XXII Simpósio Internacional de Engenharia Automotiva
SIMEA 2014

Blucher Engineering Proceedings
Agosto de 2014, Número 2, Volume 1

CREATING NEXT GENERATION HIL SIMULATORS WITH FPGA TECHNOLOGY

Chris Washington and Jordan Dolman

National Instruments

E-mails:chris.washington@ni.com, jordam.dolman@ni.com

ABSTRACT

As embedded control devices become more common in today's electro-mechanical systems, HIL Simulation is growing in its importance to the success of these systems. HIL testing provides a simulated environment for the unit under test, simulating the parts of the system that are not physically present. As these systems grow in complexity, traditional HIL simulation techniques are falling short. Fortunately, technologies such as Field Programmable Gate Arrays (FPGAs) are being applied to produce the next generation of HIL simulators.

FPGAs enable test system developers to create custom hardware that can be easily reconfigured without physically modifying the device. In addition to being reconfigurable, for certain applications, FPGAs can offer superior performance compared to microprocessors.

More specifically for HIL test systems, FPGA-based I/O devices provide superior determinism, on the order of nanoseconds, enabling realistic simulation of plant components not typically realizable with microprocessor-only based systems. They are also used to off-load some of the processing that would otherwise be required of the test system microprocessor increasing the total system bandwidth. Because of the ease with which their personalities can be reconfigured, FPGAs are also used in HIL test systems to create custom IO interfaces as well as IO interfaces that can adapt to multiple UUT types or changes to UUT interfaces that evolve during product development.

In this paper, we will discuss examples of how FPGAs are being used for sensor simulation to create better, more adaptable HIL test systems.

INTRODUCTION

The greatest benefit to simulating sensors is the ability to push past the operational limits of a specific environment and test fault conditions that would otherwise be damaging or dangerous. Corrective changes to system components can also be implemented and tested without fear of destroying expensive equipment. An engine control unit (ECU), for example, can be verified without running an actual engine at high temperatures for extended periods of time. The signals being simulated can range from simple analog waveforms to custom digital protocols, and in the past, each type of sensor required its own piece of custom hardware with a dedicated microcontroller for concurrent simulation. This architecture tends to be very costly and expensive to maintain. By taking advantage of inherent parallel processing, field-programmable gate array (FPGA) hardware provides the performance and flexibility to simultaneously simulate a variety of sensors in real-time.

1. FPGAs FOR SENSOR SIMULTION

1.1. Benefits

FPGA-based hardware is ideal for sensor simulation, primarily because of the ability to adapt to multiple sensor types with precise timing requirements. Each sensor output can be customized down to nanoseconds, and various signals can be completely synchronized to realistically create a specific state of operation. In many cases, however, sensors function independently and update at different rates. The true parallel nature of FPGAs also allows dedicated blocks of silicon to operate without any interference from other parts of the application.

Deterministic operation is essential for sensor simulation, in order to accurately characterize the performance of the controller. A processor-based approach will typically use a real-time operating system to schedule and prioritize all parts of an application, since only one operation can execute at a time. Different tasks must compete for processor time, and can often preempt one another. This can severely affect the deterministic response required when trying to simulate sensors. By embedding code on an FPGA chip, sensor logic can achieve the maximum level of determinism, with true hardware-timed reliability.

While the majority of sensors produce an analog signal based on their measurements, there are many sensors that convey information digitally, using methods like pulse width modulation or serialized protocols. An FPGA-based approach can easily integrate the processing required to generate complex digital signals as well as arbitrary analog waveforms without affecting the performance of other tasks in the application. A common example is SPI communication, in which sensors pass data values serially at high-speeds. Each bit on the data line is latched using a master clock line, and then translated into engineering units based on the specifications of the sensor. Other types of digital output sensors could use I2C, RS-232 or even custom digital protocols. FPGAs also include embedded block RAM, in which look-up tables can reside for translating sensor values into data all in real-time. Once developed, the function blocks for specific protocols can then be reused in different parts of the FPGA application and none of the processing required to simulate these digital sensors will affect the update rates of other analog sensors being simulated.

1.2. Challenges

As performance demands increase, many applications will find that typical computer software is no longer effective. When an application necessitates increased speed and efficiency, many engineers consider hardware implementations. While there are obvious advantages to using FPGA hardware for various sensor simulations, there is a clear question of implementation for those who might be inexperienced in FPGA programming. Historically, FPGA technology has been limited to hardware design engineers with in-depth knowledge of hardware description languages (HDLs). Many experts in the field of Automated Test Equipment (ATE), however, have little or no background in FPGA development, seldom having knowledge of mainstream HDLs like Verilog or VHDL (VHSIC Hardware Description Language). When it comes time to start programming the FPGA, the need for higher-level tools becomes quite apparent. Traditionally, these test engineers did not have the means to simulate a sensor output and may have been forced to use the actual sensor for validating controller prototypes. As FPGA technology grows in popularity, industry needs to give domain experts and design engineers alike a higher-level language for programming FPGAs. Take, for example, the different levels of abstraction for computer programming, and notice how going from programming in assembly to C++ and beyond has enabled more people to create increasingly complex software. Continued abstraction of hardware is equally as crucial for continued innovation and increased accessibility for a larger pool of potential FPGA users.

