

ELE3312
Microcontrôleurs et applications
Laboratoire 7

Auteur : Jean Pierre David

Introduction

Contrairement à ce qu'on pourrait penser, lorsqu'on écrit un programme, c'est rare que le microcontrôleur l'exécute du début jusqu'à la fin. Il arrive souvent qu'il soit interrompu. Lorsque cela arrive, il arrête l'exécution du programme en cours pour exécuter un autre programme plus prioritaire. Lorsque ce programme est terminé, le processeur revient au premier programme pour le continuer. Évidemment, il faut s'assurer que l'état de tous les registres est restauré avant de reprendre l'exécution. Les sources d'interruptions sont multiples. Typiquement, elles viennent des périphériques. Dans ce laboratoire, vous allez apprendre à utiliser les interruptions et les horloges (*timers*).

Objectifs

1. Découvrir et utiliser les horloges, et en particulier le *Systick Timer*
2. Découvrir le concept de machine à états logicielle

1^{ère} partie : préparation à la maison

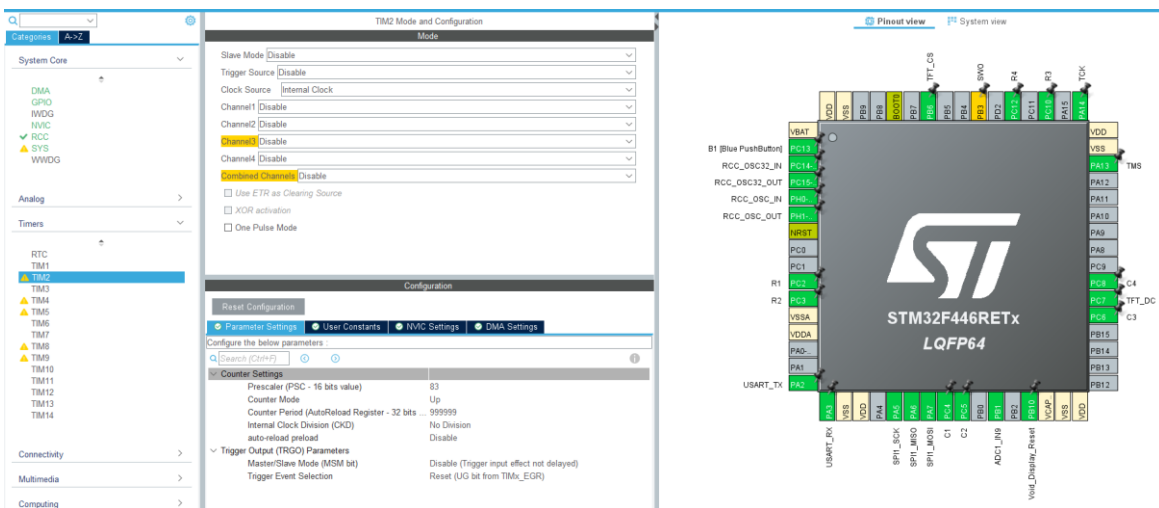
Un projet de départ est fourni dans l'archive **projet_laboratoire7.zip**.

Ce projet se fonde sur le laboratoire 6. La configuration pour les broches du clavier est donc déjà présente. Il inclut aussi le code nécessaire afin de compléter et de tester les expériences 1 à 3. Vous n'avez qu'à commenter/décommenter les bouts de code pour sélectionner l'expérience souhaitée.

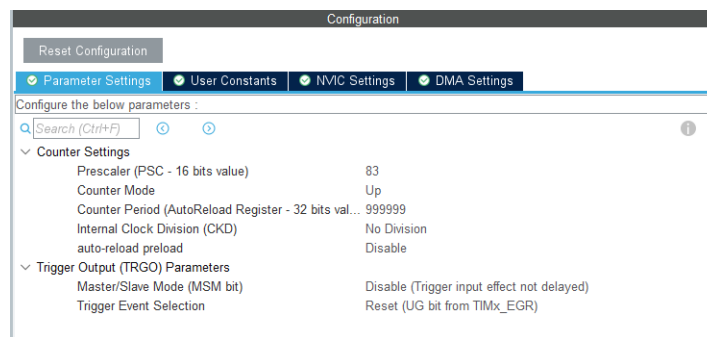
ATTENTION : les instructions de configuration surlignées en bleu dans la suite de l'énoncé doivent tout de même être effectuées.

Expérience 1

1. Ouvrez le projet STM32Cube et activez le *Timer 2* :



2. Allez dans l'onglet « Configuration », ouvrez les *Parameter Settings* du *Timer 2* configurez-le comme suit :



3. L'horloge interne (qui est à 84 Mhz sur votre processeur) est d'abord divisée par (83+1) dans le *Prescaler* pour arriver à une horloge de 1 Mhz. Ensuite, on configure le *Timer 2* pour compter de 0 à 999999, soit 1 million de valeurs. L'interruption sera donc générée à chaque seconde.

