MIPI Alliance Test and Debug - NIDnT-Port

White paper

Approved Version: 1.0 – 10 January 2007

MIPI Board Approved for Public Distribution

NOTE and DISCLAIMER:

This is an informative document, not a MIPI Specification.
Various rights and obligations which apply solely to MIPI Specifications (as defined in the MIPI Membership Agreement and MIPI Bylaws) including, but not limited to, patent license rights and obligations, do not apply to this document.

The material contained herein is not a license, either expressly or impliedly, to any IPR owned or controlled by any of the authors or developers of this material or MIPI. The material contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material and MIPI hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THIS MATERIAL. IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR MIPI BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT RELATING TO THIS MATERIAL, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

MIPI™ is a jointly owned trademark.
Copyright © 2007, MIPI Alliance Inc.

Questions pertaining to this document, or the terms and conditions of its provisions, should be addressed to:

MIPI Alliance, Inc.
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Board Secretary

1. Introduction

1.1. SW Debug

Until now debugging standardization groups focused on prototype and early device bring up debugging. SW development was either done by emulation, or run-mode debugging over serial interfaces.
Tracing and debugging SW is difficult using only normal output devices such as display or serial interfaces that come out of the processor. Therefore the processors have dedicated interfaces for tracing and debugging. They are frequently used during the R&D phase and with some limitations also when the products are in the market.
When a product matures and is put inside its actual mechanics these standardized tracing and debug interfaces are left inside these covers and cannot be accessed anymore. If some bug or fail is discovered at the late phase of the project, it is difficult to trace or debug the software since there is no access to these dedicated trace & debug interfaces.
Additionally to standard debugging, the interface is also be useful for 3rd party SW developers, for field testing and by operators to observe the whole system (e.g. terminal device + network), which could help them identify cellular network issues.
To meet this needs, the Mobile Industry Processor Interface (MIPI) Test and Debug Working group is exploring a maintenance port, called NIDnT-Port (**N**arrow **I**nterface for **D**ebug a**n**d **T**est: Speak NIDENT). NIDnT is based on IEEE P1149.7 and the System Trace Module, which includes the MIPI System Trace Protocol (STP) and uses the MIPI Parallel Trace Interface (PTI) for data export.

1.2. Silicon Debug

The number of gates in integrated circuits will roughly double in 18 months. Large ASICs can have tens of millions of transistors. The transistor size is very small but the density is high. This means that there isn't any possibility to observe the internal behaviour of the chip without dedicated test structures inside the ASIC. The visibility to the ASICs internal state is very limited, which complicates ASIC HW debugging. Also the visibility to external operations and many trace & debug points inside the ASIC is limited.
However, guess and try is still the most used "debugging method". When a hypothesis about the root cause of a problem has been formed, the same test has to be repeated in a simulation environment to get some hint of the internal operations of the ASIC. Unfortunately this is not always possible, since a simulation environment typically behaves slightly different from the actual operating environment and does not model all external components. In addition, simulation speed limits the test case length. The most difficult cases happen rarely (e.g. only once a day) and thus are impossible to simulate.
In some cases, the problem is very difficult to find out. It is unknown whether the bug is in SW or HW as well as the exact location of the problem is difficult to track. An example could be a bug where the data will corrupt between the CPU and the external memory interface. Data from the external memory can be measured with a logic analyser and the data from the CPU registers can be read with a debugger by using a JTAG connection. Unfortunately there is no way to see where the data is corrupted in between. This means that the problem cannot be tracked to a smaller portion of the design. This is starting to be an increasing problem since the complexity and the amount of logic inside the chip has increased. To find such problems, we need to increase visibility to the ASIC internal flip-flops.

1.3. NIDnT-Port

The NIDnT initiative aims to bring debug, trace and test capabilities to end terminals at virtually no costs. This goal is achieved by reusing already existing external interfaces. An implementer could split the test & debug (T&D) interfaces over several interfaces (e.g. MMC card slot for trace and USB for basic debug). In this whitepaper, we talk about sharing an external memory port such as the micro SD card slot with standard T&D interfaces. Additionally we talk about different system level options such as networking

2. Terminal Views

2.1. Single chip devices

A simple board might consist out of one single ASIC like shown in Figure 1. The chip has a dedicated port for debug and trace, which might consist of several different interfaces. Most end products will not have this port available due to design constrains, costs and security restrictions. NIDnT-Port provides access to existing T&D interfaces, where security concepts already exists. Therefore the already existing security concepts apply to this port also and don't introduce a new thread. This paper doesn't cover such security aspects. MIPI T&DWG deals only with the debug and test aspects.

