

Performance Assessment of Embedded Hw/Sw Systems

J.P. Calvez, O. Pasquier

IRESTE La Chantrerie CP 3003 44087 Nantes cedex 03 France

fax (33).40.68.30.66, email: jcalvez@ireste.fr

Abstract

Performance assessment of embedded Hw/Sw systems built with various types of VLSI components, i.e. heterogeneous multi-processor architectures, is important to help the development of complex real-time applications. To design such a tool, two issues are to be solved, relevant information gathering simultaneously on several components without disturbing the application behavior, display of the performance results in a way easily interpreted by designers.

This paper presents an interesting solution for the two above issues. We first describe what the goal for designers is and what kind of applications are concerned. Then we describe the principle of collecting an event trace and the technique to evaluate the selected performance indexes. The monitoring technique, based on a specific ASIC, is non-intrusive and allows to capture real-time event occurrences from software tasks and even from hardware functions implemented in ASICs. Each event is automatically time-stamped, collected and processed in real-time to evaluate the performance indexes selected by the designer. We also describe the display tool which clearly shows to the designer the results according to different representations. This technique and the associate real-time performance analyzer are integrated in a whole development process based on the MCSE methodology.

1: Introduction

Today, the technology available leads to use a wide range of VLSI components to build distributed systems based on several less expensive microprocessors and microcontrollers instead of centralized solutions with a powerful processor. As a matter of fact, embedded real-time systems considered in this paper are heterogeneous multi-processor architectures resulting from the association of several microprocessors and different ASICs. During the implementation and test steps of such systems, the verification and validation tasks are critical. They concern at least the verification of the system behavior and the assessment of performance constraints.

The behavior verification requires two kinds of tasks: data transformation verification (check the algorithm), inter-function dependencies and timing constraints (check the overall behavior). The performance assessment needs to monitor the system embedded in its real environment, in order to capture appropriate information and evaluate from it the performance indexes for which constraints are imposed. This technique is also useful even to measure the performances of the final product.

In the last decade, tools have been developed to help engineers to design and implement both the hardware and the software parts of real-time systems but each separately, and to help the integration of the two parts in the ultimate environment. The usual way, cyclic debugging and then measuring some specific parameters, used for single processor systems and sequential programs, is not efficient for real-time and concurrent task applications. Moreover, for distributed processors, the global behavior is not completely deterministic and the relevant information to be measured is not easy to extract from the system without disturbing the real-time behavior. For multi-task and multi-processor systems, a technique to collect relevant information during each real-time execution and an efficient graphical tool to display it, are necessary to understand the complete application behavior, to detect errors and to verify performance constraints. The problem is more complicated for systems developed according to a CoDesign technique because parts of them are implemented in hardware.

Another major problem concerns the performance evaluation during the design step. Designers have first to define the appropriate functional architecture and then to find the partitioning and the allocation on the selected hardware. In hardware/software CoDesign, the solution is deduced from the required performance constraints.

Answering these problems requires to consider the whole development life cycle and to base system developments on a complete design model and methodology. The work presented here is based on the use of the MCSE methodology [3] and specifically on the benefits of its functional model which allows first to

describe Hw/Sw real-time systems and then to understand and measure system behavior in real-time. This model allows to automatically select all relevant information to be collected during the system runtime to allow performance assessment.

In this paper, we describe an efficient technique to collect relevant information from embedded Hw/Sw systems with a very low overhead minimizing the application behavior disturbance. We also describe a real-time performance analyzer we are currently prototyping which is useful to assess performance properties and constraints. Section 2 presents the main goals for performance assessment. Section 3 shortly analyzes similar works and explains our distributed hybrid monitoring technique. Section 4 describes the technique used to evaluate in real-time the designer's performance indexes. Section 5 gives an overview of the tool. Its integration into a whole framework of tools for MCSE is presented in Section 6. Conclusions are drawn in the last section.

2: Goals for performance assessment

The final quality of systems that designers develop depends on two important related issues: the involved technology, the development process. Here we are specifically concerned with embedded hardware/software systems. To clarify the desired objectives, we first precise the currently used technology and some difficulties to assess the operational status of a solution based on it. Secondly we consider the whole development life cycle and show the importance of strong links between architectural design, implementation and test & verification & validation. Appropriate goals are given for each of them with more precision on the kind of expected performances.