4. Il faut aussi activer les interruptions :

Configuration			
Reset Configuration			
Parameter Settings	User Constants	NVIC Settings	DMA Settings
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	<input checked="" type="checkbox"/>	0	0

5. Régénérez le code (Ctrl-Shift-G) et ouvrez le projet Keil correspondant.

6. Démarrez le *Timer 2* :

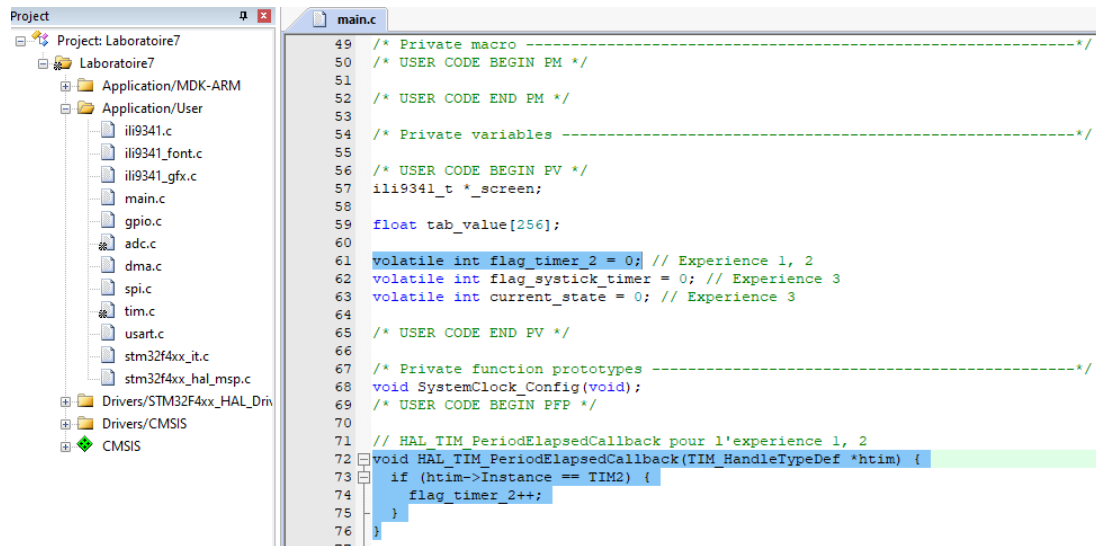
```

139 MX_TIM2_Init();
140 /* USER CODE BEGIN 2 */
141 _screen = ili9341_new(
142     &hspi1,
143     Void_Display_Reset_GPIO_Port, Void_Display_Reset_Pin,
144     TFT_CS_GPIO_Port, TFT_CS_Pin,
145     TFT_DC_GPIO_Port, TFT_DC_Pin,
146     isoLandscape,
147     NULL, NULL,
148     NULL, NULL,
149     itsNotSupported,
150     itsNotSupported);
151
152 ili9341_fill_screen(_screen, ILI9341_BLACK);
153 ili9341_text_attr_t text_attr = {&ili9341_font_11x18, ILI9341_WHITE, ILI9341_BLACK, 0, 0};
154
155 HAL_TIM_Base_Start_IT(&htim2); // Experience 1
156
157 /* USER CODE END 2 */
158
159 /* Infinite loop */
160 /* USER CODE BEGIN WHILE */
161 while (1)
162 {

```

HAL_TIM_Base_Start_IT(&htim2);

7. Nous utiliserons un drapeau de type volatile. *volatile* signifie que la valeur peut être modifiée en dehors du flot d'exécution normal (ici ce sera lors de l'interruption). Également, définissez la fonction qui sera appelée automatiquement par les interruptions du *Timer 2*:

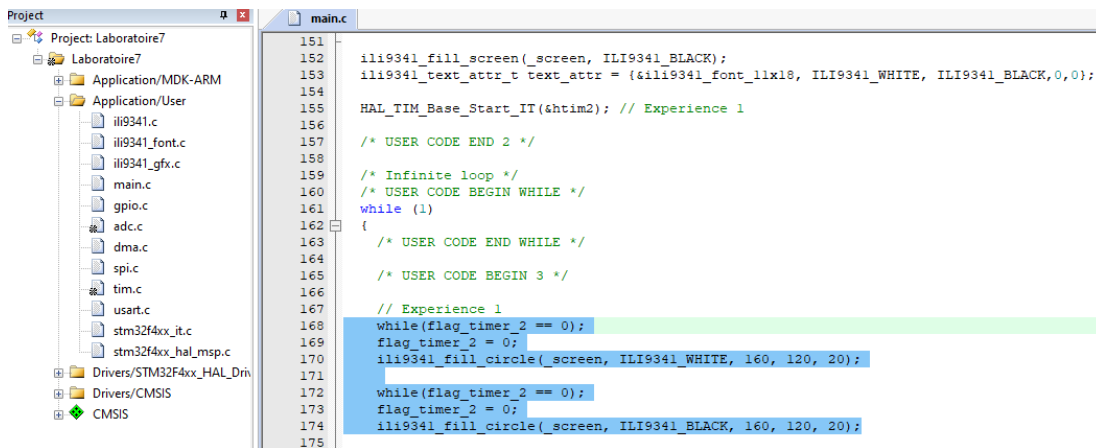


```
49 /* Private macro ----- */
50 /* USER CODE BEGIN PM */
51
52 /* USER CODE END PM */
53
54 /* Private variables ----- */
55
56 /* USER CODE BEGIN PV */
57 ili9341_t *_screen;
58
59 float tab_value[256];
60
61 volatile int flag_timer_2 = 0; // Experience 1, 2
62 volatile int flag_systick_timer = 0; // Experience 3
63 volatile int current_state = 0; // Experience 3
64
65 /* USER CODE END PV */
66
67 /* Private function prototypes ----- */
68 void SystemClock_Config(void);
69 /* USER CODE BEGIN PFP */
70
71 // HAL_TIM_PeriodElapsedCallback pour l'experience 1, 2
72 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
73     if (htim->Instance == TIM2) {
74         flag_timer_2++;
75     }
76 }
```

```
volatile int flag_timer_2 = 0;
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {
        flag_timer_2++;
    }
}
```

