CS7004 / Embedded Systems

Coursework Project

Lecture Theatre Lighting Control System

Implementation Report

Student Name:
Fan Li (14301044)

Lecturer:
Jonathan Dukes

December 19, 2016

Trinity College Dublin

The University of Dublin

1 Description

In this lab assignment, the main task is to design an embedded system to control the LED lighting in a lecture theatre. This is based on the *LPC2468* development kit and the *FreeRTOS* operating system.

The lighting system will devide the lecture theatre into four zones: the whiteboard spotlights, the lecture area, the aisle lighting and the seating area. Each LED connected to the *PCA9532 LED dimmer* are used to represent the lighting for one zone. The user interface which is displayed on the LCD with touchscreen can control these lights to be turned on/off, increase/decrease brightness and show states of motion sensors in each area. The master can control 4 LEDs at the same time, the LED on Pin P2.10 simulates the state of master. A reconfigurable preset is designed to fast control the states of four LEDs. The PIR motion sensors are used to automatically turn on the light when motion is detected in different parts of room, which are simulated by three push buttons. By degrees, the light gradually faded and eventually turned off within a preset time. However, there is no PIR sensor in the seat area since it's not reasonable to make people sit still and doesn't move, I use one button to simulate the smoke sensor which can trigger the smoke alarm.

In summery, the functionality of the lighting system are:

- **Master on/off switch for the lighting system**, simultaneously control lighting states of four zones, the master state is represented by LED on Pin P2.10. If master is not turned on, all the four LEDs can't be turned on or dim except a smoke alarm is triggered.
- **Independent on/off/dimmer control of each area**. This functionality only works when master is on. The brightness parameters are: 1 (turn on), 0.5, 0.1, 0 (turn off).
- **Stored and configurable preset**: a default preset is stored in the program, user can update the preset value by storing the current lighting state. This functionality won't work unless master is on.
- **PIR motion sensor**: When a motion is detected, the correspond sensor on LCD will turn red, if the light of correspond area is turned off currently, it will automatically turned on, gradually faded and eventually turned off after 3 seconds. Motion sensor doesn't work if smoke alarm is on.
- **Smoke alarm**: If a smoke alarm is triggered (push-button 3 is pressed), the four LEDs will be automatically turned on, the LED on Pin P2.10 will blink and the smoke alarm on LCD will flash red and cyan. The alarm will last for 5 seconds and the 4 LED will convert back to original state before the alarm is triggered.

2 Design

2.1 User Interface

Based on the functionality, a simple UI which has a set of buttons are displayed on LCD touch screen. The role of each button and LED is shown in Figure 1.

