A Seamless Tool Access Architecture from ESL to End Product

Albrecht Mayer
Infineon Technologies AG, 81726 Munich, Germany
albrecht.mayer@infineon.com

Abstract

Tool access to processor cores is needed from the first ESL (Electronic System Level) models, over chip design simulation, silicon on wafer, silicon on test board and prototype systems up to the real end product. Tools are in particular debuggers, but also performance optimization tools and domain or system specific tools. This paper will start with applications and requirements, discuss then the basic Tool Access Architecture (TAA) and finally describe a solution in more detail, which has been implemented (productive or at least prototypes) for all development stages from ESL, chip simulation and silicon up to the automotive target system.

1. Introduction

The motivation for a unified tool access are the significant cost and time to market benefits due to reuse and the possibility to execute tasks earlier.

Systems are getting more and more complex. A multi-core SoC used within a demanding real-time system is now standard system architecture in many application areas. The other trend is the shift of functionality from hardware to software. Both together require that software is already at least partly developed in the concept phase based on ESL models. This is beneficial for proving the hardware and software concepts, but requires that the standard software tooling is available also for ESL models. Since there are in such projects already more software than hardware developers, and this shift is still going on, this is an interesting market for tool vendors. However it needs to be understood, that the rules come from the software side, where the reference is the free GNU compiler and not a multi 100k $ HDL synthesis tool license.

For complex system design the development flow starts with software development on the ESL model and ends with debugging the target system. For these development steps the tool access to the SoC serves not only as the software debugger connection, it is the key entry point for analyzing the whole system. A tool architecture, which allows using the same tools over the whole development process, will significantly reduce cost and risk.

2. Applications

Beside the most obvious use case for a unified TAA, using the same software tooling from ESL model to end product, there are also other important TAA applications:

- System and software debug and optimization
- Silicon debug
- Silicon validation tests development
- Tool chain debug

Table 1 shows the mapping of the TAA application to the different development steps.

	ESL Model	HDL Simul.	FPGA Proto.	Silicon /Device	Target System
SW Dev.	X	(X[1])	X	X	X
Silicon Debug				X	(X[2])
Silicon Val. Test Dev.	(X[3])	(X[4])	(X)	X	
Tool Chain Debug	(X)	X	X	X	

Table 1: TAA Applications and Targets

[1] Very low level software like the start-up software in the boot ROM. Another special application is running known (bug-free) software on a new hardware design (e.g. new processor pipeline implementation).

[2] Special use case is the analysis of field returns

[3] Only for high-level silicon validation tests

[4] For a very detailed debugging of silicon validation tests

In the following the terms “silicon” and “device” are used for the same object but for different perspectives: “Silicon” is used when the debug of the chip hardware itself is in the focus, “device” in case of tooling for software and system debug.

2.1 Software Development

For software development the tool access is needed for the steps ESL model, possibly an FPGA prototype, the device on an evaluation board and/or for the final target system. Since this setup has by far the most users, the TAA needs to support optimum tool performance.

2.2 Silicon Debug

Functional silicon debug uses similar methods and tools as software debug. A software debugger allows accessing all peripheral registers and supports the user with semantic information about register bits. Silicon debug is started on the wafer and then mainly done with special analysis boards. However an interesting use case is also silicon debug within the (customer’s) target system, for analyzing a potential chip problem (field return) without desoldering.

2.3 Silicon Validation Test Development

Silicon validation tests use, beside downloaded test programs, the access to special hidden hardware registers (e.g. by JTAG scan chains) to stimulate and observe special behavior. This is controlled over the chip’s tool interface by a test sequencer tool. For developing these tests it is very important to have parallel access to the target by the test sequencer tool and a software debugger tool for the downloaded test programs.

2.4 Tool Chain Debug

A reliable and robust tool connection is a mandatory prerequisite for any kind of (test) software development. In particular when the first wafer comes from the fab, the bring-up of the tool connection is in the critical path. A unified TAA allows preparing this bring-up not only with an FPGA prototype but with the HDL simulation already.

