SWEN-563/CMPE-663/EEEE-663

Real-Time and Embedded Systems

Project 2a - April 19, 2017

Zachary Weeden | zdw7287@rit.edu

Dinesh Bashkaran | dab8730@rit.edu

Overview:

We were given the task of creating a pulse width modulation signal on the STM board to move a pair of supplied servos to designated positions. The user could control the behavior of the servos independently of one another. Another facet of this project was the inclusion of recipes. Recipes were a pseudo command language for the servos in which a translation of byte could be interpreted as various actions. The top 3 bits of the recipe determined the operation and the remaining 5 lower bits was the parameter for that particular recipes operation. This allowed for quick testing of functionality as you could specify a command/operation with a parameter and could see if the results were as to be expected. A UI interfacing with the serial com port was also implemented for user input movements to be carried out. User input movements included left, right, none, continuing, pausing and restarting the recipe snippet.

Areas of Focus:

Zachary Weeden: Responsible for PWM signal, recipe processing, UI and report
Dinesh Bashkaran: Servo behavior, Demo
(see https://github.com/zweed4u/Real-Time-and-Embedded-Systems/)

Analysis/Design:

This program is primarily comprised of 3 source files; timer and GPIO functionality, servo structure as well as recipe declaration and the top which includes main. In this top includes the parameterized functions to move a passed motor to a passed position and parsing/determination of servo behavior based on recipe as well as user input.

In the timer source file, PA0 and PA1 GPIO pins are enabled for the pulse width modulation signal to be output upon. A delay function is also defined as to allow the servo proper time between movements. The passed parameter is the milliseconds needed for the delay which is calculated by the number of positions moved by a motor * 200, as it takes 200ms to move 1 position. This can be seen in the top file.

The servo file makes use of a structure in which we can declare attributes and data for a servo. This servo model is used in the top file with its attributes constantly updated to match and represent the current state and data sent to each motor. Test recipe snippets are also declared here. Some snippets include the required test as defined in the project document as well as a sweeping move in which all positions are tests for both servos moving right to left then all the back to origin. Another test included is to test the override command with a faulty recipe with a preemptive RECIPE_END and lastly a faulty recipe to test the parameter boundaries of a given operation (MOV in this case).

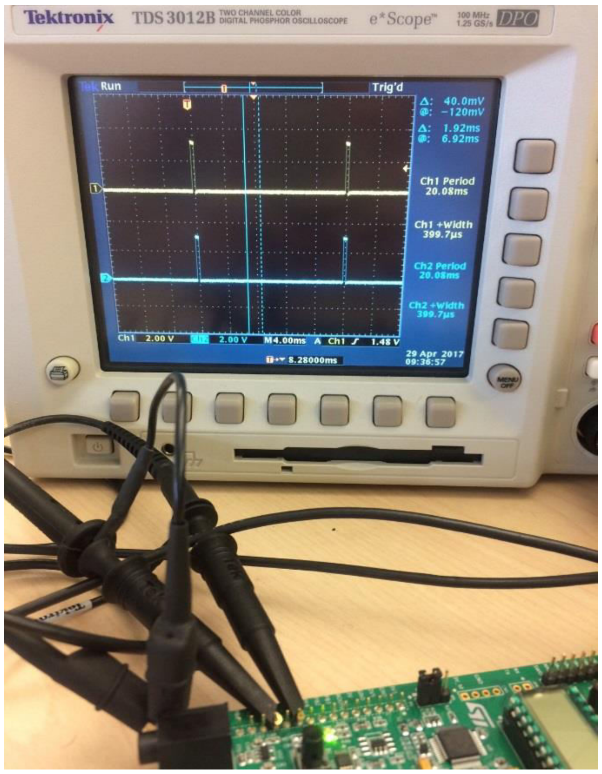
The top file contains the basic logic and the main here is responsible for the flow control of the program. After system/peripheral initializations an indefinite loop polls for user input from the UART. A character buffer array is then populated with null terminators to act as placeholders. This makes it easy to determine the end of the user input command. A carriage return matching condition is then used to determine if the user has 'entered' the command. Another condition must be met as to ensure that the 'x' key has not been entered and if so, the command is negated and the user is prompted again for command. Each proper element from the buffer is then passed into a function where the 2 servo's behavior is determined based on a case statement matching on char. Each servo structure's data is updated accordingly and movement occurs if the command entered is either 'l', 'r' or 'b' and 'c' if the recipe declared involves movement. A check on the position is used prior to movement to ensure that we don't move left or right and go out of bounds if we are already at the left most or rightmost position. Because 'b' is defined as restarting the recipe we make sure to update the index of the recipe to the first element. If the user entered command involves movement and a position change, the delay function declared in the timer source is made use of. Because a user can only specify 'l' or 'r' without directly stating how many positions to be moved, a delay of 200ms is sufficient.

