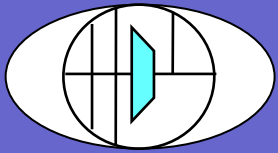


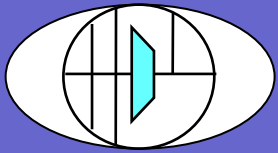
On Chip Instrument Application to SoC Analysis : Challenges and Approaches

Dr. Neal Stollon, HDL Dynamics
neals@hdldynamcs.com



Topic Overview

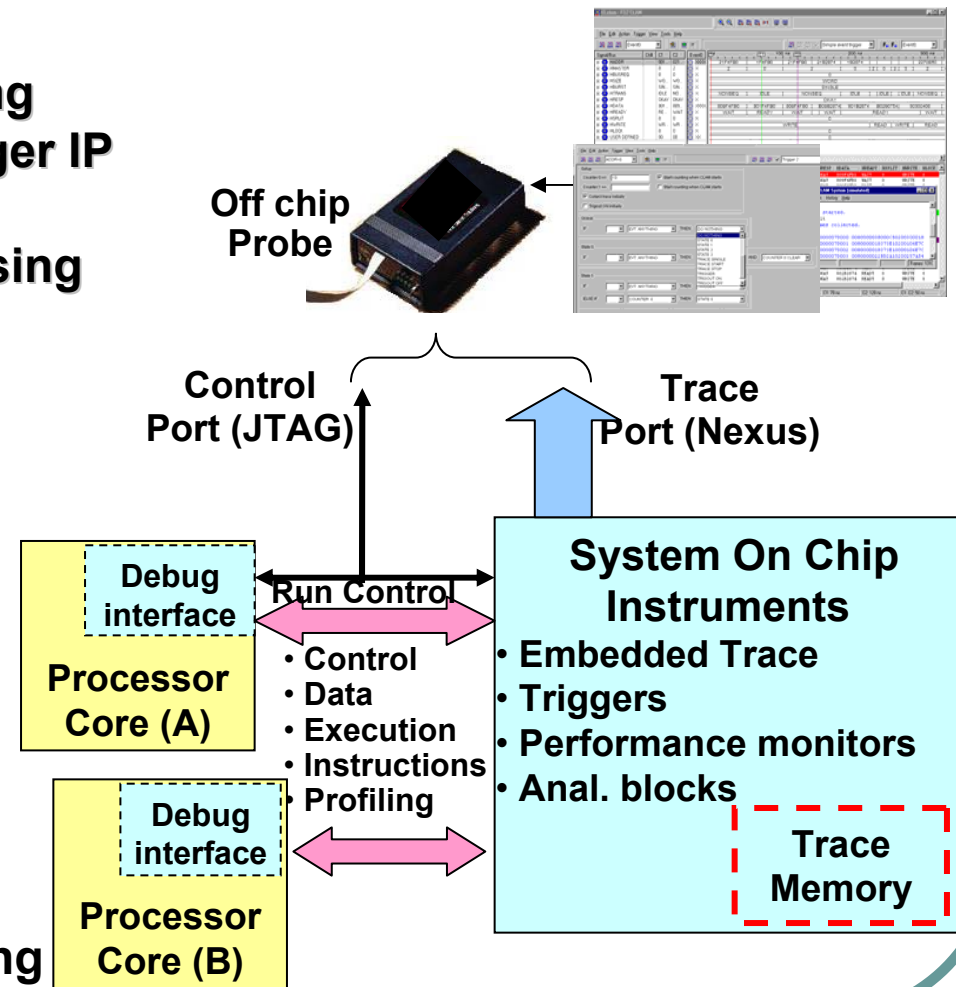
- So What is On Chip Instrumentation
- Challenges and Issues
- Emerging Industry Efforts
- Example Application Solutions

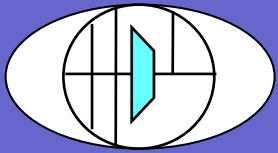


What is Instrumentation?

On Chip Instruments

- **Logic Event monitoring using**
 - Parallel/Serial/User Trigger IP
 - Assertion Checker IP
- **Processor & bus analysis using**
 - Bus Tracer
 - Bus Range Checker IP
 - Processor Run Control
 - Instruction Trace
- **Storage of trace data**
 - Transaction Registers
 - History Registers
 - Trace RAM
- **Post capture tools**
 - Fault Finder
 - IP map back to RTL
 - Instruction/cache profiling





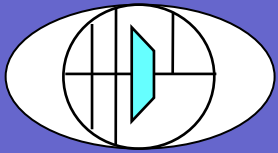
On Chip Instrument in 2008

Lots of Design for Debug standards activity

- OCP-IP has released Debug Interface spec
- Power.org has released Debug spec
- SPRINT has released Debug API spec
- Nexus 5001, 1149.7 set to release specs by early 2009
- Interest in emerging areas of instrumentation

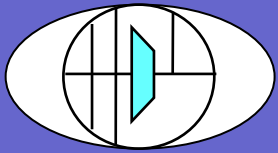
Lots of DfD industry activity

- Recognition that SoC requires embedded analysis capabilities
- More on-chip debug IP and interfaces are now the norm for higher end embedded parts



Challenges and Issues

- **Challenge 1 - User Perception**
Design Community: Debug capabilities as key features
Perception: "Real engineers don't need to debug"
Research Community: Debug as an area of formal analysis
Perception: "Debug is ad hoc"
- **Challenge 2 – Intra-Domain Integration**
Better multi-vender SoC debug Interoperability
"Adhering to standards"
- **Challenge 3 – Inter-Domain Integration**
Integration with EDA, test, software worlds for win-win SoC solutions
"There are no islands in SoC design"



Challenge 1

Design for Debug perception as a necessary evil
vs. value added feature

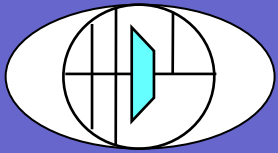
*“Debug features are only needed by engineers who
make errors!”*

Limited users for HW debug = Why waste silicon ??

Debug does add its own complexity

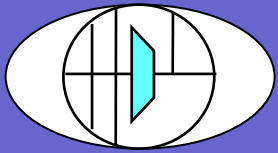
Debug infrastructure bar goes up with complexity

