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

1 Overall

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

2 Proteus Schematic

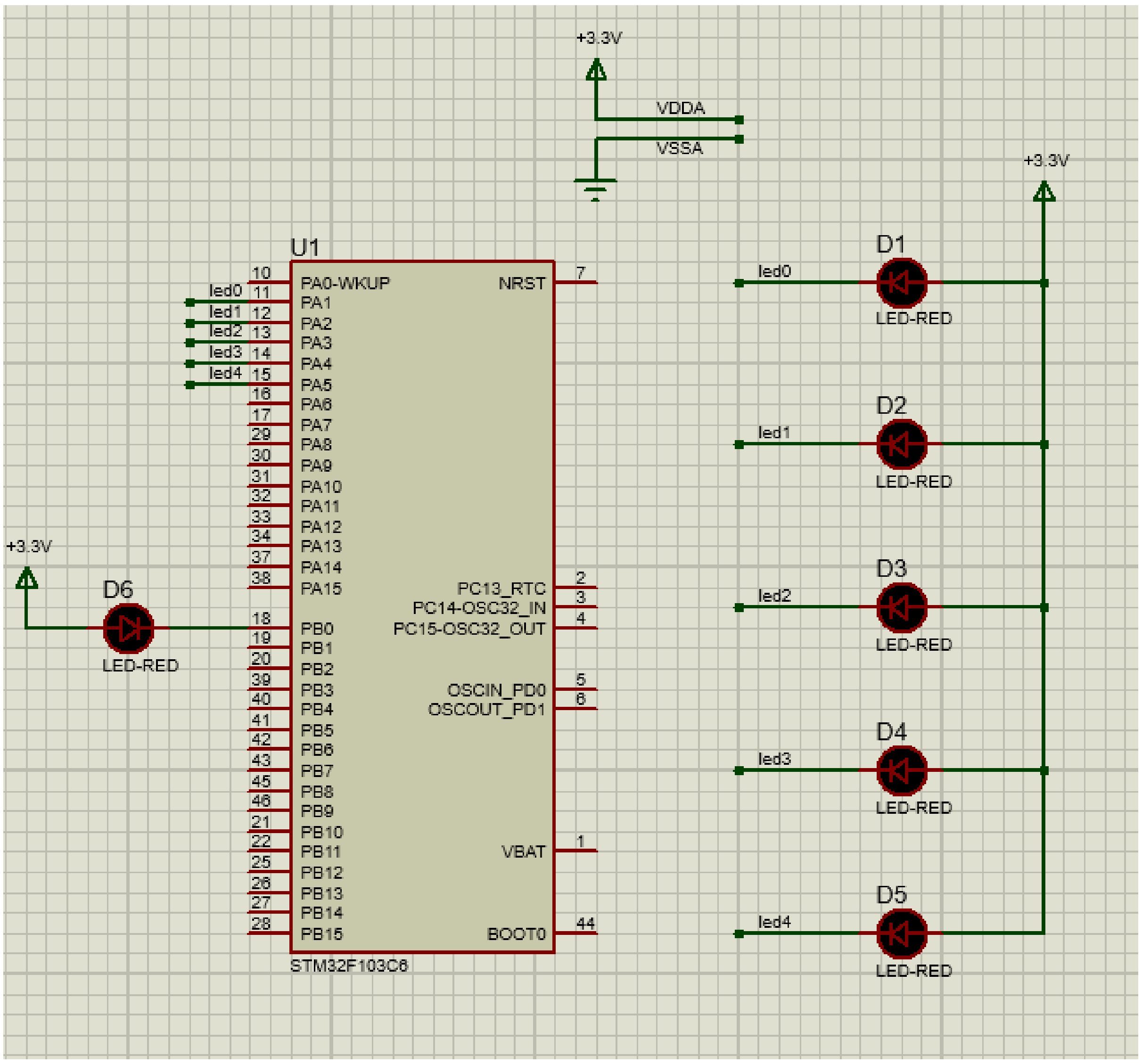

Figure 2.1: Lab4 schematic

3 Header file code

3.1 main.h

```c
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines
   -----------------------------------------------------------*/
#define led0_Pin GPIO_PIN_1
#define led0_GPIO_Port GPIOA
#define led1_Pin GPIO_PIN_2
#define led1_GPIO_Port GPIOA
#define led2_Pin GPIO_PIN_3
#define led2_GPIO_Port GPIOA
#define led3_Pin GPIO_PIN_4
#define led3_GPIO_Port GPIOA
#define led4_Pin GPIO_PIN_5
#define led4_GPIO_Port GPIOA
#define debug_Pin GPIO_PIN_0
#define debug_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
```

3.2 sch.h

```c
#ifndef INC_SCH_H_
#define INC_SCH_H_

#include <stdint.h>

#define SCH_MAX_TASKS 40
```

```
#define NO_TASK 0xFF

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

void SCH_Init(void);
void SCH_Update(void);
void SCH_Insert_Task_Sorted(uint8_t task_index);
void SCH_Dispatch_Tasks(void);
uint32_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY,
   uint32_t PERIOD);
uint8_t SCH_Delete_Task(uint32_t taskID);
uint32_t SCH_Get_Time(void);

#endif /* INC_SCH_H_ */
```

3.3 tasks.h

```
#ifndef INC_TASKS_H_
#define INC_TASKS_H_

void Task1(void);
void Task2(void);
void Task3(void);
void Task4(void);
void Task5(void);

void Timer_10ms(void);

#endif /* INC_TASKS_H_ */
```

4 C source file code

4.1 main.c

```c
#include "main.h"
#include "sch.h"
#include "tasks.h"

/* Private includes
    ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef
    -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
    ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
    -------------------------------------------------------------
    */
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
    ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM2) {
        SCH_Update();
    }
}
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  HAL_TIM_Base_Start_IT(&htim2);

  SCH_Init();

  SCH_Add_Task(Timer_10ms, 0, 10);
  SCH_Add_Task(Task1, 0, 50);
  SCH_Add_Task(Task2, 0, 100);
  SCH_Add_Task(Task3, 0, 150);
  SCH_Add_Task(Task4, 0, 200);
  SCH_Add_Task(Task5, 0, 250);

  while (1)
  {
    SCH_Dispatch_Tasks();
  }
  /* USER CODE END 3 */
}
```