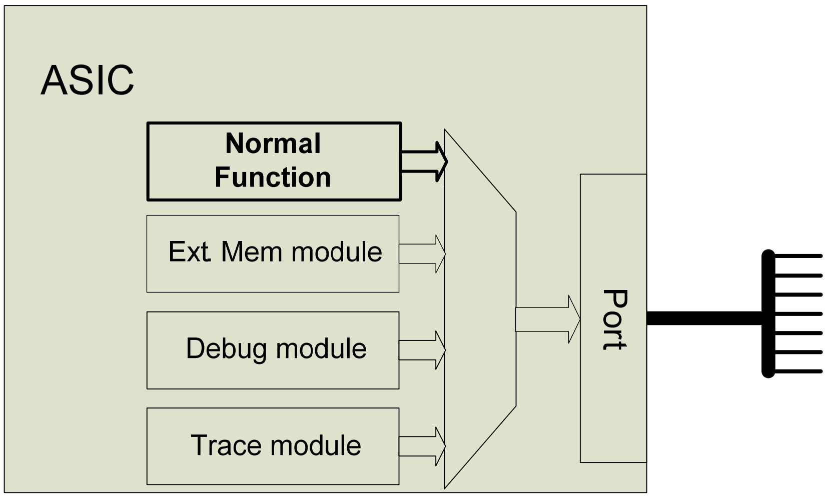

Figure 1: Single Chip Solution

Such an ASIC might offer the following interfaces for test and debug:

- Basic debug link (IEEE 1149.1/ P1149.7 or ARM Ltd SWD)
- Instrumentation trace (MIPI STP over MIPI PTI or NEXUS)
- Program flow trace (e.g. ARM Ltd ETM; NEXUS Trace)
- Serial communication link (NEXUS AUX, UART type, USB)
- Parallel production test interface
- Silicon debug link (IEEE 1149.1)
- Other standard or proprietary T&D interfaces

In this paper we explore the reuse of an external memory interface (e.g. Micro SD) with an STM, implementing the MIPI STP protocol and IEEE P1149.7. Other combinations would also be possible, but are not considered in the MIPI application space.

2.2. Multi chip devices

Today many mobile terminals have dedicated ports per chip, comprising several interfaces, like shown in Figure 2. This is easy to implement and ensures highest visibility into each chip. On the down side, it's an expensive debugging solution for end products, as it requires many pins, several connectors and large board space. Therefore end products will not implement such a solution.

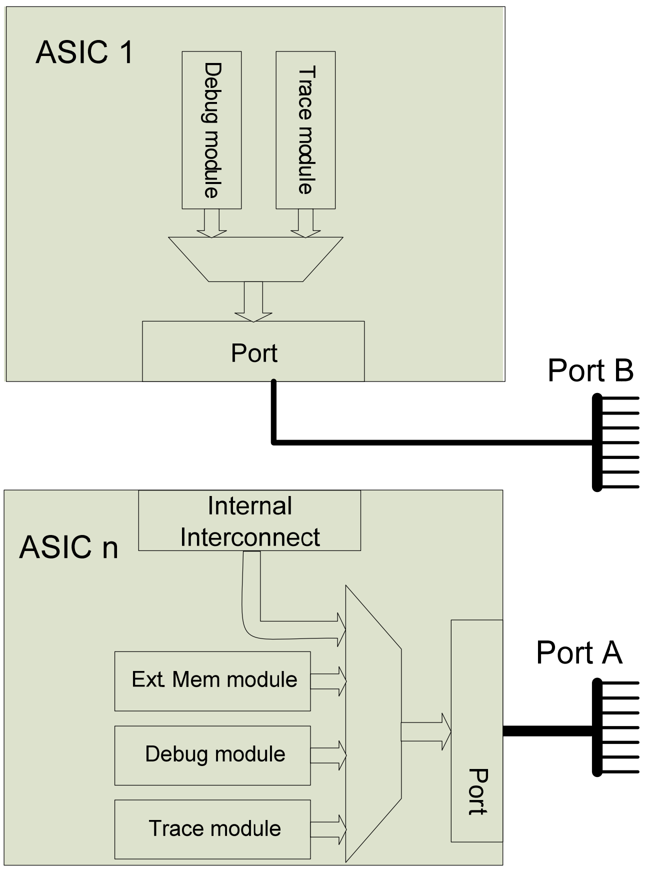

Figure 2: Multi Chip System: Dedicated Ports

To overcome some of the issues shown in Figure 2, mobile terminals might only implement one debug port. This could be the entry point to only one chip (i.e. one of the chips is assumed to be a black box and not debuggable) or as shown in Figure 4 to the whole system.
To gain more visibility a networking approach has to be taken. I.e. the sharing of the port could be done by a real multi drop communication link like e.g. IEEE P1149.7 for standard debug and by tri-stating the trace outputs of the chip not under debug (See Figure 3). I.e. the basic debug communication would work similar to a network structure in parallel to all chips in the system (broadcasting), but the trace pins would only be assigned to one ASIC in the system at the time. The DTS would work as a master of the system.

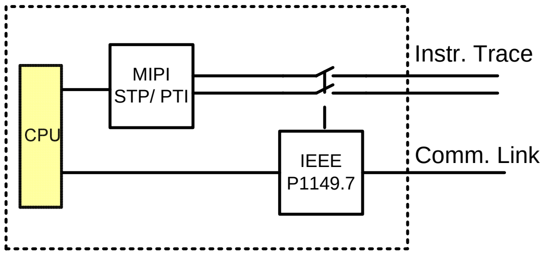

Figure 3: Broadcast Interface

Such a broadcast T&D Interface would allow building a flexible structure in which IEEE P1149.7 is connected to all ASICs in the system to provide basic debug communication. If a higher class of this interface is implemented, it can also be used for background data exchange (BDX). I.e. if only lower bandwidth is needed, the basic debug communication link can be used also for multi-drop instrumentation trace.
If higher trace rates are needed, the System Trace Module, implementing the MIPI System Trace Protocol can be configured and activated. But there is only one ASIC that can output higher speed trace data at a time. All other chips have there dedicated instrumentation trace interface tri-stated.

