

ELE3312  
Microcontrôleurs et applications  
Laboratoire 4

Auteur : Jean Pierre David

## Introduction

Outre le port série, déjà utilisé pour le débogage dans un laboratoire précédent, le clavier et l'écran LCD sont deux périphériques très souvent employés dans les systèmes à microcontrôleur. Le clavier est en effet le moyen privilégié pour recevoir de l'information de l'utilisateur alors que l'écran est le dispositif privilégié pour lui en fournir. Les premiers écrans LCD (Liquid Crystal Display) étaient assez rudimentaires et ne permettaient d'afficher que des caractères constitués de segments préfabriqués. Aujourd'hui, on utilise de plus en plus des écrans constitués de pixels, ces petits points en couleur ou monochromes qui, ensemble, peuvent reconstituer à peu près n'importe quel effet visuel. Dans ce laboratoire, vous aurez l'occasion d'utiliser un écran couleur TFT LCD de 320x240 pixels avec des couleurs sur 16 bits (65536 couleurs).

## Objectifs

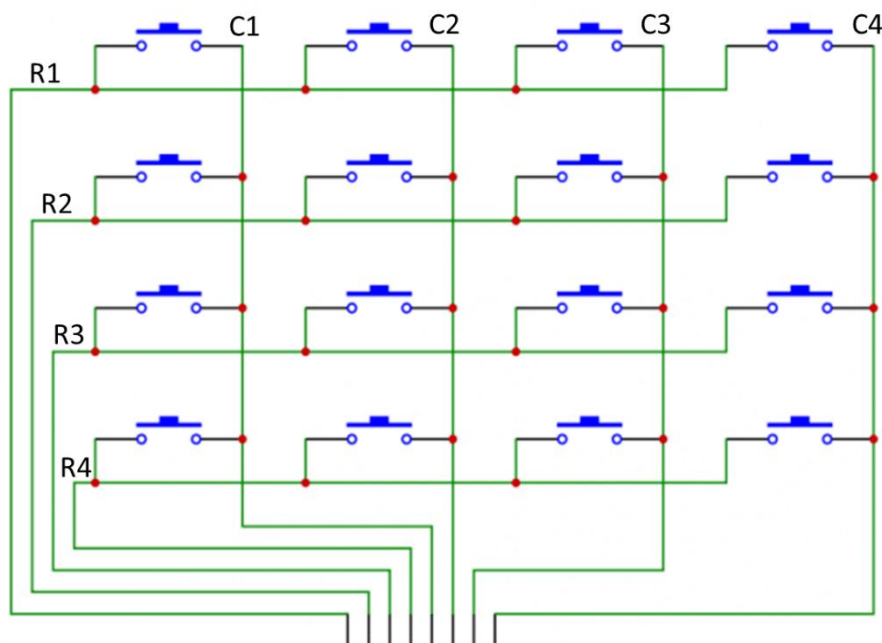
1. Comprendre le principe de fonctionnement d'un clavier de type matriciel et son interfaçage avec un microcontrôleur
2. Comprendre le principe de fonctionnement d'un écran TFT LCD et son interfaçage avec un microcontrôleur
3. Comprendre le principe de fonctionnement et l'utilisation du SysTick Timer
4. Réaliser une application complète dotée d'un clavier et d'un écran

## 1<sup>ère</sup> partie : préparation à la maison

### Mode opératoire

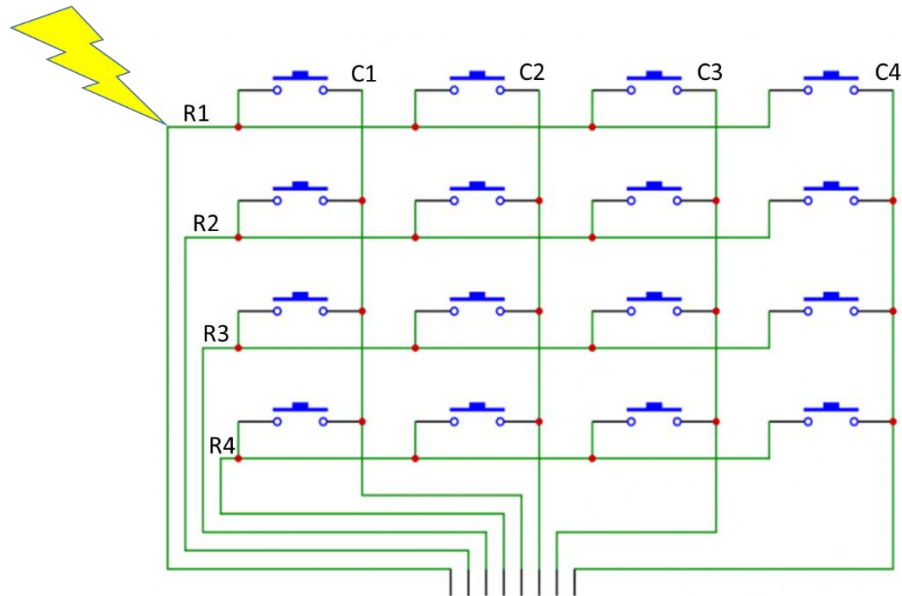
#### Fonctionnement d'un clavier matriciel

Soit le clavier suivant constitué de 16 interrupteurs organisés en 4 rangées (R1 à R4) de 4 colonnes (C1 à C4) :



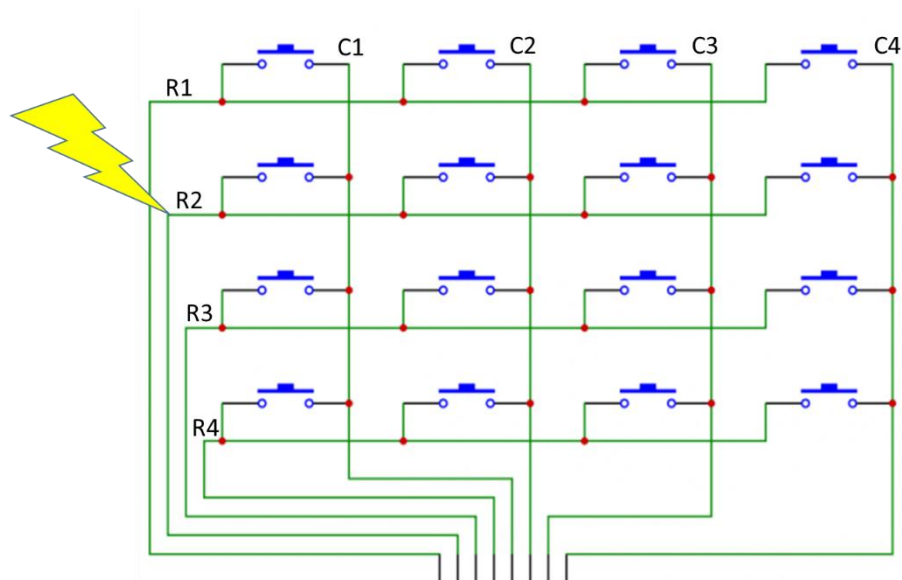
Lorsqu'on pèse sur un interrupteur, un court-circuit se produit entre la ligne et la colonne concernées. Si on connecte les 4 lignes et les 4 colonnes au microcontrôleur, la question devient : « Comment programmer le microcontrôleur pour qu'il détecte un court-circuit entre une ligne et une colonne ? ». Le principe utilisé est toujours à peu près le même :

