Abstract— We have seen the practical use of multi-processors in complex SoCs and systems grow in the past several years, and the discussion range from architectures through to programming models. One of the issues that poses several challenges to design and verification teams is that of multi-core debug, especially in heterogeneous systems where the processors may be from different vendors, and even when from the same vendor, may be very application-specific. In this panel, designers and researchers who have practical experience with heterogeneous multiprocessor systems, both commercial and research, will draw on those experiences.

Keywords-debugging, SoC, multicore, manycore

I. ALBRECHT MAYER: YOU CAN DESIGN IT – BUT CAN YOU DEBUG IT?

Will designers get more pins and silicon area for addressing this challenge or will the management view that debug support is only needed by engineers who make errors, curb these dreams? The sweet spot is between these positions, but where?

Points for the discussion:

- What are the specific multi-core debug challenges?
- What challenges are encountered in real life?
- What works in heterogeneous multiprocessor debug?
- What are the successful methods and approaches?
- Do we need a DfD (Designed for Debug) SoC Architecture?
- Does analysis and performance optimization need more attention for multi-core?
- How much debug can be done with (C-)models?
- On-chip or off-chip trace?
- Can debug resources used also within the end-user product?
- Will software instrumentation do the trick?
- Would new standards help meet the challenges?
- Are there new standards developments coming?

II. THE PANELISTS

Uwe Steeb, Blue Wonder, Dresden, Germany:

STANDARDIZING SYNCHRONIZED DEBUGGING

The synchronization of processors from different vendors for debug purposes is still an open problem. Synchronizing here means to stop all processors on an SoC in case a debug event occurs on one of them, and starting them at the same time again when all involved debuggers have finished debugging.

As synchronized stopping is almost solved by a de-facto industry standard, the problem of synchronized restart is complex and the available proprietary solutions are not applicable to a system consisting of heterogeneous processors and debuggers from different vendors.

Further, it is desirable to not only synchronize processors, but also peripherals on an SoC like timers and DMAs. This can be seen as “communication centric debugging”.

As a central approach for standardization we suggest to add a dedicated HALT input signal to each processor, and to all peripherals that shall be synchronized. This introduces a HALT state to the processor additional to the already existing RUN and DEBUG state. External debuggers can be extended to show this state to the User, but this is not essential for the concept to work.

It could be shown that this additional state solves the main synchronization problems. The hardware effort inside the SoC is low, as many processors already have a HALT or STALL signal. Existing debuggers can be re-used without change, which is one of the biggest benefits.

Daniel Sobe, NXP, Dresden, Germany:

(No position submitted).
**Professor Gerhard Fettweis, TU Dresden, Germany:**

The Intel 4004 was a revolution in the market place of electronic system design. By putting 2300 transistors on a chip, a new component level of functionality was available, being one basic ingredient of kicking off the information society.

Today we are facing 2300 processors being implemented on a chip in near future. As an example, for implementing the cellular terminal modem of the year 2015+ (LTE-Advanced) alone the signal processing power of up to 1000 processors will be needed. Adding application functions as multimedia and graphics on an integrated cellular terminal chip will lead to reaching the level of the 4004, however, in complexity the transistor being replaced by the processor.

Today, we are often assuming that every function on a chip is matched to a specific hardware implementation. Just as a foundry cannot afford supporting a zoo of transistor types, we will also have to consolidate to a limited number of processor types, accepting lacks of efficiency to receive a manageable design process. Assuming this, the next question to be addressed is the debugging of the chip during the design process. Again, supporting a zoo of processor types cannot be the way forward, as it leads to an unmanageable debugging process.

A real innovation needs to be carried out at the MPSoC architecture level to enable a clear and easy debugging process. To *enable debugging*, some additional questions are e.g.

- What is the correct MPSoC programming model?
- What is the correct interfacing model between functional units?
- How to implement hierarchy in design, as well as debugging interfaces?
- How can the VLSI success of “divide-and-conquer” be carried over and defined at MPSoC level?
- How can a design be structured to create reproducible bugs, and therefore to be able to debug a system?
- What is the on-chip state space that needs to be monitored at the system level, which has to be known to reliably track bugs successfully?

**Stephan Lauterbach, Lauterbach, Munich, Germany:**

Complex SoCs of today are usually the combination of cores and IPs from different sources. This results in a mix of different debug technologies on one chip. Restrictions in pins and silicon area leads to the need for debug tools that can support this technology mix with one tool thru a minimum number of pins. Complex power management schemes put additional burdens on the debug tools.