Graphical System Design for Embedded Control Systems

Mike Trimborn, LabVIEW FPGA Product Manager, National Instruments

Mike Trimborn (mike.trimborn@ni.com) is a LabVIEW FPGA Product Manager at National Instruments. He earned a BS in mechanical engineering from the University of Texas at Austin. While at National Instruments, Mike has also served as Product Manager for LabVIEW Real-Time and as an Applications Engineer for working on specific implementations using LabVIEW to design, prototype, and deploy embedded

Embedded systems are becoming more common as engineers throughout many industries are placing more and more intelligence onto devices. You can find embedded processors in objects such as shoes, phones, toasters, and automobiles and being able to develop high-quality and optimized designs are becoming much more difficult.

With so many engineers needing embedded technologies and so few having embedded expertise, the search is on for new tools that can bring embedded technology to an order of magnitude more engineers. Think of it as finding the next "American Idol." You have a few classes of tools that are trying to win the hearts and minds of engineers across this land:

- Traditional text-based embedded tools that are attempting to simplify the design process by providing reference designs
- IP vendors that are selling tested and packaged IP blocks for use in traditional system design tools
- Graphical system-level design tools that are providing higher levels of abstraction to design embedded systems

Engineers such as machine builders and automotive experts understand the mathematics or processes required for their systems, but they lack knowledge of low-level embedded development tools and semantics. Helping these domain experts over this hurdle requires the last approach mentioned above, graphical system design.

Graphical system design is a revolutionary approach to embedded design that blends intuitive graphical programming and flexible commercial off-the-shelf (COTS) hardware to help engineers and scientists more efficiently design, prototype, and deploy embedded systems. With the graphical system design approach, you can use a single environment for all design stages to increase productivity, save money, and bring embedded technology to the domain expert.

Graphical Programming for Designing Embedded Systems

Many embedded systems run autonomously and must execute multiple tasks in parallel with specific timing requirements. Consider a machine control system that needs to control a linear stage, rotate multiple shafts, control lighting, and read in video data. In a system such as this, there are multiple processes that must happen deterministically, in real time, and in parallel. When using traditional text-based programming languages, complexity expands exponentially as you add in the necessary components of an embedded system.

Let's consider a comparison of a graphical (LabVIEW) versus a text-based (C) approach to creating an embedded application. When creating a simple single-task application, it is relatively simple to write an application in C or in graphical code making it a choice of personal preference at this level. However, as soon as we start adding complexity, the productivity benefits of a graphical abstraction begin to emerge. Figure 1 shows two parallel loops that are acquiring data and sending that data over a network. Even with two simple processes running concurrently, the graphical approach automatically abstracts the system complexity.

The loops shown continue to add sophistication by incorporating hardware timing with a built-in Timed Loop structure – a semantic that natively represents time and concurrency. This structure allows you point and click to set OS priorities, delays, loop rates, etc. To continue on this path with C would far exceed the bounds of what could be shown in single graphic. At this point, the text code is obviously a barrier to a broader audience where the graphical representation is simply much more clear and accessible to scientists and engineers.

Thinking of future systems that include multiple processors, you could imagine developing this application and 'slicing' the processes on different processors. You can see that LabVIEW is able to manage parallel

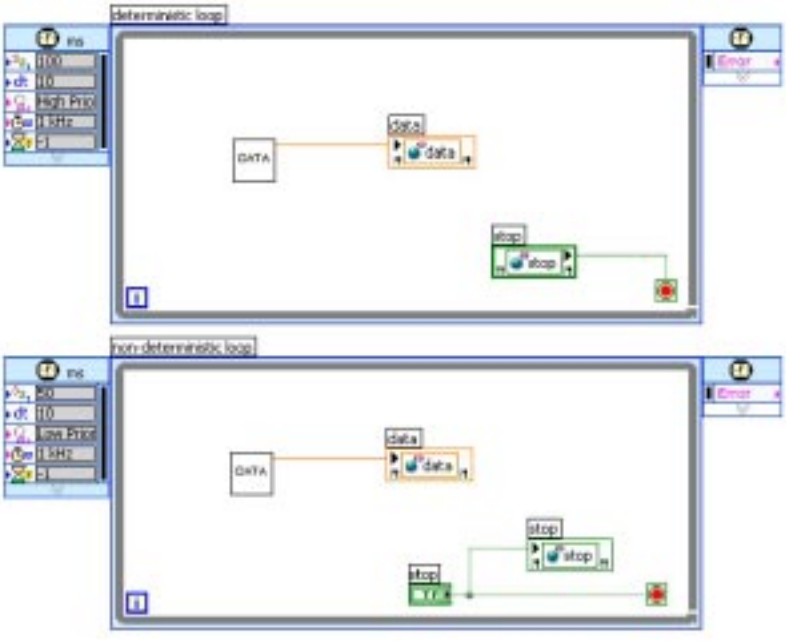

Figure 1. Parallel timed loops in NI LabVIEW intuitively display parallel tasks.

processes and stay consistent and scalable.

