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

MICROCONTROLLER MICROPROCESSOR (CO3009)

Lab Report

Lab 4

COOPERATIVE SCHEDULER

Teacher: Huynh Phuc Nghi
Student: Nguyen Thanh Hien (2111203)

GitHub: Lab 4

HO CHI MINH CITY, OCTOBER 2023

Contents

List of Figures

List of Tables

1 EXERCISE

1.1 Requirements

Your system should have at least four functions:

- void SCH_Update(void):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, for example 10 ms.
- void SCH_Dispatch_Tasks(void): This function will get the task in the queue to run.
- uint32_t SCH_Add_Task(void(*pFunction)(),uint32_t DELAY,uint32_t PERIOD): This function is used to add a task to the queue. It should return an ID that corresponds with the added task.
- uint8_t SCH_Delete_Task(uint32_t taskID):Thisfunctionisusedtodeletethetask based onits ID.

You should add more functions if you think it will help you to solve this problem. Your the main program must have 5 tasks running periodically in 0.5 seconds, 1 second, 1.5 sec ends, 2 seconds, 2.5 seconds

1.2 Proteus Simulation

In order to simulate the problem, we built a basic button schematic with 5 LEDs that would blink every: 500ms, 1000ms, 1500ms, 2000ms, and 2500ms; 1 LED to fire a one-shot task; 1 LED to toggle with every button pressed.

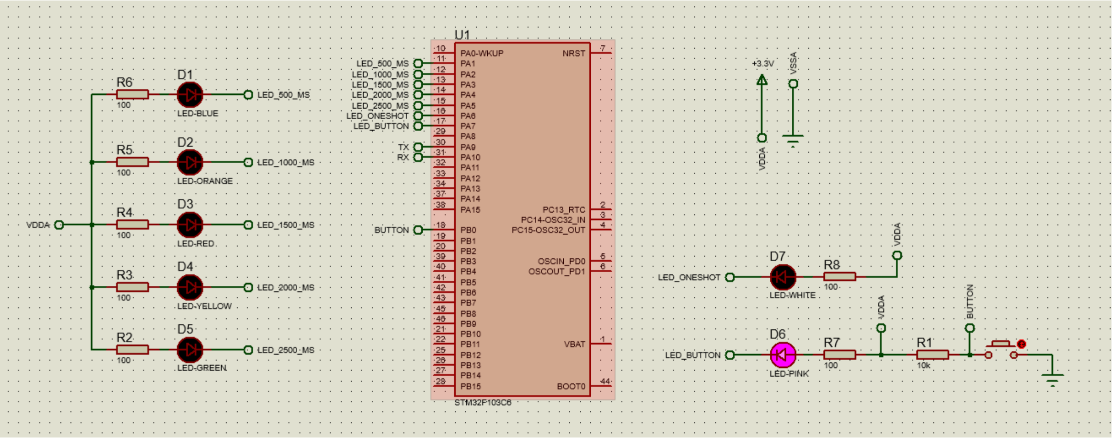

Figure 1: Schematic for testing purpose

1.3 Design idea

- **SCH_Update_Task**: Traverse all tasks in the array to modify the delay value, and raise a flag when timeout. This operates in O(n).
- **SCH_Add_Task**: Add a new task to the end of array only when there's space left. This operates in O(1).
- **SCH_Delete_Task**: Delete one-shot task and other task, then shift the remaining task forward. This operates in O(n).
- **SCH_Dispatch_Task**: Traverse every task in the array to find the task that is due to run (flag raised). This operates in O(n).

1.4 Implementation

1.4.1 Components

```c
#define MAX_SCHEDULE_TASK   40
typedef struct _SchedulerTask{
  void (*pTask)(void);  // Pointer to the task
  uint32_t period;     // Interval between subsequent runs
  uint32_t delay;      // Time remain before executing next task
  uint8_t flag;
} SchedulerTask;
SchedulerTask taskArray[MAX_SCHEDULE_TASK];

/* Public function declaration */
void SCH_Init_Task();
void SCH_Update_Task();
uint16_t SCH_Add_Task(void (* pFunction) () ,
         unsigned int delay,
         unsigned int period);
void SCH_Delete_Task(uint16_t taskID);
void SCH_Dispatch_Task();
void SCH_Go_To_Sleep();

/* Task function */
void blinkLED500();
void blinkLED1000();
void blinkLED1500();
void blinkLED2000();
void blinkLED2500();
void blinkLEDoneshot();
void blinkLEDButton();
```