8. Ajouter ce code dans le main



```
151
152 ili9341_fill_screen(_screen, ILI9341_BLACK);
153 ili9341_text_attr_t text_attr = {&ili9341_font_11x18, ILI9341_WHITE, ILI9341_BLACK,0,0};
154
155 HAL_TIM_Base_Start_IT(&htim2); // Experience 1
156
157 /* USER CODE END 2 */
158
159 /* Infinite loop */
160 /* USER CODE BEGIN WHILE */
161 while (1)
162 {
163     /* USER CODE END WHILE */
164
165     /* USER CODE BEGIN 3 */
166
167     // Experience 1
168     while(flag_timer_2 == 0);
169     flag_timer_2 = 0;
170     ili9341_fill_circle(_screen, ILI9341_WHITE, 160, 120, 20);
171
172     while(flag_timer_2 == 0);
173     flag_timer_2 = 0;
174     ili9341_fill_circle(_screen, ILI9341_BLACK, 160, 120, 20);
175 }
```

```
while(flag_timer_2 == 0);
flag_timer_2 = 0;
ili9341_fill_circle(_screen, ILI9341_WHITE, 160, 120, 20);
while(flag_timer_2 == 0);
```

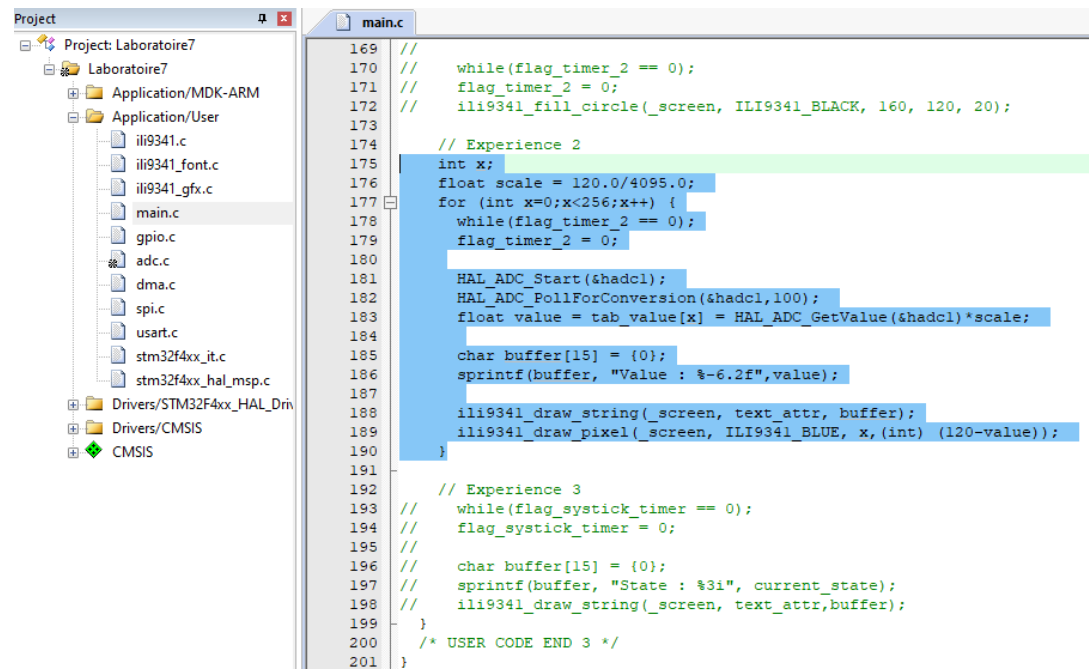
```
flag_timer_2 = 0;  
ili9341_fill_circle(_screen, ILI9341_BLACK, 160, 120, 20);
```

9. Compilez et exécutez le programme en mode debug. (Vous pouvez également compiler (F7) + téléverser (F8) et ensuite appuyer sur le bouton RESET sur le côté de votre écran.) Vous devriez voir un cercle blanc au centre de l'écran clignoté à tous les secondes.

Expérience 2

Nous allons maintenant réécrire l'application du laboratoire 6 en utilisant le Timer 2 pour déclencher la conversion analogique/digital. De cette manière, l'intervalle entre deux échantillons sera beaucoup plus précis. Nous allons aussi avoir besoin d'un moyen de communication entre la routine d'interruption et le programme principal.

1. Dans le programme principal, il suffit d'attendre que le drapeau `flag_timer_2` soit vrai. À ce moment-là, on peut lire la valeur de l'ADC et l'afficher comme auparavant. Il ne faut pas oublier de remettre le flag à 0 :



```
169 //
170 // while(flag_timer_2 == 0);
171 // flag_timer_2 = 0;
172 // ili9341_fill_circle(_screen, ILI9341_BLACK, 160, 120, 20);
173
174 // Experience 2
175 int x;
176 float scale = 120.0/4095.0;
177 for (int x=0;x<256;x++) {
178     while(flag_timer_2 == 0);
179     flag_timer_2 = 0;
180
181     HAL_ADC_Start(&hadcl);
182     HAL_ADC_PollForConversion(&hadcl,100);
183     float value = tab_value[x] = HAL_ADC_GetValue(&hadcl)*scale;
184
185     char buffer[15] = {0};
186     sprintf(buffer, "Value : %-6.2f",value);
187
188     ili9341_draw_string(_screen, text_attr, buffer);
189     ili9341_draw_pixel(_screen, ILI9341_BLUE, x,(int) (120-value));
190 }
191
192 // Experience 3
193 // while(flag_systick_timer == 0);
194 // flag_systick_timer = 0;
195 //
196 // char buffer[15] = {0};
197 // sprintf(buffer, "State : %3i", current_state);
198 // ili9341_draw_string(_screen, text_attr,buffer);
199 //
200 /* USER CODE END 3 */
201 }
```

