On Hardware and Hardware Models for Embedded Real-Time Systems

Jakob Engblom*
Dept. of Information Technology, Uppsala University
P.O. Box 325, SE-751 05 Uppsala, Sweden
jakob@docs.uu.se / http://www.docs.uu.se/~jakob

Abstract

When building an embedded real-time systems, the choice of hardware platform is very important to create an analyzable and predictable system. Also, the quality of the models of the hardware used in software tools is very important to the correctness of timing analysis and the integrity of the system.

In this paper, we discuss some of the aspects of how to build hardware models that are correct visavi the hardware, and how to select hardware that allows real-time systems to be constructed in a reliable fashion.

The purpose of this paper is to inspire some discussion regarding how real-time systems are designed and built and verified, from software, hardware, and tools standpoints.

Keywords: Embedded Systems, Real-Time Systems, Computer Architecture, Hardware Selection, Hardware Analysis, Hardware Modelling

1. Introduction

Thanks to the ever-advancing performance of embedded computer technology, an increasing number of systems are being built and designed where microprocessors and microcontrollers perform substantial controlling tasks. Vehicles, appliances, industrial plants, toys, mobile phones: embedded systems are everywhere around us. Most of these embedded systems have real-time constraints, and potentially dangerous effects if deadlines are missed or some other timing-related bugs manifest themselves.

When developing embedded real-time systems, designers and programmers rely on various forms of scheduling and timing analysis. At some point, all such analyses must account for the hardware used to obtain execution time information, and if the analysis method does not accurately reflect the hardware characteristics, the result is likely to be a bad analysis and potentially a bad system.

The choice of hardware, specifically the microprocessor or microcontroller to use, has a profound influence on the analyzability of a system. The timing behavior of a complex CPU core is very hard to understand (even without a cached memory system). Thus, systems have to be designed to facilitate analysis, from the hardware and up.

2. On the Quality of Hardware Models

In embedded systems development, *hardware models* (software running on the cross-development system) are used for most of the development work and analysis of a system. The quality of such models form a key part of the correctness argument for a system. The use of a bad hardware model (even for eminently predictable and analyzable hardware) will put the system integrity at risk.

Looking at the design flow for hardware models, several sources of errors can be identified. Figure 1 shows a schematic flow of the design and implementation work separating a CPU simulator (the most important type of hardware model) from the physical chips shipped by the manufacturer.

The *manual writing* is probably the biggest source of errors. The manual writers have to interpret the chip design, and this interpretation can be different from that of the hardware implementers. Simple typos can introduce more errors.

* This work is performed within the Advanced Software Technology (ASTEC, http://www.docs.uu.se/astec) competence center, supported by the Swedish National Innovation Systems' Administration (VINNOVA, http://www.vinnova.se). Jakob is an industrial PhD student at IAR Systems (http://www.iar.com) and Uppsala university, sharing his time between research and development work.

Presented at the IEEE Real-Time Embedded System Workshop, Dec. 3, 2001.

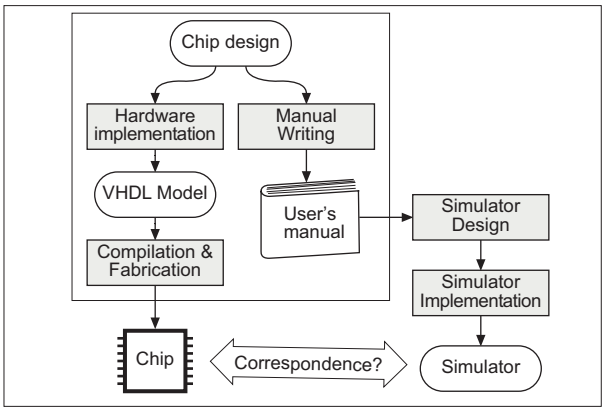

Figure 1. Workflow of Hardware Modeling

Often, many of the details needed to implement a simulator (or an optimizing compiler) are abstracted away or glossed over. Many manuals do not describe the detailed function of a CPU pipeline, but instead give simple approximate cycle counts for instructions (which is a grave oversimplification for a pipelined processor) [1, 3, 10]. Thus, the manuals typically have a rather tenuous relation to the chips they are supposed to document, which is a major problem for hardware modeling.