Dans un premier temps, on met du courant sur une rangée, par exemple la rangée R1 :



Ensuite, si l'utilisateur appuie sur un des interrupteurs de la première rangée (R1), le courant sera transmis à la colonne correspondante. En mesurant s'il y a du courant sur chacune des colonnes, on peut maintenant savoir si un ou plusieurs interrupteurs de la première rangée est/sont pesé(s).

Pour mesurer l'état des interrupteurs de la deuxième rangée (R2), on met du courant sur la deuxième rangée :

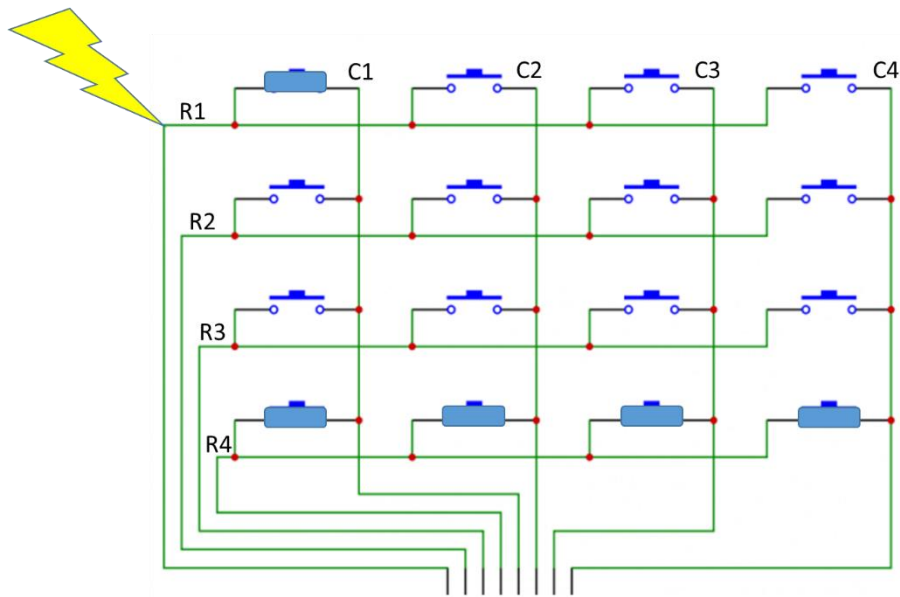


En mesurant s'il y a du courant sur chacune des colonnes, on peut maintenant savoir si un ou plusieurs interrupteurs de la deuxième rangée est/sont pesé(s).

En passant alternativement d'une rangée à la suivante, on finit par savoir quels interrupteurs sont pesés.

Dans les faits, les choses sont un peu plus complexes pour deux raisons :

- 1) Le microcontrôleur ne peut pas envoyer « du courant » sur une de ses lignes. Il peut la mettre à « 1 » (3.3V), à « 0 » (0V) ou encore la laisser flottante (haute impédance). Par ailleurs, le microcontrôleur ne peut pas lire s'il y a du courant ou pas sur une entrée, il peut juste mesurer si la tension d'entrée est 0V ou 3.3V
- 2) Lorsqu'on pèse sur plusieurs interrupteurs, il peut se produire des court-circuits qui peuvent prêter à confusion, par exemple :

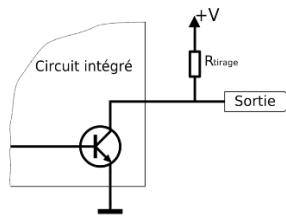


Dans ce cas de figure, le premier interrupteur de la rangée R1 est pesé et tous les interrupteurs de la rangée 4 sont pesés également. Lorsqu'on met du courant sur la rangée R1, on constate que ce courant se transmet à la colonne C1, ensuite il se transmet à la rangée R4 et finalement il arrive sur toutes les colonnes. Comme on mesure du courant sur toutes les colonnes, on pourrait faussement en déduire que tous les interrupteurs de la première rangée sont pesés alors qu'en fait, il n'y en a qu'un mais toutes les colonnes sont court-circuitées à cause de la rangée R4.

Pour parer au point 1, on va utiliser deux stratagèmes :

- a) Sur chaque colonne, on va ajouter une résistance de pull-up (en série avec l'alimentation 3.3V). De cette manière, en temps normal (lorsqu'aucun interrupteur n'est pesé), le microcontrôleur mesure un « 1 » logique sur chacune des colonnes.
- b) Plutôt que d'envoyer du « courant », on va créer un court-circuit entre la rangée qu'on veut tester et la masse. De cette manière, si un interrupteur est pesé, la colonne correspondante sera connectée à la masse également et on mesurera un « 0 » logique à la place d'un « 1 ». Les rangées qui ne sont pas testées doivent rester « déconnectées », c'est-à-dire en haute

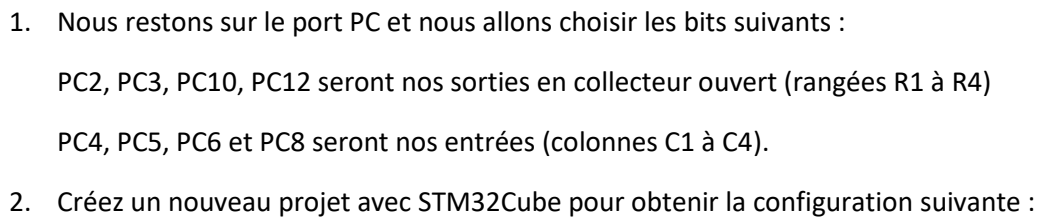
impédance. Ce mode de fonctionnement est exactement ce qu'on appelle une sortie à collecteur ouvert (open collector ou encore open drain) :



Lorsqu'on active le transistor, la sortie est connectée à la masse, sinon elle n'est connectée à rien et la résistance de pull-up la force à l'état « 1 ». Il faut remarquer que pour avoir un court-circuit avec la masse (un « 0 » logique), le microcontrôleur doit bien mettre un « 0 » dans le bit de configuration de la sortie (même si cela correspond à un état haut sur la base du transistor). Lorsqu'il met un « 1 » dans le bit de configuration (ce qui correspond à un état bas sur la base du transistor), en fait il laisse sa sortie en haute impédance. Il faut encore remarquer que si plusieurs sorties sont connectées au même signal, en temps normal, on peut avoir des conflits. Avec le mode « collecteur ouvert », il n'y a jamais de conflit. Dès qu'au moins une sortie impose un « 0 », le signal vaut « 0 ». Si toutes les sorties sont à « 1 » (haute impédance), la résistance de pull-up impose une sortie « 1 » également.

