A Hardware and Software Monitor for High-Level System-on-Chip Verification

Mohammed El Shobaki and Lennart Lindh
Mälardalen Real-Time Research Centre
Department of Computer Engineering
Mälardalen University, Västeras, SWEDEN
mei@mdh.se, llh@mdh.se

Abstract

Verification of today's Systems-on-Chip (SoC) occur at low abstraction-levels, typically at register-transfer level (RTL). As the complexity of SoC designs grows, it is increasingly important to move verification to higher abstraction-levels. Hardware/software co-simulation is a step in this direction, but is not sufficient due to inaccurate processor models, and slow hardware simulation speeds. System-level monitoring, commonly used for event-based software debugging, provides information about task scheduling events, inter-task communication and synchronisation, semaphores/resources, I/O interrupts, etc.

We present MAMon[1]*, a monitoring system that can both monitor the logic-level and the system-level in single/multiprocessor SoCs. A small hardware probe-unit is integrated in the SoC design and connects via a parallel-port link to a host-based monitoring tool environment. The probe-unit collects all events in the target system in run-time, and timestamps them with a resolution of 1 μs. The events are then stored in a database on the host for further processing. The paper will describe MAMon and how it works for software and hardware monitoring. The paper also describe how system-level monitoring can be achieved non-instrusively by using a hardware-based Real-Time Kernel.*

1 Introduction

Already today many System-on-Chip (SoC) applications are hard to verify and optimise. The complexity is increasing and to verify a whole system using computer model simulations is time consuming, and some times impossible due to inaccurate models. It is also difficult to model the real-world (i.e. the environment) around the SoC.

[1] Multipurpose/Multiprocessor Application Monitor

Often a bug in a system is traced from a high-level view of the system, commonly referred to as the *system-level*, downto the *register-transfer level* (RTL) (or even lower levels if things are really bad). In this paper we refer to the system-level as to denote both the process/task-level information in software, and the behavioural-level information in hardware. As figure 1 illustrates, the number of events occuring in a system are fewer at the system-level, which motivates a top-down debugging strategy.

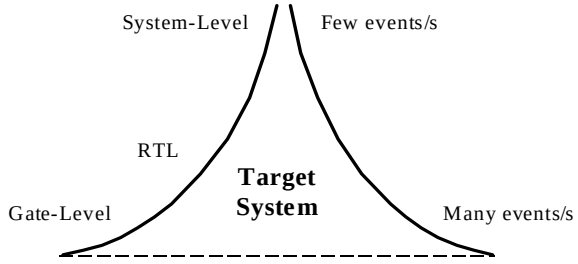

Figure 1. Events in a target system

Observability into SoC designs is today mostly supported for RTL verification. As SoC designs tend to increase in size and complexity, the verification process need also to take place at the system-level. Support for system-level verification already exist for the software part of a SoC, where a common technique is the use of *software monitors*. These monitors are typically found in RTOS (Real-Time Operating System) development tool environments ([10, 1]), and provides the developer with process-level information such as task-scheduling events (start/stop, block/resume, taskswitch, etc.), inter-process communication events (e.g. send and receive messages), synchronisation and resource utilization (CPUs, semaphores, I/O), external interrupts, etc. For this information to be extracted it is often required to instrument the software with special monitor instructions or processes, which later

can be removed after the validation phase. The drawbacks of instrumentation is that it utilizes target resources such as memory space and execution time from the CPUs. Also, when the instrumentation is removed, the system may change behaviour due to timing differences. This problem, commonly referred to as the *probe effect* ([6, 9]) causes many timing- and synchronisation-related errors in concurrent/distributed systems and real-time systems [12].

In this paper we present MAMon, a hardware-based monitoring system that makes a SoC observable at different abstraction-levels both in hardware and software. MAMon is a solution that integrates a small hardware component into the SoC. It works like a probe, either by listening to logic- or system-level events in a passive manner, or by being activated by software that writes to a specific register. Detected events are time-stamped and sent via a link to a host-based tool environment where the events are stored in a database. The tool environment includes a set of fascilites to view, search, and analyse the events in the database. Depending on the capacity of the database disk system, and the rate at which events occur, a monitoring session (or execution history) may be several days long.

