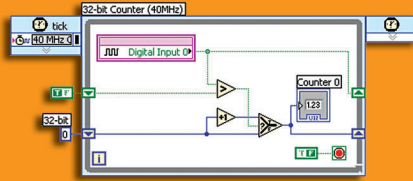


# Abstraction Simplifies Heterogeneous System Design

## Software Components

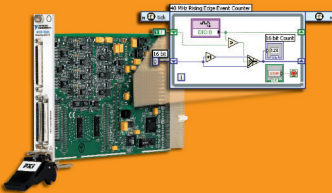
Sophisticated systems are always built using some type of reusable software components, whether they're function source code, libraries, or graphically designed models. Model-based components have the advantage of providing run-time, design-time, and possibly debug interfaces so developers aren't required to build this additional scaffolding. This reduces the level of programming expertise required to use the component.



## Building Blocks

System components are a mix of hardware and software. The base is always hardware, and software sits on top of the platform.

Building blocks that are more sophisticated require more expertise in their construction but typically less expertise in their use. While high-level components can generate large amounts of code, this code isn't necessarily inefficient. In fact, standard or frequently used components often are optimized, so it behooves designers to reuse components instead of reinventing them.



### Model-Based Modules

- Interfaces: design-time, run-time, and debug
- Characteristics: property based, graphical interconnects
- Debugging: modules incorporate debug interface
- Examples: LabVIEW virtual instruments, UML classes

### Modules And Frameworks

- Interfaces: design-time and run-time
- Characteristics: require the use of a specific infrastructure
- Debugging: add module- or framework-specific tools
- Examples: Java and .NET frameworks

### Runtime Libraries

- Interfaces: run-time
- Characteristics: class/method or module/function oriented
- Debugging: some run-time tools
- Examples: C++ Standard Template Library (STL)

### Operating-System Board Support Packages

- Interfaces: function header definitions
- Characteristics: board-specific
- Debugging: low-level, device-specific
- Examples: vendor-specific board support packages

### Hardware

- Examples:
- Chips: custom ASICs, FPGAs, microcontrollers
  - Modules: COM Express, PMC/XMC/AMC
  - Boards: PXI, PC/104 Plus, AdvancedTCA

More target platforms

More design flexibility

## System Design

### Homogeneous Systems

A symmetrical multiprocessing (SMP) system is an example of a homogeneous platform that can be complex. But having the same cores and memory architecture greatly simplifies application design.

### Heterogeneous Systems

Unfortunately, getting the best performance, economy, or other metric from a system often means choosing hardware or software modules that are quite different, such as mixing a DSP and a general-purpose processor to handle a multimedia application. Higher levels of abstraction such as those used in model-based designs can simplify this job of creating the application by hiding the underlying hardware and software differences so the environment appears to be a homogeneous system.

### Model-Based System Design

- Heterogeneous designs: complexity hidden by modules, modules include user interface for development and debug
- Single graphical environment for multiple targets
- Application-specific modules and toolkits

### Text-Based Application Design

- Heterogeneous designs: complexity hidden by modules and libraries but text configuration is required
- Underlying textual environments for various targets
- Configuration tools for hardware interconnects

### Standalone Text-Based Application Design

- Heterogeneous designs: communication between applications is platform-specific (hardware, OS, drivers, etc.)
- Required expertise in target integration
- Embedded programming

### Device Drivers And Operating Systems

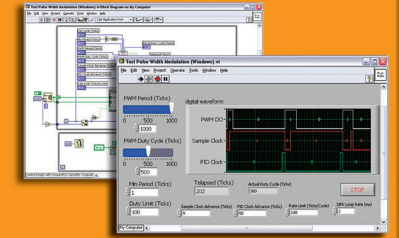
- Heterogeneous designs: communication between devices is implementation-specific
- Required expertise in register-level programming
- Kernel development and assembly programming

### Hardware Design

- Heterogeneous designs: must contend with hardware and protocol-level communication
- Required expertise in board-level analog and/or digital design
- Electrical specifications and digital protocols

## Graphical Model-Based Design

This approach hides underlying complexity, so designers can deal with a heterogeneous system as if it were a homogeneous system. Components typically have attributes that can address design-time, run-time, and debug aspects of the design cycle. These attributes can address the underlying system while minimizing its impact on the high-level design.



### Model-Based Graphical IDE

- Expertise: graphical programming, application-specific
- Pros: hides underlying system architecture and complexity
- Cons: building models for targets is a complex task
- Examples: LabVIEW

### Graphical Development Tools

- Expertise: programming language, command line tools
- Pros: more flexible and powerful development environment
- Cons: less control over system-level details
- Examples: Eclipse, Visual Studio

### Text-Based Development Tools

- Expertise: programming language, command line tools
- Pros: access to low-level timing and interface
- Cons: developer must deal with system-level details
- Examples: Eclipse, Visual Studio

### System Debugger

- Expertise: hardware and operating-system background
- Pros: access to devices and operating system
- Cons: requires experience with drivers
- Examples: GDB, command line debuggers

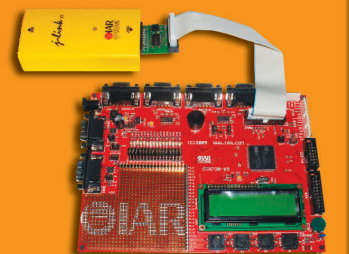
### Hardware Debuggers

- Expertise: hardware and operating-system background
- Pros: access to low-level timing and interface
- Cons: require experience with hardware
- Examples: JTAG, BDM

## Development Tools

Designers swear by and at their tools. They're generally designed to simplify or automate development tasks so designers can get more work done. The tools also can hide underlying complexity, so developers with more application expertise may not require as much programming expertise to get the job done.

Using a single tool for heterogeneous system design and development reduces user expertise requirements with respect to tools, but a particular tool often may limit the number of targets. The alternative is to use tools for different parts of a design and hope they can be integrated in some fashion.



Faster design cycle

Application expertise