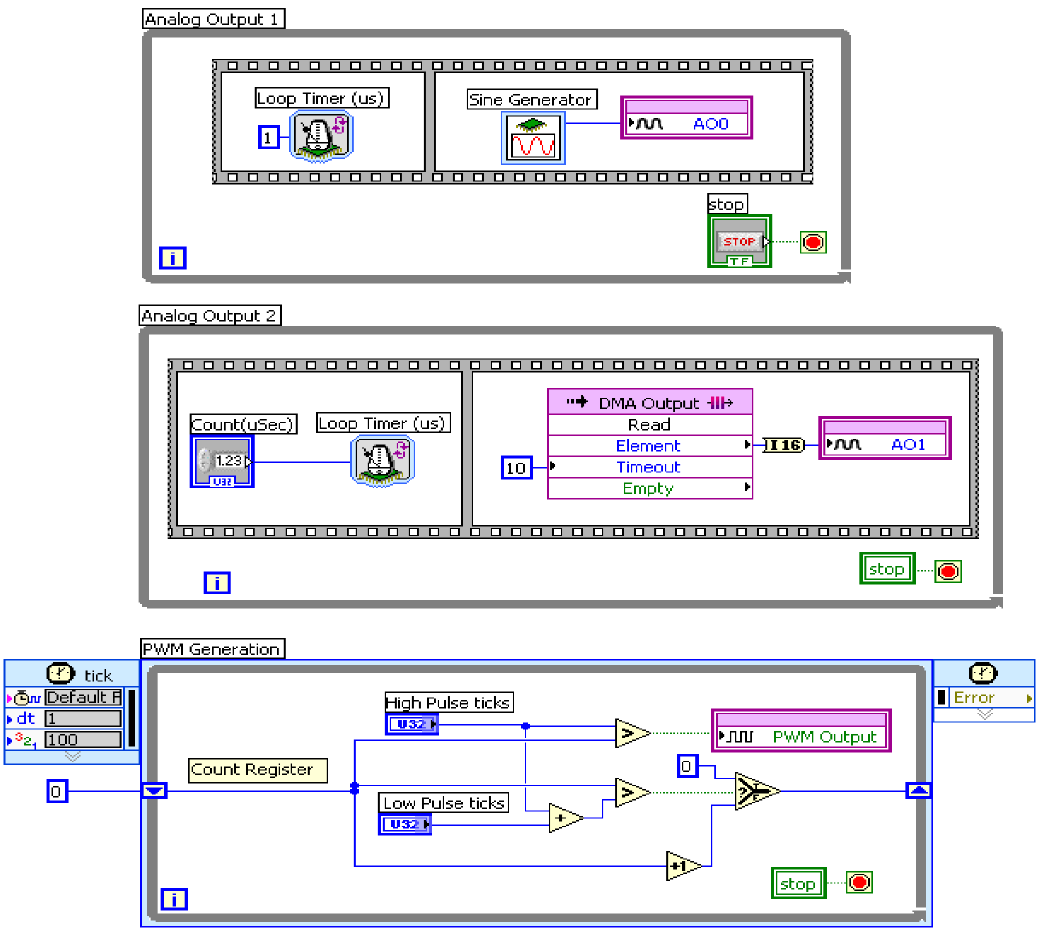
1.3. Challenges

There are some vendors in the marketplace who have products tackling part of this software/hardware abstraction issue. [1] FPGA vendors have tools for generating VHDL IP, but they are still based on current HDLs. Other vendors have created C-to-VHDL converters, which are certainly a much-needed step in the right direction. Many engineers typically have at least a working knowledge of C-style programming. Even though there are significant differences in these converters and a typical C language, the paradigm is familiar.

The most commonly used design ’style’ for synthesizable VHDL models is what can be called the ’dataflow’ style. A larger number of concurrent VHDL statements and small processes connected through signals are used to implement the desired functionality. Reading and understanding dataflow VHDL code is difficult since the concurrent statements and processes do not execute in the order they are written, but when any of their input signals change value. It is not uncommon that to extract the functionality of dataflow code, a block diagram has to be drawn to indentify the dataflow and dependencies between the statements. The readability of dataflow VHDL code can compared to an ordinary schematic where the wires connecting the various blocks have been removed, and the block inputs and outputs are just labeled with signal names. [2]

Utilizing graphical programming, LabVIEW FPGA for example is a function block, dataflow, graphical programming language which is compiled for FPGAs. FPGA implementations are often modeled as a state diagram or flow chart for visualization purposes. LabVIEW FPGA takes the next step by allowing the FPGA to be actually programmed using intuitive diagrams. This approach allows domain experts or test engineers to configure and visualize complex systems in hardware without any knowledge of VHDL or C, while retaining the power afforded by either. Layers of abstraction increase the number of people that can take advantage of FPGA technology for automated test application. With low-level I/O details abstracted from the user, system developers can focus on test algorithms and system-level concerns. This means that simulating a sensor with an FPGA does not have to include code for communicating with a digital-to-analog converter (DAC), or other code necessary to retrieve and assert pre-defined test vectors. With higher level languages, test engineers can spend more time on the actual test, adding coverage and reliability, with less time spent ironing out details of the sensor simulation implementation

Figure 1: FPGA block diagram with parallel loops



Source: authors

The block diagram in Figure 1 shows three loops running simultaneously in a single FPGA application. The top loop is generating a sine wave signal using a digital-to-analog converter (DAC) that updates every microsecond. The middle loop, however, is generating a user-defined waveform that is being streamed across the PCI bus and updated at a user-customizable rate. The third loop uses a timed loop structure to execute once every 25ns, and generate a pulse-width modulated signal. The duty cycle of the output signal can be varied by changing high pulse and low pulse parameters accordingly. While each loop is running at completely independent rates, they are all referencing the same register for a synchronized stop condition. This shows how multiple loops can access the same resource for global parameterization.