In systems running a software RTOS, the process-level events can be extracted by instrumenting the code with *hook routines* that writes information to the MAMon hardware. Hook routines are small functions that can be attached to the RTOS functions, e.g. task scheduling events, system-calls, and so on. Except the extra recourses that the instrumentation would require, it may also render problems if it has to be removed after system validation. An alternative could be to monitor the RTOS via the CPU's bus activities. This would however require that bus activities be visible outside the CPU, so cache-memories may need to be turned off if the CPU does not support monitoring of internal bus logic.

When using a hardware-based RTOS kernel [7] for the management of SoC software, we will show how the process-level events can be monitored without software overhead and with no instrusion on the system's timing behaviour. Moreover, since it is non-instrusive, it gives the SoC developer the free choice of either keeping or removing the monitor component in the final product.

The use of MAMon may be different for FPGA SoCs and ASIC SoCs. Leading FPGA manufacturers (Xilinx [4] and Altera [3]) are now promising SoC solutions with embedded *soft* or *hard* processor cores (IP). In such FPGA-based SoCs, MAMon's probe component could be instantiated and connected to the RTL and system-level description of the design during the verification and validation phase. After synthesis for the FPGA, the connections to the design are easy to change, and re-synthesise. For ASIC implementations, however, it is much more costly to redesign. In this case MAMon may be more suitabel for long-term usage as a monitor for CPU busses, and hardware-based real-time kernel events.

The paper is organised as follows: Section 2 gives an overview of MAMon and how it is used for hardware and software monitoring respectively. Moreover, we give a detailed description of the internal layout of the hardware, and the host interface to the tool environment. The tool environment is only described slightly. Finally, section 4 and 5 summarizes the paper with a discussion on further work and some concluding remarks.

2 MAMon

The proposed monitoring system, called MAMon, aims at providing means for event-based hardware and software debugging of single- and multiprocessor SoCs. An overview of MAMon is depicted in figure 2. In this approach the monitoring system is comprised of three parts; *(i)* the *Probe Unit* which connects to internal SoC logic, *(ii)* a *Tool environment* residing on a host computer system, and *(iii)* the communication link between the Probe Unit and the Tool environment, called the Host Interface.

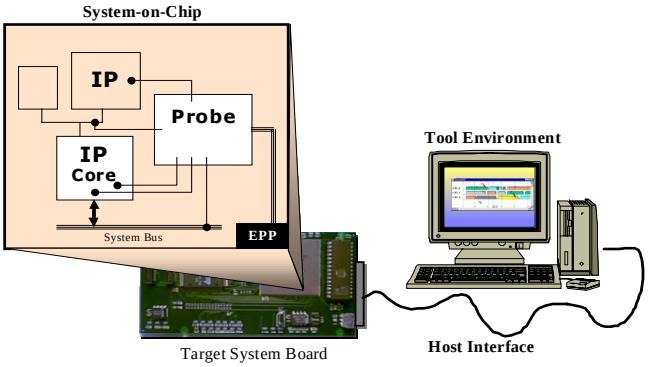

Figure 2. Overview of MAMon

The Probe Unit (PU, described in next section) is integrated in the design HDL code, and connected to signals that constitue the events to be monitored. For instance, an event could be defined as an access of a SoC component (IP), a certain condition on a bus (address, data, or control), arrival/contents of communication data, or an interrupt assertion, and so on. Then, in run-time, the PU performs detection, timestamping, and recording of events.

Recorded events are transferred, via the Host Interface, to the Tool environment, where a database is used to store the events for further processing in display and analysis tools (section 2.3).

In certain cases there is need to cause events from software, for instance to monitor the system-level events occuring in a software real-time kernel. Such events are produced by inserting software instructions (software probes) that writes to a PU register connected to the system/processor bus. Software probes can also be used as checkpoints in the code (*flags*), or to report memory contents.

Not only may system-level events be monitored, but also the hardware logic itself can be analysed and depicted against higher-level events, e.g. in waveform graph tools. This feature is useful for tracking down hardware logic errors which cannot be analysed using conventional probing methods (e.g. logic analysers and oscilloscope).

2.1 The Probe Unit

Figure 3 shows a block-diagram of PU's internal organisation. The component illustrated in top of the figure is the *event-detector* which merely performs conditional comparisons (comparator) on input signals. The input signals are hard-wired (in HDL) from selected points in the SoC. Also the condition expressions that defines events are hard-coded in the event-detector. When a certain condition for an event is detected, a sample is collected and stored immediately along with a timestamp in an on-chip memory buffer. Events generated by software, i.e. software probes, are detected as write-accesses to a 32-bit register (SWPROBE_REG) in the PU's bus-interface (address/data/control).