Running alongside this function is another function to step through the recipe that is declared in the servo source and determine appropriate servo behavior whilst updating each servo model's data along the way. With the help of bit masking and another case statement, we can easily determine what the element of the recipe array entails. The top 3 bits, operation are 'cased' upon and behavior is determined further by the parameter for each bit masked opcode. For instance, in the MOV case, the parameter (lower 5 bits) of the recipe command are checked to be less than 5 otherwise alert with a red led indicating an error has occurred. The position is current position is kept track of as well as the difference of the parameter so we know how long we should delay to allow the servo time to move and complete the recipe. The index of the recipe array is incremented to move along with the recipe snippet processing. The wait parameter is parsed in the wait cased and makes use of the delay function. The loop case first ensures that the servo is not already in a loop state as nested loops are not permissible. This flag is toggled when this case is initially hit. The index of the beginning loop command is noted and is needed to determine where in the recipe to loop as well as when the loop finally ends. The recipe end case marks the completion of the recipe snippet.

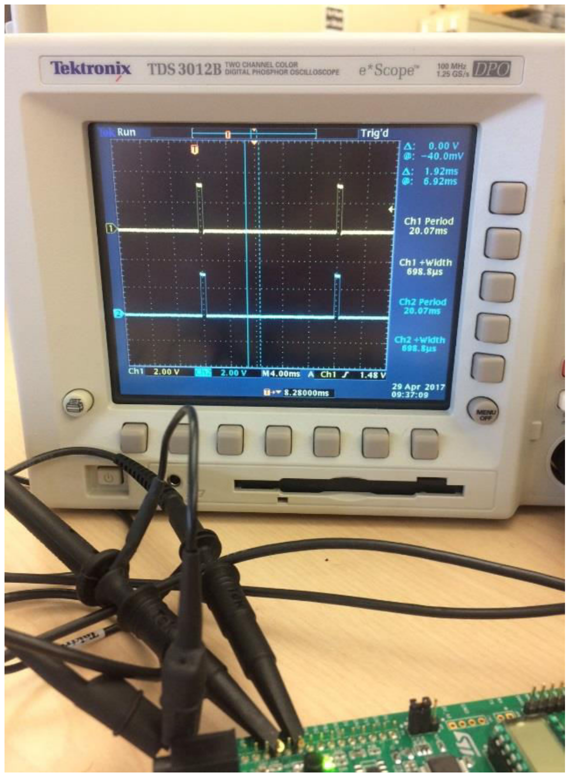
Also glossed over are the values used for compare capture register defined in the servo file. This is an array of numbers that represent the duty cycle/positive pulse width for servo behavior. These are used to determine the duty required to hold the servos at a certain position. Below in the test plan section are screenshots at each position that exhibit each CCR value.

Test Plan:

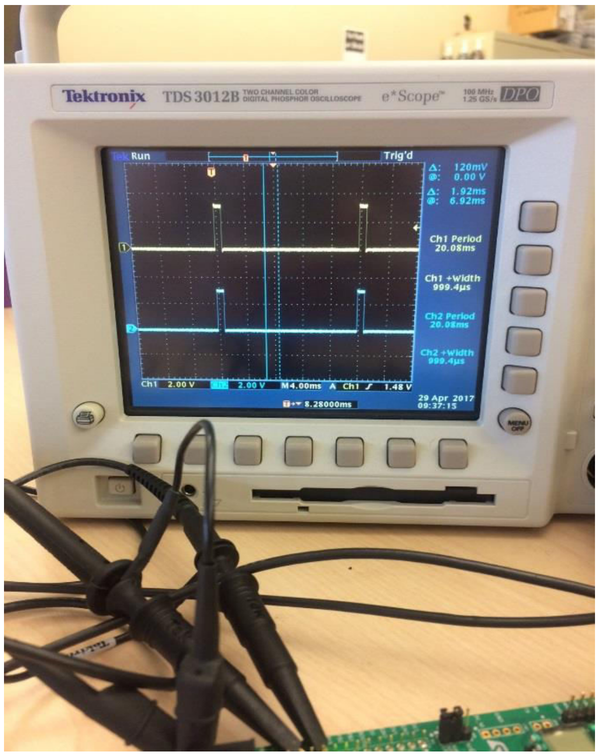
Because it was pivotal that the pulse width modulation signal was clean as to not cause hardware failure as well as correct servo functionality, I made sure with an oscilloscope that the signal on PA0 and PA1 was correct and updated correctly on user input in PUTTY without the servos. On the oscilloscope I made use of the measure tool for period and positive duty cycle for each pin.



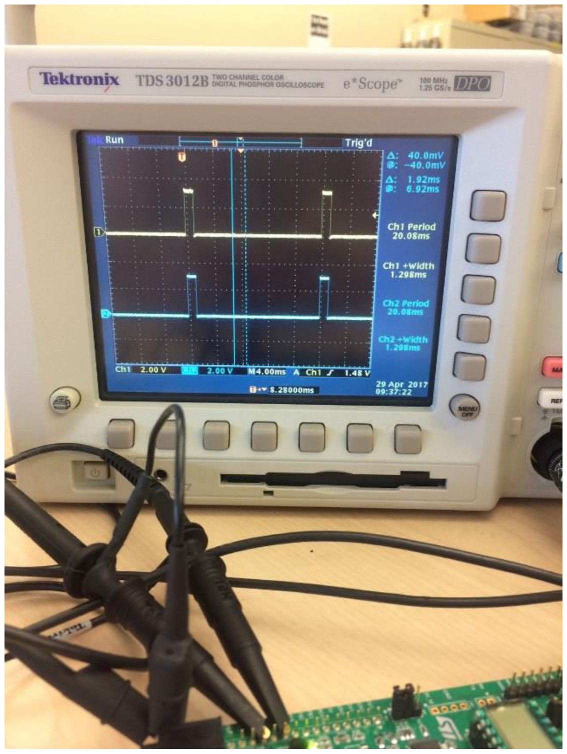
CCRx=4 shown effect on PA0 and PA1 (399.7 micro)



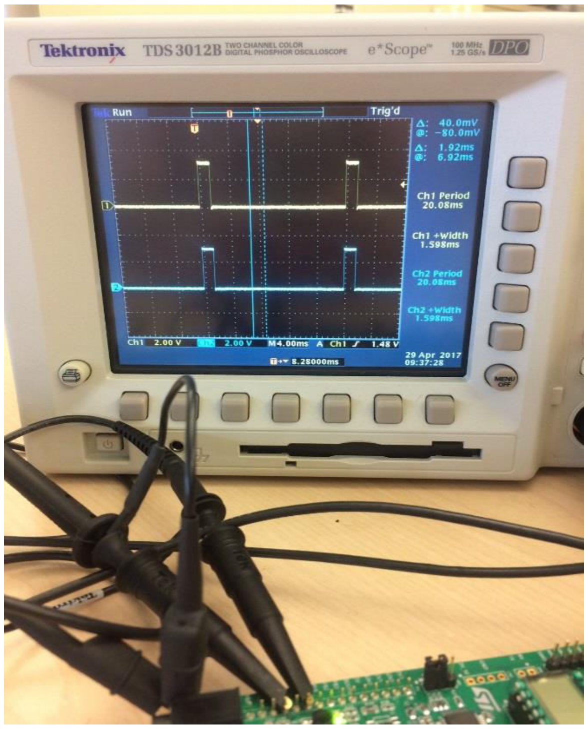
CCRx=7 shown effect on PA0 and PA1 (698.8 micro)