2. SENSOR EXAMPLES

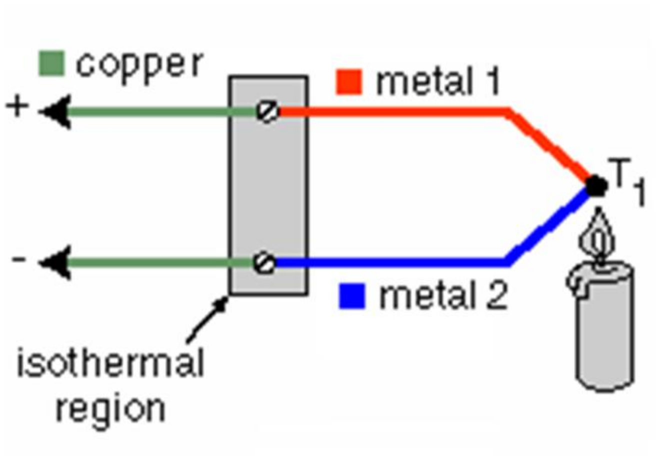
For the following specific implementation examples we will use LabVIEW FPGA as a higher-level programming language for sensor simulation with FPGAs. The hardware options for LabVIEW FPGA include integrated analog and digital I/O, as well as data communication circuitry and standard bus interfaces. This section will focus on the implementation of three different sensor types: thermocouples, LVDTs (Linear Variable Differential Transformers), and resolvers, while discussing others at a high level.

A LabVIEW FPGA program is divided into two applications, called VIs: the host VI and the FPGA VI. The host VI runs in either a host PC or a real-time system. Typically, the host stores test vectors, implements a user-interface, and performs preliminary floating point math before passing digital data to the FPGA VI. The FPGA VI receives the digital data representing the simulated signal, performs various processing steps, and outputs the correct voltage levels with tight control. The FPGA may need to generate or receive excitation signals which are often integrated in the final simulated sensor signal. In addition, the FPGA chip can be used to simulate noisy environments for realistic conditions.

2.1. Simulating Thermocouples

Before simulating any type of sensor, we must understand how the sensor works at its lowest level. Thermocouples, for example, use the Seebeck effect, which says that the junction of two dissimilar metals creates a small passive voltage proportional to temperature [3]. Figure 2 is a depiction of how thermocouples work. Because of the robust materials and absence of electronics, thermocouples are extremely durable in harsh environments and have the ability to measure extreme temperatures, all without excitation. From a hardware standpoint, simulating this sensor is particularly difficult because it requires very small voltages. Most thermocouple signals are in the range of -10 mV to 50 mV with a resolution of roughly 50 µV to 100 µV per degree. However, from the software side it is a matter of outputting a simple DC, albeit low, voltage.

Figure 2: How thermocouples work

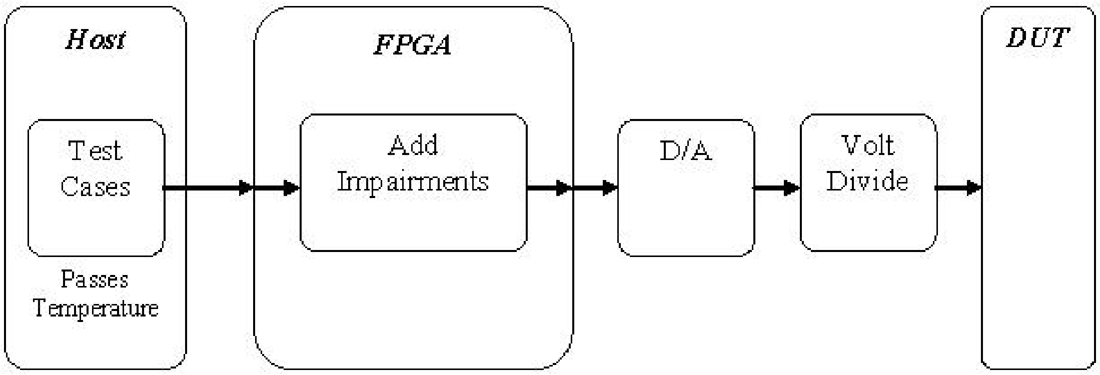


Source: authors

Keep in mind that this thermocouple sensor may be part of a system with many other types of needed sensors. It is very difficult to find one piece of hardware that can supply large voltage ranges needed for simulating sensors with excitation as well as small voltage ranges needed for passive sensors like thermocouples. Even a 16-bit output ranging ±10V can only achieve ~300 µV resolution, which is unacceptable for thermocouples that might change down to 10 µV per degree Celsius. In order to use the same hardware for both types of output voltages, it may be necessary to design an output attenuator circuit, particularly for simulating thermocouples. For ease of use, the attenuator could be as simple as a passive resistor divider network. By necessity, thermocouple inputs on a device under test (DUT) have very high impedances. Therefore, the network has little effect on the measurement quality. Nevertheless, because of the inherent errors associated with resistor networks in addition to the errors associated with any piece of output hardware, it is important to calibrate the output with a thermocouple measurement device, correlating output voltage to desired simulated temperatures.

