Develop Software Earlier: Panelists Debate How to Accelerate Embedded Software Development

A lively and open discussion explored tools and methodologies to accelerate embedded software development.

by: Lauro Rizzatti  July 05, 2017

At the Embedded Systems Conference (ESC) Boston in May, I moderated a panel titled, “Develop Software Earlier.” Joining me were Jason Andrews, Principal Solutions Architect in the Software Group at ARM; Mike Dini, Mike President of DINI Group; and Russ Klein, Technical Director in Mentor Graphics’ Emulation Division.

What follows is an excerpt from our lively and open discussion on tools and methodologies to accelerate embedded software development.

Lauro:  What are the technologies to develop software and what are the most effective ways to use them to start and finish software development earlier?

Russ: When people think about starting to develop software earlier, they often don’t think about emulation. Software can be developed on emulation. Emulation represents the earliest cycle-accurate representation of the design capable of running software. An emulator can be available very early in the design cycle, as soon as the hardware developers have their design to a point where it runs, albeit not fully verified, and not ready to tape-out. At that point, they can load memory images into the design and begin running it. Today’s emulators support powerful debug tools that give much better debug visibility into the system, and make it practical to start software development on emulation, sooner than ever before.

Jason: There are six techniques for pre-silicon software development: FPGA prototyping; Emulation; Cycle-Accurate/RTL simulation; Fast instruction set simulation; Fast models and an emulation hybrid; operating system simulation (instruction set abstracted away). Most projects adopt two or three since it is too difficult to learn, setup and maintain all of them.

When you start a project, you write code, and the first thing that occupies your mind is how to execute that code. Software developers are creative about finding ways to execute their software. It could be on the host machine, a model, or a prototype.

The next thing you need to think about is how to debug the software, because the code we write doesn’t always work the first time. There are various ways to perform early software debugging.

Then, less known, are performance analysis and quality of software.

You should try to get a good handle on performance as soon as possible. It could be done with the emulator, an FPGA prototype board, or other means. It is important to get some grasp on
the performance of your code because often you are doing algorithm work and you need to get it under control as quickly as possible.

The last thing is the quality of the software. In pre-silicon software development, we always have milestones like system boots or test pads that don’t tell you much about the quality of your software.

You want to run, debug, get a handle on performance early on and try to do what you can for boosting software quality.

**Mike:** I’m the hardware guy, and I have to deal with software people. While early software development is fine and good, it’s too slow and too abstract. You find that software people tread water and until they get further along the hardware cycle and have something that gives blue screens of death or smoke when there are bugs. That’s when real software development is accomplished. There is only so much you can do until you are far enough in the hardware development process to convince software people to actually start developing their code.

FPGAs can be used to prototype complete systems long before real silicon is available. FPGA-based prototypes are much closer to the final system in speed and functionality, making this approach a good fit for early software development. It is, unfortunately, hard to deploy and use.

**Lauro:** Jason, you are known as a leading authority on virtual prototyping. What are the advantages?

**Jason:** At ARM, we provide various types of models to perform early software development, but there are tradeoffs, and pros and cons. Virtual prototyping is extremely flexible. You can run your codes on a model sitting at your desk. It’s abstracted, runs fast but may not have all the detail. Still, it is probably the best way to get functional issues out of your code right away.

The good thing about fast models is they are available early. ARM has tons of processors coming out all the time and many of our partners are adopting leading-edge processors. Unfortunately, it’s going to be a while before they see any hardware. What this means is, they can use fast models to start running their code.

The minus is the case of the missing model, a problem for 20 years. We provide models of ARM IP but a chip isn’t all ARM IP. Many IP come from different vendors from different places in different forms, and there is always the case of a missing model at the OEM level — if you are buying chips and putting them on boards, if you have a problem with your microcontroller, a model may not be readily available, the hurdle over the last two decades.

The technology is available to run software extremely fast and, more often than not, as fast as your actual hardware. The missing models plague the landscape.

**Mike:** Sounds good, except the models never exist and really performance is not that good. In my experience, it’s hertz rather than megahertz.