Even though hardware engineering is in much better shape than software engineering, bugs can be introduced in the *hardware implementation*. A famous example is the Pentium `FDIV` bug [20]. The presence of extensive errata lists for hardware further indicate that hardware does contain bugs which are not found until after products are shipping. It should be noted that bugs regarding the timing behavior of chips are quite common while functional bugs are comparatively rare, since instruction timing is usually not validated as extensively as functional correctness [12, 17].

The *hardware compilation and fabrication* do not usually generate user-visible bugs.

The simulator implementation also has some bug sources. In the *simulator design* step, it is possible to introduce errors by misinterpreting the manuals and ignoring important features in the model. During *simulator implementation*, errors are introduced as the designed simulation model is converted into actual program code.

In summary, the "timing correct" hardware models we use in real-time development and research are several bug-inducing steps away from the actual hardware, which makes the use of such models quite questionable. It is necessary to find a way to *validate* the hardware models used when building real-time systems. To investigate this, we look at some work performed in validating CPU simulators.

Looking at Figure 1, it seems that it would be possible to shortcut the process by using the VHDL code in some manner. Unfortunately, VHDL simulators are orders of magnitude slower than a regular cycle-correct simulators, and automatic abstraction from VHDL to a cycle-level simulator is not feasible.

2.1. Experience Report 1

Black and Shen [4] report their experience in validating a model of a PowerPC 604 processor. The method employed was to execute many test cases (small programs) on both their simulator and a real CPU. The *performance counters* of the PowerPC were used to check the correctness of the model. For such a complex chip, this is a reasonable approach. As a sanity check, they also executed a number of larger programs and compared the cycle counts on the simulator and the hardware.

The results are interesting, since they never manage to get a perfect correspondence. Fixing an error in the model sometimes revealed other errors that had been masked by the first, and the total error could thus increase for some fixes. However, over time, the accuracy of the model increased, both for the performance counter agreements and the cycle counts for the tested larger programs. For their final model, about 20 percent of the individual small tests failed to be within one clock cycle of the hardware timing, while the large test programs showed an error between 2 and 10 percent. The errors were both positive and negative, indicating overestimates and underestimates of the execution time.

Their conclusion is that systematic validation against hardware is necessary to build a performance model that users can have confidence in using, and that achieving a reliable and precise model requires a large development effort. The complexity of current high-end microprocessors makes the development of reliable hardware models a "great challenge".

2.2. Experience Report 2

Gibson et al. investigated the quality of the models used in the research and development of the Stanford FLASH multiprocessor [8]. The FLASH project employed several different simulators during the development of the system, and collected a large amount of data on how well the simulators and the hardware correlated. Their testing was initially based on running large multi-processor numerical applications and comparing the resulting execution times. Smaller benchmarks were introduced to help pinpoint specific performance mismatches later in the process.

The CPU models used were all based on generic simulation packages like Mipsy and MXS, which were given

parameters to closely emulate the MIPS R10000 CPUs used. However, these models were not able to capture all relevant details of the CPU, giving execution time results very different from the hardware for key parts of their code like TLB miss handlers. This is claimed to be a general problem with generic CPU simulators: they are not sufficiently configurable to model all the quirky cases of the real hardware.

The conclusions for simulator construction is that simulation technology is barely able to keep up with the increased complexity of modern computer systems, and that hardware to compare with is necessary in order to build a good model.

2.3. Experience Report 3

Montán [16], working within our project, validated a trace-driven simulator for the NEC V850E [19], used in our WCET research, by comparing execution times with the real hardware. The NEC V850E is a simple scalar design without caches, typical for today's 32-bit RISC microcontrollers. The scope of the validation effort was limited to the pipeline core. The hardware used was a V850E emulator, which, according to NEC, has exactly the same core as used in the normal chips, the only difference being in the packaging and access to internal signals.

The experiments used both short test cases of a few instructions each, and some larger programs. For each test, cycle counts were compared, and when a difference was found, the test was analyzed in detail, trying to find the fault in the simulator that caused the discrepancy. Many errors in the manuals were discovered when a clock cycle count deviated from the expected. To give a feeling for the types of errors found in this experiment, a short selection is given in Figure 2.