3. Tool Access Architecture Requirements

The scope of the unified TAA is from ESL to end product. This range can only be covered, when the tool interface is on software level. The second general question is the abstraction and functionality of the access primitives. For the scope ESL to end product, the access functionality needs to be on the level of processor core run control, triggers and trace and not just on the simple read/write target access level.

3.1 General Requirements

One of the most important requirements for the tool access is reliability and robustness. It is not acceptable for the user if artifacts, created by the tool access, increase the complexity of debugging. Beside the simple requirement that all information retrieved from the target is correct, this also means that the tool operation itself doesn’t cause any changes in the target’s behavior. Otherwise so called “Heisenbugs” can be the result, which only occur, when no tool is connected.

3.1.1 Cost and Performance

For devices with many different customers, like microcontrollers, it is very important for the silicon vendor to have a cost effective tooling solution for evaluation boards. On the other side it needs to be possible to offer high performance TAA implement-tations for more demanding use cases.

3.1.2 Parallel Multi-Tool Access

Figure 1 shows a typical multi-device, multi-core system. For the applications ESL Model and Target System, the TAA needs to support such a scenario. For the other applications (Table 1), the (multi-) tool access is on device level. Please note that there can be not only several instances of a core, but also different core types, coming from different IP vendors and being supported by different tool vendors.

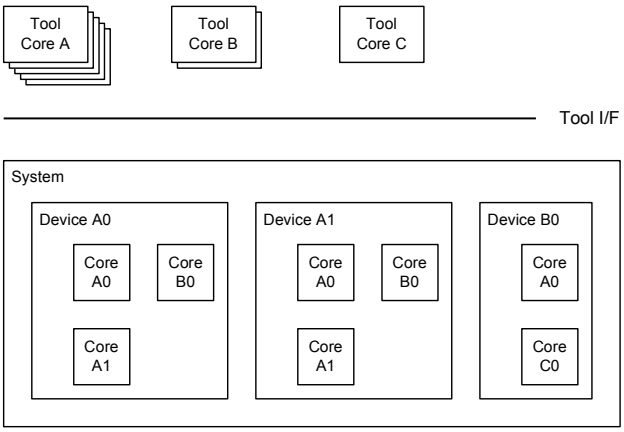

Figure 1: Multi-Device, Multi-Core System

3.1.3 Standard Interfaces

For reducing development effort and cost, standards are always the preference, if they provide the necessary functionality.

3.1.4 Ownership and Testability

It is very important that all components of the TAA have a clear ownership (silicon vendor, tool partner, etc.) and have straightforward unambiguous interfaces, which allow a rigorous testing. Only by careful analysis of what's needed for covering all TAA application features, the interfaces can be reduced to the minimum complexity.

3.2 Special Requirements

Beside the general requirements, which cover the Software Development application, there are specific requirements for other TAA applications and TAA targets (Table 1).

3.2.1 ESL

As already mentioned, the ESL Model application enforces that the TAA tool interface functionality is on the level of core run control, triggers and trace in addition to read/write accesses. The reason for this is that ESL models have abstract representations of the on-chip debug IP, which are more powerful and simulation speed friendly. Without the ESL model requirement, the unified TAA could be restricted to read/write target access without more detailed semantics.

