

# Take Electronic Motor Drives to the Next Level

When you combine embedded processors on FPGAs with Xilinx motor drive components, there's no limit to what you can achieve.

by Geir Kjosavik  
Embedded Processing Marketing Manager  
Xilinx, Inc.  
[geir.kjosavik@xilinx.com](mailto:geir.kjosavik@xilinx.com)

Electric motors are everywhere. I counted 334 of them in my home, and the number keeps growing. If I were to buy a new PC, for example, I'd have to count many more. Two or three motors drive different cooling fans. The DVD/ROM drive and DVD burner each contain four motors: tray control, spindle drive, pickup sledge drive, and laser focus servo. The hard drive has two motors, and even though floppy drives are no longer included, common peripherals such as force feedback joysticks, printers, and scanners more than make up for their loss.

Outside the home, trains, cranes, and electric cars contain motors that can output anything from 100 to several thousand horsepower. Stationary applications contain big motors as well. Industrial compressors, pumps, and fans are just a few examples.

Different applications also require different types of motors. A small stepper motor can position an inkjet printer head with incredible precision, but it takes a monstrous three-phase AC induction motor to drive a city water supply pump.

For almost every motor, an electronic motor drive circuit controls its speed and torque. The electronics that control the print head motor and the water pump motor are as different as the motors themselves. A \$.50 8-bit microcontroller can drive a stepper motor, while the current drive comprises a few surface-mounted metal-oxide semiconductor field-effect transistors (MOSFETs).

On a water pump motor, the fat copper rails connecting it to the power stage hint at large electric currents. The power stage is based around gigantic insulated gate bipolar transistor (IGBT) transistors that alternate the direction of the current through the motor's three-phase windings.

## **Serious Computing Power Needed**

The computing horsepower required to efficiently control pump motors is almost as impressive as the motor itself – at least to an electrical engineer. At the heart of the

# A clever motor drive design requires that software, power stage, and PWMs work together in perfect unison.

motor drive sits a high-performance CPU, and a busy one at that. Among its many tasks is generating the three 120-degree out-of-phase sine waves on which AC induction motors thrive. It varies the duty cycle of the drive currents at high frequency, and the inductive load “sees” the waveform as a smooth sine wave. This is pulse width modulation (PWM).

The sine waves, amplified many times by the power stage, make the motor spin. The CPU controls the speed and torque of the motor by varying the frequency and amplitude of the sine wave, respectively. However, as in an automobile, holding the accelerator in a fixed position doesn't guarantee travel at a constant speed. This is why some clever engineers invented the cruise control, and also why equally clever engineers came up with electronic motor controls.

## Throw Some Math in There

In simple terms, the CPU controls the speed and torque of the motor this way:

1. Read speed and torque values
2. Compare with desired settings
3. Calculate increments or decrements
4. Adjust speed and torque using values from #3
5. Repeat series from #1

The most commonly used method to calculate the adjustments in step #3 is known as PID – proportional integration and derivation. PID comprises a series of computations that take the differences between desired and current operation as inputs and spit out suitable adjustments proportional to those differences – small adjustments for small deviations, bigger changes if the motor is really off-kilter. You can't just add or subtract the entire difference, because it takes several control loop iterations for the motor to react to a change in input. Such an approach would lead to

instability, with wildly oscillating values.

The motor drive measures the speed of the motor by decoding signals from an optical or magnetic encoder on the motor's drive shaft. A state machine known as quadrature encoder interface (QEI) decodes these signals. Measuring phase currents at certain points in the PWM cycle gives you the torque.

You can then feed other parameters to the control algorithm, such as voltage and current waveform profiles. If you take the latter and throw complex math at it, such as an algorithm bearing the “Star Trek”-like name of flux vector control, you can predict the motor's behavior very accurately without an opto-mechanical or magnetic interface. This is useful when controlling smaller motors, where gadgets like shaft encoders take up too much space and cost more than the incremental CPU power to eliminate them.

Most industrial motor drives are also connected to some kind of network, so when you add the computational powers required to run a TCP/IP or DeviceNET stack on top of everything, performance requirements can reach as much as several hundred MIPS – if you solve your embedded computing needs with brute force. Like hunting sparrows with a rocket launcher, it works, but there are more elegant approaches to meeting performance requirements.