RAM vs. IO Bandwidth tradeoffs

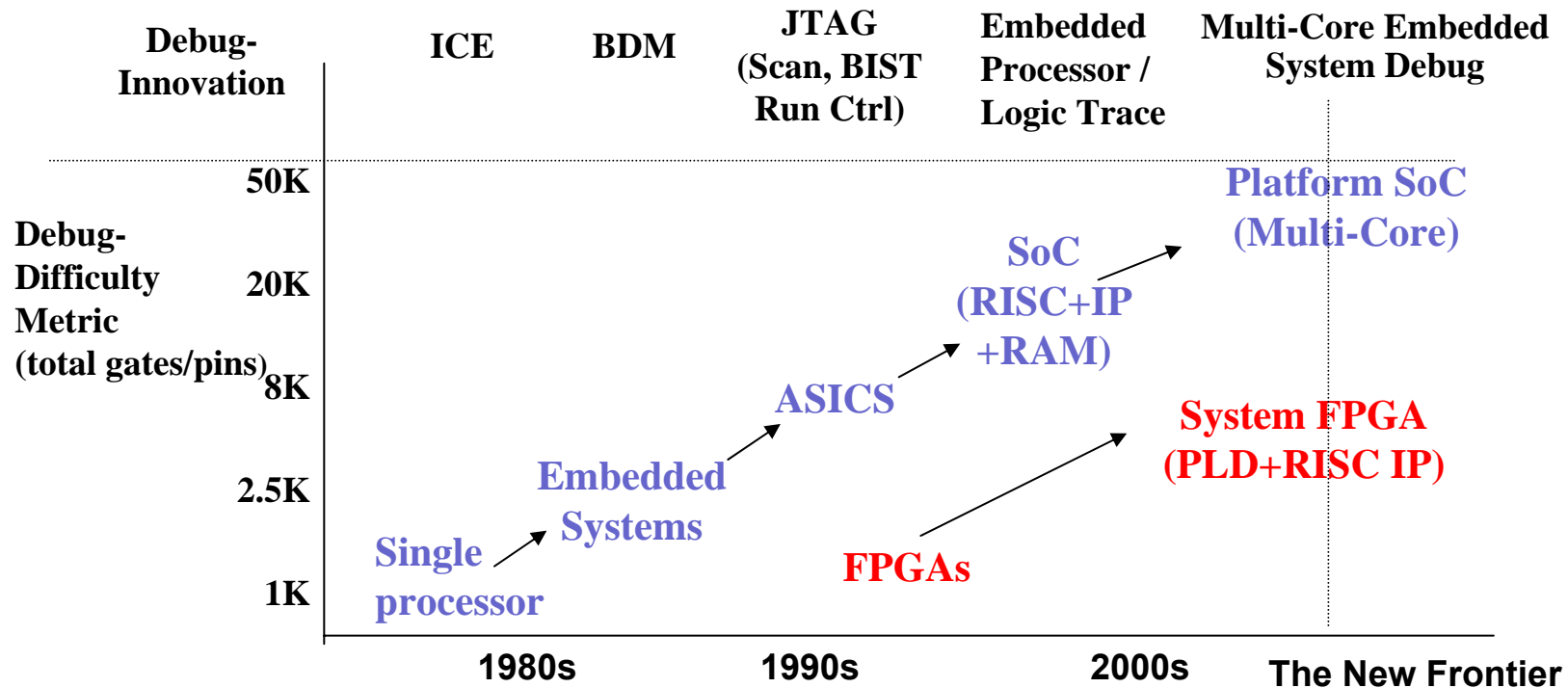


Challenge 1

- More painless solutions (automation)
- Software is king
 - Better visualization
 - Localizing problems
- HW should be interoperable
 - Migration from JTAG
- 3rd party value vs. in-house solutions

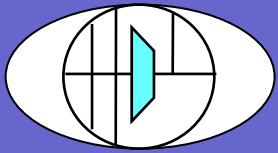


SoC On Chip Debug Evolution

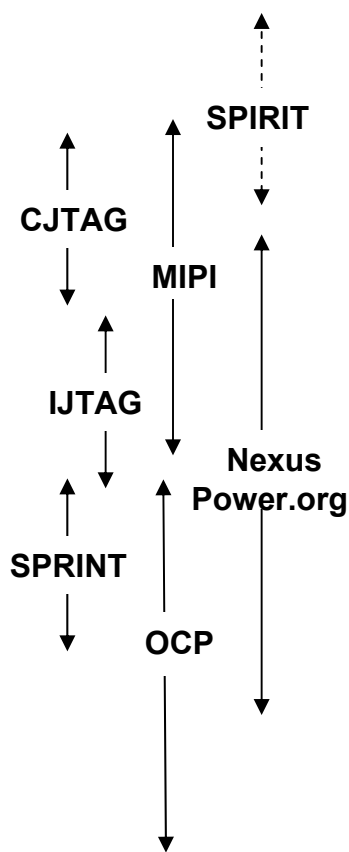


Embedded Debug Complexity increasing on log scale

- Gates increase geometrically - Pins increase linearly
- Significant debug difficulties for complex architectures
- More complex debug needs new Instrumentation approaches



The Debug Landscape

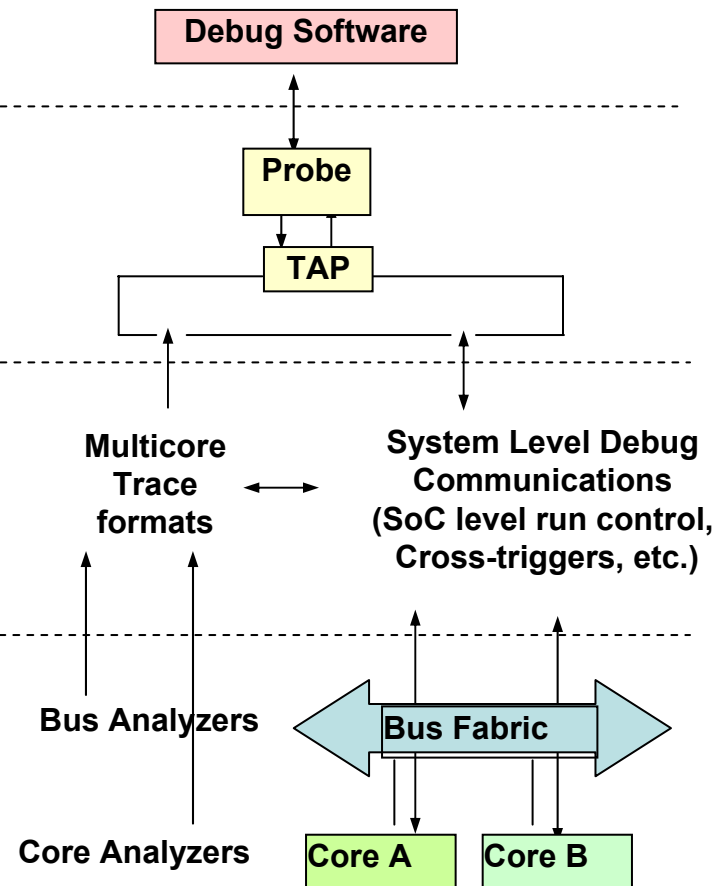


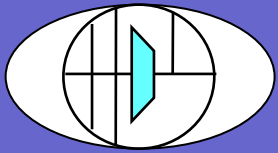
Tool Level of Integration
 • SW/EDA Tool level APIs
 • HW tools level APIs

IO Level of Integration
 • JTAG – different flavors
 Nexus 5001
 MIPI NIDnT

Protocol Level of Integration
 • Nexus 5001
 Multicore IP & probe APIs
 Other Proprietary

Instrument Level
 Many vendor/IP solutions
 CoreSight™
 MIPS EJTAG
 MCDS
 Nexus





Challenge 2

The DfD community is internally fractured

Many groups reinventing the same wheels

Little to no inter-company commonality on instrument interface/tool feature/API infrastructure

MIPI Roadmap

Nexus Roadmap

OCP-IP Roadmap

POWER.ORG Roadmap

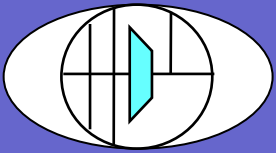
SPIRIT Roadmap

...

Does this converge to a
DfD Industry Roadmap ?

Will design community adopt
Without an Industry Roadmap ?

What is driving incentives to common interfaces??



Challenge 2

Many Roadmaps vs. Industry roadmap

Diverging IO ...parallel, Serdes

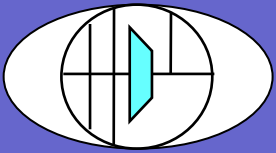
Differing processor trace capabilities, formats