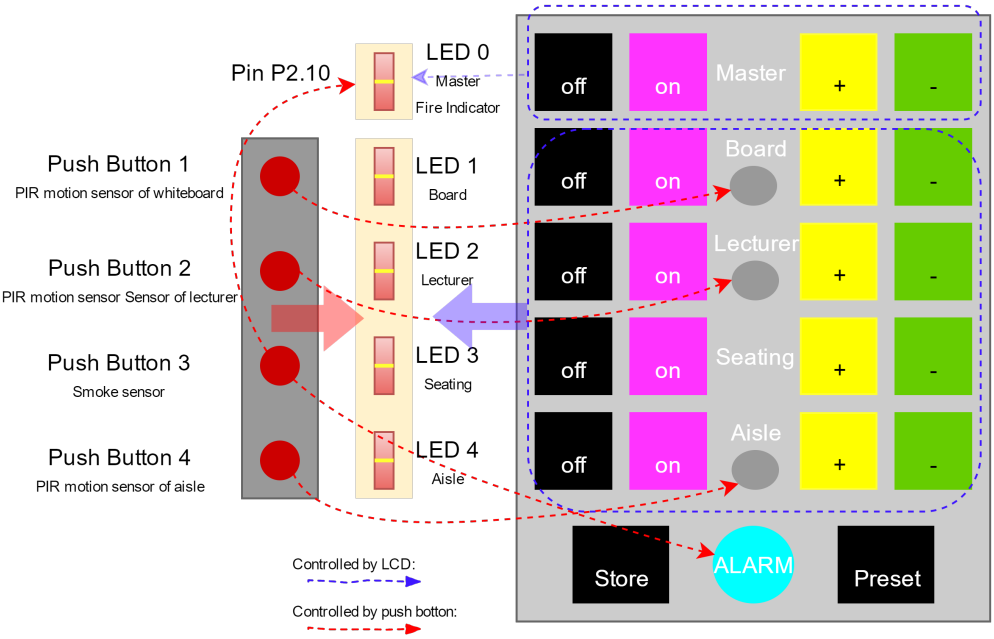

Figure 1: *User Interface*

The four buttons in 1st row is the master on/off/dimmer switch which can control LED_0 to LED_4. The whole system won't work until the master is turned on. From the 2nd row, the buttons are lighting controllers of each zone, which means they only affect the LED corresponded to that particular area.

In this system, the brightness is divided into 3 levels: full brightness, 50% dim and 90% dim. Once the "+" button is pressed, the brightness of corresponding LED_x is adjusted to its upper level. If it's already fully lit up, just keep the brightness. Similarly, if "-" button is pressed, the brightness goes down to its lower level. If it's already turned off, just keep the current state.

2.2 Command Message

The command is the message type which is exchanged via queue, it contains two factors: index of LED and value. The Index is used to identify correspond LED, the command value is used to perform specified functionality when tasks receive the message from queue. A set of predefined commands value are shown below, most of them are produced by LCD and sent to sensor except 2 commands: FIREALARM and PIR_MotionDetected. Separating the

command value produced by different tasks is important since it prevents the possible mess when two tasks both sending and receiving from a same queue.

- **LED_ON/LED_OFF**: Command to on/off switch.
- **DIM_INCREASE/DIM_DECREASE**: Command to dimmer control.
- **PIR_MotionDetected/PIR_MotionAction**: When motion detected command is received, the program will check the state of master (master on) and state of corresponding LED state (LED off) before performing the action.
- **STORE_PRESET/USE_PRESET**: Command to store and use the reconfigurable preset.
- **FIREALARM/AUTO_STORE/ALARM_RESTORE**: Command to trigger smoke alarm and restore system.

2.3 Task synchronize

The interaction between user and the lighting system is mainly two actions: touch the LCD display screen and press the push-buttons. The approaches to detect these events are polling and interrupts. The touch screen can raise interrupt when it's touched, but to detect the touch position the interrupts should be disabled. Here a binary semaphore is used to synchronize the LCD task, the process is shown in Figure 2.

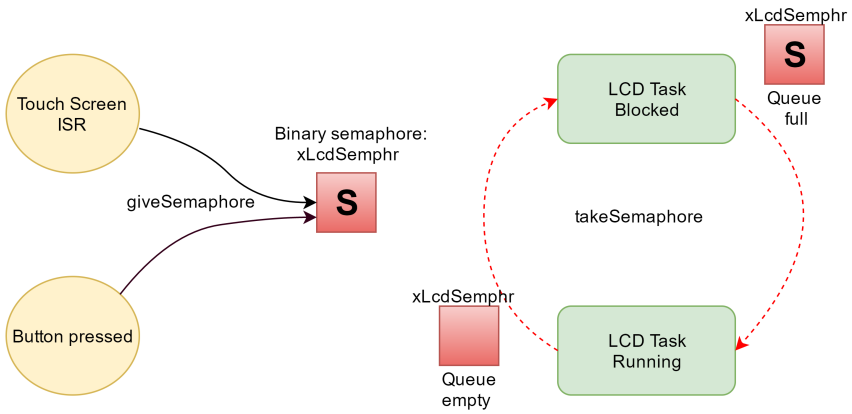

Figure 2: *Task synchronize*

3 Implementation

The program structure is shown in Figure 3, the queue and semaphore are created in main() function and passed as parameters to tasks. Initially I was using two queues, one stored the messages sent from sensor to LCD, the other stored messages sent from LCD to sensor. However using two queues increase a large amount processing time in each loop. Since there is no conflict on commands from different task, I used single queue in the end.