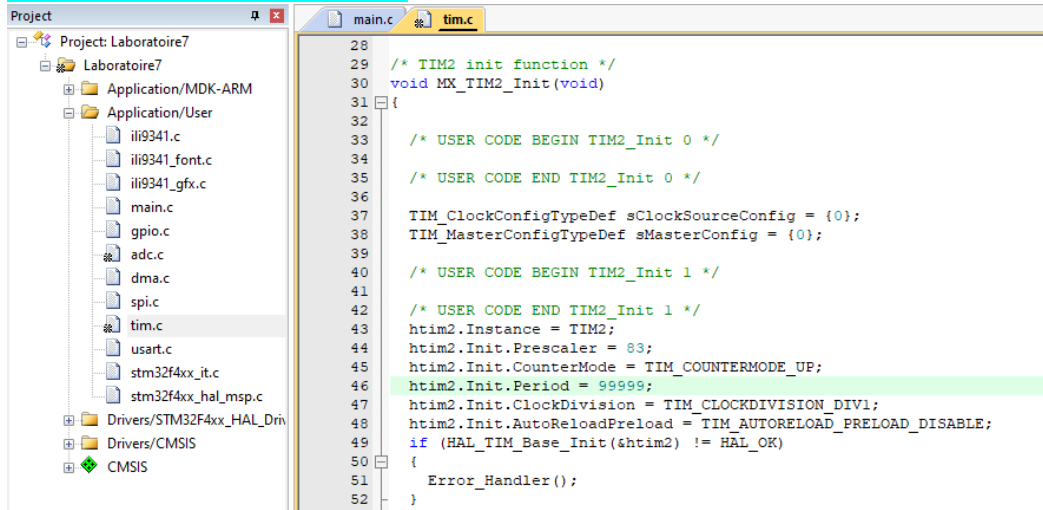
```
int x;
float scale = 120.0/4095.0;
for (int x=0;x<256;x++) {
    while(flag_timer_2 == 0);
    flag_timer_2 = 0;

    HAL_ADC_Start(&hadcl);
    HAL_ADC_PollForConversion(&hadcl,100);
    float value = tab_value[x] = HAL_ADC_GetValue(&hadcl)* scale;

    char buffer[15] = {0};
    sprintf(buffer, "Value : %-6.2f",value);

    ili9341_draw_string(_screen, text_attr, buffer);
    ili9341_draw_pixel(_screen, ILI9341_BLUE, x,(int) (120-value));
}
```

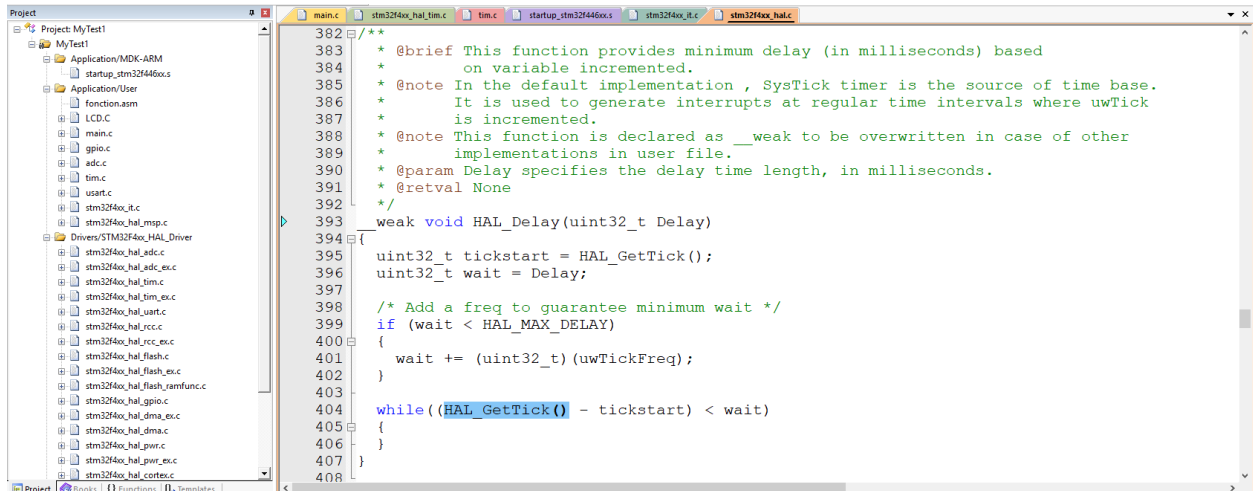
2. Il vous appartient de configurer le *Timer 2* pour avoir la bonne période d'échantillonnage (100ms). Pour cela, vous pouvez utiliser STM32Cube ou encore modifier les paramètres directement dans le fichier `tim.c` :



```
28
29 /* TIM2 init function */
30 void MX_TIM2_Init(void)
31 {
32
33     /* USER CODE BEGIN TIM2_Init 0 */
34
35     /* USER CODE END TIM2_Init 0 */
36
37     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
38     TIM_MasterConfigTypeDef sMasterConfig = {0};
39
40     /* USER CODE BEGIN TIM2_Init 1 */
41
42     /* USER CODE END TIM2_Init 1 */
43     htim2.Instance = TIM2;
44     htim2.Init.Prescaler = 83;
45     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
46     htim2.Init.Period = 99999;
47     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
48     htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
49     if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
50     {
51         Error_Handler();
52     }
```

