Real-Time and Embedded Systems

Project 04: Game

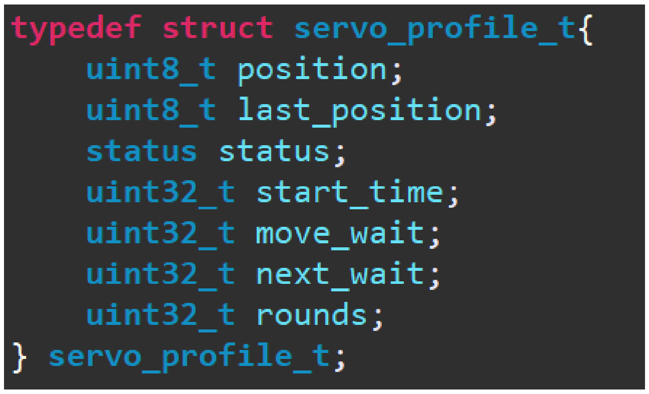
Author: Liam Sullivan
Email: lds7381@rit.edu
Submission Date: 11/04/2022

Analysis and Design

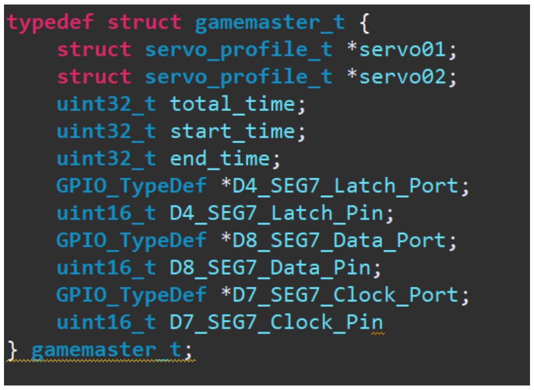
This project entailed the creation of a game on a STM32 microcontroller. This game was a servo position matching game where a user would try to match a servo arm's position to another's that was randomly chosen and moved to. Every time the lead servo (the servo that moves to a random position) moves to a new random position is the start of a new round. The round ends when the user servo (servo controlled by the user playing the game) gets to the lead servo's same position or the lead servo moves to a new position. The lead servo will always start a new round every 1-4 seconds regardless of whether the user servo has a matched position. The score is calculated by the time that the servos are not matching in the round. There are 5 rounds total in each one of the games. The user playing the game wants the lowest score they can get across 5 rounds of playing. The score of the game was also displayed on the STM32 microcontroller's shield for the user to see during and after the game. This game also contained similar needed functionality to project 2. Some of the functionality behind servo positioning and servo movement with PWM and timers were the same as described in the project 2 report.

To get the user input for moving the servo and begging the game, the STM32 microcontroller's shield that contained three separate buttons were used. Two of the buttons, buttons 1 and 3, were used to control the user servo's position. Button 1 was used to move the servo to the left one position and button 2 to the right one position. The third button, button 2, was used to start and restart the game.

To design this game on the STM32 microcontroller, FreeRTOS was an essential component. FreeRTOS tasks were used to create tasks for each of the servos and a game master task that would control the flow and functionality of the game. Each one of the servo tasks used a servo profile struct and the gamemaster task used a custom game master struct. These structs can be seen below in Figure 1.



```
typedef struct servo_profile_t{
    uint8_t position;
    uint8_t last_position;
    status status;
    uint32_t start_time;
    uint32_t move_wait;
    uint32_t next_wait;
    uint32_t rounds;
} servo_profile_t;
```



```
typedef struct gamemaster_t {
    struct servo_profile_t *servo01;
    struct servo_profile_t *servo02;
    uint32_t total_time;
    uint32_t start_time;
    uint32_t end_time;
    GPIO_TypeDef *D4_SEG7_Latch_Port;
    uint16_t D4_SEG7_Latch_Pin;
    GPIO_TypeDef *D8_SEG7_Data_Port;
    uint16_t D8_SEG7_Data_Pin;
    GPIO_TypeDef *D7_SEG7_Clock_Port;
    uint16_t D7_SEG7_Clock_Pin
} gamemaster_t;
```

