Luca Meschiari

Burglar Alarm System

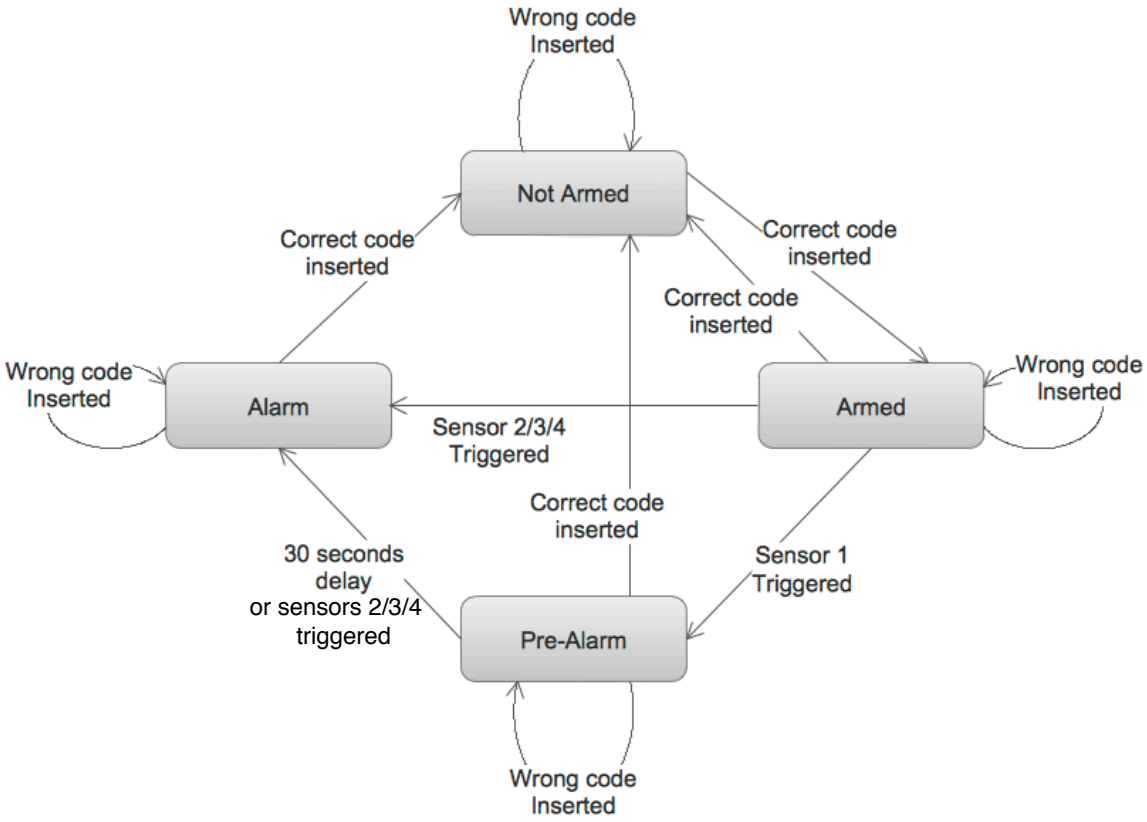
Declaration

Program overview

The program developed is a simple house burglar alarm system, it uses 4 sensors to check if there is any effraction, 4 led to signal on what sensor the anomaly is sensed and a touch screen to give the user the basic informations about the system and get input from the visualized virtual keyboard.
The sensors are simulated by 4 pushbutton connected to the I2C controller and the first sensor(so the first button) is assumed to be the one connected to the front door.
The system has 3 main states:

- Not armed: when the alarm is turned off
- Armed:when the alarm is turned on and is checking for effractions
- Alarm: when the system has recognized an anomaly on one of the sensors and so is ringing.