Pour parer au point 2, on peut utiliser un clavier avec une diode en série avec chaque interrupteur. Toutefois, les claviers que vous avez au laboratoire n'en sont pas équipés donc il ne vous sera pas possible de faire la différence entre toutes les configurations possibles de touches pesées.

Nous avons besoin de 4 sorties à collecteur ouvert et 4 entrées. Toutefois, comme nous devons aussi connecter l'écran LCD et que celui-ci utilise toutes les broches de type Arduino (celles en rose dans l'image ci-dessous), nous devons choisir attentivement des broches qui sont disponibles :





```

void selectRow (int r) {
    if (r==1) HAL_GPIO_WritePin(R1_GPIO_Port, R1_Pin, GPIO_PIN_RESET);
    else HAL_GPIO_WritePin(R1_GPIO_Port, R1_Pin, GPIO_PIN_SET);
    if (r==2) HAL_GPIO_WritePin(R2_GPIO_Port, R2_Pin, GPIO_PIN_RESET);
    else HAL_GPIO_WritePin(R2_GPIO_Port, R2_Pin, GPIO_PIN_SET);
    if (r==3) HAL_GPIO_WritePin(R3_GPIO_Port, R3_Pin, GPIO_PIN_RESET);
    else HAL_GPIO_WritePin(R3_GPIO_Port, R3_Pin, GPIO_PIN_SET);
    if (r==4) HAL_GPIO_WritePin(R4_GPIO_Port, R4_Pin, GPIO_PIN_RESET);
    else HAL_GPIO_WritePin(R4_GPIO_Port, R4_Pin, GPIO_PIN_SET);
}

int readCol() {
    int result = 0;
    if (HAL_GPIO_ReadPin(C1_GPIO_Port, C1_Pin) == GPIO_PIN_RESET) result += 1;
    if (HAL_GPIO_ReadPin(C2_GPIO_Port, C2_Pin) == GPIO_PIN_RESET) result += 2;
    if (HAL_GPIO_ReadPin(C3_GPIO_Port, C3_Pin) == GPIO_PIN_RESET) result += 4;
    if (HAL_GPIO_ReadPin(C4_GPIO_Port, C4_Pin) == GPIO_PIN_RESET) result += 8;
    return result;
}

```

**5. N'oubliez pas d'ajouter les lignes de code nécessaires pour rediriger la fonction printf() vers l'UART, tel que vu au laboratoire 3**

```

#include "stdio.h"

...
int fputc(int ch, FILE *f)
{
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}

```

**6. Pour tester nos fonctions, utilisez le code principal suivant :**

```

/* USER CODE BEGIN 3 */
    selectRow(1);HAL_Delay(10);
    int row1=readCol();
    printf("Row 1 : %x\r\n", row1);

    selectRow(2);HAL_Delay(10);
    int row2=readCol();
    printf("Row 2 : %x\r\n", row2);

    selectRow(3);HAL_Delay(10);
    int row3=readCol();
    printf("Row 3 : %x\r\n", row3);

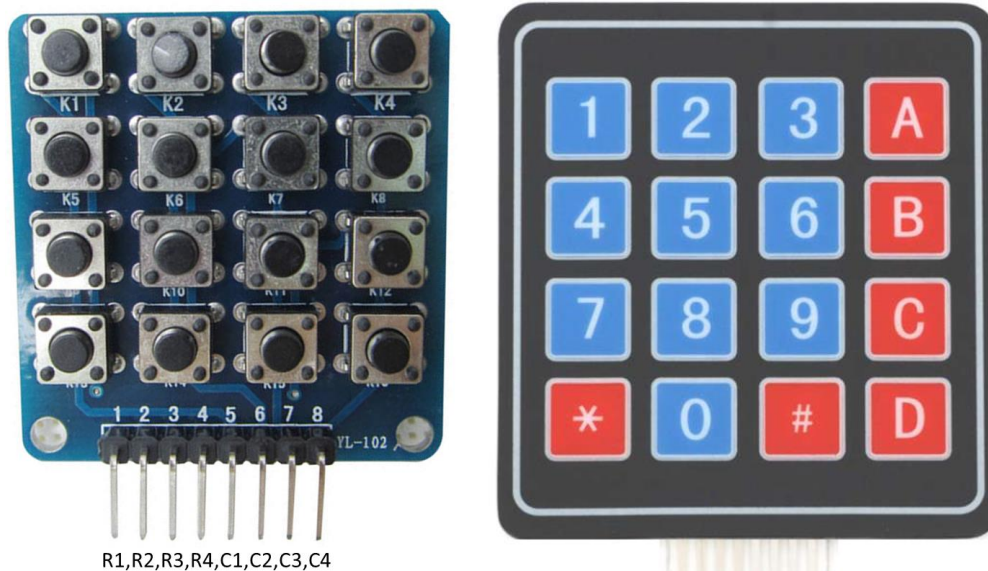
    selectRow(4);HAL_Delay(10);
    int row4=readCol();
    printf("Row 4 : %x\r\n", row4);

    HAL_Delay(1000);
}
/* USER CODE END 3 */

```



- Connectez le clavier sur le breadboard et utilisez les fils pour relier chaque broche du clavier au port qui lui correspond :



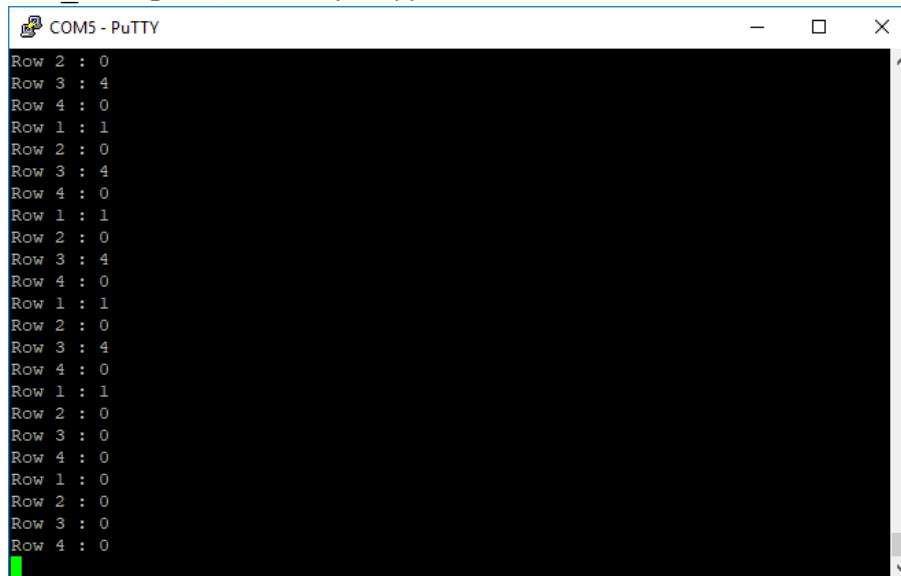
(pour illustration seulement, référez-vous à la fiche technique de votre clavier qui se trouve sur Moodle dans le répertoire « Fiches techniques pour le kit de laboratoire » )

