CMPE 663 Project 2

Servo Control

Author: Gloria Mbaka

Email: gm6629@rit.edu

Submitted: 10/07/24

Instructor: Prof. Wolfe

TA: Bertola Thomas

Analysis/Design

Project Overview

This project was developed to create a servo control system capable of managing two servo motors simultaneously, using both user inputs and predefined recipes. The system relied on Pulse Width Modulation (PWM) signals generated by an STM32 microcontroller to control the positioning and movement of the servo motors. The project incorporated various commands, including reversing, pausing, continuing, and recipe-based movements. The primary focus was to explore embedded systems design principles and learn how to manage peripherals, in this case, the servos, using real-time commands and state-based logic. The key objective of the project was to implement both user-controlled and recipe-based operations, allowing the servos to either operate autonomously according to the recipes or respond to direct user inputs. To manage this effectively, interrupts, state machines, and timers were used to control the movement of the servos and ensure responsive and smooth operation. Error handling was also a critical objective, as the system needed to detect invalid commands or states and provide immediate feedback to the user. Each servo could move to any of six defined positions, ranging from 0 to 5. A set of user commands was implemented, including pause, continue, reverse, move left, and move right, all communicated via UART, giving users direct control over the system. Additionally, the system can handle errors such that if an invalid opcode was encountered during recipe execution, the system would transition into an error state. To make the system's status clear to the user, LEDs were used to indicate the status of each servo, whether it was running, paused, or had encountered an error. The system used Timer 4 to trigger regular state updates and check the progress of the recipe execution, ensuring that everything functioned efficiently. Timer 2 was responsible for generating the PWM signals necessary for controlling the servo motors.

Hardware Design

STM32 Nucleo Board

The STM32 Nucleo L476RG board was used to implement this project. The following peripherals were configured:

- TIM2 (Timer 2)**:** Used for generating PWM signals to control the position of the servos.
- TIM4 (Timer 4**)**: Used to generate periodic interrupts that trigger servo state updates and recipe execution.
- USART2 (UART)**:** Handles serial communication with the user for sending commands and receiving feedback.
- GPIO Pins**:** PWM outputs are configured on GPIO pins connected to the servos, and onboard LEDs indicate servo status.

Timer 2 (PWM Control)

- Prescaler**:** The timer's clock is derived from the system clock (80 MHz), and the prescaler was set to 79 to reduce the clock to 1 MHz.

- ARR (Auto Reload Register): The ARR was set to 20,000 to achieve a 50 Hz PWM signal (20 ms period) required to control the servos.

Timer 4 (Interrupts)

- Purpose: Timer 4 was configured to generate periodic interrupts that occur every 100 ms to ensure the state machine executes in real time and recipes are updated on schedule.
- Setup: The prescaler for Timer 4 was configured to generate interrupts at a frequency of 10 Hz (100 ms intervals).

Servos

Two MG90S Micro Servo Motors were connected to the PWM outputs. These servo motors have a range of 0 to 180 degrees, and their positions were controlled by adjusting the duty cycle of the PWM signals.

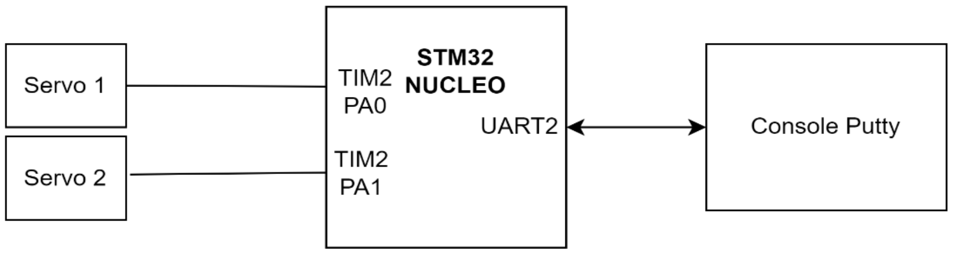


Figure 1: Hardware Block Diagram

Software Design Approach

Functionality:

The main components of this servo control system include PWM generation, state machine control, command processing, and interrupt handling.

PWM Generation (TIM2) was used to generate the PWM signal that controls the servo motors. The servo's position was adjusted by changing the duty cycle of the PWM signal, allowing precise movement between different positions.

A state machine manages the servo's behavior, with the system cycling through different states:

- state_moving: The servo is actively moving between positions.
- state_paused: The servo movement is temporarily halted.
- state_error: An invalid command or recipe has been encountered, requiring user intervention.
- state_waiting: The servo is in a waiting period as specified by the recipe instructions.
- state_recipe_ended: The servo has completed all the commands in the current recipe.