2.1: Monitoring issue for embedded real-time systems

A wide range of components is useful to implement embedded real-time systems [14]: microprocessor, microcontroller, digital signal processor (DSP), a large variety of ASICs (EPLD, CPLD, FPGA, gate arrays, etc.), specific VLSI components for communication, graphic display, I/O, high-performance parallel computing, etc. Most of these components are programmable; this means that the software mainly defines the whole system functionalities. An embedded software is usually a set of communicating processes. More and more often, an embedded system is implemented as a distributed system based on heterogeneous processors for several reasons: it is easier to meet hard deadlines by using different CPUs for time-critical tasks, the cost may be cheaper than using a single but more powerful processor, the number of devices (modularity) may be important and these devices may

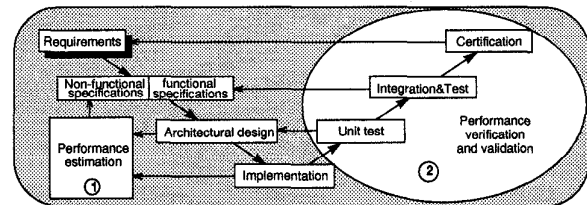
require to be geographically distant. In [14], Wolf uses the engine metaphor to illustrate such implementation architectures.

One can imagine the difficulties to debug such solutions: some functions are implemented in software and others in hardware, they have to communicate, timing constraints and performance must be satisfied and proved, dependability requirements may also be added. An appropriate monitoring technique is the main solution to collect relevant information. But with the increase of chip integration, we are near to find all parts - conventional microprocessor, DSP, I/O drivers, a field-programmable gate array to add custom functions - in only one component. Collecting appropriate information interesting for designers is therefore not an easy task because of the small visibility of the whole solution from the outside of such chips. One of our research goals is to propose an efficient technique to partly answer this increasing issue.

2.2: Performance assessment issue in the development process

A performance qualifies the behavior of the system relatively to observation criteria which may be external to the system (response time, throughput, etc.) or internal (utilization of a resource, bus throughput, etc.) [1]. Each kind of performance is called a performance index. Here we are concerned with dynamic performances which are the most difficult ones to evaluate and prove.

Monitoring techniques are used to observe relevant information, but what is this relevant information and how can appropriate performance indexes be evaluated? First of all, the final designers' objective is that the resulting system must satisfy all the customer's requirements. Test and verification after implementation are only a part of the solution. The whole development process has to be considered. Figure 1 shows the conventional V development life cycle. The first step is concerned with the customer's requirements which are then translated into functional and non-functional specifications. Performance constraints are one category of non-functional specifications. Performance constraints are one category of non-functional specifications.



-Figure 1 - The V development life cycle and performance assessment.

From specifications, designers have to decide on an architecture able to satisfy the application functionalities and performance constraints. The distributed nature of the architecture and the performance allocation on components

are decided during this step. The next step concerns the implementation. Then, unit tests, integration of all parts, tests of the whole system within its environment lead to the final certification stage.

Figure 1 identifies two important issues. In (1), designers have to estimate the performances of a selected architecture and to compare them with the required performances. During the implementation step, more accurate information is available to refine the performance estimation and if necessary correct the design. In (2), performances have to be measured and compared to the specifications.

In addition to performance evaluation, it is also important, first to efficiently test the functionalities of the system. The knowledge of the state of each task at any moment is an invaluable help to understand the application behavior. If the time-stamping of state modifications is accurate enough, temporal verifications such as timing constraints are then possible. It is also necessary to be able to observe the task inter-dependencies. These dependencies explain when an event or a message is generated, which task produces it, which task is addressed, when the addressee task uses it and becomes active. This kind of information is particularly relevant to interpret the global behavior because it represents the structural organization of the solution and helps to understand the task or function couplings. The behavior of an application, and therefore of some specific tasks, also depends on the value of some pertinent internal state variables. The history of these variable values gives complementary information to the designer to analyze the activity evolution and then leads to better understand the inter-task relation ordering.

