SWEN 563/CMPE 663/EEEE 663
REAL-TIME & EMBEDDED SYSTEMS
ROCHESTER INSTITUTE OF TECHNOLOGY

PROJECT-2a: CONTROL OF TWO SERVO MOTORS BY INDEPENDENT RECIPES WITH THE STM32 DISCOVERY BOARD

Submitted to: Larry Kiser(llkiee@rit.edu)

Submitted by:

Nicolas Delanou (nxd7208@rit.edu)

D.O.S: 10/13/2016

Areas of Focus

Writing of the code:

- Sai Pradeep Reddy Bijjam: 0%
- Nicolas Delanou: 100%

This is my personal report. My teammate, Sai Pradeep Reddy Bijjam, told me he will do the entire report as I did 100% of the code during the project. On October 12th, he told me that he didn't even started so we decided to both submit our own report. He also agreed to take the entire responsibility for the delay. I allow him to use my code for his report but I would like to be graded independently. Sorry for the delay. This is a bit frustrating as we were one of the first group to do the demo. Thank you for your comprehension.

Note: *I only use the word "We" through my report for political correctness.*

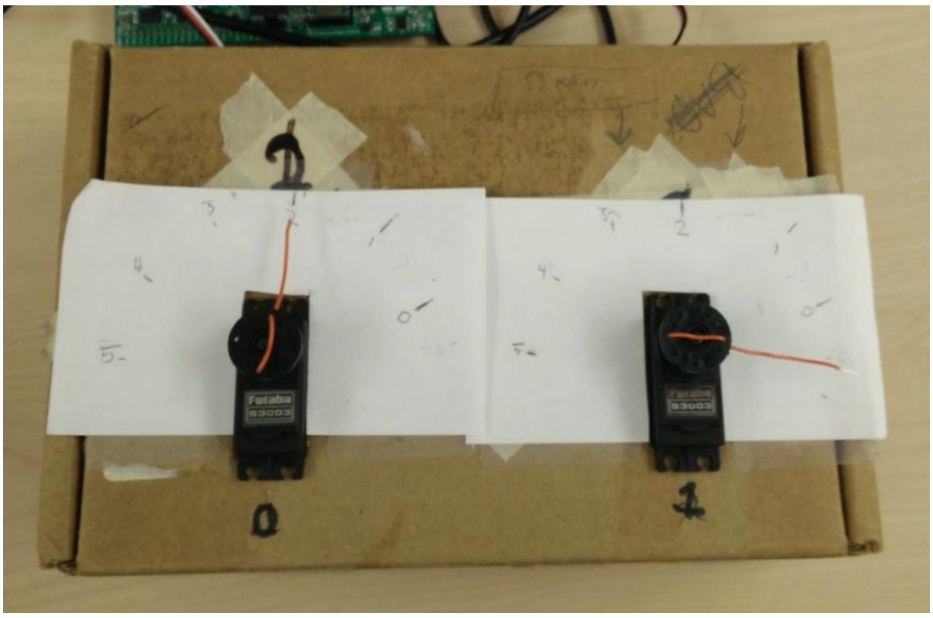
Analysis / Design

In this project, we used the concept of bare metal coding using C-language. The type of board used is the STM32L476VG Discovery Board using a Cortex-M4 32-bit RISC core with operating frequency of up to 80 MHz.

We are developing our code under the Keil uVision IDE.

The servo motors used for this project are two S3003. They are controlled by two independents PWM signals generated by the board on ports PA0 and PA1.

Here is a picture of the servo motors:



Test Plan

PWM generation

The first part of the project has been the generation of a correct PWM signal to control a single servo on PA0.

We decided to use Timer2 on channel 1 for the PWM generation. We first initialized the timer with a frequency of 80Mhz, the we changed the value of the TIM2->ARR register in order to have a period of 20ms. We then configured the register to work in in compare mode. Finally, we set the correct value of the TIM2->CCR1 register in order to have a correct ratio for the PWM.

Position	Angle (°)	CCRx	ARR	PWM high level (%)
Position 0	0	31040	1600000	1.94
Position 1	36	58976	1600000	3.69
Position 2	72	86912	1600000	5.43
Position 3	108	114848	1600000	7.18
Position 4	144	142784	1600000	8.92
Position 5	180	170720	1600000	10.67

We the do the same with the channel 2 in order to generate a PWM on PA1.