In the end, for test cases not affected by the hardware bug, the small test cases passed without error. Errors for all but one of the test programs were down to about 10 cycles, which was attributed to measurement differences. One test program indicated some hardware feature that was not correctly modeled. Overall, the resulting simulator offers a very good correspondence with the hardware.

3. On Cache Modeling

Another important aspect of a CPU is its cache system. Building a model of a cache system is usually quite simple, but analyzing the behavior in execution time analysis is more complex.

Research in worst-case execution time analysis have addressed the analysis of single-level direct-mapped caches and set-associative caches with least-recently-used (LRU) replacement [7, 14, 21, 18]. Separate

Manual Writing
The multiply instruction sometimes writes two registers, which causes a delay of one cycle (probably the register bank has only one write port). This was not documented.
Some forms of jump instructions were documented as being one cycle faster than they actually were.
Simulator Design
The V850E has a main pipeline and a secondary pipeline. It was possible for instructions to continue issuing to the secondary pipeline even if the main pipeline was busy, which is not possible in the hardware.
One instruction had been forgotten in the simulator implementation.
Simulator Implementation
Signed and unsigned division has a one clock-cycle difference in execution time, which was implemented the wrong way around.
Some bit-manipulation instructions did not keep certain resources locked for their entire execution, letting other instructions slip by, eventually leading to internally inconsistent states and thus a crash of the simulator.
Hardware Bugs
There is no interlock on the hardware for successive load instructions writing the same register, making it possible to destroy data in certain conditions, and leading to faster execution of some scenarios. This condition has been reported as a hardware bug by NEC, and was not modeled in the simulator.

Figure 2. Selection of Bugs

data and instruction caches makes the analysis problem tractable; unified caches are much harder [6]. Also note that multiple-level cache hierarchies, also popular on current desktop machines and servers, make the analysis problem harder.

The cache analysis methods are efficient and effective, but unfortunately, many embedded architectures have cache systems that do not fulfill the assumptions used in the cache analysis research.

LRU replacement for a set-associative cache is conceptually simple and effective, but it is not practically implementable in hardware when going above associativity four, due the memory and logic required to track the latest access date for all cache lines in a set larger than four lines [9]. Thus, many set-associative cache systems used in embedded systems use other replacement policies. The *Intel X-Scale* architecture ("StrongARM 2") has a *round-robin* cache replacement scheme for its 32-way set-associative cache [11]. The *ARM 7* cores use a 4-way unified cache with *random* replacement policy, presenting a really hard problem for WCET analysis (and predictable performance) [1]. Analysis of random and round-robin replacement schemes are bound to be very pessimistic, since any access can invalidate any other line in the same set [6].

A related issue is that for complex, superscalar, processors with dynamic out-of-order dispatch, cache misses might cause shorter execution times than cache hits, which invalidates the basic assumptions behind worst-case cache analysis [15]. This points to the un-

suitability of dynamically scheduled processors in hard real-time systems.

4. Summary and Recommendations

This paper has presented an overview of work performed in validating hardware models against the hardware being modeled, and a discussion on the properties of cache memories and their models. The purpose is twofold: to point out that it is very difficult to build correct hardware models and to inspire some discussion in the hardware design of real-time systems.

We believe that a real-time system has to be designed from the ground up for predictability in timing. Selection of processor and memory system design is very important to produce a reliable system. Just selecting a "very fast" processor is not enough, since real-time system design is not about being fast but about being predictable.

Without a predictable hardware platform and correct timing analysis tools, scheduling theory and analysis is rather useless, since the input to a scheduling algorithm has to be correct for the generated schedule to be correct.

Based on the information presented above, we have the following concrete advice to offer developers of embedded real-time systems:

- The documentation for a processor should be assumed to contain errors or incomplete information, and building a correct hardware model requires validation against real hardware. Only the hardware represents correct information about itself (and in many circumstances, the VHDL code).
- Accurate hardware models are possible to build for simple pipelined processor architectures, but not for advanced superscalars. Thus, simple pipeline architectures are preferable for embedded real-time systems.
- Hardware should be selected for predictable performance, modelability, and analyzability. In the DSP field for example, predictability has been maintained even as performance has been pushed higher [5].
- Cache systems should be kept simple and predictable, since this both enables analysis and makes the worst-case behavior less bad. Replacing the cache by programmer-controlled fast SRAM is often a very good option to keep performance and increase predictability, at some cost in program complexity [2, 13].

