Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

New Mechatronics Development Techniques for FPGA-Based Control and Simulation of Electromechanical Systems

Brian MacCleery * **and Zaher M. Kassas** **

* *Industrial Embedded Design Senior Product Manager, National Instruments, Austin, TX (e-mail:* brian.maccleery@ni.com*).*
** *Control Design and Simulation R&D Engineer, National Instruments, Austin, TX (e-mail:* zkassas@ieee.org*)*

Abstract: Field programmable gate arrays (FPGAs) have been widely adopted in high volume commercial applications, but not as much in the industrial control and simulation arenas. Due to the attractive features of FPGAs, such as their inherent flexibility, performance, parallelism, and low-level reconfigurability, industrial control design and simulation vendors have been creating the next generation FPGA development tool chains that are designed for engineers with little or no digital design expertise. The goal of these next generation system-level design tools is to empower control design, simulation, and signal processing engineers to harness the full power of the FPGA technology, while providing relatively competitive performance and resource usage, as compared to traditional text-based hardware description level (HDL) methods. This paper discusses some of the traditional challenges that prohibited wide adoption of FPGAs in the industrial control and simulation fields, and how new graphical system design tools are helping mechatronics engineers leverage the full power of FPGAs as deployment platforms. Moreover, it discusses some particularly useful development techniques for FPGA-based control and simulation in mechatronics applications.

Keywords: Mechatronics, hybrid simulation, FPGA, implementation

1. INTRODUCTION

Field programmable gate array (FPGA) technology is gaining more popularity and is increasingly being adopted in several industrial applications, Mears et al. [2006], Gomez and Goethert [2006], Dase et al. [2006]. This is due to different factors, such as their inherent flexibility, performance, parallelism, and low-level reconfigurability, besides the fact that their price is rapidly decreasing. FPGA chips combine software programmability and field upgradability with the performance, robustness, and flexibility of a custom circuit design. Unlike processors based on the Von Nuemann architecture, which provide a fixed instruction and register set, FPGA devices provide a reconfigurable matrix of elemental computing elements that can be arranged and optimized for a particular computing task. Recent benchmark reports suggest that some FPGAs may have recently bypassed processors and digital signal processors (DSPs) with respect to cost and power consumption per signal processing unit, BDTI [2007].

FPGA compilers use a process called synthesis to convert processing logic and algorithms into a highly optimized digital logic design that gets implemented at the hardware level in silicon gates; thus, effectively translating the design requirements from the software domain into the hardware domain. This results in a true parallel processing implementation of the design, with the ability to partition and reconfigure processing resources for each task in the application. For example, one portion of the FPGA device may be reading motor current by communicating with analog to digital converters (ADCs) via the serial peripheral interface (SPI) communication protocol, while another portion of the chip is executing a proportional-integral-derivative (PID) control algorithm, and a third section is reading hall effect sensors and is performing brushless motor commutation logic.

Many processing tasks can be executed within a single clock tick of the FPGA; hence bringing computation and event response speeds down to the order of tens of nanoseconds. Unlike traditional processor or DSP devices, in which the chip vendor pre-determines the instruction set, I/O interface resources, and configuration registers; the FPGA can be customized down to the silicon level by the designer. Fig. 1 illustrates a typical FPGA chip architecture.

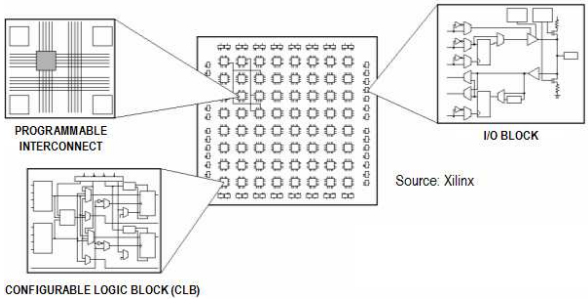

Fig. 1. Internal architecture of a typical FPGA chip

The rest of this paper is organized as follows. Section 2 discusses some of the traditional challenges that prohibited the wide adoption of FPGAs in industrial control and simulation applications, and how such challenges are being resolved with the evolution of new development tools for FPGAs. Section 3 discusses some of the necessary characteristics that FPGA simulation environments for mechatronics applications should possess. Section 4 demonstrates some FPGA simulation techniques for timing and triggering. Section 5 outlines some useful techniques for developing FPGA intellectual property (IP) blocks, whereas Section 6 discusses experimental validation results. Concluding remarks are discussed in Section 7.

