CS7004 Embedded Systems
Assignment #2 - Burglar Alarm
Will Browne
09389822

Problem

The problem is to design an embedded system to control a simple domestic burglar alarm, making use of the FreeRTOS operating system using the LPC2468 development kit. The burglar alarm system (see Figure 1) consists of four sensors and is controlled by a touchscreen LCD display (see Figure 3). Each sensor is simulated by a push button on the LPC2468 board. The LCD should display ten numerals (for user input) and the current alarm status (armed, entry delay, unarmed etc.). LEDs physically adjacent to the push buttons should be used to indicate which of the sensors have been triggered, and should only be turned off when the alarm is disarmed.

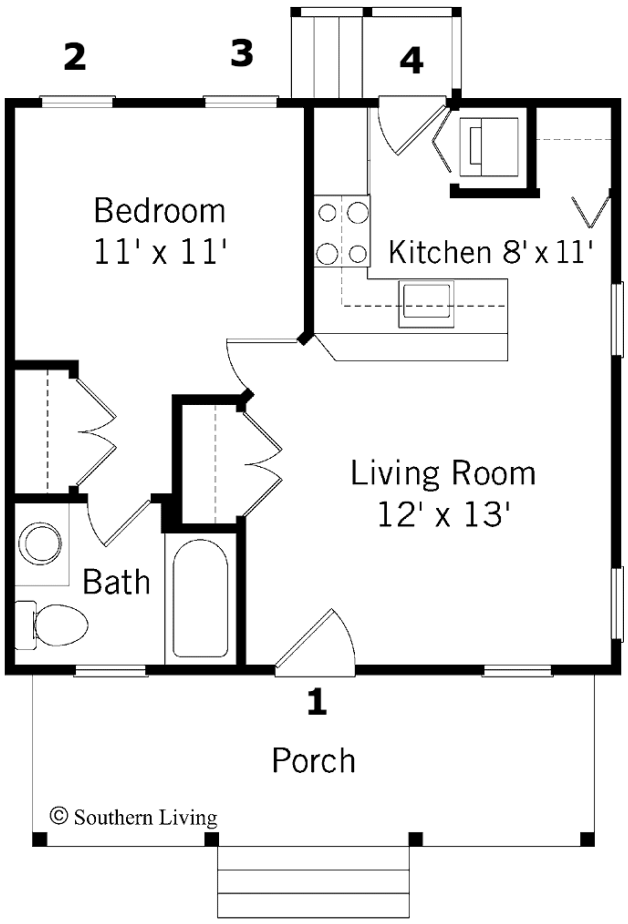

Figure 1: A hypothetical example of the burglar alarm. Each sensor is numbered 1 through 4, where the first represents the main entrance sensor. To allow someone to enter the house (thus triggering sensor 1), the system should wait for a period of 30 seconds to allow the alarmed to be disarmed before it sounds. At any stage, if any other sensor is triggered, the alarm should sound immediately.

20% of the assignment is based on any additional features added. For my solution I implemented:

- The ability for the user to enter their own code at the beginning of the application
- A clean and simplistic state machine design of the problem
- For both the alarm delays, a countdown timer appears on the LCD screen to provide feedback to the user

Design

State machine

My solution to the problem is designed as a state machine. There is a single task (master/controller) whose role is to control the current state of the system and communicate with each of the peripheral devices (LCD, pushbuttons, speaker/timers).

The different states of the system are as follows (these states are kept in a State enumeration):

- ARMED – the system is currently active and the alarm will sound if sensors 2-4 are triggered, or if sensor 1 (door) is tripped and the alarm code isn't entered within a certain timeframe
- DISARMED – the system is currently inactive and triggering the sensors has no effect
- ALARM SOUND – the alarm is currently sounding, indicating a burglary
- ENTRY DELAY – the door sensor has been triggered and there is an small amount of time to enter the alarm code before the alarm is sounded
- EXIT DELAY – the alarm code has been entered and the house must be exited within a certain amount of time before the system is armed
- UNITIALIZED – the system has not yet been initialized and requires a code to be configured by the user

In order to provide clean message communication, different structs are used to represent each different message:

- Master
    - {SOUND_ALARM, CODE_SUCCESS, PRIMARY_SENSOR, SECONDARY_SENSOR, LCD_TIME_UPDATE, INITIALIZE}
- LCD
    - {State state; /* represents one of the five different system states */
    - unsigned int duration; /* used for the delay states to display current countdown numeral */
    - }
- Sensors:
    - {LEDS_OFF, LED_ON, SECONDARY_OFF}
- Alarm:
    - {START_ALARM, STOP_ALARM, START_ENTRY_DELAY, STOP_ENTRY_DELAY, START_EXIT_DELAY, STOP_EXIT_DELAY}

State transitions:

*Message received from master queue

*Action taken (command put on peripheral queue)

- ARMED
  - CODE_SUCCESS → DISARMED
  - PRIMARY_SENSOR → ENTRY_DELAY (start entry delay)
  - SECONDARY_SENSOR → ALARM_SOUND (LED on)
- DISARMED
  - CODE_SUCCESS → EXIT_DELAY
  - PRIMARY_SENSOR/SECONDARY_SENSOR → (LEDs off)
- ALARM_SOUND
  - CODE_SUCCESS → DISARMED (stop alarm, turn LEDs off)
  - PRIMARY_SENSOR/SECONDARY_SENSOR → (LED on)
- ENTRY_DELAY
  - CODE_SUCCESS → DISARMED (stop entry delay, LEDs off)
  - SECONDARY_SENSOR → ALARM_SOUND (start alarm, LED on)
  - LCD_TIME_UPDATE → increment code entry delay
  - SOUND_ALARM → ALARM_SOUND (start alarm)
- EXIT_DELAY
  - CODE_SUCCESS → ARMED (stop exit delay)
  - PRIMARY_SENSOR (LEDs off)
  - SECONDARY_SENSOR → ALARM_SOUND (start alarm, LED on)
  - LCD_TIME_UPDATE → increment code entry delay
  - SOUND_ALARM → ALARM_SOUND (start alarm)
- UNITIALIZED
  - INITIALIZE → DISARMED
  - PRIMARY_SENSOR/SECONDARY_SENSOR → (LEDs off)

In order to provide communication between the different parts, four queues are used in my design:

1. Master queue
2. LCD queue
3. Sensors queue
4. Alarm queue

There is a total of six tasks used, that each provide the necessary implementation for the interaction of the different peripherals with the master controller:

