

Hardware and Software, Together at Last

Remember the last time you integrated new hardware and software? Painful, wasn't it? Why was the experience less than relaxing? Quite simply, it was because specifications are rarely sufficient enough in detail to pin down the exact functionality required. Projects are difficult enough when they reach a critical mass where a system must be partitioned for allocation to separate teams from a single discipline, but when the system function is divided for separate implementation by hardware and software teams, something is going to give.

Not that development team members aren't diligent, far from it, but communication problems occur because the two groups have very different educational backgrounds and different ways of looking at a problem. What seems like an acceptable solution to one group is less appealing to the other. An aggravating factor exists when the two parts of the system (hardware and software) do not come together again until late in the project, with no time left to resolve unexpected integration difficulties.

SYSTEM INTEGRATION SOLUTIONS TODAY

One way to combat the problem of hardware and software integration is to formalize the interface between the development teams, but in the end this is a costly strategy. Meetings consume valuable time of capable senior developers on the project, throwing the budget and timeframes off track.

It's often possible to build engineering models for the software people to work with, but this is also time consuming for hardware engineering personnel. To be of any real value, such equipment must be maintained and upgraded constantly to reflect the behavior of the latest hardware as it evolves during the project. It is also extremely rare for there to be enough of these resources to go around for all the software developers who need access to it.

Another approach is to use a simulator. This allows a developer to load a simulation of the chosen processor together with a specification for the hardware part of the system written in VHDL or Verilog. The software part of the system is then loaded into the simulated processor and subsequently executed. When Input/Output (I/O) registers are written by the software the write event causes the appropriate signals to pass across the boundary into the hardware domain where the simulation of the hardware design processes them and provides the specified responses. While this goes a long way towards early detection of faults caused by interactions between hardware and software, it requires high performance host workstations and will not run in real-time, so it cannot be used to evaluate interactions between a system and the real world.

So you fire up a system containing hardware and software almost straight from the drawing board. Most likely, when the two first come together, the system doesn't work properly and you begin the tedious process of debugging.

Let us suppose the system does work? You're home

and dry, right? Well, not quite. There's an annoying reset that keeps randomly occurring and no one can localize it. To further the frustration, the system is just too slow. By knowing this in the beginning you could have chosen the next larger Gate Array and transferred the Fourier Synthesis functionality to hardware, right? Now, it is way too late. The amount of rework needed to transfer that much function to hardware is uneconomic.... Wrong.

MODELING THE PROBLEM

New hardware/software co-design techniques are emerging which will leave this kind of dilemma in the past. In this new process, the system is specified graphically, in full detail, without prior knowledge of which parts will be carried out by hardware and which by software. This specification is sufficiently precise enough that it can be formally executed. The required system functionality can thus be fully verified independent of hardware or software. As a separate step in the process, the hardware/software partition is chosen and the co-design platform then automatically generates the software code and hardware specifications, exactly as requested by the hardware and software design teams.

These process products are then passed to their respective conventional tool chains, compiled to machine object code in the case of the software, and to netlists in the case of the hardware. Inherent in the co-design platform generators is the ability to know how data should be passed back and forth between hardware and software. The required interfaces are generated automatically at the same time as the rest of the system. Now, when a performance problem is identified, the system can be repartitioned so that the bottleneck can be quickly reallocated to hardware. Alternatively, a hardware function that uses a vast gate count can be executed in software after repartitioning and regenerating the source code for the system (accomplished in a few hours, not in months).

This technique lends itself to deployment on multi-

processor systems as well. The system may be partitioned by processor just as easily as it is partitioned into hardware and software. In this way, it is possible to generate code for a system that includes mixed targets such as mainframes, micro- and signal processors, ASICs and the interfaces between them.

Other possibilities then present themselves. How about an entire product range with different throughput capabilities, all designed just once? With this type of hardware software co-design products with identical behavior, each with different performance characteristics, are produced simply by repartitioning the system. Remember, the behavior is specified at the system level without regard to its implementation. Recall too that the behavior can be fully tested before commitment to either hardware or software. Due to these two properties, any new partitioning created will very likely work correctly, straight off the drawing board.

Off the shelf hardware and operating systems can also be integrated with this kind of system development process. Because the code for the system is generated, companies need not become tied to one particular vendor. If a new product from another vendor appears on the market, the code generators can be reconfigured to produce code for the new hardware, device or operating system, often in a very short period of time.

TARGET INDEPENDENT SYSTEM MODELING

How does this new technology work? The key lies in the technology neutral way that system behavior is specified. To properly support hardware/software co-design, tools must allow the designer to think and work in a much more abstract way than at present.

This approach to co-design is currently supported by BridgePoint tools from Project Technology. In BridgePoint tools, the Unified Modeling Language (UML) is used for specifying the required system behavior. Where textual languages are utilized, they are made to be target language independent. Verifier, the

simulator component, is similarly technology independent and shows how the system components interact without making assumptions about whether those interactions will be implemented with a function call, circuit board jumper or a bus implemented in a gate array.

The UML graphical representations mentioned above are stored in a database defined by a published format, known as a metamodel. Quite separately from the description of the system functionality itself, designers work to specify how this database will be transformed by the Generator component into high level language source code, HLL (in the case of software), or into hardware definition language, HDL.

Where an interface exists between hardware and software, the Generator produces the appropriate code to write data to an I/O register (in the case of software). If generating hardware, HDL is produced when needed to declare the register together, of course, with the logic needed to perform the required processing on the written data. Is the I/O register too gate hungry? No problem. Configure the tool to generate the HDL necessary to address dual port RAM, or implement a serial interface or use any other interfacing technology you can think of.