**Jason:** There are two kinds of models. In one domain, you have fast models. They are called fast because they are fast and use dynamic binary instruction translation. If you are targeting an ARMv8 instruction set, it’s running those instructions on your X86 workstation at a pretty high
speed. Normally, 100 megahertz would be a reasonable speed depending on the size of your system.

The other domain is what we call cycle models. These are the accurate models but they run much slower. Cycle models are extremely limited in terms of what you can do with them because anyone who needs them wants to see the accuracy of a processor, interconnects and the memory controller. They tend to be used in the hardware architecture domain to run low-level stuff to figure out throughput and latency. They are not made for serious software development other than benchmarking boot code, and executing low-level hardware tests.

Lauro: What is the advantage of using FPGA prototyping over emulation or virtual prototyping?

Mike: There are places in the design cycle for virtual prototyping followed by emulation followed by FPGA prototype. But, first let me discuss design capacity.

The largest FPGA shipping today maps approximately 30-million gates. This is the Xilinx UltraScale 440. When your design exceeds 30-million gates, you have to partition your design across multiple FPGAs. Partitioning is a nightmare. The larger the design, the more FPGAs needed. When you try to partition it across multiple FPGAs in prototyping hardware, whether an in-house custom FPGA board or a commercial FPGA board, you have a few hundred connections between FPGAs but need thousands or tens of thousands of connections. Now you have to multiplex the I/O pins that slow down the speed of your prototype. There are automated tools to implement multiplexing, but it may be worse than nothing, and not helpful. It can be difficult to partition a design effectively. In terms of max design capacity, while many vendors claim they can handle a billion gates, 200-300 million in the range of eight to 12 FPGAs, is probably the max before it becomes untenable.

Even though prototyping is hard to do, the advantage of FPGAs is that it’s much faster than emulation. Emulation may run in single digit megahertz. It is not unreasonable to get 50 megahertz out of an FPGA. If your ASIC designers are part of the team, you can get faster speed than that. You can get about a tenth of the frequency of your ASIC. When you are running it with all the peripherals in place, your software guys are seeing somewhat real-time performance out of the unit. This is about 10 to 20X of what you get out of an emulator. IoT designs require real time debug. Often times the emulator is too slow in response time to do what you need. This is a clear advantage of FPGA prototyping.

The other advantage of FPGA prototyping is that they are dramatically cheaper than emulators. A 100-million gate FPGA board may sell for $100,000 where an equivalent emulator may cost you $1 million. If you have a large team and want to replicate these things, it may be worth the effort and time if you have enough information to do an FPGA prototype rather than use emulation.

Lauro: How do you view today’s verification landscape? Is it one size fits all or specialized per application?

Russ: The goal of verification is dependent upon what the final target is and how much you can verify. Security and correctness are going to be different for a medical device versus a toy.
Those two products are going to be validated to different levels. As security and correctness become more important, we have more and more systems where the correctness of that system is going to impact whether people live or die. The amount of verification you want to achieve is substantially higher than it has been in the past. Jeremy Clarkson, who is the host of Top Gear BBC show, a while back quipped that one day your automobile is going to have to decide whether to kill you or not. He was talking about a car that will be an embedded system with hardware and software. If it goes around the corner and sees some school kids on the sidewalk or a brick wall, what do you do? Cars will have to decide. We as tool vendors need to enable folks to verify things to the degree necessary to assure it is alright.

**Jason:** Unless you work in a large company with an infrastructure to do virtual prototyping, FPGA prototyping and emulation together, by the time you learn all of them, your project is over. You have limited time to decide what to use to get the software done with higher quality and better performance, whatever it is.

I think people choose different techniques for early software development and make a project decision about which approach make sense when. One possible scenario is a design group using FPGA systems for running tests in their lab. They have virtual prototypes to run regression tests on a subset of the system or to execute certain software algorithms just to make sure they don’t break anything or to assure the code is functioning.

**Russ:** And you are going to get different characteristics out of these different models. A fast model is going to perform fast. It’s going to tell you how functionally the system will work but limited in terms of the detail or performance. An emulator is going to be much more accurate in terms of clock cycle counts but it’s going to run lot slower. It’s what are we focusing on to verify at that particular stage of the development?

**Lauro:** Outline the ideal verification flow for a large and complex networking chip that will go into a data center. Budget is no limit, resources are plentiful but the timeline is not. All the tools will work together seamlessly. Which ones would you implement?