Command processing occurs via UART, where the user can send real-time instructions such as Pause (P), Continue (C), Move Right (R), Move Left (L), Begin (B), or Reverse (V). The system immediately interprets these commands and adjusts the servo behavior accordingly.

Interrupt handling is managed using Timer 4 (TIM4), which generates interrupts every 100 milliseconds to update the servo's state and execute the next recipe instruction on time.

Software Flow:

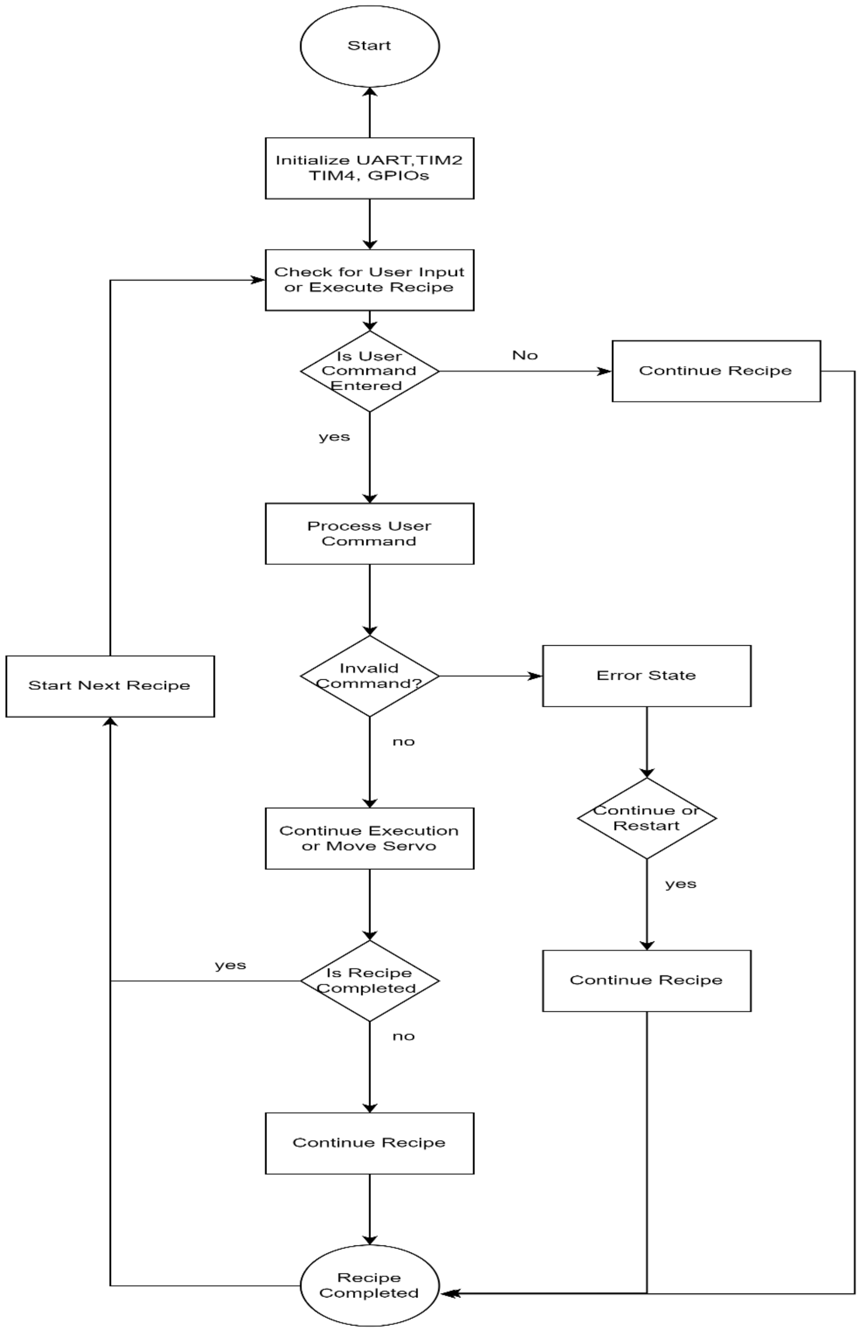


Figure 2: Software Design Flow Chart

Test Plan

The system was tested in several phases to ensure correct functionality:

1. Recipe Execution**:** Various predefined recipes were loaded, and the servos were observed to ensure they moved according to the commands.
2. User Command Testing: All available user commands (Pause, Continue, Reverse, Move Right/Left) were tested to verify they controlled the servos as expected.
3. Error Handling**:** Invalid opcodes were injected to ensure the system entered the error state and notified the user.

