

Embedded Systems Verification with FPGA-Enhanced In-Circuit Emulator

M. Meerwein, C. Baumgartner, T. Wieja
Robert Bosch GmbH
Automotive Equipment Division 8
72703 Reutlingen, Germany
Matthias.Meerwein@de.bosch.com

W. Glauert
Institute for Computer Aided Circuit Design
University of Erlangen-Nuremberg
91052 Erlangen, Germany
whg@lrs.e-technik.uni-erlangen.de

Abstract

In this paper we present a novel coverification concept for embedded microcontrollers that satisfies industrial requirements. Based on a commercially available CPU in-circuit emulator coupled with FPGA boards, it verifies the correctness of an implementation in terms of function and timing within a real-world environment.

Using our system, the software engineer can write, test and optimize programs for a chip that is not yet physically existent. In addition the system is used to obtain software module characterization data required for system partitioning. Its ability to integrate analog circuitry enables verification of the complete system-on-chip. Our methodology is fully integrated into the ASIC design flow providing ease of use and a high level of verification accuracy.

1. Introduction

At the end of the partitioning and coding phase in the design cycle of an embedded system, the chosen implementation needs to be verified prior to its implementation. While this is easy for the hardware partition through VHDL or gate level simulation, the verification of the software part and the HW/SW interface suffer from the absence of the hardware platform at this time.

We are addressing automotive applications like intelligent bus-linked sensors or actuators and small control units. Being mostly control-dominated, they have to meet strict real-time and safety requirements and thus demand for thorough in-system verification. Since these systems are cost-critical high-volume products, substantial effort is spent on software optimization to keep hardware cost low.

Cosimulation using a functional representation of the CPU (e.g. VHDL model or instruction set simulator) is a viable option, but is usually impaired by low performance resulting in very long simulation runs. Furthermore, modelling the physical system environment – which is essential for systems running under stringent real-time requirements such as automotive control applications – is troublesome and often inaccurate. Reconfigurable emulation machines

accelerate the verification dramatically but still don't adequately address the real-world interaction of the circuit.

Furthermore, it is necessary to accurately characterize real-time behavior and cost (memory usage) of software modules that will be reused in a new design iteration or in another project. This data can be used for timing estimation during interactive partitioning or it may govern an automated hardware/software partitioning algorithm.

2. Related Work

A large number of hardware/software coverification approaches were described in the literature. Codesign frameworks like POLIS [1], SpecSyn [2], CoWare [5] or COSSAP offer cosimulation. The VHDL-C cosimulation approach presented in [7] performs efficient functional verification but lacks exact timing prediction and can't include assembly language modules. More advanced methods like those presented by Liu [4], Tabbara [8] and Lajolo [9] increase accuracy and efficiency of this concept by integrating an instruction set simulator and by generating models describing the behavior of the software partition either from the partitioned system specification or from a modified C compiler. Coverification schemes based on reconfigurable logic were presented for example in [3] and [6].

But even with these concepts, the handling of the software part is a complex task without the download and execution control of an in-circuit emulator. After compiling and linking, the binary code must be programmed into an EPROM which is inserted into a socket on the verification board. This procedure must be repeated for every software modification. Debugging and optimization are especially inconvenient since the absence of the emulators monitoring features leave logic analyzer and storage oscilloscope as the only means to observe what the program is doing.

3. The Coverification Concept

In our codesign suite, the coverification portion serves three purposes:

- Checking for correctness both in terms of operation and real-time reactive behavior.

- In some cases, software modules can't be synthesized but must rather be hand-written. This implies the need to write and debug code portions before the associated hardware is available.
- The characteristic parameters (execution timing, ROM and RAM usage) of software modules that have been synthesized or hand-written have to be evaluated. This profiling data is fed back into the partitioning system for high-accuracy prediction of the timing behavior (figure 4).