Figure 3 is a block diagram of the different sensor simulation system components. The host VI should store the temperature profiles and test cases, convert the desired temperatures into a raw binary format according to the calibration tables, and pass this to the FPGA hardware.

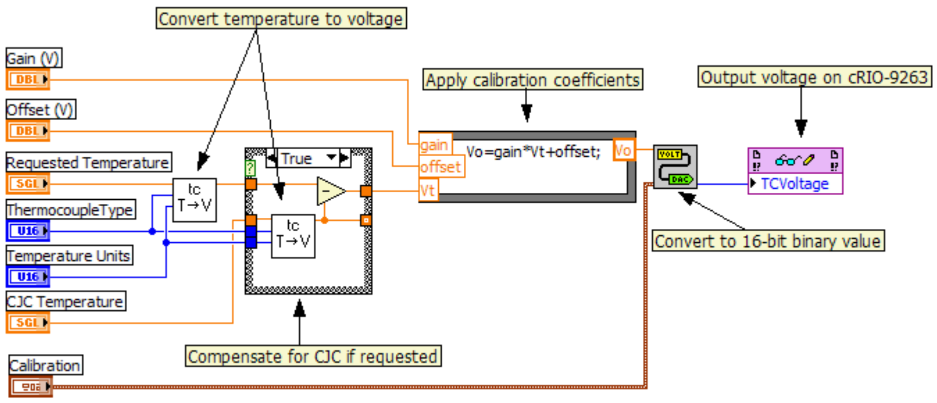
Figure 3: Thermocouple simulation system



Source: authors

The example implementation shown in Figure 4 receives a “Requested Temperature,” does the necessary compensation and calibration, and passes the final value to the FPGA through a variable called TC Voltage. In this case, the host VI allows for multiple thermocouple types and CJC compensation if needed.

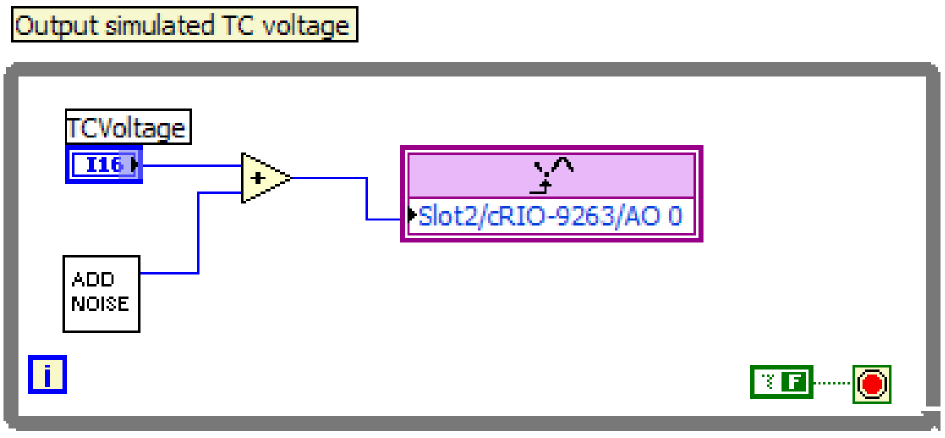
Figure 4: Host interface code for thermocouple simulation



Source: authors

Figure 5 shows a loop in LabVIEW FPGA that continuously polls the TCVoltage register and adds the expected noise impairments before writing that value to the digital-to-analog converter.

Figure 5: FPGA code for thermocouple simulation



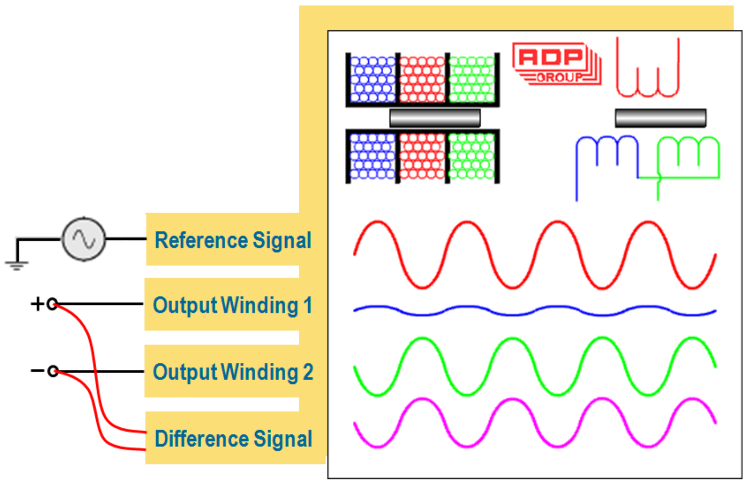
Source: authors

The major benefit to simulating a thermocouple signal is the ability to safely create harsh environments when testing fault conditions. If the unit under test was an engine control unit (ECU), for example, we can test safety circuitry and see how the controller responds without actually overheating an engine.

