**Project 2b**

*Real-Time and Embedded Systems*

Chaithanya Gadiyam (cg8327@rit.edu)

Michael Moffitt (mjm6398@rit.edu)

**Area of Focus**

For this project, we were able to divide work on the main “kernel” used to run the servos, and some of the opcode processing and PWM operation. However, the division was not clean and both of us still worked on all elements of the project.

**Analysis / Design**

For this project the mode of operation stood out immediately as being much like an emulator for a simple system. Each servo has a struct representing its state, and a pointer to its program, or “recipe”. By creating a kernel which steps through the servo's program one cycle at a time, we can easily control many servos in parallel by time-sharing between different servo structs at a fixed interval. While this program only executes two servo recipes at once, it supports up to three (the number of i/o ports).

This program runs on four threads for parallel execution of user input processing, servo recipe execution and pulse generation for the two servo motors. Main thread polls on user input, processes those commands and controls the state machine. Servo thread executes the recipe of two servos and controls the state machine. Pwm threads generate pulse with duty cycle proportional to the servo position using nanospin and writes to i/o ports port A and port B.

For improved organization and code re-usability, the various functions and structures of the program are in their own files. For example, servo.c contains definitions, structures, and routines related to the servo kernel and pwm.c contains pulse generation based on the servo position.

Minimal input has been implemented to allow the program control flow from the user. Execution begins once the user inputs 'B' (for begin) for either or both servos, and the user may pause execution of servo using the input ‘P’ and resume the execution using the input ‘C’. The user can manipulate the servo position using inputs ‘L’ and ‘R’ to move left and right respectively, after pausing the recipe execution.

The unused opcode 011 has been used to design the operation MOVC, clockwise relative move. And the parameter is the number of positions the servo is to move relatively, it has a range of 0 to 5. An out of range parameter value produces an error. (graduate extension)

Error condition is displayed on console, when the state machine hits the error state.

**Block Diagram**

**Test Plan**

We have a number of test recipes from the assignment document, as well as two we wrote ourselves to demonstrate the graduate extension “MOVC” instruction (Clockwise relative move). We have tested the program with recipes which include nested loops and invalid opcodes, to test error conditions. We ensured that all commands work as expected, and that the programs executed as expected in the correct amount of time.

**Lessons Learned**

We have observed that the pulse width generated is not accurate with the usage of either usleep or nanospin, if the time period is configured to 20 ms. But better results are observed with a time period of 7.2 ms, with which proper operation of servo can be ensured. Also the parallel operation of two servos is affected if both the pulses are generated at the same time, whereas parallel operation of two servos can be achieved if a delay of 2-3 ms is allowed between the two spikes.

**Source Code**

The program source is included in the electronic archive in which this documentation sits.