Microcontroller based Motor Control for Space Applications

Aaron Maurice
Advanced Programs Group
Moog Space and Defense
Gilbert, Arizona
amaurice@moog.com

Mark Broadbent
Advanced Programs Group
Moog Space and Defense
Gilbert, Arizona
mbroadbent2@moog.com

***Abstract*—This paper discusses the benefits and challenges of using a microcontroller for motor control in space applications instead of traditional FPGA or ASIC based designs.**

Keywords— Rad-Hard microcontroller, SAMRH Family, motor controller, embedded flight software

I. Introduction

Motor controllers are widely used in space applications for pointing, attitude control, deployment and other critical mechanisms required for the successful operation of a spacecraft. Most motor controllers for space applications use an Application-Specific Integrated Circuit (ASIC) or an Field Programmable Gate Array (FPGA) for the logic in the design. The motor controller accepts commands from the spacecraft, controls the driver circuits to deliver power to the motor, and may incorporate feedback to better control the motor torque, jitter, velocity, and/or position of the mechanism. Recent advances in microcontrollers such as the Microchip SAMRH Family have enabled a new solution for space applications in motor controller design that can be more flexible than an ASIC and lower cost than a FPGA with similar performance.

II. Motor Controller Functional Comparision

A simplified motor controller block diagram is shown in Fig. 1. The motor controller's spacecraft interfaces include power and command/data. Commands are first transmitted to the motor controller's logic. The motor controller's logic then turns on and off the driver circuits to deliver power to the motor per the required commutation scheme. Optionally, feedback can be added to precisely control the delivered current or sense and control the position/velocity of the mechanism.

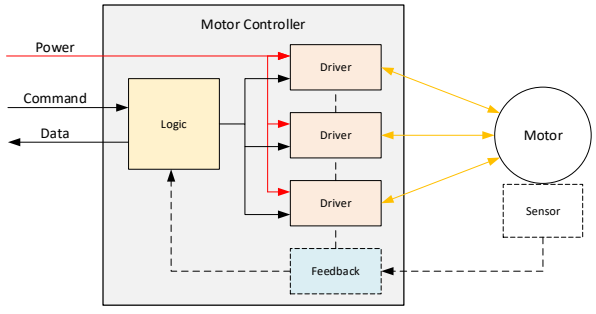

Fig. 1. Simplified Motor Controller Block Diagram

The lowest cost motor controllers are typically an ASIC based solution like the Moog Electronics Control Unit (ECU) [1]. These designs work well for applications that drive a motor with a fixed command interface and simple or no feedback.

FPGA based solutions are extensively used for more advanced motor controllers such as the Moog Gimbal Control Electronics (GCE) [2] or Moog Rikishi Electronics Unit (REU) [3]. The configurability of the FPGA allows the designer to create a motor controller that can support almost any command interface, multiple motors and almost any feedback. Some advanced FPGAs also support softcore or hardcore processors that can provide floating point math for more complex control loop implementations. These solutions usually require support circuitry including multiple power rails, memory, analog and digital interfaces in addition to the driver and feedback circuits.

The less expensive FPGA based solutions typically use antifuse parts that can only be programmed once. These are commonly used for motor controllers because they can start running almost immediately after power up and don't require time to load the FPGA image. This allows spacecraft to keep these motor controllers powered down or idle and then quickly resume operations when needed. Antifuse FPGA solutions cannot update the logic after it is first loaded, and these FPGAs have less configurable logic which can limit the complexity of the implemented control scheme.

The more expensive FPGA based solutions are SRAM or Flash based. Many of these designs can support complex interfaces, multiple motors and run multiple control loops. While these types of designs offer high performance and customization, the cost of the FPGA solution drives higher motor controller costs and higher power consumption. An important strength of these parts is that they can be reprogrammed for different applications or updated during development to improve or tune the control response.

An alternative approach is to use a microcontroller such as the Microchip SAMRH707© instead of the ASIC or FPGA for the logic in the motor controller such as in the Moog Motor Control Electronics (MCE). Modern microcontrollers have evolved to a complete system on a chip (SOC) design with integrated bootloaders, RAM, non-volatile memory, oscillators and multiple hardware peripherals providing communication links, external memory interfaces, pulse width modulation (PWM) blocks, analog to digital converters (ADC) and digital

to analog converters (DAC). The block diagram of the SAMRH707 is shown in Fig. 2.

