

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY  
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



## Microprocessor - Microcontroller

---

### Lab Report - CO3010 - CC02

# Lab 4

---

Advisor(s): Phan Văn Sỹ

Student(s): Lương Đức Hiếu 2352324

HO CHI MINH CITY, OCTOBER 2025



## Contents

<b>1</b>	<b>Overall</b>	<b>3</b>
<b>2</b>	<b>Proteus Schematic</b>	<b>3</b>
<b>3</b>	<b>Header file code</b>	<b>4</b>
3.1	main.h . . . . .	4
3.2	sch.h . . . . .	4
3.3	tasks.h . . . . .	5
<b>4</b>	<b>C source file code</b>	<b>6</b>
4.1	main.c . . . . .	6
4.2	sch.c . . . . .	8
4.3	tasks.c . . . . .	12

## 1 Overall

The GitHub link for the Lab 4 project is at : <https://github.com/hieuld1003/MPU-MCU>

## 2 Proteus Schematic

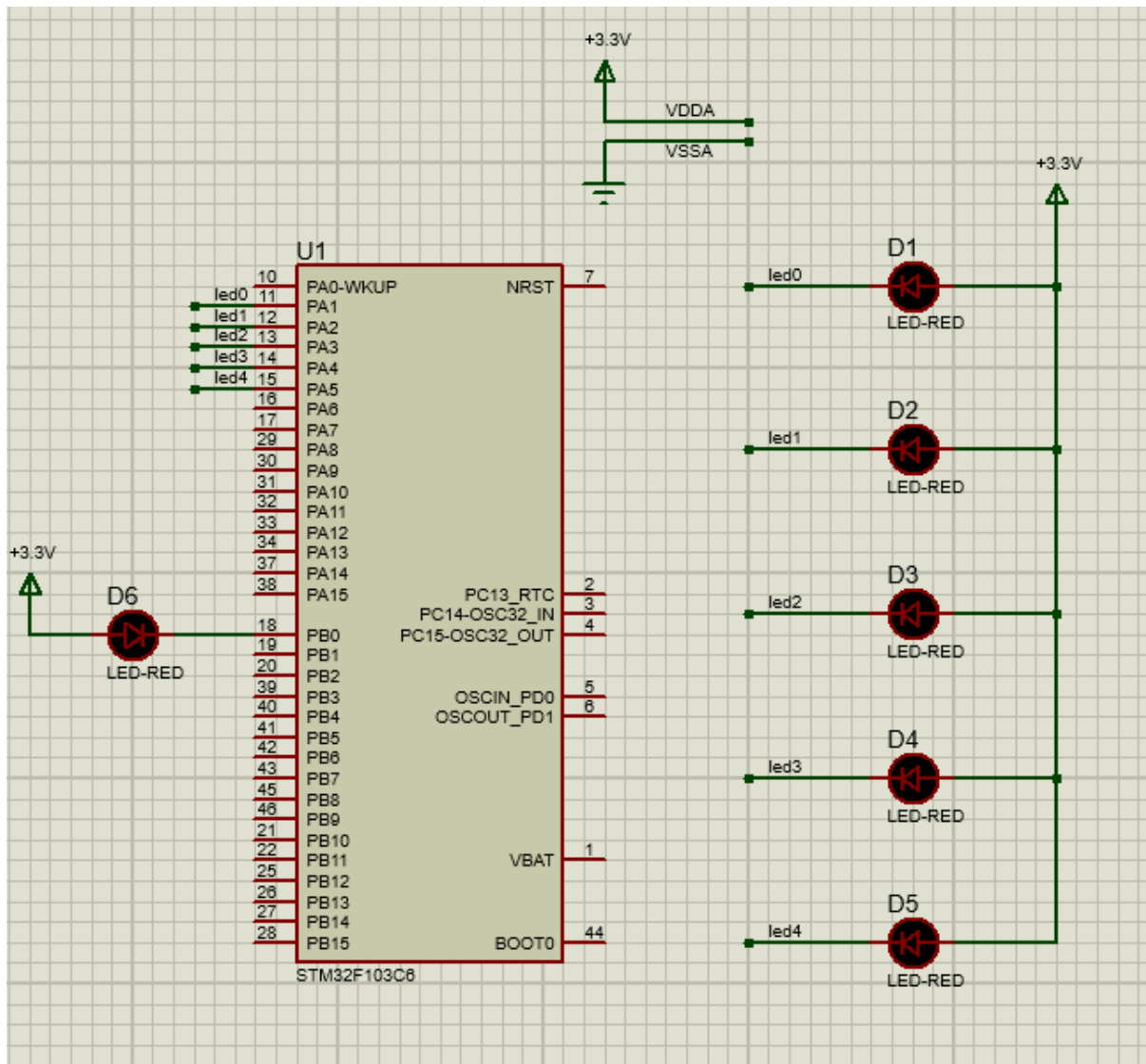


Figure 2.1: Lab4 schematic



## 3 Header file code

### 3.1 main.h

```
1 void Error_Handler(void);
2
3 /* USER CODE BEGIN EFP */
4
5 /* USER CODE END EFP */
6
7 /* Private defines
   -----*/
8 #define led0_Pin GPIO_PIN_1
9 #define led0_GPIO_Port GPIOA
10 #define led1_Pin GPIO_PIN_2
11 #define led1_GPIO_Port GPIOA
12 #define led2_Pin GPIO_PIN_3
13 #define led2_GPIO_Port GPIOA
14 #define led3_Pin GPIO_PIN_4