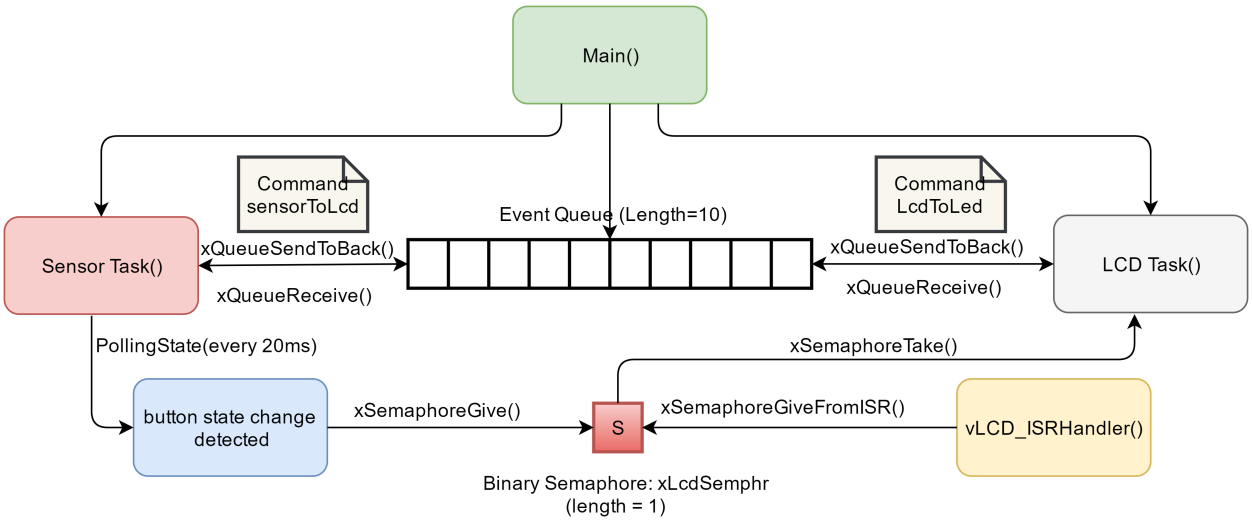

Figure 3: *Program Structure*

3.1 Turn on/off switch

To turn on or off the LEDs, we need to write expected LED state to *PCA9532 LED Controller* register. For switch of each LED, it's necessary to reserve the state of other LEDs while turn on/off a particular LED. The **bit operation logic** I used has two steps: first clear the correspond 2 bits of LED_x[1] using AND operation, then write 0 (off) or 1 (on) to those bits using OR operation. The state compute formula and data transfer process is shown in Figure 4.

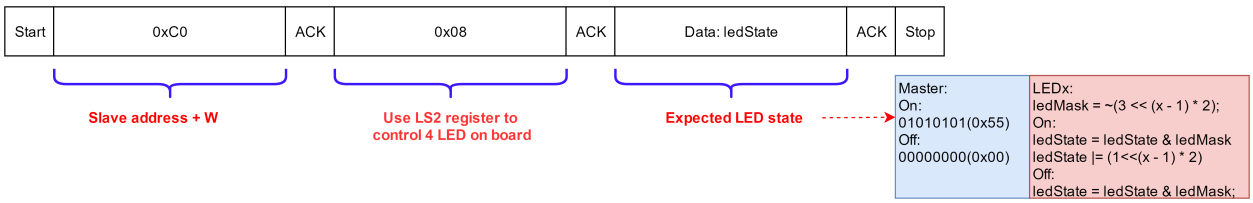

Figure 4: *Write to PCD9532: on/off switch*

3.2 Dimmer controller

The reason why designed 2-level dim is that we can use the 2 registers (PWM0 and PWM1) provided by *PCA9532* to store the predefined dim parameters. The *PWM0* saves the 50% brightness value and *PWM1* saves the 10% brightness value. The dimmer control of LED_x should not effect other LEDs, so the **bit operation logic** has three steps: first check the correspond 2 bits of LED_x and get the current brightness level, clear those two bits using AND operation, then using OR to update the brightness level of LED_x. The process is shown in Figure 5. The state compute formula and data transfer process is shown in Figure 6.

