

basics of Design

Unified Graphical System Design Environments Boost Design Success Rates

Bill Murray
Contributing Editor

Why do so many embedded system designs fail to meet their goals?

A survey published in 2003 by market research company Embedded Market Forecasters reports that over 6000 design projects out of 45,000 surveyed were cancelled—a 13% failure rate.¹ Of those design starts that were completed, 54% were late, with an average delay of four months versus plan. Even then, many of these “successful” designs did not meet performance and functionality expectations. Over 70% of designs missed performance targets by more than 30%, and 30% of designs missed functionality specifications by at least 50%.

The direct cost of failure and the opportunity cost of unmet expectations must surely have a significant impact on companies’ bottom lines, albeit one that may have been accepted because it is “normal.”

The survey also gives the reasons for failure. Of the 947 respondents, about two-thirds identified “limited visibility into the complete system” as a major cause of failure; just over half reported “limited ability to trace” as another cause; and well over a third reported “limited ability to control execution.”

In other words, the design methodologies employed suffered seriously from inadequate controllability and observability. So, enhancing the controllability and observability of a design methodology can directly affect the bottom line. According to its proponents, that is what a unified graphical system design environment can do.

Consider three stages in embedded system design: algorithm modeling and validation, algorithm prototyping, and system implementation. Each task is generally undertaken by different members of the design team, each using its own task-specific development tool flow. This task partitioning makes sense from a skill-set point of view, but it also fragments the designers’ visibility.

Consequently, integration of these individual tasks and tool flows into a unified graphical system development environment can give the designer a “whole system” view. Combined with graphical design and debug, this view mitigates the problem of “limited visibility into the complete system.” Moreover, interfaces to external standard development boards and measurement equipment can apply the “real-world” stimuli necessary to perform realistic simulations—and expose those rare, corner-case conditions and behaviors that a designer might find difficult to anticipate.

Such a design environment enables virtual system integration and simulation very early in the flow, so designers can rapidly evaluate alternative implementations and to detect and debug problems before committing to a final, custom implementation. It also significantly eases the development of design upgrades and derivative designs.

In addition, a unified design environment can be an effective design sharing mechanism. The design, from an algorithm to an entire system design, can be transferred complete with the tools that verified it and the conditions under which it was verified. So, how does a unified graphical system design environment work?

Algorithm Development

Take algorithm development and validation. This may be a completely new algorithm development; or the reuse (or modification and reuse) of an existing algorithm; or the reuse of an industry-standard algorithm from an open source or a commercial provider; or it may be a combination of all of the above. These algorithms are generally modeled in a variety of graphical, symbolic, or textual modeling environments such as C language, LabVIEW, Matlab, MathCAD, Scilab, and Xmath.

The unified graphical system design environment is a mathematical modeling tool that supports the mix-and-match of the afore-

mentioned tools and model types. This enables designers to use their established flows and simultaneously leverage algorithm models from multiple, disparate sources.

Working with these tools, the environment enables the designer to simulate the algorithm with real-world data acquired from online sensors. It also supports inline and online signal generation, modification, execution, and analysis. Moreover, it provides a graphical analysis capability—a useful enhancement to the analysis capabilities of textual third-party tools.

The application of realistic stimuli improves the controllability of the design, while the enhanced analysis capability improves the observability of the design. This integration of math with real-world I/O and instrument control is particularly useful when reusing an algorithm with which the designer is not familiar, because it provides greater visibility of behavior under realistic conditions than could a “theoretical” simulation.

Finally, the environment can provide two productivity boosts. It automatically creates an interactive GUI concurrently with algorithm design, eliminating the time, effort, and errors involved in a separate GUI development. Also, it deploys a graphical and textual library of hundreds of functions of the kind that are used repeatedly in development, such as signal synthesis, spectral analysis, optimization, evaluation, and monitoring.

Algorithm Prototyping

There are two objectives here. One is to devise the hardware/software partitioning that delivers the requisite performance and power consumption within manufacturing cost constraints. With an eye to the future, the partitioning would also comprehend subsequent reusability. The other objective is to validate the algorithm implementation, both hardware and software, as quickly and as early as possible to detect and eliminate bugs before integration into the production design.

System design flow

Algorithm development

- Design algorithms with a combination of graphical and textual tools
- Integrate mathematical models from different applications, such as LabVIEW, MathCAD, Matlab, Scilab, and Xmath
- Verify with real-world data

Algorithm prototyping

- Prototype software on standard hardware platforms
- Prototype hardware in FPGA
- Validate with real-world data

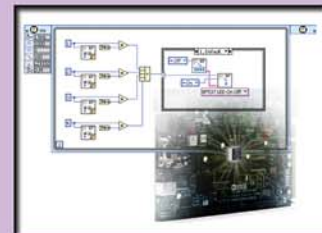


Mathematical modeling and analysis



- Supports mix-and-match of multiple tools and model types, so requires no change to the developer's algorithm design flow
- Enhances simulation with real-world data from online sensors, avoiding surprises at implementation
- Enables inline and online signal generation, modification, execution, and analysis
- Leverages library of hundreds of graphical and textual functions for synthesis, analysis, optimization, evaluation, and monitoring to boost productivity and speed time-to-market