2.2. Simulating Linear Variable Differential Transformers (LVDTs)

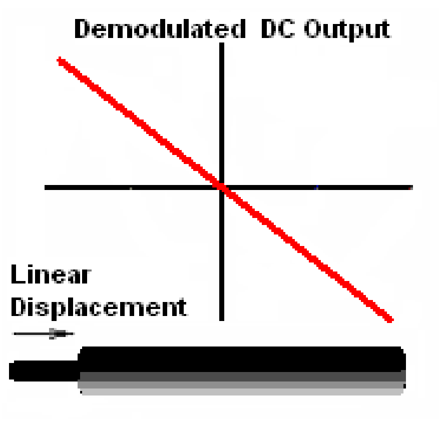
An LVDT is a sensor that incorporates a differential transformer with a sliding magnetic core. Driven by an AC (alternating current) excitation source, the LVDT generates a pair of AC output signals that are modulated according to the mechanical position (displacement) of the core as shown in Figure 6a. [4]

Figure 6a: How LVDTs work



Source: www.rdpe.com

Figure 6b: Graph of the demodulated amplitude signal as function of linear displacement changes

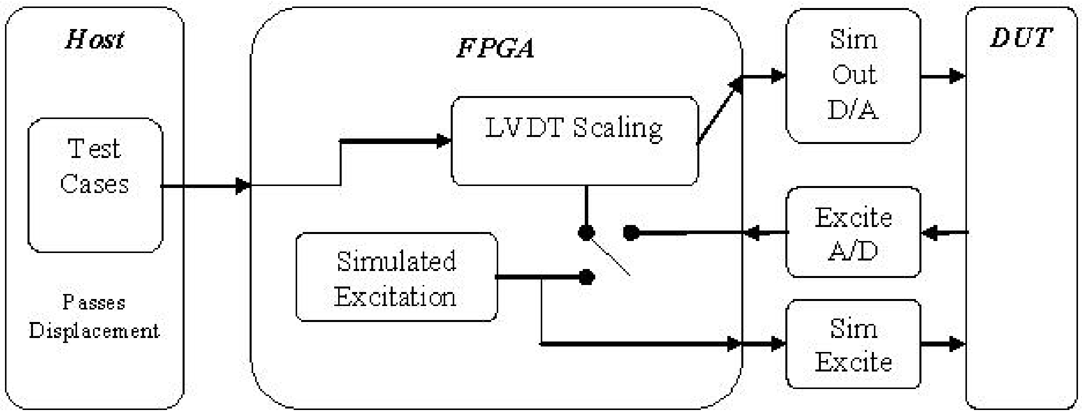


Source: authors

The output signals can be demodulated to recover the position information. The simulation of this sensor might simply be the demodulated linear signal shown in Figure 6b. However, this would only be the case if the sensor already had onboard signal conditioning. A more interesting simulation problem would be a raw LVDT where one's simulator takes in an excitation AC signal, and outputs a scaled wave according to the simulated linear displacement. We have chosen to tackle this more complex situation in the following implementation.

Figure 7 is a block diagram of the different sensor simulation system components.

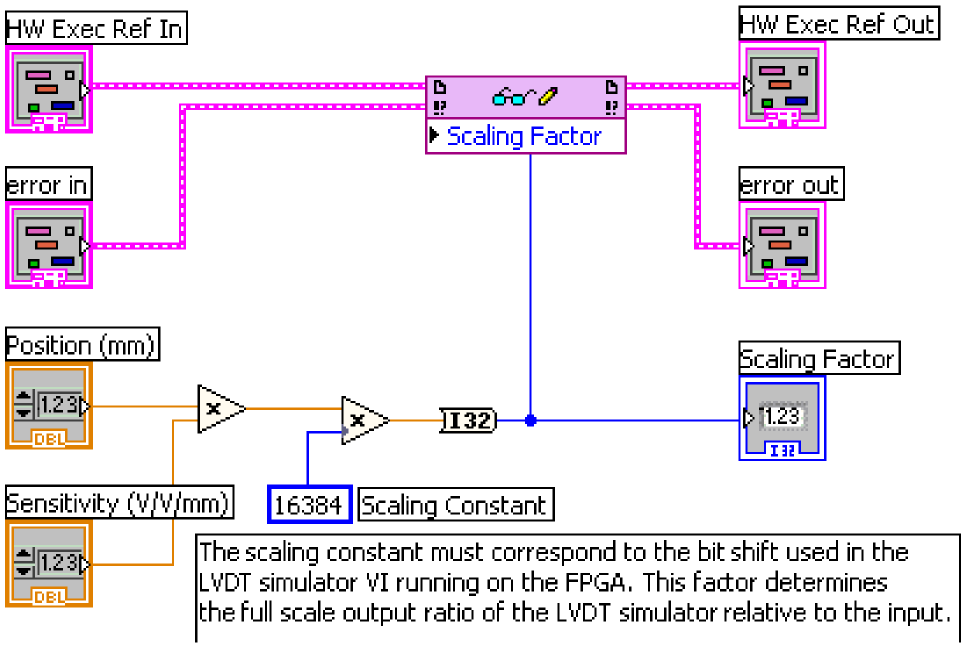
Figure 7: LVDT simulation system component



Source: authors

The ideal output of a LVDT without signal conditioning is a scaled version of the excitation signal. This scaling factor can be positive or negative and is proportional to distance from the mechanical middle of the device. The host computer passes the displacement in the form of a scaling factor to multiply with either the generated or real-world excitation signal. The host VI uses inputs of position and desired sensitivity, and initial scaling is done to get the results to binary form. This is passed to the FPGA through the "Scaling Factor" variable. Figure 8 is the graphical host interface code for LVDT simulation.