## Look Ma, No Brushes

The AC induction motor is generally considered the workhorse of all industrial motor types. Another popular motor for high-performance applications is the brushless DC motor (BLDC), also known as the permanent magnet AC motor. The apparent confusion arises from the fact that it is built on the principles of a DC motor but operates from an alternating current. The BLDC motor doesn't need a sine wave, but the motor drive must know its exact rotational position at any time, thus calling for a dif-

ferent position encoding/decoding mechanism than AC induction motors. When it comes to controlling currents and voltages, PWMs are still the name of the game.

## Doing It My Way

Because you cannot build a high-performance motor drive without PWMs, many embedded processors and microcontrollers come with PWMs built-in. These PWMs are the result of long discussions between the chip vendor's marketing and engineering departments. It's a battle between customer requirements on one side and die size estimates, design time, and testability on the other. The resulting design is a compromise between what the “average” customer wants and what can be done within the confines of cost and time-to-market requirements.

## A Square PWM in a Square Hole

PWMs come with different features and settings: varying resolution and speed, edge-aligned versus center-aligned, dead time generation, and symmetrical outputs. Most on-chip modules offer software configurations of these parameters, but more often than not, off-the-shelf modules will not meet your exact requirements. Even if they did, that technology would also be available to your competitors. Using a standard module often requires limiting adaptations to the power stage and control software.

A clever motor drive design requires that software, power stage, and PWMs work together in perfect unison. In other words, you need full freedom in designing all three, which effectively rules out on-chip PWMs.

## FPGAs to the Rescue

Custom PWMs = custom logic = FPGAs. With an FPGA, you can also do custom QEI interfaces, an ADC interface, safety circuitry, and timer arrays, all designed exactly the way you want them.

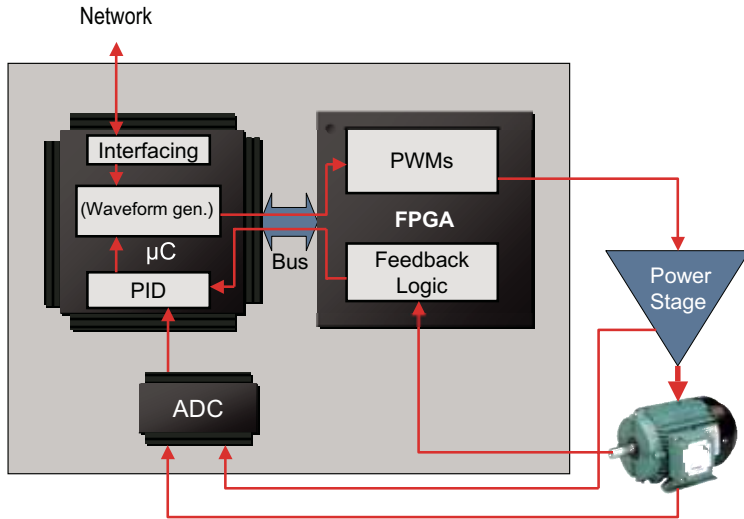


Figure 1 – Block diagram traditional motor drive design

### The Next Step: Embedded Integration

The traditional high-performance AC induction motor drive comprises an FPGA and an embedded processor, as shown in Figure 1. This configuration works well, except that the control loop traverses the bus between the two components twice. Because this bus is often shared with other system functions, performance is indeterministic and easily becomes an unnecessary bottleneck in loop response time.

Xilinx® embedded technology allows you to bring the embedded processor onto the FPGA. The benefits of this approach go beyond fewer components, smaller size, and fewer suppliers. It also improves both performance and design time.