1. Master task
2. LCD task
3. LCD State task
4. Sensors task

5. LED task
6. Timer task

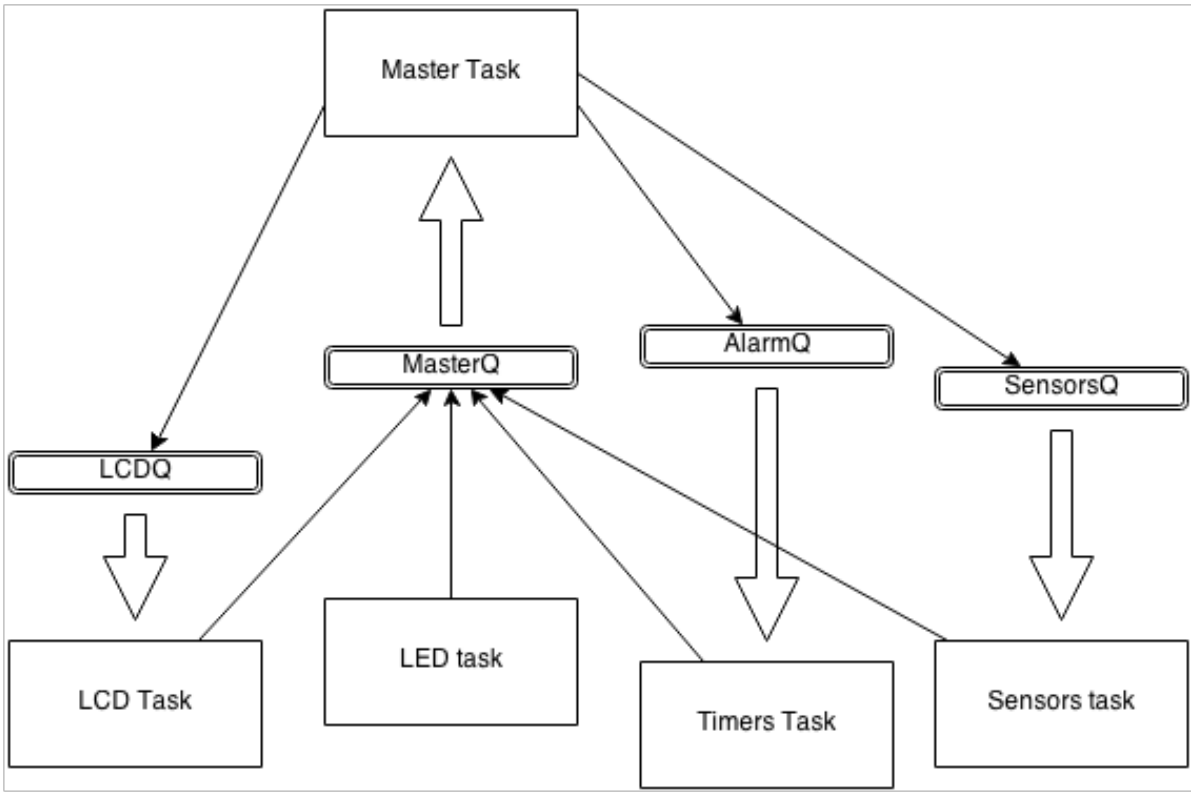

Figure 2: Overview of system interaction. The thick hollow arrows represent receiving from queue, and the thinner solid black arrows represent placing items on the queue.

I also made use of 4 software timers – alarm sound, entry delay, exit delay and delay LCD update. For testing purposes, both entry and exit delays are set to a period of 5s rather than the specification 30s.

Master task & Master queue

This is the main task that controls the overall flow of the burglar alarm system. It was the responsibility of this task to manage the state and flow of the system.

The master task has a corresponding queue (master queue), from which it receives information from all other peripheral devices. Once the master receives one of these commands from the queue, a decision is made depending on the current state of the system. For instance, if the current system state is ARMED and the master task receives a CODE_SUCCESS from the master queue, the state will transition to DISARMED as the LCD has registered a successful alarm code entry. (see State Transitions section above).

The master task has reference to each of the other peripherals queues in order to communicate with them (Note: as FreeRTOS only allows single queue handles to be passed into tasks, I had to create a QueueGroup struct in order to pass all other queue handles to the master).

The master task begins in an UNITIALIZED state, waiting for the LCD to send an INITIALIZE method, symbolizing that the user has created a code. From this point, the alarm will be put into a DISARMED state and the system is ready to use. Having received an item from the master queue, the master decides the state transition and then updates the LCD with this new state chance.

It is also the responsibility of the master task to maintain a variable that represents the delay countdown timer. This is used to provide feedback to the user through the LCD display and as mentioned before is set to 5s rather than the specification 30s.

LCD tasks & LCD queue

There are two tasks used to provide the functionality required for the LCD display.

1. LCD task
2. LCD State task

The LCD task's responsibility is to:

- Provide responsive and accurate touchscreen functionality through an interactive keypad
- Allow the user to input their own alarm code
- Register alarm code attempts and determine whether or not an inputted alarm code is correct
- Provide correct system feedback when necessary

Jonathan provided some of the functionality for this task online so I decided to reuse this code – most notably the touchscreen release code.

In order to draw the keypad grid, two nested for loops are used, making use of a token array (an array of characters that represents all of the tokens that are used on the LCD display e.g. digits 0-9, clear and submit) to draw each to their corresponding button (see Figure 3). In order to register a correct button touch, the x position and y position of the touch release event is captured and fed into a function. This function uses both values to find the corresponding row and column values, treating the button grid like a 2D array (4x3).

```
int col = floor(*xPos / 70); /* 70 is width of button box */
int row = floor((*yPos-40) / 65); /* 40 pixels at top of LCD we ignore and 65 is length of button box*/
```

Using these row and column values we can determine an index into the token array in order to register what button the user has pressed. For example, if number 6 is pressed on the grid, I want to be able to register that as index 5 (element “6” in the token array).

```
return (row * (ROWS-1)) + col;
```

As the digits are entered, the attempt is displayed on the screen. The user can choose to clear a digit once entered, and also submit the attempt once a 4-digit code has been entered. This functionality is provided using a codeIndex variable, which represents the index of the next alarm code digit to be entered. In order to display each alarm digit properly on the LCD, an xOffset variable was used to move forwards or backwards in the entry sequence depending on what button was pressed.