Figure 1: Servo and Gamemaster Structs

These structures contained all of the necessary information for each of the servos to be properly controlled and for the game master to properly run the game and calculate and show the score. The lead servo task, on the start of the game, would get a random amount of time 1-4 minutes, wait that time, then move to a new random position (lead servo can not get the same random position two times in a row). Once in position it would then get another new random amount of time that would then start the next round. Each time the servo would move positions and be in position the status of that servo would be updated to signify the current status of the lead servo so the game master would know. The user servo task would control the servo position of the user servo. This task was a loop that would continually check buttons 1 and 3 that would move one position left and right. This task would also calculate the movement time of the servo from position to position along with updating the statuses of the lead servo. This task would only run during the game, the user servo could not move prior to the game being started or after it had ended. The game was determined to be over for both of the servo tasks when the amount of rounds exceeded five. The gamemaster task was used to check the servos positions and whether they match and calculate the score of the game. To get the score of the game the task would start counting time in milliseconds when the lead servo got into position at the start of a new round. When the user servo moved into the same position the timer would stop counting and add that count to a total score that was displayed to the seven segment display on the STM32 microcontroller's shield's seven segment display. The pointers to the GPIO ports that control the seven segment display were given in the gamemaster's struct seen in Figure 1. When the max rounds had been met the game master would continue to display the final score of the game. The task after the game would then check for the button 2 to be pressed to restart the game. It restarted the game by resetting the amount of rounds that had been performed, the total score of the game, and the servo positions back to zero. All three of these tasks running together successfully allowed the STM32 to use FreeRTOS to execute this game and properly measure the final score.

Another addition to this game that was added in was servo calibration. This is performed prior to every game that is being played. This is done to get the positions that the servo will move to during the game. This calibration ensures that the STM32 microcontroller will not send a PWM duty cycle signal to the servo that is too large or too small. This is done by having the user move the lead servo to the minimum duty cycle for the rightmost position that the servo can go to without strain in the gears. This is then done by having the user press the buttons 1 and 3 for left and right and increasing or decreasing the duty cycle by 0.5% each press. When both maximum and minimum duty cycles for the positions are found 6 distinct and equidistant positions are calculated for the positions to be used in the game. This function is also performed prior to any of the FreeRTOS tasks starting up.

Test Plan

One of the biggest tests when the game was first being played was making sure that the time that was being calculated for the score of the game was within a reasonable range. This was done by just letting the user servo arm never match with the random servo arm. This allows for the total amount for time that the game is running for to be displayed as that time. This time should be in a certain range that the random number generator creates. If that score is within that range on different games then the score and game time that is being used are correct. This range is between 5 and 20 seconds, or a score of 5,000 - 20,000. When performing this test on this project solution the resulting score was always between this range. This showed that the use of the timers and random round time generateration was implemented correctly.

Another test that was performed was the use of the seven segment display to update the score of the game exactly when the user servo matches the lead servo. This test showed that the timing for when the user servo gets exactly to a position is correct. If the seven segment does not update to the score that the user got at that exact time that the servo gets in position then the timing is wrong. When performing this test, the seven segment would always update at the correct time showing that the timing for when servos get into position was correct.

Project Results

The game that was created using the design talked about prior was successfully able to meet all requirements for this game. One of the biggest aspects of this game was the ability to be able to accurately measure time across multiple tasks in FreeRTOS. By designing the program to use statuses and positions to know when the servo was in an exact position and whether it has fully moved there was a big key to properly being able to measure this time. By sending the pointers of each of the servo structs to the gamemaster task, the game master was able to know these statuses and positioning. This meant that the gamemaster could know exactly when the servo moved to position and exactly when it got there. This allowed for the program to get the most accurate time possible for the amount of time that the lead servo and user servo's arms were not matching. Allowing the tasks to communicate by statuses and positions instead of times ultimately led to easier implementation and more accurate results.

The resulting game was able to successfully implement servos and displays with real time components with FreeRTOS. This overall made the game be able to run as soothing as it did and made it into the fun game that it is.

Lessons Learned

One of the biggest challenges during this project was the use of the seven segment display. Prior to this project I was unable to get the seven segment display running the way that I wanted to. However, after a lot of research it was discovered that the seven segment display had to be constantly being updated. This led to a display that was being updated every time the game master task was run and updating each segment one at a time. Prior to this project the display would not be constantly updated and would be incorrect. This led to a lot of resolved confusion about the seven segment display and will make future implementation of it much easier.

This project also led to a deeper understanding about how FreeRTOS is used and how to properly use it when trying to inference with physical peripherals such as servos. Prior to this project we had used FreeRTOS with timers but were waiting on other tasks to complete. This project we had to wait for the servos instead which led to a little bit of a different implementation with the FreeRTOS that was not used before. Any experience with FreeRTOS will only help understanding how to use it more and this project showed that.