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

Lab 4

Advisor(s): Phan Văn Sỹ
Student(s): Chu Le Hoang 2352346

HO CHI MINH CITY, OCTOBER 2025

Contents

1 Overall **4**

2 Problem **4**

3 Implementation **4**

3.1 Problem 5: Demonstration 4

4 Proteus Schematic **5**

5 Source code **5**

5.1 sch.h (Scheduler Header) 5

5.2 sch.c (Scheduler Implementation) 7

5.3 tasks.c (Task Implementations) 10

5.4 main.c (Main Program) 11

6 Summary **13**

1 Overall

Lab schematics and the source codes are submitted via GitHub link: https://github.com/KennyLe298/MPU-MCU

2 Problem

The goal of this lab was to design and implement a cooperative scheduler capable of running multiple tasks at different, accurate time intervals.

3 Implementation

The core requirement was to implement a scheduler with the following:

- `void SCH_Init(void)`: Initializes the scheduler.
- `uint32_t SCH_Add_Task(..., uint32_t DELAY, uint32_t PERIOD)`: Adds a function to the task queue.
- `uint8_t SCH_Delete_Task(uint32_t taskID)`: Removes a task from the queue.
- `void SCH_Update(void)`: Called by a timer ISR to update task delays.
- `void SCH_Dispatch_Tasks(void)`: Runs tasks that are due.

This system must run 5 periodic tasks to toggle 5 different LEDs at intervals of 0.5s, 1s, 1.5s, 2s, and 2.5s.

3.1 Problem 5: Demonstration

The demonstration must show:

- A regular 10ms timer tick (100Hz).
- A periodic task running every 10ms.
- Printing a timestamp value (from a `SCH_Get_Time()` function) to prove accuracy.
- Concurrent tasks running at different intervals.
- One-shot tasks (PERIOD = 0) with an initial delay.

4 Proteus Schematic

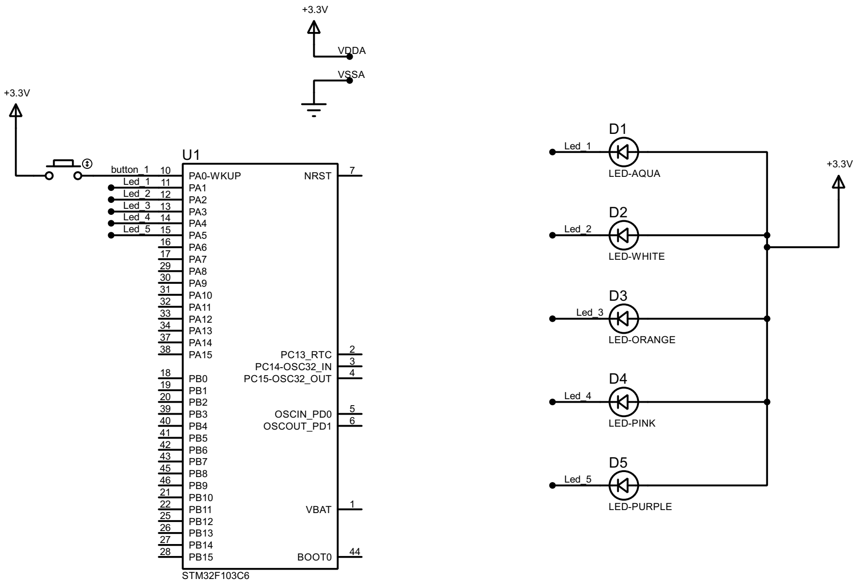

Figure 4.1: Lab 4 schematic

5 Source code

The project is divided into several files. The core logic for the scheduler is in `sch.c` and `sch.h`.

5.1 sch.h (Scheduler Header)

This file defines the task structure `sTask` and the error codes, as well as the function prototypes for the scheduler.

```
#define SCH_MAX_TASKS 10
#define ERROR_SCH_INVALID_INDEX               101
#define ERROR_SCH_TOO_MANY_TASKS              102
```

```c
#define ERROR_SCH_WAITING_FOR_SLAVE_TO_ACK          103
#define ERROR_SCH_WAITING_FOR_START_COMMAND_FROM_MASTER   104
#define ERROR_SCH_ONE_OR_MORE_SLAVES_DID_NOT_START      105
#define ERROR_SCH_LOST_SLAVE                106
#define ERROR_SCH_CAN_BUS_ERROR               107
#define ERROR_I2C_WRITE_BYTE_AT24C64              108
#define ERROR_SCH_NULL_POINTER                109
typedef struct
{
  void (*pTask)(void); // pointer to the task function
  uint32_t Delay;      // delay in ticks before task run
  uint32_t Period;     // interval between runs
  uint8_t RunMe;
  uint32_t TaskID;    // task id
} sTask;
void SCH_Init(void);
uint8_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY,
   uint32_t PERIOD);
uint8_t SCH_Delete_Task(uint32_t TaskID);
void SCH_Update(void);
void SCH_Dispatch_Tasks(void);
void SCH_Go_To_Sleep(void);
uint8_t SCH_Delete_All_Tasks(void);
void SystemClock_Config(void);
uint32_t SCH_Get_Current_Tasks(void);
uint32_t SCH_Get_Error_Code(void);
uint32_t SCH_Get_Time(void);
```