Our proposition consists in using the same technique and tool for both tasks: performance estimation in (1) and performance verification in (2). In both cases, we use an event trace of relevant information from a simulation model of the architecture in (1), and from the prototype or final product running in real-time in (2). The interpretation of results, must be easy and the monitoring technique has to be as user-transparent as possible.

3: A distributed hybrid monitoring technique

The problem of monitoring and testing distributed real-time systems is described in several papers [10][11]. Solutions to collect relevant information from microprocessor systems can be classified in three categories: hardware monitoring, software monitoring, hybrid monitoring.

The hardware approach consists in connecting probes to the hardware system in order to observe its behavior without disturbing it. Logic analyzers and emulators are examples of this technique. This kind of solution often

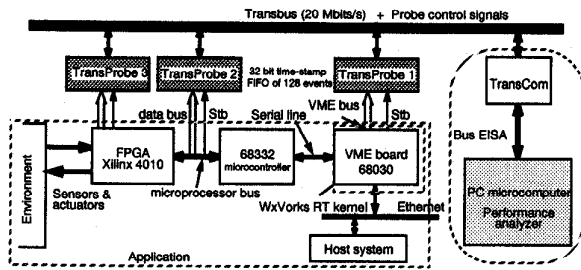
requires to develop a specific hardware system including the software to monitor it [9][12]. Information collected in this way has a low level of abstraction and is difficult to interpret. Today the main drawback of this solution appears with the new generation of microprocessors because they include internal memory, or cache memory. With future VLSI components including all parts (CPU(s), memory, FPGA) in the same chip, this technique does not work.

The software approach consists in adding to the software a set of instructions enabling to collect all useful information on the system behavior during the runtime. This kind of solution offers the advantage to be fairly independent of the target architecture. It also provides information at a high level of abstraction directly usable by designers or by a graphical display tool. Adding the necessary instructions in the program code can be automatically done by a tool which modifies the source program before compilation or by the compiler itself [8]. The main shortcoming of this solution is related to the time overhead introduced by the information gathering. Therefore, the system behavior is disturbed, and for real-time applications, the disturbance may be too important so that results are wrong and timing constraints are not met. For hardware parts, this technique is not possible.

The third approach, called hybrid monitoring, is a compromise between the two above solutions. It is based on the addition of few instructions in the software code in order to select adequate information useful to evaluate system properties. This information is collected with a specific hardware and is transmitted to a host system [4][6]. This approach has the advantages of the two other approaches because it gives information at a high abstraction level and introduces a low overhead. Unfortunately, solutions proposed in papers are often dedicated to specific hardware architectures [7][10][13]. The cost is also high because the solution is based on several boards interconnected through a data channel (ZM4 in [7]).

Our solution is based on the third approach. It allows to monitor the software and also hardware parts by adding specific wires. For FPGA and EPLD components, this is possible with the synthesis technique based on VHDL. Appropriate VHDL statements are added in the synthesizable hardware description to implement a way to observe the needed information.

Figure 2 represents the architecture of our solution and an example of embedded real-time system we have been developing.



-Figure 2 - Architecture of our solution for performance measurement.

The application is based on a distributed system including: a VME 68030 CPU board running the real-time kernel VxWorks from Wind River Systems and connected to a host through an Ethernet LAN, a 68332 microcontroller connected to the VME board through a serial line, an FPGA Xilinx 4010 in which specific hardware functions can be implemented.

The real-time performance analyzer is implemented on a PC microcomputer. A specific ASIC called TransProbe has been developed to capture events and time-stamp them. Many Transprobe components (up to 255) may be connected to an efficient 20 Mbits/s serial bus called TransBus [2] that we developed. In the PC microcomputer, a TransCom board is used to receive all events.

The TransProbe component includes:

- a 1 μ s precision clock able to time-stamp (31 bits) each event encoded in 32 bits,
- a 128 FIFO memory to record time-stamped events,
- a transmission function of these events on the 20 Mbits/s serial bus.