[1] LED_x: The particular LED correspond to the button pressed

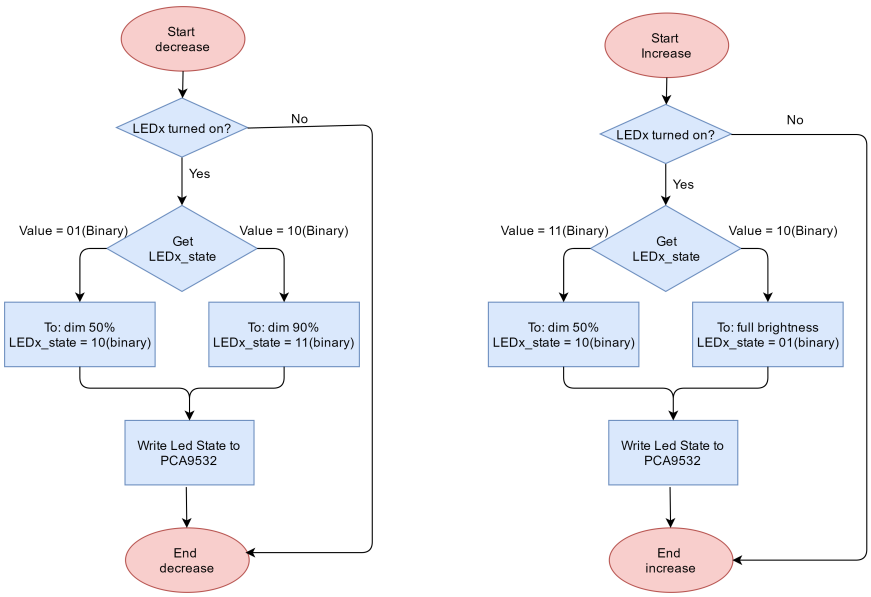

Figure 5: *Dimmer control logic*

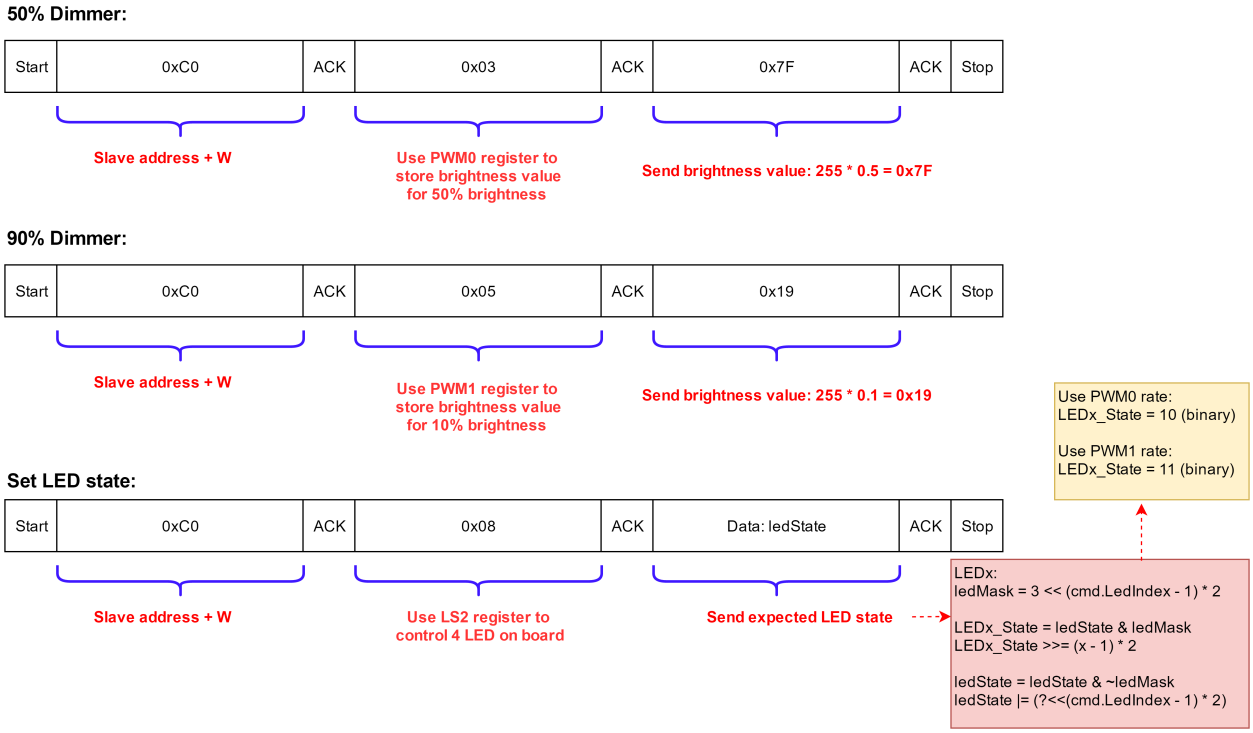

Figure 6: *Write to PCD9532: dimmer*

3.3 PIR Motion Sensor

Since PIR motion sensor is simulated by push-button and *PCA9532* can't raise an interrupt, we need to poll the buttons' state at the frequency of 20ms to detect changes between previous state and new state. The process of reading from *PCA9532* over I^2C is shown in Figure 7.

When a button is pushed and released, there are 2 changes detected. So the mod 2 operation is used to ensure that only one "button pushed" event is sent to queue. Also, the semaphore is given to LCD task when push-button is pressed to unblock LCD task. The logic flow is shown in Figure 8.

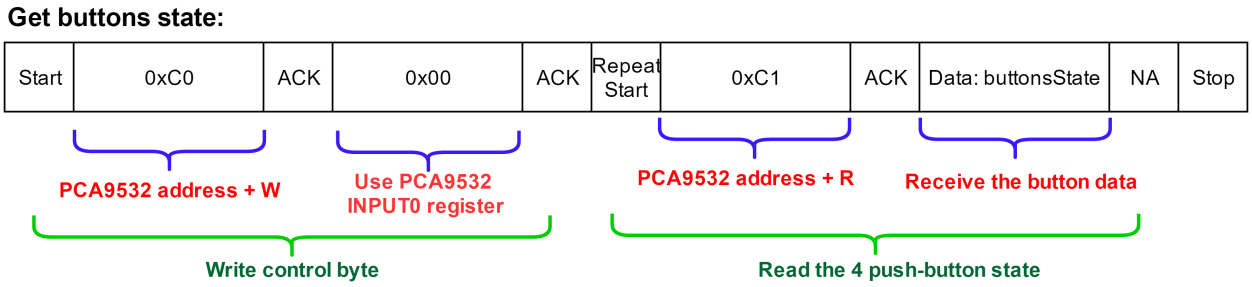

Figure 7: *Read from PCD9532: PIR motion sensor state*

3.3.1 Timer

After detecting a motion, the preset time to automatically turn off the light is 4 seconds. This is achieved by using *Software Timer* provided by *FreeRTOS*. In order to implement the gradually faded effect, the timer is set to 1 second. The timer's callback function is executed when the timer's period expires, this function will dim the brightness of correspond LED, redraw the sensor with red color on LCD, and reset the timer again, until the LED eventually turned off.

If we want to implement a more precise timer, it's better to use physical timer rather than software timer. Initially I used the *TIMER0* but the whole program is blocked. The reason is that *FreeRTOS* uses *TIMER0* to do the preempt, using *TIMER0* will block *FreeRTOS*. This problem should be avoided by using other physical timer.