Differing bus debug capabilities

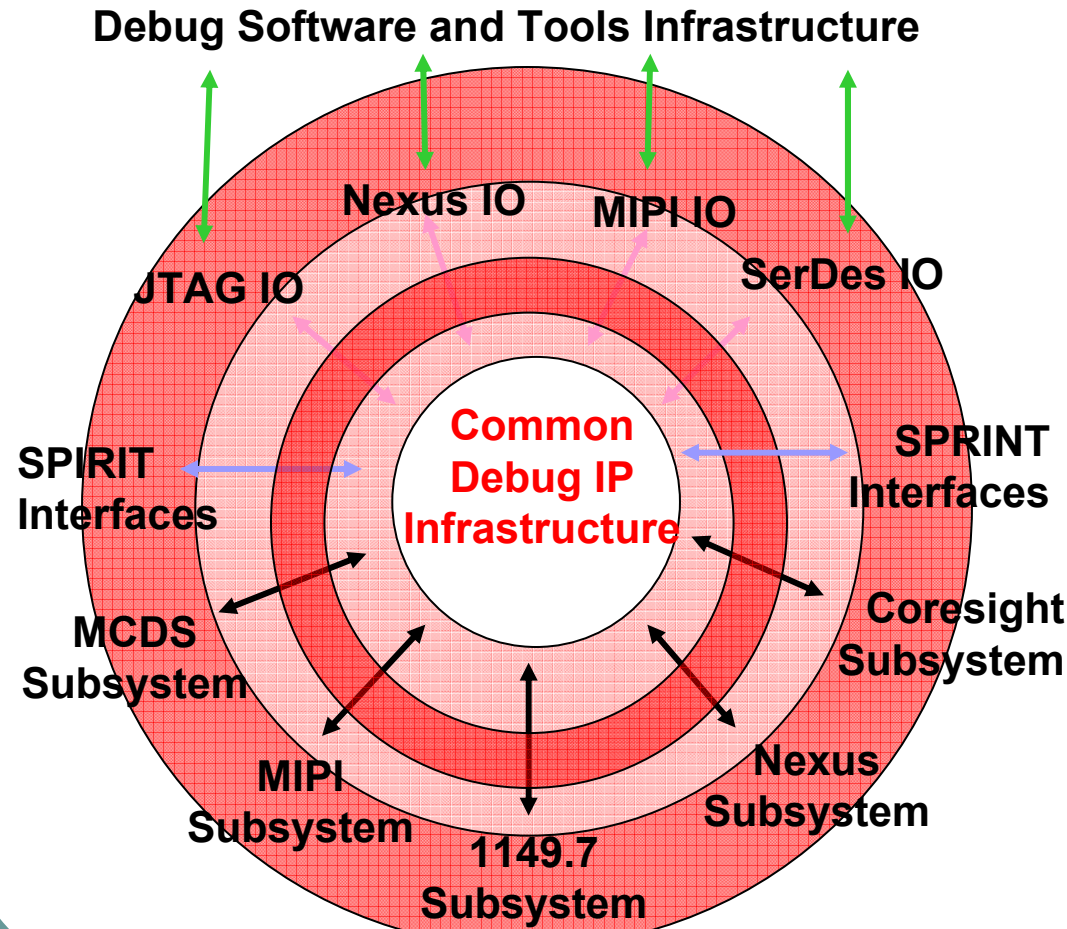
Emerging debug solutions for Multicore

Common, low cost debug solutions for low end MCUs

Basic MCU in 5 years will be the complex SoC of today



The Target : Hitting Range of Debug Solutions

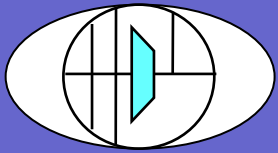


REQUIREMENTS

- Define standardized
- Tool to tool debug APIs
 - Tool to HW APIs
 - EDA to Debug Interfaces

- Define standardized
- Debug IP to Debug IO
 - Templates – Decouple Debug IP from debug IO

- Define standardized
- Hardware protocols
 - Common debug interfaces
 - Level set of features
 - Compatible with many system debug options



Challenge 3

System and SoC Debug is an island to the greater EDA/Test community

Integration with other aspects of design and verification flow are currently limited and adhoc.

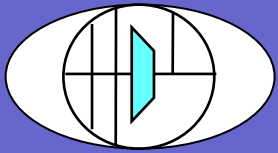
Why – SW centric focus, need for HW IP, lower margins

Debug breakthroughs will occur in conjunction with EDA community

EDA vendors are starting to see the opportunity beyond emulation

Synplicity – Identify

Novas - Verdi



Challenge 3

Changing Perceptions

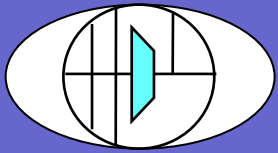
SoC Debug is Test issue

SoC Debug is Design Optimization issue

SoC Debug is Verification issue

Examples of converging capabilities

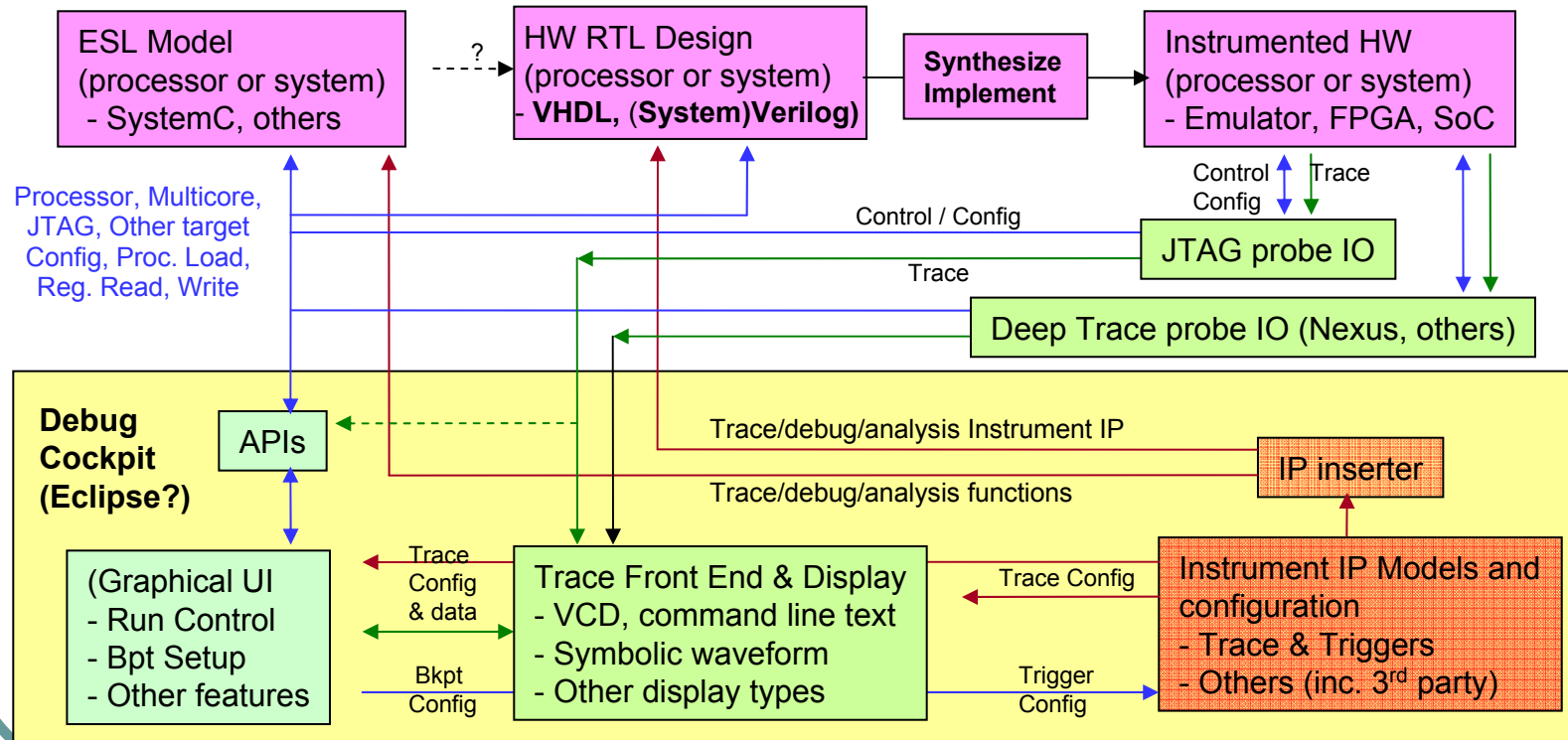
- **Annotating simulation analysis and debug data**
- **Automating debug using testbenches and assertions**
- **Common GDB implementations for ESL and debug Applications**
- **Common Display methods integrate ESL and Debug**

