Rapid Design, Verification, and Optimization of ARM Technology-Based Embedded Systems

**Introduction**

The challenges of concurrently designing hardware and software in embedded systems, particularly for today’s system-on-chip (SoC) designs, are well documented in the electronics trade literature and in internal system design documents. Figure 1 shows the results of a 2003 survey by Collett International in which respondents indicated that 67 percent of first-time silicon failures are due to logical or functional design errors. Also worth noting in the graph is that incorrect firmware, indicating hardware/software integration problems, is the cause of 13 percent of first-time failures.

Any complex SoC design requires several design iterations before reaching design convergence. Current design, verification, and refinement cycles take too long, lengthening time-to-market and, possibly, dooming the system incorporating that chip to failure. Often a product must be shipped to market prior to fully verifying that the chip meets design specifications, with the hope that “all will go well” when the chip goes to fabrication. This is a bad idea since, according to Electronic Market Forecasters (June 2003), most embedded systems are too slow — 70 percent of all embedded systems miss their design performance goals by more than 10 percent.

A good hardware/software design and verification environment detects both functional and firmware problems and accelerates design convergence prior to commitment to silicon. Such hardware/software design environments must address the shortcomings and limitations inherent in typical hardware/software co-design tools and tool suites.

**Integrated Solution**

This article presents a hardware/software design and verification environment that addresses the design capture, verification, analysis, and refinement tasks necessary to meet chip and system specifications. Because this set of tightly integrated tools specifically supports the ARM architecture, it speeds the initial design capture and verification of ARM microprocessor-based SoCs, increasing the chances of first-pass success. The solution is comprised of three tools, each addressing critical areas of a chip’s design entry and iteration cycles (Figure 2).

- **Platform Express** — Rapidly captures and builds verification environments for ARM microprocessor-based SoCs. Platform Express enables changes (resulting from the verify-analysis-refine cycle) to be

**Figure 1:** There are several causes of first-time silicon failures, dominated by logical and functional design errors.
configure, and instantiate IP in a system block diagram. There are three steps involved in capturing an SoC with Platform Express: selecting and configuring the IP, instantiating the IP into the design, and building the complete SoC (Figure 3).

**Selecting and Configuring the IP**

You start by selecting the desired ARM solution from a menu of processors, memories, and peripherals. As the IP is dragged onto the design, you can use the automatically generated pop-up configuration window to set any user-configurable options. If some of the parameters are unfamiliar, it is safe to click on the OK button — because if the default options are not the optimum settings for this design, reconfiguring the component later is easy. In fact, if the IP designer is familiar with the IP that you are trying to instantiate into the design, he may have already set up the configuration options — you may not even be aware that there were options to be set.

**Instantiating the Design**

Next, you instantiate the selected component by placing it into the Platform Express graphical editor. Many components have multiple interfaces to connect to the various system busses. In Platform Express, it is very easy to drag the various bus connectors onto the desired busses.

For instance, the ARM926EJ-S processor features separate AMBA AHB bus master connections for address and data transactions. You can connect both connectors to the same AMBA system bus (in which case, Platform Express will automatically generate the appropriate AMBA bus logic and arbitration). Alternatively, to increase the overall system bandwidth, the separate AMBA data and instruction bus connectors can be utilized in more exotic ways. For instance, ARM supplies a multi-layer AMBA bus infrastructure component which acts like a telephone switch within the design. Normally, when all components are connected together on the same bus, one master can talk to one slave at a given time. All other communications have to wait until that initial transaction is complete. In multi-layer AMBA designs, multiple masters can communicate with multiple slaves simultaneously (a bit like a telephone exchange where lots of telephone calls can be sustained simultaneously, but two callers cannot call the same number at the same time).

Multi-layer AMBA components have up to 16 AMBA connections, and allow multiple masters to talk to multiple slaves at the same time: this can increase the overall system bandwidth significantly. There are important design trade-offs to be made so that selecting increased bandwidth does not result in excessive logic in the design, but the complexity of generating alternative designs to find the optimum configuration of logic and bandwidth can be complex and time consuming.

However, in Platform Express it is easy: simply drag the multi-layer AMBA component onto the design and drag the AMBA connectors from the processor onto the appropriate input ports. The whole design will regenerate automatically. If there are not enough AMBA connectors onto which the various components can be connected, a couple of clicks of the mouse will regenerate the multi-layer AMBA component with the required number of connectors.

When you rapidly reconfigure design architectures in this way, the programmer’s view of the design changes significantly. In multi-layer AMBA multiprocessor systems, you may find that the same components appear in different address locations in the address maps of each processor. Software that worked in single bus, multiprocessor architectures may need to be reconfigured to work in the new version of the design.

Platform Express automatically tracks the changes in the programmer’s view, and the full address map as seen from any point in the system design is always on display. In fact, software modules can be regenerated in Platform Express to be consistent with the changing memory map.
Building the Complete SoC
Platform Express automatically generates the system design along with the software drivers and the stimuli to run the design. Platform Express also invokes verification tools, generating and executing the scripts that run on them.

