Programming a Microcontroller to Drive a Fisher & Paykel SmartDrive™ Washing Machine Motor to Power a Go-Kart.

Andy Fendall
Waikato Institute of Technology
Andy.Fendall@wintec.ac.nz

ABSTRACT

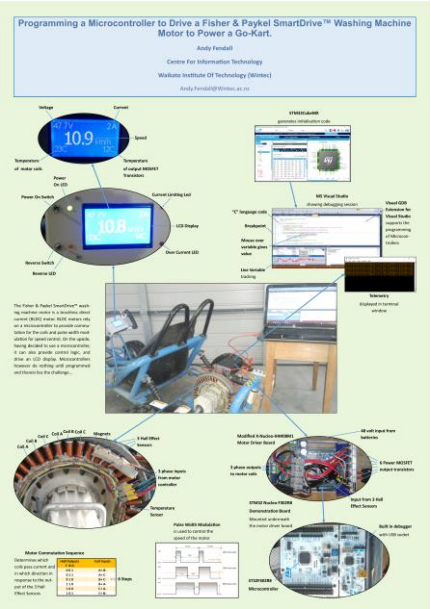
Microcontrollers are one of the basic building blocks of the "Internet of Things". In this paper, I describe the tools and processes needed to program a STM32F302R8 microcontroller to drive a Fisher & Paykel SmartDrive™ brushless direct current motor. The motor is used to power a go-kart. The STM32F302R8 is an ARM® Cortex®-M4 microcontroller, programmed with the "C" programming language. The F&P SmartDrive™ motor is a brushless direct current motor, which relies on a microcontroller unit to provide commutation for the coils.

Keywords: *Microcontroller, C Programming, Internet of Things, Programming Embedded Devices*

1. INTRODUCTION

The Fisher & Paykel SmartDrive™ washing machine motor (Fisher & Paykel Appliances Ltd, 2019) is a brushless direct current (BLDC) motor, driven by a microcontroller in a washing machine. It is a good choice of motor to replace a petrol engine in a go-kart because of its low cost, ready availability at recycling centres, and high torque. BLDC motors typically run at 90% efficiency and produce maximum torque at zero rpm. With a 48vdc power supply from four 12AH lead acid batteries in series, it can provide about one kilowatt of power while drawing about 20 amps. This setup gives over an hour of running between charges.

The drawback of BLDC motors is that they require a microcontroller to switch the current at the correct time and in the correct direction though two of the three sets of coils (three phases). It is also needed to control the speed of the motor by providing pulse width modulation to the output MOSFET



This quality assured paper appeared at the 10th annual conference of Computing and Information Technology Research and Education New Zealand (CITRENZ2019) and the 32nd Annual Conference of the National Advisory Committee on Computing Qualifications, Nelson, NZ, Oct. 9-11, during ITx 2019.

transistors. On the positive side, having decided to use a microcontroller, it can also provide control logic, and drive an LCD display. An STM32F302R8 microcontroller (STMicroelectronics International N.V., 2019) was chosen because it comes mounted on a demonstration board with a built-in debugger for a little over $20. Considerably cheaper than an Arduino. (RS Components Ltd, 2019) It can be mated with a Motor Controller demonstration board at a cost of about $100. (RS Components Ltd, 2019)

While demonstration software is provided by STMicroelectronics for the microcontroller demonstration board and motor controller board combination, the software did not work with the SmartDrive™ motor. The author was therefore forced to program the microcontroller from the beginning, without the benefit of a previously written working program.

2. WHAT THE PROGRAM NEEDS TO DO

The microcontroller needs to be programmed to: -

- Process several digital inputs.
- Measure the voltage on several analogue inputs.
- Output digital signals to turn display LEDs on and off.
- Produce 3 pulse width modulated signals to control 3 gate drivers which, in turn provide the necessary signals to control the output MOSFET transistors.
- Build a bitmap picture for the LCD display and output it in the correct format.
- Provide some control logic.
- Produce telemetry and output it in asynchronous serial format via a UART.
- Instantly switch the MOSFETS off, if the over current state is detected.