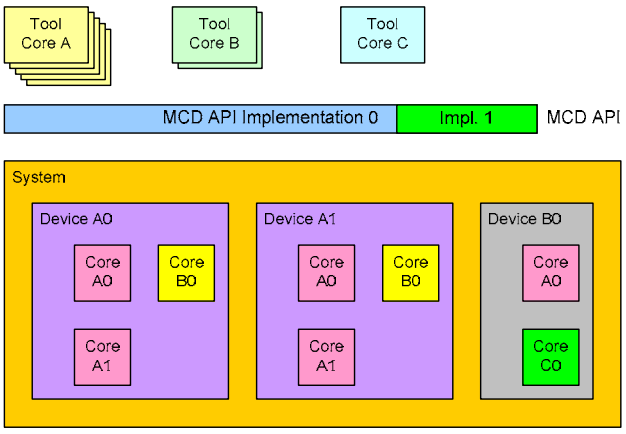

Figure 2: Multi-Vendor IP-Model System

Another ESL specific requirement is shown in Figure 2. The system integrator will be in a situation, to integrate models from different vendors (different colors in Figure 2), connected to tools from different tool vendors. Silicon Debug

Silicon debug requires additional special access methods for hidden test registers. Usually these test registers are implemented as on-chip JTAG scan chains. So the TAA needs to support a native JTAG scan access beside the normal, address based device access methods.

3.2.2 HDL Simulator

The HDL (Hardware Description Language) model of a chip behaves like the real device, except that it is very slow. The TAA needs to take this into account for instance with configurable timeouts.

Since such simulations usually run on Linux compute farms whilst the tool is on a Windows PC, the TAA needs to support connecting these two domains over a LAN.

3.2.3 End Product Debug

Due to cost and space constraints, the end product doesn't have in most cases a tool connector (e.g. JTAG I/F). For getting access to such targets, the TAA needs to abstract the physical interface and support tool access across any type of physical device interface (CAN, USB, etc.), even shared with the target application.

4. Tool Access Architecture (TAA)

The basic tool access architecture is to have a generic standard interface for the different tools and a layered implementation which will differ for the different stages ("Targets" in Table 1) of the system development flow.

4.1 TAA Block Diagram

Figure 3 shows the block diagram for such a generic TAA with the software and hardware layers from device to tool.

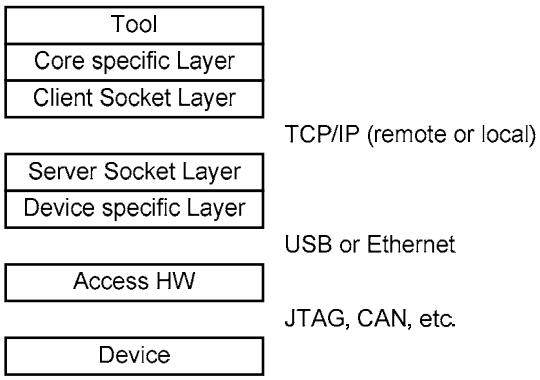

Figure 3: TAA Block Diagram

The device Access HW in Figure 3 can be a simple and cost effective USB to JTAG converter chip or a powerful microcomputer with FPGA accelerator for the physical device connection (JTAG, CAN, etc.) as the other extreme.

The Device specific Layer converts the tool requests like "read X bytes from address Y" in the device specific physical connection command sequences. The device independent communication with Clients is handled by the Server Socket and the Client Socket Layers respectively. In case of the powerful Access HW, the Server Socket and the Device specific Layers can be running on the microcomputer as well, when the connection to the host computer is Ethernet.

The layering on the Client/Tool level starts with the Tool itself on top, which accesses the target across the Core (type) specific Layer. This layer translates requests like "set a SW breakpoint" into the core specific sequence of register and memory accesses. This layer is core type specific, but it is independent from the physical device connection. Only for ESL models usually another implementation of this layer is needed. In this case the Core specific Layer implementation can be even to a great degree core type independent, when the on-chip debug resources have the same abstract representation in the ESL model.

4.2 TAA Concept Considerations

One of the guiding principles for the TAA was to encapsulate specifics as much as possible locally, so that higher layers can stay generic and straightforward. The art is on the other side to avoid making specific but interesting features of the target not accessible anymore due to the restrictions of such a standard interface.

The TAA in Figure 3 encapsulates device type and physical device connection type dependencies in a very low layer. This hides for instance specific start-up behavior of devices with e.g. on-chip voltage regulators. Since such a device's start-up behavior can require hard real-time operation from the tool, it is even mandatory to handle it on this level.

