Virtual Platform Environment for the Bring Up and Test of a Secure Many-Core RTOS

Atsushi Shinbo and Shuzo Tanaka
eSOL TRINITY Co., Ltd.
Tokyo, Japan
s-tanaka@esol-trinity.co.jp

Duncan Graham and Larry Lapides
Imperas Software Ltd.
Oxford, England
larryl@imperas.com

Masaki Gondo
eSOL Co., Ltd.
Tokyo, Japan

***Abstract* — The increasing numbers of cores in the individual SoCs, the move to multiple SoCs in Electronic Control Units (ECUs) and the increase in complexity of software for automotive electronics has led to the need for many-core support for real time operating systems (RTOSs). In addition, security requirements on the systems are directly flowed down to security requirements on the RTOS. With this increasing complexity of hardware, software and security requirements, the challenge to bring up and test the RTOS and basic software (BSW) has become much difficult.**

This paper reports on the use of a virtual platform (software simulation) based environment for the bring up and testing of a secure, many-core RTOS on an ECU. The RTOS is the eMCOS RTOS from eSOL, the hardware represented in the virtual platform is two Renesas RH850F1H devices (SoCs), and the virtual platform tools are from Imperas. In addition to the RTOS, AUTOSAR and a Real Time Environment (RTE) were brought up in the virtual environment.

This paper will describe the process of building the virtual platform models, and the bring up and testing of eMCOS, AUTOSAR and the RTE. Overall simulation performance of about 200 million instructions per second (MIPS) was achieved. Key, unique and innovative features of this approach include platform-centric debugging, non-intrusive tools and a powerful set of APIs for modular tool development.

Keywords — RTOS, virtual platform, many-core, security, software simulation, virtual prototype, automotive, AUTOSAR, real time environment

I. Introduction

In recent years, automotive electronics hardware has seen a significant increase in complexity due to the increased number of processor cores in the individual SoCs and the increased number of SoCs in Electronic Control Units (ECUs). The software for these ECUs has seen an even larger increase in complexity, due to the increased complexity of the hardware, the increasing number of system scenarios that must be satisfied and the increasing safety requirements for automotive systems.

Part of the increased complexity of the software is embodied in the use of many-core real time operating systems (RTOSs) on these SoCs and ECUs. On top of the RTOS is usually AUTOSAR, plus a run time environment (RTE). While the use of a software stack including a many-core RTOS, AUTOSAR and RTE brings a level of complexity, one of the benefits of this stack is reduced complexity for software engineers using the environment.

Also, one other requirement has been added to the system: security. Security presents an additional challenge for software development, since secure software by its very nature should not be visible on the hardware, so debug using normal hardware-based techniques is exceedingly difficult.

Virtual platforms (software simulation) do not have the restrictions in debug visibility that hardware-based platforms have. Virtual platforms for software development, debug and test have additional advantages over hardware platforms such as early availability, increased controllability and observability, ease of automation of the development environment and ease of replication of the development environment.

This paper reports on the use of a virtual platform (software simulation) based environment for the bring up and testing of a secure, many-core RTOS, AUTOSAR and RTE on an ECU. The advantages of using virtual platforms are elaborated, followed by a description of the process used to build the virtual platform. The simulator engine and debug and test tools are described. Finally, the case study is completed with details of bring up methodology, including simulation results.

II. Virtual Platforms for Software Development

The standard methodology for embedded software development is to use some type of hardware as the development platform. This could be a previous generation of the SoC, a hardware emulator, a FPGA prototype, or some other type of development board. These platforms have the benefit of cycle accurate execution of the software, which is needed for some software development, especially that software that contains real time execution requirements.

While there are advantages to using a hardware-based development methodology, there are also disadvantages. These disadvantages include

- Limited physical system availability
- Limited external test access (controllability)
- Limited internal visibility

Also, depending on the exact hardware platform being used, it can be months from project start until a hardware platform is available for the software engineering team.

Instruction accurate virtual platforms are not cycle or timing accurate. However, these virtual platforms do have significant advantages:

- Early system availability
- Full controllability of platform both from external ports and internal nodes
- Full visibility into platform
- Easy to replicate platform and test environment to support automated testing on compute farms

Also, while instruction accurate virtual platforms do not have the timing information, the majority (75 – 90%) of bugs in these applications are purely functional.