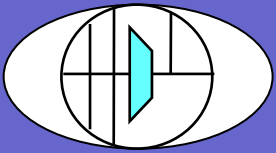


Integrating ESL and Instrument interfaces

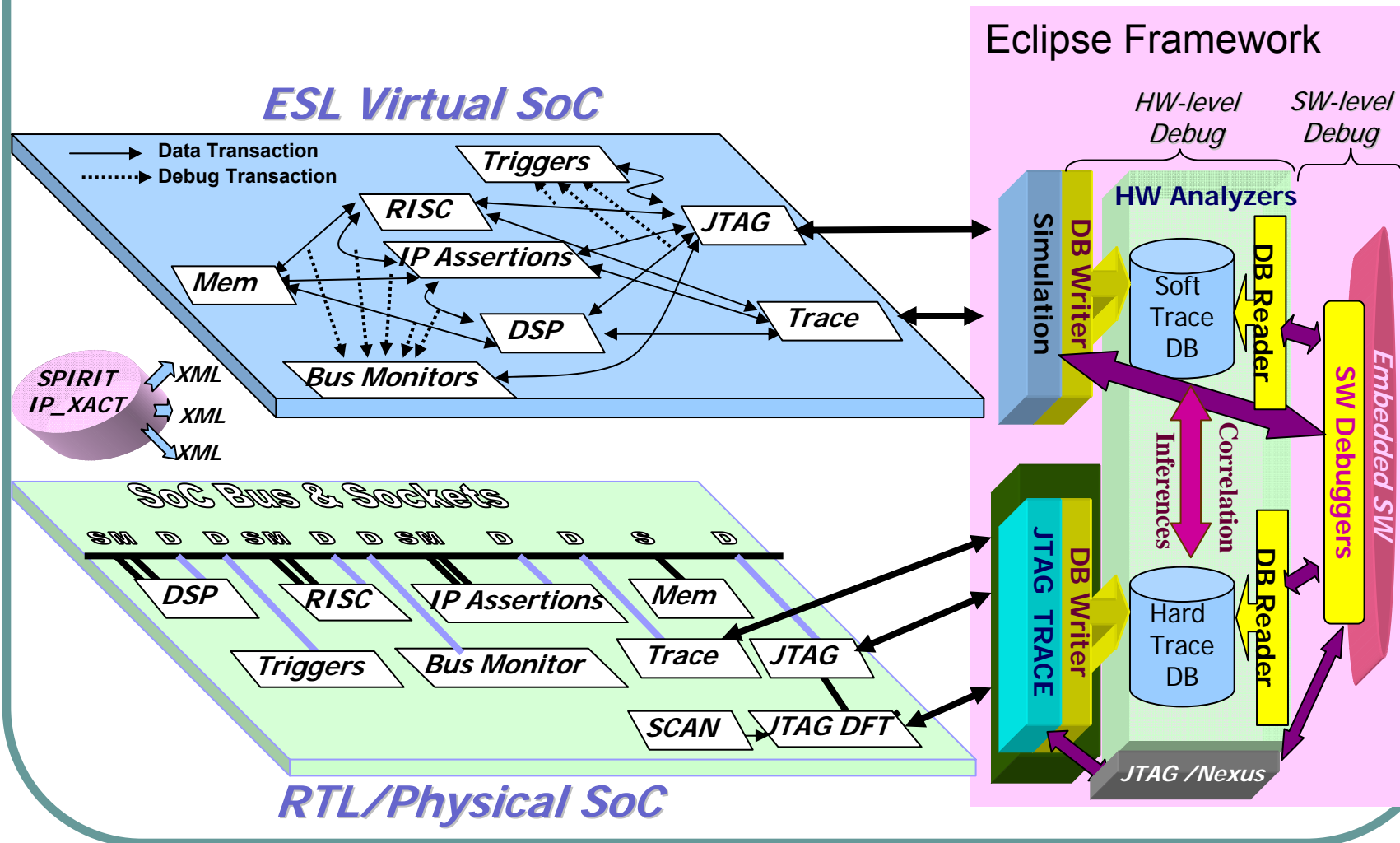
- **Common mechanism for Trace and Instrumentation interfaces**
- **Define Debug parameters and data once, reuse later**

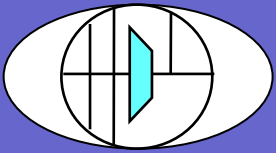
Design/Analysis Flow



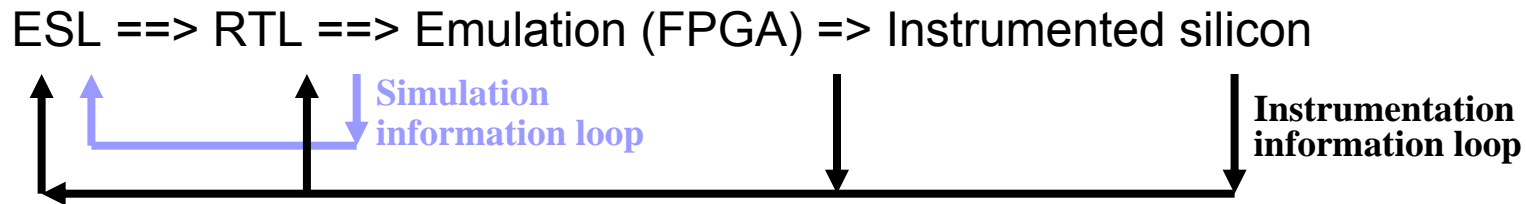


Next Stage Debug Challenges: Integrating ESL and Debug

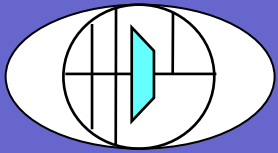




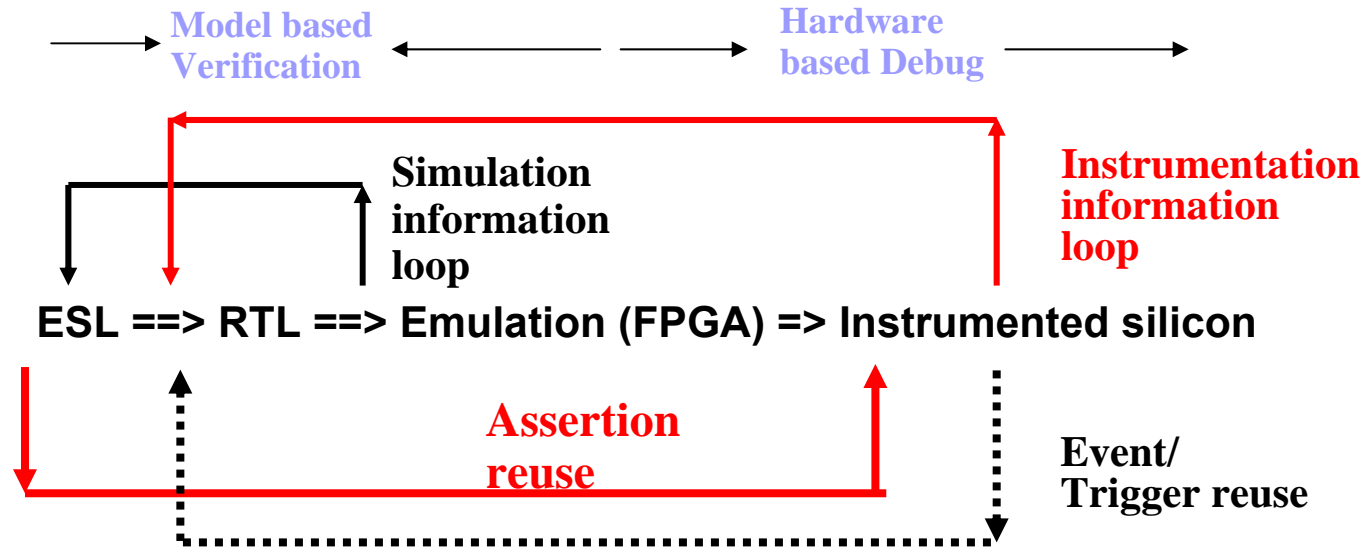
Trace and Simulation Integration



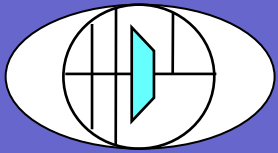
- Instrumentation upstream integration
 - Allows hardware Trace Instrumentation in VCD formats to be used for validation of simulation models.
 - Trace timestamps and trigger information align with simulation at transaction, behavioral, RTL level simulation
 - Automate compare of silicon vs. simulations
 - Real life analysis of corner cases, combinations of inputs, system level latencies and stalls not easily modeled