2. TRADITIONAL CHALLENGES AND THEIR RESOLUTIONS

Despite the attractive features of FPGAs, their adoption by industrial control and signal processing engineers has been slower than processors and DSPs. This is due to several factors. First, these engineers traditionally programmed processors and DSPs using higher level languages, such as C. However, FPGAs possessed complex development tool chains that required designs to be specified using hardware description level (HDL) and register transfer level (RTL) semantics. Furthermore, traditional FPGA development tools lacked intellectual property (IP) blocks for common industrial applications, such as ADC and encoder interface logic, pulse-width modulation (PWM) and commutation logic, timing and triggering functions, PID control algorithms, memory management, and data transfer functions. In addition, FPGAs natively supported integer data types only, which significantly increased development complexity for analog control and signal processing applications that required math, control, and DSP algorithms to be implemented using floating point data types. Also, traditional FPGA simulation tools operated at the digital signal level and were not interoperable with the type of dynamic simulation tools used by control systems and signal processing engineers for modeling continuous-time dynamic systems. Moreover, FPGAs compilation times were relatively long, as compared to processors and DSPs. For example, typical FPGA compilation times today range from 15 to 90 minutes, whereas processor and DSP compilations are typically completed in less than one minute. Finally, the sequential text-based semantics of traditional register level development tools made it relatively difficult to specify timing and concurrency among parallel processing tasks in a way that leverages the inherent parallel processing capability of FPGA devices, Lee and Neuendorffer [2004].

Despite these traditional development challenges, the successful adoption of FPGAs in application areas such as consumer electronics, and the resulting drop in the price of FPGAs has spurred the interest of industrial control design and simulation vendors. Such vendors are creating the next generation FPGA development tools that are designed for engineers with little or no digital design expertise. The goal of these next generation "system-level" graphical design tools is to empower control, simulation, and signal processing engineers to harness the full power of the FPGA technology, Falcon and Trimborn [2006]. Graphical system design tools are intended to provide a more intuitive, high level programming paradigm that simplifies the creation of complex parallel processing and control applications. Also, they are intended to provide relatively competitive performance and resource usage, as compared to traditional text-based HDL development tools.

Graphical data flow programming languages are a natural fit for FPGA development due to their inherent sense of parallelism and concurrency that intuitively maps to hardware design. Also, recent technological advances are enabling designers to place their FPGA code within a high level dynamic simulation environment. This ability to cross the boundary between the digital domain of the FPGA and the analog multi-physics domain of the system is facilitating a "true" mechatronics approach to development, in which the complex interplay between FPGA silicon logic, power electronic components, electric motor drives, and mechanical systems can all be simulated in a virtual environment without the need to wait for long FPGA compilations. The ability to quickly iterate and optimize the FPGA logic design in a mechatronics simulation environment, combined with the new high level programming tools for FPGAs is lowering the barriers that prohibited wide adoption of FPGAs in industrial control.

In addition to the improved design and simulation capabilities for FPGA designers, the next generation tools are providing a rapidly growing library of IP blocks for common control and DSP algorithms through online code sharing services, NI [2007a]. For instance, the table below exemplifies a snippet of the library of available IP through online code sharing, whereas Fig. 2 shows some of the IP in the LabVIEW FPGA math and signal processing palette.

Furthermore, new fixed point math tools are becoming available, which is simplifying the development of complex control and DSP applications. These factors are significantly lowering the complexity barrier for developing custom sophisticated math, control, and signal processing IP on FPGAs, NI [2007b].