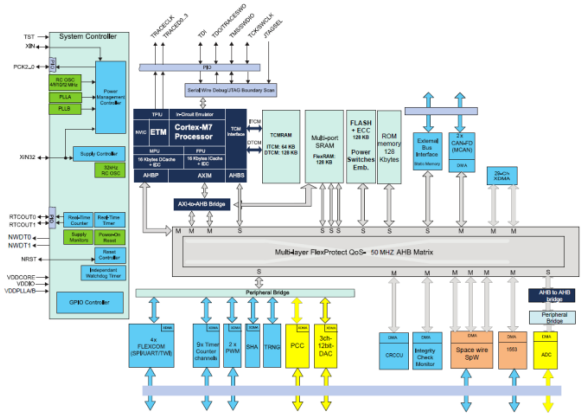

Fig. 2. SAMRH707 Block Diagram ©Microchip

A. Motor Controller Command and Telemetry Interfaces

At a minimum, the command interface needs to communicate the simple commands to move a motor such as when to move and what direction to go. More advanced interfaces can change operational modes, adjust open or closed loop control options, and query status from the controller.

The command interface to an ASIC solution for a stepper motor is typically a discrete interface of step, direction and enable. For an ASIC controlling a BLDC motor, it is common to see enable and direction discretes with an adjustable voltage input to control speed/torque. More complex command interfaces are possible, but that adds significant complexity to the ASIC design which may limit the applications the ASIC is well suited for.

An FPGA solution can implement almost any command interface such as RS-422, LVDS, SpaceWire, 1553, discretes and Ethernet. The interface is implemented in logic in the FPGA and can be customized for the application.

The command interface for the microcontroller solution can support almost any spacecraft interface like the FPGA including a UART (RS-422/LVDS), SpaceWire, 1553, discretes, CAN-FD and discretes. The bootloader can be directly accessed via the UART (RS-422/LVDS) or SpaceWire which provides a way to load software without running Application Software (ASW) first. Microcontroller-based designs can provide a way to load new software without needing board level access as this is not always available later once testing has started. This can also be helpful to initially load software or adjust motor control and performance on orbit. The integration of these control interfaces also decreases the time to implement a standard interface compared to an FPGA. Configuring hard interface blocks are made easier with tools like Microchip's freely available Harmony software development framework [4] for interface configuration vs writing custom VHDL code.

B. Motor Torque, Velocity and Position Feedback

The simplest motor controllers have no feedback sensing controlling the motor in an open loop mode. This is most common in simple stepper controllers because the relative position of the stepper motor can be reliably inferred based on how many steps the motor has been commanded to move.

Many motor controllers include current sensing feedback which is necessary to reliably control the output current over the wide range of operating conditions. Output current is important as it is proportional to the torque output of the motor. Most current sense circuits amplify the differential voltage over a shunt resistor which provides a voltage proportional to the current that can then be read by an Analog to Digital Converter (ADC) and converted back to current for the logic device. Less common current sense approaches include hall-effect current sensing.

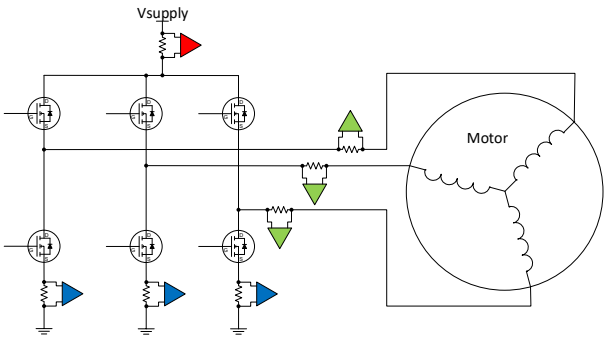

Fig. 3. Typical current sensing options on a 3-phase motor

There are three main options for motor current sensing. The simplest approach is a low-side current sense using an inexpensive amplifier and sense resistor as shown in blue in Fig. 3. The disadvantage of this approach is that it disrupts the ground path and cannot detect a short in the motor. Another option is to use a high-side current sense as shown in red in Fig. 3 to reduces the dynamic voltage impacts on the system, but this requires a more expensive amplifier design for the sense resistor that can handle the higher common mode voltage. The high-side current sense can detect a motor short but doesn't provide insight into which phases are carrying the current, thus preventing ideal motor control. The preferred in-line current sense shown in green in Fig. 3 reads the positive or negative current over a shunt on each phase providing the true current the motor is consuming. In most applications, only two out of three phases need to be sensed to capture the total current through the motor. This requires the most complicated and expensive amplifier design, but provides the best current feedback for the motor torque and jitter control.

