Multi-Motor Control

By Alberto Fabian Sosa Miranda

Contents

[Project Purpose 3](#_Toc176877944)

[STM 32 Chip Configuration 3](#_Toc176877945)

[4](#_Toc176877946)

[Hardware 4](#_Toc176877947)

[Communication protocol 5](#_Toc176877948)

[Program Structure 6](#_Toc176877949)

[Control Algorithm 7](#_Toc176877950)

[Frequency 7](#_Toc176877951)

[Step Count 8](#_Toc176877952)

[Acceleration / Deceleration 9](#_Toc176877953)

[Feed to Position 10](#_Toc176877954)

[Feed to Length 11](#_Toc176877955)

[Seek Home 11](#_Toc176877956)

[Zero 11](#_Toc176877957)

[Stop 11](#_Toc176877958)

[Run a speed 12](#_Toc176877959)

# Project Purpose

The purpose of the project is to change the independent motor control to a motor control using STM32G474 and driver (unknown depends on Andy), with a company made control system. The program is developed and tested on NUCLEO – H723ZG and DRV 8825.

# STM 32 Chip Configuration

1. Log into your stm32 account
2. Create a new project in STM32Cube IDE for your MCU
3. Enable FreeRTOS CMSIS\_V2 in the middleware section of Cube-MX
4. Configure a USART3 and activate the global interrupt with a pre-emption priority of 5.
5. Import MODBUS-LIB
6. Update the include paths in the project’s properties to include the Inc folder of the MODBUS\_LIB folder. Properties -> C/C++ build -> setting -> MCU GCC Compiler -> Include paths, in the section include paths type ..\Inc and ..\Config to include the files in step six and seven.
7. Create a ModbusConfig.h using the ModbusConfigTemplate.h and add it to your project in your include path.
8. Go to Properties -> C/C++ General -> Paths and Symbols-> Source Location-> Add file -> add Src file. I don’t know what this does but it works.
9. The timer 1 has channel 1 PWM output, with a PSC of 99 and its connected to PE9 on the chip which translates to pin D6 on the board. And all the interrupts are enabled.
10. The timer 8 has channel 1 PWM output, with a PSC of 1499 and its connected to PC6 on the chip which translates to pin D6 on the board. And all the interrupts are enabled.
11. The direction pin on the board (D7) is connected to the PG12
12. The enable pin on the board D5 and its PE11.
13. Under pinout & configuration -> RCC -> HSE disabled and LSE disabled
14. Under pinout & configuration -> SYS -> Timbase Source = Tim 6.

# A computer screen shot of a diagram Description automatically generated

Figure 1 Clock configuration

# Hardware

The intent of the hardware designer is to use STM32G474. The designer wants to use the same timer for all the motors. The question is: can the signal timer produce 10 pwm with different frequencies that can be changed in real-time and have high resolution to match the microseconds required for the application? The HT Timer can provide such a solution, as seen in the description:

“The high-resolution timer (HRTIM) allows generating digital signals with high-accuracy timings, such as PWM or phase-shifted pulses. It consists of 7 timers, 1 master and 6 slaves, totaling 12 high-resolution outputs, which can be coupled by pairs for deadtime insertion. The high-resolution is available on the 12 outputs in all operating modes: variable duty cycle, variable frequency, and constant ON time”

The parameters for a common application are:

That means the frequency is for a PSC of 32.

Hence frequency is 1/9665.68 = **0.0001 or 0.1microsecond**

# Communication protocol

For the application layer, the master is operated by QModBus and the slave is the MCU. The MCU follows the protocol from <https://github.com/alejoseb/Modbus-STM32-HAL-FreeRTOS>. In the installation progress I skipped DMA Mode.