When the device connection is established, all accesses from the higher layers are memory mapped. This is natural for memory mapped registers and memories but it can be also expanded by using virtual addresses for specific accesses and operations. So the same infrastructure and communication scheme can be used for very different purposes. This scheme needs to be extended by the concept of atomic transaction lists. Such a list will be executed completely without being interrupted by requests from other Clients/Tools.

5. Infineon's DAS and MCD API based TAA

The DAS (Device Access Server [1]) architecture was designed for multi-device multi-core systems with very demanding emulation requirements. It is broadly used within Infineon for microcontrollers, but also outside of Infineon it has been adopted in the OCP-IP Debug Interface Specification.

The goal of the DAS architecture is to provide one single interface for all types of tools, which fulfills all performance and reliability needs.

The tool interface is on software level (DAS API) and implemented in a generic DLL. It provides the abstraction of the physical device connection, which becomes just a parameter value in the connection setup phase. During operation the physical connection (e.g. JTAG for real device or directly for C-models) is fully transparent for the tool. On DAS API level the physical device connection is represented by address based accesses (DAS Transaction Lists) and prioritized, stream based data exchange (DAS Channels).

Recently the MCD API support has been added on top of the proven DAS API. The MCD API was specified within the SPRINT project [4] with the partners ARM, Infineon, Lauterbach, NXP, STMicroelectronics and TIMA. It fulfills exactly the requirements specified in section 3. Figure 4 shows the resulting layer structure of Infineon's TAA.

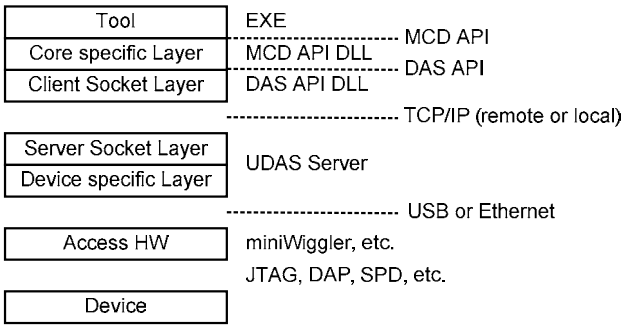

Figure 4 Infineon's TAA Block Diagram

At the moment most tools from tool partners access directly DAS, but it is planned to make gradually the higher level MCD API to the default interface.

In Figure 5 a setup is shown where different tools access different devices. The Access HW is an on-board wiggler or a miniWiggler, which supports JTAG, DAP and SPD. DAP (Device Access Port) is a well accepted Infineon automotive tooling standard interface which enables a high-speed long thin cable. SPD (Single Pin DAP) is the most cost effective (single device pin) tool connection variant.

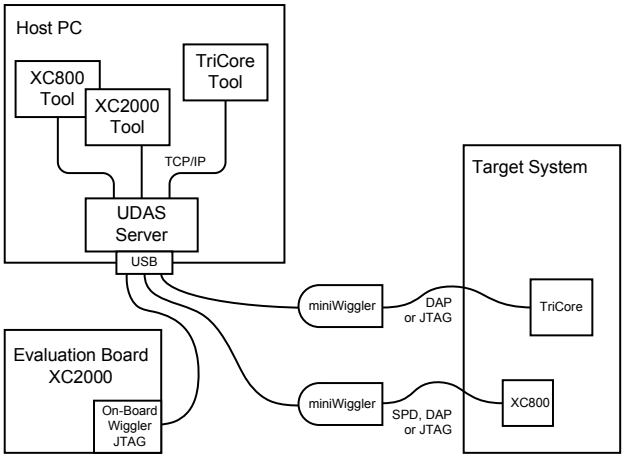

Figure 5: Multi-Device Operation

In Figure 6 a device is accessed by two tools. In this example, one of them is located on a remote computer. The use case behind is debugging of silicon validation tests, which are executed by the test sequencer tool [5]. Another use case example is debugging in parallel to automotive measurement.

