Implement efficient control of multiple Brushless DC motors

Robert Murphy and Greg Verge, Cypress Semiconductor - September 29, 2010

The use of Brushless DC (BLDC) motors is rapidly growing in today's electronic world. Because of their increased use, there is a corresponding need for low-cost methods to control them, especially in applications requiring multiple motors. BLDC motors are found in such consumer electronics applications as power tools and air conditioning units, in transportation vehicles such as electric scooters and bicycles, and in industrial machines that are replacing multiple brushed DC, AC, and universal motors with electronically controlled BLDC motors. This article will discuss the key issues behind designing a system controlling multiple BLDC motors using quad rotor helicopters as an example.

Quad copters are radio-controlled flying vehicles that use four motors to maneuver an airborne vehicle with the precision of a helicopter and the ease of control and maintenance of an airplane. Quad copters are increasingly used both in consumer and military applications. In the consumer market, there is a desire to do this as efficiently as possible to reduce system cost. Current implementations to control the BLDC motors found in quad copters require one microcontroller per BLDC motor. This is due to the overhead of the CPU to consistently monitor and control the motors. At the high speed these motors operate, this can be fairly CPU-intensive to keep up with motor commutation at 30,000 RPM. In most quad copter designs using this approach, five microcontrollers are required, four to commutate the BLDC motors and one to integrate gyro and accelerometer sensors with the radio controller to maintain stable flight. To understand the load that is placed on the motor controllers, let's look at what it takes to control a BLDC motor.

BLDC Motor Control

Common BLDC motors follow a three-phase topology where two phases are driven simultaneously - with one phase driven high and the other driven low - while the third phase is left floating. In order to efficiently drive a BLDC motor, the position of the rotor needs to be detected in order to produce the maximum amount of torque by commutating at the appropriate time. Each time the motor commutates, the rotor will rotate 60 degrees. The commutation sequence for a BLDC motor is shown in Figure 1.
Traditionally, rotor position is detected using Hall-effect sensors to determine when the rotor is aligned with the sensor. The problem with implementing a Hall-effect sensor or other external sensing mechanism is that they add additional cost and increase failure mechanisms, reducing reliability. To decrease motor controller cost, a sensorless design can be used in applications that exceed a minimum speed such as blowers, pumps, and conveyers.

In every commutation sequence, one phase is energized positive, another is energized negative, and the remaining phase is left floating. When the floating phase Back Electromotive Force (BEMF) voltage crosses from positive to negative or vice versa, this is the zero crossing point which occurs 30 electrical degrees after the last commutation point and 30 degrees before the next commutation point. The minimum speed requirement allows the zero crossing to be detected using the BEMF of the motor. By monitoring when this zero crossing point occurs, a BLDC motor can efficiently function without sensors present, producing the same results as a sensored design but at a lower cost. A typical sensorless BLDC schematic is shown in Figure 2.
Almost all microcontrollers available today provide the required peripherals to implement sensorless BLDC motor control. On the input end of the system, a comparator or ADC senses the zero crossing and a timer measures when the zero crossing occurred relative to the optimal 30 electrical degree point. On the output end, a PWM and digital IOs drive the half bridges in the correct commutation sequence. The CPU is triggered by the comparator and timer inputs and calculates the new commutation period to adjust the output drive signals to advance the commutation sequence. A number of motor control-focused MCUs are available that typically provide a multiple channel PWM with motor control optimized features like dead band as well as a quadrature-encoder interface for motion control applications.

The typical limitation on the number of motors a single microcontroller can control is the requirement that the CPU process every commutation cycle. Commutation processing requires prompt reading of the commutation timer after a zero crossing and adjusting the analog inputs to sample the next phase winding in the commutation sequence. The new commutation period must then be calculated by the CPU and transferred back to the timer before the end of that commutation period. When the commutation timer expires 30 electrical degrees after the zero crossing, the digital IO pins must be configured by the CPU to drive the next commutation phase, and then the process is restarted. Each of these CPU based tasks is latency critical and cannot be missed otherwise the motor will miss a commutation event, loose lock with the motors rotor, and cease to rotate. Latency requirements of less than 50us combined with high processing requirements take up to 50% of CPU cycles while also requiring that the highest interrupt priorities be devoted to ensuring correct commutation. In all but the fastest and most costly devices, the CPU is limited to safely controlling only one sensorless BLDC motor. An additional limitation in many devices is the quantity and routing flexibility of analog and digital peripherals to support four independent motors.

**Figure 2: Typical Sensorless BLDC Schematic**

Controlling Multiple Motors
To be able to control multiple sensorless BLDC motors with a single microcontroller, available processing resources need to be able to monitor and control all motors while not affecting the performance of any of the other motors in the design. This is where many designs encounter problems. Since software is used to monitor and control many time critical tasks such as zero crossing detection, monitoring, and adjusting the commutation timer, a fair amount of processing load needs to be offloaded from the CPU in order to accommodate additional motors. A significant portion of the motor load on the CPU can be reallocated to hardware. Additionally, all timing-critical functions can be moved to hardware to eliminate software latency as an issue, therefore leading to better motor controller performance.

