VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Engineering

CC02 — Lab Report

Microprocessor - Microcontroller
Lab 4

Supervisors: Nguyen Thien An
Students: Nguyen Thanh Tai 2252722

Ho Chi Minh City, November 26, 2024

Contents

1 Schematic

The GitHub link for the lab schematics is at here or in this link: https://github.com/ThanhTaiNguyen24/mcu-mpu-lab1.

The schematic for the lab work is located here:

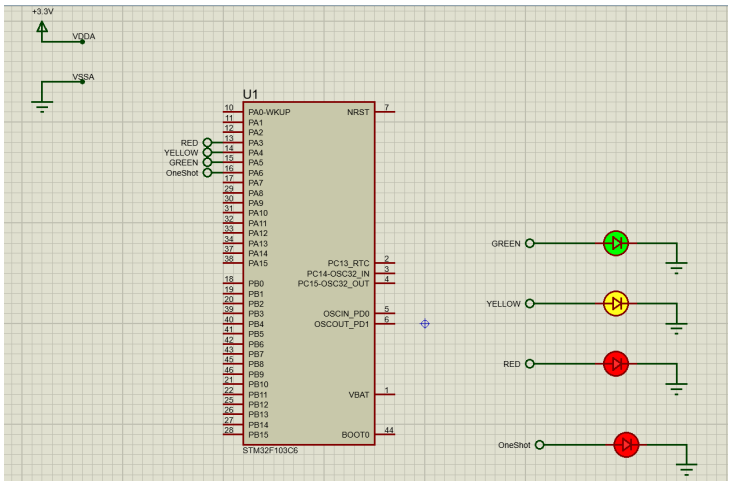

Figure 1: *The schematic for Lab 4.*

2 Problem

This is the source code for the lab work problem:

```
#include "scheduler.h"
#include "stdint.h"


typedef struct {
  void ( * pTask)(void);
  uint32_t Delay;
  uint32_t Period;
  uint8_t RunMe;
  uint32_t TaskID;
} sTask;

static sTask SCH_tasks_G[SCH_MAX_TASKS];

unsigned char Error_code_G = 0;

void SCH_Init(void){
    unsigned char i;
    for (i = 0; i < SCH_MAX_TASKS; i++) {
        SCH_Delete_Task(i);
    }
}

void SCH_Update(void) {
    unsigned char Index;
    for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
        if (SCH_tasks_G[Index].pTask) {
            if (SCH_tasks_G[Index].Delay == 0) {
                SCH_tasks_G[Index].RunMe += 1;
                if (SCH_tasks_G[Index].Period) {
                    SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].Period;
                }
            } else {
                SCH_tasks_G[Index].Delay -= 1;
            }
        }
    }
}

uint32_t SCH_Add_Task(void (* pFunction) (), uint32_t DELAY, uint32_t PERIOD) {
    unsigned char Index = 0;

    while ((SCH_tasks_G[Index].pTask != 0) && (Index < SCH_MAX_TASKS)) {
        Index++;
    }

```

```
47      SCH_tasks_G[Index].pTask = pFunction;
48      SCH_tasks_G[Index].Delay = DELAY;
49      SCH_tasks_G[Index].Period = PERIOD;
50      SCH_tasks_G[Index].RunMe = 0;
51      return Index;
52  }
53
54
55
56  void SCH_Delete_Task(uint32_t TASK_INDEX) {
57      if (SCH_tasks_G[TASK_INDEX].pTask != 0) {
58      SCH_tasks_G[TASK_INDEX].pTask = 0x0000;
59      SCH_tasks_G[TASK_INDEX].Delay = 0;
60      SCH_tasks_G[TASK_INDEX].Period = 0;
61      SCH_tasks_G[TASK_INDEX].RunMe = 0;
62    }
63  }
64
65
66  void SCH_Dispatch_Tasks(void) {
67      unsigned char Index;
68      for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
69          if (SCH_tasks_G[Index].RunMe > 0) {
70              (*SCH_tasks_G[Index].pTask)();
71              SCH_tasks_G[Index].RunMe -= 1;
72              if (SCH_tasks_G[Index].Period == 0) {
73                  SCH_Delete_Task(Index);
74              }
75          }
76      }
77  }
```

3 Demonstration

A project that has pin corresponding to the Proteus schematic with 10ms timer interrupt.