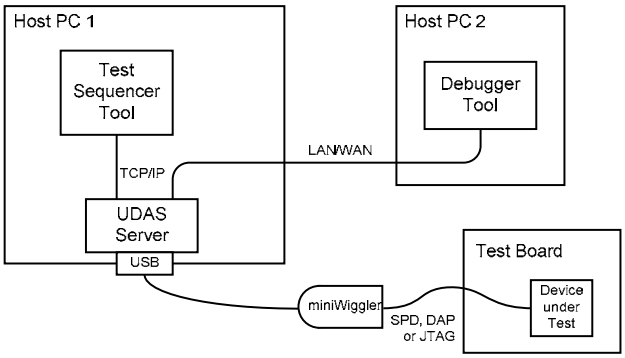

Figure 6: Multi-Tool Operation

The last example in Figure 7 shows that the Infineon TAA hides completely the physical device connection and device representation (real silicon or HDL model). With a Lauterbach debugger it has been demonstrated that beside normal miniWiggler access to a 32-bit TriCore microcontroller also the access over an XCP [5] slave and even to the HDL model, running on a simulation computer is possible. Over the MCD API, all these three devices behave exactly similar, except that the response from the HDL model device is very slow (ca. 10-20s) for a single step

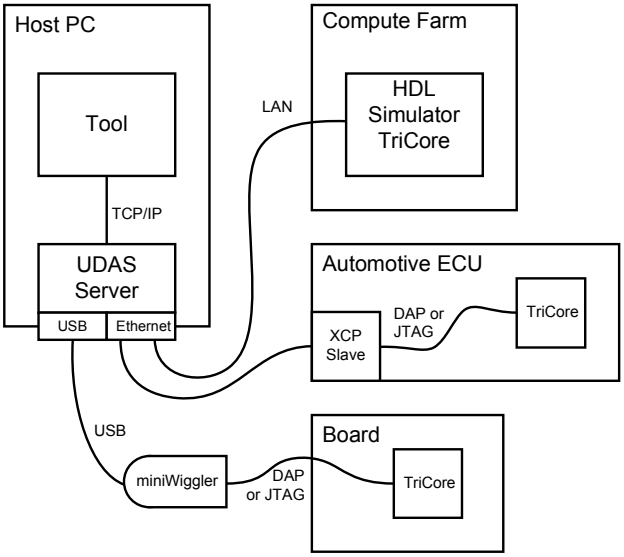

Figure 7: Abstraction of Device and Connection

The Infineon TAA implementation uses a pragmatic approach concerning configurability. In relation to a chip design project, the adaptation of tool access

components is negligible. So it is acceptable to modify TAA components (Core specific Layer, Device specific Layer in Figure 4) directly (C/C++ code) instead of developing highly complex configurable components.

6. Conclusions

A unified Tool Access Architecture for different device types (e.g. 8, 16 and 32 bit microcontrollers), different physical interfaces (e.g. JTAG, CAN, etc.) and different device representations (C-model, HDL simulator, FPGA and silicon) has significant benefits for the silicon vendor, customers and tool partners. These benefits are cost and risk reduction by reuse and the possibility to execute tasks earlier without prohibitive effort. This has been proven Infineon internally using Infineon's own DAS tooling. With the recently released MCD API a standardized tool interface is available now, which allows to cover the full range from ESL to end product.

7. References

[1] www.infineon.com/DAS

[2] MCD API V1.0 specification http://www.lauterbach.com/mcd_api.html

[3] OCP-IP Debug Interface Specification V1.0 www.ocpip.org

[4] SPRINT Project www.sprint-project.net

[5] European patent EP1349073B1, "Control system"

[6] ASAM MCD-1.XCP standard , "The Universal Measurement and Calibration Protocol Family" http://www.asam.net