The goal is to extend the communication protocol to multiple motors, and multiple functions. The first command will be a single register writing to indicate the motor that will operate.

|  |  |  |  |
| --- | --- | --- | --- |
| Slave ID | Write to coil code | Starting address | Data (Motor) |
| 11 | 05 | 00 00 | 1-5 |

Table 1 Motor modbus request

The second command will be to write the function code and the parameters that go along the function.

|  |  |  |  |
| --- | --- | --- | --- |
| Slave ID | Write to multiple registers. | Starting address | Number of Coils |
| 11 | 10 | 00 7c | 00 05 |

Table 2 OpCode modbus request

The remaining bits in the Table 2 OpCode modbus request include the parameters of the operation that will be given to the motor.

# Program Structure

Table 3 Motor struct

|  |  |
| --- | --- |
| Parameter | Value |
| Dir | GPIO PIN |
| Step | GPIO PIN |
| Enable | GPIO PIN |
| TIM\_TypeDef Timer 1 | Reference the timer class |
| TIM\_HandleTypeDef Timer 1 |  |
| TIM\_TypeDef Timer 2 | Reference the timer class |
| TIM\_HandleTypeDef Timer 2 |  |
| Function | Running function. |
| Current position | Current position of the motor, updated on every step, and when set to zero or seek home |
| Zero | The zero position of the motor |

Table Operation Struct

|  |  |
| --- | --- |
| Parameter | Value |
| opcode | 0x67 |
| angle of rotation | Input |
| max velocity | Input |
| Initial velocity | Set by the programmer based on manufacturer of the motor |
| acceleration | input |
| direction | Calculated based on the shorted path |
| Counter | Set by the |
| State |  |
| Totals steps |  |
| Acceleration state |  |
| Step rate |  |
| Absolute position | Absolute position to move to during the function Feed to position. |

The purpose of developing structs for each operation and motor is to automate the input and to develop an object-oriented program. The first input into the program is the motor to be selected, then the next check is if the corresponding motor has an undergoing operation. The motor selected is indicated by an index in an array of the Table 3 Motor struct. The motor field function has a value ranging from 0 to 4 then it is doing an operation, if it is negative, however, its not operating and will proceed to execute an operation.

There are six operations: Feed to position, Feed to length, Seek home, Zero, Stop, Run a speed and the steps in executing these operations

1. First read the register 124 containing the OpCode which selects the operation.
2. Then the field function of the motor selects a value available from 0 to 5, to communicate the motor which opcode it will be operation. The field value is selected on the first free available value, meaning no other motor has the specific field value.
3. Next a switch is used to select what function to fill out and execute. The Table 5 Operation Struct contains all possible parameters of all the functions, however when they are filled out only the corresponding parameters are filled. An array of structs contains all the functions and is its size matches the number of motors, because a motor can only do one operation at the same time. This exception must be broken for the stop function where it must stop at any time.

In the interruption of a step, first iterate over the motors to check which motor has the interruption. Once the motor is found check which timer has the step interrupted. The first timer is for the pwm interrupt, and the second timer is for the acceleration pwm. If the function does not have a second timer, it will not run this section of the code. In the first timer the counter is updated and the current position as well. If the counter reaches the total steps, all parameters of the instance Table 5 Operation Struct are set to zero for the next operation. In addition the step to stop function is called in the interrupt.

# Control Algorithm

## Frequency

A timer is going to be used to produce a PWM, and the PWM is going to control the motor. The important aspects to control of the PWM is frequency and number of pulses. I have set two functions that calculate the values for the frequency (hertz), and steps. The registers for changing the frequency are: period = ACC, pulse = TIMx\_CCRx = per /2 (always 50%).

For frequency the equation used is

|  |
| --- |
|  |

Equation 1 Frequency

And the equation for period is

|  |
| --- |
|  |

Equation 2 Period

## Step Count

To acquire a precise movement of the motor, the motor needs to stop at a microstep. The hardware is connected in a manner that the microstep is always 32, which means 32 steps per step. In the motor one step is 1.8 degrees, and the degree for a micro step is 1.8/32 or 0.05625. The number of microsteps required to change an angle input is then #microsteps = Angle\_input / 0.05625. The step count is used to stop and start the PWM timer. When steps are zero, the PWM starts and when the count of the PWM steps matches the microsteps then the PWM stops.