The first time the user attempts to use the system, they are asked to enter their own code. This code is then saved in a global array and the master is then informed through an INITIALIZE command. From that point (the system is now DISARMED), if a successful code entry is made, a CODE_SUCCESS command is put on the master queue.

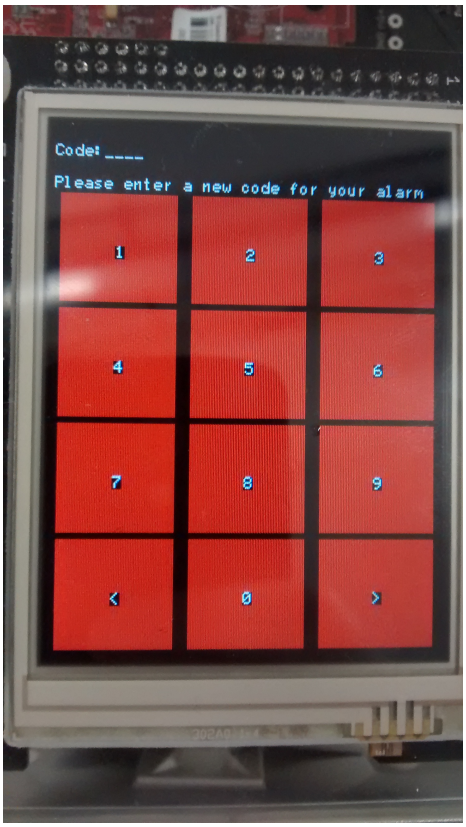

Figure 3: The alarm's LCD display.

The LCD State task's responsibility is to receive system state updates and display them on the LCD display. This information is transmitted through an LCDMessage struct and is received every time the master changes state.

Sensors Task & LED Task

Both these tasks are closely related as they involve the use of the I^2C bus and some of the code used in these tasks is reused from examples provided by Jonathan.

The sensors task provides the functionality of the 4 push buttons. In order to determine the current states of the buttons, polling is used over the I^2C bus to read in the PCA9532 INPUT0

register. The state of these buttons is kept in a global variable, which is put on the master queue every time a button is pushed. Depending on what button is pushed, a PRIMARY_SENSOR/SECONDARY_SENSOR is put on the master queue so the master knows what type of sensor has been triggered.

The LED task's responsibility is to turn on/off the LEDS depending on the information in the shared button states variable. It does this through a function *setLEDs(buttonStates)*. This function transmits information over the I^2C bus, by setting a register that controls the LEDs output (see Figure 4).

This task is capable of receiving 2 different types of information from the master task:

1. LED_ON – turn on the LEDs according to the current button states variable
2. LEDS_OFF – turn off all of the LEDs

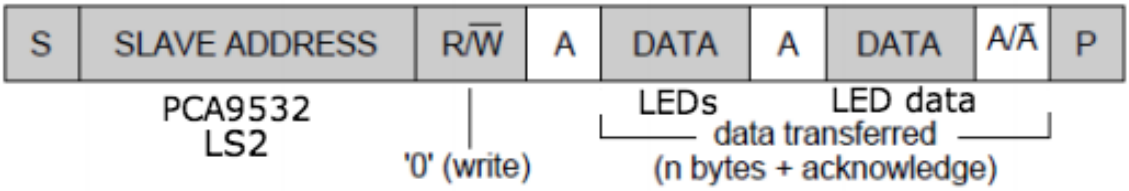

Figure 4: Interaction over the I^2C in order to set the LEDs. After the start is sent, the PCA9532 register address is transmitted. As we are working with LEDs 8-11, we need to send 0x8 (LEDs) as the data to the PCA9532, and then we need to write what lights we want to turn on (LED data).