Figure 8: Host interface code for LVDT simulation

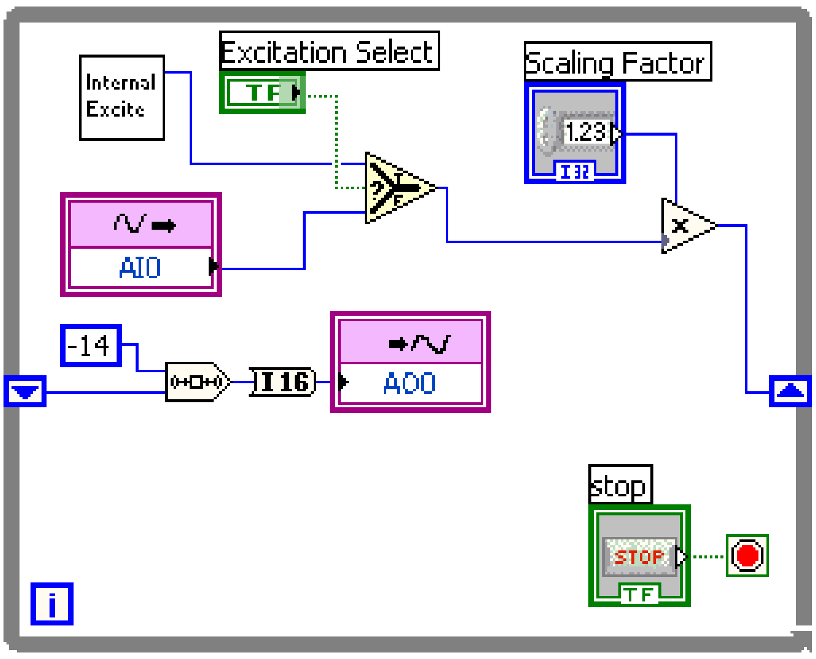


The scaling constant must correspond to the bit shift used in the LVDT simulator VI running on the FPGA. This factor determines the full scale output ratio of the LVDT simulator relative to the input.

Source: authors

On the FPGA the user can programmatically decide whether to use internal or external excitation and passes that to the multiplier. This applies the appropriate scaling to the signal based on the simulated displacement. The data is then passed to the next iteration of the loop to be re-factored to 16 bits and asserted to an analog output channel. The technique of passing the data to the next iteration is LabVIEW's method of pipelining for throughput. Notice that while current iteration is outputting a value, the FPGA is taking a new excitation voltage and implementing scaling in parallel. This new data is ready for the next iteration where it will be output. VHDL programming requires specialized coding to implement a pipeline, LabVIEW FPGA implements with an intuitive data tunnel on each side of the loop. Figure 9 is the graphical FPGA code for LVDT simulation.

Figure 9: FPGA code for LVDT simulation

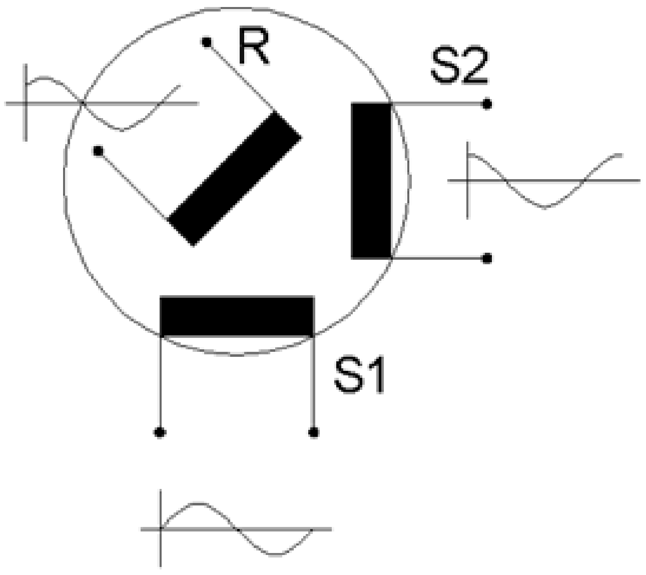


Source: authors

2.3. Simulating Resolvers

Being the analog counterpart to rotary encoders, resolvers measure the absolute rotary position of a rotating shaft. There are two fixed windings at right angles and a third spinning winding which is excited by some reference signal. [5] The reference signal (R) is induced onto the fixed windings with a magnitude representative of the third winding's angular position as it spins. Because the two fixed output windings (S1, S2) are at right angles to each other they produce a sine and cosine magnitudes (as shown in Figure 10) which have a unique combination at every given point in the rotation.