Program 1: schduler.h

```c
int main(void)
{
  SCH_Init_Task();

  /* USER CODE BEGIN WHILE */
  SCH_Add_Task(blinkLED500, 0, TIMER_LED_500_MS / DEFAULT_TIMER_MS);
  SCH_Add_Task(blinkLED1000, 0, TIMER_LED_1000_MS / DEFAULT_TIMER_MS);
  SCH_Add_Task(blinkLED1500, 0, TIMER_LED_1500_MS / DEFAULT_TIMER_MS);
  SCH_Add_Task(blinkLED2000, 0, TIMER_LED_2000_MS / DEFAULT_TIMER_MS);
  SCH_Add_Task(blinkLED2500, 0, TIMER_LED_2500_MS / DEFAULT_TIMER_MS);
  SCH_Add_Task(blinkLEDoneshot, TIMER_LED_ONESHOT_MS / DEFAULT_TIMER_MS, 0);
  SCH_Add_Task(blinkLEDButton, 0, DEFAULT_TIMER_MS / DEFAULT_TIMER_MS);

  while (1)
  {
    SCH_Dispatch_Task();
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  SCH_Update_Task();
}
```

Program 2: main.c

1.4.2 SCH_Update_Task

```c
void SCH_Update_Task(){
  for(uint16_t index = 0; index < MAX_SCHEDULE_TASK;
   index++){
    if(taskArray[index].delay > 0){
      taskArray[index].delay -= 1;
    } else{
      taskArray[index].flag = 1;
    }
  }
}
```

Program 3: Function: **SCH_Update_Task** in scheduler.c

1.4.3 SCH_Add_Task

```c
uint16_t SCH_Add_Task(void (* pFunction) () ,
         unsigned int delay,
         unsigned int period){
  // Array is full
  if(currentTaskID >= MAX_SCHEDULE_TASK)
    return MAX_SCHEDULE_TASK;
  // Add task to the last index
  taskArray[currentTaskID].pTask = pFunction;
  taskArray[currentTaskID].delay = delay;
  taskArray[currentTaskID].period = period;
  taskArray[currentTaskID].flag = 0;
  currentTaskID++; // Move to next task
  return currentTaskID;
}
```

Program 4: Function: **SCH_Add_Task** in scheduler.c

1.4.4 SCH_Delete_Task

```c
void SCH_Delete_Task(uint16_t taskID){
  if(taskArray[taskID].pTask != 0 &&
      taskID >= 0 && taskID < MAX_SCHEDULE_TASK){
    return;// No task to delete
  }
  // Shift task forward
  for(uint16_t index = taskID; index < currentTaskID - 1; index++){
    taskArray[index].pTask = taskArray[index + 1].pTask;
    taskArray[index].delay = taskArray[index + 1].delay;
    taskArray[index].period = taskArray[index + 1].period;
    taskArray[index].flag = taskArray[index + 1].flag;
  }
  // Delete task at the back
  taskArray[currentTaskID - 1].pTask = 0x0000;
  taskArray[currentTaskID - 1].delay = 0;
  taskArray[currentTaskID - 1].period = 0;
  taskArray[currentTaskID - 1].flag = 0;
  currentTaskID--; // Reduce number of tasks
}
```

Program 5: Function: **SCH_Delete_Task** scheduler.c

1.4.5 SCH_Dispatch_Task

```c
void SCH_Dispatch_Task(){
  for(uint16_t index = 0; index < currentTaskID; index++){
    if(taskArray[index].flag == 1){
      // Execute task & Clear flag
      (*taskArray[index].pTask)();
      taskArray[index].flag = 0;
      taskArray[index].delay = taskArray[index].period;
      // Delete one-shot task
      if(taskArray[index].period == 0){
        SCH_Delete_Task(index);
      }
    }
  }
}
```