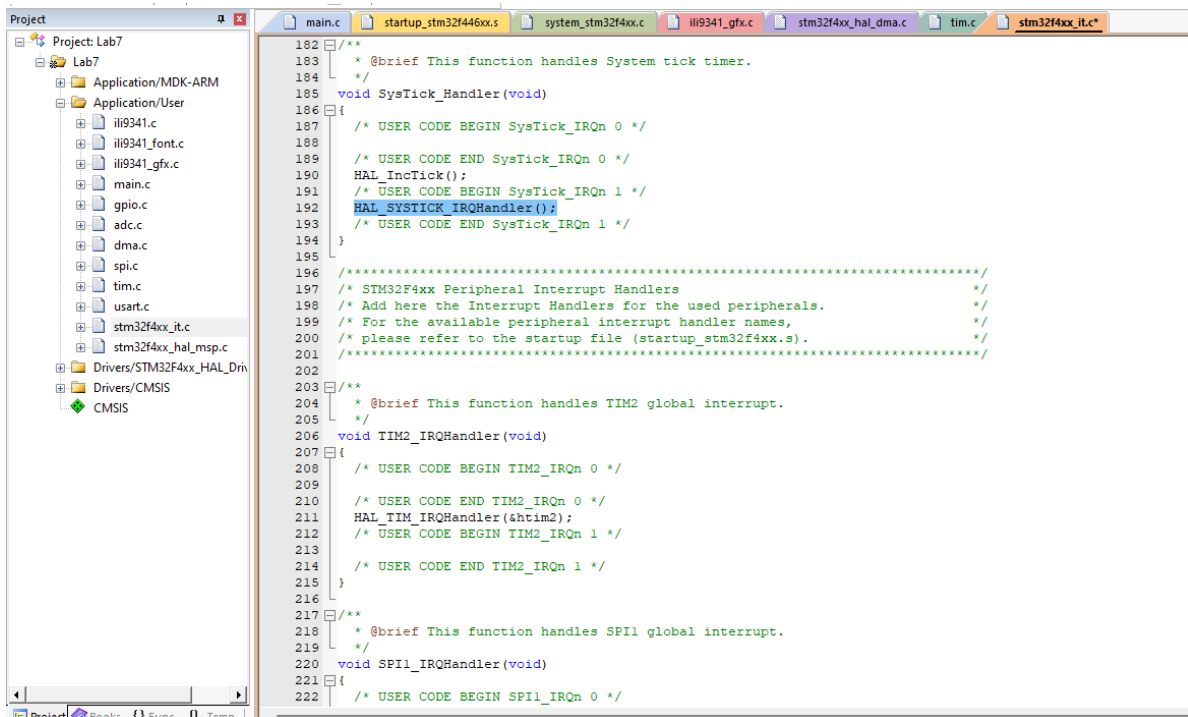

Expérience 3

Il y a une horloge qui est toujours configurée par défaut. C'est le *Systick Timer*. En général, il est programmé pour générer une interruption à chaque milliseconde. Cette horloge est notamment utilisée par les fonctions HAL_GetTick() et HAL_Delay(). La première retourne le nombre de millisecondes écoulées depuis le démarrage et la deuxième, que vous connaissez déjà, est implantée comme suit :

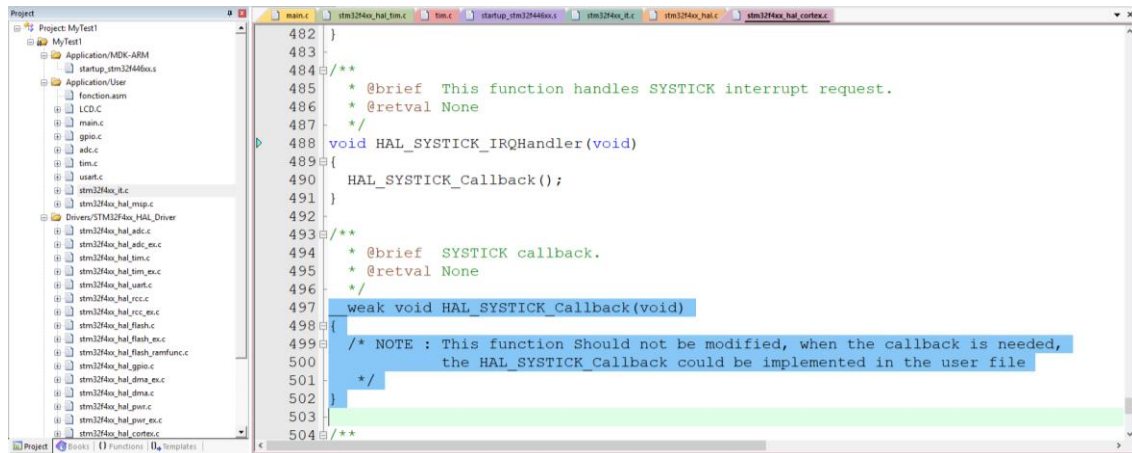


```
382 /**
383  * @brief This function provides minimum delay (in milliseconds) based
384  *        on variable incremented.
385  * @note In the default implementation , SysTick timer is the source of time base.
386  *       It is used to generate interrupts at regular time intervals where uwTick
387  *       is incremented.
388  * @note This function is declared as __weak to be overwritten in case of other
389  *       implementations in user file.
390  * @param Delay specifies the delay time length, in milliseconds.
391  * @retval None
392  */
393 __weak void HAL_Delay(uint32_t Delay)
394 {
395     uint32_t tickstart = HAL_GetTick();
396     uint32_t wait = Delay;
397
398     /* Add a freq to guarantee minimum wait */
399     if (wait < HAL_MAX_DELAY)
400     {
401         wait += (uint32_t)(uwTickFreq);
402     }
403
404     while((HAL_GetTick() - tickstart) < wait)
405     {
406     }
407 }
408
```