**Mike:** You start with a high-level model of what is going to run, some sort of virtual prototype, and start software development. As you get farther down, put it in an emulator and do it faster, closer to the real hardware, and then toward the end, just before tape-out, put it in FPGAs and run it quicker yet.

**Jason:** My take is different because I see power users, that is, people deploying leading-edge technology, adopting the latest ARM CPUs in the largest systems and go right through to an emulator. They don’t waste time with much of anything else. They have farms of emulators, and throw more emulators and more embedded software people at the problem.

**Mike:** That sounds like an infinite budget. We are talking tens of millions of dollars or more.

**Jason:** In the old days, we used to create testbenches and perform block-level verification and build up a chip design piece by piece. Now there is so much IP reuse that people are buying from so many sources and a race to market. People are buying Verilog files from all the places they can get to make a chip, throwing it into the emulator and turning loose couple of hundred software people to make sure the system works. You write software, tests, all the functionality of
all the peripherals and everything else to the shortest path with an unlimited budget and unlimited people.

**Mike:** And it does occur, actually. There are customers who will tell you that budget is not an issue. There are not many of them, but there are three, four or five that I can think of.

**Russ:** What a wonderful vision that is.

I would argue that there is probably a lot of software from the operating system on up that you would never want to take into the realm of an emulator. You want to run on a virtual prototype. In an ideal world, you want the software guys peddling as soon as possible and get a virtual prototype ahead of an emulation design. Again, in an ideal world with unlimited money, you want to get detailed virtual prototype and start software development six months before you start emulation for more work done up front.

**Lauro:** What trends do you see in the verification space, what new challenges are emerging?

**Russ:** One trend we clearly see is as we get more and more processors and processors get cheaper and faster is more of the functionality of the overall system moving into software because it is cheaper to create than hardware. To the degree we have the ability to do that, more and more of that functionality goes into software. Hence, the functioning of the overall system now depends not just on hardware, but hardware and software working together. Fundamentally, you can’t wait until somebody throws the hardware over the wall to start developing software because it’s going to be an integral part of that system. As hard as it is to debug them together, we have got to start sooner.

**Jason:** Software complexity is getting much worse. Think about hypervisors, virtualization, like in automotive where you are going to have this type of virtualization. In the old days, you had a CPU and you could start it up, write some assembly code and get your application. Now, it’s complicated to deal with. Even simple interrupt controllers in an ARMv8 core has a thousand system registers. Do you program all one thousand correctly? Do you need to program even any of them? Software complexity is getting more and more complicated. Reuse goes up and people start cutting and pasting, copying code, and the system looks like it’s working but there is some fundamental problem that’s hard to detect. That scares me — all the complexity in the software stack, getting it right so that it is secured and the quality is good.

**Mike:** I don’t see anything revolutionary. It’s evolutionary where it gets bigger, faster.

In the past, we built boards with 20 FPGAs. Today, we build boards with four Virtex UltraScale FPGAs and stack the boards to expand design capacity. The question is, how many boards can we stack before we hit a wall? For instance, on a system that includes four different stacked boards, we can achieve 10 megahertz because of the pin multiplexing, although the testbed speed may reach 20 megahertz.

**About the Moderator**

Lauro Rizzatti, Verification Consultant
Dr. Lauro Rizzatti is a verification consultant and industry expert on hardware emulation.
Previously, Dr. Rizzatti held positions in management, product marketing, technical marketing, and engineering.

About the Panelists

**Jason Andrews, ARM**
Jason is Principal Solutions Architect in the software tools group at ARM. Previously, he worked at Carbon Design Systems and Cadence in numerous pre-silicon software development projects utilizing fast models, cycle accurate models, emulation and FPGAs.

**Mike Dini, The Dini Group**
Mike Dini is President of Dini Group. He has been a specialist in the design and application of FPGAs for the last 30 years.

**Russell Klein, Mentor Graphics**
Russell is a Technical Director in Mentor Graphics' Emulation Division. Russ has more than 20 years of experience developing design and debug solutions that span the boundary between hardware and software.

Reprinted with permission from Design News – UBM Americas