Embedded Operating Systems

FA 2024_EEL3791C.02 ENGR

Experiment Date: 10/30/2025-12/1/2025

Submission Date: 12/1/2025

Motor Final Project

David Ledyaev: Lab Writer, Assist Coder

Aidan Tallaksen: Physical Component Manager

Aidan Sieger: Main Coder, Assembler during break.

Table of Contents

Introduction

FreeRTOS is a software that allows for the simulation of multiple processes happening at the same time, all with a single core. It does this by prioritizing tasks via different methods in order to make sure that the tasks can be done when they need to. And we shall use it to construct a useful application in this experiment.

Objectives

The objective for this lab is to use FreeRTOS in order to construct a working operation. In this case, a motor which could have adjustable speed and display that display onto a console and a 7-segment display.

Materials and Methods

Material List

1 Breadboard

1 Nucleo-F446RE Board

2 LEDS (1 red, 1 green)

1 7-segment display

3 resistors

1 L298N Motor Driver

1 DC Motor

Several Male-to-Male Jumper Wires

3 Male-to-Female Jumper Wires

Block/Wiring Diagrams.

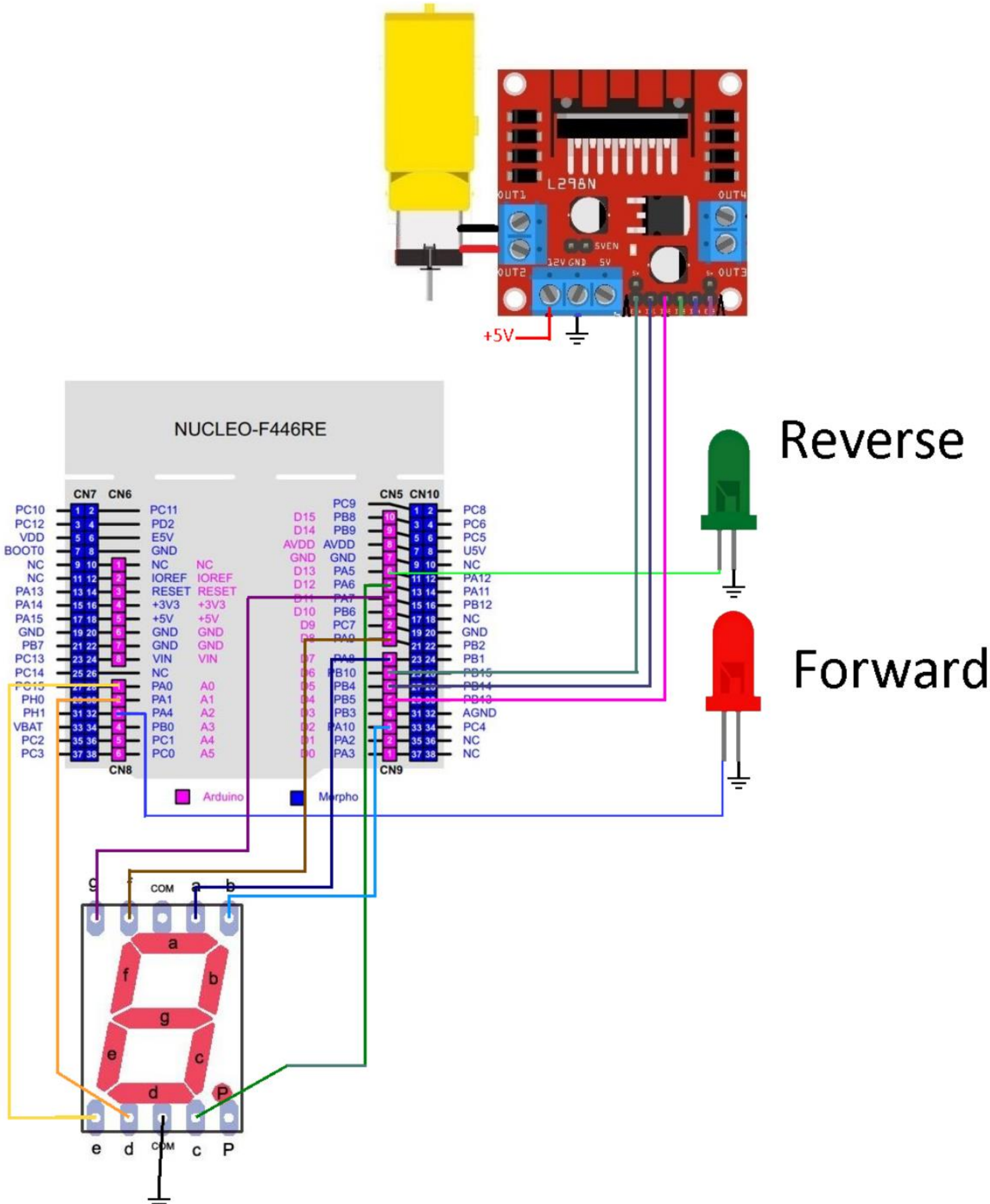

Figure 1: Block Diagram for Motor and Displays

Code Explained in Simple Terms.