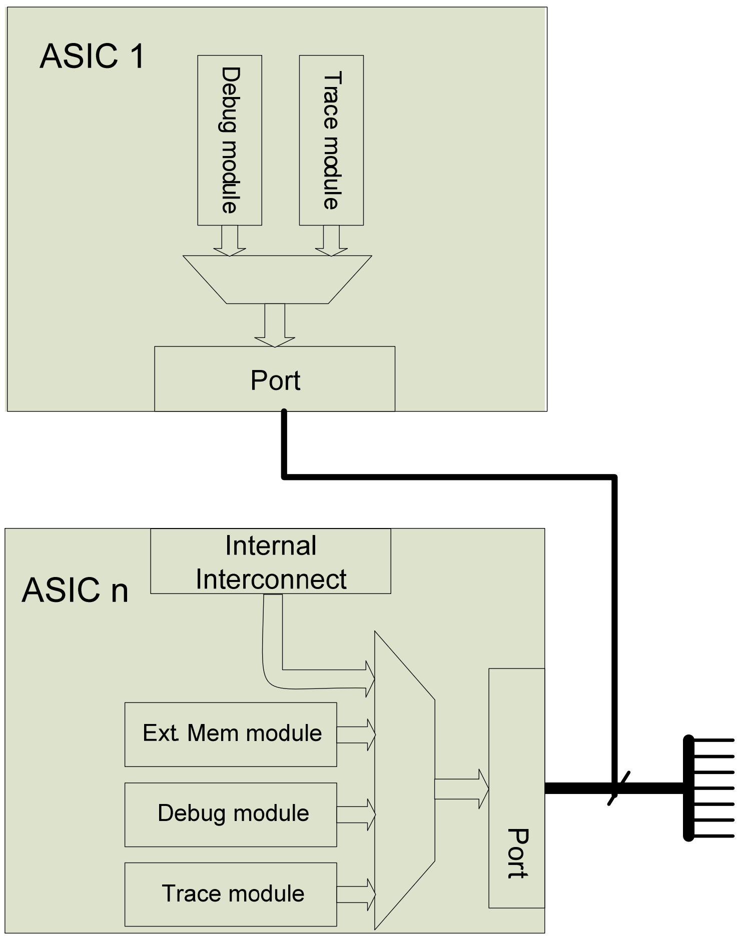

Figure 4: Multi Chip System: Shared Port

If the ASIC pins are not the limiting factor, a chip to chip debug link like shown in Figure 5 should be considered. This will allow full system visibility, but of course system latencies have to be taken into account.
The debug link could either be a dedicated standard debug link or tunnelled over other, already existing high speed links.

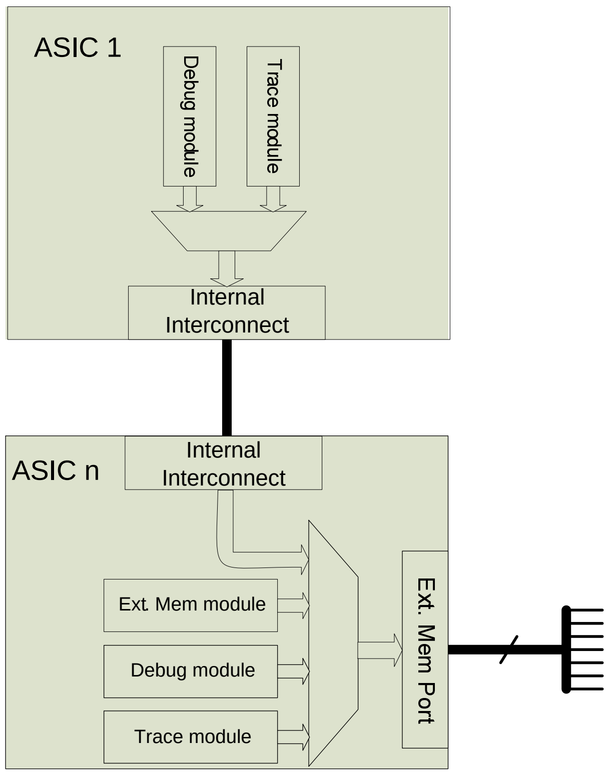

Figure 5: Multi Chip System: Single Port

Copyright © 2007 MIPI Alliance Inc. All Rights Reserved.

3. Integrated Hardware/Software Debug Environment

Current debug systems focus mainly on debugging the software that runs on top of one or more processors in the chip. By using an open debug API, it is possible to use multiple debug tools to access the same target (See Figure 6). This is especially useful for SoCs that contain both processors, for which SW debuggers are commercially available, and proprietary processors, for which internal SW debuggers exist and need to be used.