To start with, the primary function of the microcontroller is to drive a brushless DC motor. Three Hall Effect sensors, located on the stator, provide information about the position of the rotor which contains embedded ferrite magnets. This is in the form of a three-digit code. The microcontroller reads these signals in response to a signal change. A switch statement is used in the program to provide the right combination of digital signals to switch three gate drivers into one of six possible configurations. As the magnets pass a set of three coils, all six steps are processed to drive the output MOSFET transistors to switch the

current through the coils. The current sets up a magnetic field in each coil which either pushes or pulls each magnet.

The speed of the motor is controlled by a potentiometer connected to an analogue to digital converter peripheral, built into the microcontroller. The program averages the digital output to remove noise and the result is used to control pulse width modulation generated by a three-channel timer peripheral built into the microcontroller. The higher the voltage produced by the potentiometer, the wider the pulses (longer duty cycle) which are fed to the gate drivers, and the higher the voltage applied to the coils in the motor.

Other analogue inputs are used to measure the voltage available to the motor controller, the current drawn by the coils, and the temperature measured at the output transistors and on the motor itself.

The motor controller board provides hardware sensing of the current. If the current exceeds about 30 Amps a pin on the microcontroller is driven high. The program instantly detects this over current state and shuts the motor down for a period of two seconds, saving the MOSFETS from instant annihilation. A flashing LED indicates the overcurrent state.

Two switches are used to control the motor. One is an on/off switch and the other is used to switch between forward and reverse directions. Another Timer peripheral is used to trigger reading the state of the switches every tenth of a second. Two LEDs are used to display the "on" state of the motor and the "reverse" state. The program provides control logic so the motor will not enter the reverse state from the forward state until the speed of the motor has dropped to zero.

To avoid the motor controller going into the over current state when the go-kart is climbing a steep hill, another function of the program is to provide proportional control of the current. If the current exceeds a threshold, it is held close to 20 Amps by proportionally reducing the duty cycle of the pulse width modulation.

The current is also limited to stop the motor controller going into the over current state when the speed control is suddenly increased to full throttle. Another function in the program fixes this by only allowing the pulse width modulation duty cycle to increase in proportion to the speed of the motor.

The speed of the motor is measured in revolutions per minute (rpm). It is calculated over a period lasting a tenth of a second, by counting the number of six step cycles and applying a scale factor.

The LCD Display displays the speed of the go-kart in kilometres per hour, scaled from the motor rpm. It also displays the battery voltage and current drawn, as well as the motor temperature and the temperature of the output MOSFETS. The program uses font maps to construct a bitmap, stored in an array. The contents of the array are then clocked out to the LCD Display along with command sequences. This process happens twice a second to refresh the screen.

Finally, telemetry is assembled, consisting of motor on state, motor reverse state, voltage, current, motor rpm, output stage temperature and motor temperature. The data is concatenated into a JSON string and output from the UART peripheral. It can be viewed in a terminal emulator window on a laptop via a USB cable, but future work could enable the telemetry to be uploaded to a cloud server using a radio link.

3. THE STRUCTURE OF THE PROGRAM

The main program starts by initialising the clock tree of the microcontroller, followed by the NVIC (Nested Vector Interrupt Controller), DMA (Direct Memory Access), ADC (Analogue to Digital Converter), timers, GPIO pins and UART (Universal Asynchronous Receiver Transmitter). It then goes into an infinite loop. The only code in the loop is for constructing the bitmap for the LCD display and then transmitting the bits from two GPIO pins connected to the display. This is a long running task but is not time sensitive.

The most time sensitive task is to read the state of the three Hall Effect sensors and switch the PWM timer outputs to the gate drivers. This is triggered by one the Hall effect sensor pins transitioning from low to high, or high to low. This raises an interrupt which is handled by an Interrupt Service Routine (ISR). The code in the ISR does the work of selecting one of six combinations of outputs to the three gate drivers. This ISR gets the highest priority, controlled by the NVIC, which means that all other code execution is halted and transferred to the stack while the ISR is being processed. This results in very low latency in switching the current to the motor coils.

The next most time sensitive process is to transfer the output of the ADC. The ADC uses DMA to write the output of its conversions to an array. The Conversion Completed call-back handles the interrupt generated by the ADC and the code copies the contents of the array to other arrays for averaging.

The Timer Period Elapsed call-back handles the interrupt given the third highest priority, which is raised every tenth of a second. This code reads the state of the on/off and reverse switches and applies some control logic. It averages the various outputs of the ADC and calculates the length of the duty cycle of the pulse width modulation, which controls the speed of the motor. In doing this calculation it considers current limiting necessitated by high current draw or low RPM.