Since the state of the buttons is abstracted away from the master and is set in the LED task, each button press has to be communicated to the master. This means that for every button press, the new state is recorded, but may shortly be discarded depending on the current system state. For instance in the DISARMED state if button 2 is pressed, a SECONDARY_SENSOR command will be put on the master queue. No sensors are supposed to be triggered in this scenario so the LEDS_OFF is put on the sensors queue, so that the button states will be set to 0.

Timer Task

I made use of 4 different software timers in my implementation:

- AlarmTimer (2 ms) - The alarm timer is used to control how frequently the speaker is sounded. In the callback function, a 10-bit sequence in register DACR is alternated to control the speaker output.
- EntryDelayTimer (5s) – This timer is used to control the period of time allowed for an individual to input the code after triggering sensor 1 in ARMED mode. In the callback function, a SOUND_ ALARM command is placed on the master queue signalling that a code has not been inputted during the delay (if a code is inputted, the Timer Task will received a STOP_ENTRY_DELAY message and stop the entry delay timer). If a

secondary sensor is triggered during this ENTRY DELAY state, it will transition to ALARM_SOUND and will start sounding.

- ExitDelayTimer (5s) – This timer is used to control the period of time allowed for an individual to leave the house after putting the system in ARMED mode (will ignore sensor 1). In the callback function, a CODE_SUCCESS command is placed on the master queue signalling that the house has been exited securely. If a secondary sensor is triggered during this EXIT DELAY state, it will transition to ALARM_SOUND and will start sounding.
- lcdTimerDisplay (1s) – This timer is used to send periodic updates to the LCD display so that a countdown can be displayed to the user when in the ENTRY/EXIT DELAY state. This is to provide the user with feedback on how long they have to enter the code before the alarm sounds/they leave the house.

Testing

In order to ensure that my solution worked as outlined in the specification I made various test cases and observed the results.

Test cases

- LCD
    - Have a desired 4 digit code and attempt to enter it - ensure it registers the correct digits on screen
    - Enter a code and press the delete key - ensure that previously entered digits can be cleared
    - Enter an incorrect alarm code – check feedback is displayed
    - Enter an correct alarm code – check correct state transition
    - For each different state change, ensure that it appears via the LCD
- General
    - UNITIALIZED
        - Trigger all sensors - ensure no effect on the alarm system
    - ARMED
        - Trigger primary sensor – entry delay started
        - Trigger secondary sensor(s) – alarm sounds immediately and corresponding LED(s) turn on
        - Triggering primary & secondary together – ensure alarm sounds
        - Enter wrong code – feedback message explaining wrong code entered
        - Enter correct code – exit delay is triggered
    - DISARMED
        - Trigger primary sensor/secondary sensors – observe no effect
        - Trigger multiple sensors – observe no effect
        - Enter wrong code – feedback message explaining wrong code entered
        - Enter correct code – exit delay is triggered
    - ENTRY DELAY
        - Trigger primary sensor – observe no effect

        - Trigger secondary sensor – immediate alarm
        - Triggering primary & secondary together – immediate alarm
    - EXIT DELAY
        - Trigger primary sensor – observe no effect
        - Trigger secondary sensor – immediate alarm
        - Triggering primary & secondary together – immediate alarm
    - ALARM_SOUND
        - Trigger primary sensor – corresponding LEDs turn on (this may already be on depending previous state was entry delay)
        - Trigger secondary sensor – corresponding LEDs turn on
        - Trigger sensor that's already turned on – no effect

A big pain that I experienced was having LEDs on during the ALARM_SOUND state when they shouldn't have been. This was because the button states variable (which is used over the I^2C but to turn LEDs on) always get set no matter what the state, as mentioned before, as the state lies in the sensors task rather than the master task. This means that for certain states I need to reply to the task and set the button states variable back to 0. To ensure that I had gotten all cases I had two test cases I frequently ran through:

- Triggering sensors during INITIALIZED state
- Triggering sensors during DISARMED state

Following both these cases, I put the system into an alarm mode and checked that only correct LEDs were active.

I was quite sceptical initially about having the sensors task take control of the button states rather than the master. My reasoning was that since I believe that the alarm code validation should take place in the LCD task, I thought it would be consistent to have the sensors task structured similarly. Although this does seem to break away from an overall clean design, I don't think it could have been avoided.

Conclusion

I am happy with my design of the system and that my implementation functions accordingly to the specification.