15 #define led3_GPIO_Port GPIOA
16 #define led4_Pin GPIO_PIN_5
17 #define led4_GPIO_Port GPIOA
18 #define debug_Pin GPIO_PIN_0
19 #define debug_GPIO_Port GPIOB
20 /* USER CODE BEGIN Private defines */
21
22 /* USER CODE END Private defines */
23
24 #ifndef __cplusplus
```

### 3.2 sch.h

```
1 #ifndef INC_SCH_H_
2 #define INC_SCH_H_
3
4 #include <stdint.h>
5
6 #define SCH_MAX_TASKS 40
```



```
7 #define NO_TASK 0xFF
8
9 typedef struct {
10     void (*pTask)(void);
11     uint32_t Period;
12     uint8_t RunMe;
13     uint32_t TaskID;
14     uint32_t Next_Run_Time;
15     uint8_t Next_Index;
16 } sTask;
17
18 void SCH_Init(void);
19 void SCH_Update(void);
20 void SCH_Insert_Task_Sorted(uint8_t task_index);
21 void SCH_Dispatch_Tasks(void);
22 uint32_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY,
23     uint32_t PERIOD);
24 uint8_t SCH_Delete_Task(uint32_t taskID);
25 uint32_t SCH_Get_Time(void);
26 #endif /* INC_SCH_H_ */
```

### 3.3 tasks.h

```
1 #ifndef INC_TASKS_H_
2 #define INC_TASKS_H_
3
4 void Task1(void);
5 void Task2(void);
6 void Task3(void);
7 void Task4(void);
8 void Task5(void);
9
10 void Timer_10ms(void);
11
12 #endif /* INC_TASKS_H_ */
```

