Embedded Systems
Project 2a

Date of Submission: March 8, 2019

Joel Yuhas : jxy8307@rit.edu
Athaxes Alexandre : axa2012@rit.edu

Overview

The STM32 Discovery board is a microcontroller capable of running “bare-metal(no operating system)” programs. This project interfaced with the STM32’s PWM interface in order to drive two servo motors independently. Software had to be written for the STM that allowed for a program to take and run two “recipes” independently, which dicated how the servo motors would move. The program also allowed for a user to input commands to the program, and have the program react to those commands accordingly. Overall, all of the desired objectives were met and the project was a success.

Areas of Focus

Project Assignments	Team Member
PWM and Timer Initialization	Joel + Athaxes
Recipe Input and Execution	Joel + Athaxes
User Input Features	Joel + Athaxes
Report	Joel + Athaxes

Analysis/Design

The main objective of the program was to read a built in “recipe” that dicated how a servo motor would move. The servo was set up in a way that let the motor rotate into 6 distinctive positions, (0 - 5). Figure 1 shows the figure from the lab detailing the servo layout.

Figure 1: Servo Layout

In order to get the servo to move to specific positions, the PWM had to be initialized on the STM board. Timer 2 (TIM2) was used for servo 1 while timer 5 (TIM5) was used for servo 2. Both of these timers had PWM features. The main initialization was done by using the STM CUBE application. This program set up the PWM with the desired inputs. For this exercise, it was desired to have a period of 20ms and a pule between 0.4ms and 2 ms. The default clock was set to 80 MHz. The ARR value was set to 12499 and a prescaler of 127 was used. These values were chosen because they were recommended by one of the books located in the classroom. The equations below were gathered from in class materials and show how the pulse values were calculated.

$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$

$$\text{Period} = (1 + \text{ARR}) * \text{Clock Period}$$

$$\text{Duty Cycle} = \frac{\text{CCR}}{\text{ARR} + 1}$$

$$\frac{80\ \text{MHz}}{127 + 1} = 625{,}000\ \text{Hz} \quad (1)$$

$$20\text{ms} = (1 + \text{ARR}) * 625\text{KHz} \quad (2)$$

$$\text{ARR} = 12499 \quad (3)$$

$$\frac{2\text{ms}}{20\text{ms}} = 10\% \quad (4)$$

$$\text{Duty Cycle} = \frac{\text{CCR}}{12499 + 1} \quad (5)$$

$$\text{CCR} = 1250 \quad (6)$$

The recipes that the program received were broken down into 5 commands. The commands were 8 bits in length, with the first 3 bits being the “opcode” or the command, the next 5 being the command parameter. Table 1 shows the layout of the recipe commands:

Table 1: Recipe Commands

Mnemonic	Opcode	Parameter	Range	Comments
MOV	001	The target position number	0..5	An out of range parameter value produces an error.
WAIT	010	The number of 1/10 seconds to delay before attempting to execute next recipe command	0..31	The actual delay will be 1/10 second more than the parameter value..
LOOP	100	The number of additional times to execute the following recipe block	0..31	A parameter value of “n” will execute the block once but will repeat it “n” more times.
END_LOOP	101	N/A		Marks the end of a recipe loop block.
RECIPE_END	000	N/A		
WAVE	110	N/A		Moves the servo between position 4 and 5, 6 times.

The program was designed to receive a list of these commands and execute them in the order that they appeared. Since there were two servo motors, the program had to be designed to run two recipes at the same time.

Additionally, the program was designed to receive user commands. The user would input two letters, each letter corresponding to a specific user command and the position of the letter (1st letter or 2nd letter) would determine what servo it would effect. Table 2 shows the list of commands available to the user

Table 2. User commands

Command	Mnemonic	Comments
Pause Recipe execution	P or p	Not operative after recipe end or error
Continue Recipe execution	C or c	Not operative after recipe end or error
Move 1 position to the right if possible	R or r	Not operative if recipe isn't paused or at extreme right position
Move 1 position to the left if possible	L or l	Not operative if recipe isn't paused or at extreme left position
No-op no new override entered for selected servo	N or n	Nothing happens here
Begin or Restart the recipe	B or b	Starts the recipe's execution immediately

Lastly, the program indicated its status through the red and green LEDs on the STM board. Table 3 shows the following statuses and associated colors.

Table 3. Status LEDS

Status	Indication
Recipe Running	Only the green LED is on.
Recipe Paused	Both LEDs are off.
Recipe Command Error	The red LED is on.
Recipe Nested Loop Error	The red and green LEDs are both on.

