Lecture de clavier matriciel

D.Moreaux

21 octobre 2018

1 Principe de fonctionnement

Lorsque l'on veut lire un grand nombre de boutons, on peut rapidement arriver à court de broches d'entrée-sortie sur le microcontrôleur. Pour éviter cela, on disposera les boutons sous forme d'une matrice de m sur n boutons.

On pourra ainsi lire 30 boutons avec 11 broches (5+6) au lieu de 30 [1].

Une matrice 3x4 aura la structure reprise à la figure 1. On voit qu'ici on dispose de 12 boutons arrangés en 4 lignes de 3 colonnes.

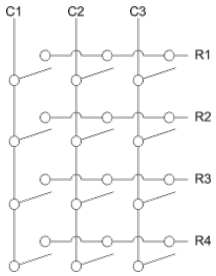

FIGURE 1 – Clavier matriciel

Pour lire ce clavier, il faudra activer une ligne (colonne) à la fois et à chaque fois, lire les colonnes (lignes) pour déterminer les boutons poussés. Les autres lignes (colonnes) seront désactivées.

Pour lire les boutons d'une ligne, on utilisera les entrées correspondant aux colonnes en `INPUT_PULLUP` et on travaillera comme s'il s'agissait de boutons isolés (gestion du rebond, transition de coupé à activé).

La ligne active devra donc être au niveau bas (OUTPUT, signal LOW). On pourrait être tenté de laisser les autres lignes au niveau haut (OUTPUT, signal HIGH) mais si plusieurs boutons sont enfoncés, cela pourrait résulter

1. A noter que l'Arduino ne dispose que de 20 broches utilisables en entrée-sortie ce qui serait insuffisant pour associer une broche par bouton

à un court circuit entre deux sorties [2]. On basculera donc les lignes inactives en INPUT sans pull-up.

En enfoncant plusieurs boutons, des boutons "fantômes" peuvent être détectés, le courant passant d'une ligne à l'autre par deux boutons enfoncés sur la même colonne. Il est possible d'éviter cela en plaçant des diodes en série sur les boutons mais généralement, on se contente d'ignorer le problème pour des raisons de coût.

2 Lecture du clavier

Plutôt que de faire la lecture du clavier à la main, on utilisera généralement la bibliothèque `Keypad` qui se chargera du scan du clavier, de l'anti-rebond, de la détection d'appui et du décodage du clavier.

On commencera par définir trois tableaux : les broches utilisées pour les lignes, les broches utilisées pour les colonnes et les valeurs des touches.

On crée ensuite un objet `Keypad` qui reprendra ces informations ainsi que les nombres de lignes et de colonnes. Cet objet se chargera lui-même de configurer les broches en entrée et en sortie selon les besoins.

Il suffira alors d'utiliser les fonctions de l'objet `Keypad` pour lire le clavier.

```
#include <Keypad.h>

#define LIGNE 4
#define COL 3
char keys[LIGNE][COL] = {
{'1','2','3'},
{'4','5','6'},
{'7','8','9'},
{'*','0','#'}
};
byte rowPins[LIGNE] = {2, 3, 4, 5};
byte colPins[COL] = {8, 7, 6};

Keypad kpad = Keypad( makeKeymap(keys), rowPins, colPins, LIGNE, COL );
```

2. par exemple si R1C1 et R2C1 sont enfoncés, si R1 est LOW et R2 est HIGH, on a un court-circuit

k = *kpad*.getKey()	Retourne dans k le caractère correspondant à la touche enfoncée. Retourne 0 si aucune touche n'est enfoncée
kpad.getKeys()	rempli le tableau *kpad*.key[] avec l'état des touches. Si au moins une touche est active, retourne *true*
kpad.setHoldTime(n)	Défini le temps pour qu'une touche passe de PRESSED à HOLD (en ms)
kpad.setDebounceTime(n)	Défini le temps pour l'anti-rebond

key[i].stateChanged	*true* pour les touches dont l'état a changé depuis le dernier getKeys
key[i].kstate	PRESSED, HOLD, RELEASED, IDLE selon l'état de la touche
key[i].kchar	lettre associée à la touche
`LIST_MAX`	Nombre de touches à scanner dans le tableau key[]

Si on est juste intéressé à lire une touche à la fois, on utilisera getKey(). Si par contre on désire être capable de détecter le fait que plusieurs touches soient enfoncées, on utilisera getKeys() puis on parcourra le tableau key[] pour connaître l'état des touches individuelles.

Le tableau key[] ne reprend pas toutes les touches du clavier mais un maximum de `LIST_MAX` (normalement 10) touches.

Les lignes sont remises en `INPUT_PULLUP` à chaque appel à getKey ou getKeys et les colonnes doivent rester (ou pêtre remis en) des entrées d'un appel au suivant.