## 4 C source file code

### 4.1 main.c

```
1 #include "main.h"
2 #include "sch.h"
3 #include "tasks.h"
4
5 /* Private includes
   ----- */
6 /* USER CODE BEGIN Includes */
7
8 /* USER CODE END Includes */
9
10 /* Private typedef
   ----- */
11 /* USER CODE BEGIN PTD */
12
13 /* USER CODE END PTD */
14
15 /* Private define
   ----- */
16 /* USER CODE BEGIN PD */
17 /* USER CODE END PD */
18
19 /* Private macro
   -----
   */
20 /* USER CODE BEGIN PM */
21
22 /* USER CODE END PM */
23
24 /* Private variables
   ----- */
25 TIM_HandleTypeDef htim2;
26
27 /* USER CODE BEGIN PV */
28
```



```
29 /* USER CODE END PV */
30
31 /* Private function prototypes
   -----*/
32 void SystemClock_Config(void);
33 static void MX_GPIO_Init(void);
34 static void MX_TIM2_Init(void);
35 /* USER CODE BEGIN PFP */
36
37 /* USER CODE END PFP */
38
39 /* Private user code
   -----*/
40 /* USER CODE BEGIN 0 */
41
42 /* USER CODE END 0 */
43 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
44     if (htim->Instance == TIM2) {
45         SCH_Update();
46     }
47 }
48 /**
49  * @brief The application entry point.
50  * @retval int
51  */
52 int main(void)
53 {
54     HAL_Init();
55     SystemClock_Config();
56     MX_GPIO_Init();
57     MX_TIM2_Init();
58     HAL_TIM_Base_Start_IT(&htim2);
59
60     SCH_Init();
61
62     SCH_Add_Task(Timer_10ms, 0, 10);
63     SCH_Add_Task(Task1, 0, 50);
64     SCH_Add_Task(Task2, 0, 100);
```

```
65 SCH_Add_Task(Task3, 0, 150);
66 SCH_Add_Task(Task4, 0, 200);
67 SCH_Add_Task(Task5, 0, 250);
68
69 while (1)
70 {
71     SCH_Dispatch_Tasks();
72 }
73 /* USER CODE END 3 */
74 }
```