Selon les kits que vous avez reçus, vous aurez soit un clavier vert munis de gros boutons noirs, soit un clavier bleu et rouge mou. Dans le premier cas, il faudra inverser les colonnes avec les rangées, sur base de l'image ci-dessus, ce qui donnera C1, C2, C3, C4, R1, R2, R3, R4. Dans le second cas, il faut inverser l'ordre des pins par rapport à l'image ci-dessus : 1 devient 8 et vice-versa pour donner C4, C3, C2, C1, R4, R3, R2, R1.

- Démarrez l'application PUTTY sur le port série du STLink Virtual COM Port (voir laboratoire 3)
- Compilez (F7), passez en mode debug (Ctrl-F5) et démarrez l'application (F5). Vous devriez avoir les messages suivants sur la console PUTTY :

```
COM5 - PuTTY
Row 2 : 0
Row 3 : 0
Row 4 : 0
Row 1 : 0
Row 2 : 0
Row 3 : 0
Row 4 : 0
Row 1 : 0
Row 2 : 0
Row 3 : 0
Row 4 : 0
Row 1 : 0
Row 2 : 0
Row 3 : 0
Row 4 : 0
Row 1 : 0
Row 2 : 0
Row 3 : 0
Row 4 : 0
Row 1 : 0
Row 2 : 0
Row 3 : 0
Row 4 : 0
```

10. Chaque fois que vous pesez sur une touche suffisamment longtemps (pour sortir du `HAL_Delay (...)`), vous voyez apparaître son code dans la console PUTTY :



11. Regardez ce qui se passe si vous pesez sur plusieurs touches en même temps.
12. Essayez de mettre en évidence des touches « fantômes » car le clavier n'a pas de diodes intégrées. Une touche fantôme est une touche qui semble pesée alors que, en réalité, elle ne l'est pas mais d'autres touches font court-circuit avec la même colonne.

## Fonctionnement de l'écran TFT LCD

L'écran TFT LCD dispose de son propre contrôleur intégré, avec lequel on communique au moyen de commandes de 8 bits permettant d'accéder aux registres internes du contrôleur à travers une interface SPI. L'initialisation de l'écran et les commandes graphiques sont disponibles dans des bibliothèques Open Source, que nous utiliserons dans ce laboratoire. Voici comment configurer votre microcontrôleur pour l'utilisation de l'écran.

1. Configurez le projet STM32Cube avec :
  - Le SPI1 : dans les « Categories » -> « Connectivity », sélectionnez le SPI1 en mode « Full-Duplex Master »  
Les broches PA5 (SPI1\_SCK), PA6 (SPI1\_MISO), PA7 (SPI1\_MOSI) vont être configurées automatiquement.  
Dans l'onglet « Configuration » -> « DMA settings », ajoutez une DMA Request SPI1\_TX.  
Dans l'onglet « Configuration » -> « NVIC settings », activez le SPI1 global interrupt
  - Les GPIO Output : PC7 (TFT\_DC), PB6 (TFT\_CS) et PB10 (Void\_Display\_Reset)
2. Copiez les fichiers *ili9341.h*, *ili9341\_font.h*, *ili9342\_gfx.h* dans le répertoire de votre projet /Inc
3. Copiez les fichiers *ili9341.c*, *ili9341\_font.c*, *ili9342\_gfx.c* dans le répertoire de votre projet /Src
4. Ajoutez les fichiers *ili9341.c*, *ili9341\_font.c*, *ili9342\_gfx.c* dans votre projet (Clic droit sur « Application/User » ensuite « Add Existing File ... »)
5. Éditez le fichier main.c comme suit :

```

/* USER CODE BEGIN Includes */
#include "ili9341.h"
#include "ili9341_gfx.h"
/* USER CODE END Includes */

...

/* USER CODE BEGIN PV */
ili9341_t *_screen;
/* USER CODE END PV */

```

The screenshot displays the STM32CubeIDE interface. The main editor shows the `main.c` file with the following content:

```

4  * @file      : main.c
5  * @brief     : Main program body
6  *
7  * @attention
8  *
9  * Copyright (c) 2023 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is provided AS-IS.
15 *
16 *****
17 */
18 /* USER CODE END Header */
19 /* Includes -----*/
20 #include "main.h"
21 #include "dma.h"
22 #include "spi.h"
23 #include "usart.h"
24 #include "gpio.h"
25
26 /* Private includes -----*/
27 /* USER CODE BEGIN Includes */
28 #include "ili9341.h"
29 #include "ili9341_gfx.h"
30 /* USER CODE END Includes */
31
32 /* Private typedef -----*/
33 /* USER CODE BEGIN PTD */
34
35 /* USER CODE END PTD */

```

The left sidebar shows the project structure for `FD_STM32`, including files like `main.c`, `gpio.c`, `dma.c`, `spi.c`, `usart.c`, `stm32f4xx_it.c`, `stm32f4xx_hal_msp.c`, `Drivers/STM32F4xx_HAL_Drv`, `Drivers/CMSIS`, and `system_stm32f4xx.c`.

The bottom window shows the **Build Output** with the following text:

```

compiling gpio.c...
compiling dma.c...
compiling usart.c...
compiling main.c...
compiling stm32f4xx_it.c...
compiling spi.c...
compiling stm32f4xx_hal_msp.c...
compiling ili9341.c...
compiling ili9341_gfx.c...
linking...
Program Size: Code=10804 RO-data=520 RW-data=12 ZI-data=3948
FromELF: creating hex file...
"FD_STM32\FD_STM32.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:03
Load "FD_STM32\FD_STM32.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 18:46:08

```

```

37  /* Private define -----*/
38  /* USER CODE BEGIN PD */
39
40  /* USER CODE END PD */
41
42  /* Private macro -----*/
43  /* USER CODE BEGIN PM */
44
45  /* USER CODE END PM */
46
47  /* Private variables -----*/
48
49  /* USER CODE BEGIN PV */
50  ili9341_t *_screen;
51
52  /* USER CODE END PV */
53
54  /* Private function prototypes -----*/
55  void SystemClock_Config(void);
56  /* USER CODE BEGIN PFP */
57
58  /* USER CODE END PFP */
59
60  /* Private user code -----*/