Category	Example
Mathematics & DSP	DC & RMS measurements, custom digital filters
Data Acquisition	Multirate analog output, 64-bit counters, PWM
Signal Generation	Sine wave, white noise, look-up tables
Linear & Nonlinear Control	PID, Zero-order hold backlash, relay, saturation
Communication Buses & Digital Protocols	SPI, Huffman decoder, I2C custom serial protocol, SSI
Sensor Simulation	Thermocouples, resolvers, LVDT sensors

3. A MECHATRONICS APPROACH TO FPGA SIMULATION

3.1 Background

Mechatronics-oriented design attempts to synergistically combine the disciplines of mechanical, control, electronics, and embedded computing. As such, mechatronics development tool chains should allow the developer to transition seamlessly from one discipline to the other throughout the

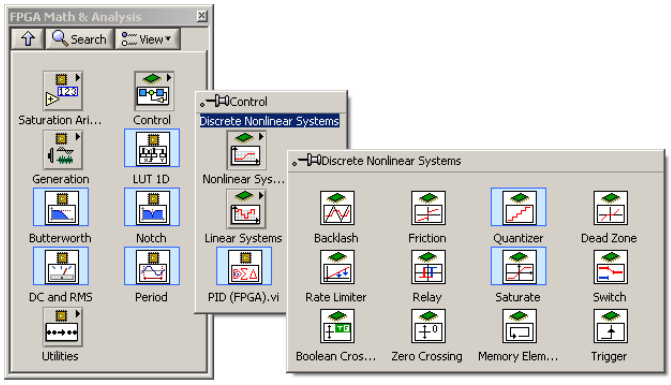

Fig. 2. LabVIEW FPGA Math and Signal Processing palette. Each subpalette contains several IP blocks that are functionally grouped. Shown (expanded) is the Discrete Nonlinear Systems subpalette

design cycle, which typically consist of four stages, namely modeling and identification, design, simulation, and deployment and implementation. Fig. 3 depicts the different engineering domains that require simulation capability to enable a truly multidisciplinary approach to embedded system design.