An event-sample comprises the event-type, the timestamp, and an event-defined parameter field, see figure 4. The parameter field is used to store additional information about an event. For instance, for various access-events to IP-components to be enough informative, the parameter field might contain call parameters, e.g. a bus-vector. The parameter field for a software-probe constitutes the 32-bit value that was written to SWPROBE_REG.

The timestamp comes from a 32-bit timer device which denotes the relative system time with a resolution of 1 μs per tick. A default timer device is included in the PU.

Since the amount of event samples can be rather large before it can be communicated to a host computer, the on-chip memory requirements may not be feasable because of area or economical constraints, especially in FPGA

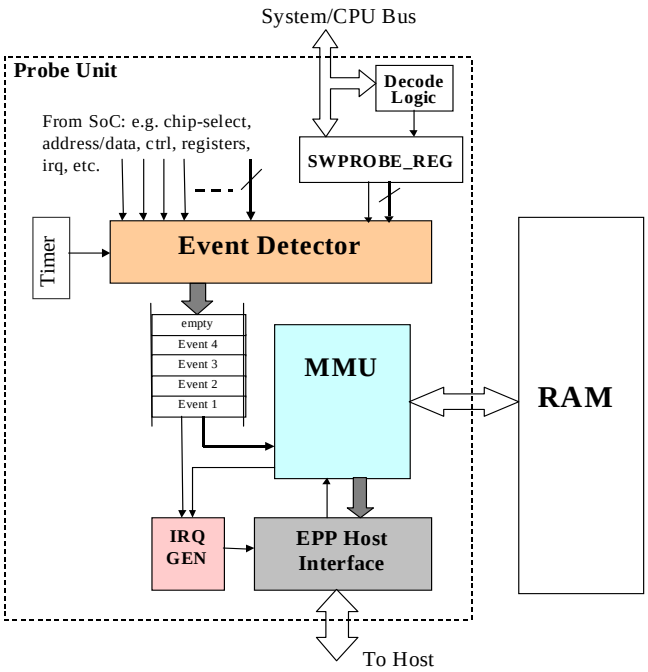

Figure 3. Internal organisation of the Probe Unit

SoCs. Therefore an external (off-chip) RAM buffer could be required. If an external RAM buffer is used, the on-chip buffer would still be needed in order to avoid write-access latencies to the external RAM. With this buffer configuration it is possible to detect and store up to 5 events occuring within 1 μs, if a clock speed of 10 MHz is assumed. This is quite useful in some extreme situations, for instance, when monitoring timing-dependant response to external interrupts when there are several sources of competing interrupts.

The Memory Management Unit (MMU) is responsible for moving event samples from the internal buffer to the external RAM. Both buffers are organised as circular FIFO buffers for maximum space utilisation. The current version of MAMon is implemented with a 128kB RAM buffer. With an event sample size of 10 bytes (figure 4) this means that more than 13 000 events can be stored before the buffer needs to be emptied. Furthermore, the MMU manages requests to move sample data from RAM to the host computer by way of the parallel port interface (described in next section).

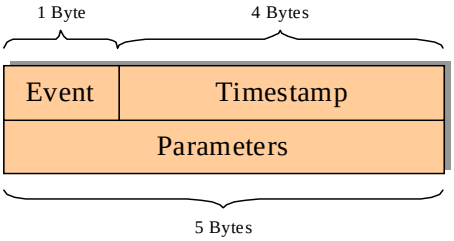

Figure 4. The event sample format

2.2 Host interface

Since the on-board event buffer is limited it is important that event samples are transferred to the host with a guaranteed high communication bandwidth. Therefore, a parallel port implementing the bi-directional *Enhanced Parallel Port* protocol (EPP 1.9 [2]) is used as the host communication interface. With EPP the event samples can be transferred with a rate up to 2MB/s, or more than 200k events per second.

As part of the host interface is the *interrupt generator*, refer to figure 3. This component can be programmed to interrupt the host computer whenever there are new events in the buffer. When enabled, the interrupt generator can be set into one of three modes:

- Interrupt whenever new events are detected
- Interrupt when the RAM buffer is half-full
- Interrupt when the RAM buffer is full

The first two modes are useful when continous monitoring is desired. The third mode is more useful if the PU is set to sample from a given command until the buffer becomes full, and then stop. Providing the ability to choose the interrupt mode gives a customised solution that best suits the capabilities of the host computer performance, the tools, or the user. When the interrupt function is disabled, events can still be acquired in *polled* mode. Control of the PU's behaviour, and acquisition of event samples and other status information, is all done via the EPP register interface, i.e. from the host computer system.