Program 6: Function: **SCH_Dispatch_Task** in scheduler.c

1.5 Testing

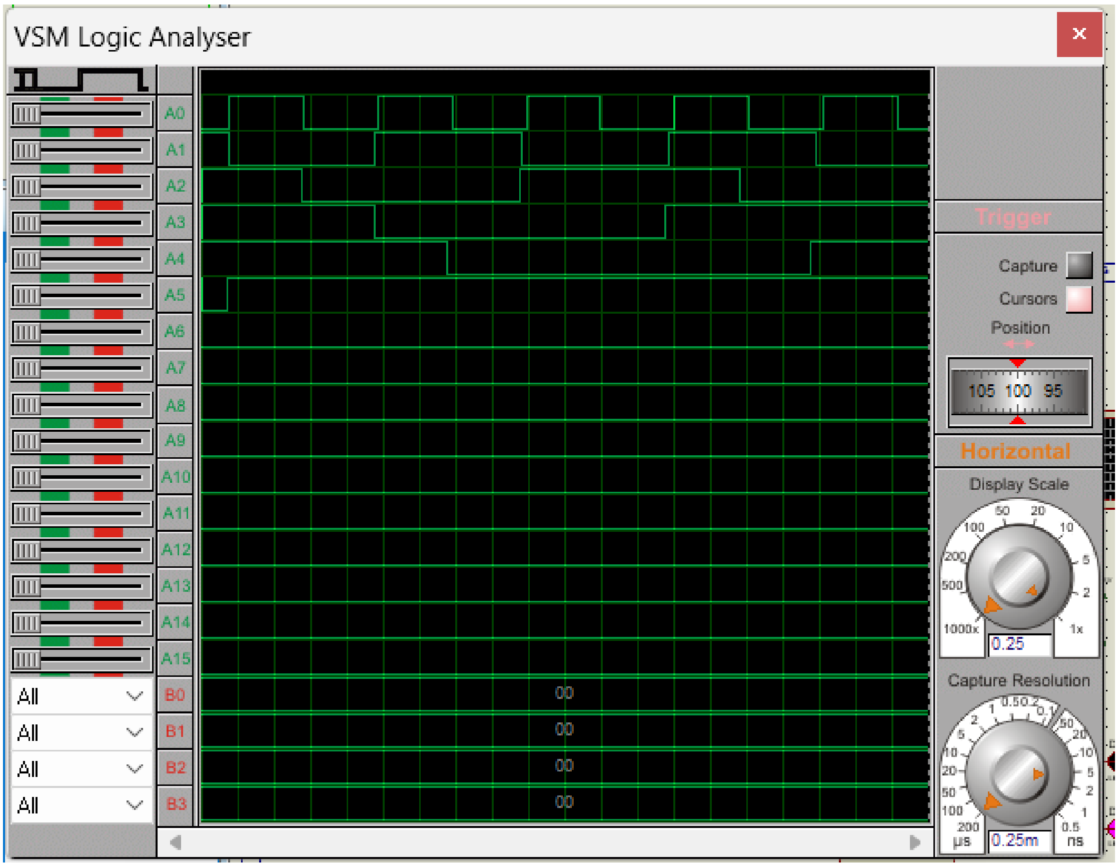

Figure 2: Logic Analyzer screen setup

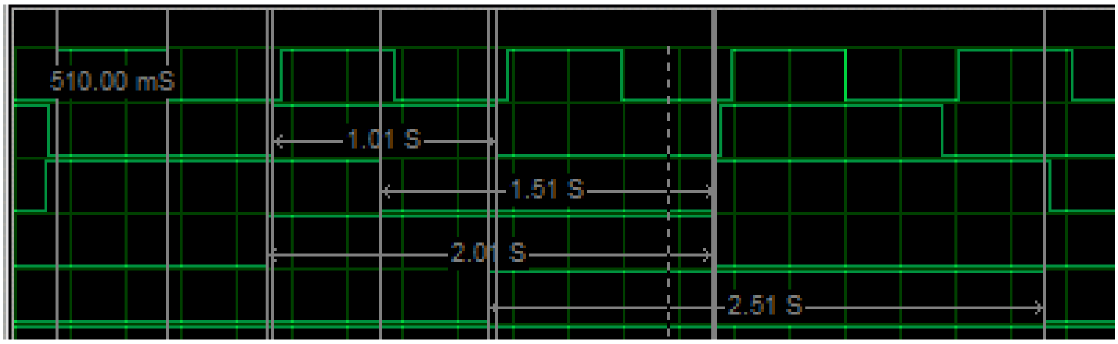

Figure 3: Logic Analyzer with time interval