There is also one more state that I have introduced to allow the user to have 30 seconds to deactivate the system when he enter from the front door before the system pass in the alarm state. This fourth state is called in this project Pre-Alarm state.
The activation and the deactivation of the system, since there is only the fully armed option, is done entering a for digits code code. **(The code used is 1-2-3-4)**.
The touch screen lcd display offers a graphical interface to the user communicating the state of the system also informing the user on the amount of time remaining if the sensor 1 is triggered (in the armed state) before switching to the alarm state. it also show a virtual numeric keyboard which can be used to enter the code to manage the system, the code inserted is shown on the screen and in case a wrong code is typed an error message appears on the screen to notify the user.
The LEDs will show the sensors who has detected the effraction (only when the system is in Alarm state) and also if another sensor is triggered its corresponding LED would be turned on. Thanks to that the user can have a full overview of the state of the system.
For a better understanding of the functionalities of the system the following state diagram can be helpful:



System Design:

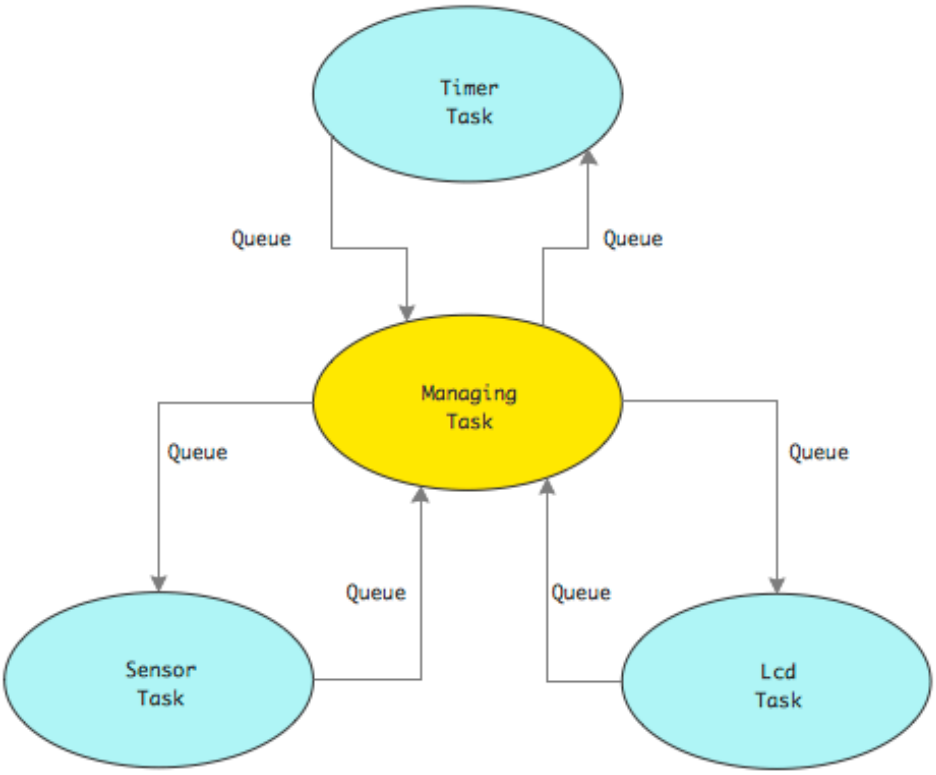
Basically the burglar alarm software need to handle the following inputs and outputs:

- Sensors/Button input
- LED output
- LCD output
- LCD input
- Timer management to control the 30 seconds delay

Some of this functions can be grouped since they have a similar interface, this is the case for the Sensor and LED management and the LCD management.
All the needed functionalities, grouped together when possible, are provided by three tasks:

- Lcd task: manage the LCD input and output.
- Sensors task: manage the Button(sensors) input and the LEDs output
- Timer task: used to control the 30 seconds delay.