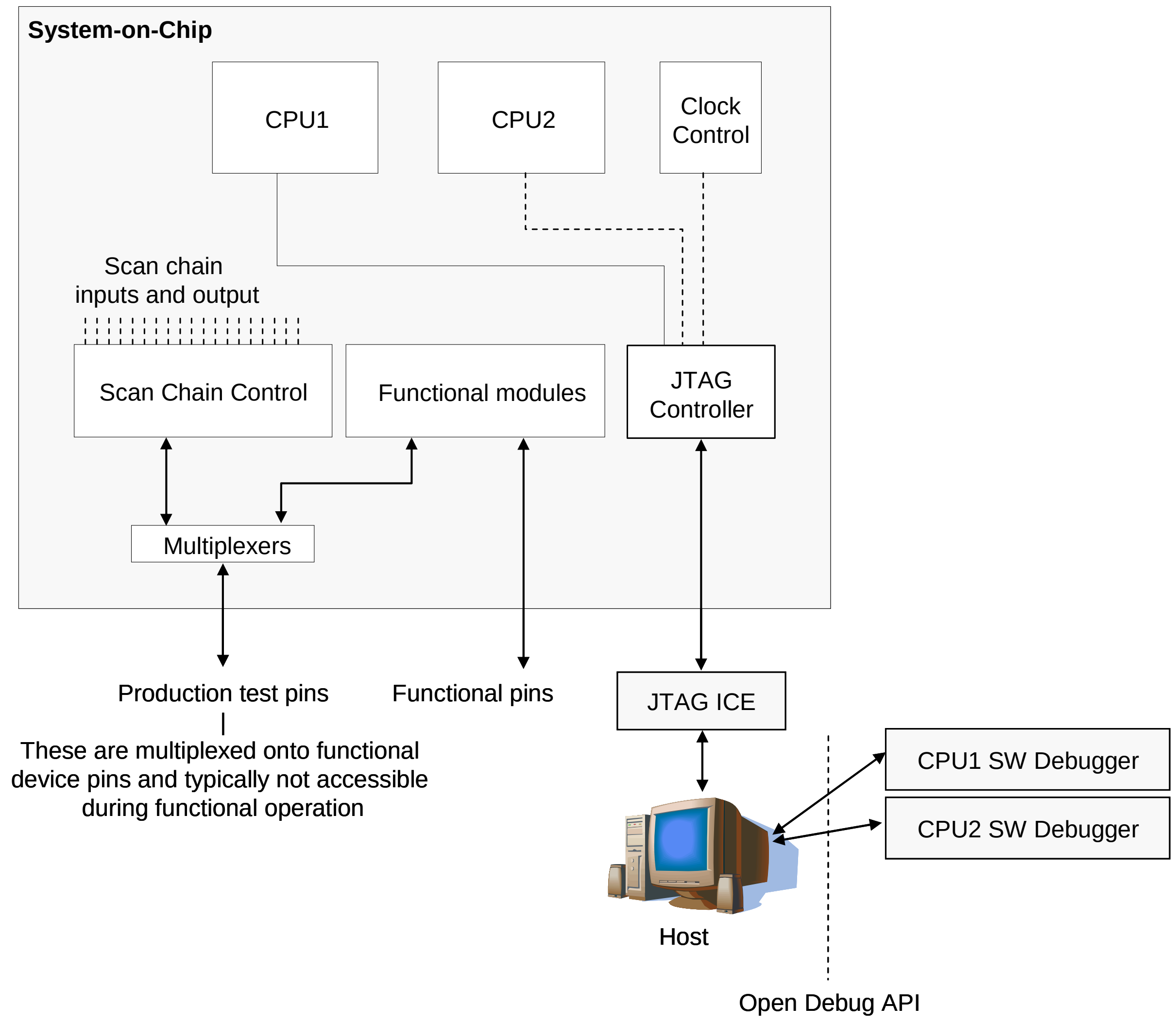

Figure 6: Current Software-only Debug Environment

The SW debuggers provide detailed information on the execution of the software, in relation to the programmer's model of the processors. This software view is however often not sufficient to properly and quickly localize the root-cause of a system failure. This is because these systems do not include a view on the underlying hardware, and/or on the hard-wired peripherals connected to the processors via a system interconnect. To provide this hardware view an extension to these software-only debug systems is required, together with minor modifications to the SoC design.

Many SoCs have scan chains implemented to support manufacturing test. These scan chains are typically routed through all flip-flops in the chip. During manufacturing test, these scan chains are accessible from the device pins to provide full control over the state of the SoC's internal registers. This access allows Automated Test Pattern Generation (ATPG) tools to achieve a very high fault coverage (>90%) in a short amount of time.

Copyright © 2007 MIPI Alliance Inc. All Rights Reserved.

The scan chains are normally only used in manufacturing test mode and are generally not (made) accessible in functional operating mode. During system bring-up however, these scan chains can also be very useful for SoC internal debugging, when their contents can be output via a serial interface like the IEEE P1149.7 interface.

The main approach is therefore to make all flip-flops in the scan chains visible to hardware debug tools via the same debug interface as used by the software debug tools (See Figure 7).