```

```

85  SystemClock_Config();
86
87  /* USER CODE BEGIN SysInit */
88
89  /* USER CODE END SysInit */
90
91  /* Initialize all configured peripherals */
92  MX_GPIO_Init();
93  MX_DMA_Init();
94  MX_USART2_UART_Init();
95  MX_SPI1_Init();
96  /* USER CODE BEGIN 2 */
97  _screen = ili9341_new(
98      &hspi1,
99      Void_Display_Reset_GPIO_Port, Void_Display_Reset_Pin,
100     TFT_CS_GPIO_Port,    TFT_CS_Pin,
101     TFT_DC_GPIO_Port,    TFT_DC_Pin,
102     isoLandscape,
103     NULL, NULL,
104     NULL, NULL,
105     itsNotSupported,
106     itsNormalized);
107
108  ili9341_fill_screen(_screen, ILI9341_BLACK);
109
110  ili9341_draw_line(_screen, ILI9341_YELLOW, 50, 200, 200, 200);
111
112  ili9341_draw_rect(_screen, ILI9341_BLUE, 40, 40, 50, 100);
113
114  ili9341_draw_circle(_screen, ILI9341_RED, 250, 100, 50);
115
116  ili9341_text_attr_t text_attr = {&ili9341_font_11x18, ILI9341_WHITE, ILI9341_BLACK, 100, 100};
117  char text[] = {"! Hello !"};
118  ili9341_draw_string(_screen, text_attr, text);
119
120  /* USER CODE END 2 */
121
122  /* Infinite loop */

```

Build Output

```

"FD_STM32\FD_STM32.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:02
Load "FD_STM32\FD_STM32.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 17:55:06
Load "FD_STM32\FD_STM32.axf"
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 17:55:09

```

(Vous pouvez copier-coller le code ci-dessous)

```

_screen = ili9341_new(
    &hspi1,
    Void_Display_Reset_GPIO_Port, Void_Display_Reset_Pin,
    TFT_CS_GPIO_Port,    TFT_CS_Pin,
    TFT_DC_GPIO_Port,    TFT_DC_Pin,
    isoLandscape,

```

```

NULL, NULL,
NULL, NULL,
itsNotSupported,
itnNormalized);

ili9341_fill_screen(_screen, ILI9341_BLACK);
ili9341_draw_line(_screen, ILI9341_YELLOW, 50, 200, 200,
200);
ili9341_draw_rect(_screen, ILI9341_BLUE, 40, 40, 50, 100);
ili9341_draw_circle(_screen, ILI9341_RED, 250, 100, 50);
ili9341_text_attr_t text_attr = {&ili9341_font_11x18,
ILI9341_WHITE, ILI9341_BLACK, 0, 0};
char text[] = {"! Hello !"};
ili9341_draw_string(_screen, text_attr, text);

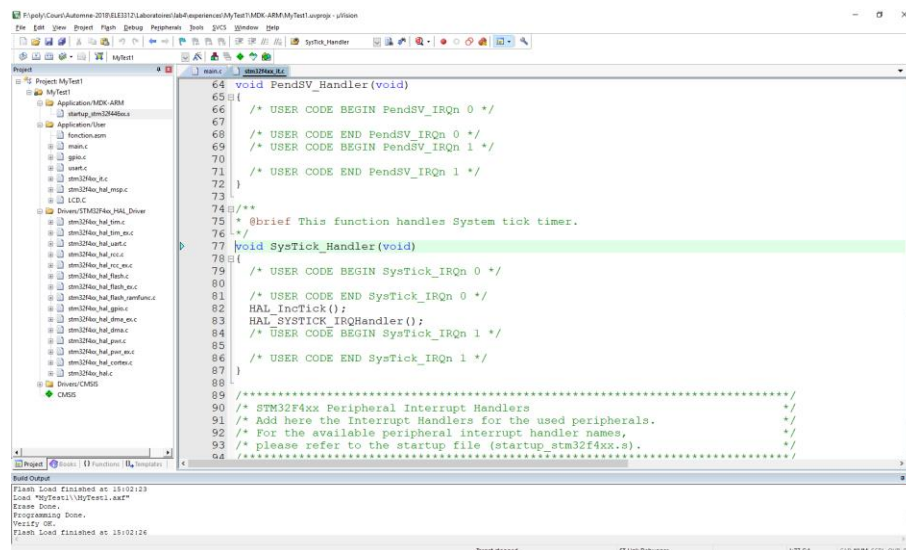
```

6. Enfoncez l'écran dans la carte de développement (connecteur Arduino) si ce n'est pas déjà fait.
7. Compilez (F7), passez en mode debug (Ctrl-F5) et démarrez l'application (F5).
8. Vous devriez voir apparaître trois formes à l'écran, avec au centre le message « ! Hello ! »

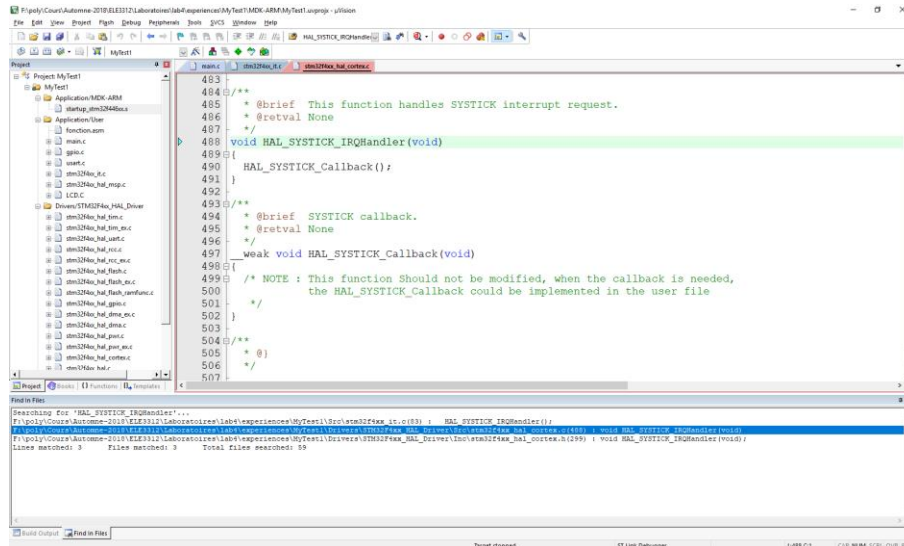
## Expérience 1

*Réalisez une application qui affiche une horloge*

Pour y parvenir, nous allons utiliser le *Systick Timer*. Sans entrer dans tous les détails de ce Timer à ce stade, il faut savoir que le *Systick Timer* est une horloge interne au microcontrôleur qui permet de déclencher une interruption à intervalle régulier, typiquement à chaque milliseconde. Lorsqu'une interruption est déclenchée, une routine est appelée pour la traiter. Dans le cas présent, il s'agit de la routine appelée `SysTick_Handler`, que vous pouvez trouver dans le fichier `stm32f4xx_it.c` :