The presented coverification methodology solves these requirements. Based on an in-circuit emulator for the CPU coupled with FPGA boards it provides a pin-to-pin exchangeable, functionally exact model of the ASIC. It offers a significantly higher confidence level than simulation due to its closeness to the final implementation. Even if not running at the same speed as the ASIC it is still orders of magnitude faster than simulation. A novelty is the ability to fully integrate embedded analog portions of the ASIC into the coverification flow. By offering the program execution control and observation features of the in-circuit emulator, our concept facilitates development, debugging, test and optimization of the software part.

Subsequently we describe the in-circuit coverification methodology consisting of two interacting parts: A hardware portion that models the later ASIC and its integration into the ASIC design flow.

3.1. The Hardware Platform

Our coverification hardware (figure 1) consists of an in-circuit emulator connected to FPGA boards. This is viable and beneficial for a standardized CPU architecture that is not modified or generated to suit the application.

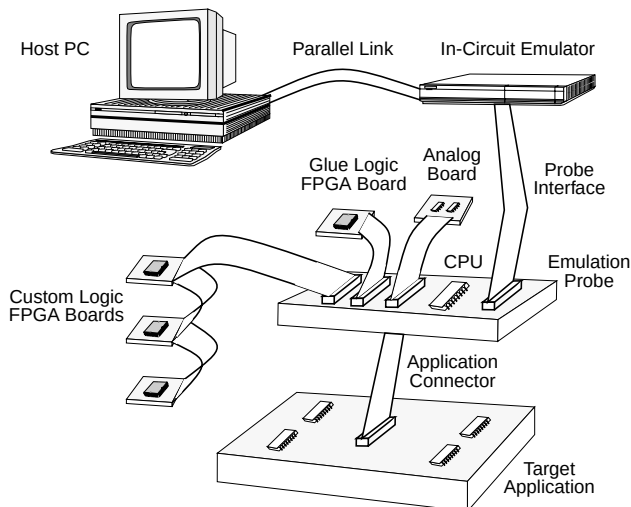


Figure 1: The hardware for coverification

Using the in-circuit emulator instead of mapping the CPU core on FPGAs saves their resources for the applica-

tion specific hardware components and extends the range of applications to software development and profiling.

The emulator part is commercially available from the CPU core vendor and provides the processor functionality while facilitating software development by offering source level debugging, breakpoints, register examination / manipulation and execution time profiling. An industry standard personal computer runs the user interface software.

Originally, the emulator is connected to a vendor-supplied probe board containing the microcontroller plus some support circuitry. However, if this microcontroller chip includes application-specific modules, it is nonexistent early in the design process when coverification is required. This problem is overcome by our emulation probe (figure 2).

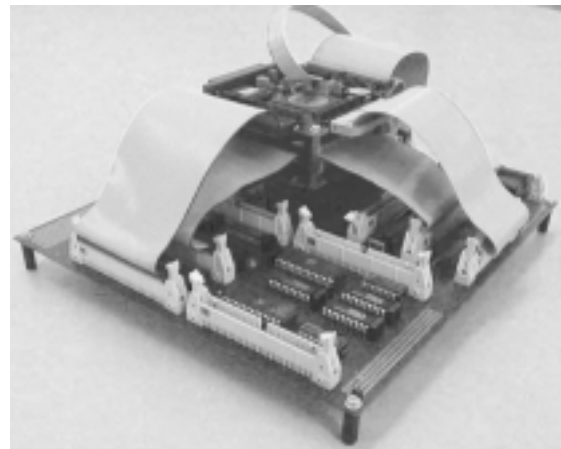


Figure 2: Emulation probe with FPGA boards

The emulation probe assembly consists of the following components:

- A microcontroller based on the same CPU core as the target system. It is running in an "Emulation Mode" in which the CPU core including its busses and control signals is accessible from outside. Its on-chip peripherals are accessible but can be disabled to free their address space and interrupt lines.
- Optional on-board RAM (2K) and ROM (32K) to allow the probe to operate without being connected to the in-circuit emulator. This feature is useful in our application domain since it enables convenient in-vehicle testing when the software has reached a mature state and emulator execution control is no longer needed.
- One or more reconfigurable FPGA boards containing the application-specific hardware modules. They are connected to the address, data and control busses of the CPU. The hardware components are accessible for communication with the software partition by mapping registers into the CPU address space.
- One FPGA board with glue logic. It includes address decoding, interrupt request handling, bus interface logic, sys-

tem clock selection and reset sequencing. Parts of this logic (e.g. address decoding, interrupt vector generation) must match the custom logic mapped onto the other FPGAs and hence be implemented in a reconfigurable device. This component also contains the control logic for the in-circuit emulator interface (the two empty pin headers visible in the foreground of figure 2) of the probe board.

- One (optional) analog board where analog or mixed-signal parts of the later ASIC like A/D or D/A converters can be modeled using standard components.

3.2. Extending the Design Flow

Closeness of the FPGA-modeled hardware partition to the fabricated ASIC is ensured by embedding the FPGA coverification system into the ASIC design flow as depicted in figure 3. The following files serve as starting point:

- A VHDL file describing the hardware partition of the system. This code is either generated by POLIS from the partitioned specification or hand-coded or checked out from a module database.
- The address mapping configuration file specifying the locations of the registers for hardware/software communication in the CPU address space.
- The VHDL specification of the glue logic required to interface the in-circuit emulator hardware with the CPU core and the custom logic FPGA boards. This file is invariable and does not depend on the two other files.

For FPGA synthesis (coverification), the address mapping configuration file is parsed by a code generator producing a VHDL description of the address decoding logic as its output. The generated address decoder becomes part of the glue logic FPGA. The VHDL code of the hardware partition is also synthesized and mapped onto the custom logic FPGAs. To stay as close as possible to the ASIC design flow, we do not use the VHDL front-end of the FPGA design software (MaxPlusII). Instead we synthesize with Synopsys Design Compiler (using appropriate libraries) and pass the resulting gate level netlist to MaxPlusII.

Depending on the size of the hardware partition, up to fourteen FPGAs can be used in this process. The FPGA mapping software supports this by automatically distributing large modules over several FPGAs. The pinout of the devices is a-priori fixed by the probe board design which limits the utilization to about 50%. Hence, the total hardware partition size is bounded to approximately 350 kgates with our current FPGA technology (Altera FLEX10K50).

When the system is to be mapped onto the target ASIC technology, the address decoder is also generated from its textual description file. However, it is now synthesized together with the hardware partition VHDL code to form the custom specific part of the ASIC.

Both the FPGA and ASIC design flows are very similar and use the same source files. None of them has to be mod-

ified manually during the FPGA→ASIC transition. This minimizes the functional differences between the in-circuit validation system and the system-on-a-chip implementation yielding a high level of verification accuracy.