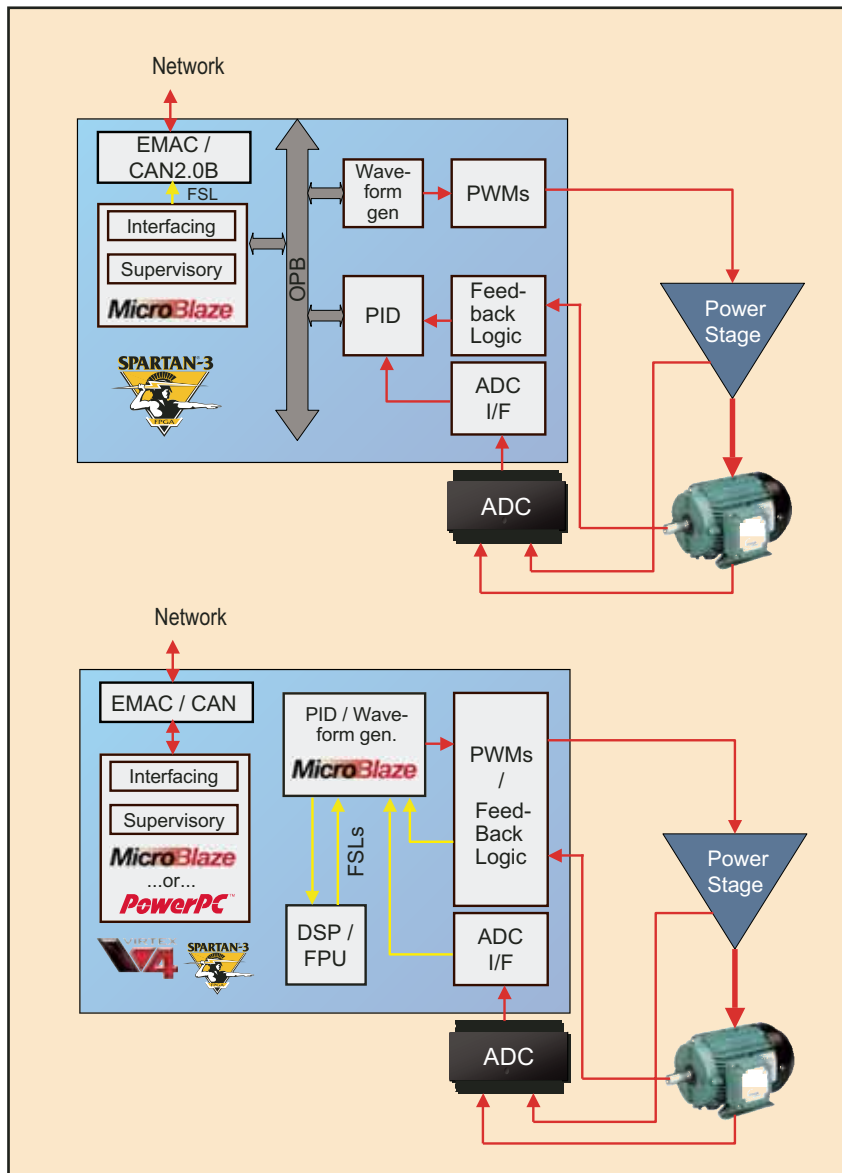


Figure 2 – Two-block diagram of embedded FPGA design

### Co-Processor Interface

It does not matter whether you choose the 32-bit MicroBlaze™ soft-core processor or hard-coded PowerPC™ 405 processor. Both offer some unique benefits through circuitry dedicated to interfacing with on-chip peripherals in the FPGA fabric. Figure 2 shows some design examples in which all control loop data travels to and from the CPU over a dedicated, deterministic link that is not shared with any other resources.

Eliminating bottlenecks in a design is like peeling away the layers of an onion. You design around one hurdle, and the next one reveals itself immediately. Now that an important hardware pipe has been effectively unclogged, you should not be surprised to see a potential for improvements in your software. Figure 3 shows the software analyzer that comes packaged with the Xilinx Embedded Development Kit (EDK). This tool could prove exceptionally helpful in identifying resource-consuming functions. Motor control software, like anything else, is likely to have its share.

### Intelligent Versus Brute Force

I have already mentioned the brute-force approach to designing embedded systems. FPGA processors with a co-processing interface effectively put an end to throwing raw MIPS at any performance