3.4 Smoke Alarm

Since PIN P2.10 is used as *GPIO* function, The smoke alarm is triggered when the 3rd push-button is pressed (shown in Figure 3). If the smoke alarm is triggered, it will last for 5 seconds and all other functionalities of the system will be temporarily unavailable. In order to make LED P2.10 blink and smoke alarm flash red and cyan within the 5 seconds, the *FreeRTOS* software timer is used again. Here the time period is set to 5 seconds, during this time, the semaphore is given to LCD task in each loop to prevent the task being blocked. A counter variable is increased in each loop, by using mod operation, the state of LED P2.10 and color of alarm sign on LCD are updated in a human visible frequency. When the time is due, timer callback function restore the lighting system by clearing the alarm flag and send ALARM_RESTORE command to queue. The logic flow is shown in Figure 8.

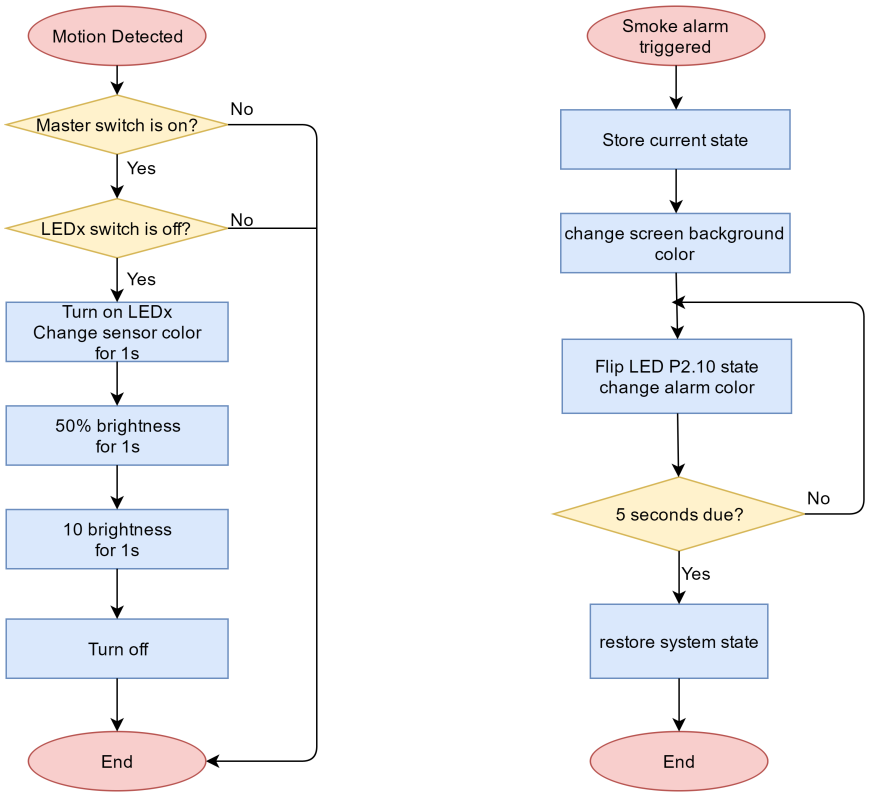

Figure 8: *Logic flow of motion detected event and alarm event*

4 Test Cases

4.1 Buttons on Touch Screen

Tested	**Before**					**After**				
button	LED_0(M)	LED_1	LED_2	LED_3	LED_4	LED_0(M)	LED_1	LED_2	LED_3	LED_4
Master "on"	off	off	off	off	off	on	on	on	on	on
Master "off"	on	off	off	on	on	off	off	off	off	off
LED_1 "on"	on	off	on	off	off	on	on	on	off	off
LED_1 "on"	off	off	off	off	off	off	off	off	off	off
LED_1 "off"	on	on	off	on	on	on	off	off	on	on
LED_2 "on"	on	off	off	off	off	on	off	on	off	off
LED_2 "on"	off	off	off	off	off	off	off	off	off	off
LED_2 "off"	on	on	on	on	on	on	on	off	on	on