Reuse and Assertion Verification

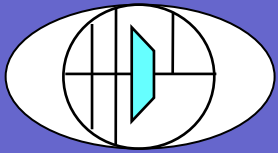


- Many System Assertions are synthesizable
- Synthesizable assertions support debug methods
 - Create Event based Monitoring, Triggering, etc...
 - Converse (Event/Trigger reuse as assertions) is useful too



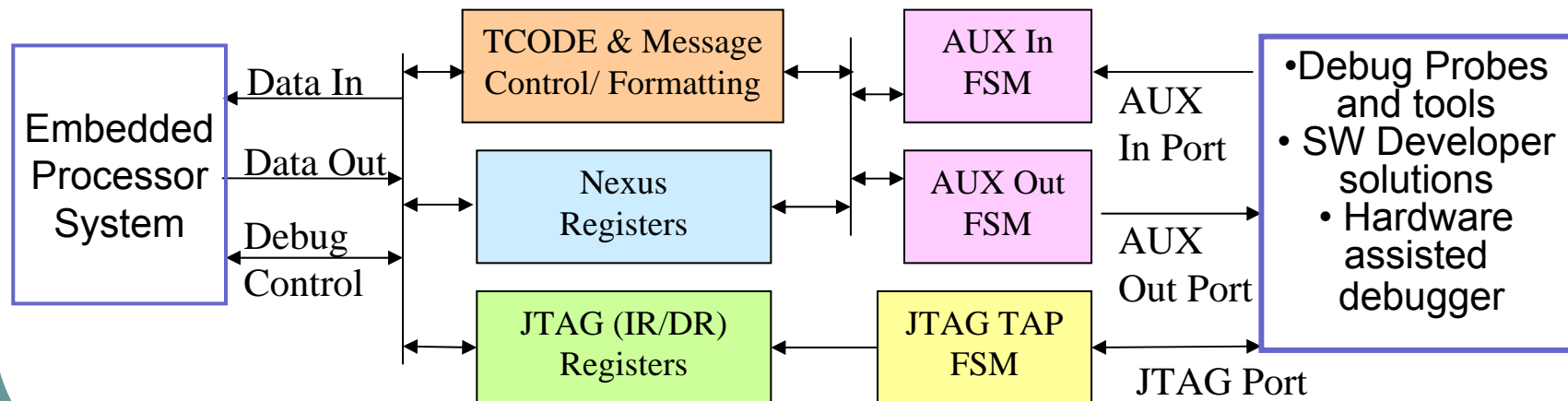
Industry Efforts

- **Ongoing industry driven standards and methodology efforts**
 - Nexus – IEEE std 5001
 - OCP-IP Debug Working Group
- **Established and emerging on chip instrument and tool venders**
 - Temento Systems - Dialite

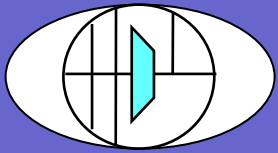


Nexus - IEEE 5001

- The Nexus 5001™ Forum is a program of IEEE-ISTO 5001-1999
- Focus on vendor neutral trace and calibration solutions, with emphasis on standardized embedded processor debug interface
 - The Nexus standard defines **standard on-chip features, auxiliary pins, transfer protocol, connectors**
 - **Many commercial silicon and tools solutions:** PPC, ARM, Tensilica,...
- Current efforts focusing on higher performance

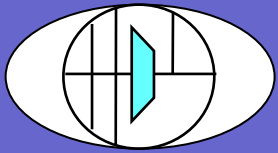


Nexus Architecture



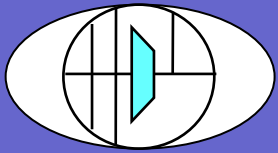
Nexus IEEE 5001 Instrumentation

- **Addresses Instrumentation interfaces for**
 - **Embedded run control, Breakpoints , Instruction/data trace**
 - **Memory and Register configuration and access**
- **Multiple Access Methods supported**
 - **Serial– JTAG**
 - **Parallel– Aux. Read (Trace) / Write (Configuration) Ports**
 - **High speed Serdes - 2008 standard**
- **Tools support from leading vendors**



Nexus Features

- Nexus offers a standardized connector pin-out definition
- Defines two types of IO port configuration
 - Aux Port only: Auxiliary Input/Output ports
 - Combined JTAG/Aux Port
 - Auxiliary Out is used primarily for trace output
 - Auxiliary In can be used for calibration
- Nexus supports debug of multiple cores over one or more ports
- Nexus defines 4 observation and trace recording classes
 - Class 1 : R/W user registers and user memory, Single step debug instructions breakpoint/watchpoint debug
 - Class 2 : Class 1 plus monitoring process ownership
 - Class 3 : Class 2 plus monitoring data, R/W memory locations
 - Class 4 : Class 3 plus remote processor control, advanced trace

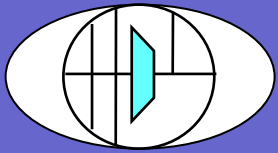


Nexus Standard Update

IEEE 5001- 2008

- Standardized connector specifications
- Support for Gigabit Serdes Channels
 - based on Xilinx Aurora interface standard
- Support for 1149.7 (2 wire JAG) interface
- Streamlining Nexus protocol
- Internal review ongoing
- Voting by end of year

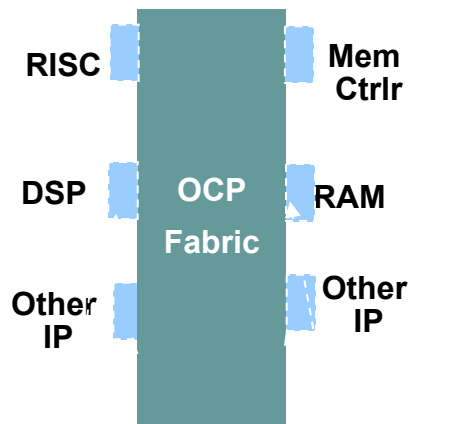
Nexus specifications are available from
www.nexus5001.org



OCP Debug Methodology

- OCP-IP is industry group - On Chip signal Protocol for SoC IP connection
- **OCP-IP Debug WG announced Debug Standard in 2007**
- Defined on chip instrument architecture www.ocpip.org/socket/whitepapers

OCP 2.1 Socket



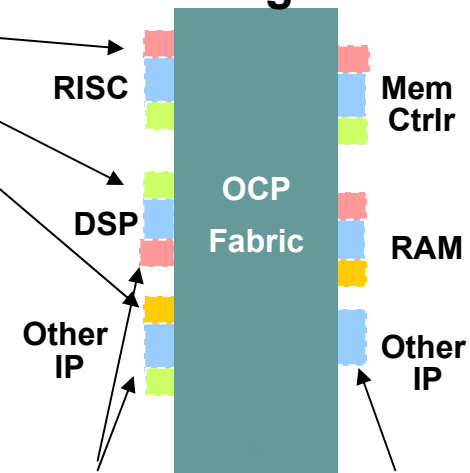
Standardized transfer socket
Master or slave, with optional
bursting, thread signals

Baseline Instrument

Sockets

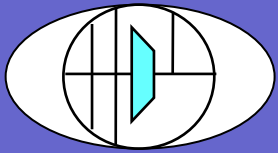
1149.1 JTAG
Debug resets
Generic Processor
debug handshakes
Cross Trigger Interfaces
Synchronize Trace/Debug
Trace Triggers
TimeStamp Interfaces
Power Management
Debug Security