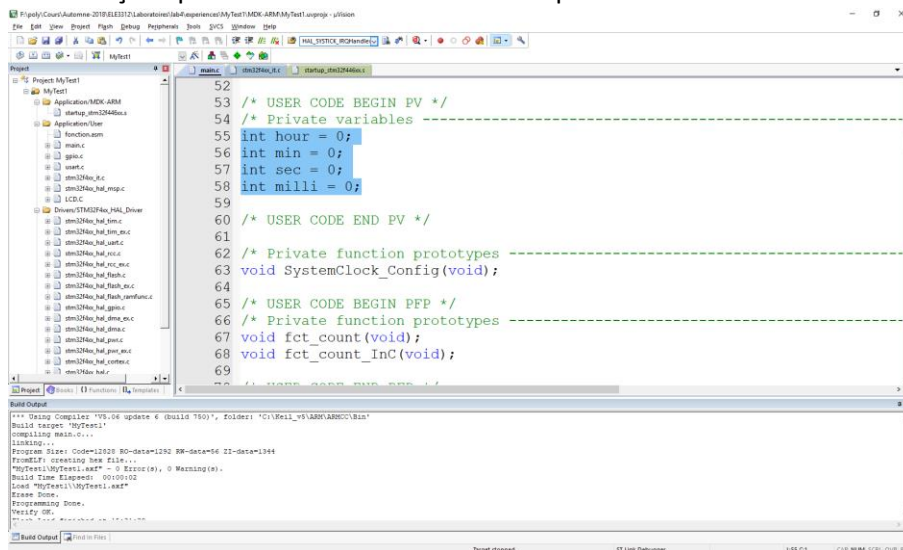
Cette routine appelle elle-même une routine `HAL_SYSTICK_IRQHandler()` (fichier `stm32f4xx_hal_cortex.c`, ajoutez-la si elle n'apparaît pas dans le `SysTick_Handler`):



... qui finalement appelle une fonction `HAL_SYSTICK_Callback(void)` ... qui ne fait rien pour le moment.

Cette dernière fonction est précédée du mot-clef `__weak` dans sa définition, ce qui nous autorise à redéfinir cette fonction à un autre endroit. Nous allons donc définir une nouvelle fonction `HAL_SYSTICK_Callback(void)` dans le fichier `main.c`. Cette fonction sera automatiquement appelée à chaque milliseconde. Elle doit donc être très brève sinon elle n'aura pas le temps de s'exécuter avant qu'elle doive être appelée à nouveau. La fonction que nous allons écrire va compter les heures, minutes, secondes et millisecondes écoulées depuis le début du programme.

## 1. Commençons par déclarer les variables correspondantes :



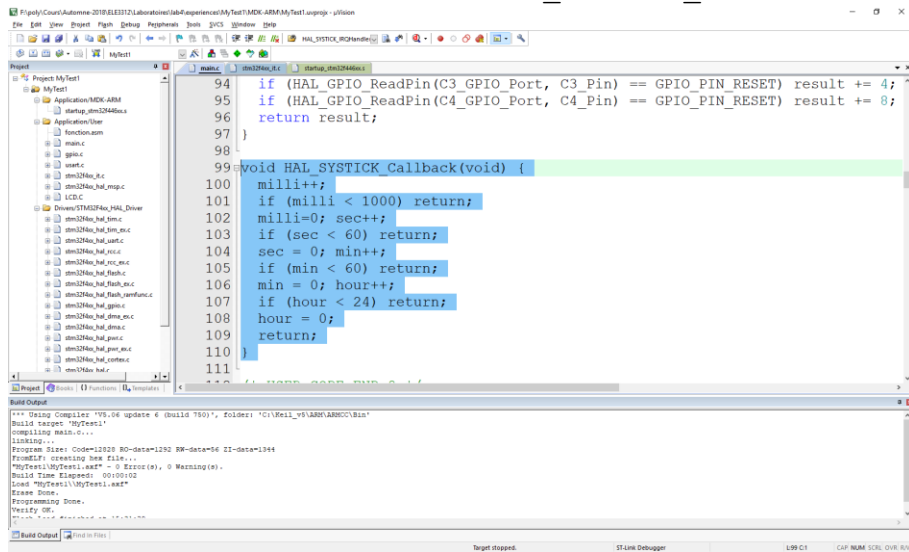
(Vous pouvez copier-coller le code ci-dessous)

```

int hour = 0;
int min = 0;
int sec = 0;
int milli = 0;

```

## 2. Écrivons maintenant la nouvelle routine HAL\_SYSTICK\_Callback(void) :

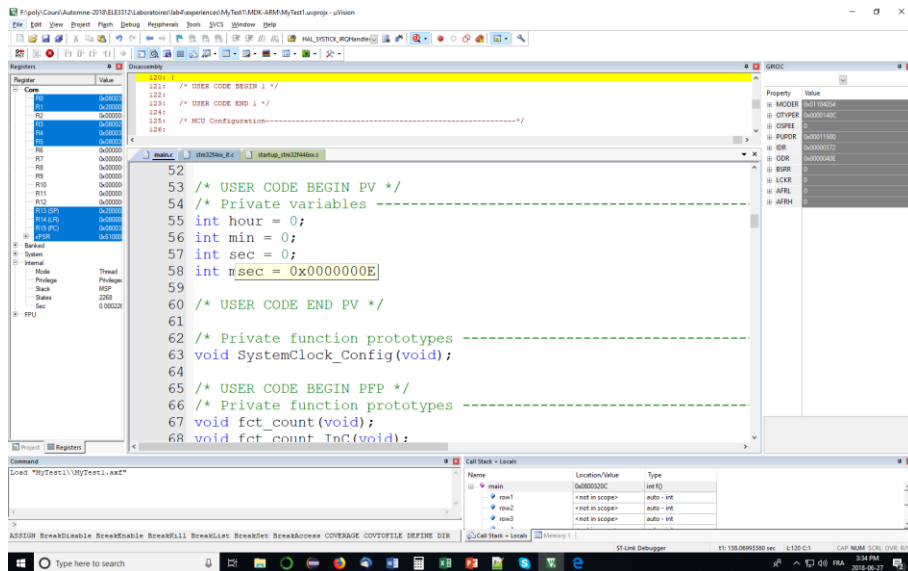


(Vous pouvez copier-coller le code ci-dessous)

```
void HAL_SYSTICK_Callback(void) {
    milli++;
    if (milli < 1000) return;
    milli=0; sec++;
    if (sec < 60) return;
    sec = 0; min++;
    if (min < 60) return;
    min = 0; hour++;
    if (hour < 24) return;
    hour = 0;
    return;
}
```