5.2 sch.c (Scheduler Implementation)

This file contains the logic for the scheduler.The `SCH_tasks_G` array holds all the tasks.

```c
volatile sTask SCH_tasks_G[SCH_MAX_TASKS];
static uint32_t currentTasks = 0;
static uint32_t ERROR_CODE_G = 0;
```

```
static volatile uint32_t timestamp = 0;
void SCH_Init(void)
{
  SCH_Delete_All_Tasks();
  currentTasks = 0;
  timestamp = 0;
  HAL_GPIO_WritePin(Led_1_GPIO_Port, Led_1_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(Led_2_GPIO_Port, Led_2_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(Led_3_GPIO_Port, Led_3_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(Led_4_GPIO_Port, Led_4_Pin,
   GPIO_PIN_RESET);
  HAL_GPIO_WritePin(Led_5_GPIO_Port, Led_5_Pin,
   GPIO_PIN_RESET);
}
uint8_t SCH_Add_Task(void (*pFunction)(), uint32_t DELAY,
   uint32_t PERIOD)
{
  uint8_t index = 0;
  if (pFunction == 0) {
    ERROR_CODE_G = ERROR_SCH_NULL_POINTER;
    return ERROR_CODE_G;
  }
  if (currentTasks >= SCH_MAX_TASKS) { //check number of
   tasks
    ERROR_CODE_G = ERROR_SCH_TOO_MANY_TASKS;
    return ERROR_CODE_G;
  }
  while ((SCH_tasks_G[index].pTask != 0) && (index <
   SCH_MAX_TASKS)) {
    index++;
  } //check for available slot
  if (index == SCH_MAX_TASKS) {
```

```
    ERROR_CODE_G = ERROR_SCH_TOO_MANY_TASKS;
    return ERROR_CODE_G;
  }
  SCH_tasks_G[index].pTask = pFunction;
  SCH_tasks_G[index].Delay = DELAY;
  SCH_tasks_G[index].Period = PERIOD;
  SCH_tasks_G[index].RunMe = 0;
  SCH_tasks_G[index].TaskID = index;
  currentTasks++;
  return index;
}
uint8_t SCH_Delete_Task(uint32_t TaskID)
{
  if (TaskID >= SCH_MAX_TASKS) {
    ERROR_CODE_G = ERROR_SCH_INVALID_INDEX;
    return ERROR_CODE_G;
  }
  if (SCH_tasks_G[TaskID].pTask == 0) {
    ERROR_CODE_G = ERROR_SCH_INVALID_INDEX;
    return ERROR_CODE_G;
  }
  SCH_tasks_G[TaskID].pTask = 0;
  SCH_tasks_G[TaskID].Delay = 0;
  SCH_tasks_G[TaskID].Period = 0;
  SCH_tasks_G[TaskID].RunMe = 0;
  currentTasks--;
  return 0;
}
uint8_t SCH_Delete_All_Tasks(void)
{
  uint8_t returnCode = 0;
  if (currentTasks == 0) {
    ERROR_CODE_G = ERROR_SCH_INVALID_INDEX;
    returnCode = ERROR_SCH_INVALID_INDEX; //error when no
 task to delete
```

```
  }
  //reset everything
  for (uint8_t i = 0; i < SCH_MAX_TASKS; i++) {
    SCH_tasks_G[i].pTask = 0;
    SCH_tasks_G[i].Delay = 0;
    SCH_tasks_G[i].Period = 0;
    SCH_tasks_G[i].RunMe = 0;
    SCH_tasks_G[i].TaskID = i;
  }
  currentTasks = 0;
  return returnCode;
}
void SCH_Update(void)
{
  timestamp++;
  for (uint8_t Index = 0; Index < SCH_MAX_TASKS; Index++) {
    if (SCH_tasks_G[Index].pTask != 0) {
      if (SCH_tasks_G[Index].Delay > 0) {
        SCH_tasks_G[Index].Delay--;
      } else {
        SCH_tasks_G[Index].RunMe++;
        if (SCH_tasks_G[Index].Period > 0) {
          SCH_tasks_G[Index].Delay = SCH_tasks_G[Index].
   Period;
        }
      }
    }
  }
}

void SCH_Dispatch_Tasks(void)
{
  for (uint8_t Index = 0; Index < SCH_MAX_TASKS; Index++) {
    if ((SCH_tasks_G[Index].RunMe > 0) && (SCH_tasks_G[Index
   ].pTask != 0)) {
```

```
      SCH_tasks_G[Index].RunMe--;
      (*SCH_tasks_G[Index].pTask)();
      // delete one_shot
      if (SCH_tasks_G[Index].Period == 0) {
        SCH_Delete_Task(Index);
      }
    }
  }
  // scheduler into idle mode
  SCH_Go_To_Sleep();
}
void SCH_Go_To_Sleep(void)
{
  HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON,
   PWR_SLEEPENTRY_WFI);
}
uint32_t SCH_Get_Current_Tasks(void)
{
  return currentTasks;
}
uint32_t SCH_Get_Error_Code(void)
{
  return ERROR_CODE_G;
}
uint32_t SCH_Get_Time(void) {
  return timestamp;
}
```