For the initial setup, PB5 and PB4 are set as GPIO_Outputs. PA0, PA1, PA4, PA6, PA7, PA8, PA9, PA10 are also set up as GPIO_Outputs. Obviously, the System timer is turned on, with the timer set to timer 1. Timer 2 is set to an internal clock source, and channel 3 is set to PWM Generation CH3. The global interrupts for timer 2 is also enabled. The parameters for timer 2 are as follows: The pre-scaler is set to 89, the counter mode is set to Up, the counter period is set to 1999, and the rest of the settings are set as default. The clock configuration was edited though, setting the HCLK to 180 MHz. We then have the UART settings, we have the default settings for UART2: Baud rate at 115200 Bits/second. The global interrupts for UART2 is also enabled. Finally, we have the FreeRTOS settings, as we set the interface to CMSIS_V1. We then create 3 tasks, Direction Task (osPriorityAboveNormal, 128 stack size), Speed Task (osPriorityNormal, 128 stack size), and Segment Task (osPriorityBelowNormal, 128 stack size). We did create a Display Task as well, but it was not used, so feel free to ignore it. It should also be mentioned that you will have to add the -u_printf_float to the property settings (settings, properties, C++ build, settings, MCU/MPU GCC Linker, miscellaneous)

Onto the code itself, we start by adding the following headers: string.h, stdio.h, FreeRTOS.h, task.h, timers.h, event_groups.h, and stdlib.h. After that, we go ahead and define the variables for the outputs and the tasks. Two volatile variables should be created: Movement and PowerLevels (we also defined Velocity, but it was not used). You should also define the char arrays buff[20], way[56], and the uint8_t rx_data so that data can be taken into the console. Then you also type out a UART_Send function.

Then create a HAL_Init() function, HAL_TIM_PWM_START(&htim2, TIM_CHANNEL_3) function, and a HAL_UART_Receive(&huart2, $rx_data, 1). (We also had a custom UART_SEND to test if it was sending or not. It is optional if you know what you are doing)

We scroll down until the first TaskStart function, the StartDirectionTask specifically, where first have a switch case with Movement as our indicator. If Movement is 0, write the GPIOB and GPIOA pins, to the motor and LED respectively, have it set ReverseLED to 1 and ForwardLED to 0. If it is 1, do the opposite. For the default case, do nothing.

Then we have the StartSpeedTask, where we have another switch case, this time with PowerLevel as the input. If the case is zero, we run the function __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0). If its 1, set it to __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 400). The same goes for 2, 3, 4, and 5, with each one increasing the final variable, the speed, by 400 each time. Thus we get values of 800, 1200, 1600, and 2000.

Then we have the StartSegmentTask. This also takes in the PowerLevel for a switch case, but unlike the other task, instead of adjusting the motor speed, it changes the pins activated based on the 7-segment display. A case of zero would configure the 7-segment to display 0, 1 would display 1, ect. (We also have a StartDisplayTask, but it is useless, so feel free to ignore it)

Finally, we have the HAL_UART_RxCplitCallback(UART_HandleTypeDef *huart) function, the function that makes the console read a character. We start off with initialing a char array of Message[100], before activating the HAL_UART_Receieve_IT(huart, &rx_data, 1) command. Then we have a series of if and if else statements, based on the reading of rx_data. If its zero, set the PowerLevel to zero, and the same goes for 1, 2, 3, 4, and 5 for their respective numbers. If the variable is the '-' symbol, then Movement is set to 0, and if it is set to '+', Movement is set to 1. This makes the motor spin backwards and forwards respectively. Then we activate the StartDirectionTask, the StartSpeedTask, the StartSegmentTask, before using UART to print the adjusted settings to the console to read.

And that is our code in a nutshell. Not the most complicated code to follow, but not simple either.

Results & Discussion

After coding and building the circuit, we tested various inputs to make sure that it worked. Below are images of some of our tests.