The full visibility feature of virtual platforms is key to development of secure software. Virtual platforms can provide access to secure parts of the system that are not visible, deliberately, to any of the hardware platforms. This visibility is important for conventional debug, however, it is also important for other virtual platform tools including such tools as code coverage, memory monitors and OS-aware tools. A further advantage of these tools is that the tools are completely non-intrusive: no modification or instrumentation of the source code, or compiling of a debug kernel, is necessary for the use of these tools in the virtual platform environment.

Looking at the complete software development methodology, virtual platforms should be used, exclusively, early in the development process. However, the virtual platforms, due to the visibility and controllability and the software development tools available, can continue to be used and add value throughout the duration of the software project. The hardware platforms, including especially the final hardware, can be used when they become available, with hardware based testing and virtual platform based testing providing complementary benefits to software engineers.

III. Building the Virtual Platform

The virtual platform is a set of instruction accurate models that reflect the hardware on which the software will execute. The virtual platform could be a single SoC, multiple SoCs, a board, a system; it is a virtual platform, and there are no physical limitations. This is the traditional instruction set simulator concept, now extended to a complete platform, with a simulator that supports a full set of software development tools.

These instruction accurate models are built with a set of APIs which are supported by the virtual platform simulator. The running of the virtual platform then just involves the execution of the models with the simulator linked in. There are different APIs used for the processor core models, the non-processor models (called "peripheral models" here) and the platform connections including memory and busses. Components in the virtual platform are connected just as in the hardware, and the memory map is the same as for the hardware. Also, peripheral models, e.g. of CAN, ethernet or USB components, can be connected to the real world via the appropriate port on the host x86 workstation.

These models should have only as much information in the models as the software developers need. There is no timing information needed in these models, and any unnecessary information will slow the virtual platform performance.

Just as with the models themselves, the platform should only have the models that are necessary for the specific task that the virtual platform is being used for. Virtual platform based development does not replace hardware based testing, so models of components needed for hardware based testing should not be included in the virtual platform, except at an instruction accurate level needed for execution of the software.

A key point about virtual platforms is that the combination of virtual platform models plus simulator executes exactly the same binary software stack as will eventually run on the hardware. No compiling for the host x86 workstation; if the system uses an ARM or Renesas processor, the same cross compilation tool chain and flow are used to create the ARM or Renesas binary executables to run on the virtual platform or on the hardware.

Open Virtual Platforms (OVP) [1] C language models and modeling APIs were used to build the virtual platform of the Renesas RH850F1H devices and virtual ECU used here because of the availability of the processor core models in the OVP library, the performance of the processor core models, the ease of use of model development with the APIs and the tool power that is enabled when running the platform with the Imperas simulators [2].

OVP includes a library of models, APIs for developing models and platforms and a reference simulator (OVPsim) for executing those virtual platforms. The model library includes over 200 processor models, over 300 peripheral models and over 50 reference platforms. The reference platforms are Extendable Platform Kits (EPKs) of devices and development boards with software – bare metal or an operating system – running on the virtual platform. These EPKs give a user a known good starting point, from which extension of the virtual platform of the EPK to meet the specific project goals can be easily accomplished.

The RH850F1H devices include a two separate RH850G3M processors and assorted peripheral components.

For this project, instruction accurate models of the RH850F1H devices were built, using the RH850G3 Fast Processor Model from the Open Virtual Platforms (OVP) Library, and using the OVP APIs to build the peripheral models and the virtual platform. A block diagram of the RH850F1H is given in Fig. 1. The RH850F1H virtual platform is shown in Figure 2. Note that there is only the minimal set of models needed to run the RTOS, AUTOSAR and RTE. The single virtual platform of the RH850F1H can be considered a "pseudo-ECU".