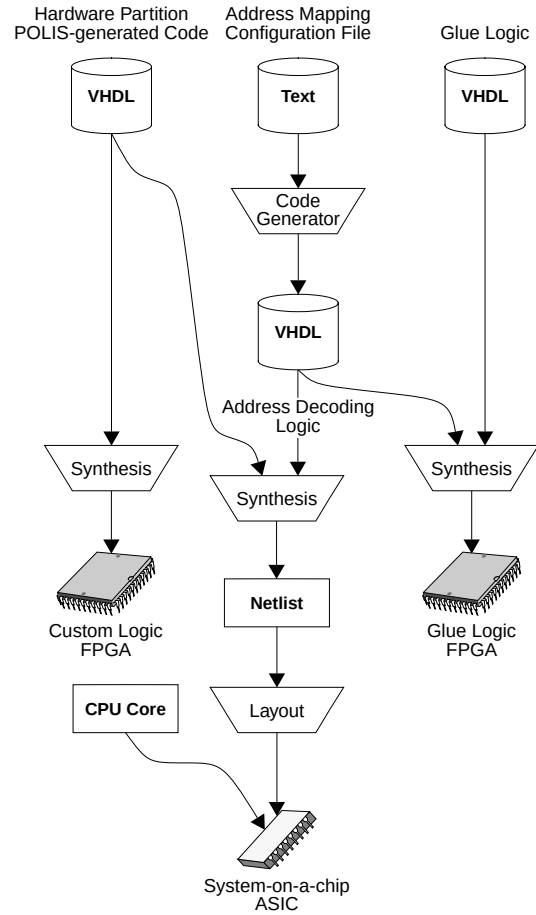


Figure 3: Design flow integration

4. Software Module Characterization

Beyond design validation, our coverification system provides characterization data for software modules. The real-time behavior is of special interest for timing estimation when reusing the modules and for automated hardware/software partitioning. A detailed description of the reuse-oriented codesign environment we integrated the profiling system into can be found in [10]. The profiling process can be performed manually or automatically.

4.1. Manual Software Profiling

First, the software module to be examined is compiled and linked in the usual way. Then the emulator control software on the host PC is invoked and the module is transferred into the emulator memory.

At this point, the user has very precise control over the program execution: Breakpoints may be set, registers and

memory location may be modified and the CPU is started. The emulation device can be connected with the target system (cf. figure 1) so that the interaction with it is taken into account. Since the emulator operating software offers various powerful profiling functions, sophisticated evaluations like determining data-dependencies in execution timing, measuring reaction delays for external control signals and preparing data access histograms may be performed. At the end of this process the user enters this data into the database of reusable modules (figure 4).

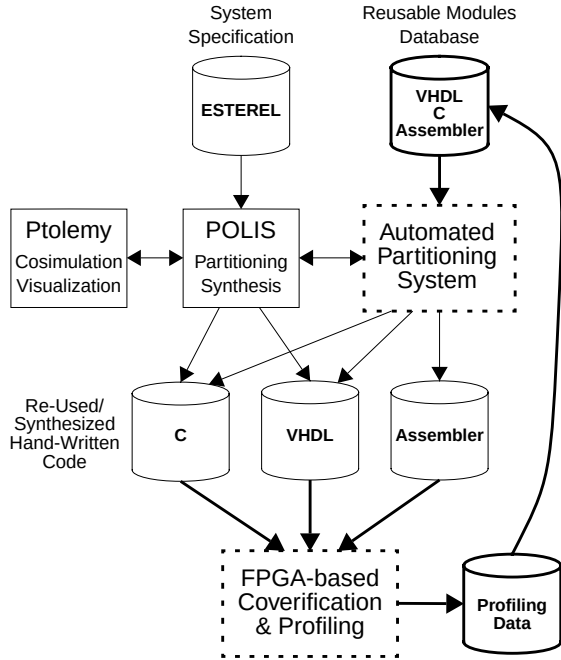


Figure 4: Feeding back profiling data

While this method allows to obtain the most detailed view over a piece of software, it requires intensive interaction by the user. Thus, this method is appropriate for thorough characterization of a small number of modules.

4.2. Automated Software Profiling

In order to efficiently characterize a large number of existing modules when the profiling requirements are less complex, the automated batch-mode profiling system (figure 5) is used. The hardware setup is identical with the one shown in figure 1 excepted that the host PC is connected directly to the module database server.

A batch file-controlled automated profiling program running on this PC fetches the source code of the modules from the database – one by one – and passes them to compiler/assembler (depending on the language) and linker. Communicating with the FPGA-enhanced in-circuit emulator via a manufacturer-supplied library (DLL) it downloads the binary code of the modules into the emulator memory. Next, it sets a breakpoint at the last code address,

parameterizes the profiler module (consisting of a cycle counter, address bus comparators and control machine) integrated into the glue FPGA and starts the CPU. After the execution has finished, it fetches the number of elapsed machine cycles from the profiler module and stores it in the module database.