OCP Debug Socket



Different Instruments
For different sockets

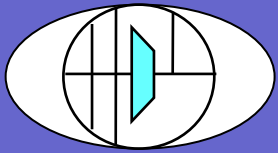
Debug Features
are optional



Debug Socket for Multi-Core

Focused on defining instrumentation related signaling between cores and other embedded subsystems

- **Identifies Basic and Extended sets of socket level signals**
- **Leverage other work for IOs (JTAG, Nexus), APIs, etc**
- **Supports debugging of multiple processor cores connected with the OCP interface.**
 - **Proprietary solutions supported (OCP Debug wrapper**
- **Allows designers to distribute debug signals as part of the system interface scheme.**
- **Enhances system providers ability to prepare multi-core (heterogeneous processors) debug hardware and software.**



The OCP-IP Debug Specification

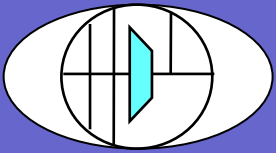
OCP Basic Debug Signals Interface

- **Debug Control and Run Control for Cores**
 - **Consistent (multiple) processor software debugger interfaces**
- **Cross-Triggering between Multiple Cores and Events**
 - **Scalable to on-chip/off-chip cross triggering**
- **Trace Interface**
 - **Bus traffic observation (system trace) and control (triggering)**
- **New classes of debug errors (different from system errors)**

OCP Extended Signals Interface (Special features)

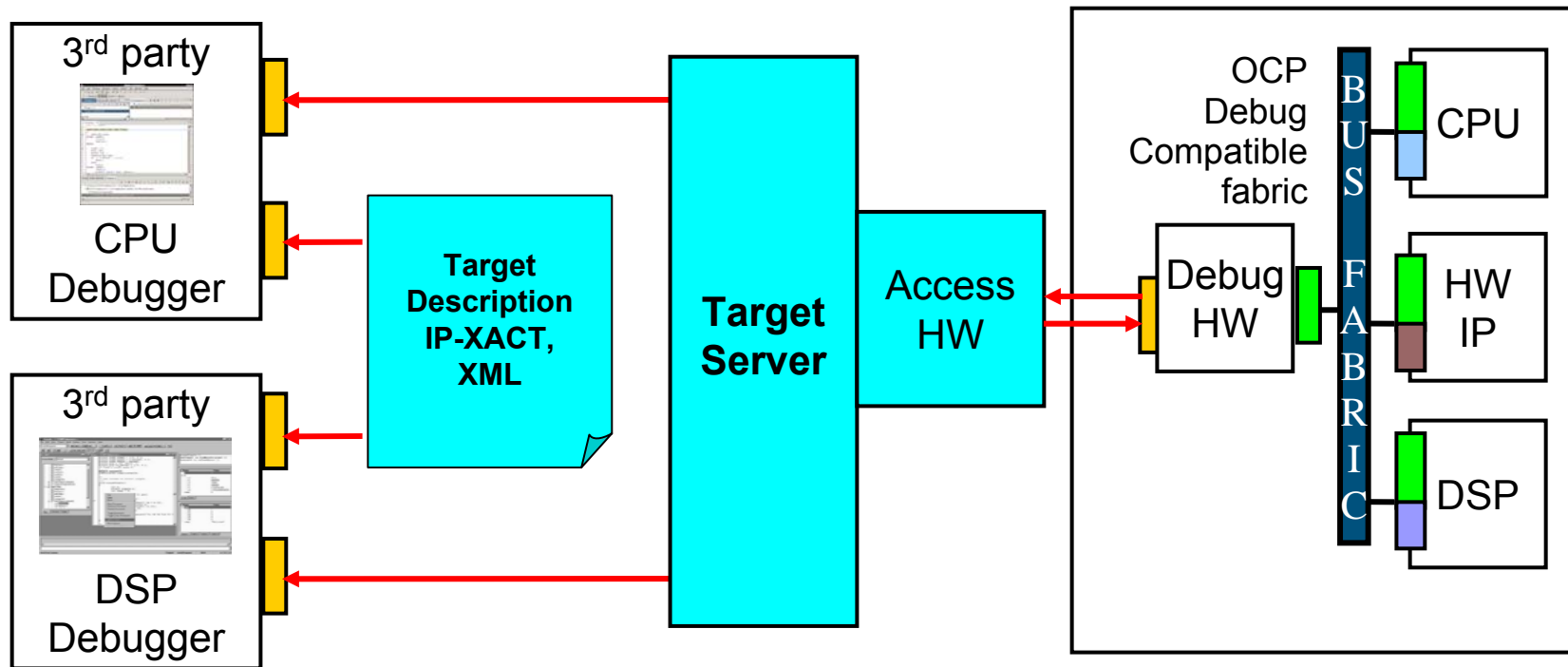
- **Performance Monitor (single and multi-core)**
- **Time-stamping (for asynchronous systems)**
- **Power Monitoring (voltage islands, gated clock islands)**
- **Security islands (application and system driven)**