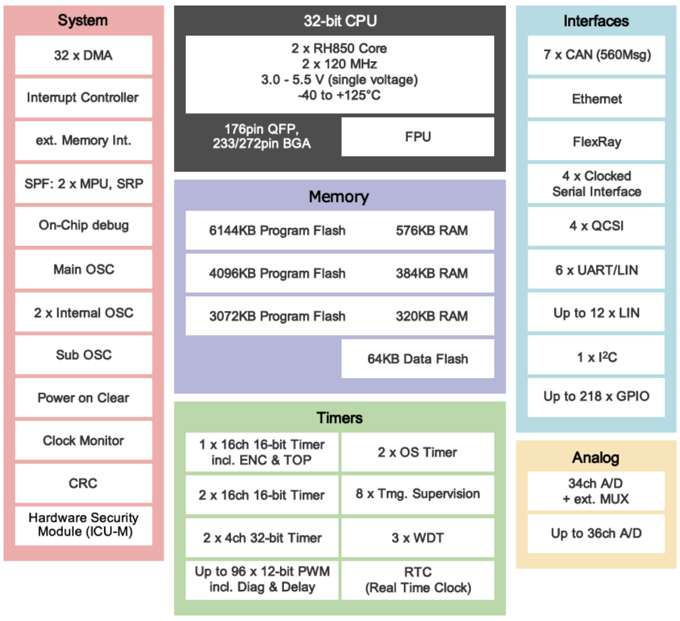

Fig. 1. RH850F1H block diagram. (Courtesy Renesas Electronics.)

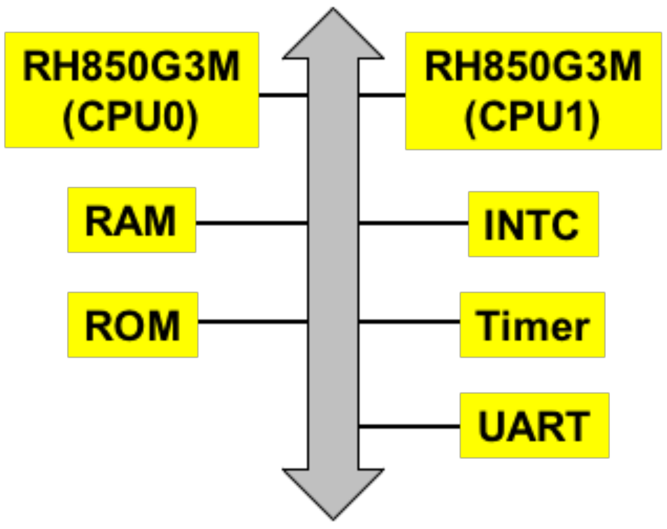

Fig. 2. RH850F1H virtual platform block diagram. Note that only the minimal peripheral models needed to boot the software stack are used.

For this work, focused on the many-core eMCOS RTOS / AUTOSAR / RTE, running on multiple processors and ECUs, two instances of the RH850F1H pseudo-ECU, connected via a two UARTs, were implemented in the virtual platform to represent the two ECUs. While in the real hardware the ECUs are connected via a CAN bus, in this virtual platform the model of the CAN peripheral is not needed, since the focus of the testing was just to verify the communication content, not the protocol. This is shown in Fig. 3.

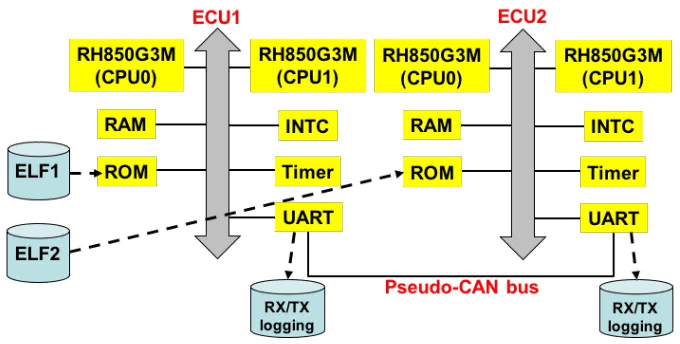

Fig. 3. Virtual platform containing two RH850F1H pseudo-ECUs.

IV. Debug and Test of the Software Stack

A. *eMCOS RTOS*

The eMCOS many-core RTOS [3] has been available from eSOL for several years. Originally developed as a general purpose RTOS, it was enhanced to add AUTOSAR Classic Platform (CP) and the RTE for automotive electronics. RTE provides an API to the AUTOSAR CP application called SW-C (Software Components). It provides communication between SW-Cs on the same ECU, and also between SW-Cs that reside in different ECUs, via CAN for example.

Recently, with security requirements imposed on automotive systems, eMCOS has been enhanced to support secure software execution.

The eMCOS RTOS uses a distributed microkernel architecture that is different from any existing single-core or multi-core RTOS. This enables it to make the best use of many-core processor hardware, because it does not depend on cache coherency mechanisms. A microkernel is allocated to every core to offer basic services, including inter-core message passing, local thread scheduling and thread management. Because eMCOS supports POSIX and AUTOSAR APIs, developers can reuse their existing software assets.

eMCOS also uses the MPUs in the target hardware to allow users to designate secure memory regions. The eMCOS RTOS operation is shown in Fig. 4.