References

[1] ARM Ltd. *ARM 720T Data Sheet*, 3^{rd} edition, September 2000. Document no. DDI 0192A.

[2] ARM Ltd. *ARM9E-S Flyer*, 2001. Document no. D0I 00798A.

[3] ARM Ltd. *ARM 9TDMI Technical Reference Manual*, 3^{rd} edition, March 2000. Document no. DDI 0180A.

[4] Bryan Black and John Paul Shen. Calibration of Microprocessor Performance Models. *IEEE Computer*, pages 59–65, May 1998.

[5] Jennifer Eyre. The Digital Signal Processor Derby. *IEEE Spectrum*, 38, June 2001.

[6] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and Precise WCET Determination for a Real-Life Processor. In *Proc. First International Workshop on Embedded Software (EMSOFT 2001), LNCS 2211*, October 2001.

[7] C. Ferdinand, F. Martin, and R. Wilhelm. Applying compiler techniques to cache behavior prediction. In *Proc. ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems (LCT-RTS'97)*, 1997.

[8] Jeff Gibson, Robert Kunz, David Ofelt, Mark Horowitz, John Hennessy, and Mark Heinrich. FLASH vs. (Simulated) FLASH: Closing the Simulation Loop. In *Proc. 9^{th} ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS00)*, November 2000.

[9] Jim Handy. *The Cache Memory Book*. Academic Press, 2^{nd} edition, 1998.

[10] Infineon. *Instruction Set Manual for the C166 Family*, 2^{nd} edition, March 2001.

[11] Intel. *Intel XScale Core Developer's Manual*, December 2000.

[12] Intel. *Intel Pentium 4 Processor Specification Update*, April 2001.

[13] Philip Koopman. Perils of the PC Cache. *Embedded Systems Programming*, pages 26–34, May 1993.

[14] S.-S. Lim, Y. H. Bae, C. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Ki. An accurate worst-case timing analysis for risc processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, July 1995.

[15] T. Lundqvist and P. Stenström. Timing anomalies in dynamically scheduled microprocessors. In *Proc. 20^{th} IEEE Real-Time Systems Symposium (RTSS'99)*, December 1999.

[16] Sven Montán. Validation of Cycle-Accurate CPU Simulator against Actual Hardware. Master's thesis, Dept. of Information Technology, Uppsala University, 2000. Technical Report 2001-007, `http://www.it.uu.se/research/reports/2001-007/`.

[17] Motorola Inc. *MPC850 Family Device Errata Reference*, February 2001.

[18] Frank Mueller. Timing Analysis for Instruction Caches. *Real-Time Systems Journal*, 18(2/3):209–239, May 2000.

[19] NEC Corporation. *V850E/MS1 32/16-bit Single Chip Microcontroller: Architecture*, 3^{rd} edition, January 1999. Document no. U12197EJ3V0UM00.

[20] Dick Price. Pentium FDIV flaw–lessons learned. *IEEE Micro*, April 1995.

[21] Friedhelm Stappert. Predicting pipelining and caching behaviour of hard real-time programs. In *Proc. of the 9^{th} Euromicro Workshop of Real-Time Systems*, June 1997.

Real-time hardware platform examples. â€¢ Desktop PC with real-time OS (RTOS). â€“ as long as the hardware meets certain system requirements. â€¢ 8-, 16-, and 32-bit microprocessors â€¢ PXI with real-time controller. â€“ often used for high-performance real-time systems such as hardware-in-the-loop testing. â€¢ NI FPGA â€¢ NI CompactRIO â€¢ NI Single-Board RIO â€¢ NI CompactVision â€¢ Industrial PCs/Controllers â€¢ NI Compact FieldPoint.Â â€¢ Developing the LabVIEW Real-Time application for deterministic floating point analysis and control as well as communication with a networked host computer. â€¢ Developing the LabVIEW for Windows application for graphical. user interfaces, supervisory control and data logging. NI CompactRIO Reconfigurable Embedded System. Embedded systems within medical equipment are often powered by industrial computers.[8].Â Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs. Embedded systems are not always standalone devices. Many embedded systems consist of small parts within a larger device that serves a more general purpose.Â They run with limited computer hardware resources: little memory, small or non-existent keyboard or screen. User interface[edit].