EMBEDDED-SOFTWARE DEVELOPMENT IS AN IMPORTANT PART OF MOST SOC DESIGNS.

CHOOSING THE RIGHT METHOD TO VERIFY THE SOFTWARE CAN MEAN YOUR PROJECT’S SUCCESS OR FAILURE.

Comparing debugging methods for embedded software

An SOC (system on chip) includes many components, such as processors, timers, interrupt controllers, buses, memories, and embedded software. It is a complete system that designers a few years back would have assembled on a board but, thanks to advances in semiconductor technology, can now fit on a single chip. The traditional RTL-to-layout design-and-verification flow proves inadequate for these multimillion-gate systems, which have the added complexity of embedded software running on them. Engineers cannot wait for the silicon or even the FPGA-based emulation board to be available to start debugging the software. The advent of virtual platforms allows designers to model hardware-component functions at a higher level of abstraction, which simplifies the debugging of embedded software. Virtual platforms offer reduced development cost, early availability, powerful hardware-visualization features, and the ability to model erroneous and rare conditions.

In general, virtual platforms provide a better option than traditional approaches to debugging embedded software using FPGA boards. However, you cannot count on only one approach during different stages of the software-design cycle. Trade-offs are possible, considering cost and usage patterns, as well as application functions and time-to-revenue considerations. For functional testing, virtual prototypes and FPGA prototype boards are useful. For performance tuning, an RTL cosimulation platform helps, especially for small, low-level hardware drivers. Software designers must wait for the silicon prototype of the product to become available before they can do the final performance tuning. The length of the performance-tuning cycle depends on the application in question and generally accounts for 10 to 30% of the software-development cycle.

With the advent of software-rich chips, presilicon architectural analysis and verification of embedded software are increasingly gaining importance, leading to a widespread increase in the development of “software-rich” SOCs (Reference 1). Designers of SOCs are responsible for making these platforms extremely reliable from an operation standpoint. Engineers need to validate the performance and function of hardware and to validate the embedded software in these systems. Choosing the right prototyping method is sometimes more difficult than it sounds. Designers can use many approaches to validate the performance of these systems, from hardware-based simulation to software-based simulation (references 2 and 3). The following options are available for validation of both software and hardware:

- Functional testing entails the development of functional/architectural models (virtual platforms) that emulate the functions of the hardware. It includes using an ISS (instruction-set simulator), such as ARMulator, to execute the embedded software (references 4 and 5).
- An RTL-based approach relies on methods such as cosimulation mechanisms and verification using behavioral models.
- Designers use reconfigurable FPGA-based prototyping.

Comparing debugging methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Cycle Time (Months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>30</td>
</tr>
<tr>
<td>FPGA-Based Prototyping</td>
<td>24</td>
</tr>
<tr>
<td>RTL Cosimulation</td>
<td>22</td>
</tr>
<tr>
<td>SystemC Functional Models</td>
<td>16</td>
</tr>
</tbody>
</table>

Figure 1 Total development time for a digital-camera chip varies among platform approaches for the validation of embedded software at STMicroelectronics.
design feature  Embedded-software development

totyping boards to map all or part of the hardware for verification of a relevant piece of embedded software.

• ASIC boards allow engineers to verify the software on silicon.

The RTL-based cosimulation mechanisms and the verification of embedded software using prototyping boards and ASICs are relatively common and mature methods. The functional-testing approach is still in its infancy and is becoming popular with the advent of TLM (transaction-level modeling) using SystemC (references 6 and 7). These system-hardware components are represented at a higher abstraction level than RTL models. TLM of SOCs leads to a new way to test embedded software. Using this method, engineers model all the hardware IP (intellectual property) at the transaction level using SystemC and an ISS-processor model. The embedded software runs on the ISS model, which uses TLM models of hardware IPs. You can analyze the benefits of TLM-based methods for embedded-software development from various points of view.

DEVELOPMENT TIME

Because SOCs contain increasingly complex embedded software, it is important to verify its performance before fabricating silicon. Traditional hardware-simulation approaches are inefficient, because they take a long time to develop and validate the RTL models of IPs. You cannot build reconfigurable boards unless the RTL models are available. Engineers can develop functional models more quickly than comparable RTL models. The development of SystemC-TLM models generally requires one-tenth the time of the equivalent RTL models (Reference 8). Thus, these models can be ready before the RTL models or the prototyping boards are. Building and debugging evaluation boards further delay the validation of embedded software. Moreover, the hardware is rigid with respect to performance changes. A small change in the function of an IP may delay the verification of the overall platform due to the absence of updated hardware further increasing the overall cycle time. Figure 1 shows the overall cycle time for the development of the latest STMicroelectronics digital-camera chip using different approaches for validating the embedded software.