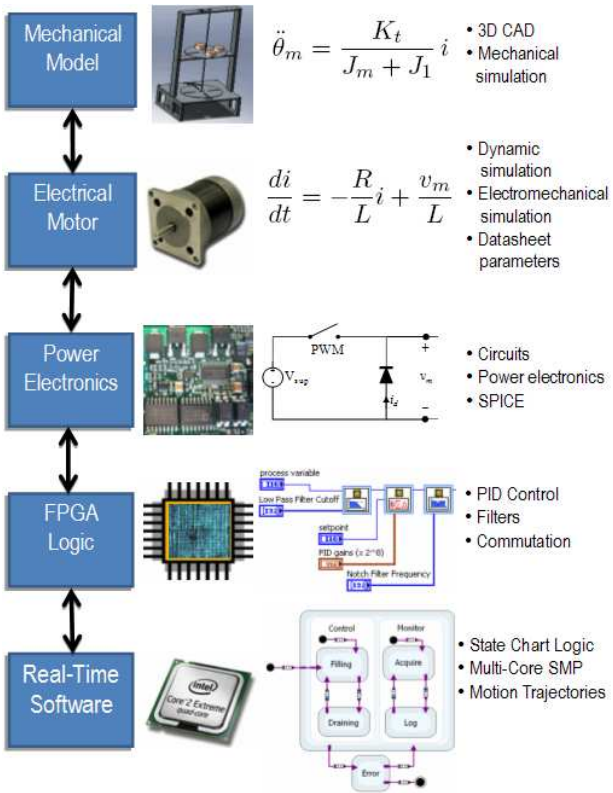

Fig. 3. Typical mechatronics design tools

As the adoption of FPGAs as deployment platforms for mechatronics systems becomes more widespread, design and simulation tools must be able to simulate the coupled dynamics between the FPGA logic and multi-physics systems more tightly. Traditionally, HDL simulation tools provided information about the clock, digital bus, and signal level timing performance of the FPGA logic. Unfortunately, it has been often labor intensive, if not impossible, to validate system level behavior across the multi-physics domain boundaries between FPGA hardware logic, software code running in a processor, ADC converters and signal conditioning circuitry, insulated-gate bipolar transistors (IGBTs) and metal-oxide semiconductor field-effect transistors (MOSFETs), electric motors and hydraulic actuators, and mechanical assemblies. Validating the design of such FPGA-based control systems requires a simulation environment that can span the domain boundaries to accurately model the coupled dynamic behavior of the interconnected electromechanical system. This mechatronics approach to design enables the designer to simulate the FPGA in the context of the dynamical world in which it is embedded, as well as to examine the complex interplay between the FPGA logic inside the chip and the outer electromechanical world.

3.2 Mixed Continuous/Discrete Simulation

A special requirement for electromechanical simulation is to enable FPGA logic validation with co-simulation of continuous-time electromechanical systems like motors, drives, hydraulics, and pneumatics, Dase et al. [2006]. In this respect, the FPGA code must simulate in a functionally-correct and time-correct manner in a mixed continuous/discrete time simulation environment containing variable time-step solvers. Designers must be able to specify the discrete execution rate of the FPGA block and run the code in the simulation environment without skipping execution cycles. This is sometimes referred to as "cycle-by-cycle" or "cycle accurate" simulation.

For example, consider the simulation of a brushed DC motor, where the electrical model (motor armature circuit) and mechanical model (motor torque and inertial load) are characterized, respectively, by the differential equations

$$\frac{d\,i(t)}{dt} = -\frac{R}{L} i(t) - \frac{K_e}{L}\frac{d\theta(t)}{dt} + \frac{v_m}{L}$$
$$\frac{d^2\theta(t)}{dt^2} = \frac{K_t}{J} i(t) - \frac{B}{J}\frac{d\theta(t)}{dt},$$

where $i(t)$ is the armature current; R and L represent the resistance and inductance of the motor armature circuit, respectively; K_e is the motor back electromotive force (emf) constant; $\frac{d\theta(t)}{dt}$ is the shaft velocity that results in a back emf voltage; v_m is the motor terminal voltage; K_t is the torque constant; J is the rotor inertia; and B is the friction torque constant; see Fig. 4.

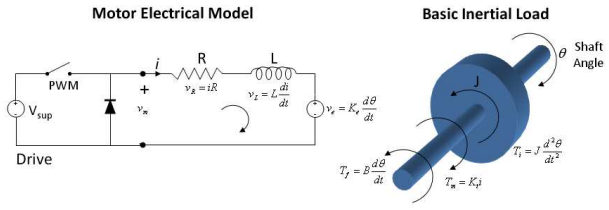

Fig. 4. Electrical and mechanical models of a brushed DC motor

In this particular example, the simulation environment, illustrated in Fig. 5 and 6, is simulating the DC motor dynamics by solving the differential equations through a Runge-Kutta 45 variable time-step solver. On the other

hand, the PWM FPGA block (subVI) has been set to run at discrete-time instants at a period of 25×10^{-9} seconds. Fig. 7 shows how to set the simulation solver parameters as well as the timing of the different blocks inside the simulation node.

