire un programme pour la gestion d'un clavier matriciel 16 touches





P0..P3 PORTA en entries

P4…p5 PORTB en sortie

L'appui sur une touche affiche le code binaire de la touche sur le PORTD.

NB : N'oublier pas d'insérer des temporisations pour gérer le problème de rebondissement des boutons.

(Attention !! il faut suivre les instructions de configuration des différents micro-switches pour chaque exemple).

```
Exemple 1 TP3
void main() {
  GPIO_Digital_Input(&GPIOA_IDR, _GPIO_PINMASK_0); // Set PA0 as  input
  GPIO_Digital_Output(&GPIOD_ODR, _GPIO_PINMASK_ALL); // Set PORTD as O
  oldstate = 0;
      GPIOD_ODR =0;
  do {

      if (~GPIOA_IDRbits.IDR0)           // detect zero on PA0 pin

{
        GPIOD_ODR = GPIOD_ODR+1;

                  delay_ms(2);
    }
  } while(1);
}
```

EXEMPLE2

```
unsigned int oldstate;
void main() {
  GPIO_Digital_Input(&GPIOA_IDR, _GPIO_PINMASK_0); // Set PA0 as  input
  GPIO_Digital_Output(&GPIOD_ODR, _GPIO_PINMASK_ALL); // Set PORTD as O
  oldstate = 0;
      GPIOD_ODR =0;
  do {

      if (~GPIOA_IDRbits.IDR0)                        // detect logical one
on PA0 pin
      oldstate = 1;
    if (oldstate && (GPIOA_IDRbits.IDR0))
{
GPIOD_ODR = GPIOD_ODR+1;
                oldstate = 0;
                //delay_ms(50);
    }
  } while(1);
}
```

Exemple3

```
unsigned int oldstate;
void main() {
  GPIO_Digital_Input(&GPIOA_IDR, _GPIO_PINMASK_0); // Set PA0 as  input
  GPIO_Digital_Output(&GPIOD_ODR, _GPIO_PINMASK_ALL); // Set PORTD as O
  oldstate = 0;
      GPIOD_ODR =0;
  do {
    if (Button(&GPIOA_IDR, 0, 0, 0))      // detect logical one on PA0 pin
      oldstate = 1;
    if (oldstate && Button(&GPIOA_IDR, 0, 0, 1) )       // detect one-to-
zero
     { GPIOD_ODR = GPIOD_ODR+1;              // invert PORTD value
      oldstate = 0;
      }

  } while(1);
}
```

```
Exemple 4
unsigned int oldstate;
void main() {
     GPIO_Digital_Input(&GPIOA_IDR, _GPIO_PINMASK_0); // Set PA0 as  input

       GPIOA_CRL=0x00000008  ;     //input pull up ou down
       GPIOA_ODR=0x00000001;         // pull up
       RCC_APB2ENR= 0x00000004  ;


  GPIO_Digital_Output(&GPIOD_ODR, _GPIO_PINMASK_ALL); // Set PORTD as O
  oldstate = 0;
      GPIOD_ODR =0;
```

```
  do {
    if (Button(&GPIOA_IDR, 0, 10, 0))      // detect logical one on PA0 pin
      oldstate = 1;
    if (oldstate && Button(&GPIOA_IDR, 0, 10, 1) )      // detect one-to-
zero
     { GPIOD_ODR = GPIOD_ODR+1;              // invert PORTD value
      oldstate = 0;
      }

  } while(1);
}
```