2.3 The tool environment

The proposed tool environment provides the user with facilities to view and search the event samples received from PU. In order to manage possibly huge amounts of events that can be produced from long system runs, the received data must be stored in file-structures that are optimal for searching. A database will therefore be used for storage of the event samples. As illustrated in figure 5, the database then acts as a server for various requests from the tools.

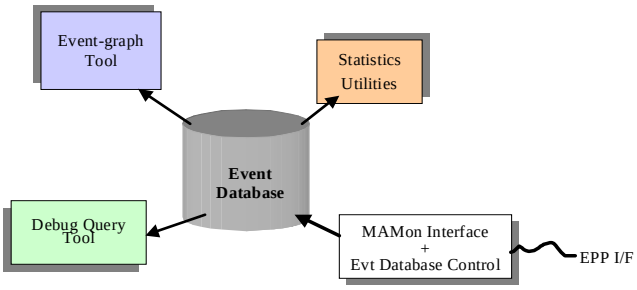

Figure 5. MAMon's tool environment

An event-graph that displays portions of the event history is a necessity in order to help the user in finding erroneous execution patterns. The event-graph tool, illustrated in figure 6, collects events from the database, and displays them along a timeline. Apart from standard functions such as zooming and scrolling, there is also support for time-markers that are used for timing measurements, and search-markers that can be used to locate event conditions and patterns. For logic-level events, the tool looks and behaves similar to a waveform graph. The difference however, is that the tool is able to show a mix between logic- and system-level information on the same timeline, giving the user the ability to correlate events in the hardware and software.

In order to ease visibility, and understandability of the execution, an event-filter can be used to hide excess information. The filter-tool can also reduce the search-space which will improve performance of the database.

The event-database is also suitable for other post-analysis, such as extraction of various statistics. Examples on such applications are; diagrams and histograms showing task's execution-time, processor utilisation, IPC frequencies, interrupt-response times, etc.

3 An Ideal Example: Monitoring a Hardware Real-Time Kernel

When connecting the MAMon system to a hardware-based RTOS kernel, the process-level can be extracted with zero software overhead, and thus, without changing the timing behaviour of the system. A hardware RTOS kernel implements traditional (software) RTOS functions (e.g. scheduling algorithms, task management, inter-process communication primitives, synchronisation, semaphores, event flags, etc.) in hardware. The RTU (Real-Time Unit [7, 5]) is such a component that has proven to be suc-

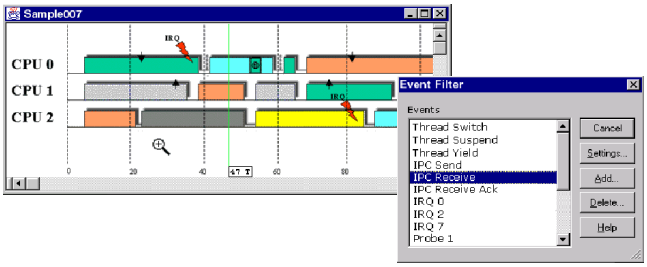

Figure 6. Example tools: Event-graph and event-filter

cessful for increasing RTOS performance and operational predictability. Moreover, it can also be used as the single RTOS for both single- and multiprocessor systems [8].

The connection between MAMon's Probe Unit and the RTU is done using signals in VHDL ([11]). The recorded events are then transferred to the host-based tool environment, where a process-view can be displayed using the event-graph tool. The provided fascilities in the event-graph tool can help the designer to find erroneus execution patterns and/or be used to tune performance and load-balance in a multiprocessor SoC.

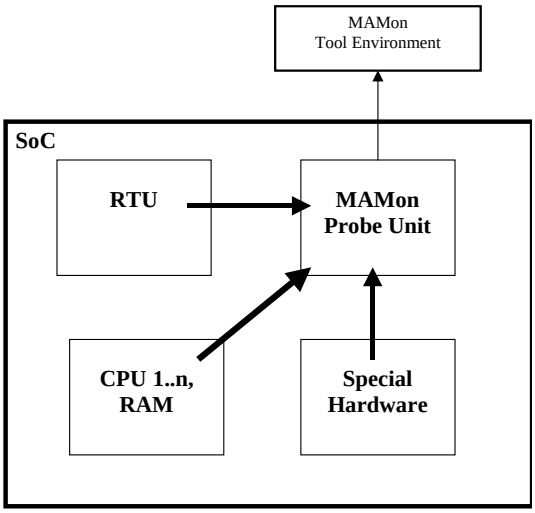

Figure 7. A RTL and system-level monitoring configuration using MAMon

Figure 7 illustrates a SoC configuration using the RTU for scheduling of one or more CPUs, and MAMon for monitoring the system at both the RTL and system-level. In this configuration no additional monitoring software is required on the target.