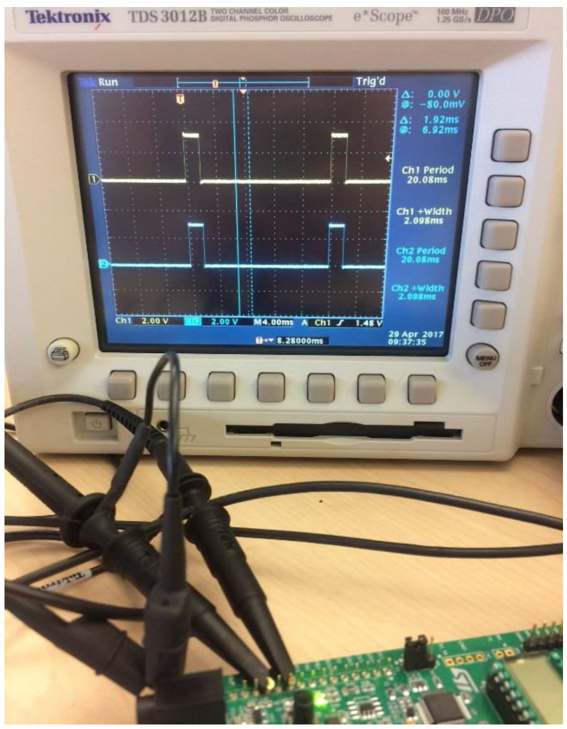


CCRx=10 shown effect on PA0 and PA1 (999.4 micro)



CCRx=13 shown effect on PA0 and PA1 (1.298 milli)

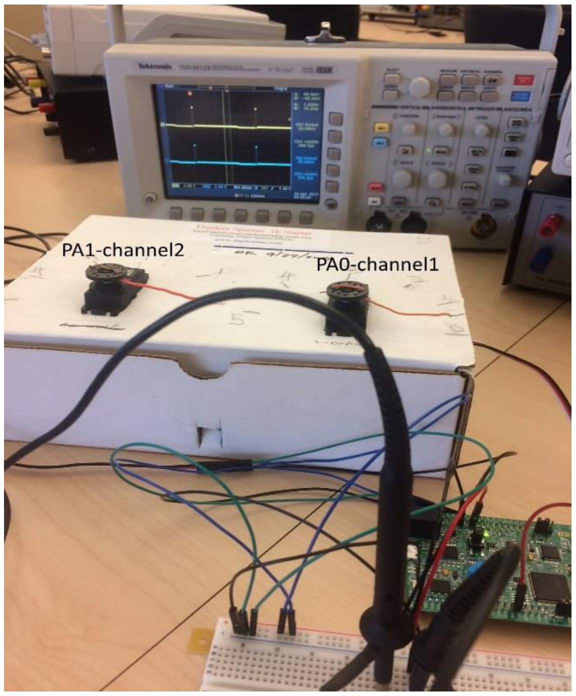


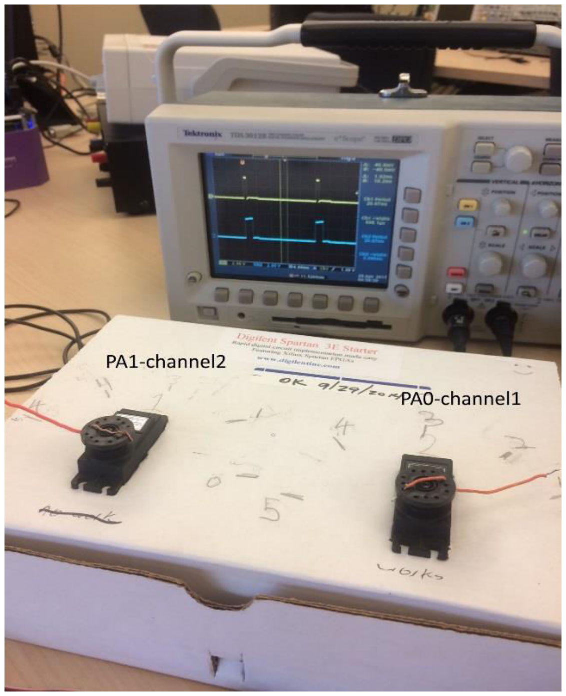


CCRx=16 shown effect on PA0 and PA1 (1.598 milli)

CCRx=21 shown effect on PA0 and PA1 (2.098 milli)

Once it was determined that the signal was correctly output and 'obeyed' user input it was then time to introduce the servos. With the use of a breadboard with 6 male-male wires for ground, power, signal connections for each servo and 4 male-female wires for STM connection pins PA0, PA1, GND and 5V the system was complete.