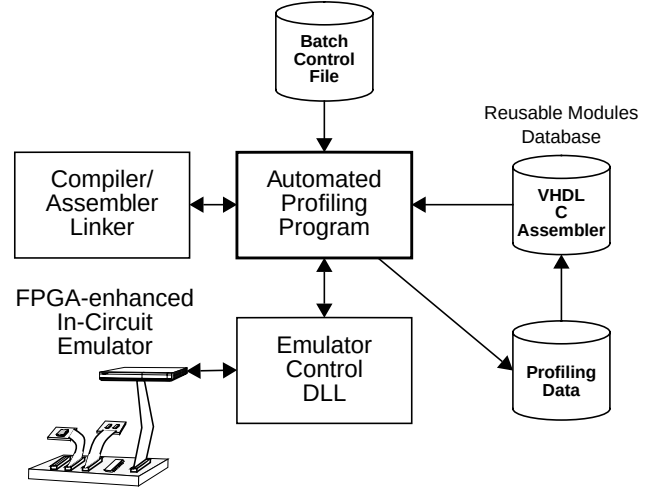


Figure 5: Automating the Profiling Process

Here the type of characterization is limited to data-independent single-function runtime measurement. Furthermore, profiling is restricted to simple modules containing only a single procedure without inter-module dependencies. Modules requiring specific stimuli to yield meaningful results have to be embedded into appropriate test benches prior to the batch run.

5. Experimental Results

A prototype of the coverification system has been implemented using FPGA boards populated with Altera FLEX10K50 devices together with an 8-bit microcontroller (68HC05 derivative). It was used for functional verification in the following design projects:

- A bus node controller for a 1 Mbit/s Controller Area Network (CAN) with three buffers for message transmission and reception.
- An arithmetic co-processor (ACP) with scalable data word length. A word length of 16 bit was chosen for the sample implementation.
- A CORDIC co-processor (CCP) transforming a vector from complex notation (Re, Im) into polar notation (ϕ , Mag) with 16 bit accuracy.

The characteristic results for our examples obtained from the analysis functions of the FPGA tool are summarized in table 1. The obtained clock frequencies seem quite low (for ASIC synthesis of all three modules a 16 MHz clock constraint was set and easily met) compared to mod-

ern standards even in the context of 8-bit machines, but they are perfectly suitable for in-system validation where the timing of the environment can usually be scaled down appropriately. The cell utilization of the glue logic FPGA (10K50) in these configurations was around 5%, while over 50% of the i/o pins were allocated. If the software characterization function is required, a profiler module (cf. chapter 4.2 for details) has to be added. This option allocates another 7% of the glue FPGA cells.

	CAN	ACP	CCP
ASIC Gate Count (kGates)	11	8	5
10K50 Cell Utilization (%)	72	54	27
Max FPGA f_{Clock} (MHz)	3.69	4.48	10.27

Table 1: Implementation characteristics

To evaluate the profiling functionality, we performed execution time measurements for a basic arithmetic calculation and a coordinate transform operation using the ACP and CCP modules described above. Data type for operand and result was chosen to be 16 bit signed integer in both cases. The results of this evaluation are shown in table 2.

Execution time (system clock cycles)	$\frac{a \cdot (b - c) + d}{e}$	(Re,Im)→ (φ,Mag)
Pure SW Implementation (no ACP, no CCP)	820 cy	1325 cy
Partial HW Implementation (with ACP, without CCP)	340 cy	785 cy
Pure HW Implementation (with ACP and CCP)	340 cy	52 cy

Table 2: Software profiling results

They demonstrate the large speed-up that can be achieved for the arithmetic calculation when the ACP is used while the CCP is obviously of no use for this class of operations. The coordinate transform algorithm however does not profit as much from the ACP, but can be sped up dramatically when the CCP is used. The impact of these findings on the partitioning decision is discussed in [10].