3. Compilez (F7), passez en mode debug (Ctrl-F5) et démarrez l'application (F5).
4. En passant avec la souris sur la variable `sec`, vous devriez voir sa valeur augmenter avec le temps :





- Écrivons maintenant le code qui va afficher le temps à l'écran :

```

131 while (1)
132 {
133     /* USER CODE END WHILE */
134
135     /* USER CODE BEGIN 3 */
136     char buffer[20] = {0};
137     sprintf(buffer, "%2i:%2i:%2i",hour,min,sec);
138     ili9341_fill_screen(_screen, ILI9341_BLACK);
139     ili9341_text_attr_t time_attr = {&ili9341_font_11x18, ILI9341_WHITE, ILI9341_BLACK,0,0};
140     ili9341_draw_string(_screen, time_attr,buffer);
141 }
142 /* USER CODE END 3 */
143 }

```

(Vous pouvez copier-coller le code ci-dessous)

```

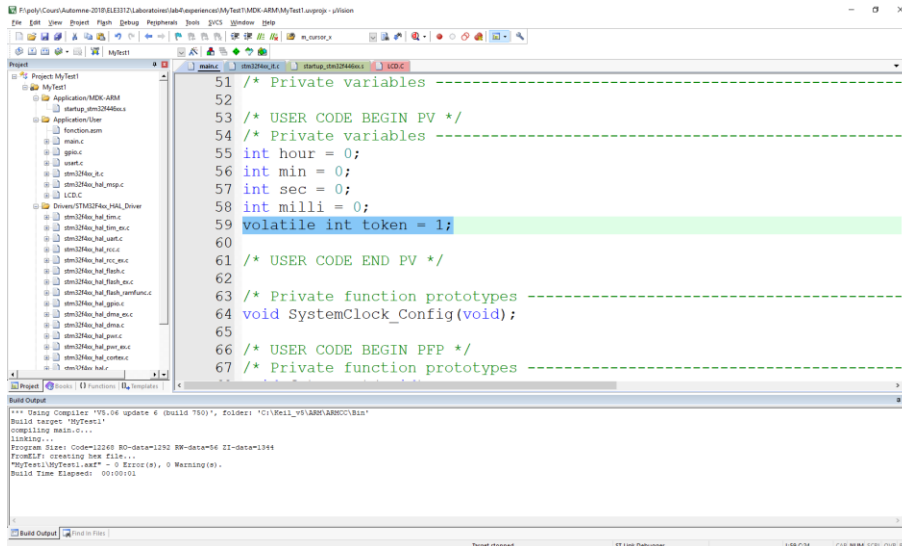
char buffer[20] = {0};
sprintf(buffer, "%2i:%2i:%2i",hour,min,sec);
ili9341_fill_screen(_screen, ILI9341_BLACK);
ili9341_text_attr_t time_attr = {&ili9341_font_11x18,
ILI9341_WHITE, ILI9341_BLACK,0,0};
ili9341_draw_string(_screen, time_attr,buffer);

```

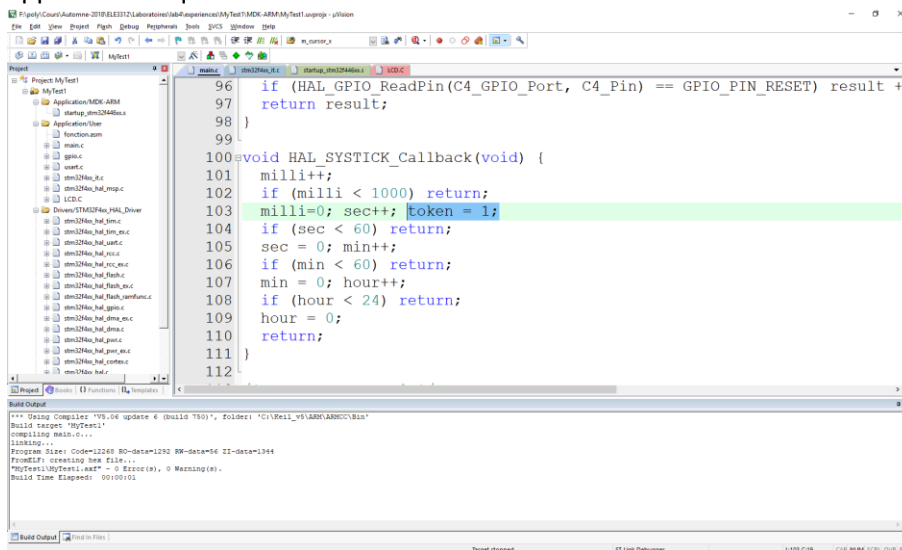
NB: pour utiliser la fonction « sprintf », n'oubliez pas d'inclure la librairie « stdio.h »

- Compilez (F7), passez en mode debug (Ctrl-F5) et démarrez l'application (F5).
- Vous observez un écran qui clignote de manière très dérangeante. Cela est dû au fait que dès qu'un affichage est réalisé, il efface l'écran et recommence, et ce plusieurs fois par seconde. Pour éviter cela, nous allons utiliser une variable `token` qui sera validée chaque fois que les secondes changent. Essayez donc le code suivant :





8. Le mot clef `volatile` indique que la variable peut aussi être modifiée en dehors de l'exécution normale d'un programme (par exemple pendant une interruption). La routine appelée à chaque milliseconde devient :



9. Tandis que le code de la boucle principale devient :

```

135 /* USER CODE BEGIN 3 */
136 while (token == 0);
137 token = 0;
138 char buffer[20] = {0};
139 sprintf(buffer, "%2i:%2i:%2i", hour, min, sec);
140 ili9341_fill_screen(_screen, ILI9341_BLACK);
141 ili9341_text_attr_t time_attr = {&ili9341_font_11x18, ILI9341_WHITE, ILI9341_BLACK, 0, 0};
142 ili9341_draw_string(_screen, time_attr, buffer);
143 }
144 /* USER CODE END 3 */

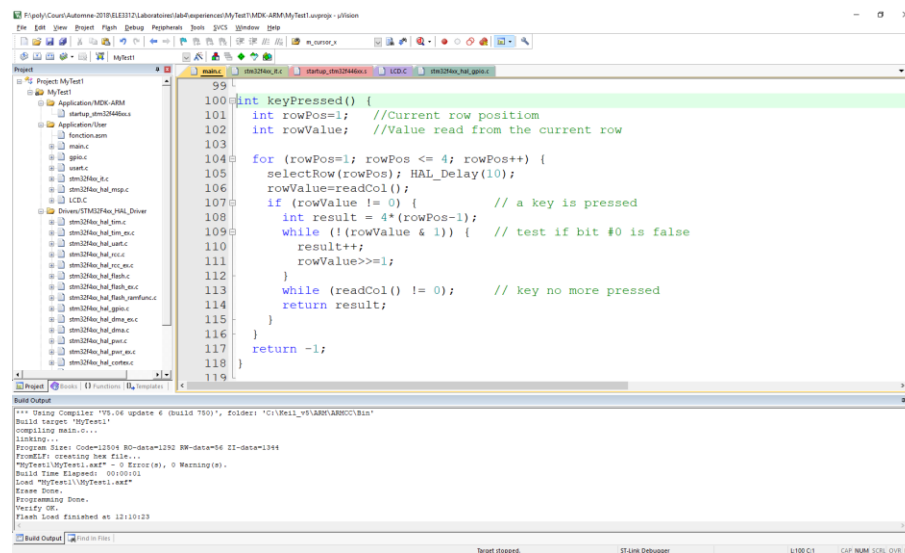
```