Le *Systick Timer* a une fonction de callback HAL_SYSTICK_Callback() définie comme suit. Elle est donc appelée à chaque milliseconde et peut être utilisée pour réaliser diverses tâches. Afin de l'utiliser, assurez-vous que la fonction HAL_SYSTICK_IRQHandler() est bien appelée par le SysTick_Handler() (voir laboratoire 4).

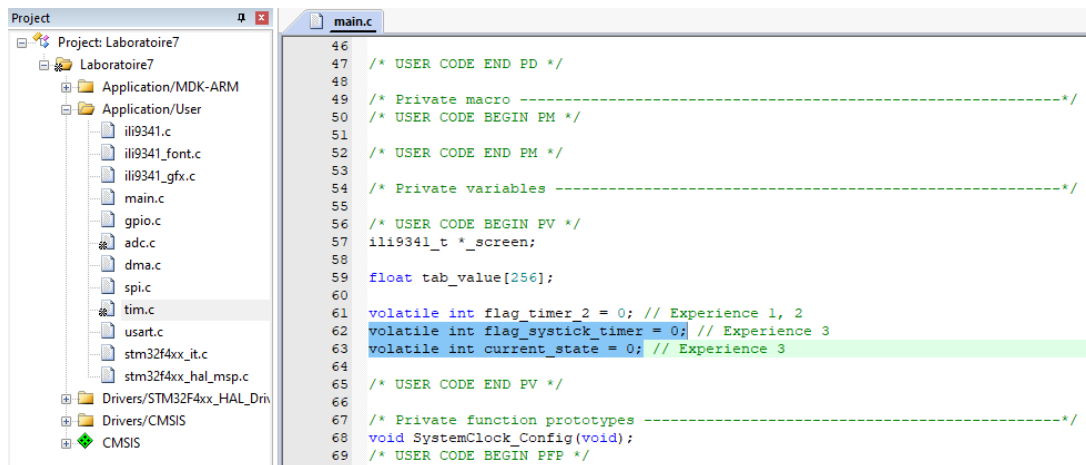


```
182 /**
183  * @brief This function handles System tick timer.
184  */
185 void SysTick_Handler(void)
186 {
187     /* USER CODE BEGIN SysTick_IRQn 0 */
188
189     /* USER CODE END SysTick_IRQn 0 */
190     HAL_IncTick();
191     /* USER CODE BEGIN SysTick_IRQn 1 */
192     HAL_SYSTICK_IRQHandler();
193     /* USER CODE END SysTick_IRQn 1 */
194 }
195
196 /*****
197  * STM32F4xx Peripheral Interrupt Handlers
198  * Add here the Interrupt Handlers for the used peripherals.
199  * For the available peripheral interrupt handler names,
200  * please refer to the startup file (startup_stm32f4xx.s).
201  *****/
202
203 /**
204  * @brief This function handles TIM2 global interrupt.
205  */
206 void TIM2_IRQHandler(void)
207 {
208     /* USER CODE BEGIN TIM2_IRQn 0 */
209
210     /* USER CODE END TIM2_IRQn 0 */
211     HAL_TIM_IRQHandler(&htim2);
212     /* USER CODE BEGIN TIM2_IRQn 1 */
213
214     /* USER CODE END TIM2_IRQn 1 */
215 }
216
217 /**
218  * @brief This function handles SPI1 global interrupt.
219  */
220 void SPI1_IRQHandler(void)
221 {
222     /* USER CODE BEGIN SPI1_IRQn 0 */
223
224     /* USER CODE END SPI1_IRQn 0 */
225
226     HAL_SPI_IRQHandler(&hspi1);
227     /* USER CODE BEGIN SPI1_IRQn 1 */
228
229     /* USER CODE END SPI1_IRQn 1 */
230 }
231
```



Dans cet exercice, nous allons l'utiliser pour implanter une petite machine à état logicielle qui reste dans l'état A pendant 10 seconde, passe ensuite dans l'état B, y reste 2 secondes, passe ensuite dans l'état C, y reste 5 secondes et revient finalement dans l'état A pour que tout recommence.