There are some inherent things about graphical programming, like complexity abstraction that offer a preferable solution over text-based approaches. This graphical representation is clear and accessible to a broad range of domain experts, enabling them to design systems with complex timing and parallelism. In contrast, many of these domain experts lack the expertise to implement this complexity using traditional text-based approaches.

Another key requirement for embedded system design is that the software platform should address the various algorithm design views common in real-time embedded design. These design views - sometimes referred to "as models of computation" - include:

- Text-based math
- Continuous time simulation
- State diagrams
- Graphical dataflow models

These varied views are needed because different elements of an embedded system are best captured using different representations. For example, state diagrams are the most common way to represent control functions, while text-based math is a better way to represent complex algorithms. If a tool supports all of these views, it helps minimize the complexity of translating system requirements into a software design. Over the last several years, graphical programming has evolved to incorporate all of the models of computation mentioned above to better meet the needs of embedded system designers and their varied skill sets.

Innovating Software Algorithms

Signal processing and control algorithms deployed onto distributed devices add significant value to systems and end products. Therefore the ability and freedom to design and iterate on algorithms is critical to innovation and optimizing system performance. Domain experts are the engineers and scientists that hold the expertise to develop these algorithms and processes. While they understand the mathematics and semantics of the process, they often don't understand how to deploy these processes on the wide variety of embedded computing devices ranging from floating point architectures such as PowerPCs, x86, ARM, etc…, to fixed-point MPU, DSP, or FPGA architectures.

Implementing these processes usually requires the expertise of embedded computing (either through learning or by hiring another developer.) This embedded developer who understands embedded computing platforms very well; however, he or she may not have complete understanding of the algorithm. This presents a large communication gap and can not only add large inefficiencies to the design process; it can cause serious flaws in the end system or product.

Graphical programming languages deliver on these communication challenges for certain vertical areas such as motion control or digital filter design. Considering a design task such as designing digital filters, a domain expert can interactively translate a high-level description of the filter parameters, such as those for the low-pass filter response shown in Figure 2 and automatically convert it to fixed point using LabVIEW. Once converted to fixed point, the tool can run simulations to compare fixed and floating point implementations. The filter designer can iterate on his or her design until he or she is satisfied with the simulation results, at which time he or she will hand the code over to the embedded engineer for implementation.

Conversely, if the embedded developer needs to make changes to the algorithm, he or she can go back into the design, make the needed changes and regenerate new LabVIEW or C code. Using one platform to make this translation is efficient since both experts are working with the same tool all the way from design to implementation.

COTS to the Rescue: Saving Time and Money by Buying Off the Shelf

With so many designs running late, never releasing to market, or failing after release, something needs to be done to get more high-

Figure 2. Designing a digital filter using a configuration window.

quality products out more quickly. One way to address both of these issues is to prototype the systems better by integrating real-world signals and real hardware into the design process earlier. This way, high-quality designs can be iterated on and problems would be found (and fixed) earlier.

Today, if you are creating custom hardware for final deployment, it is difficult to develop the software and hardware in parallel because the software is never tested on representative hardware until the process reaches the system integration step. This is a problem because waiting until the system integration to test the design with real I/O and real signals may mean that you discover problems too late to meet design deadlines.

Most designers use evaluation boards to prototype their systems. However, these boards often only include a few analog and digital I/O channels and rarely feature vision, motion, or the ability to synchronize I/O. Additionally, designers often have to waste time developing custom boards for sensors or specialized I/O just to complete a proof-of–concept design.

Using flexible COTS prototyping platforms can streamline this process and eliminate much of the work required for hardware verification and board design. Graphical system design strives to achieve a level of prototyping standardization similar to PCs, for which anyone can go to an electronics store and combine components such as memory, motherboards, and peripherals to create a PC. With graphical system design, engineers can take the specific components they need for their systems and use software to easily integrate and program the overall system.

For most systems, a prototyping platform must incorporate the same components of the final deployed system. These components include a real-time processor for executing algorithms deterministically, programmable digital logic for high-speed processing or interfacing the real-time processor to other components, and a variety of I/O types and peripherals (see Figure 3). Finally, as with any system, if the off-the-shelf I/O does not serve all of your needs, the platform should also be extensible and customizable when needed.