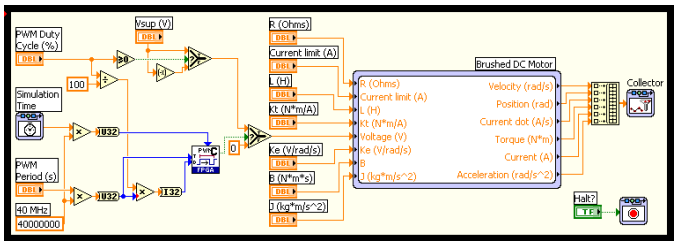

Fig. 5. LabVIEW Simulation block diagram of a brushed DC motor

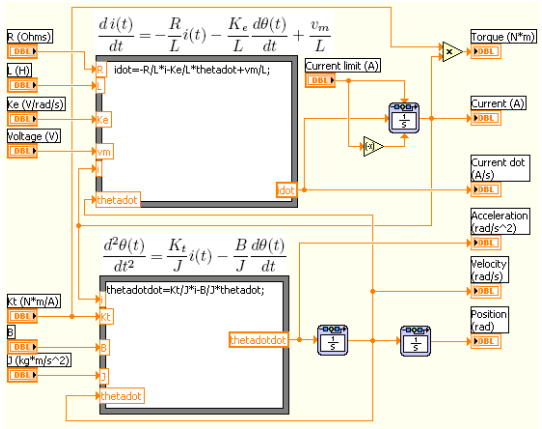

Fig. 6. Simulating the differential equations of the brushed DC motor

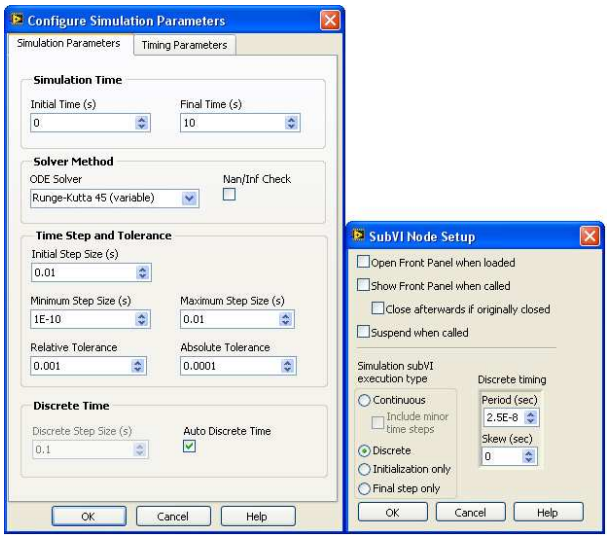

Fig. 7. Simulation solver parameters and subVI node setup

3.3 FPGA Code Validation Using Functional Simulation

Next generation FPGA development tools, such as LabVIEW FPGA, use the same programming language for programming the FPGAs as they do for programming general purpose operating systems, such as Windows. This means that the FPGA functions can be dropped into a Windows application for test-bench simulations to validate the functional behavior of the FPGA code. For functional simulation, the FPGA code must behave in a logically equivalent manner when running in a Windows test-bench to when the same code gets deployed to the FPGA target. This enables the designer to validate the code and test it using the full suite of debugging capabilities found in the development tool chain for Windows code development. Using functional simulation techniques, designers can be confident that their FPGA logic is correct before compiling the code to the FPGA. In particular, functional simulation helps identify logical and timing mistakes in the FPGA code that would be difficult to capture during full speed execution in hardware.

4. FPGA SIMULATION TECHNIQUES FOR TIMING AND TRIGGERING

4.1 Simulating the FPGA Clock

Since FPGA functions typically depend on a clock timer, the simulation environment must have a way to pass a simulated clock signal to the FPGA function block to mimic the hardware timer. For instance, a PWM function uses a timer to determine when to switch its digital output on and off. Fig. 8 shows how to simulate the FPGA clock. In this respect, the "Simulation Time" clock is getting multiplied by 40 million and converted to a 32-bit unsigned integer for use by the PWM FPGA function during simulation. Within the FPGA code a "Conditional Disable" structure is used. This causes the compiler to use the simulated clock signal when running under Windows. When the same code is compiled for the FPGA, however, a 40 MHz FPGA hardware clock is used and no unnecessary logic for the simulated clock signal is instantiated in the FPGA fabric.