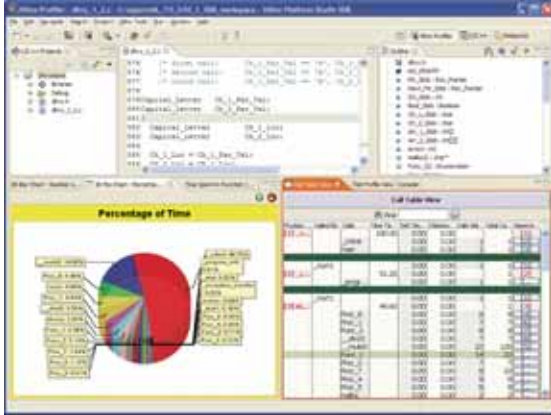


Figure 3 – Software profiling screen shot

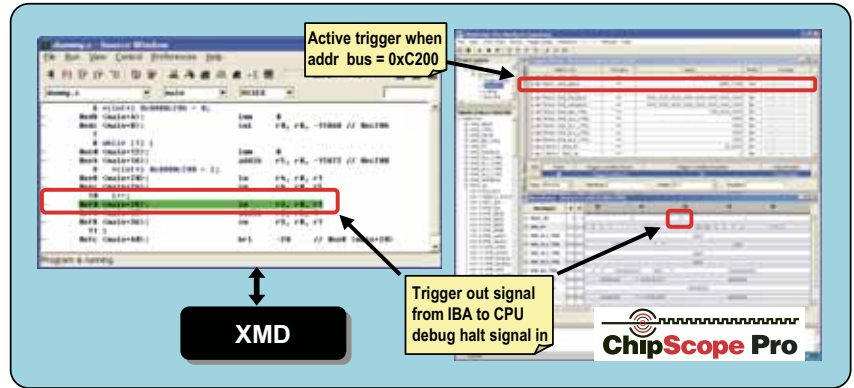


Figure 4 – ChipScope screenshot

challenge. A carefully designed hardware/software mix can reduce CPU performance requirements by several orders of magnitude.

Typical software bottlenecks in a drive are floating-point math functions, DSP functions, and of course, the PID function. As the designer, you should define the optimum mix between hardware and software modules. The FPGA gives you full freedom to balance this any way you want.

### A New World of Debugging

The traditional design split of an embedded processor plugged onto an FPGA can be a nightmare to debug. With two different sets of debugging tools probing two different chips, visibility of interaction between the two is extremely limited.

The Xilinx ChipScope™ Pro analyzer lets you debug software and hardware inter-

action like never before. You can set the debugger to trigger when the base counter inside the PWM reaches 0x3FF and observe what code lines were executed within a 50 μs window around that point – without halting execution. Or set the debugger to trigger on each Phase X zero crossing in the sine table and observe the resulting PWM waveform over the next 5 ms.

With full visibility of any chain of events within the system, you can track down even the most elusive bugs in minimum time.

### Motor Control Components Ready to Go

As much as we at Xilinx recognize the fact that you are the drive design experts, we do provide a few bits and pieces to give you some ideas of what's possible with embedded processing on FPGAs.

Our reference design, “Spinning Wheels,”

contains complete implementations of one BLCD and one AC induction motor drive. The IP of the solution is openly available as VHDL source code:

- BLDC drive unit
- Hall effect BLCD decoder
- AC induction drive unit
- QEI

The solution comes packaged with reference designs that include MicroBlaze integration with C code examples, tutorials, application notes, and simulation test benches if you don't have a power stage and a motor handy at all times.

Take a look at the blocks and play with them to get a feeling for what's possible. Once you have pieced them together, feel free to sprinkle the design with an FPU and a CAN2.0B interface, or whatever it takes to make it the perfect drive.

### Conclusion

Motor control design is tough work. The quest for smarter, faster, and more power-efficient designs imposes an enormous strain on software and hardware engineers. By adopting FPGAs with on-chip embedded processors, you are free to implement the creative design ideas needed to pull ahead of your competitors. With full flexibility in PWM design and control over the hardware/software split in the control functions, the possibilities are endless. ●●

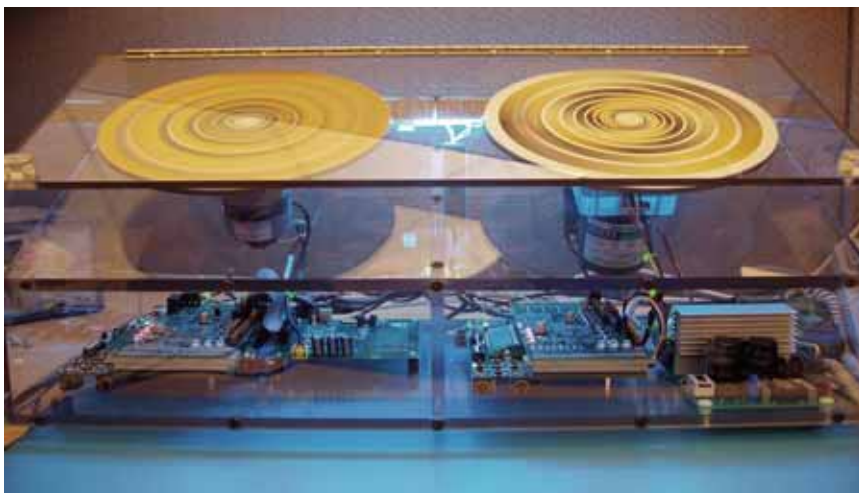


Figure 5 – Spinning Wheels reference design