Velocity and/or position feedback is required for a motor controller to perform close loop control for servo or gimbal applications. Common position and velocity feedback options include optical encoders, potentiometers, resolvers, linear variable differential transformers, hall effect sensors, quadrature decoders, 2-bit Gray up/down counter and back EMF. The data from these options can be a telemetry interface, analog output that is read by an ADC or a discrete as shown in TABLE I.

TABLE I. COMMON FEEDBACK OPTIONS AND INTERFACE REQUIREMENTS

Feedback	Per Motor
Bus Voltage	1 ADC Channel
Motor Current	1-4 ADC Channels
Optical Encoder	1 Telemetry interface
Potentiometers	1+ ADC Channels
Resolvers	Telemetry interface or 3+ ADC channels
Hall Effect Sensors	3+ Discrete inputs
Quadrature Decoder	2 Discrete inputs with timer
2-bit Gray up/down count	2 Discrete inputs with timer
Back EMF	3+ ADC channels

Many ASIC solutions lack feedback because feedback options vary significantly based on the application. Some ASIC solutions can support current sensing with fixed or variable current setpoints and velocity control based on an analog voltage input. Other ASIC designs feature passthrough circuits that allow feedback sensing to be passed through the motor controller back to the spacecraft. This allows the ASIC to be used in more applications but pushes the sensing and control burden to the spacecraft to handle the additional sensing and control requirements.

The FPGA can interface to any feedback described in TABLE I. , but many require additional circuitry to be added to the design and have limitations. Feedback that can be sensed by an ADC will use an ADC integrated circuit (IC). A common 12-bit ADC used in applications like this feature eight channels and communicates back to the FPGA with a SPI connection. The main limitation of this type of a design is that the sample time is not closely tied to the PWM waveform as the FPGA must first communicate to the ADC to sample, and then communicate again to retrieve the data. The resulting timing latency is often multiple microseconds or greater.

If the solution requires more than eight ADC channels (such as current control plus resolver sensing), multiple ADCs may be required. More specialized ADC solutions also exist such as the ability to read multiple channels in parallel at higher rates. In most cases, more specialized parts are much more expensive.

The FPGA can also support feedback from telemetry interfaces or discretes. Almost any source of feedback can be integrated as the logic can be customized for the application.

The microcontroller has an integrated 12-bit 16 channel ADC that can run up to a 1 MSPS conversion rate. This ADC can be triggered from the PWM block to sample at consistent or ideal locations in the PWM waveform. This allows for more accurate current sampling with less latency that can be fed back into a current control loop necessary for many types of motor control.

The 16 available ADC channels can support current sensing on every motor phase and any other position/velocity feedback shown in TABLE I. without running out of available ADC channels. The combination of the integrated ADC with precise timing control relative to the PWM waveform and quantity of ADC channels is a significant cost and complexity advantage over an FPGA based solution.

The microcontroller also has four FLEXCOM interfaces that can be configured to support UART, SPI or I2C interfaces. One or two of these interfaces may be allocated to support the spacecraft UART or external memory, but the remaining interfaces can be used to interface to sensors with telemetry feedback such as an encoder.

The analog and digital interfaces are also configured using the Harmony software development framework to set up each interface as required for the application.

C. Motor Driver

A motor controller needs to be able to source or sink current to each phase of the motor. The configuration of the sinking or sourcing of each phase varies over time to form the commutation for the motor. The most common way to do this is with a half bridge circuit as shown in Fig. 3. This example is a 3-phase motor, but the addition of a fourth phase can allow the motor controller to handle 2-phase, 3-phase, or 4-phase motors. Typically, current will be driven from the high side of one or two phases through the motor to a ground connection on one or two phases. Some motor controllers are configured for unipolar applications where the motor phase is always powered, and the motor controller only switches on the low side.