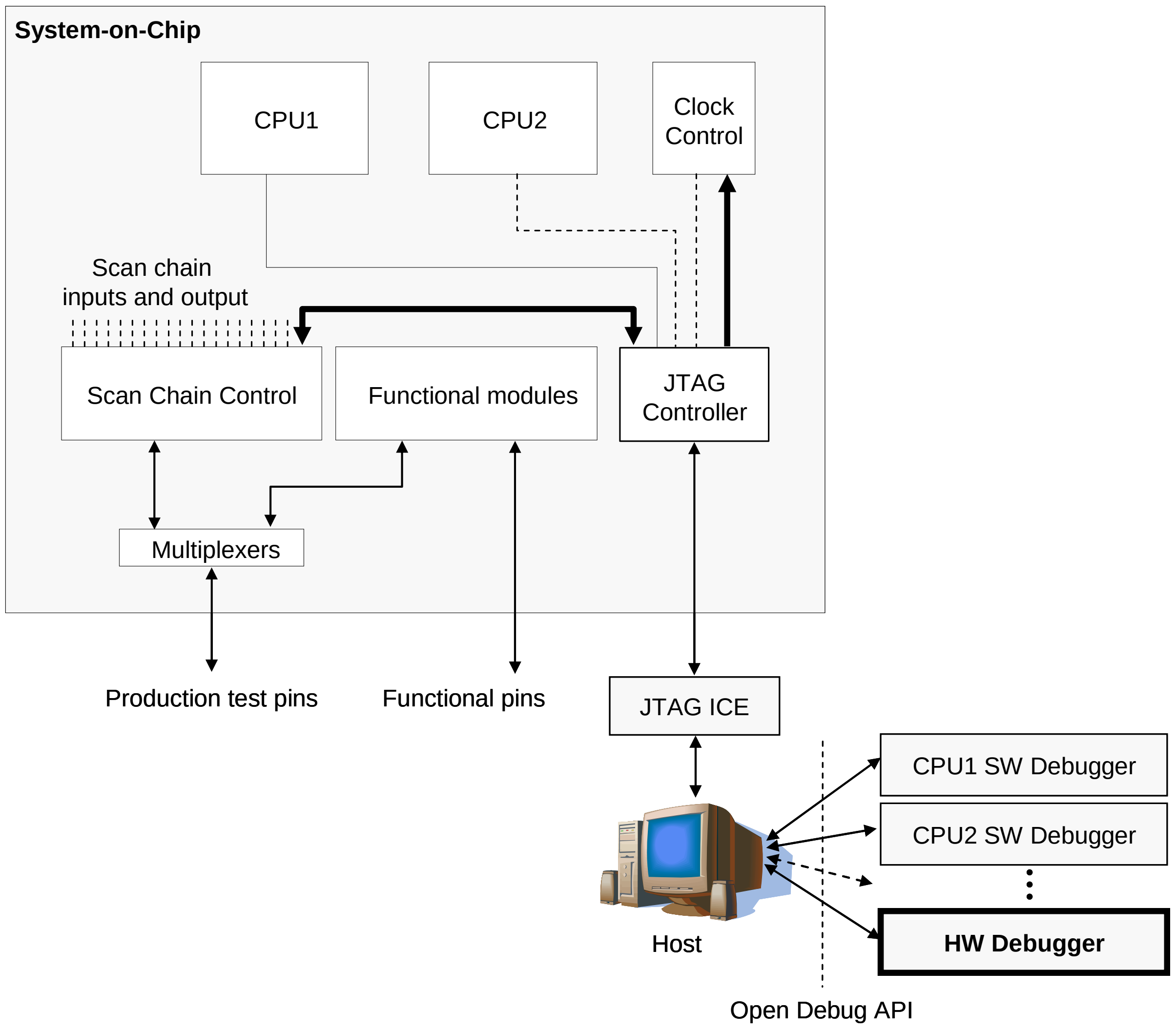

Figure 7: Integrated Hardware/Software Debug Environment

Best case, the hardware debug tools can control the content of each flip-flop. This can be implemented by allowing a TAP controller to access the scan chains in functional mode as well. An external software or hardware debug tool, or a CPU inside the SoC can control the point in time when the scan chains need to be output. As a result, it is possible to scan out and investigate the content of any internal register at the time the debugger tool(s) or CPU initiates the trigger. This improves low-level, hardware visibility of the ASIC's internal operations. The biggest benefits are expected in hardware debugging of real products and prototypes.

In a system debug environment, the hardware debug tool would first select the scan chain to be captured and then signal the start of the capture operation to the chip. This will halt the complete system, effectively freezing the flip-flop values. After the system is halted, the system can be switched into a production test debug mode. In this production test debug mode, the flip-flops form long scan chains, which can be selectively shifted out via the debug interface. The debugger can thus get a snapshot of the complete,

Copyright © 2007 MIPI Alliance Inc. All Rights Reserved.

current state of the flip-flops. A standard debug interface, e.g. IEEE 1149.1 TAP, is preferred since it is always available in the system environment. The shifted values are stored in an external device such as a debugger or data logger. The order of the flip-flops in each scan chain is known from the design database. Based on the scan chain number and the location of the flip-flop in the scan chain, each flip-flop's content can be identified. This allows to trace the data value from almost all flip-flops and thus seeing the instant data of the system.