The probe chip is a 48 pin component which may be wired in a way similar to a pod of a logic analyzer. It may also be added as a specific component to the hardware during the board design. To make use of a valid global reference time, all clocks in probe chips are reset by a common signal and controlled by a master clock signal included in the bus and provided by the analyzer. This probe can be easily connected to any architecture: it has a 32 bit-wide input and needs only a strobe input signal STB to indicate the occurrence of an event, it may also be connected to a 16 bit-wide bus. In Figure 2, TransProbe 1 is connected to the VME bus, TransProbe 2 is connected to the 16 bit-wide bus of the microprocessor, TransProbe 3 is connected to specific outputs of the FPGA to have access to internal events and variables. Each relevant event is encoded into a 32 bit word (see Figure 3). Each code is a constant value which has a symbolic meaning for the system activity. To record an event occurrence, only one Write instruction in the software code is necessary (less than 1 μ s intrusion). To collect the value of a 32 bit shared variable, the value is encoded into two successive events.

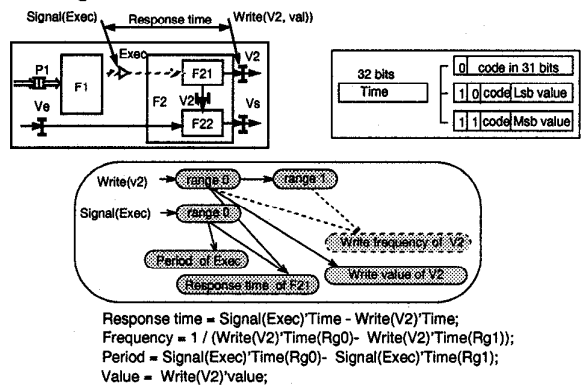
The TransBus bus allows a 1 Mbyte/s effective bandwidth, which leads to a 100,000 ev/s throughput. The bottleneck is not the bus but of course the PC microcomputer.

4: Principle of performance evaluation

One main problem is to add the appropriate instructions for event recording to the relevant locations in each task. To illustrate our technique, let us consider the simple example given in Figure 3 which concerns three functions F1, F21 and F22 located on the same or on different components. The description is conform to the functional model recommended in the MCSE methodology [3].

The task F1 is activated on the reception of a message in the port P1 and generates the event Exec. F21 is activated by Exec and modifies the value of the output V2. While F21 is running, it can modify the V21 variable shared with the task F22.

For performance evaluation, each event associated with the functional structure (task state modification, task dependencies by event synchronization, variable sharing, message transfer) may be interesting. Let us suppose we are interested in measuring the response time between the activation event Exec and the modification of the output V2. The occurrences of two events have necessarily to be captured: the event Exec generated by the primitive Signal(Exec), the event V2 write generated by the primitive Write(V2, Val). The response time is the time difference between the two events of the same range (range0) and can be calculated after the occurrence of the event V2 write. Evaluations have to be processed progressively in real time according to the arrival of events.



-Figure 3 - Example of a functional structure, event encoding and real-time performance evaluation.

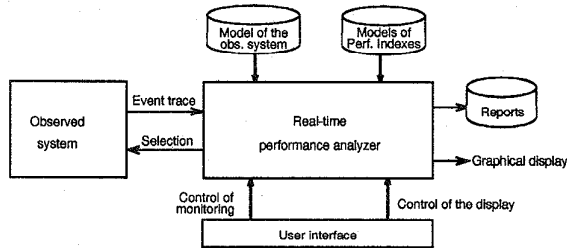
It is interesting to note that, with this technique and from only the two events, three other performance indexes may be evaluated: Value history of V2, write frequency of V2, period or frequency of Exec.

Based on the functional model of MCSE, we have done an inventory of more than thirty performance indexes whose evaluations are based on a time difference formula.

These performance indexes are implemented in the performance analyzer. All interesting events are representative of a state change in the functional model and have a meaning which is technology-independent.

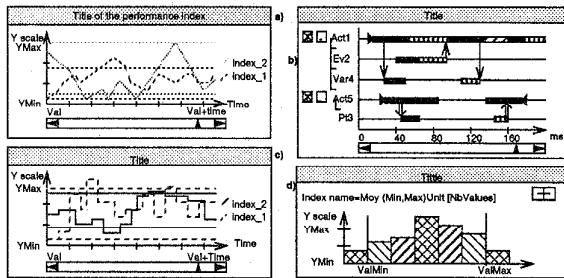
5: Result display

Figure 4 shows the delimitation of the real-time performance analyzer and its inputs and outputs. The tool uses the functional model of the observed system to correctly interpret each encoded event and to allow the user to select his performance measurements. The models of performance indexes can be user-defined to allow the enhancement of the tool by users.