The hardware and software design teams decide which technologies they will use to implement the system and the corresponding hardware/software interface. They meet only to agree how data and events will be exchanged across the interface. It is important to realize that they need only agree one protocol for all such transfers, as opposed to working case by case as conventional wisdom usually requires. There may, of course, be good reasons to have more than one way to transfer data or control. This is easily accomplished in the co-design process, so long as coherent rules can be stated which allow the Generator to select the appropriate interface protocol as it generates the code. This usually means stipulating reasons for design choices on previous projects. These reasons usually resolve to questions of latency, bandwidth or cost.

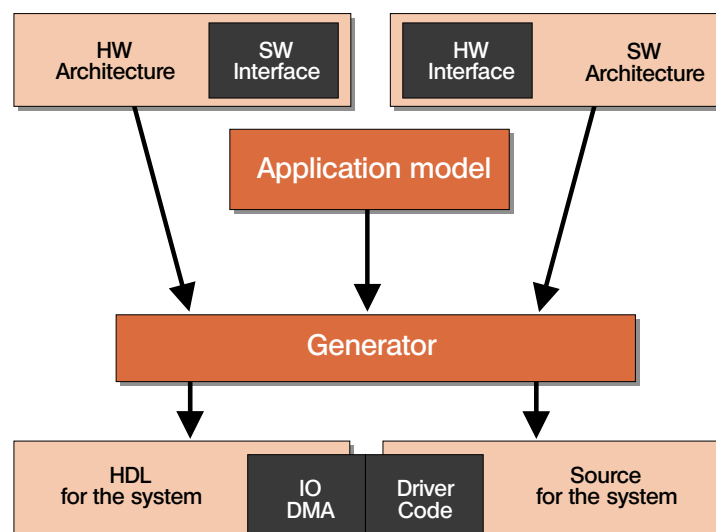


Figure 1.

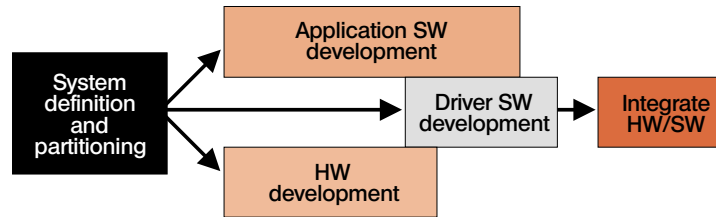


Figure 2.

Specifying the code necessary to implement the semantics of the symbols used to denote the system is quite independent of what is actually shown by those graphics. In other words, the configured generator will produce correct code for any system because they are application independent. This application independence allows HLL and HDL know-how to be captured for later reuse. It also allows companies such as Project Technology to provide commercial off the shelf Generator configurations, called Architectures, aimed at particular vendors' tool chains such as Mentor or Cadence, at particular programming languages such as 'C', 'C++' or Java and target popular RTOS's such as Embedded Linux and VxWorks. These architectures may be modified, by embedded developers themselves or by the supplier, to meet their own exacting requirements.

DON'T REINVENT THE WHEEL

In adopting this new process, existing tools, including state of the art tools like SeamLess, remain useful. The output of this new technology is high level source code, so debuggers, in-circuit emulators and the tool chains used to convert HDL's to silicon are used the same way they always have been. They are not superseded, just moved a little ways downstream.

In the future, co-design tools will provide much improved support for the process, including the ability to estimate important system characteristics before any hardware or software is generated. Already, the BridgePoint Verifier simulation tool has a simple resource allocation capability that helps an engineer predict performance in a software system. Soon, we can expect this capability to be extended to predict other characteristics of a mixed hardware/software product such as overall system latency, battery consumption, thermal characteristics, code space and data sizes, interface bottlenecks etc.

Co-design is a real and emerging technology using new, relatively low-cost tools, which promises reduced time to market, better portability and vendor independence. Combine this with improved risk management, reduced defect incidence and increased intellectual property capture and reuse and we have technology, which is catching up to the industry's demand.

NOTES

In the traditional hardware/software design process, system partitioning is carried out very early in the project. All downstream deliverables; from documentation to tests are inextricably coupled to this decision. This makes it extremely difficult to rework sub-optimal design decisions. The finish to start dependencies (broken arrows) means that there is little flexibility for decoupling the phases of the project and reducing time to market.

Using a co-design strategy, system partitioning is not necessary until late in the project. The application behavioral model cannot therefore be in any way coupled to the hardware/software partition. The partitioning is therefore easily iterated to find the optimum combination of hardware and software required for the system. The activities are very decoupled, so delays in one part of the project do not impact others. Finally, the Hardware and Software Architecture deliverables are Application independent, so these become reusable corporate assets ■

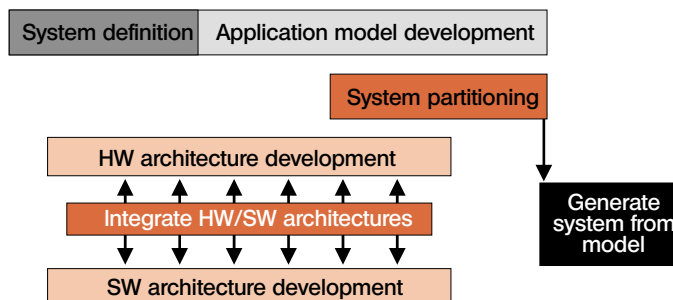


Figure 3.

Campbell McCausland began his interest in hardware and software at the University of Bristol in the United Kingdom where he built and programmed his own microcomputer system. He has accumulated over 15 years experience in the field of embedded systems development. He spent 10 years with GEC and Unisys in the United Kingdom before leading the Master of Science in Software Engineering at Napier University in Edinburgh, Scotland. After this time he joined Project Technology, initially in the UK as Technical Director of PT International, and later as Director of Research at PT's head office in Tucson, Arizona.