I use the function HAL\_TIM\_PWM\_PulseFinishedCallback to count the interruptions in every pulse.

## A diagram of steps Description automatically generatedAcceleration / Deceleration

Figure 2 Motion Curve for Acceleration and Deceleration percentage strategy.

The motor accelerates and decelerates to avoid slipping at the start and end of a motion. The initial velocity is 50 steps / second. The data collected as input is max velocity [Steps /s], point\_point\_dis [angle], acceleration [Steps / ] and direction of the motor [CW = 0, CCW = 1].

The algorithm uses two timers, the timer for the velocity (steps / s) and acceleration (steps / s^2). The timers are set when the pulse count is zero and until the input has been read. The timer for the velocity is set using the function period\_Update, and the same function is used for changing the period when acceleration and deceleration. However, the timer for acceleration (TIM8), remains the same through the profile and for that reason it’s set at the start and never changed. The equation for setting the frequency of the TIM1 are Equation 1 Frequency and Equation 2 Period. The TIM8 follows the same equation for Equation 2 Period, but the frequency differs. Frequency is the step / per second, and acceleration is steps to generate in a second that will be added to the velocity at the rising edge, hence acceleration input is frequency.

The configuration for the timers is found in STM 32 Chip Configuration. The period Update also must convert the input to RPM to use the frequency equation, this is done with the equation below.

Equation 3 Steps rate to RPM

The profile of the acceleration or deceleration is controlled by the count of the pulses. When the count is starting from zero, profile is accelerating and when the count equals steps to stop. The value of steps to stop is found using the function Steps\_To\_Stop. The goal of the function is to velocity from the current velocity to the initial velocity. The essence of the function is that time traverses the same for both timers. For every acceleration pulse the period of the velocity changes, however you can add the number of steps taken by the velocity pwm between every pulse of the acceleration pwm to acquire the steps to stop. The equation for this is seen below.

Equation 4 step to stop formula

The velocity denoted by the variable name step rate is updated in the interrupt HAL\_TIM\_PWM\_PulseFinishedCallback. The acceleration\_state variables is used to move between a switch case, with the cases NOACC, ACCEL, DECEL, STOP. When accelerating the step\_rate increases by 1 every pulse by TIM8 and decreases by one in every pulse but in the deceleration stage. The STOP is for making sure the motor stops at the initial velocity, and the NOACC state is for the motor to move at a current speed after reaching the desired speed in the ACCEL state.

## Feed to Position

The goal of the function is to move to an absolute position. Each motor has two fields in their corresponding struct, current position and zero. They hold the steps taken and the location of the zero. Moving the motor to a precise location in the shortest path requires the following. First indicate the direction to take, clockwise or counterclockwise. This can be done by checking if the absolute position to move is on the right half of the current position or the left half. If Equation 5 Feed to Position motor direction, holds true then the direction of the motor is CW, otherwise is CWW.

Equation 5 Feed to Position motor direction

The second step, now that the direction is chosen, is to figure out the angle of rotation. The angle of rotation differs from the absolute position value, because angle of rotation is a relative value compared to the current position of the motor. There are four cases for calculating the angle of rotation. In the first case, the direction is clockwise and the current position is greater then the absolute position. This case is for when there is an overflow from 360 to 0. In the second case, the direction is clockwise and there is no overflow. Similarly, the third case is for when there is an overflow in the opposite direction conter clock wise, 0 to 360. The fourth case is when it is counter clock wise and there is no overflow.

Equation 6 Feed to Position angle of rotation case 1

Equation 7 Feed to Position angle of rotation case 2

Equation 8 Feed to Position angle of rotation case 3

Equation 9 Feed to Position angle of rotation case 4

## Feed to Length

The function adds to the current position of the motor; however the program keeps count of the motor position.

## Seek Home

The motor moves at a constant speed until a GPIO pin is turned on. Not fully implemented because no sensor is available for this.

## Zero

Set the position to angle zero. Default function, after turning on.

## Stop

Stop the motor

## Run a speed

Run at a certain speed with the motor and do not stop.