-Figure 4 - Specification of the real-time performance analyzer.

The user controls the tool through an easy to use interface: control of monitoring parameters, control the display of results. Examples of interesting graphical representations are indicated in Figure 5.

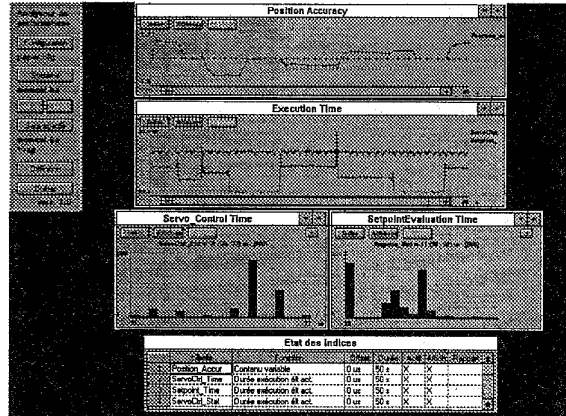


-Figure 5 - Examples of results and their graphical representation.

We have identified four useful index categories: time or frequency, number of elements or ratio, state of a component or element, value of a message or variable. Similarly, we have selected seven interesting index representations: time-continuous Fig 5-a), discrete-time Fig 5-c), bargraph, histogram Fig 5-d), list of values, component state and interdependencies Fig 5-b), pie-graph.

Figure 6 shows the human/computer interface and an example of graphical results obtained in real time. Results were captured from the hardware architecture of Figure 2. The objective was to monitor the position servo-control of a motor. The software was running on the VME board

with the VxWorks real-time kernel. The first graph displays the deviation between the position setpoint and the measured position to verify the maximum deviation. The second graph displays the execution time of two tasks - servo-control function and position setpoint evaluation - in order to verify timing constraints. These two results are also displayed like histograms to analyze the value distribution.



-Figure 6 - Results for a position servo-control of a motor.

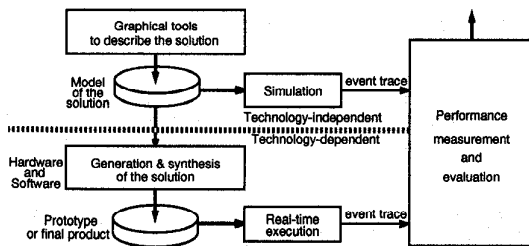
The design of this performance analyzer was done with the MCSE methodology. Because of the real-time aspect, a specific study was carried out concerning the real-time part of the tool: event collecting, index evaluation, display. The design of the graphical interface was also an essential part of the project to obtain an easy to use tool. An object-oriented approach was chosen for that purpose.

All functionalities are not yet implemented. Performance indexes may be easily added. A rough estimation of the real-time ability of the tool is between 1,000 and 10,000 event occurrences/s with a 486 DX2-based microcomputer. A new version will use a Pentium-based microcomputer and a specific powerful interface between the TransBus bus and the PCI bus. As a matter of fact, the bottleneck is the transfer of each 10-byte message (7 μs on TransBus) in the main memory of the PC. The limited efficiency of Windows is also another problem for real-time applications.

6: Prototype generation

Our monitoring technique and the real-time performance analyzer are an integral part of a set of tools we have been developing as a help to the MCSE methodology. Figure 7 shows the strategy we are currently experimenting to answer the goal of

performance estimation and evaluation during the whole development life cycle (point (1) and (2) in Section 2.2).



-Figure 7 - Tools and procedure to performance assessment.

Graphical editing tools are used to capture the functional structure, the executive (or hardware) structure, the allocation and configuration. The description is technology-independent. A simulation model is produced from this description (VHDL and C++), the run of which generates an event trace in conformance to the above description. Therefore, the performance analyzer is usable to predict performance during the top-down design step.

