SWEN 563

Real Time & Embedded Systems

Project 2A

Christopher Broderick *cjb9317@rit.edu*

Muhammad Adib Yahaya *my3197@rit.edu*

February 29, 2018

Overview

Using the STM32 discovery board we created a program that exhibited multitasking characteristics by simultaneously controlling two servo motors using custom control language. The program was responsive to outside user commands while still running a preset list of commands (recipe). This was executed by using PWM signals for each individual command that were then sent to the servo motors individually.

Areas of Focus

Muhammad:

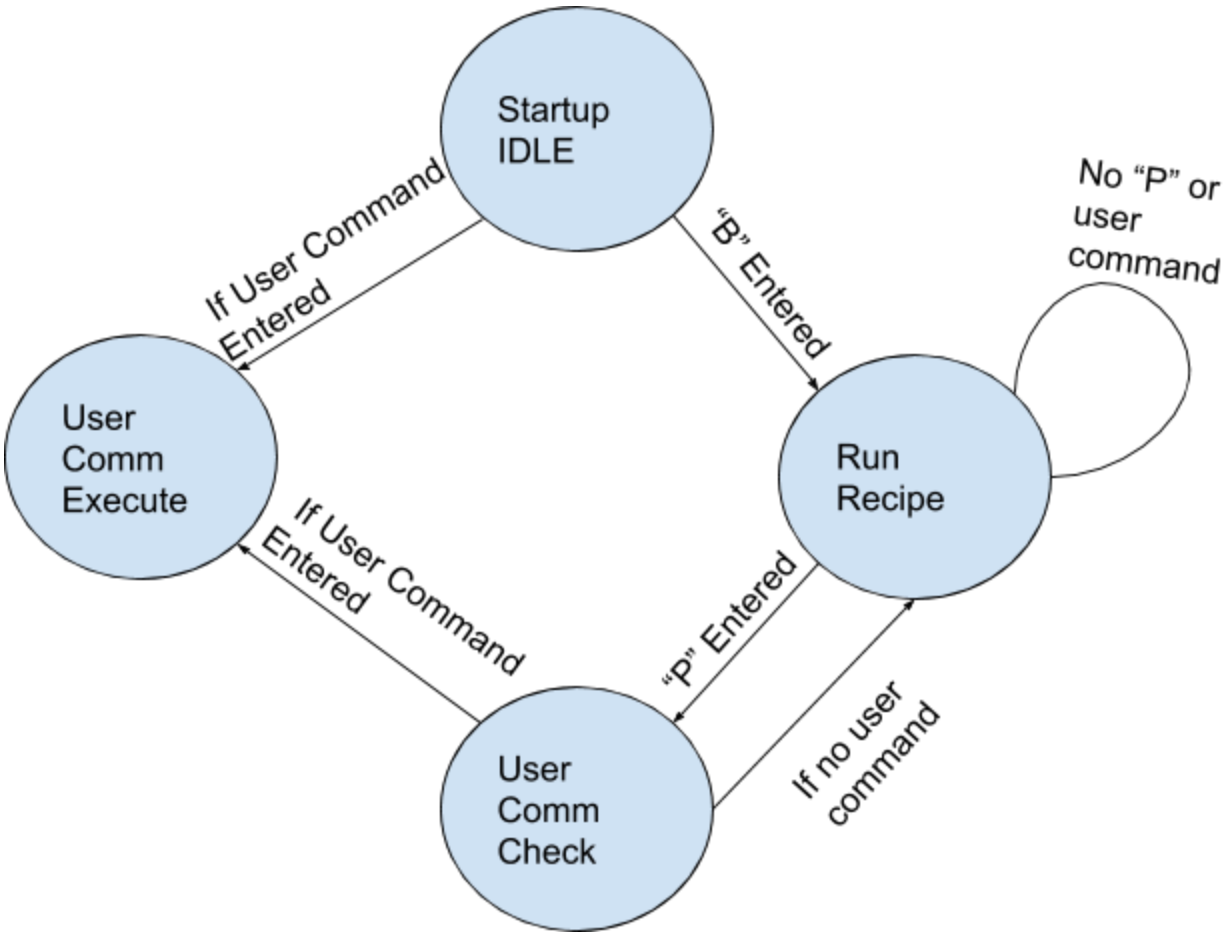
1. Determine what timers to use.
2. Create two PWM channels using timer2.
3. Set the pulse widths / duty cycle for move commands.
4. Review the duty cycle changes with an oscilloscope.
5. Develop the state machine.
6. Process valid user commands and display them on the terminal.
7. Use timer5 to generate delays every 100 ms and when moving servos.
8. Display servos' status using LEDs.
9. Test user commands on the Servos.
10. Test recipe commands with an oscilloscope and the servos.

