**Code Explanation Documentation for the ESP32 Automated House**

By Jacob Beardsley and Carlos Gomez Gutierrez

This document provides an in-depth explanation of the Automated House and its implementations created by this team. The automated house features an ESP32 microcontroller that controls various components throughout the house utilizing different sensors. The house’s program uses FreeRTOS to help provide scheduling and synchronization through the house and its components. As a part of FreeRTOS, we utilized semaphores/mutexes, timers, queues, and tasks to help achieve that synchronization.

The automated house features an indoor light controlled by a switch and an outdoor light controlled by a photoresistor that detects how much light is present outside. The alarm system features three different states: Inactive, Active, and Intruder. Each of these statuses displays a different color in the RGB LED: blue for inactive, green for active, and red for intruder. The system utilizes a magnetic sensor to detect whether the front door is closed or not and a button to switch the alarm from inactive to active, active to inactive, and intruder to inactive. The system switches from active to intruder in the case that the alarm is active and the front door is open. When the alarm is in the intruder mode, the buzzer sounds off in one-second intervals using software timers. After 2 minutes, the buzzer is timed out. The garage door is controlled via a servo, and an indoor and outdoor IR sensor, both of which are connected to interrupts that tell the garage door to close and open, respectively. The button is also able to manually open or close the garage door via a short press. The garage door can also be reversed while closing or opening. For example, if the garage door had been closing for .4 seconds, and the button is pressed, the garage would open for .4 seconds, so as to return to the initial state. The house also features a temperature and humidity sensor which reads and reports the temperature and humidity every 100ms as dictated by a software timer. The temperature and humidity are also displayed in real-time via an LCD display that uses I2C. The garage door, front door, and alarm status are displayed on an OLED that also uses I2C. The OLED displays a changing image when the alarm is in intruder status. The smoke sensor detects the presence of gas and smoke in the house, and when it is above a certain value, it turns the fan on for 20 seconds to vacuum the air inside the house out.

These components and behaviors are governed through FreeRTOS tasks. The button task makes use of a library called ezButton to handle debouncing, as well as to detect when the button is pressed and released. The task uses variables to compare when the button was pressed and released to see if a long or short press occurred. Depending on the time of press, it informs the appropriate global variable uses. This task and others utilize the updateVariable function to efficiently update a variable using a mutex.

The IndoorLED1 grabs the switch status and updates the indoor LED appropriately. This task uses the ezButton library to debounce the switch and get the status of the switch. This task also updates the global switch state variable using a mutex so the indoor LED status can be seen elsewhere. This task also checks to see if the light has changed status, and if it has, it updates a queue so the SD card task can log the change.

The indoor and outdoor IR sensors are attached to interrupts that send a value to the xTaskIn and xTaskOut queues, respectively. The setIndoorIR and setOutdoorIR tasks receive values from those queues and the values of their respective global variables so the garage can know if there is a car inside or outside.

The setServo task governs the function of the garage door. It uses a variety of variables to decide whether or not and how to activate the garage door. If a car is only detected outside and the garage is currently closed, it opens it, and vice versa. There are also a variety of other cases it checks for like when it is in the process of opening and closing so it can reverse the appropriate amount of time. This task uses the ESP32Servo library to write to the servo motor. The rotation of the servo motor opens and closes the garage door. The task uses a switch statement to organize the servo’s behavior depending on its current state. Once it finishes opening or closing, it returns to the opened or closed state. We also have an if statement that sends a value to a change queue if the garage status changes so the SD task can log it.

The setLightSen (photoresistor), setSmokeSen (MQ-2 Smoke Sensor), setDoorState (Hall Effect Sensor), setTempSen (SHT31-D), and setHumiditySensor (SHT31-D) tasks all query their respective sensors and update their respective global variables using mutexes. The setTempSen and setHumiditySensor tasks are communicating to the same sensor using I2C communication, which is protected by a mutex. The setSmokeSen activates the fan if it detects enough gas/smoke in the house. The setLEDDark task uses the information from the setLightSen task to determine if it’s dark enough outside to activate the outdoor LED.

The setAlarmState task governs the behavior of the house’s alarm system. It gets information from the button and door tasks to see if a long press is detected or if the door is open or closed. Depending on the particular case, it switches the alarm to the appropriate state. If the door is closed, the alarm is inactive, and the button is pressed long enough, the alarm switches to the active state. If pressed again, it switches back to the inactive state. The only time that the alarm switches to the intruder state is when the alarm is active and the door is opened. Then, the only way to deactivate it is to long press the button, switching it to inactive. Depending on the state, a switch statement governs the status of the RGB and buzzer. If in the intruder mode, a software timer is activated to alternate the buzzer between on and off states with 1 second for each. This timer’s callback function sends a value to the xAlarmTimerQueue which lets the task know to switch the alarm state. If still in the intruder mode after 2 minutes, the alarmTimeOutCallback function is called which turns the buzzer off. After the state is switched from intruder to another state, the timers remain active but are reset along with their queues when the alarm is switched back to the intruder mode. There is also a condition in this task to detect an alarm status change so the SD task can record it.

The setOLED task writes the status of the garage, alarm, and front door to the OLED. This task uses the SSD1306 library to communicate with the OLED. Using mutexes, the task grabs values from the respective global variables to write to the OLED using if statements to interpret the values from the variables. Once done, it sends the proper message to the OLED in its appropriate cursor location on the screen. If the alarm state is in intruder mode, an image of a speaker is displayed corresponding to the status of the buzzer. This task uses the I2C mutex to protect its communication as the OLED uses I2C.

The LCD task writes the temperature and humidity measurements from the SHT31-D sensor. It uses the I2C mutex as the LCD uses I2C. This task also uses the LiquidCrystal\_I2C library to communicate with the LCD.

The writeSD task checks to see if changes have occurred to the garage door, front door, indoor light, outdoor light, and alarm state. If changes have been made, the task appends these changes to the log file. Each log is preceded by a time code generated from the getTimeCode function. The time is calibrated by the NTP server when the ESP32 boots. The SD card receives the information to send to the SD card from the change queues in the tasks of the components that we’re recording changes in. This task uses the FS, SD, and SPI libraries to communicate with the module. This module uses SPI to communicate with the ESP32.

The repDALState runs every 10 seconds and reports the status of the front door, indoor light, outdoor light, alarm, and temperature. It retrieves values from the global variables using mutexes.

At setup time, we initialize a variety of objects to begin communication to the components like the OLED, SD module, LCD, etc. During setup, the ESP32 attempts to connect to wifi for 10 seconds. If it succeeds, it gets the time from the NTP server and attempts to do so for 10 more seconds. If successful, we disconnect from the wifi. If not, the ESP32 uses a default time for the time codes. If there’s an SD card in the module, the setup code writes a system boot message to the SD card. In setup, all of the timers, queues, mutexes, and tasks are created.

Lastly, the OLEDbitmaps file stores the bitmaps for the OLED images.