There are various portions of the control system that can be reallocated to hardware. To understand what portions those are, break the motor control function into two sections. The first section includes the commutation logic to monitor the BEMF as well as the timing functions for measuring when the zero crossing occurred and determining where the end of the commutation period is located. This section can be thought of as the inputs of the motor controller. The second portion is the drive logic where control of the high-side and low-side transistors is performed and the parameters of the drive PWM are modified to adjust the speed of the motors. This is the output portion of the motor controller.

Starting with the commutation logic shown in Figure 3, there are several modifications that can be made to a design to reduce the CPU load. The first change is for BEMF detection of the zero crossing where the comparator needs to be monitored by the controller to detect the zero crossing point. Each time a new commutation period begins a timer internal to the microcontroller device resets and begins counting. When the output of the comparator triggers, the timer captures the number of clock counts the zero crossing point occurred after the start of the commutation period. This captured value is later used in a control loop for the motor to optimize the commutation period. However, depending on the speed of the motor, this can be highly interrupt-intensive and cause the microcontroller to remain in the Interrupt Service Routine (ISR) for a significant portion of time when it may need to be handling other functions. To reduce ISR overhead, the comparator output can be tied to the capture pin of the timer. Once the zero crossing point is detected, this will capture the time that the zero crossing occurred and store it into a timer register without interrupting the CPU.

Using Direct Memory Access (DMA), we can take the zero crossing time information out of the timer and move it into a location in RAM which will later be used by the CPU calculated control loop. Alternatively, the CPU load can be removed by using CPLD-based programmable logic to process the control loop in hardware. When complete, DMA can transfer the data to the appropriate commutation timers and drive PWMs. By using DMA to move the data through the control system, the CPU is freed from these control loop tasks. Implementing the timer control functionality in hardware eliminates the constant interrupts that require the CPU to stop what it was doing, perform a software capture of the timer, and then move the data into RAM. Optionally, a hybrid approach can be used to simplify the hardware logic by using hardware to calculate a simplified real-time control loop and letting the CPU optimize the hardware control loop parameters at a much slower rate, whenever free cycles are available. Combining this timer logic with the BEMF detections allows a complete commutation cycle to occur with the CPU never needing to be aware that one has occurred until the capture, transfer, and updates are complete.
Implement efficient control of multiple brushless DC motors
For the drive logic portion of the BLDC motor controller shown in Figure 4, the CPU load can be reduced several ways. Traditionally, the CPU is required to keep track of the current commutation state. This functionality can be implemented in a state machine using digital logic. When the timer in the commutation logic expires, it can be used to increment the next commutation state in the sequence. The state machine logic will then select which motor phases are to be driven high, low, and left open. The state logic also implements a hardware-controlled analog multiplexer that switches the comparator input to the high impedance motor phase to detect the rising and/or falling edge of the zero crossing. This reduces the load of the CPU to constantly monitor the computation state and adjust appropriate registers to change the MUX input either internally if using a microcontroller device with internal comparators and routing or externally if using an external analog multiplexor. The only time the CPU will be required to drive commutation drive logic is to produce the initial start commutation sequence to get the motor started and up to the minimum speed at which the BEMF can be used to detect zero crossings. After that, the hardware will take over.
With the relocation to hardware of the time-critical and maintenance tasks of BLDC motor control, the CPU has significant idle time, which can be used to control other motors as well as non-motor system functions. This allows one microcontroller to control multiple motors and still have processing and hardware resources available for other features.

The increasing availability of low-cost, mixed-signal System on Chip (SOC) devices provides the ability to migrate from traditional CPU-dominated commutation algorithms to fully-hardware controlled commutation. The ideal device to implement the required sensorless BLDC controller in hardware has a CPU for high-level functions, programmable logic for low-level motor commutation, digital peripherals with configurable analog routing to switch BEMF phases for sensing, and enough comparators for zero crossing detection of multiple motors. For the example application, the Cypress PSoC 5 family of Programmable System on Chip devices was used due to its ability to integrate the hardware-based commutation engine required for each motor. The device family includes programmable analog routing and functional blocks for analog peripherals, PLD logic to create digital peripherals, and an ARM Cortex-M3 CPU core. Figure 5 shows the quad copter four motor controller built using hierarchical design tools that allow the creation of a single low-level BLDC motor controller component and then placement and configuration of four of these components to quickly provide a working prototype.
Continued advancement in the flexibility and integration of mixed-signal microcontrollers allows for the low-cost implementation of multiple hardware-based sensorless BLDC motor controls in a single microcontroller. In the case of a quad copter design, five microcontrollers were replaced with a single device. With the spare CPU cycles freed up by hardware, the gyro, accelerometer, and radio inputs can now be calculated to provide stable and controlled flight commands to the motors as well. Aspects of configurable hardware-based motor control can be extended to all motor types allowing significant cost saving in many multiple motor designs.

About the Authors

Greg Verge is an Application Engineer Principle at Cypress Semiconductor and graduated from Seattle University and holds BSEE and MBA degrees. For the last ten years he has focused on mixed signal design of Microcontrollers, Systems on Chip and motor control.

Robert Murphy is an Application Engineer at Cypress Semiconductor and graduated from Purdue University with a Bachelors Degree in Electrical Engineering Technology.