Command management

Servos are supposed to follow recipes given by the user. Those recipes are implemented as an array of consecutive instructions. Every instruction is a byte (8 bits). We are using the variable type “uint8_t” to store the value of an instruction.

The 3 most significant bits of an instruction represent the opcode of the instruction. The 5 other bits are used to as parameters. Here is a list of all the instruction implemented in this project:

Instruction name	Opcode	Purpose	Range of the argument
MOV	001	Move the servo to position X	0..5
WAIT	010	Wait for X*100ms	0..31
LOOP_START	100	Start of the loop. X repetition	0..31
LOOP_END	101	End of the loop	
RECIPE_END	000	End of the recipe	
Additional instructions (graduated student)			
JUMP	110	Jump to the instruction number X (wrote by Nicolas)	0..31
SYNC	101	Wait for both recipes to be in Synching state to start both recipe again. (Wrote by Nicolas)	

We then set an interruption function called every 100ms in order to update the recipe. We then configured the Timer5 in capture mode and used the function “TIM5_IRQHandler()”. At the beginning of the function, we have to set the interruption flag down in “TIM5->SR”. We then proceed to the engines management.

We created a new structure variable type called “Engine” and created two global variables of type Engine. The structure contains all the information about the engine state and his recipe.

```
typedef struct Engine Engine;
struct Engine
{
    uint8_t recipe[50];
    int wait_cpt;
    int index_recipe;
    int index_loop;
    int start_loop_index;
    int current_postion;
    int recipe_ended;
    enum Channel chan;
    int in_Pause;
    int in_Sync;
};
```

User Terminal

We used a serial terminal to communicate with the user. From another machine, the user can enter a two characters command that allow him to control the engines independently.

Here is all the available command:

Character	Command
P or p	Pause the running recipe
C or c	Continue the recipe
R or r	If recipe stopped, move the engine one position to the left
L or l	If recipe stopped, move the engine one position to the left
N or n	Nothing changes for this engine
B or b	Restart the recipe

Here are some command examples:

- *pc* : pause engine’s 1 recipe and continue engine’s 2 recipe
- *Rb* : Move engine 1 one position to the right if his recipe is stopped and restart engine 2 recipe
- *nB* : Nothing happened for engine 1 and restart engine’s 2 recipe

LEDs

We used the green and red LEDs on the board to give information about the program state.

- Green=ON & Red=OFF: At least one of the two engine’s recipes is running.
- Green=OFF & Red=ON: Both engine’s recipes are stopped.
- Green=OFF & Red=OFF: Both engine’s recipes are ended.

Project Results

After verifying with the professor that the PWMs generated were correct, we plugged the PA0 and PA1 to the servos.

We did tests with the recipe from the subject's PDF. We added instructions after the RECIPE_END instruction in the recipe to check if it stopped and changed the number of loops from 0 to 5 to see if the loop feature was working.

After checking this was working successfully, we loaded two different recipes into the two engines and we figured out that this was working.

We then added the additional instructions to the recipes. The JUMP instruction was working perfectly, allowing us to go wherever we want in the 32 firsts instructions of the recipes. The SYNC instruction worked great to. We can, for example, give the same recipe (with a SYNC instruction on it) to the two engines, launch the program, then stop one of the engine. The engine running will continue and wait the stopped engine at the SYNC instruction while the stopped engine isn't at the SYNC instruction.

We also tried all different kind of instruction on the USART Terminal and everything worked as planned. The LEDs responded perfectly with the asked commands.

In conclusion, 100% of the features asked in the subject are working correctly. 2 extra instructions have been implanted in the instruction set and have been verified during the demo.

Lessons learned

During this project, we spend less time looking for information in the documentation because we were already familiar with the board. However, we had to look in the Reference Manual to know how to configure the Timer in Compare Mod in order to generate the PWM.

Now, we are able to use Timers in Compare and Capture mods and we know how to generate and handle interrupts.

The creation of different features like, the engine management, the USART communication or the LEDs display forced us to organize our code in order handle all of them flawlessly.