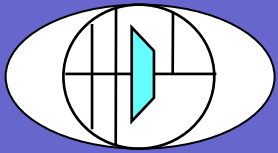
Of huge value
in final silicon



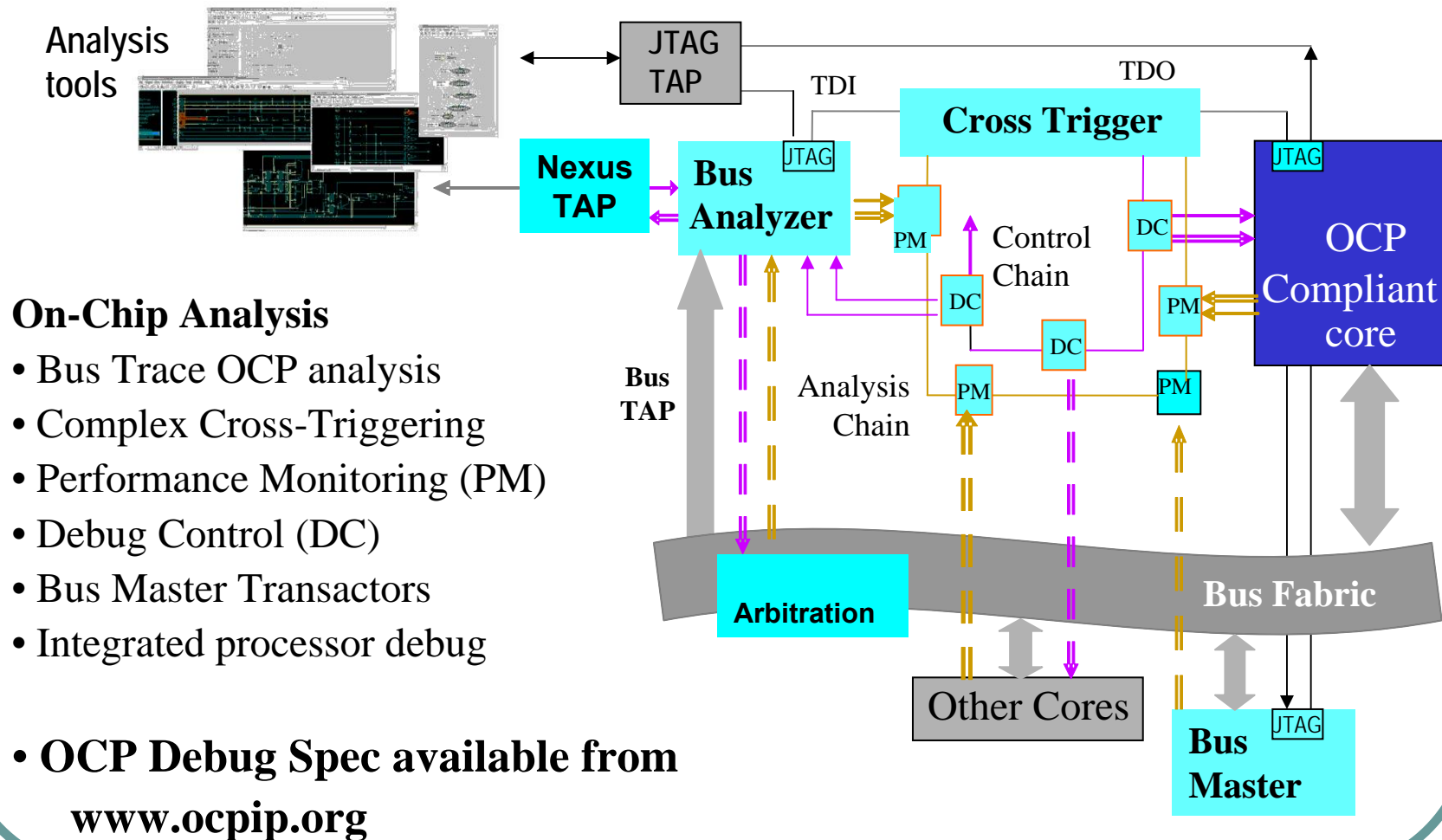
Debug Tool Interfaces

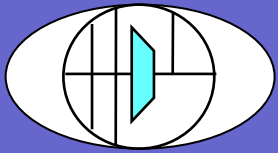
■ Standard Debug API (SPRINT??) ■ Basic OCP Debug Interface ■ Extended OCP Debug Interfaces





System Debug Instrumentation OCP/Nexus Example

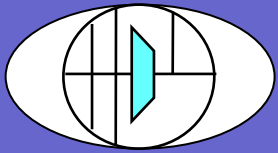




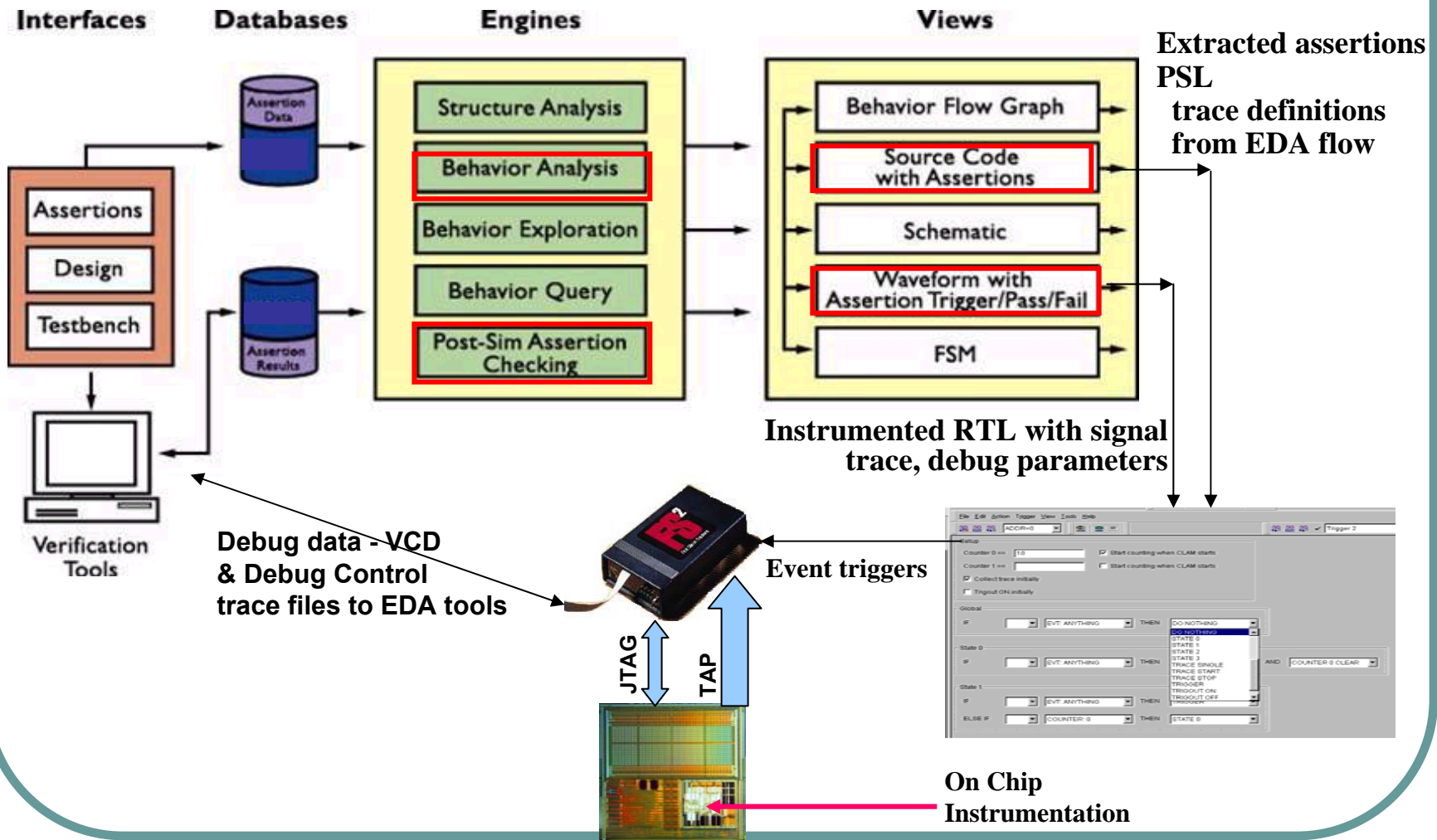
Example Application Solution

- **Work being done with Temento Systems**
 - **System enhancement of their Dialite instrumentation product**
 - **Instrumentation insertion from ESL flow**
 - **Debug data insertion to ESL flow**
 - **Tradeoffs of abstractions and timing vs. resources**

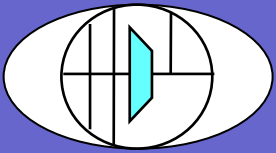
Available as part of current Dialite product release