SIMULATION SPEED

Simulation speed is an important criterion for testing embedded software. The embedded software should not take a long time to execute on a platform so that it does not affect the engineer’s productivity in editing, compiling, and debugging. The requirement becomes even more important when you need to verify application software written for an operating system. Even changing the application software entails compiling the software and rebooting the operating system, because you cannot store and reload the system state at different checkpoints. If the simulation is slow, as with RTL-based platforms, each test can consider only small portions of the software.

Simulating software running on the actual target hardware is almost always out of the question. Custom ASIC hardware provides maximum simulation speed. However, using FPGA-based approaches, designers can run the actual operating system, all the firmware, and application software at simulation speeds usable for early development and debugging, long before the actual silicon is available. Functional models provide better throughput than RTL models but are slower than hardware-based approaches by an order of magnitude. Figure 2 shows the times required to boot the Symbian OS on the ARM Primexsys Platform using different approaches. The most important development has been in the area of SystemC-based functional platforms, which now achieve simulation speeds comparable with the FPGA-based prototyping platforms by increasing the speed of the ISSs.

RESULTS ACCURACY

When modeling with behavioral models or ISSs, designers sacrifice accuracy but improve speed and ease of debugging by using general-purpose workstations and servers. Using full-HDL or gate-level simulations improves accuracy, but execution speed drops so low that engineers can analyze only small portions of the design in any one test, and integration tests take days to run. FPGA-prototyping boards provide more accuracy with reasonable simulation speeds, but they are unsuitable for verifying analog blocks and do not accurately support asynchronous modes of operations.

TABLE 1—ANALYSIS OF COST FACTORS AMONG DEVELOPMENT PLATFORMS

<table>
<thead>
<tr>
<th>Type</th>
<th>Custom ASIC</th>
<th>FPGA-prototyping Boards</th>
<th>RTL cosimulation</th>
<th>SystemC functional models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle time</td>
<td>Poor</td>
<td>Poor</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>Simulation speed</td>
<td>Good</td>
<td>Good</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Good</td>
<td>Poor</td>
<td>Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Cost</td>
<td>Good</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Hardware visualization</td>
<td>Poor</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Interactive applications</td>
<td>Good</td>
<td>Good</td>
<td>Poor</td>
<td>Poor</td>
</tr>
<tr>
<td>Error modeling</td>
<td>Poor</td>
<td>Poor</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

The time to boot the Symbian OS on the ARM Primexsys varies as a function of the platform approach.

Figure 1

Figure 2
best option is to verify the software on the actual ASIC to test all possible aspects of the platform before certifying it as bug-free.

COST

Cost is another factor that needs special consideration, and it encompasses both development and deployment if you purchase off-the-shelf IP from third-party vendors. The development of virtual platforms explicitly for software validation to decrease the overall development time can add significant cost to the project. RTL models must be developed for FPGA-based prototyping boards and custom ASICs. Thus, these virtual platforms require additional overhead. From the perspective of deployment, the RTL-based cosimulation-platform approach is less costly, due to the minimal incremental cost in setting up the simulation environment. Third-party vendors generally supply the RTL models of all the IP blocks they license, but the development team incurs the cost of integrating the models and assembling the hardware-prototyping boards. This activity can be costly, making the strategy infeasible. Licensing virtual platforms or developing custom ASIC for embedded-software debugging is expensive. Amounts vary depending on the type and complexity of IP; Figure 3 shows a qualitative comparison. The overall cost of debugging the embedded application would decrease by initially using virtual platforms and developing only the number of prototyping boards at a later stage to complement the virtual platforms.

HARDWARE VISUALIZATION

Efficient and cost-effective hardware-visualization techniques are necessary for debugging embedded software. You can use many tools on FPGA-based prototyping boards, including logic analyzers; bus analyzers, which monitor activity of system buses; and LEDs. By connecting a JTAG cable from the FPGA prototype to a PC, designers can load and examine memory and all the critical internal CPU registers, as well as provide software-breakpoint capability. Also, FPGA-mapping tools offer designers additional ways to see any signal in the prototype. However, the scope of each of these tools is limited in the sense that you cannot record or view every activity that takes place inside the IP, thus comprising the system. Moreover, debugging the signals using these analyzers is a tedious job.