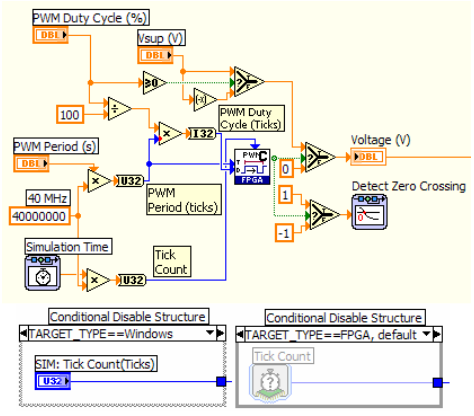

Fig. 8. Simulating the FPGA Clock and converting from floating point to integer values

4.2 Capturing Discrete Events

Many FPGA blocks base their execution time on digital events, rather than on a fixed discrete-time step. Hence, the simulation environment must also have the ability to trigger execution based on events that occur at arbitrary

times during the simulation, Rabbath et al. [2000]. In Fig. 8, a "Detect Zero Crossing" function is used to help the dynamic simulation tool capture asynchronous discrete-time events, such as the Boolean transition of the PWM digital output. For variable step-size solvers, this function forces the differential equation solver to evaluate and update the simulation output when the discrete event occurs.

4.3 Triggering Logic

The simulation environment must be able to accurately model the triggering logic that is used to time and synchronize the execution of the various subsystems in the FPGA application. For example, a typical motor control application involves a function for PWM, current sampling, PID control, and encoder feedback processing. In a typical loop synchronization scheme, the current is sampled at the midpoint of the PWM high-time, based on the rising edge of the "Sample Clock" signal, while the PID function is executed near the end of the PWM cycle on the rising edge of the "PID Clock" signal. Sampling the armature current at the mid-point of the PWM high time provides an approximation of the average current, since the current ripple in a motor is roughly triangular in shape, Song et al. [2000], see Fig. 9.

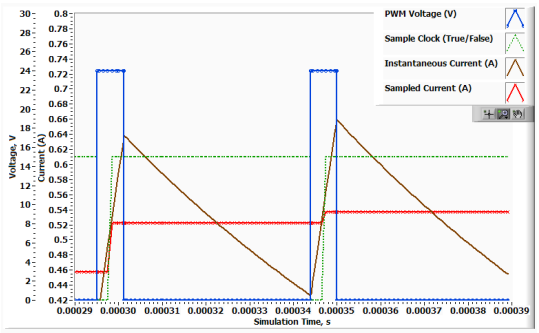

Fig. 9. Motor current is often sampled asynchronously at the mid-point of the PWM high-time

Fig. 10 shows the FPGA logic for acquiring the motor armature voltage and current after the rising edge of the "Sample Clock" signal (Sample CLK) occurs. In order to create a high fidelity simulation of the motor control circuit, the FPGA simulation tool must be able to support this type of high precision timing and triggering logic.

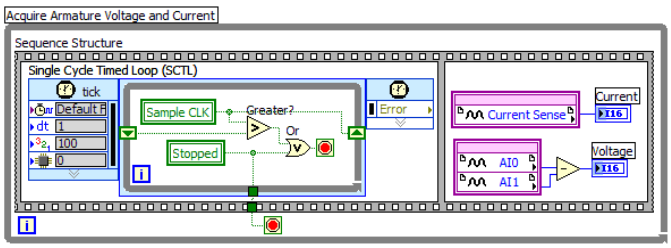

Fig. 10. FPGA code for sampling voltage and current on the rising edge of the "Sample CLK" signal