4.2 sch.c

```
#include "sch.h"

static sTask SCH_tasks_G[SCH_MAX_TASKS];
static uint32_t current_time_ticks = 0;
static uint32_t next_task_id = 0;
static uint8_t head_index = NO_TASK;

void SCH_Init(void) {
    uint8_t i;
    for (i = 0; i < SCH_MAX_TASKS; i++) {
        SCH_tasks_G[i].pTask = 0;
        SCH_tasks_G[i].Next_Run_Time = 0;
        SCH_tasks_G[i].Period = 0;
        SCH_tasks_G[i].RunMe = 0;
        SCH_tasks_G[i].TaskID = 0;
        SCH_tasks_G[i].Next_Index = NO_TASK;
    }
    current_time_ticks = 0;
    next_task_id = 1;
    head_index = NO_TASK;
}

void SCH_Update(void) {
    current_time_ticks++;

    while (head_index != NO_TASK &&
             current_time_ticks >= SCH_tasks_G[head_index].
                Next_Run_Time) {

        SCH_tasks_G[head_index].RunMe += 1;

        if (SCH_tasks_G[head_index].Period > 0) {
            SCH_tasks_G[head_index].Next_Run_Time =
                current_time_ticks + SCH_tasks_G[head_index].
                   Period;

            uint8_t current_task_index = head_index;
            head_index = SCH_tasks_G[head_index].Next_Index;

            SCH_Insert_Task_Sorted(current_task_index);
        } else {
            head_index = SCH_tasks_G[head_index].Next_Index;
        }
    }
}

void SCH_Insert_Task_Sorted(uint8_t task_index) {
    if (head_index == NO_TASK ||
        SCH_tasks_G[task_index].Next_Run_Time < SCH_tasks_G[
           head_index].Next_Run_Time) {
        SCH_tasks_G[task_index].Next_Index = head_index;
        head_index = task_index;
        return;
    }

    uint8_t current = head_index;
    while (SCH_tasks_G[current].Next_Index != NO_TASK) {
        uint8_t next = SCH_tasks_G[current].Next_Index;
        if (SCH_tasks_G[task_index].Next_Run_Time < SCH_tasks_G[
           next].Next_Run_Time) {
            SCH_tasks_G[task_index].Next_Index = next;
            SCH_tasks_G[current].Next_Index = task_index;
```

```
                return;
            }
            current = next;
        }

        SCH_tasks_G[current].Next_Index = task_index;
        SCH_tasks_G[task_index].Next_Index = NO_TASK;
}


void SCH_Dispatch_Tasks(void) {
        uint8_t Index;

        for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
            if (SCH_tasks_G[Index].pTask && SCH_tasks_G[Index].RunMe
                > 0) {
                SCH_tasks_G[Index].RunMe -= 1;
                (*SCH_tasks_G[Index].pTask)();

                if (SCH_tasks_G[Index].Period == 0) {
                    SCH_Delete_Task(SCH_tasks_G[Index].TaskID);
                }
            }
        }
}

uint32_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY,
    uint32_t PERIOD) {
        uint8_t Index = 0;

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

        if (Index == SCH_MAX_TASKS) {
            return SCH_MAX_TASKS;
        }

        SCH_tasks_G[Index].pTask = pFunction;
        SCH_tasks_G[Index].Next_Run_Time = current_time_ticks + DELAY
           ;
        SCH_tasks_G[Index].Period = PERIOD;
        SCH_tasks_G[Index].RunMe = 0;
        SCH_tasks_G[Index].TaskID = next_task_id;
        SCH_tasks_G[Index].Next_Index = NO_TASK;

        SCH_Insert_Task_Sorted(Index);

        return next_task_id++;
}

uint8_t SCH_Delete_Task(uint32_t taskID) {
        uint8_t Index;

        for (Index = 0; Index < SCH_MAX_TASKS; Index++) {
            if (SCH_tasks_G[Index].TaskID == taskID) {
                if (head_index == Index) {
                    head_index = SCH_tasks_G[Index].Next_Index;
                } else {
                    uint8_t current = head_index;
                    while (current != NO_TASK && SCH_tasks_G[current
                       ].Next_Index != Index) {
                        current = SCH_tasks_G[current].Next_Index;
                    }
                    if (current != NO_TASK) {
                        SCH_tasks_G[current].Next_Index = SCH_tasks_G
                           [Index].Next_Index;
                    }
                }

                SCH_tasks_G[Index].pTask = 0;
                SCH_tasks_G[Index].Next_Run_Time = 0;
                SCH_tasks_G[Index].Period = 0;
                SCH_tasks_G[Index].RunMe = 0;
                SCH_tasks_G[Index].TaskID = 0;
```

```c
                SCH_tasks_G[Index].Next_Index = NO_TASK;
                return 1;
            }
        }
        return 0;
}

uint32_t SCH_Get_Time(void) {
    return current_time_ticks * 10;
}
```

4.3 tasks.c

```c
#include "tasks.h"
#include "main.h"
#include "sch.h"

void Task1(void) {
    HAL_GPIO_TogglePin(led0_GPIO_Port, led0_Pin);
}

void Task2(void) {
    HAL_GPIO_TogglePin(led1_GPIO_Port, led1_Pin);
}

void Task3(void) {
    HAL_GPIO_TogglePin(led2_GPIO_Port, led2_Pin);
}

void Task4(void) {
    HAL_GPIO_TogglePin(led3_GPIO_Port, led3_Pin);
}

void Task5(void) {
    HAL_GPIO_TogglePin(led4_GPIO_Port, led4_Pin);
}

```

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