Custom ASICs are less suited to this purpose than are FPGA-prototyping boards, because they are black boxes, making any kind of hardware visualization impossible without predefined explicit support. In virtual platforms, such as SystemC-based TLM platforms and in RTL-based platforms, probes can record traffic at both the transaction and the signal levels. Engineers can view the traffic using commonly available waveform viewers. This information is helpful when the embedded software depends on the external interface. For example, in case of interrupts, you would like to know when the interrupt asserts and when the interrupt-service routine starts. This type of the information is difficult to extract in FPGA-based platforms but is readily available with both TLM and RTL platforms.

The generation of hardware event logs is another important aspect of hardware visualization. The logs can contain information corresponding to each IP and bus in the system. You can generate logs corresponding to register reads/writes, in/out transactions with external devices, interrupts that the hardware raises, and so on. Virtual platforms can use debuggers, such as gvd/gdb, to step through the hardware code. It is difficult to achieve equivalent functions on custom boards that need to enter a special testing or debugging mode to provide the same amount of debugging flexibility. The user has more control when using virtual platforms, so it is possible to freeze the simulation and analyze the state of the IP. This approach is impossible when using actual hardware; for example, in a hardware platform, the timer would continue to work even after the software simulation stops, and it would be hard for an engineer to see the state of the timer when an error occurs.

Virtual platforms, such as TLM-based platforms, provide a host of features, including probes to record bus transactions. Standard debuggers, which you can use to step through the hardware code, come with mechanisms to regulate the logging corresponding to each IP and engineers can always use standard ISS de-
buggers to step through the software code. Thus, these platforms provide a completely transparent view of both the hardware and the software, thereby greatly aiding the debugging of embedded software. The ability to provide a series of random, directed, or specific input stimuli to the platform to emulate real-life scenarios is an important requirement in validating embedded software and involves I/O devices, such as keyboards, mice, LCDs, touchscreen panels, hard-disk drives, and memories. FPGA-based prototyping boards and custom ASICs enjoy a slight advantage in this regard, because they are readily available at a lower cost, and you can integrate them with a complete system. Virtual platforms require an additional modeling cost proportional to the number of devices they require.

ERROR INJECTION

In many cases, the embedded software comes with error-handling routines that execute whenever the system detects an error. It is important to validate these error-handling routines, because any bug in them may render useless an erroneous device. As an example, embedded software for a memory almost always includes ECC (error-correction-code) routines that execute whenever a sector or track in memory becomes erroneous. To check these routines, you must have a mechanism to induce the errors in the memory. It is difficult in prototyping and custom boards to induce errors in components unless the designers make special provisions. Even in those cases, however, the types of errors are rigid. For behavioral and functional platforms, designers can induce user-defined errors at any level of granularity—in memory, for example, a single address error or a row/column error. A configuration file that you specify at runtime can provide the sequence of errors.

This approach emulates a real-use scenario. When error handling is important to the system, virtual platforms are indispensable for debugging error-handling routines. Engineers can use the same approach to emulate rare conditions that may arise in a component, thus allowing for greater code coverage and increased quality of the embedded software. The limitation of this method is that you need to explicitly model each error case, and the accuracy of the underlying mathematical model limits the quality of results.

RESULTS

Table 1 compares the results of using different platforms. The initial purpose of debugging software is to test its correct functions. This process requires high-speed simulation and good hardware-visualization features. SystemC-based functional models are best for early debugging. Once they are confident of the performance, engineers aim to optimize and improve the accuracy of the embedded software. These activities require debugging using custom-ASIC models and RTL-based cosimulation mechanisms, because these approaches provide the best approximation to system execution. FPGA-based prototyping boards and SystemC functional models are less useful at this stage.

When developing large systems, simulation speed is important, and SystemC-based approaches offer engineers the best approach. The cost of debugging can be a considerable factor for software-rich chips, and the amount you’d like to invest in software development can become the determining factor in choosing a debugging platform. Although engineers developing interactive applications may prefer FPGA-based techniques, SystemC may be a better choice for applications with lots of error handling.

REFERENCES


7. Jindal, Rohit, Kshitiz Jain, Srikanth Reddy, Early Hardware/Software Co-design with TLM using SystemC.


AUTHORS BIOGRAPHIES

Kshitiz Jain is a senior design engineer at STMicroelectronics, where he serves as a project leader of the System Level Design Flow development team for SOCs.

Rohit Jindal, a design engineer at STMicroelectronics, serves as the project leader of the System Level Design Flow development team for SOCs.

Mamta Bansal is a design engineer at STMicroelectronics, where she serves as a member of the System Level Design Flow development team for SOCs.

Bhuvan Middha is a design engineer at STMicroelectronics, where he serves as a member of the System Level Design Flow development team for SOCs.

TALK TO US
Post comments via TalkBack at the online version of this article at www.edn.com.