## 4.2 sch.c

```
1 #include "sch.h"
2
3 static sTask SCH_tasks_G[SCH_MAX_TASKS];
4 static uint32_t current_time_ticks = 0;
5 static uint32_t next_task_id = 0;
6 static uint8_t head_index = NO_TASK;
7
8 void SCH_Init(void) {
9     uint8_t i;
10    for (i = 0; i < SCH_MAX_TASKS; i++) {
11        SCH_tasks_G[i].pTask = 0;
12        SCH_tasks_G[i].Next_Run_Time = 0;
13        SCH_tasks_G[i].Period = 0;
14        SCH_tasks_G[i].RunMe = 0;
15        SCH_tasks_G[i].TaskID = 0;
16        SCH_tasks_G[i].Next_Index = NO_TASK;
17    }
18    current_time_ticks = 0;
19    next_task_id = 1;
20    head_index = NO_TASK;
21 }
22
23 void SCH_Update(void) {
24     current_time_ticks++;
```



```
25
26 while (head_index != NO_TASK &&
27         current_time_ticks >= SCH_tasks_G[head_index].
28         Next_Run_Time) {
29
30     SCH_tasks_G[head_index].RunMe += 1;
31
32     if (SCH_tasks_G[head_index].Period > 0) {
33         SCH_tasks_G[head_index].Next_Run_Time =
34             current_time_ticks + SCH_tasks_G[head_index].
35             Period;
36
37         uint8_t current_task_index = head_index;
38         head_index = SCH_tasks_G[head_index].Next_Index;
39
40         SCH_Insert_Task_Sorted(current_task_index);
41     } else {
42         head_index = SCH_tasks_G[head_index].Next_Index;
43     }
44 }
45
46 void SCH_Insert_Task_Sorted(uint8_t task_index) {
47     if (head_index == NO_TASK ||
48         SCH_tasks_G[task_index].Next_Run_Time < SCH_tasks_G[
49             head_index].Next_Run_Time) {
50         SCH_tasks_G[task_index].Next_Index = head_index;
51         head_index = task_index;
52         return;
53     }
54
55     uint8_t current = head_index;
56     while (SCH_tasks_G[current].Next_Index != NO_TASK) {
57         uint8_t next = SCH_tasks_G[current].Next_Index;
58         if (SCH_tasks_G[task_index].Next_Run_Time < SCH_tasks_G[
59             next].Next_Run_Time) {
60             SCH_tasks_G[task_index].Next_Index = next;
61             SCH_tasks_G[current].Next_Index = task_index;
```

```
59         return;
60     }
61     current = next;
62 }
63
64 SCH_tasks_G[current].Next_Index = task_index;
65 SCH_tasks_G[task_index].Next_Index = NO_TASK;
66 }
67
68
69 void SCH_Dispatch_Tasks(void) {
70     uint8_t Index;
71
72     for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
73         if (SCH_tasks_G[Index].pTask && SCH_tasks_G[Index].RunMe
74             > 0) {
75             SCH_tasks_G[Index].RunMe -= 1;
76             (*SCH_tasks_G[Index].pTask)();
77
78             if (SCH_tasks_G[Index].Period == 0) {
79                 SCH_Delete_Task(SCH_tasks_G[Index].TaskID);
80             }
81         }
82     }
83
84     uint32_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY,
85         uint32_t PERIOD) {
86         uint8_t Index = 0;
87
88         while ((SCH_tasks_G[Index].pTask != 0) && (Index <
89             SCH_MAX_TASKS)) {
90             Index++;
91         }
92
93         if (Index == SCH_MAX_TASKS) {
94             return SCH_MAX_TASKS;
95         }
96     }
97 }
```



```
94
95     SCH_tasks_G[Index].pTask = pFunction;
96     SCH_tasks_G[Index].Next_Run_Time = current_time_ticks + DELAY
97     ;
98     SCH_tasks_G[Index].Period = PERIOD;
99     SCH_tasks_G[Index].RunMe = 0;
100    SCH_tasks_G[Index].TaskID = next_task_id;
101    SCH_tasks_G[Index].Next_Index = NO_TASK;
102
103    SCH_Insert_Task_Sorted(Index);
104
105    return next_task_id++;
106}
107
108uint8_t SCH_Delete_Task(uint32_t taskID) {
109    uint8_t Index;
110
111    for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
112        if (SCH_tasks_G[Index].TaskID == taskID) {
113            if (head_index == Index) {
114                head_index = SCH_tasks_G[Index].Next_Index;
115            } else {
116                uint8_t current = head_index;
117                while (current != NO_TASK && SCH_tasks_G[current
118                ].Next_Index != Index) {
119                    current = SCH_tasks_G[current].Next_Index;
120                }
121                if (current != NO_TASK) {
122                    SCH_tasks_G[current].Next_Index = SCH_tasks_G
123                    [Index].Next_Index;
124                }
125            }
126        }
127
128        SCH_tasks_G[Index].pTask = 0;
129        SCH_tasks_G[Index].Next_Run_Time = 0;
130        SCH_tasks_G[Index].Period = 0;
131        SCH_tasks_G[Index].RunMe = 0;
132        SCH_tasks_G[Index].TaskID = 0;
```

```
129         SCH_tasks_G[Index].Next_Index = NO_TASK;
130         return 1;
131     }
132 }
133 return 0;
134 }
135
136 uint32_t SCH_Get_Time(void) {
137     return current_time_ticks * 10;
138 }
```

### 4.3 tasks.c

```
1 #include "tasks.h"
2 #include "main.h"
3 #include "sch.h"
4
5 void Task1(void) {
6     HAL_GPIO_TogglePin(led0_GPIO_Port, led0_Pin);
7 }
8
9 void Task2(void) {
10     HAL_GPIO_TogglePin(led1_GPIO_Port, led1_Pin);
11 }
12
13 void Task3(void) {
14     HAL_GPIO_TogglePin(led2_GPIO_Port, led2_Pin);
15 }
16
17 void Task4(void) {
18     HAL_GPIO_TogglePin(led3_GPIO_Port, led3_Pin);
19 }
20
21 void Task5(void) {
22     HAL_GPIO_TogglePin(led4_GPIO_Port, led4_Pin);
23 }
24
```



```
25 void Timer_10ms(void) {  
26     HAL_GPIO_TogglePin(debug_GPIO_Port, debug_Pin);  
27 }
```