Software and hardware implementation



- Enables rapid HW/SW partitioning and analysis and rapid repartitioning for optimum design
- Supports standard software development platforms and eliminates the time-to-market risk, cost, and effort of custom design
- Implements VHDL in FPGA
- Uses real-world stimuli modules that connect directly to sensors and actuators
- IP library of more than 140 hard-optimized signal-processing functions boosts productivity and speeds time-to-market

Unified graphical system design environment

To maximize design flexibility and minimize time-to-market, the design team would prefer to implement in software whatever it can and implement in custom hardware whatever it must. To minimize the hardware design effort, the unified graphical system design environment supports the use of off-the-shelf platforms such as PXI, PCs, external software-programmable devices, standard processor-specific software development platforms, and FPGAs. This also lets the design team repartition the functionality relatively quickly to find the optimum HW/SW partition. And, as before, the design is validated using real-world stimuli.

There are actually two software development approaches here. The software developer can

use the standard development platform, its associated simulator, and established software development tools to develop the C/C++ code. The code can then execute on the development platform while being exercised with real-world stimuli controlled by the graphical system design environment. In this scenario, the tool chain and operating system (OS) must first be integrated into the environment.

Yet the environment also offers graphical programming capability, with a library of more than 400 signal-processing and analysis functions together with native programming structures. This is obviously useful for design teams that lack C/C++ software development expertise. But it can also be used by expert teams that

wish simply to get the software up and running quickly to validate it, and then possibly optimize it manually after validation. Consequently, the environment also supports a mixture of the two approaches, as manually developed C/C++ can be dropped into the graphically generated code.

System Implementation

This is where the design team moves to final implementation. The objective is to design a production system, integrating the prototyped algorithms into a standard prototype system for small production volumes or a custom printed-circuit board (PCB) design for larger volumes.

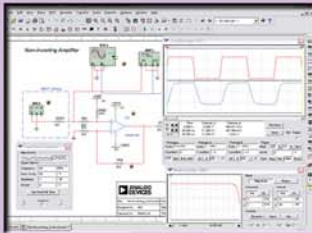
The graphic shows an external software-programmable device that doubles as a develop-

System implementation

- Deploy embedded system
- Select components; design, simulate, lay out board
- Verify with real-world data



PCB design, layout, and verification



- Speeds part research and reduces the need for evaluation boards with interactive tools that simulate real operating conditions
- Improves simulation accuracy by driving models with real-world signals
- Uses interactive tools to compare simulation results with physical prototype attributes to validate and debug design



- Industrial control and acquisition system supports system design and low-volume production
- Performs analysis, post-processing, data logging, or communication to a networked host computer
- Provides direct access to the I/O circuitry of each I/O module
- Transfers data in real time to and from the system design environment
- I/O module types include -80-mV thermocouple inputs, -10-V simultaneous sampling analog I/O, 24-V industrial digital I/O, and $250\text{-V}_{\text{RMS}}$ universal digital inputs

- Supports whole system design
- Validates design with real-world data
- Interfaces to standard development boards and measurement equipment for validation and debug
- Mathematical modeling environment integrates models from third-party tools
- Offers graphical software programming capability
- Works with standard PCB design and layout tools and Spice simulator to verify board design
- Advanced graphical debug capability

Environment

ment platform and a low-volume production vehicle. Incorporating a real-time processor, reconfigurable FPGA, and a variety of hot-swappable I/O modules with direct connection to sensors and actuators, the platform can be used to implement a wide range of control and acquisition applications.

PCB design for price-pressured higher-volume production encompasses standard component selection, circuit design, simulation, and board layout. As before, the whole ensemble must be verified with real-world data. The graphical system design environment therefore works with popular PCB design tools and Spice simulators.

Standard component selection using component models, simulated in a virtual design, is

faster and simpler than evaluating the chip itself using test and measurement equipment. Mathematical models of, for instance, an operational amplifier can be used in a graphical system design environment to evaluate primary parameters such as gain, input offset voltage, and bandwidth under realistic operating conditions. Component providers are moving to Spice models to model and analyze transient responses and noise effects.

The system is designed using the design team's established tool chain within the graphical system design environment. Both the design and the PCB layout are simulated with Spice and real-world stimuli. Interactive tools can be used to compare simulation

results with physical prototype attributes to debug the design.

Bottom Line

The graphical system design environment enables engineers to see what they're doing and get it right. And the results go straight to the bottom line.

Reference

"Embedded Software Development Issues and Challenges: Failure is NOT an Option—It Comes Bundled With The Software," by Jerry Krasner, Embedded Market Forecaster; www.embeddedforecast.com/emf_esdi&c.pdf

ED ONLINE 15429