If current control is required for the application, the output switches are typically Pulse Width Modulated (PWM). A PWM scheme will select a switching frequency and then turn on the switch for a portion of that period for a duty cycle. The longer the period, the more current flows through the motor.

Multiple PWM schemes can be implemented from high-side 2-quadrant, low-side 2-quadrant, 4-quadrant, trapezoidal, sinusoidal and Field Oriented Control (FOC). Within each scheme, multiple variations exist that provide small improvements in control or to mitigate feedback for certain applications.

Low power motor driver circuits can be directly integrated into an ASIC design without an external driver circuit reducing the amount of support circuitry required for the motor controller and in turn reducing the cost of the solution. Higher power motor drives typically use the same external motor driver circuits as used in FPGA or microcontroller applications.

The strength of the FPGA is in the motor driver interface. Any external motor driver circuit can be used here and interfaced to available IO and logic. The design of the FPGA is well tailored for synchronizing multiple outputs to ensure the multiple phases for a motor switch at the same time for optimal control. It is common to have multiple motors attached to a single FPGA as most FPGAs have more than enough IO to support two or more motor drives and their feedback. This in turn allows for multiple motors to be synchronized in an application to implement system level control.

The microcontroller interfaces to the motor drivers with two PWM controllers. Each PWM controller generates outputs pulses on four channels independently according to parameters defined per channel. Each channel controls two complementary

square output waveforms with selectable characteristics such as period, duty-cycle, polarity, and dead-times. The channels in each PWM controller can also be synchronized to enable multiple channels to switch at the same time.

The implementation of the PWM controller in hardware replicates the best feature of a FPGA based motor controller with synchronized channel outputs. The PWM controller can also be set up in the Harmony software development framework and updated with short commands in code. Once running, the PWM controller operates independently freeing up the microcontroller to sense the feedback, run the control loops, update the PWM parameters and send telemetry. The PWM parameters are also double buffered to prevent an update from affecting the behavior of the current PWM waveform and only changing the behavior of the next waveform.

D. Control Logic

Many motor controllers implement control loops to optimize the motor movement for the application. The most common is a current control loop that uses the feedback from the current sensing to adjust the PWM duty cycle to meet the current setpoint as shown in Fig. 4. This is important in applications that have precise control requirements or wide temperature variations.

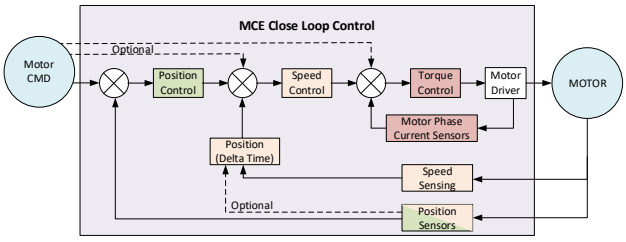

Fig. 4. Simplified Motor Closed Loop Control

Motor controllers for gimbled and servo control applications typically require position and/or velocity feedback. This feedback is used in the outer control loops to drive a current setpoint for the inner current loop to achieve precise motion control.

An ASIC used as a simple stepper only needs to respond to input commands to increment or decrement commutation in an open loop control mode. If the ASIC supports current feedback or a speed input, a control loop is used to PWM the motor driver circuits to achieve the setpoint. In most cases, ASICs perform very simple control loops due to the lack of feedback.

Most control loops are possible to implement in an FPGA. Some FPGAs support soft or hardcore processors to help with floating point math in control loop calculations. With feedback, it is common to see current control loops implemented along with position/velocity control. One advantage of implementing the control loops in FPGA logic is that the user can typically bound the timing performance of the control loop more tightly than is possible in a microcontroller. In some cases, bit depth in calculations is sacrificed due to the limitations of the FPGA.

The microcontroller's strength in motor control is supporting a wide variety of control loops. Because the control loops are written in C code, they can be implemented very close to how they are developed in a motor control simulation. This makes it much easier to verify the implemented control loops and ensure the implementation matches the simulation for a project.

The execution time of the control loops will vary more than in a FPGA based solution, but this is rarely a problem as the microcontroller can run more calculations per second than most FPGA solutions and natively perform floating point math.

The microcontroller also shares a common architecture with other Microchip Arm based microcontrollers. These microcontrollers are commonly used in commercial and industrial applications in much higher volumes than what we see in space applications. This creates a large ecosystem of examples and demos that can be ported to the microcontroller to enable the user to experiment with various control schemes which can save significant time compared to starting from scratch.