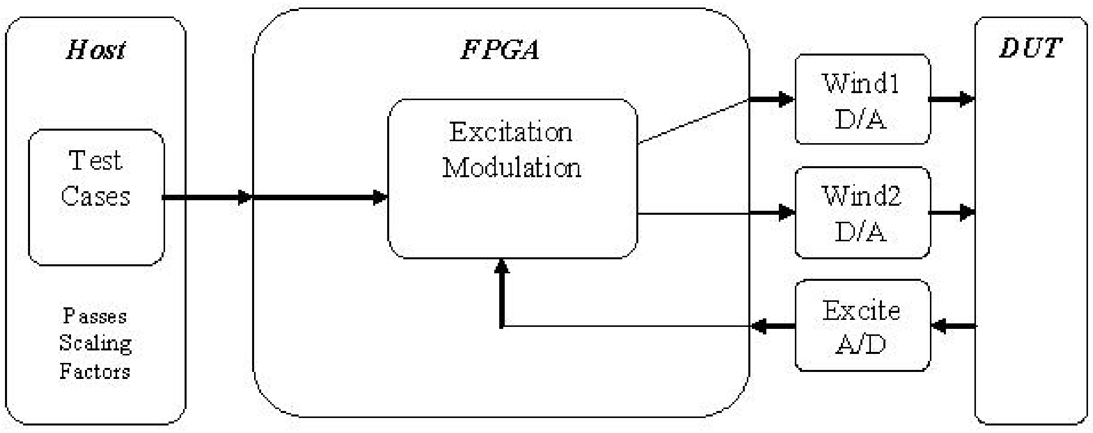
Figure 10: How resolvers work



Source: authors

The spinning winding is typically excited with a reference signal of 115 Vrms at 60 Hz or 400 Hz (for ground-referenced applications like manufacturing machines) or 26 Vrms at 400 Hz (for non-referenced applications like vehicles). Simulating these output signals on an FPGA requires two output modulated waves which correspond to the user-defined speed and position of a simulated shaft. Figure 11 is a block diagram of the different sensor simulation system components.

Figure 11: Resolver simulation system components

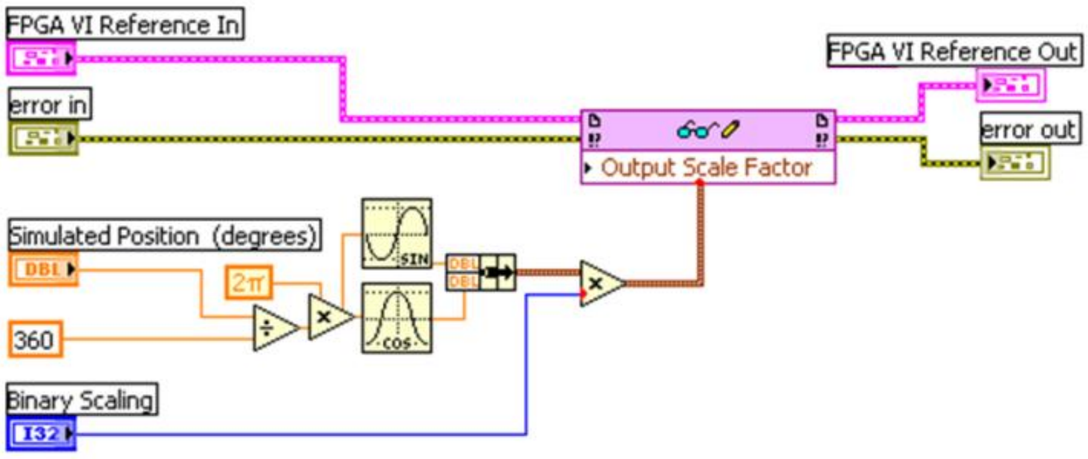


Source: authors

In a deployed system, the two winding signals are typically fed into a resolver-to-digital converter. This device may be purchased off-the-shelf or integrated into the DUT. Either way, there are specifications for input voltages from the raw resolver signals. Therefore, it is important to have a flexible platform which can be used to serve multiple voltage level considerations.

The host PC stores a library of motion profiles. These profiles should contain an array of desired set points simulating static angular displacements, typical motion paths, corner cases, and out-of-spec behavior. The host passes these test points to the FPGA where the output signals on both windings are calculated from the modulation of the test point with the excitation signal. The host VI converts the desired position in degrees to a binary scaling factor for each winding (sine and cosine). These factors are passed to the FPGA through the variable "Output Scale Factor." Figure 12 is the graphical host interface code for resolver simulation.