Calculations

Step 1: Prescaler and ARR Calculation for PWM

The STM32 Nucleo board runs at 80 MHz, and it was needed to configure Timer 2 to generate a PWM signal at 50 Hz (period = 20 ms) to control the servo motors.

- Prescaler Calculation**:**

$$Prescaler = \left(\frac{80{,}000{,}000}{1{,}000{,}000}\right) - 1 = 79$$

  The prescaler was set to 79 to slow down the clock to 1 MHz.

- ARR Calculation**:**

$$ARR = \frac{1{,}000{,}000}{50} = 20{,}000$$

  The ARR was set to 19999 to achieve the 50 Hz PWM frequency.

Step 2: Duty Cycle Calculation

The duty cycle determines the servo's position, and the formula used is:

$$Duty\ Cycle = 500 + (position \times 300)$$

Where the position is mapped to a range between 0 (extreme right) and 5 (extreme left).

1. Minimum Position (0°)**:**

   Timer Counts = 0.5 ms / 1 µs = 500

$$CCR = 500(at\ 0.5\ ms\ pulse\ width)$$

2. Maximum Position (180°)**:**

$$CCR = 2000(at\ 2.0\ ms\ pulse\ width)$$

3. Intermediate Positions**:**
   The intermediate positions are calculated linearly, with the duty cycle increasing by 300 timer counts for each step from position 0 to 5.

Table 1: Calculated Values for the Positions

Position	Angle (Degrees)	Pulse Width (ms)	Timer Count (CCR)
0	0	0.5	500
1	36	0.8	800
2	72	1.1	1100
3	108	1.4	1400
4	144	1.7	1700
5	180	2.0	2000

Project Results

Recipe Execution

The system successfully executed multiple recipes sequentially. Each servo moved to the correct positions based on the recipe instructions, and the system provided feedback for every command.

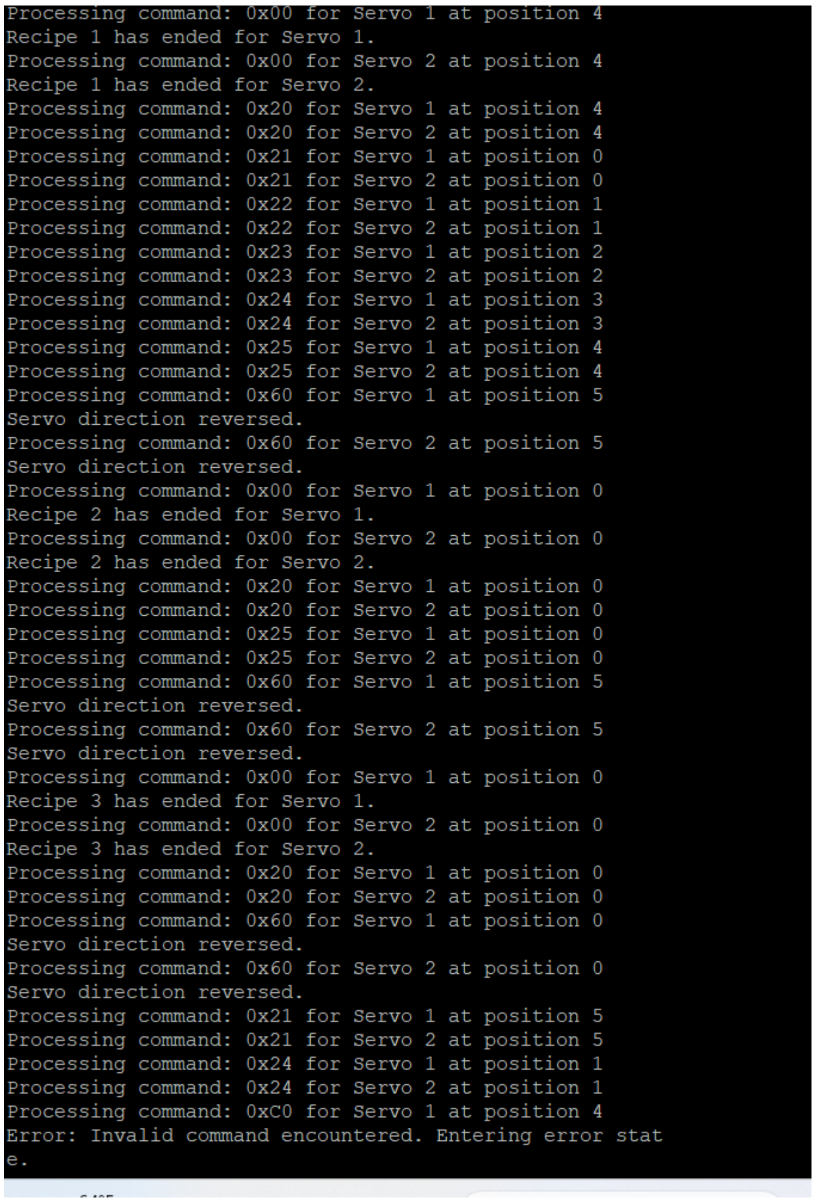
Error Handling

Invalid commands such as 0xC0 and nested loops caused the system to enter an error state, as expected. The servos stopped moving, and an error message was transmitted to the user.