Designers can perform several verification tasks using Platform Express capabilities:

- Conduct RTL hardware verification using an HDL simulator such as ModelSim. Platform Express generates the relevant ModelSim setup and waveform displays.
- Generate drivers and hardware diagnostics. Platform Express puts the diagnostic code for each of the design’s peripheral and memory components into a single C-code application for running on the system hardware. It is now ready to verify with Mentor’s XRAY debugger.
- Prepare the design for running on Seamless, Mentor’s hardware/software co-verification tool.

Design Verification and Analysis
After capturing or modifying the embedded system, the Seamless hardware/software co-verification environment lets you create a virtual prototype of the ARM platform before a physical prototype is available. By executing embedded software on simulated hardware, you can detect and correct hardware/software interface errors very early in the design cycle. This reduces the possibility of a problem remaining hidden until system integration, at which time software is complete, delaying design closure while the problem is corrected. Seamless also gives you full visibility and control of hardware and software execution, accelerating the software debug operation, which is often a limiting factor in system design completion.

Dynamic Optimization
A key element of hardware/software co-verification is the ability to run meaningful amounts of code against the virtual hardware. During software execution, an ARM embedded processor generates millions of memory transactions, both instruction fetches and memory reads and writes. Servicing millions of transactions with a logic simulator running at 10 to 20 clocks/second would take days. For example, booting the LynxOS RTOS initiates 16.8 million memory transactions. At 20 clocks/second, it would take 9.7 days to boot the LynxOS. Seamless uses patented technology to accomplish this task in about 2 to 3 minutes.

Seamless maintains a unique storage array, the Coherent Memory Server, for the memory space that is accessed frequently by the ARM processor. This lets you choose whether access to a given memory range is serviced by the logic simulator or intercepted by Seamless. Either path results in access to the same storage array, but Seamless can process a request 10,000 times faster than a logic simulator (Figure 4).

You can switch memory access modes at any time during simulation to achieve either detailed simulation of memory cycles or rapid execution of software, depending on the situation. For example, early in the design process, it is important to validate the design’s memory subsystem to ensure that the physical prototype will boot in the lab. Once a number of access cycles of a given memory region have been simulated, Seamless memory optimizations can be applied to speed execution. If an area of interest lies deep in the simulation, Seamless optimizations can move quickly to that point and then be switched off to provide the full simulation detail required to isolate a problem.

Because memory contents are stored in only one location — the Coherent Memory Server — data integrity is maintained no matter how many times the access method is changed from optimized to detailed operations. The Coherent Memory Server supports multiprocessor designs, processors with multiple memory spaces, and a wide range of common memory-management techniques. Seamless makes use of more than four thousand Denali memory models. Any that are not available to Seamless can be easily converted from their Verilog or VHDL source code to a Seamless model. Conversion does not require any change to the system’s embedded code.

Processor Support Models
Mentor Graphics continues to be a leading supplier of ARM processor co-verification models. Seamless has processor support packages for the ARM7™, ARM9™, ARM9E™, ARM10™, and ARM11™ families of processors. Each processor model includes three components: an instruction set simulator (ISS); a graphical software debugger (XRAY or RealView); and a bus interface model (BIM) that connects Seamless to an HDL or C hardware simulation environment.

C-model Support
Seamless C-Bridge technology supports C hardware modeling, taking advantage of the faster simulation speed of C models over HDL models. A simple API connects Seamless to the C models. Designers can work with either an all C hardware-modeling environment or with a mix of C and HDL design descriptions. The C hardware models can be in C, C++, or SystemC.

When a design team chooses to model some or all of the hardware in C, they must eventually make the transition to HDL and re-verify the design before tape-out. Seamless C-Bridge is an ideal tool for making this transition. It allows easy integration of C hardware descriptions and
enables switching between a C and HDL model for a given function with a simple change to the memory map.

**Performance Analysis**

An important feature of Seamless is the ability to graph key performance metrics. This capability allows you to visualize performance bottlenecks and understand the effects of the design changes you make to eliminate these bottlenecks. Seamless can display charts of software profiling, memory transactions (including cache analysis), bus transactions, and bus-arbitration delay.

Seamless profiles software functions as either a Gantt chart, showing the sequence and duration of function calls (Figure 5), or a bar chart, showing how much time each function call consumes (Figure 6). The Gantt chart representation helps the designer determine if critical functions are occurring at the right times, helping to prevent errors such as incomplete data transfers or dropped packets. Knowing which functions are the most time consuming allows you to improve overall system performance by speeding up these functions. Techniques such as rewriting C-code routines in assembly code or moving software-implemented functions to hardware can be used.