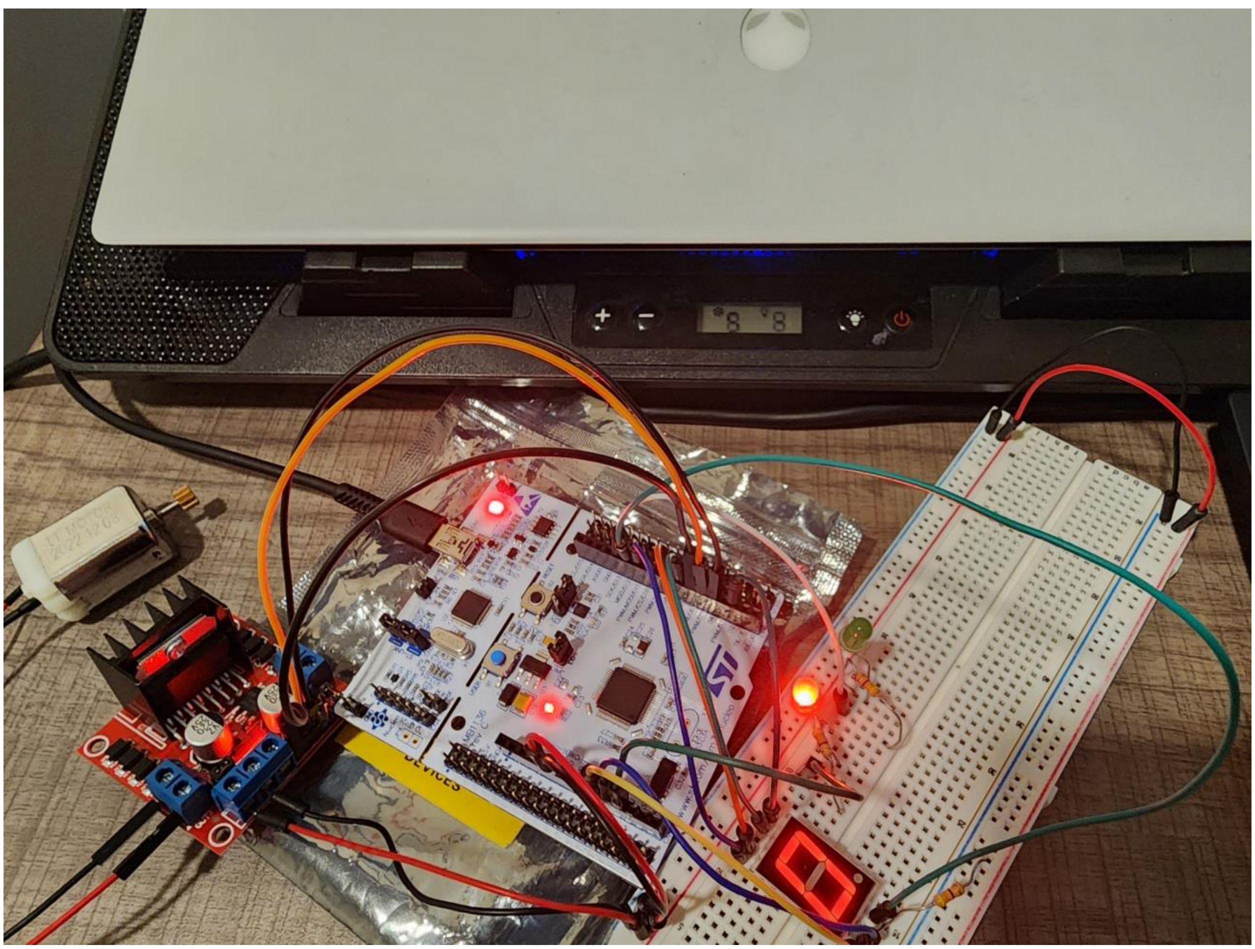

Figure 2: Motor at Power Level 0 in forward direction

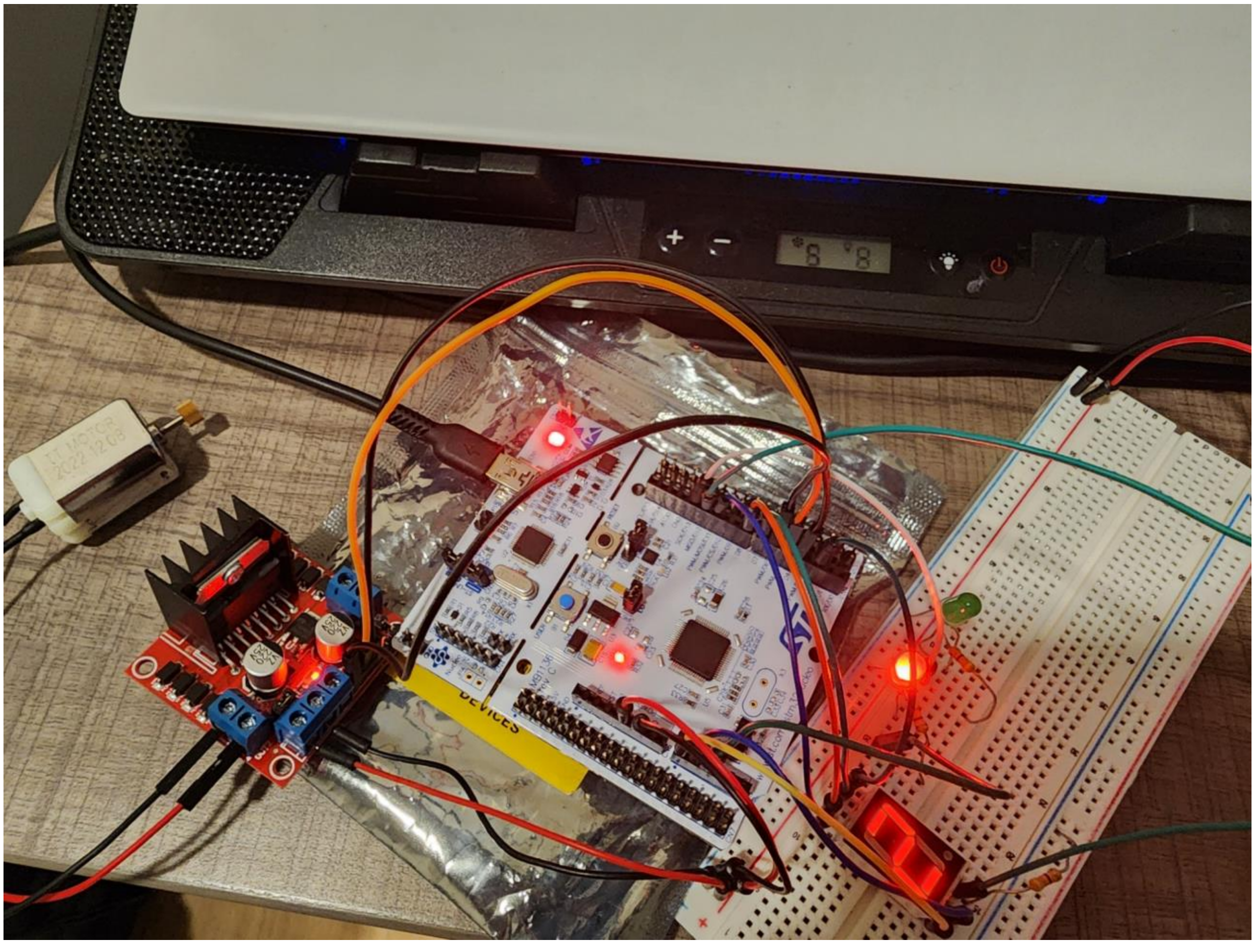

Figure 3: Motor at Power Level 2 in forward direction

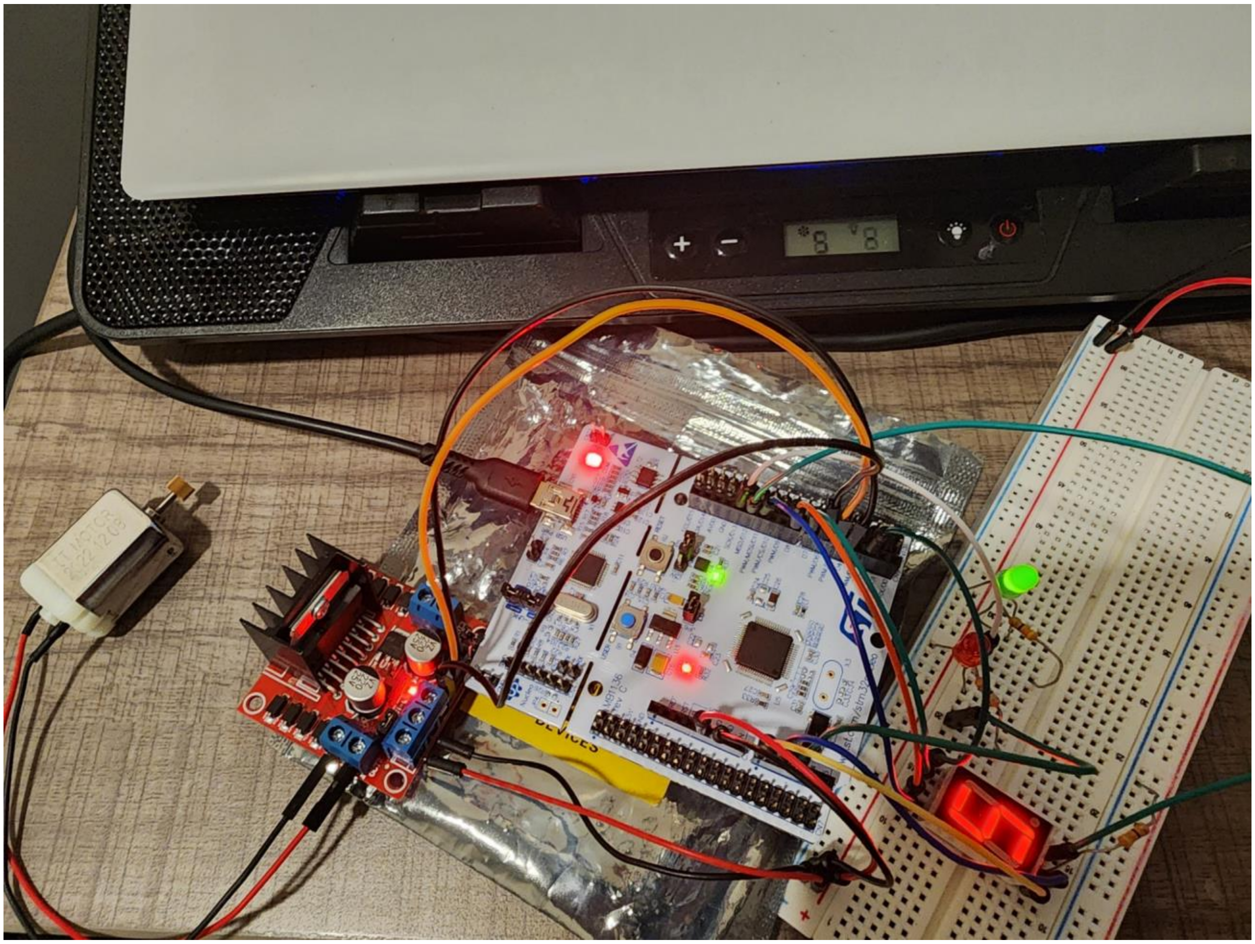

Figure 4: Motor at Power Level 5 in backward direction

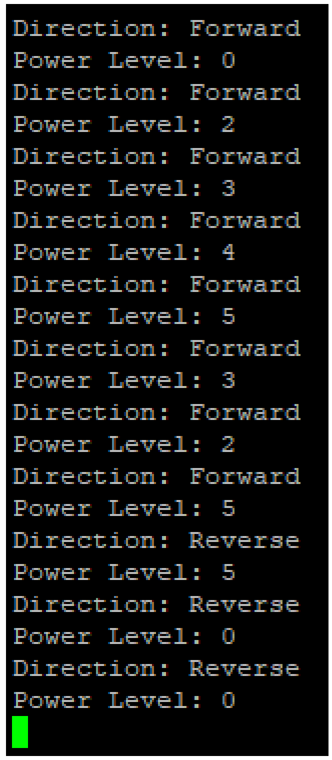

Figure 5: Data displayed on PuTTY terminal

After reviewing the results, we confirmed that the circuit works appropriately

Conclusion

In conclusion, we managed to create a fully functioning controllable motor. It is able to spin forwards and backwards. It is adjustable, being able to spin from speeds settings 1-5, as well as being able to be turned off with a speed of zero. It uses tasks and queues in order to handle the changes, and uses UART in order to print these changes to console, as well as to take in the inputs to change the motor in the first place.

Acknowledgements

This assignment was a group effort that could not have been done without the help of every contributor. The motor and motor driver module was supplied to us by Doctor Hoan T. Ngo.

References

To understand controlling a DC motor with a DC motor driver and Nucleo-F446RE, we used the following video: https://www.youtube.com/watch?v=26-3AUVJldA.

Link to GitHub Repo: https://github.com/RoadCode2/CDA3631-Final-Project