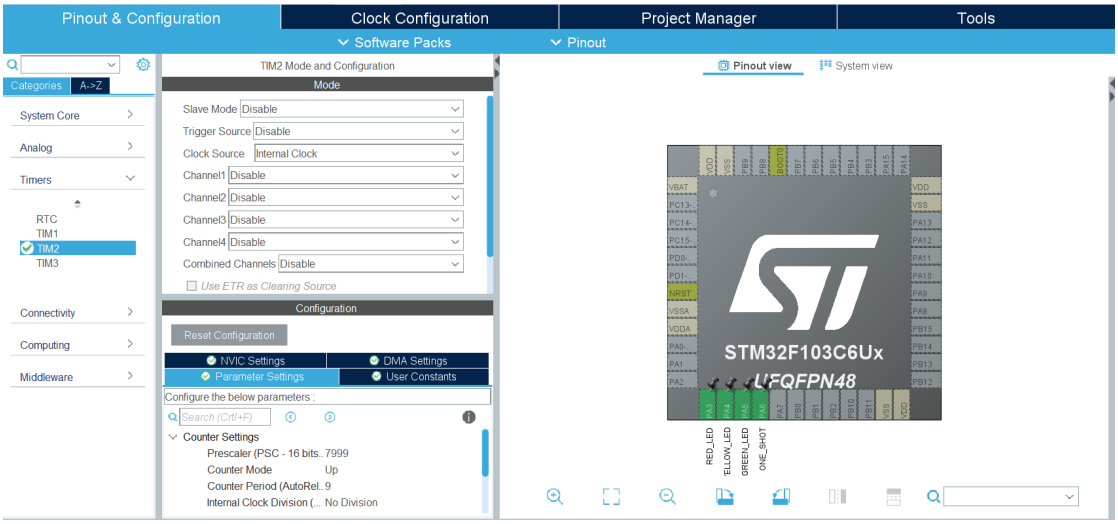

Figure 2: *Lab 4 project.*

The **SCH_Update** function is placed in the timer interrupt to ensure that tasks are scheduled and executed at the correct times based on their delays and periods

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    SCH_Update();
}
```

The code snippet demonstrates the addition of four tasks to the scheduler using the **SCH_Add_Task function**. Each task is associated with a specific function and has its own delay and period parameters. The **ledgreen** task is added with an initial delay of 0 and a period of 50 time units, meaning it will run immediately and then every 0.5 second. The **ledyellow** task is added with an initial delay of 100 and a period of 100, so it will start after 1 second and then run every 1 second. The **ledred** task has an initial delay of 200 time units and a period of 150, starting after 2 seconds and running every 1.5 seconds thereafter. Finally, the **ledoneshot** task is added with an initial delay of 300 and a period of 0, indicating it will run once after 3 seconds and not repeat (One Shot task). This setup allows for precise control over the timing and repetition of each task within the scheduler.

```
void ledred(){
    HAL_GPIO_TogglePin(RED_LED_GPIO_Port, RED_LED_Pin);
}
void ledyellow(){
    HAL_GPIO_TogglePin(YELLOW_LED_GPIO_Port, YELLOW_LED_Pin);
}
void ledgreen(){
    HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin);
```

```
}
void ledoneshot(){
    HAL_GPIO_TogglePin(ONE_SHOT_GPIO_Port, ONE_SHOT_Pin);
}

SCH_Add_Task(ledgreen, 0, 50);
SCH_Add_Task(ledyellow, 100, 100);
SCH_Add_Task(ledred, 200, 150);
SCH_Add_Task(ledoneshot, 300, 0);
```

Within this loop, the **SCH_Dispatch_Tasks()** function is called repeatedly to execute any tasks that are ready to run. This function checks each task in the scheduler's task array and runs those that have their RunMe flag set. After executing a task, the RunMe flag is decremented, and if the task is not periodic (i.e., one shot task), it is removed from the scheduler.

```
while (1)
{
    SCH_Dispatch_Tasks();
}
```

References