This state dump data can be compared to expected and/or simulated state data, to help find the root-cause of a difference between the real system and the expected/simulated behaviour. This approach greatly supports finding the exact location of the problem and thus shortens hardware debug time. This method is useful especially, when the problem is difficult to find, and/or when it is not clear if it is in HW or SW.

In this manufacturing test debug mode, values can also be serially loaded into the flip-flops to then turn the system back to functional mode. This allows changing e.g. one flip-flop value and test if it has any effect on the problem.

4. Connectivity

As already shown, the NIDnT connectivity focus is on combining already existing interfaces. This is achieved by e.g. overlaying an external memory interface port with the combination of IEEE P1149.7 and System Trace Module (STM) implementing the MIPI STP protocol.

The IEEE P1149.7 link is not only used for basic debugging, but also as a back channel for the STM, which is used for instrumentation trace.

A converter from the external memory port (e.g. micro SD) to the MIPI Basic Debug Connector (e.g. 20 pin header as defined in the MIPI Alliance Recommendation for Test & Debug – Debug Connector) will allow to debug the end terminal with standard debug tools. The pin out can either share the clock from the system trace interface with IEEE P1149.7 or run completely independent (Examples see Table 1). This is a design consideration. If the focus is on system trace (i.e. higher bandwidth is needed), a shared clock might be chosen. However, this will slightly reduce the peak performance of IEEE P1149.7.

Copyright © 2007 MIPI Alliance Inc. All Rights Reserved.

Table 1: Example Connector Pin-out for Micro-SD card

Micro SD card pin	**Debug Interface**		**MIPI basic debug connector**	
	Figure 8	Figure 9	Figure 8	Figure 9
CLK	TCK (shared with STM)	TCK	#4	#4
CMD	TMSC	TMSC	#2	#2
Data 0	STM D0	STM D0	#14	#14
Data 1	STM D1	STM D1	#16	#16
Data 2	STM D2	GND	#18	GND
Data 3	STM D3	STM CLK	#20	#12

Table 2 shows one pin-out of the MIPI basic debug connector, which provides all needed functionalities for NIDnT.

Table 2: MIPI Basic debug connector: Narrow JTAG with trace pinout

VREF_DEBUG	**1**	**2**	TMSC
GND	**3**	**4**	TCK
GND	**5**	**6**	BCE
KEY	**7**	**8**	EXT
GND	**9**	**10**	nRESET
GND	**11**	**12**	TRC_CLK
GND	**13**	**14**	TRC_DATA0
GND	**15**	**16**	TRC_DATA1
GND	**17**	**18**	TRC_DATA2
GND	**19**	**20**	TRC_DATA3

5. System Overview

As mentioned in chapter 4. the pin-layout of NIDnT-Port may vary according to it’s main use case. Figure 8 puts its focus on maximum bandwidth for instrumentation trace. This is achieved by using a maximal trace data width (width =4 in this example) and sharing the trace clock with IEEE P1149.7.
An external receiver/ bridge will work as interface to a standard host serial link.

Copyright © 2007 MIPI Alliance Inc. All Rights Reserved.

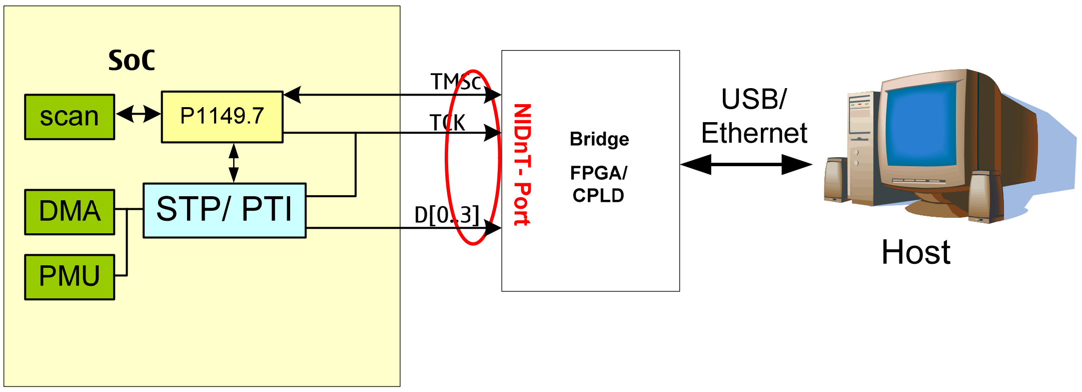

Figure 8: Priority on instrumentation trace

If the focus is more on the basic debug connection, NIDnT-Port should implement independent interface clocking. I.e. IEEE P1149.7 and MIPI PTI clock run independently, like shown in Figure 9. This might force the implementer to reduce the amount of data pins (e.g. to two in the example below). The host connectivity would be done like in the example above.