5.3 tasks.c (Task Implementations)

This file defines the functions that are executed by the scheduler.

```
void LED_Task_500ms(void){
  HAL_GPIO_TogglePin(Led_1_GPIO_Port, Led_1_Pin);
}
void LED_Task_1000ms(void){
```

```
  HAL_GPIO_TogglePin(Led_2_GPIO_Port, Led_2_Pin);
}

void LED_Task_1500ms(void){
  HAL_GPIO_TogglePin(Led_3_GPIO_Port, Led_3_Pin);
}
void LED_Task_2000ms(void){
  HAL_GPIO_TogglePin(Led_4_GPIO_Port, Led_4_Pin);
}
void LED_Task_2500ms(void){
  HAL_GPIO_TogglePin(Led_5_GPIO_Port, Led_5_Pin);
}
void OneShot_Task(void) {
  //turn all LEDs off
  HAL_GPIO_WritePin(Led_1_GPIO_Port, Led_1_Pin, GPIO_PIN_SET)
   ;
  HAL_GPIO_WritePin(Led_2_GPIO_Port, Led_2_Pin, GPIO_PIN_SET)
   ;
  HAL_GPIO_WritePin(Led_3_GPIO_Port, Led_3_Pin, GPIO_PIN_SET)
   ;
  HAL_GPIO_WritePin(Led_4_GPIO_Port, Led_4_Pin, GPIO_PIN_SET)
   ;
  HAL_GPIO_WritePin(Led_5_GPIO_Port, Led_5_Pin, GPIO_PIN_SET)
   ;
  HAL_Delay(300);
  //turn on in order
  HAL_GPIO_WritePin(Led_1_GPIO_Port, Led_1_Pin,
   GPIO_PIN_RESET);
  HAL_Delay(200);
  HAL_GPIO_WritePin(Led_2_GPIO_Port, Led_2_Pin,
   GPIO_PIN_RESET);
  HAL_Delay(200);
  HAL_GPIO_WritePin(Led_3_GPIO_Port, Led_3_Pin,
   GPIO_PIN_RESET);
  HAL_Delay(200);
```

```
  HAL_GPIO_WritePin(Led_4_GPIO_Port, Led_4_Pin, GPIO_PIN_RESET);
  HAL_Delay(200);
  HAL_GPIO_WritePin(Led_5_GPIO_Port, Led_5_Pin, GPIO_PIN_RESET);
  HAL_Delay(500);
}
void Print_Timestamp(void) {
  uint32_t time = SCH_Get_Time();
}
```

5.4 main.c (Main Program)

This is the entry point. It initializes the hardware (including the 10ms TIM2 interrupt) and starts the scheduler.

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  SCH_Init();
  init_button_reading();
  HAL_TIM_Base_Start_IT(&htim2);

  // Add tasks
  SCH_Add_Task(button_reading, 0, 2); // 20ms

  // Problem 4 Tasks (10ms tick)
  SCH_Add_Task(LED_Task_500ms, 0, 50);    // 50 * 10ms = 0.5s
  SCH_Add_Task(LED_Task_1000ms, 0, 100);   // 100 * 10ms = 1s
  SCH_Add_Task(LED_Task_1500ms, 0, 150);   // 150 * 10ms = 1.5s
  SCH_Add_Task(LED_Task_2000ms, 0, 200);   // 200 * 10ms = 2s
  SCH_Add_Task(LED_Task_2500ms, 0, 250);   // 250 * 10ms =
```

```
 2.5s

  // Problem 5 Tasks
  SCH_Add_Task(OneShot_Task, 500, 0); // Delayed one-shot:
   500*10ms = 5s
  // SCH_Add_Task(Print_Timestamp, 0, 1); // 10ms periodic
   demo task

  while (1)
  {
    SCH_Dispatch_Tasks();
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM2)
  {
    SCH_Update();
  }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if (GPIO_Pin == button_1_Pin) {
    // ... (debounce logic) ...
    SCH_Add_Task(OneShot_Task, 0, 0);
  }
}
```

6 Summary

The demo of this project will be presented onsite and graded by the lecturer.