Table 1: Test cases for switch buttons

Tested button	**Before** LED$_0$(M)	LED$_1$	LED$_2$	LED$_3$	LED$_4$	**After** LED$_0$(M)	LED$_1$	LED$_2$	LED$_3$	LED$_4$
Master "+"	off	0	0	0	0	0	0	0	0	0
Master "+"	on	0.1	0.1	0.1	0.1	on	0.5	0.5	0.5	0.5
Master "+"	on	1	0.5	0.1	0	on	1	1	0.5	0
Master "-"	on	1	1	1	1	on	0.5	0.5	0.5	0.5
Master "-"	on	1	0.5	0.5	0	on	0.5	0.1	0.1	0
LED$_1$ "+"	off	0	0	0	0	off	0	0	0	0
LED$_1$ "+"	on	0.1	0.1	0.5	1	on	0.5	0.1	0.5	1
LED$_1$ "+"	on	0.5	0.1	0.5	1	on	1	0.1	0.5	1
LED$_1$ "-"	off	0	0	0	0	off	0	0	0	0
LED$_1$ "-"	on	0.5	0.1	0.5	1	on	0.1	0.1	0.5	1
LED$_1$ "-"	on	1	1	1	1	on	0.5	1	1	1

Table 2: Test cases for dimer (unit: brightness percentage)

The default preset is: LED$_1$=1, LED$_2$=0.5, LED$_3$=0.1, LED$_4$=0. After storing another state: LED$_1$=0.1, LED$_2$=0, LED$_3$=1, LED$_4$=1, the preset is reconfigured. The second test case shows the updated preset value.

Tested button	**Before** LED$_0$(M)	LED$_1$	LED$_2$	LED$_3$	LED$_4$	**After** LED$_0$(M)	LED$_1$	LED$_2$	LED$_3$	LED$_4$
"preset"	on	1	1	1	1	on	1	0.5	0.1	0
"preset"	on	1	1	1	0.5	on	0.1	0	1	1

Table 3: Test cases for preset

4.2 Push-button

Tested button	**Before** LED$_0$(M)	LED$_1$ - LED$_4$	**0s later** LED$_0$(M)	LED$_1$ - LED$_4$	**5s later** LED$_0$(M)	LED$_1$ - LED$_4$
Button 3	off	0, 0, 0, 0 Cyan	blink	1, 1, 1, 1 Cyan/Red	off	0, 0, 0, 0 Cyan
Button 3	on	1, 1, 0.5, 0 Cyan	blink	1, 1, 1, 1 Cyan/Red	on	1, 1, 0.5, 0 Cyan

Table 4: Test cases for Smoke Sensor

Tested button	**Before** LED_0(M)	LED_1 - LED_4	**0s later** LED_1 - LED_4	**1s later** LED_1 - LED_4	**2s later** LED_1 - LED_4	**3s later** LED_1 - LED_4
Button 1	off	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey
Button 1	on	0, 1, 1, 0.5 Grey	1, 1, 1, 0.5 Red	0.5, 1, 1, 0.5 Grey	0.1, 1, 1, 0.5 Grey	0, 1, 1, 0.5 Grey
Button 4	off	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey	0, 0, 0, 0 Grey
Button 4	on	0, 1, 1, 0 Grey	0, 1, 1, 1 Red	0, 1, 1, 0.5 Grey	0, 1, 1, 0.1 Grey	0, 1, 1, 0 Grey

Table 5: Test cases for PIR motion sensors (unit: brightness percentage)

4.3 Source Code

Github link: https://github.com/vladdie/lighting-system.git

References

[1] "RTOS: Session20, FreeRtos: Software Timers ",
URL:*https://www.youtube.com/watch?v=bwKxGymYRmo*, Access time:12/15/2016.

[2] "FreeRTOS online API reference ",
URL:*http://www.freertos.org/a00106.html*, Last access time:12/18/2016.

[3] "PCA9532 Data Sheet",
URL:*https://tcd.blackboard.com/bbcswebdav/pid-739225-dt-content-rid-2474192_1/courses/CS7004-A-Y-201617/PCA9532.pdf*, page 4-8.

[4] "LPC2468 Base Board Schematic",
URL:*https://tcd.blackboard.com/bbcswebdav/pid-769729-dt-content-rid-2596676_1/courses/CS7004-A-Y-201617/assignment.pdf*, page 4.