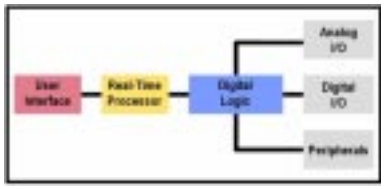

Figure 3. Typical Components of an Embedded System

National Instruments offers several types of prototyping platforms, including NI CompactRIO, which contains all of the basic building blocks of an embedded system. The controller features a 32-bit processor running a real-time operating system. The CompactRIO backplane includes an FPGA that can implement high-speed processing and also configures and provides interfaces to the I/O modules. These I/O modules feature options for analog input and output, digital input and output, and counter/timer functionality. Each of the modules includes direct connectivity to sensors and actuators as well as built-in signal conditioning and isolation. A module development kit is also available so developers can expand the platform to include custom modules – all plugging into this COTS framework.

Additionally, CompactRIO is industrially packaged (-40 to 70 °C, 50 g shock) with a small footprint (3.5 by 3.5 by 7.1 in.) and low power requirements (7 to 10 W typical), making it ideal for not only prototyping but also the deployment of in-vehicle, machine control, and onboard predictive maintenance applications.

Daewoo Electronics recently used CompactRIO and LabVIEW to decrease its time to prototyping for next-generation holographic digital data storage (HDDS) device by nearly 10 times to a month. The alternative for Daewoo was a DSP board, however, while this was available off the shelf, if did not have to breadth of packaged I/O or the ease of software design that Daewoo required. Using CompactRIO, the company implemented a completely functional electro-optical motion control system that controlled three independent motors. This motion-control system interfaced to an external 8M gate Xilinx FPGA that conducted video decoding.

Final System Deployment

Once prototyping is complete, graphical system design platforms must allow deployment of your code to a final, custom device. Lab-VIEW provides the capabilities to target code to any 32-bit processor. Using the same platform to target custom devices provides a much easier and efficient transition from prototype to deployment.

A graphical approach can provide many key advantages including a consistent interface for I/O components, inherent parallelism, and opportunities to optimize underlying C code. I/O nodes such as those shown in Figure 4 provide simple interfaces to basic I/O components for analog and digital I/O. Additionally, these I/O nodes provide interfaces to specific device drivers to read and write to peripherals such as serial or Ethernet interfaces. Finally engineers need the ability to tweak generated C code to optimize and tune the embedded performance of the device. Graphical paradigms can accomplish this by allowing C code to be written within the graphical context.

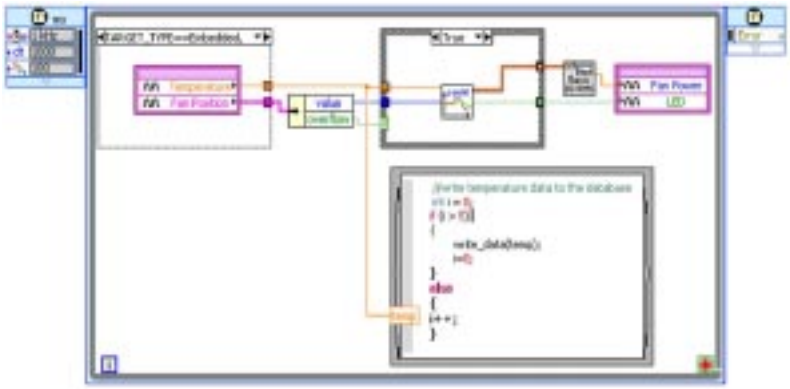

Figure 4. A control loop implemented in the LabVIEW Embedded Development Module. The I/O nodes are the purple nodes and interface the LabVIEW code to sensors and actuators.

The abstractions found in a higher-level graphical language improve the overall efficiency of the design process by assuming the burden of proper execution. This is insured by porting the graphical paradigm to a third-party C cross compiler to properly link, compile, and download the code to the custom processor.

This new technology empowers a broader range of engineers, scientists, and domain experts to design algorithms, develop applications, program logic, prototype system and deploy those systems to their targets of choice.

Conclusion

Using a single graphical programming platform for algorithm design to prototyping to deployment can be a very efficient way to implement new and innovative functionality to your devices. Whether designing a digital filter, an ECU or a machine control system, the idea behind graphical system design is porting highly optimized designs to flexible COTS hardware and then to custom hardware as needed. Embedding intelligence and signal process is becoming essential to differentiate hardware products, especially in embedded and distributed applications. Thus, it is no surprise that software is playing a major role for these systems either. Using a single platform reduces overall startup time, especially on subsequent designs. It allows the flexibility of reprogrammable processors and custom I/O modules.