User Command Execution

Commands such as reverse, pause, and continue were successfully processed. For example, sending VV moved both servos to the opposite extreme, while CC resumed the recipes from where they paused.

The result from the putty during testing is as shown in Figure 3.



```
Processing command: 0x00 for Servo 1 at position 4
Recipe 1 has ended for Servo 1.
Processing command: 0x00 for Servo 2 at position 4
Recipe 1 has ended for Servo 2.
Processing command: 0x20 for Servo 1 at position 4
Processing command: 0x20 for Servo 2 at position 4
Processing command: 0x21 for Servo 1 at position 0
Processing command: 0x21 for Servo 2 at position 0
Processing command: 0x22 for Servo 1 at position 1
Processing command: 0x22 for Servo 2 at position 1
Processing command: 0x23 for Servo 1 at position 2
Processing command: 0x23 for Servo 2 at position 2
Processing command: 0x24 for Servo 1 at position 3
Processing command: 0x24 for Servo 2 at position 3
Processing command: 0x25 for Servo 1 at position 4
Processing command: 0x25 for Servo 2 at position 4
Processing command: 0x60 for Servo 1 at position 5
Servo direction reversed.
Processing command: 0x60 for Servo 2 at position 5
Servo direction reversed.
Processing command: 0x00 for Servo 1 at position 0
Recipe 2 has ended for Servo 1.
Processing command: 0x00 for Servo 2 at position 0
Recipe 2 has ended for Servo 2.
Processing command: 0x20 for Servo 1 at position 0
Processing command: 0x20 for Servo 2 at position 0
Processing command: 0x25 for Servo 1 at position 0
Processing command: 0x25 for Servo 2 at position 0
Processing command: 0x60 for Servo 1 at position 5
Servo direction reversed.
Processing command: 0x60 for Servo 2 at position 5
Servo direction reversed.
Processing command: 0x00 for Servo 1 at position 0
Recipe 3 has ended for Servo 1.
Processing command: 0x00 for Servo 2 at position 0
Recipe 3 has ended for Servo 2.
Processing command: 0x20 for Servo 1 at position 0
Processing command: 0x20 for Servo 2 at position 0
Processing command: 0x60 for Servo 1 at position 0
Servo direction reversed.
Processing command: 0x60 for Servo 2 at position 0
Servo direction reversed.
Processing command: 0x21 for Servo 1 at position 5
Processing command: 0x21 for Servo 2 at position 5
Processing command: 0x24 for Servo 1 at position 1
Processing command: 0x24 for Servo 2 at position 1
Processing command: 0xC0 for Servo 1 at position 4
Error: Invalid command encountered. Entering error stat
e.
```

Figure 3: Test Result from Recipes

Lessons Learnt

1. It was learned that implementing a state machine is essential for efficiently managing servo operations, including movement, pausing, and handling errors. PWM signals were used to control the servo motors, and it was understood that adjusting the duty cycle changes the motor's position accurately.

2. The importance of real-time updates was highlighted through the use of Timer 4 for interrupt handling, ensuring that recipes executed without unnecessary delays or blocking..

3. It was also learned that proper error handling is critical, especially when managing invalid commands or issues in recipe execution. Integrating various components like timers, interrupts, PWM, and UART helped showcase the value of combining hardware and software elements to design a robust embedded system.

Challenges Encountered

Some challenges were encountered during the project. There were difficulties managing state transitions in the state machine, especially when handling commands like pause, continue, and error states. Ensuring smooth servo movements with accurate PWM signals was challenging, particularly when calculating duty cycles for different positions. Implementing the reverse command without interfering with normal recipe execution was also tricky. Another challenge was ensuring the recipe continued correctly after an error, either restarting or resuming based on user input.

Conclusion

This project successfully demonstrated the ability to control two servo motors using recipes and user inputs on an STM32 platform. Timer interrupts, PWM signals, and UART communication were used to create a dynamic system that responded to real-time commands and handled multiple servo states.