To obtain a prototype, we are currently developing a CoDesign method and an appropriate generator of the hardware and software parts of the solution [5]. The TransProbe architecture is added to the hardware part. The running of the system generates a real-time event trace which is processed by the performance analyzer. A back-annotation of the model is then possible to enhance its accuracy.

When the monitoring mode is wanted, the generator automatically adds necessary instructions depending on the selected performance indexes. Since events are encoded, the generator also produces a file including the meaning of every code used in the generated program.

7: Conclusion

In this paper, we have described a technique and a tool to help designers develop embedded hardware/software systems. The objective is twofold: first to estimate the performances of an architecture during the design step, second to measure and evaluate the real performances and compare them with the performance requirements.

The performance evaluation is based on event traces captured from a simulation model or from the real system or a prototype. Conceived for heterogeneous distributed architectures, our technique is based on a specific probe ASIC and a fast serial bus. The performance analyzer is programmable by the user according to the performance indexes he wants to verify. Results are displayed in graphical forms and in real time.

The method and the resulting tool are an efficient help for the design of real-time systems developed according to a complete system-level methodology. The functional model is used to easily select the relevant information that

must be monitored in order to easily understand the behavior of the application during its real-time run and to estimate its performances. For that, our tool, even usable alone, is wanted fully integrated in a complete set of tools to help CoDesign of embedded systems.

References

- [1] R. Bordewisch, W. Föckeler, B. Schwärmer, F.-J. Stewing, Non-Functional Aspects: System Performance Evaluation, In "Systems Engineering. Principles and Practice of Computer-Based Systems Engineering", Editor B. Thome, John Wiley, 1993, pp 223-271
- [2] J.P. Calvez, O. Pasquier, A TRANSpouter interconnection BUS for hard real-time systems, Transputer'92 Besançon France IOS Press, May 20-23, 1990, pp 273-283
- [3] J.P. Calvez, Embedded Real-time Systems. A specification and Design Methodology, John Wiley, 1993
- [4] J.P. Calvez, O. Pasquier, Real-Time behavior monitoring for multi-processor systems, EUROMICRO'93, Barcelona, Sept. 6-9, 1993
- [5] J.P. Calvez, D. Isidoro A CoDesign experience with the MCSE Methodology, Proceedings of the third International Workshop on Hardware/Software Codesign, Grenoble, France, Sept 22-24, 1995, pp 140-147
- [6] D. Haban, D. Wybraniec, A Hybrid Monitor for behavior and performance analysis of distributed systems, IEEE transactions on Software Engineering Vol. 16, No 2, Feb. 1990, pp 197-211
- [7] R. Klar, Event-Driven Monitoring of Parallel Systems, in Performance Measurement and Visualization of Parallel Systems, G. Haring and G. Kotsis (Editors) 1993 Elsevier Science Publishers, pp 145-173
- [8] E. Maehle, W. Obeloer, Delta-T : A user-transparent Software-Monitoring Tool for Multi-Transputer Systems, Microprocessing and microprogramming 35, North-Holland, 1992, pp 245-252
- [9] B. Plattner, Real-time execution Monitoring, IEEE transactions on Software Engineering, Vol. SE-10, No 6, nov 1984, pp 756-764
- [10] U. Schmid, W. Kastner, Monitoring of distributed Real-Time Systems: The versatile Timing Analyzer (VTA), in Performance Measurement and Visualization of Parallel Systems, G. Haring and G. Kotsis (Editors) 1993 Elsevier Science Publishers, pp 227-301
- [11] W. Schütz, Fundamental issues in testing distributed real-time systems, Journal of Real-Time Systems, 1993, pp 129-157
- [12] J.P.P. Tsai, K.Y. Fang, H.Y. Chen, A Non-invasive architecture to monitor Real-time Distributed systems, Computer Vol. 23 No 3, March 1990, pp 11-23
- [13] A. Wagner, J. Jlang, S. Chanson, Tmon: a performance monitor for transputer-based multicomputers", Concurrency: Practice and Experience, Vol 5(6), Sept 1993, pp 511-526
- [14] W.H. Wolf, Hardware-software Co-Design of embedded systems, Proceedings of the IEEE, Vol 82, No 7, July 1994, pp 967-989