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

MICROCONTROLLER - MICROPROCESSOR

LAB 4

Cooperate Scheduler

INSTRUCTOR: Huỳnh Phúc Nghị
STUDENT: Tạ Gia Bảo - 2110795
GitHub submission: github.com/zabao-qt/GIT_LAB4

Contents

1 Problems and Report

1.1 void SCH_Update(void)

```
void SCH_Update() {
    // check if the list is empty
    if (!SCH_tasks_G.TASK_QUEUE[0].pTask) {
        return;
    }
    else {
        if (SCH_tasks_G.TASK_QUEUE[0].Delay == 0) {
            // The task is due to run
            // Inc. the "RunMe" flag
            SCH_tasks_G.TASK_QUEUE[0].RunMe += 1;
            if (SCH_tasks_G.TASK_QUEUE[0].Period) {
                // Schedule periodic tasks to run again
                SCH_tasks_G.TASK_QUEUE[0].Delay = SCH_tasks_G.TASK_QUEUE[0].Period
    ;
            }
        }
        // Not yet ready to run: just decrement the delay
        else SCH_tasks_G.TASK_QUEUE[0].Delay -= 1;
    }
}
```

Source 1: This function will be updated the remaining time of each tasks that are added to a queue. It will be called in the interrupt timer.

1.2 void SCH_Dispatch_Tasks(void)

```
void SCH_Dispatch_Tasks() {
    // Dispatches (runs) the next task (if one is ready)
    for(int index = 0; index < SCH_MAX_TASKS; index++) {
        if (SCH_tasks_G.TASK_QUEUE[index].RunMe > 0) {
            // Run the task
            (*SCH_tasks_G.TASK_QUEUE[index].pTask)();
            // Reset / reduce RunMe flag
            SCH_tasks_G.TASK_QUEUE[index].RunMe -= 1;
            // schedule to delete task
            SCH_Delete_Task();
        }
    }
}
```

Source 2: This function will get the task in the queue to run.

1.3 void SCH_Add_Task(void (*pFunction)(), uint32_t delay, uint32_t period)

```
void SCH_Add_Task(void (*pFunction)(), uint32_t delay, uint32_t period) {
    // check if numofTask overflowed
    if (SCH_tasks_G.numofTask >= SCH_MAX_TASKS) {
        return;
    }
    // create new task and insert to queue
    sTask temp;
    temp.pTask = pFunction;
```

```
    temp.Delay = delay / TICK;
    temp.Period = period / TICK;
    temp.RunMe = 0;
    insert_to_list(temp);
}
```

Source 3: This function is used to add a task to the queue.

1.4 void SCH_Delete_Task()

```
void SCH_Delete_Task() {
    int index = 0;
    // "flag" check to add task into queue to run again if it has period
    int add_back_flag = 0;
    sTask temp;
    // check one-shot task
    if (SCH_tasks_G.TASK_QUEUE[index].Period) {
        add_back_flag = 1;
        temp = SCH_tasks_G.TASK_QUEUE[index];
    }
    // shift left all tasks
    for (; index < SCH_tasks_G.numofTask - 1; index++) {
        SCH_tasks_G.TASK_QUEUE[index] = SCH_tasks_G.TASK_QUEUE[index + 1];
    }
    // delete task rear after shift left
    SCH_tasks_G.TASK_QUEUE[index].pTask = 0x0000;
    SCH_tasks_G.TASK_QUEUE[index].Delay = 0;
    SCH_tasks_G.TASK_QUEUE[index].Period = 0;
    SCH_tasks_G.TASK_QUEUE[index].RunMe = 0;
    SCH_tasks_G.numofTask -= 1;
    // check flag to add back into queue
    if (add_back_flag == 1) {
        insert_to_list(temp);
    }
}
```

Source 4: This function is used to delete a task.

1.5 led_blink.h

```
#ifndef INC_LED_BLINK_H_
#define INC_LED_BLINK_H_

void init_LED();
void blink_GREEN(void);
void blink_PURPLE(void);
void blink_RED(void);
void blink_WHITE(void);
void blink_YELLOW(void);

#endif /* INC_LED_BLINK_H_ */
```

Source 5: Defining all the LED tasks.

1.6 Tasks

1.6.1 Proteus schematic simulation design

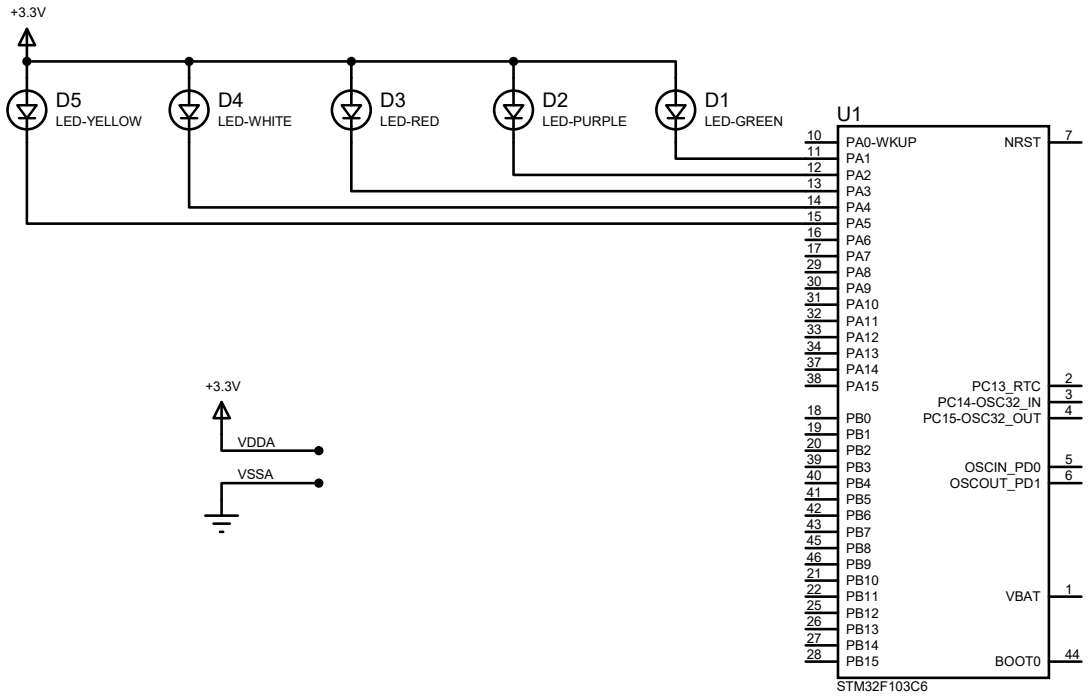

Figure 1: Schematic with 5 LEDs corresponding to 5 tasks.

1.6.2 Description

The implementation involves initializing all LEDs to the off state. Subsequently, the program executes the following tasks:

- `LED_GREEN`: Blinks every 0.5 seconds.
- `LED_PURPLE`: Blinks every 1 second and initiates with a 1-second delay.
- `LED_RED`: Blinks every 1.5 seconds and starts with a 2-second delay.
- `LED_WHITE`: Blinks every 2 seconds and begins with a 3-second delay.
- `LED_YELLOW`: Blinks every 2.5 seconds and starts with a 4-second delay.

This design ensures that each LED follows a specific blinking pattern with its designated frequency and initial delay.