5. FPGA IP DEVELOPMENT TECHNIQUES

FPGA chips might be considered the ultimate multi-core processors. Due to their parallel processing capability, FPGAs provide additional degrees of freedom compared to processors; the design may be optimized either for size (gate resources) or speed (loop rate). Unlike multiple execution threads running on a single processing core, the parallel processing operations running in an FPGA do not compete for resources with other functions so there is no penalty for running each of the parallel loops at the fastest possible rate. Typically, FPGA applications are written as a set of parallel functions, where the code for each function block acts like its own processor "core". The extremely fast execution rates of the FPGA logic can be attributed to the fact that the processing resources are synthesized and optimized by the compiler to meet the unique requirements of the particular computing task. In effect, the instruction set of the FPGA is customized for each processing task. Typically, the processing performance of the FPGA is orders of magnitude faster than the bandwidth of sensors, actuators and ADCs. Thus, FPGA developers typically spend their time optimizing the application to minimize gate resources rather than optimizing for speed. In an FPGA, the developer can partition the application into an unlimited number of parallel processing cores up until the FPGA gates are completely consumed. However, if processing speed is sufficient, and the developer wishes to reduce FPGA gate resources, the code can be implemented in a "single core" implementation in which a single instance of the processing logic is implemented rather than multiple parallel instantiations.

Some of the key development techniques that should be kept in mind, while developing FPGA IP are discussed next. First, modularize the code into blocks that can be tested individually. For example, encoder interface, PWM, PID, filtering, and ADC/DAC interfaces should all be encapsulated as independent blocks. This enhances code modularity, portability, and testability. Second, do not embed input/output (I/O) nodes for ADCs and DACs in the IP core function block. This way, the IP can be easily tested with simulated I/O signals and moved from one system configuration to the next. Third, feed-in a simulated FPGA clock time signal when running the code in a Windows simulation. This enables emulating the precise cycle-by-cycle timing of the FPGA code when running the code in the simulation environment. Fourth, use a state-machine architecture to capture a time-based or event-based triggering, rather than using hard-coded delay timers. If a hard-coded delay timer is used, the execution of the function block is halted while waiting for the elapsed time delay to occur, which makes the function unresponsive. Hard-coded delay timers blocks can also complicate the cycle-by-cycle simulation of the FPGA code, since they do not typically have the ability to use simulated clock time. Fifth, use a parallel approach if speed is desired and a serial approach if gate resource minimization is desired. For the latter, a "multichannel" or "multiplexed" approach can be used to dramatically reduce gate resources. In a multichannel implementation, a single instance of the signal processing logic is used to reduce FPGA gate resources, while the configuration, state, and I/O data for each channel is swapped into and out of memory. In a recent development project, a single-channel proportional-integral-derivative (PID) algorithm was ported from a single channel implementation to an optimized multichannel configuration; enabling up to 256

PID channels to be implemented in just 30% of a 1M gate Virtex-2 FPGA device at PID loop rates exceeding 30 kHz for all channels. This translates to nearly 100-fold reduction in gate usage per PID loop compared to the single channel implementation.

6. EXPERIMENTAL VALIDATION

The techniques outlined in Section 5 were employed recently in the development of a cascaded PID control system that provides high speed multi-axis coordinated motion control and commutation for three brushed DC motors using the NI CompactRIO reconfigurable control system. CompactRIO is an off-the-shelf FPGA-based control and acquisition system which contains a floating point processor, programmable FPGA, and hot swappable I/O modules full-bridge motor drive modules. This FPGA application includes function blocks for the optical encoder interface, PWM commutation, cascaded current and position PID control loops, spline position interpolation, and interrupt request (IRQ) signal generation. The real-time processor provides path planning and coordinated motion trajectory data for three-dimensional straight-line, arc, or contour motion profiles. The FPGA-based cubic spline interpolation algorithm enables the FPGA control system to execute at rates up to 200 kHz, by providing smoothly interpolated position set point commands in between slower motion trajectory updates. Simulation provided the ability to quickly and safely test the FPGA logic under a variety of worst-case and extreme scenarios.

By inserting the FPGA logic into the dynamic simulation model and validating its performance using a simulated physical system, the developers were able to gain insight into the system that otherwise would be difficult or impractical due to real-world physical limitations, such as the sampling rate of the current sensor ADC and the resolution of the optical encoder sensor. This enabled several logical mistakes in the original timing and triggering scheme to be detected. These simulations provided the motivation for the design of the "PID Clock" and "Sample Clock" timing scheme. Simulation results predicted that the PWM midpoint sampling method should provide an adequate estimate of average motor torque even for small inductance motors that demonstrate large fluctuations in the motor current signal during the PWM cycle.