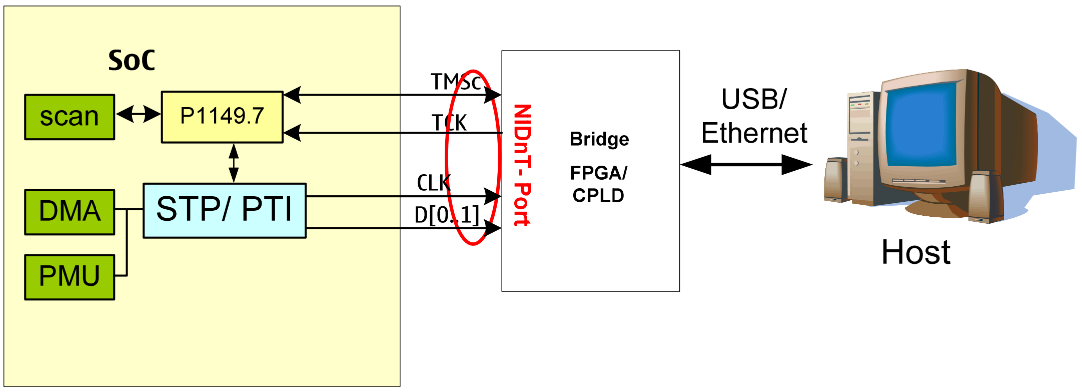

Figure 9: Priority on basic debug link

A third option might be, that the protocol conversion is not done externally to the target device, but integrated into the SoC. This is basically achieved by converting or merging the debug protocols onto a standard serial interface link like shown below in Figure 10

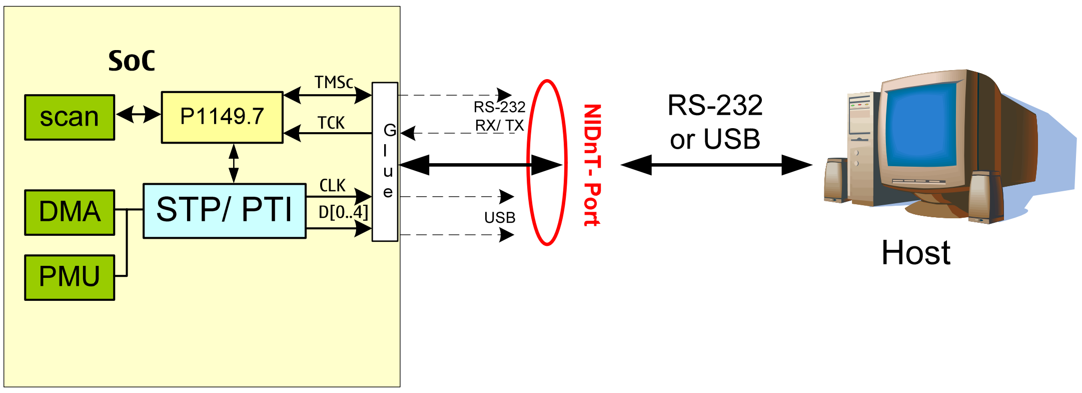

Figure 10: On-chip bridging

6. Limitations

Using the external memory port for maintenance, forces NIDnT-Port to do compromises on the link speed. Many systems will have an external level shifter and potentially serial resistors to help with EMC issues and short circuit protection like shown in Figure 11 and Figure 12.