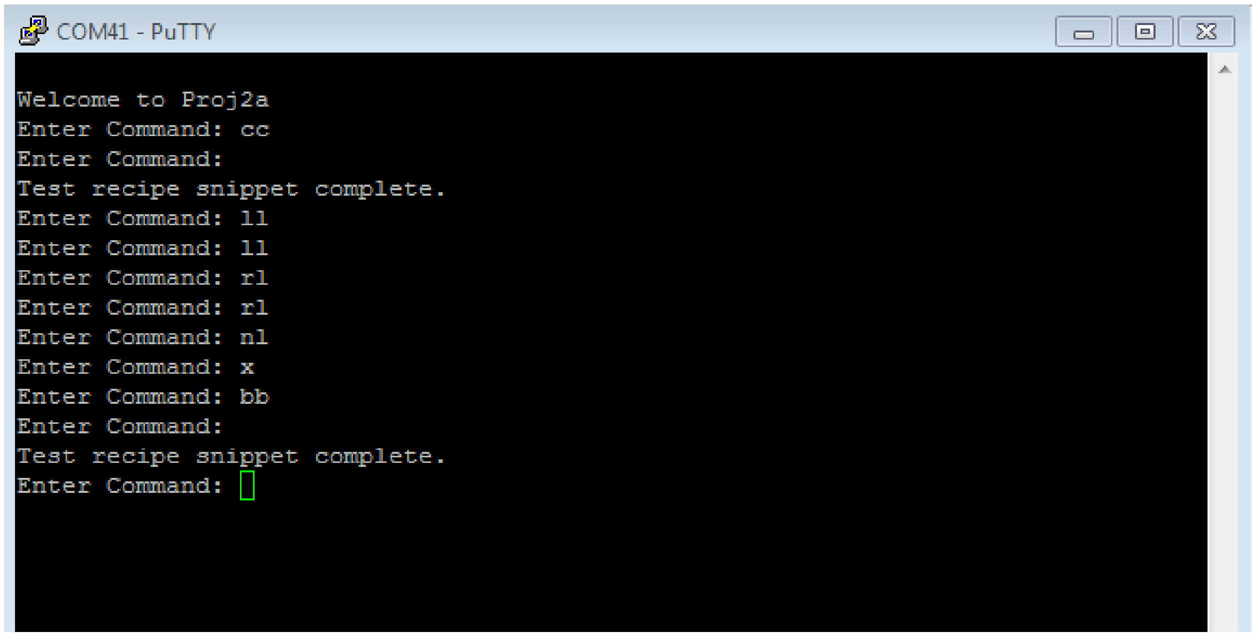




(Servo integration, CCRx=4, position 0)

(Servo independence CCR1=4, CCR2=21, position 0 and 5)

Project Results:



```
Welcome to Proj2a
Enter Command: cc
Enter Command:
Test recipe snippet complete.
Enter Command: ll
Enter Command: ll
Enter Command: rl
Enter Command: rl
Enter Command: nl
Enter Command: x
Enter Command: bb
Enter Command:
Test recipe snippet complete.
Enter Command:
```

User interface

Proper servo functionality achieved. Servos are independent of one another and process user input correctly as well as recipe snippets. Multiple recipe snippets used to test servo correctness.

Lessons Learned:

Pulse width modulation was the critical factor in this project as well as keeping track and up to date servo data while traversing the recipe array. Typical UART difficulties took place in filling the buffer. Recipe and recipe loop command handling was tricky in that index of the recipe had to be noted to determine which portion of the snippet was to be looped upon. Pausing the recipe sometimes falters as the UART hangs during motor movement. In hindsight, an interrupt implementation would be a better solution.