Chris:

1. Process commands listed in recipes.
2. Develop the recipes to be run on each servo.

Analysis / Design

Flow chart:



1. The program requires at least three states in the state machine to properly move the servos and process user inputs almost simultaneously.
2. The three states are: Idle, run_recipe, and run_userCmd.
3. The program will run with the servos in 'paused' state. This is done to make the servos only move when command 'begin' is entered by the user.

4. When there is no user input, the program will run the commands listed in the recipes.
5. The only way to interrupt the recipe to run user commands is by entering ‘p’ or ‘pause’ to the servo.
6. For every servo position, there will be 200 ms delay to allow the servo to complete a move.
7. Every 100 ms, there will be a short interrupt to allow user to type in commands.
8. Each servo has its own data structure to hold information such as current position, state, command to be executed, current recipe index, and etc.
9. The program will determine which recipe or user input to run by fetching the appropriate data in the structure.

Test Plan

1. Use timer2 to implement pulse width modulation (PWM).
2. Change the pulse width to vary the duty cycle, thus changing the servo's position.
3. Enable channel two of timer2 with PWM mode to control two servo simultaneously.
4. Hook the STM32 board with an oscilloscope to view the waveforms and verify PWM is working.
5. Modify USART_Read() function to be non-blocking.
6. Enter some commands on the terminal while the program is running multiple functions. If successful, then (5) is functioning correctly.
7. Insert the recipe arrays with all possible commands (MOV, WAIT, LOOP, END_LOOP, RECIPE_END).
8. Hook the STM32 board with an oscilloscope and then the servos to view the waveforms and verify recipe commands are executed correctly.

Project Results

PWM mode was successfully implemented on the STM32 board using two output channels on timer2.

When connected to the servos, both servos were able to change their positions according to the recipes or the user commands.

Two onboard green and red LEDs would change depending on the servo's status. Recipe command error LED (red) however, did not light up when the program encountered an invalid state.

There were two major problems. The first problem was the delay. We implemented a blocking delay that interfered the multitasking nature of this project.

The second problem was the recipe's WAIT command. Since we used a similar delay function to implement the WAIT command, it did not work as expected. Three successive WAIT command did not pause the recipe for ~9 seconds.

Beside the problems mentioned above, all other functionalities worked as expected.

Lessons Learned

In this project, we learned that having a data structure to hold servo information is crucial to avoid confusion when writing the program. When we first started this project, the program we wrote did not use structure. So it was difficult to keep track of what state the servo is in and in what position it is currently at.

After 2 weeks, we decided to rewrite the program from scratch because fixing the original code would take too much time and effort. And debugging would also be hard because the code was not organized.

The project went smoother after the code rewrite. Debugging was also easier because of the better code readability.

We also learned more about bitwise operators and bit shifting in this project. In decoding the recipe commands, both of these methods were used to get opcode and parameters of the commands.

After doing the demo of this project, we did have a few problems. The major problem was related to the way we program the master interrupt / delay. We implemented a blocking delay that caused the program to not be a ‘true’ multitasking program. If we had noticed this issue earlier, we could have fixed it.