A graph showing memory transactions versus time is valuable for highlighting peak memory utilization periods and for showing how to better balance memory utilization. This type of graph is also useful for displaying ARM processor reads, writes, and fetches to memory and for detailing instruction and data cache hits and misses. You can use this information to optimize cache size and configuration in order to accelerate firmware execution and minimize the CPU load on the main memory. The memory-transaction and software-profile graphs can be used together to correlate a memory bottleneck to the firmware or function accessing the memory.

The bus-analysis displays let you observe bus loading and bus arbitration. In today’s SoCs, bus bandwidth is at a premium, with processors, DMA controllers, peripherals, and data search engines all competing for this resource. A graph displaying bus utilization versus time that flat-lines at 100 percent (Figure 7) indicates a bandwidth-limited operation. If this occurs, you can implement changes that eliminate the bandwidth bottleneck by reviewing the appropriate software-profile graphs to determine which function calls are occurring at the time of the bandwidth limitation.

A graph showing memory transactions versus time is valuable for highlighting peak memory utilization periods and for showing how to better balance memory utilization. This type of graph is also useful for displaying ARM processor reads, writes, and fetches to memory and for detailing instruction and data cache hits and misses. You can use this information to optimize cache size and configuration in order to accelerate firmware execution and minimize the CPU load on the main memory. The memory-transaction and software-profile graphs can be used together to correlate a memory bottleneck to the firmware or function accessing the memory.

*Figure 5: A Gantt chart shows the sequence and duration of function calls, helping the designer determine whether function sequencing is correct.*

*Figure 6: A bar chart, showing the amount of time consumed by each function call, identifies which functions are best to accelerate in order to improve overall system performance.*

*Figure 7: A bus utilization graph shows the existence of bandwidth bottlenecks (flat-lining at 100%).*
profile graphs to determine which function calls are occurring at the time of the bandwidth limitation.

A bus arbiter controls access to a particular bus. Seamless can display a graph showing the wait time between a bus request and a bus grant for a specific bus master. You can use this information to choose the most efficient bus arbitration scheme and function call priorities for the system under development.

**Design Analysis and Refinement**

After evaluating an ARM technology-based design, you may decide to investigate moving one or more software functions to hardware. Before committing to this conversion, you have to answer several questions, including:

- How many clocks will the function take in hardware versus software?
- What is the communication overhead (ARM to hardware and back to the ARM)?
- How much area will be occupied by the additional hardware?
- How much performance improvement can be achieved by converting to hardware?

Mentor’s Seamless ASAP tool helps answer these types of questions. ASAP automatically converts software functions to hardware (creating one or more Assistant Processors), enhancing overall system performance.

**Estimating Assistant Processor Impact**

The ASAP Estimator shows the ramifications of moving a software function into hardware without lengthy re-simulations. The Estimator provides:

- An estimate of the Assistant Processor area
- The time to perform the function that was initially in software
- Loading changes on the ARM processor and AMBA bus

Based on these estimates, the designer can decide whether the benefit of using ASAP to move some software functions to hardware is worthwhile.

**Generating an Assistant Processor**

ASAP takes a software function, in standard ANSI C, and converts it to synthesizable HDL code. The Seamless ASAP Hardware Generator supports virtually all ANSI C constructs and coding techniques utilized by embedded developers. Thus, most embedded C can be converted to hardware without requiring the modification or addition of “hardware” C constructs, a common restriction or requirement in other solutions.

**Generating the Proper Software/Hardware Interfaces**

After creating an Assistant Processor, ASAP generates a software driver with the same function signature as the original software. This driver modifies the remaining software, allowing it to take advantage of the new hardware function. ASAP also creates the AMBA bus interface. This lets you place the Assistant Processor on the ARM core’s AMBA bus, the TCM bus, or co-processor interface.

**Creating a Testbench**

To verify the functionality of the Assistant Processor, ASAP creates an HDL testbench — with stimuli and expected responses. This testbench verifies the HDL code by executing the AMBA interface and the newly created Assistant Processor. During the performance analysis step described earlier, all software function calls were captured. The results of the HDL simulation are compared against the captured function calls, giving a simple pass/fail on the equivalence between the hardware and software simulations.

Seamless ASAP can also add the new hardware function to the Platform Express IP library to use in existing or future designs.

Figure 8 shows an example of the performance enhancement ASAP provides for an environment. Four functions were moved from software to hardware by manual conversion and by using ASAP. Hand-conversion results were only slightly better than those obtained with ASAP (6.4 percent for speed and 14.8 percent for area), but they took much longer to obtain (6 months versus 1 week).

**Accelerated ARM Embedded System Development**

The sum of these three powerful tools, along with links to other verification tools, provides an intuitive and comprehensive environment to capture, verify, analyze, and refine an ARM processor-based design. By providing tools that address all phases of system hardware/software design and verification, Mentor Graphics accelerates the system development cycle, yielding the optimum design and reducing project risk.

**Figure 8:** This example shows how ASAP produces results almost as good as those obtained with handcrafted SW-to-HW conversion, but much faster.