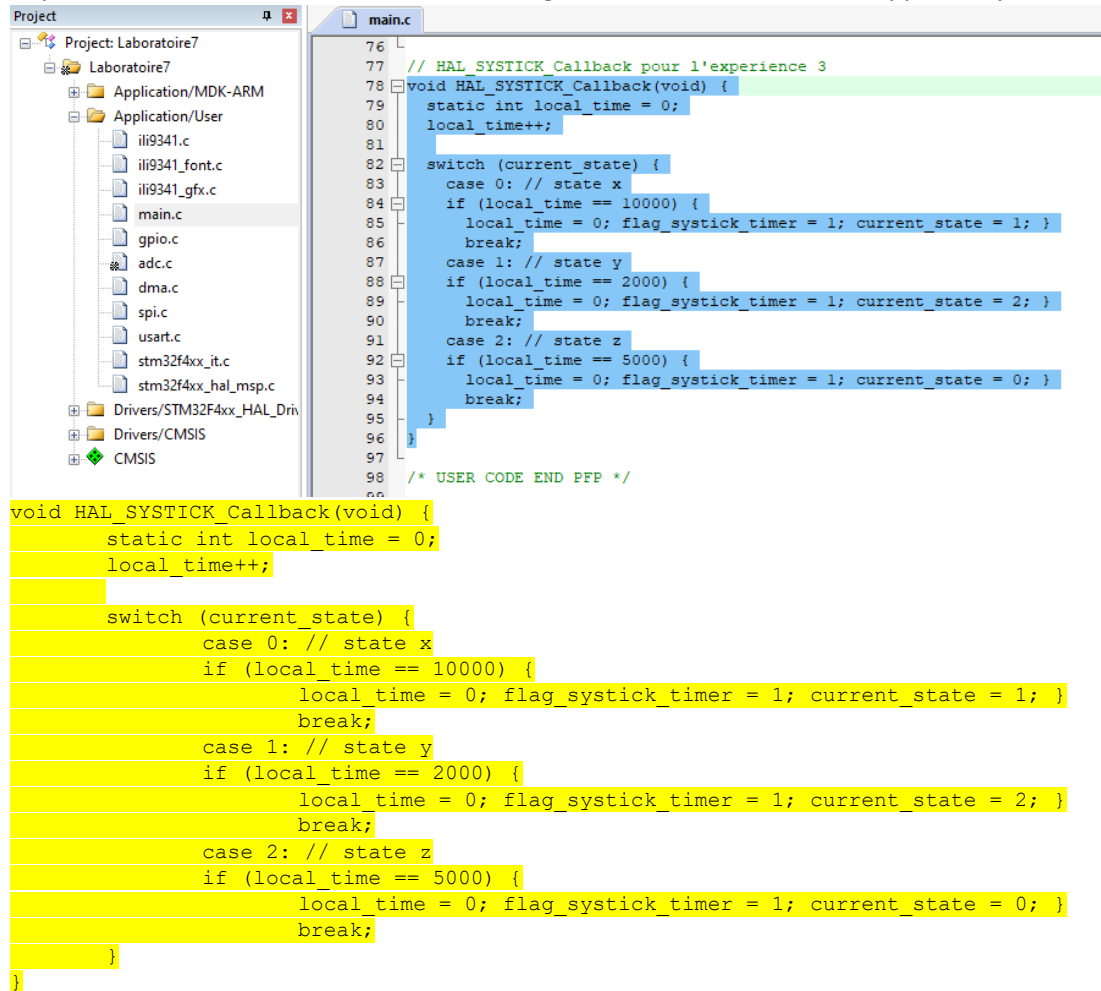
1. Commençons par définir un drapeau volatile `flag_systick_timer` et une variable d'état volatile `current_state` :



```
volatile int flag_systick_timer = 0;
```

```
volatile int current_state = 0;
```

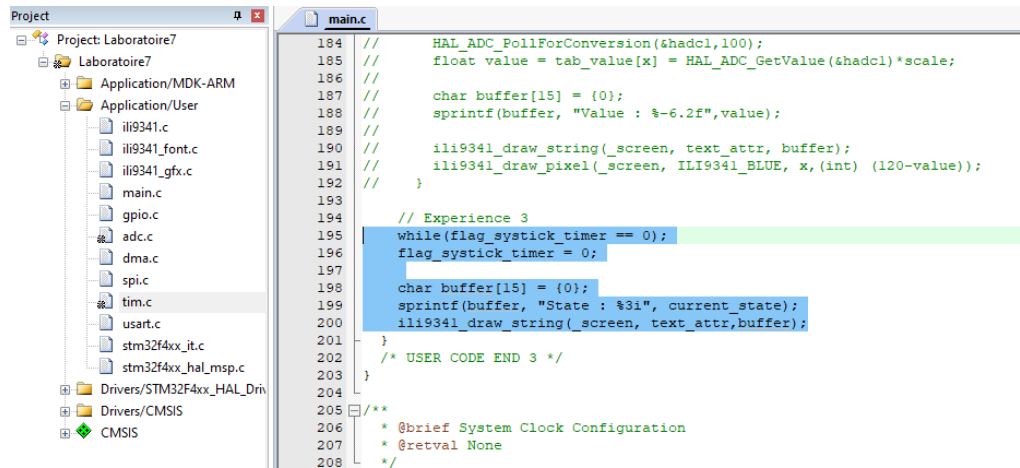
2. Implantons maintenant la machine à états logicielle dans la fonction de rappel du *Systick Timer* :



```
76 // HAL SYSTICK Callback pour l'experience 3
77 void HAL_SYSTICK_Callback(void) {
78     static int local_time = 0;
79     local_time++;
80
81     switch (current_state) {
82     case 0: // state x
83         if (local_time == 10000) {
84             local_time = 0; flag_systick_timer = 1; current_state = 1; }
85         break;
86     case 1: // state y
87         if (local_time == 2000) {
88             local_time = 0; flag_systick_timer = 1; current_state = 2; }
89         break;
90     case 2: // state z
91         if (local_time == 5000) {
92             local_time = 0; flag_systick_timer = 1; current_state = 0; }
93         break;
94     }
95 }
96
97 /* USER CODE END PFP */
98
99 void HAL_SYSTICK_Callback(void) {
100     static int local_time = 0;
101     local_time++;
102
103     switch (current_state) {
104     case 0: // state x
105         if (local_time == 10000) {
106             local_time = 0; flag_systick_timer = 1; current_state = 1; }
107         break;
108     case 1: // state y
109         if (local_time == 2000) {
110             local_time = 0; flag_systick_timer = 1; current_state = 2; }
111         break;
112     case 2: // state z
113         if (local_time == 5000) {
114             local_time = 0; flag_systick_timer = 1; current_state = 0; }
115         break;
116     }
117 }
```