The code was designed to run a constant while loop that lasted 100ms each. Within that 100ms, the program would check for user commands, if there were no user commands then it would read the opcode from the recipe, and execute the opcode (if it has not already started executing it). Because the program was designed within a while loop, many of the variables had to be set globally or at least within the main function, outside of the while loop. This allowed values for certain functions to carry over into the next executions.

Additionally, whenever certain recipes were executed, the program would have to wait a set amount of time for the command to finish. Every MOV command waited 10 loops (1 second) to finish execution while each WAIT iteration lasted 1 loop (100ms). The code below shows how the main while loop was broken down into its separate functions.

```
  while (1)
  {

      realTimeUART();        // check for user input
      servoHandle();          // Recipe and Servo 1 update
     servoHandle1(recipe5);  // Recipe and Servo 2 update
      LEDChecker();           // LED status update

      // Waiting 100ms
      time_stop = HAL_GetTick() + 100;
      while( HAL_GetTick() < time_stop){}

      }
}
```

It should be noted that the approach taken when designing the code was to create two separate functions for each servo (servoHandle() for servo 1, servoHandle1() for servo 2). This was done because when the project was first started it was designed for 1 servo in order to ensure that it would function correctly. Once the first servo was working, since only two were required, it was easier to copy the function and manually input the new PWM initialization and counters than to try to recreate the procedure to make it reusable. A more robust way to have coded would have been to create only 1 function that could take in and initialize its own parameters for each servo, to save space and organize the code. However doing so would of required more time and testing without changing the final, functional results.

Finally, an additional command was created as part of the graduate student extension. It should be noted that it was found out after implementing the grad extension that no one on the team actually was in the grad portion of the class, so the implementation wasn't necessary. However, the command that was implemented was the WAVE command, which made the servo move rapidly between position 4 and 5 in a waving like motion.

Test Plan

In order to ensure that the program was functioning correctly and would not break the servo, a mandatory PWM test was executed on the oscilloscope. This test measured the period, duty cycle, and max voltage to ensure that it was within the bounds.

A provided test recipe was included in the lab pacet. This recipe was to be the main test that was ran. Figure 2 shows the implemented recipe. It should be noted that one of the requirements to ensure that the timing was correct was that the WAIT commands seen in the 1st servo test should have lasted approximately 9.3s.

```
MOV 0
MOV 5
MOV 0
MOV 3
LOOP 0
MOV 1
MOV 4
END_LOOP
MOV 0
MOV 2
WAIT 0
MOV 2
MOV 3
WAIT 31
WAIT 31
WAIT 31
MOV 4
RECIPIE_END
```

Figure 2. Test Recipe.

Additionally, the two servos had to clearly run two different recipes at a different time in order to ensure that they were indeed operating independently. The recipe below in figure 3 shows the other test that was ran.

MOV 1
MOV 0
LOOP 3
MOV 5
MOV 4
END_LOOP
RECIPIE_END

Figure 3. Servo 2 Test Recipe

All of the user commands were tested as well, in order to ensure that the user commands worked consistently and reliably.

Finally, the status LED's statuses needed to be tested. The running status was tested when running the original recipe. The pause status was tested when entering the user commands. The command error was tested by entering commands that were not on the list of commands. The nested error loop was tested by providing the program with a recipe with a nested loop.

Results

Since this lab involved the use of physical servo motors, all of the results from this exercise came in the form of visual observation by the TA. Even though the UART terminal was used, it simply was used to input results and did not output any.

Nevertheless, all of the test described in the Test Plan were run and observed by the TA, and all of the functionalities and features were deemed to be a success.

Lessons Learned

Overall, this exercise went very smoothly and produced the expected results. The decision to use CUBE to write the initialization code proved to be tough in the beginning, but after the learning curve of using CUBE was surpassed and a better understanding of its functionalities were achieved, the CUBE process and code writing went very smoothly.

There was trouble getting the UART system to work seamlessly while running the opcodes. Some of this trouble was from understanding how the CUBE HAL commands handled the UART transmit and receiving. However this issue was quickly fixed after realizing more details about the HAL time delay and buffer sizes.

Additionally, here was also some confusion initially about how to program was to be implemented. At first it wasn't clear if the lab wanted the primary input to be from the user with

the recipes as a secondary backup, or if the opcodes were to be the main focus. After that was clarified, the rest of the coding and execution went very smoothly.

Lastly, one of the biggest things that was learned was that it would have been better to have implemented the servo function to be more reasuabel with the other servos. Since the functions were not reusable, there ended up being a large amount of code that did similar things. However, at the point when the code would of been re-written, it was already at the point when it was easier to copy the procedures rather than basically start over.

Overall the exercise was as tremendous success and was able to meet all of the requirements set out by the lab manual.