B. *Simulator and Tools*

The simulator used to run the virtual platform and test the eMCOS / AUTOSAR / RTE stack was the S*DEV product from Imperas. S*DEV enables the simulation of a virtual platform with multiple homogeneous processors, plus connection to GDB for software debug on any of the processors. S*DEV typically has performance of 200-500 MIPS, with that performance split equally between the processors in the virtual platform. For example, for a virtual platform with four processors, each processor would run at 100 MIPS.

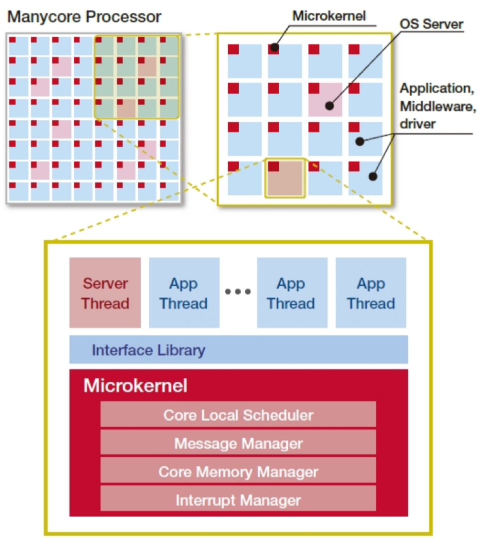

Fig. 4. eMCOS RTOS structure.

A block diagram of the test set up is shown in Fig. 5.

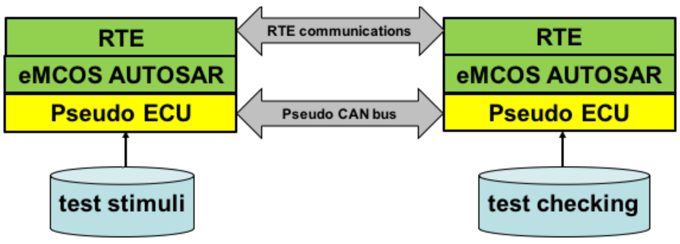

Fig. 5. Block diagram of the test set up.

C. Results

Goals for the use of the virtual platform based test environment included

- Verification of eMCOS/AUTOSAR/RTE
- Enable Continuous Integration (CI) flow
- Enable multiple teams to use the same test environment

The virtual platform environment was brought up in stages, with the initial stage consisting of developing the single pseudo-ECU and bringing up eMCOS without AUTOSAR on the single pseudo-ECU. This effort took less than 2 weeks of engineering effort.

From this initial milestone, the bring up of the full virtual platform environment, as shown in Fig. 5, proceeded in a straightforward manner, and took less than 2 months.

Overall simulation performance of greater than 200 million instructions per second (MIPS) was achieved, or about 50 MIPS per core. This performance was somewhat dependent upon the specific test cases. Performance of at least 200 MIPS was critical because of the large test cases, and large number of tests (nearly 5,000) being used. Even so, running the full test suite takes nearly 3 days.

The virtual platform, as a software executable, was easy to replicate and deliver to additional engineering teams located at different sites.

The visibility of the virtual platform was key. Using the virtual platform enabled catching bugs that would have been found much later in the test cycle, if at all. This visibility also enabled the debug of secure elements of the software stack.

Also, the S*DEV simulator can be controlled both by interactive debug tools and from the command line. This command line control, via a rich set of C commands, allowed easy integration of the simulation environment into the Continuous Integration flow.

While it is difficult to calculate the time saved in this project due to the use of the virtual platform, the ability to run tests automatically overnight coupled with the performance of the simulator enabled schedule savings of months on this project.

V. Conclusions

Using the virtual platform as a complementary tool to hardware-based testing accelerated the overall software testing task by months. More comprehensive testing was achieved, finding bugs quicker, and enabling debug of secure software.

Using the virtual platform is not a cure-all. A test plan is necessary, separating out tests for each of the platforms. The virtual platform should be used for test tasks where significant ROI can be achieved, due to the strengths of the virtual platform.

Further work is desired, including extending the use of the virtual platform to use additional simulation tools such as code coverage and fault injection.

References

[1] Open Virtual Platforms (OVP), www.OVPworld.org
[2] Imperas Software Ltd., www.imperas.com
[3] eSOL Co. Ltd., www.esol.co.jp