All this tasks provide information but a centralized task which retrieve all the gathered data, make decisions about the state of the alarm and send information to all the other tasks about how to update the output is needed. This task is called Managing task.
The communications between tasks are achieved by queues to allow tasks to send also commands and not only a synchronization signal like the semaphores. These control signals are integer numbers and they are used to exchange data between tasks. To achieve a bidirectional communication between the tasks a couple of queues is used for each connection (since a single queue not provide a full-duplex bidirectional communication channel).
The architecture of the system is explained in the following image:



To allow two task to share the same queue all the queues used for intertask communication are created in the main function and passed to the tasks as an argument when they are created.
This architecture allow to separate the access to different outputs and inputs to have a cleaner system. For example the Lcd is managed only by the Lcd Task and if the managing task want to change the data displayed it needs to send a command to the Lcd Task instead of changing that itself.

LCD management
In this section I will explain the main characterics of the LCD interface, the technical details about the LCD task are explained in the Task Design section.
The LCD interface is basically divided in two parts: the one on the top shows a touchscreen numeric keyboard which provide the user feedback about the tapping of a button changing its colour.
The bottom part of the interface shows the system status, the code entered, inform the user if a wrong code is inserted and if the system is in pre alarm mode inform the user about the situation drawing a red rectangle and showing a 30 second countdown with 5 seconds ticks.

LEDs management:
The LEDs are turned on when the system is in armed (sensors2-3-4), pre alarm (sensors 2-3-4) or alarm state and the corresponding sensor is triggered. If two or more sensors are triggered all the corresponding LEDs are lighted. If the system is in armed state and the sensor one is triggered the corresponding LED will be turned on only after the expiration of the 30 seconds delay.

Tasks Design

Since all the task are interconnected is quite complicated to explain their functionalities separately but it still useful to understand the main structure of the system.

Managing Task:

The managing task gather all the data from the other tasks and make decisions about the state of the system. After, if it is needed, it sends commands (using the queues) to the other tasks to update the outputs.
The core of the task in an infinite loop containing the following checks in order:

- Check sensor status: get the status of the sensors and if is needed (depending on the current state) change the state of the system and update the Lcd and the LED.
- Check Lcd Input: Check if the user has inserted a number and he has inserted 4 digits so far perform the matching with the correct code. In case it is correct update the state of the system(if needed) and update the outputs. If the code is wrong update the Lcd to notify the user.
- Check the timer: If the system is in pre alarm mode check the time elapsed to manage the 30 seconds delay and update the countdown on the Lcd screen.

Lcd Task:

The Lcd task manage the Lcd and draw the graphic interface on it. It uses 3 functions:

- button_pressed: using the coordinates of the pressure point highlight the pressed button and return the associated digit.
- restore_button: restore a button to the normal color if is not pressed anymore
- write_state: draw the bottom part of the display(the one showing the state of the system, the code entered, the wrong code message and the countdown)

To manage actions on the LCD an interrupt hander is introduced, when the handler is called it only manage the interrupt and send data to the main function using a queue.
In the main function of the task inside a infinite loop the following checking are performed:

- Check the LCD queue (the one from the interrupt handler) to have information about the user input, the LCD is updated and the data about the pressed button is sent to the Managing task using the queue. To avoid bouncing a 50 ms delay is introduced and The pressure on the display is checked to get a single input for every tap. To get informations from the LCD (position of the touch and pressure) the interrupt is disabled and then reenabled after the readings.
- Check the queue from the managing task to get informations about the updates to perform on the display.

Sensors Task

The sensor task manage the button input and the LED output.
It basically polls the sensors using the provided getButtons function to retrieve the state of the buttons. The previous state of the buttons is stored and compared with the new state to get information only when the button is pushed and not for the all time that is triggered. The button information is sent to the managing task using a queue. The sensors task also listen for commands sent by the managing task to update the LED output.

Timer Task

The timer task manage the 30 seconds delay and also update the managing task about the remaining time every 5 seconds when the delay is checked.
It listens for a start command form the managing task and if it is received the counting is started, The functions used to manage the time are xTaskGetTickCount to get the current

time (this function is called before executing other code to have a correct timing) and the vTaskDelayUntil to have 5 seconds ticks. The time remaining is then sent to the managing task using a queue. The task also listen for a stop command from the managing task to stop and reset the timer in case the system state changes.
To have a better precision and avoid delays the timer task has an higher priority than all the other tasks.