6. Conclusion

In this paper a FPGA-based coverification environment was presented that satisfies industrial requirements. The main advantage over commercial solutions lies in coupling the FPGA boards to an in-circuit emulator for the microcontroller core. This allows not only coverification in terms of function and timing but also easy and efficient software

development, debugging and profiling.

Analog or mixed-signal components can be included into verification. Integrating the coverification methodology deeply into the ASIC design flow maximizes the verification accuracy. Furthermore, we presented a profiling system that characterizes existing reusable software modules and feeds the resulting data back into the module database.

7. Future Work

The ability to put test and measurement instruments (over IEEE488 bus) as well as the in-circuit emulator (through the DLL interface, cf. figure 5) under software control allows the integration of the presented verification system into an automated lab environment. For this purpose, an emulator control driver for an appropriate lab automation system (e.g. LabView) has to be written.

An interesting parameter in safety-relevant systems is the frequency with which critical variables are updated by the software. Since the emulator allows to set “data breakpoints”, this feature can be easily implemented in the automated profiling system.

Due to the progress in semiconductor technology, FPGAs today have reached capacities that make the single-FPGA-integration of our PCB level construction (with the exception of an analog board) possible. This will also simplify retargeting the verification platform to another CPU type.

8. References

- [1] Balarin, F., Chiodo, M., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A., Sentovich, E., Suzuki, K., Tabbara, B.: Hardware-Software Co-Design of Embedded Systems – The POLIS Approach, Kluwer, Norwell, USA, 1997
- [2] Gajski, D., Vahid, F., Narayan, S., Gong, J.: SpecSyn: An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/Software System Design, IEEE Trans. on VLSI Systems, Vol. 6, No.1, pp. 84-100, 1998
- [3] Aubory, M., Page, I., Plunkett, D., Sauer, M., Saul, J.: Advanced Silicon Prototyping in a Reconfigurable Environment, <http://www.comlab.ox.ac.uk/oucl/users/ian.page/papers.html>
- [4] Liu, J., Lajolo, M., Sangiovanni-Vincentelli, A.: Software Timing Analysis Using HW/SW Cosimulation and Instruction Set Simulator, 6th International Workshop on Hardware/Software Codesign, March 15-18 1998, Seattle
- [5] Verkest, D.: Hardware/Software Co-design beyond Co-simulation, Proceedings of the MEDEA/ESPRIT Conference HW/SW Codesign, September 16-18 1998, Grenoble, France

- [6] Krupnova, H., Saucier, G., Nguyen, H.: An Advanced Approach to System Prototyping, Proceedings of the MEDEA/ESPRIT Conference HW/SW Codesign, September 16-18 1998, Grenoble, France
- [7] Valderrama, C., Naçabal, F., Paulin, P., Jerraya, A.: Automatic generation of interfaces for distributed c-vhdl cosimulation of embedded systems: an industrial experience, 7th International Workshop on Rapid Systems Prototyping, June 1996, Greece
- [8] Tabbara, B., Sgroi, M., Sangiovanni-Vincentelli, A., Filipi, E., Lavagno, L.: Fast Hardware-Software Co-Simulation using VHDL Models, Design, Automation and Test in Europe Conference DATE'99, March 9-12, 1999, Munich, Germany
- [9] Lajolo, M., Lazarescu, M., Sangiovanni-Vincentelli, A.: A Compilation-based Software Estimation Scheme for Hardware/Software Co-simulation, Seventh International Workshop on Hardware/Software Codesign, Rome, May 3-5, 1999
- [10] Meerwein, M., Baumgartner, C., Glauert, W.: Linking Codesign and Reuse in Embedded Systems Design, 8th International Workshop on Hardware/Software Codesign, May 3-5, 2000, San Diego, USA