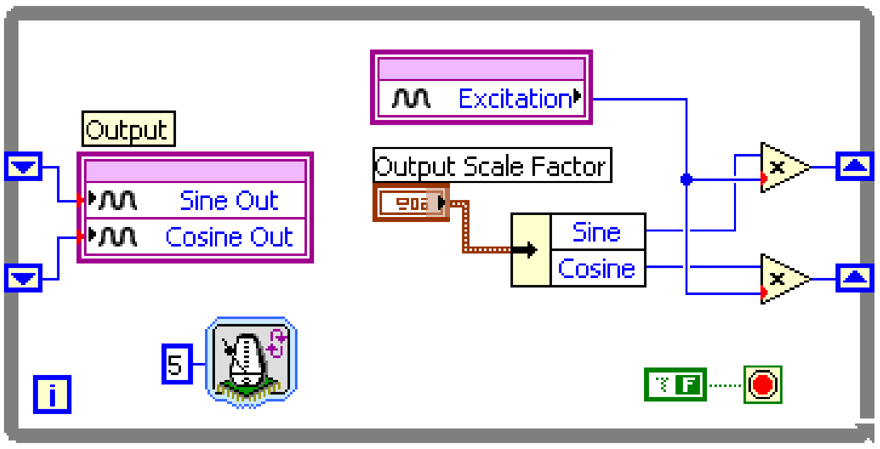
Figure 12: Host interface code for resolver simulation



Source: authors

The FPGA receives scaling factors from the host and multiplies them to the acquired external excitation. Like the LVDT implementation, the output is pipelined once for throughput. The FPGA could also be utilized to add noise or other impairments to the signal which might simulate the real deployed environment more closely. Figure 13 is the graphical FPGA code for resolver simulation.

Figure 13: FPGA code for resolver simulation



Source: authors

The three implemented systems are each analog sensor simulations which are made possible by an FPGA with the appropriate analog front-ends attached. Sensors which only deal in digital signals are even easier for FPGAs, because they require no analog support. Encoders, Hall Effect Sensors, Cam and Crank Sensors, or any sensors which transfers data with some type of digital protocol (serial or parallel) can be simulated on an FPGA.

3. VALIDATION OF SIMULATED SENSORS

Simulation is only as beneficial as it matches the real world properties. For simulated sensors this means accuracy in timing and value. The use of simulated sensors as a replacement of real ones makes it therefore necessary to provide ways to validate those properties.

There are basically three approaches to achieve this validation.

(1) Asserting an input vector to the actual algorithm being used and comparing the output vector with the expected result vector.

(2) Take approach (1) and include the real I/O interfaces with external stimulus and measurements.

(3) The actual use of the simulated sensor with real I/O together with the real DUT.

Approach (1) is the easiest one if the environment which is being used to define the FPGA implementation also provides a test framework to assert input vectors and read back the output vectors. Another benefit is the fact that no additional hardware is required.

Approach (2) accounts for the fact, that using real I/O might have a significant effect on the performance of the simulation. Algorithms being used typically rely on ideal I/O converter properties. It helps for example in finding limitations in timing accuracy as it shows latencies in the I/O process.

Approach (3) finally shows, if the simulated sensor is accepted by the DUT instead of the real one. Today's control units use a lot of very advanced diagnostic functions to determine the quality of the system status. If they detect a failure or inadequate performance of a sensor, they might stop working.

4. ECONOMIC BENEFITS

With the increasing movement to commercial-off-the-shelf (COTS) hardware, FPGAs have become even more popular for sensor simulation. It is important to consider connectivity to I/O pins for sending data to and from the chip itself, when using FPGAs for sensor simulation. Commercially available FPGA hardware with integrated I/O, PC-bus interfaces and signal conditioning has dramatically reduced development times and alleviated many of the hidden costs associated with custom hardware design. COTS hardware offers PC-buses like PCI and PXIe for a host application to interface with the FPGA application. Direct-Memory-Access (DMA) channels are also included for streaming data across these high-speed buses, with rates up to several 100MB/sec. In order to interact with the outside world, COTS FPGA hardware typically integrates I/O components like analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and digital line drivers. Combining off the shelf hardware with higher level programming software will also abstract out the communication logic needed to interface with external components, and replace it with graphical I/O nodes and DMA transfer FIFOs as shown in the earlier implementation examples. Shortening a product’s time-to-market is a major factor when simulating sensors, and using commercially available FPGA hardware ensures that prototyping and test system development time will not be the bottleneck.

5. CONCLUSION

Sensor simulation allows test engineers to incorporate real-world signals into automated test systems to simulate a broad range of operating environments. Once all functionality has been verified, using the simulated environment the critical hardware under test can then be connected to the actual system plant for final deployment. The flexible nature of FPGAs with true parallel operation, make them ideal for simultaneously simulating multiple types of sensors. Using a higher-level programming language allows experts across any industry to take advantage of FPGA technology, and COTS FPGA hardware enables high-performance prototypes and test systems to be developed quickly and easily, without prior experience in FPGA hardware design.

REFERENCES

[1] Tom Williams: FPGA Tools Target Higher Levels of Abstraction, RTC Magazine September, 2004.
http://www.rtcmagazine.com/home/article.php?id=100125&pg=1

[2] Jiri Gaisler: “Fault-tolerant Microprocessors for Space Applications”, 5: page 41, Gaisler Research AB, Sweden
http://www.gaisler.com/doc/vhdl2proc.pdf

[3] Thermocouple Simulation with CompactRIO, National Instruments White Paper.
http://zone.ni.com/devzone/cda/tut/p/id/4310

[4] Simulating an AC LVDT with the NI PXI-7831R Reconfigurable I/O Device, National Instruments White Paper.
http://zone.ni.com/devzone/cda/tut/p/id/4101

[5] Resolvers, National Instruments White Paper.
http://zone.ni.com/devzone/cda/tut/p/id/2888

ACKNOWLEDGMENT

The authors acknowledge the contribution of their colleagues to this work.