4 Current and Further Work

A first version of MAMon is currently being implemented together with a simple tool environment including the event-graph display tool. In this version only tasking activities inside the RTU are monitored. Further work will primarily focus on adding support for monitoring the other RTU features, such as inter-process communication management, and handling external interrupts. Moreover, there will be extensive work on development and improvement of the tool environment. The results will be published in a forthcoming paper, together with a case study showing the use of MAMon for debugging of typical timing and synchronisation errors. There are also plans to extend the Probe Unit with support for run-time detection of user-defined event-patterns which then can be used to halt the software, either completely (all CPUs) or partially as per CPU or task, or groups of CPUs and tasks. This feature could be utilised to implement synchronous and consistently halting breakpoints for use in cooperation with traditional source-level debuggers.

5 Conclusions

On-chip support for monitoring and debugging is becoming critically important since traditional solutions that uses in-circuit emulation (ICE) techniques, logic analysers, and oscilloscope, do not keep pace with today's system speeds. Moreover, on-chip approaches are motivated because of limited pinouts in chip-packaging, and even difficulties in reaching the physical pins (e.g. Ball-Grid Arrays, BGA).

The approach of integrating MAMon on a SoC offer an on-chip solution that also gives non-intrusive, synchronous, and consistent RTL and system-level monitoring. This, in turn, is ideal for event-based debugging and profiling of embedded real-time SoC applications. MAMon together with a hardware RTOS kernel gives a simple solution to process-level monitoring without requiring additional software overhead.

In a FPGA solution it is also convenient to monitor a mix between RTL system-level to get an effective debug and optimisation environment. In an ASIC solution MAMon is more suitable for run-time system-level monitoring, e.g. for process views.

All monitored data are time-stamped with a resolution of 10 times the system clock freqency. This capability is important so that events at different abstraction-levels can be compared and correlated with a high precision.

The requirement to manually connect SoC logic and signals to MAMon's Probe Unit, and then define event-

conditions, is unconvenient and can be tricky to handle for SoCs with many small submodules. Here, it is desirable to have a tool, preferable an interactive GUI, that automates the necessary connections and definitions in HDL. In this case, the HDL code would probably need pre-processing before compilation and synthesis.

References

[1] H. Bawtree. Real-time monitoring with StethoScope 5.1. *Software Development*, 7(9), September 1999.

[2] Enhanced parallel port v. 1.9. IEEE 1284.

[3] Altera corporation. http://www.altera.com/. 101 Innovation Drive, San Jose, CA 95134.

[4] Xilinx inc. http://www.xilinx.com/prs_rls/ibmpartner.htm. 2100 Logic Drive, San Jose, CA 95124-3400.

[5] J. Furunas, J. Starner, L. Lindh, and J. Adomat. *RTU94 - Real-Time Unit 1994 - Reference Manual*. Computer Architecture Lab, Dept. of Computer Engineering, Malardalen University, Vasteras, Sweden, January 1998.

[6] J. Gait. A probe effect in concurrent programs. *Software - Practise and Experience*, 16(3):225–233, March 1986.

[7] L. Lindh et al. Hardware accelerator for single and multiprocessor real-time operating systems. In *Seventh Swedish Workshop on Computer Systems Architecture*, Goteborg, Sweden, June 1998.

[8] L. Lindh, T. Klevin, and J. Furunas. Scalable architectures for real-time applications - SARA. In *CAD & CG'99*, December 1999.

[9] C. E. McDowell and D. P. Helmbold. Debugging concurrent programs. *ACM Computing Surveys*, 21(4):593–621, December 1989.

[10] Mentor Graphics, Microtec Division. *XPERT Profiler - Measurement and Evaluation Tool*.

[11] S. Sjoholm and L. Lindh. *VHDL for Designers (473 pages)*. Prentice-Hall, January 1997.

[12] J. J.-P. Tsai et al. *Distributed Real-Time Systems: Monitoring, Visualization, Debugging, and Analysis*. Wiley-Interscience, 1996.