10. Maintenant, la boucle while attend que la variable `token` passe à l'état vrai avant de faire l'affichage. Comme cela arrive au moment où l'interruption change les secondes, l'écran se rafraîchit maintenant à chaque seconde. Évidemment, il ne faut pas oublier de remettre la variable `token` à zéro dès qu'on sort de la boucle while.

## Expérience 2

*Réalisez une application qui affiche les chiffres pesés au clavier.*

Nous avons maintenant presque tout en main pour réaliser une application avec un clavier et un écran. Il nous manque seulement une fonction qui lit le numéro de la touche pesée au clavier, le cas échéant. Nous allons écrire la fonction `keyPressed` suivante qui lit le clavier. Si aucune touche n'est pesée, elle retourne la valeur -1. Sinon, elle attend qu'on arrête de peser sur la touche et elle retourne le numéro de la touche qui était pesée :



(Vous pouvez copier-coller le code ci-dessous)

```
int keyPressed() {
    int rowPos=1;           //Current row position
    int rowValue;           //Value read from the current row

    for (rowPos=1; rowPos <= 4; rowPos++) {
        selectRow(rowPos); HAL_Delay(10);
        rowValue=readCol();
        if (rowValue != 0) { // a key is pressed
            int result = 4*(rowPos-1);
            while (!(rowValue & 1)) { // test if bit #0 is false
                result++;
                rowValue>>=1;
            }
            while (readCol() != 0); // key no more pressed
            return result;
        }
    }
    return -1;
}
```

```

    }
    return -1;
}

```

Assurez-vous de bien comprendre comment fonctionne la méthode `keyPressed` : La première boucle (sur la variable `rowPos`) va tester chaque rangée, une après l'autre. Lorsqu'une rangée contient au moins une touche pressée (`rowValue` différent de 0), la boucle `while` va regarder quel est le premier bit vrai en commençant par les bits de poids faible. Ensuite, la méthode attend qu'on arrête de peser sur la touche. Elle retourne alors le numéro de la touche qui avait été pesée (`result` : assurez-vous de comprendre comment ce numéro est calculé). Si aucune touche n'est pesée, la méthode retourne la constante -1.

Pour tester le clavier et l'écran, vous pouvez utiliser le corps de boucle principale (`main`) suivant :

```

151     while (1)
152     {
153         /* USER CODE END WHILE */
154
155         /* USER CODE BEGIN 3 */
156         // while (token == 0);
157         // token = 0;
158         int key;
159         while ((key = keyPressed()) == -1); // wait until a key is pressed
160         char buffer[20] = {0};
161         // sprintf(buffer, "%2i:%2i:%2i",hour,min,sec);
162         ili9341_fill_screen(_screen, ILI9341_BLACK);
163         ili9341_text_attr_t time_attr = {&ili9341_font_11x18, ILI9341_WHITE, ILI9341_BLACK,0,0};
164         ili9341_draw_string(_screen, time_attr,buffer);
165     }
166     /* USER CODE END 3 */
167 }

```

## 2<sup>ème</sup> partie : Laboratoire en salle à Polytechnique

Un énoncé parmi les suivants sera assigné aléatoirement à chaque équipe.

### Énoncé 1

Écrire une application qui permet de faire des dessins à l'écran avec un crayon dont le diamètre est de N pixels (N configurable). Les commandes sont :

- 1 => Haut à gauche
- 2 => Haut
- 3 => Haut à droite
- 4 => Gauche
- 5=> Lever/poser le crayon
- 6 => Droite
- 7 => Bas à gauche
- 8 => Bas
- 9 => Bas à droite

Même quand on n'appuie sur aucune touche, le crayon est toujours en mouvement, à raison d'une position à chaque 300ms (dans la direction indiquée par la dernière touche qui a été pesée). Lorsque le crayon est posé, il laisse une trace de N pixel(s) rouge(s) de large derrière lui. Lorsque le crayon est levé, il apparaît en vert (un seul pixel) et ne laisse qu'une fine trace verte derrière lui (un seul pixel) quand il se déplace. Pour gérer la vitesse de déplacement du crayon (300ms par position), vous utiliserez directement l'interruption du SysTick Timer (**interdiction d'utiliser HAL\_Delay() ou toute autre fonction de la bibliothèque**) qui enverra un nouveau jeton à l'application à chaque 300ms. Le jeton envoyé sera le numéro de la touche enfoncée si c'est le cas, et zéro si aucune touche n'est enfoncée.

### Énoncé 2

Intégrez la gestion du clavier dans l'interruption du SysTick Timer. Dès qu'une touche est pesée, un nouveau token (que vous appellerez `keyToken`) est mis à jour avec le numéro de la touche pesée. Toutefois, vous ne lirez qu'une rangée à chaque interruption. Donc, à chaque interruption, vous lisez la rangée courante, vous regardez s'il y a une touche pesée et finalement vous préparez la rangée qui sera lue à la prochaine interruption. De cette manière, vous laissez un petit délai entre le moment où vous positionnez une rangée et le moment où vous lisez les valeurs de cette rangée. Pour illustrer le bon fonctionnement de votre clavier, intégrez-le dans l'application d'horloge vue précédemment. Utilisez 6 touches pour augmenter ou diminuer les heures/minutes/secondes.

### Énoncé 3

Réalisez un début de jeu de « Pong » dans lequel une balle de N pixels (N étant configurable) de diamètre se déplace et rebondit sur les bords de l'écran. La direction initiale est de 45 degrés par rapport à l'horizontale. ). Lorsqu'on pèse sur une touche, la balle « rebondit » sur un mur vertical invisible. On souhaite un rafraichissement de l'écran 5 fois par seconde. Pour gérer la vitesse de déplacement de la balle, vous utiliserez directement l'interruption du SysTick Timer (**interdiction d'utiliser HAL\_Delay() ou toute autre fonction de la bibliothèque**).