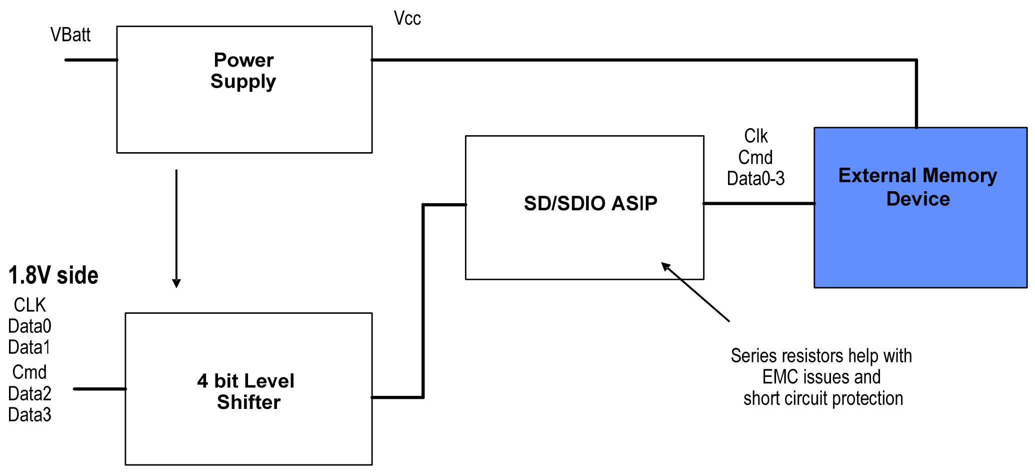

Figure 11:SoC external protection circuitry Overview

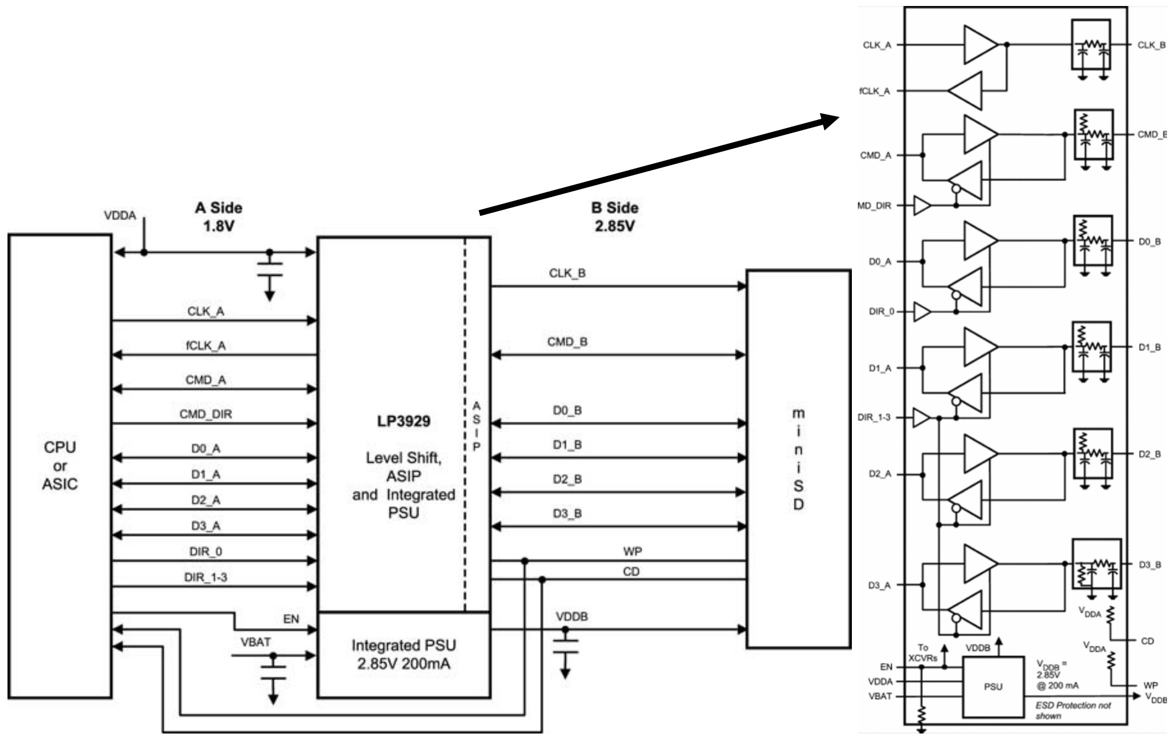

Figure 12: High Speed Bi-Directional Level Shifter (by National Semiconductor)

The direction selection mechanism of the level shifter needs special attention to get the memory interface also functional for test and debug. This could be achieved either by disabling, bypassing or enhancing the level shifter and protection mechanism.

Below in Figure 13 you see one possible solution to enhance e.g. the CMD path to be truly bidirectional and therefore support IEEE P1149.7. But this implementation has a big impact on the throughput, as it's supporting less than 2 MHz transfer frequency.

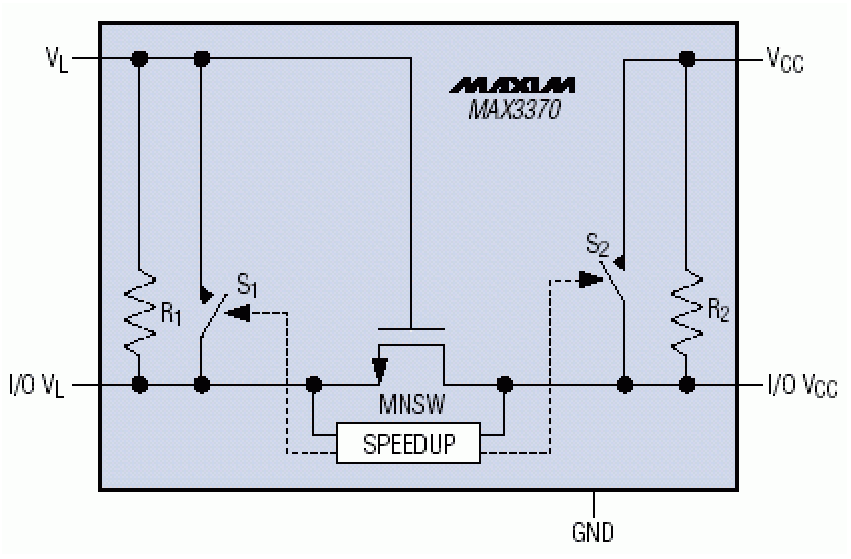

Figure 13: Transmission-gate method of level translation (by Maxim Integrated Products)

Currently external memory interfaces are designed for a maximum clock speed of 52 MHz. With this frequency limitation the STM throughput would be 26Mbytes/sec with a 2 pin DDR interface or twice as much in case the 4 bit wide interface runs efficiently.

Obviously using an already existing interface for test and debug, will make some debug and test use cases impossible to execute. This can be partly overcome by tunnelling the functional protocol over the T&D protocols and might introduce unwanted side effects.

7. Conclusion

The very high integration density forces us to merge old and new technologies. Debugging (including tracing) and testing are treated too independently. The very limited interconnect and board space in end terminals need to be addressed better, sharing interfaces will become mandatory.
NIDnT is not aiming to replace current technologies such as debugging via a SW monitor over a high level serial interface. NIDnT is aiming to implement a stable, low level but high performance interface with minimal costs to the final product. It tries to combine debug, SW trace and HW test.

Copyright © 2007 MIPI Alliance Inc. All Rights Reserved.