E. Logic and Software Configuration

The ASIC is configured by design when it is fabricated. Some ASICs allow some configuration with external jumpers or command inputs.

An antifuse based FPGA is programmed initially once before installation onto a board. Once the FPGA is programmed, it cannot be updated. A SRAM or Flash based FPGA is typically loaded with a bootloader from a saved image in memory. The bootloader can also implement desired features such as redundant software images that are often required for space applications to protect against radiation caused bit flips. These images can often be updated to implement new features, adjust operational/control schemes, and tune the motor control.

The microcontroller can load application software (ASW) directly from internal flash or from external memory through a parallel interface. A ROM bootloader is also available to load software from selected interfaces. The bootloader validates the software's CRC to ensure the software is valid before running. The available interfaces to load software through the bootloader include external SPI memory, or a direct software load on start up through a UART or SpaceWire link. The bootloader also permits the user to load software directly to memory without running ASW first. This feature is helpful to recover from a software issue that prevents the normal operation of software which is typically required to access memory.

One limiting feature of the bootloader implementation is that the user cannot specify a secondary software image to load if the primary image fails. This is often required in critical applications as memory can be corrupted in radiation environments over time. This can be mitigated at a higher system level as the bootloader can be used to write a new software image to replace the corrupted image or by loading software directly from the communication interface.

III. SUMMARY

A. ASICs

The ASIC implementation will typically require less support circuitry and have well established test and verification programs since the motor controller will only operate over a known and defined range.

The initial ASIC development can be very expensive as the developer must complete the process of designing, laying out and then fabricating a custom IC. The design process must choose the interfaces and options to support which then get built into the final motor controller. These choices also fix the interfaces which can be a constraint on the flexibility of the product for future applications.

The main downside of the ASIC are the high initial development costs and the limited flexibility of the design to support new applications as motor control concepts constantly add new combinations of feedback mechanisms, commutation schemes and control logic to tailor solutions to optimize the performance of a system. The main upside of the ASIC solution is that the cost can be much lower if the application's requirements can be met by an existing design.

B. FPGAs

FPGA based motor controllers are the most common solutions for space applications. The flexibility to configure logic enables almost any command input and control scheme. FPGAs also come in many different sizes with different feature sets enabling simple designs to extremely complex exquisite solutions. The main downside of FPGA solutions is the required complexity to add in the various support circuits and the total cost of solution.

C. Microcontroller

The microcontroller can replace the FPGA most designs that controls one or two motors. The integration of the command interface and feedback is typically quicker, and the control loops can be developed to match simulations much more easily. Integrating capabilities into the microcontroller such as the ADC also reduces the complexity and cost of the design while improving performance.

Being able to reprogram the microcontroller also adds a capability normally associated with significantly more expensive FPGAs that can significantly reduce the integration risk.

If a design requires the control of more than two motors, the reduced cost of the microcontroller compared to a larger FPGA may still result in a lower system cost overall. Many sophisticated systems feature multiple motor controllers tied back to a Single Board Computer (SBC) that commands the individual elements achieve a system level movement solution.

One additional benefit of using a microcontroller is that the pool of C software developers is significantly larger than VHDL developers. It is often easier find engineers who have experience on similar platforms and can learn a new one to start developing C software compared to VHDL.

ACKNOWLEDGMENT

Moog offers solutions in all three of these motor control approaches as it seeks to meet the demands of precision control and action needs of the space industry. Moog has built multiple microcontroller-based motor controllers for space applications that control 2-phase and 3-phase BLDC motors and 3-phase stepper motors. Moog's Motor Controller Electronics (MCE) using a microcontroller enables designs that provides a high performance, low cost, and high reliability motor control solution.

REFERENCES

[1] https://www.moog.com/content/dam/moog/literature/sdg/space/spacecraft-mechanisms/moog-2-and-4-channel-ecu-datasheet.pdf

[2] https://www.moog.com/content/dam/moog/literature/sdg/space/avionics/moog-gimbal-electronics-technical-datasheet.pdf

[3] https://www.moog.com/products/controllers-controls-software/space-controllers/spacecraft-controllers.html

[4] https://www.microchip.com/en-us/tools-resources/configure/mplab-harmony