3. Cette fonction sera donc appelée à chaque milliseconde. On commence par incrémenter le compteur de temps local (statique) `local_time`. En fonction de l'état, lorsque le temps arrive à échéance, on passe à l'état suivant. Il faut alors remettre le compteur de temps local à 0 et signaler l'événement au programme principal au moyen du drapeau `flag_systick_timer`.

4. Le programme principal est en attente constante d'un événement sur `flag_systick_timer`. Dès que cela arrive, il affiche l'état courant de la machine et remet le drapeau à 0 :



```
184 // HAL_ADC_PollForConversion(&hadcl,100);
185 // float value = tab_value[x] = HAL_ADC_GetValue(&hadcl)*scale;
186 //
187 // char buffer[15] = {0};
188 // sprintf(buffer, "Value : %-6.2f",value);
189 //
190 // ili9341_draw_string(_screen, text_attr, buffer);
191 // ili9341_draw_pixel(_screen, ILI9341_BLUE, x,(int) (120-value));
192 // }
193
194 // Experience 3
195 while(flag_systick_timer == 0);
196 flag_systick_timer = 0;
197
198 char buffer[15] = {0};
199 sprintf(buffer, "State : %3i", current_state);
200 ili9341_draw_string(_screen, text_attr,buffer);
201 }
202 /* USER CODE END 3 */
203 }
204
205 /**
206  * @brief System Clock Configuration
207  * @retval None
208  */
```

```
while(flag_systick_timer == 0);
flag_systick_timer = 0;

char buffer[15] = {0};
sprintf(buffer, "State : %3i",current_state);
ili9341_draw_string(_screen, text_attr, buffer);
```

TRES IMPORTANT

Lorsqu'on est dans une interruption, il faut réaliser le traitement le plus vite possible (et en tous cas avant qu'une nouvelle interruption arrive !!!). C'est pour cela qu'on utilise les drapeaux. La routine d'interruption fait un minimum de traitement. S'il y a des choses plus longues à réaliser (un accès au LCD par exemple ou bien un printf), on les fait dans le programme principal lorsqu'on a détecté un changement sur un drapeau.

Assurez-vous de bien comprendre le code en général et les interruptions en particulier.
Vous serez évalués en début de laboratoire.

2^{ème} partie : Laboratoire à réaliser en salle à Polytechnique

Réalisez un jeu de type « Angry Birds » dans lequel un œuf est lancé du coin inférieur gauche de l'écran pour atteindre un méchant cochon qui se cache derrière un mur à droite de l'écran.



Les spécifications techniques sont les suivantes :

- Dessinez le mur qui arrive à mi-hauteur de l'écran et se situe au milieu de sa longueur.
- Vous utiliserez la position du potentiomètre pour déterminer l'angle de tir (affichez la direction sur l'écran).
- Vous utiliserez le temps de pesée sur un bouton pour déterminer la vitesse initiale de l'œuf. Cela doit absolument se faire au moyen d'un machine à trois états (sur le Systick Timer à 1ms):
 REPOS (time = 0); //État initial (forcé en début de partie)
 COMPTAGE (time++); //On entre dans cet état dès qu'on pèse sur le bouton.
 FIN_DE_COMPTAGE; //On arrive dans cet état lorsqu'on relâche le bouton.
- Le cochon sera placé au hasard en (x,y) dans la partie droite de l'écran.
- L'œuf sera représenté par un cercle blanc plein de 10 points de rayon.
- Le cochon sera représenté par un cercle rose vide de 20 points de rayon (idéalement avec deux yeux à l'intérieur).
- La position courante de la balle est mémorisée dans deux variables (x et y).
- Utilisez les interruptions pour mettre à jour 100x par seconde la valeur de la position de la balle.
- La vitesse horizontale de la balle v_x est constante et égale à v_{x0} (x pixel chaque 10ms en fonction de la vitesse initiale). La vitesse verticale de la balle v_y varie pour former une parabole selon les lois de la physique. L'accélération verticale de la balle a_y est constante (la gravitation). Par approximation, on aura à chaque 10ms, $v_y = v_y + a_y$, avec a_y qui devra être ajusté pour avoir de belles paraboles (typiquement de l'ordre de 0.01 donc il faut garder y et v_y dans des variable float).
- Si l'œuf touche le mur, le jeu s'arrête.
- Si l'œuf sort de l'écran, le jeu s'arrête.
- Si l'œuf touche le cochon, c'est gagné ! À vous de l'indiquer au joueur à votre manière.

!!! Il vous est grandement conseillé de préparer cette partie AVANT le laboratoire !!!