Lab4: Free RT-OS based Arduino Projects

Paria Naghavi (1441396)
Anna Mai (215101)
Assignment: Lab 4 August 18, 2023

Table of Contents

Introduction

The part B of this lab highlights the power of FreeRTOS in managing complex inter-task dependencies, while synchronizing tasks such as data generation, processing and real-time outputs. This using FreeRTOS provide numerous benefits, especially as application complexity grows. However, for simple applications, a bare metal approach may be sufficient and can offer lower overheads.

Part C of this lab focuses on the more tangible applications of FreeRTOS, and how its capabilities can be implemented into the embedded systems that we see around us in our daily lives. Additionally, we also explored some previously untouched devices in our arduino kits, including the LCD display and photoresistor. Please see for bonus video here.

Methods and Techniques

For part B, the circuit components that were used in the construction of part B hardware were an 8-ohm speaker, an external LED and a resistor, in addition to the Arduino board.

For part C, the circuit components used for the interface Arduino side included: an LCD display, a 10k ohm resistor, and a button. For the player arduino, the components used included: a photoresistor, a 10k ohm resistor, a 330 ohm resistor, alligator clips, and an external LED.

Experimental Results

Part B

Setting up task 2 without using FreeRTOS, with either multitasker or ISR, we had to implement two functions for the speaker to play the theme song. However, using FreeRTOS compacts the code into one function called SpeakerTick. Similar to the previous labs, the speaker used timer 4, however we used a helper function to initialize it. To accurately calculate FFT, we use the Hamming window which is a signal processing technique to get rid of outliers generated by the abrupt beginnings or endings of sampled data. It smoothes out the abrupt transitions due to the data by a weighted curve that reduces outliers. It does that before the application of FFT, when the data is still in time- domain.

Part C

The project involves two Arduinos: one acts as the interface for a game displayed on an LCD, and the other acts as the player. The interface's game is controlled by a button press, while the player's Arduino triggers the button press using a servo based on light intensity sensed by a photoresistor. A buzzer on the interface can signal game over, reset by a sound sensor. Additional features include a jump counter on the player and potential speaker and sound sensor integration.

Code Documentation

Part B

This part is composed of 4 different tasks, some with 1 priority and other with background priority. The first task uses the function definition format with (void *pvParameters) indicating that the function accepts a single parameter. In FreeRTOS, a task in the setup defined using xTaskCreate() which can pass a pointer to tasks' parameters. The second task plays the Mario theme song for 4 rounds and pauses 1.5 sec between each song cycle. The code takes advantage of FreeRTOS's xTaskGetTickCount(), which returns the current count of ticks. Since the code for playing beats and melody was simple, the settings for OCR4A and frequency was implemented in the same function.

In the next two tasks, we illustrated an example of inter-task dependency. A few FreeRTOS queues were employed to facilitate the inter-task communication between task 3 and 4. The flow of execution goes from task 3 to 4 and returns to task 3 to print the desired output. In task 3, data array is generated with pseudo-random double values. These values are bounded between 100 and 200, using the random(), which is scaled by dividing it with RAND_MAX and then shifting the range saved in the data[]. Lastly, the xQueueSend function pushes the data onto fftQueue where it awaits consumption by task4. Once performing task4, it receives data from the fftQueue. Then, the task initializes the arduinoFFT object, from the Github library. The signal processing technique, Hamming window was applied. The time measurement before and after the FFT computation using the `xTaskGetTickCount()` was captured to measure how computationally intensive this process is. The elapsed time then send back to Task 3 through the completionQueue. Additionally, at the end of its execution, task 3 uses vTaskDelete(NULL) to free resources.

Part C
The file "Interface.ino" focuses on creating an interactive LCD display using the LiquidCrystal library and FreeRTOS. The code initializes the LiquidCrystal display and defines custom character bitmaps for a dino and a tree. It also sets up a button for user interaction. The main task "gameTask" is responsible for handling the game's interface and logic. It toggles between displaying the game menu and running the game loop based on user input. The "handleMenu" function displays the start menu and waits for the player to press the button to begin the game. The "handleGame" function orchestrates the game logic, controlling the movement of trees and the dino on the display. If a collision occurs, the "handleGameOver" function manages the game over scenario. Through these tasks and functions, the project showcases real-time updates and user interaction, contributing to a dynamic user experience.

In "Player.ino," the code sets up a system to read the light level from a connected light sensor and control an LED based on the detected light intensity. It initializes the sensor pin and LED pin, along with the threshold brightness level for LED activation. Two tasks, namely "sensorTask" and "ledTask," are created using FreeRTOS to handle reading the light sensor and controlling the LED, respectively. The "sensorTask" reads the analog value from the light sensor and displays it on the Serial Monitor. The "ledTask" checks the light level reading and turns on the LED if the light intensity exceeds the predefined threshold, demonstrating efficient multitasking and hardware interaction.

Overall Performance Summary
Please see the bonus video with narration here.

Teamwork Breakdown
This lab was completed in collaboration of Paria Naghavi and Anna Mai.

Discussion and Conclusions
Changing the mechanical component from a servo to manual playing because it did not have a percision and force required to press a button.

Reference

1. ChatGPT

2. C programming language documentations: https://en.cppreference.com/w/c/preprocessor/conditional
3. Lab provided resources.
4. Sev_seg library documentations
5. ArduinoFFT: https://github.com/kosme/arduinoFFT/tree/master