Once an actual motor drive module was available for prototyping, experimental results were compared to simulation results for design validation purposes. Comparison against the simulation benchmark revealed several flaws in the electronic design of the motor commutation circuitry, which resulted in improved performance and current sensing ability for the final product without increasing cost or delaying the development schedule. Fig. 11 shows a typical simulation test bench result used to validate the design of the FPGA-based multi-axis coordinated motion control and commutation system.

7. CONCLUSION

This paper presented the traditional challenges that have prohibited wide adoption of FPGAs in industrial control and simulation applications along with proposed resolutions. Also, it presented several development techniques

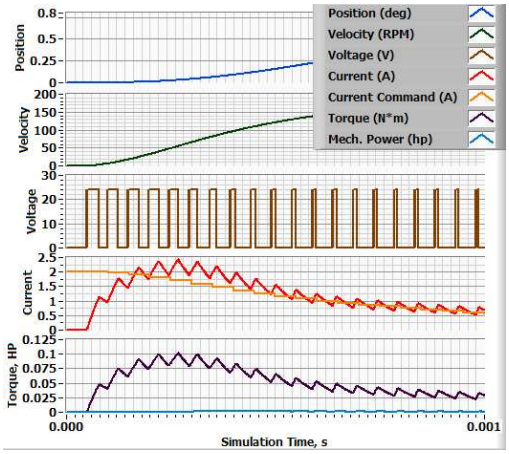

Fig. 11. Electromechanical simulation test bench application used to design a robust FPGA-based motor control and commutation system

for FPGA control and simulation of mechatronic systems. These techniques enable faster design iteration cycles by avoiding long FPGA compilation times and provide robust debugging capabilities for FPGA code by using software as the debugging scope within a dynamic electromechanical simulation environment. By using a mechatronics approach, the design can be optimized across engineering domain boundaries and engineers can reduce risk by assuring robust application performance over a wide range of simulated operating conditions.

REFERENCES

BDTI. *Berkeley Design Technology Inc. Focus Report: FPGAs for DSP*. Berkeley, CA, 2nd edition, 2007.

C. Dase, J.S. Falcon, and B. Maccleery. Motorcycle control prototyping using an fpga-based embedded control system. *IEEE Control Systems Magazine*, 26:17–21, 2006.

J.S. Falcon and M. Trimborn. Graphical programming for field programmable gate arrays: applications in control and mechatronics. In *Proceedings of the American Control Conference*, pages 1394–1400, Jun. 2006.

A. Gomez and E. Goethert. Control system design using labview fpga for a digital picture kiosk. In *Proceedings of the American Control Conference*, pages 1406–1409, Jun. 2006.

Edward A. Lee and Stephen Neuendorffer. *Formal methods and models for system design: a system level perspective*. Kluwer Academic, Norwell, MA, 2004.

M.L. Mears, J.S. Falcon, and T.R. Kurfess. Real-time identification of sliding friction using labview fpga. In *Proceedings of the American Control Conference*, pages 1410–1415, June 2006.

NI. *LabVIEW FPGA Intellectual Property (IP) Network*. 2007a. URL `www.ni.com/ipnet`.

NI. *NI Labs Experimental Prototype Pre-Release*. 2007b. URL `www.ni.com/labs`.

C.A. Rabbath, M. Abdoune, and J. Belanger. Effective real-time simulations of event-based systems. In *Proceedings of the 2000 Winter Simulation Conference*, pages 232–238, Orlando, FL, Dec. 2000.

S.H. Song, J.W. Choi, and Sul S.K. Current measurements in digitally controlled ac drives. *IEEE Magazine on Industry Applications*, 6:51–62, 2000.