Another timer generates an interrupt ever half second which is also handled by the Timer Period Elapsed call-back at a lower priority. It sets a flag (global variable) which signals to the main program to start the display refresh process.

The lowest priority interrupt is used to trigger the UART to transmit the telemetry string.

4. TOOLS USED AND DESIGN PROCESS

The microcontroller is mounted on a STM32 Nucleo-F302R8 demonstration board with a built-in debugger. The debugger connects to a laptop for programming using a USB cable, which also supports asynchronous serial communication with the microcontroller. The author's preferred IDE is Microsoft Visual Studio. Fortunately, a third-party company supplies an extension to Visual Studio called VisualGDB (Sysprogs OÜ, 2019) which sets up a new project with the Hardware Abstraction Layer (HAL) source files for the specified microcontroller. It automatically loads the microcontroller flash memory with the executable program and supports step by step debugging and real time graphical monitoring of variables.

The initialisation code for the microcontroller peripherals is generated by another software application, from STMicroelectronics, called STM32CubeMX (STMicroelectronics International N.V., 2019). The application provides a well laid out graphical user interface, allowing the user to select pins on the microprocessor for each particular purpose and set up peripherals such as the GPIO pins, the ADC controller, NVIC, DMA, Timers and UART. The application generates "C" source code which can be imported into a new VisualGDB embedded project in Visual Studio. The configuration of the peripherals can be changed in the STM32CubeMX application and the code re-generated without losing any existing code.

The only drawback with the STM32CubeMX generated code is that it initialises each of the various peripherals but does not start them. For that another line of code for each peripheral is required. A second missing line of code is also needed for each interrupt handler to receive the call-back from the interrupt controller. The functions are included in the HAL library but in each case, for the novice there is no hint of what the function could be called. Resorting to the manual for the microcontroller only leads to further confusion, because of the huge amount of detail provided. Fortuitously, electronics engineer Carmine Noviello has recently written a book called Mastering STM32 (Noviello, 2019) which is pitched at beginners and contains plenty of example code. The missing functions and their parameters were retrieved from the book along with a good understanding of how the code works.

5. CONCLUSION

The author has shown that it is possible to program an STM32 microcontroller to provide commutation for and control a Fisher & Paykel SmartDrive™ washing machine motor. The washing machine motor can power the go-kart up to 12 km/h on the flat and drive a 108kg payload up a 1:5 slope.

In future work the author intends to add a GPS receiver and radio transmitter to the project to remotely monitor the location of the go kart and gather telemetry. This would add the project to the "Internet of Things".

6. REFERENCES

Fisher & Paykel Appliances Ltd. (2019, June 30). *Smartdrive Technology*. Retrieved from Fisher & Paykel Appliances Ltd: https://www.fisherpaykel.com/nz/company/innovation/smartdrive-technology.html

Noviello, C. (2019, June 30). *Mastering STM32 book*. Retrieved from Carmine Noviello: https://www.carminenoviello.com/mastering-stm32/

RS Components Ltd. (2019, June 30). *STMicroelectronics Nucleo Board, Power MOSFET Evaluation Board X-NUCLEO-IHM08M1*. Retrieved from RS Components Ltd Web site: https://nz.rs-online.com/web/p/processor-microcontroller-development-kits/1513007/?sra=pmpn

RS Components Ltd. (2019, June 30). *STMicroelectronics STM32 Nucleo-64 MCU Development Board NUCLEO-F302R8*. Retrieved from RS Components Ltd Web site: https://nz.rs-online.com/web/p/processor-microcontroller-development-kits/8112277/?sra=pmpn

STMicroelectronics International N.V. (2019, June 30). *Nucleo-F302R8*. Retrieved from STMicroelectronics Web site: https://www.st.com/en/evaluation-tools/nucleo-f302r8.html

STMicroelectronics International N.V. (2019, June 30). *STM32CubeMX*. Retrieved from STMicroelectronics International N.V. Web site: https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stm32cubemx.html

Sysprogs OÜ. (2019, June 30). *VisualGDB*. Retrieved from Sysprogs OÜ: https://visualgdb.com/?features=embedded