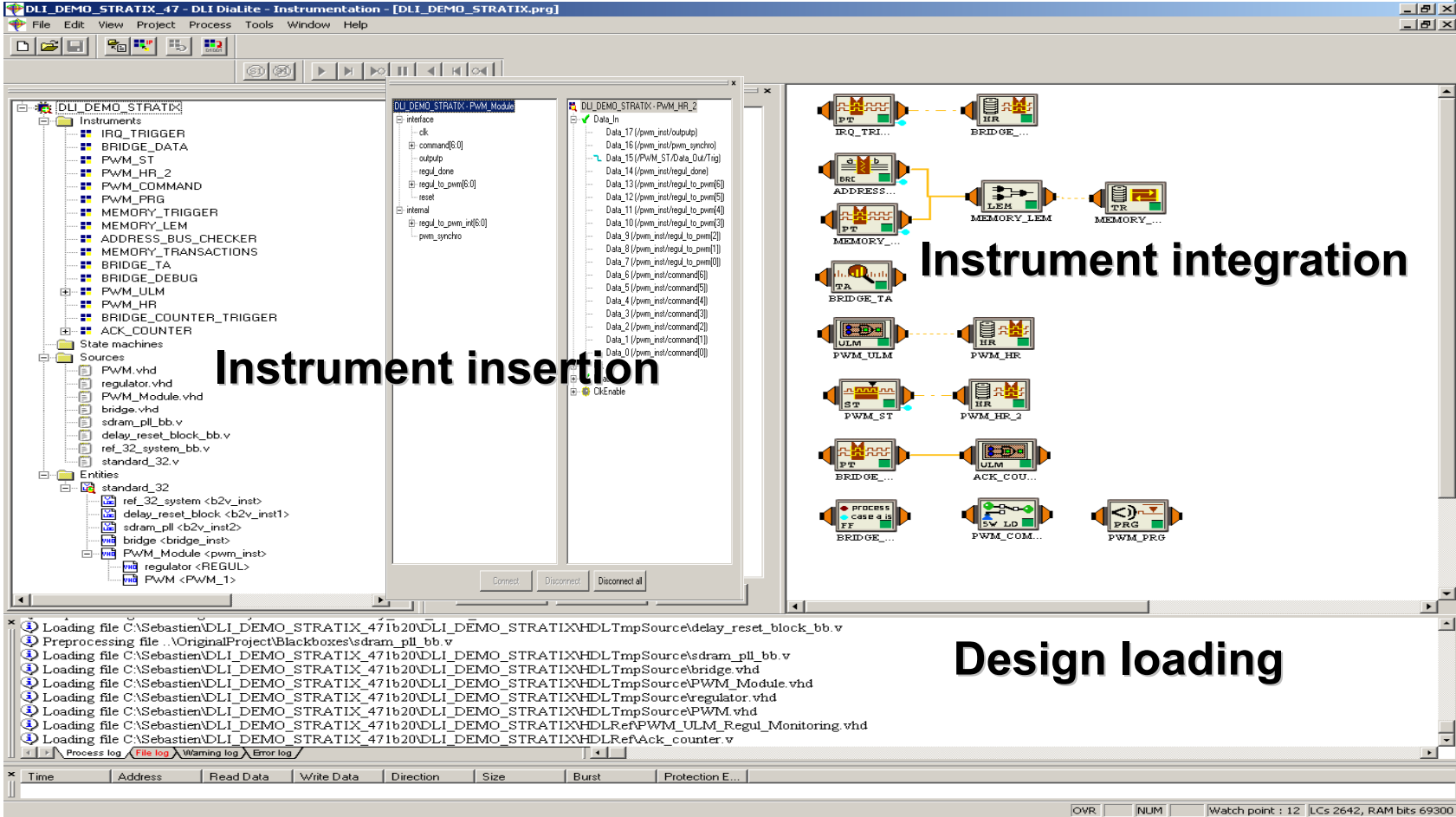
ESL Driven debug flow



HLDVT'08



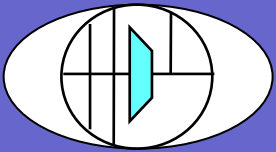
Dialite Debug Cockpit Instrument Configuration and Insertion



The screenshot displays the DLI Demo Stratix 47 software interface. The main window is titled "DLI DEMO STRATIX_47 - DLI Dialite - Instrumentation - [DLI DEMO STRATIX.prg]". The interface is divided into several panes:

- Left Pane:** A tree view showing the project structure, including "Instruments" (e.g., IRQ_TRIGGER, BRIDGE_DATA, PWM_ST, PWM_HR_2, PWM_COMMAND, PWM_PRG, MEMORY_TRIGGER, MEMORY_LEM, ADDRESS_BUS_CHECKER, MEMORY_TRANSACTIONS, BRIDGE_TA, BRIDGE_DEBUG, PWM_ULM, PWM_HR, BRIDGE_COUNTER_TRIGGER, ACK_COUNTER) and "Sources" (e.g., PWM.vhd, regulator.vhd, PWM_Module.vhd, bridge.vhd, sdram_pll_bb.v, delay_reset_block_bb.v, ref_32_system_bb.v, standard_32.v).
- Middle Pane:** A configuration window for "DLI DEMO STRATIX: Pwm_Module" and "DLI DEMO STRATIX: Pwm_Hr_2". It shows various data points and control signals like "Data_In", "Data_17", "Data_16", "Data_15", "Data_14", "Data_13", "Data_12", "Data_11", "Data_10", "Data_9", "Data_8", "Data_7", "Data_6", "Data_5", "Data_4", "Data_3", "Data_2", "Data_1", and "Data_0".
- Right Pane:** A diagram titled "Instrument integration" showing various instrument icons (e.g., IRQ_TRI, BRIDGE..., ADDRESS..., MEMORY_LEM, MEMORY..., BRIDGE_TA, PWM_ULM, PWM_HR, PWM_ST, PWM_HR_2, BRIDGE..., ACK_COU..., BRIDGE..., PWM_COM..., PWM_PRG) connected by lines, representing the instrument configuration.
- Bottom Pane:** A log window titled "Design loading" showing the process of loading files, including "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\TmpSource\delay_reset_block_bb.v", "Preprocessing file ..\OriginalProject\Blackboxes\sdram_pll_bb.v", "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\TmpSource\sdram_pll_bb.v", "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\TmpSource\bridge.vhd", "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\TmpSource\PWM_Module.vhd", "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\TmpSource\regulator.vhd", "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\TmpSource\PWM.vhd", "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\Ref\PWM_ULM_Regul_Monitoring.vhd", and "Loading file CASEbastien\DLI DEMO STRATIX_47\1b20\DLI DEMO STRATIX\HDL\Ref\Ack_counter.v".

Additional text overlays on the screenshot include "Instrument insertion" and "Design loading".



Dialite Debug Cockpit Results Display and Analysis

GTKWave - D...leon3_stratixii_demoDLI_AMBA_TRACER.vcd

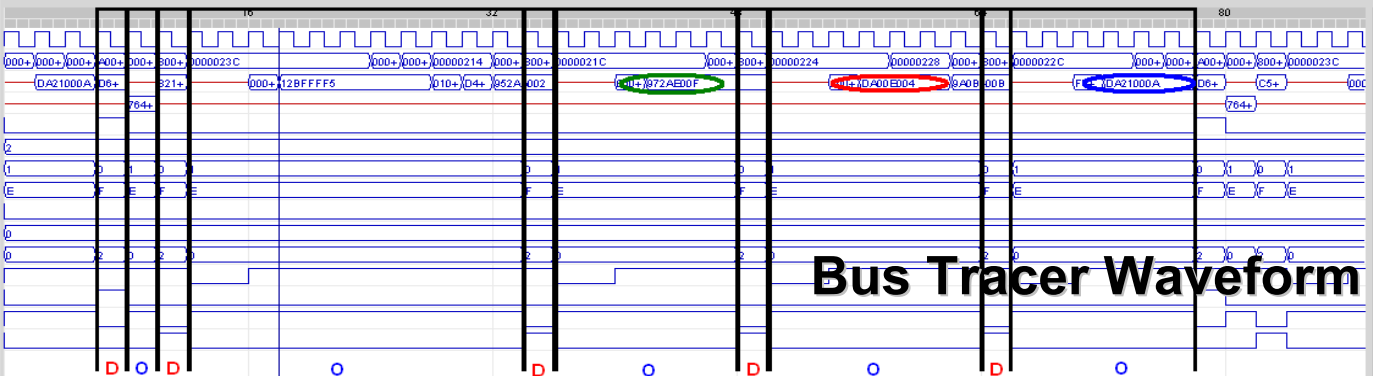
File Edit Search Time Markers View Help

From: 0 To: 92 Maximum Time : 48450 - Current Time : 23

Signals

- DLI.HCLK=1
- DLI.HADDR[31:0]=0000023C
- DLI.HRDATA[31:0]=12BFFFF5
- DLI.HWDATA[31:0]=ZZZZZZZZ
- DLI.HWRITE=0
- DLI.HSIZE[2:0]=2
- DLI.HBURST[2:0]=1
- DLI.HPROT[3:0]=E
- DLI.Master0_HBUSREQ=0
- DLI.HRESP[1:0]=0
- DLI.HTRANS[1:0]=0
- DLI.HREADY=1
- DLI.Slave2_HSEL=0
- DLI.Slave1_HSEL=1
- DLI.Slave0_HSEL=0

Waves



Bus Tracer Waveform

Time	Address	Read Data	Write Data	Direction	Size	Burst	Protection E...	Bus Reques...	Response	Transfer Type	Ready	Slave
0	0x0000022C	-	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
1	0x00000230	0xDA21000A	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
2	0x00000234	0xDA21000A	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
3	0x00000238	0xD600A018	-	WRITE	32	SINGLE	DATA ACC...	0	OKAY	SEQUENTIAL	1	1
4	0x00000238	-	0x76458020	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
5	0x80000318	0x8210600F	-	READ	32	SINGLE	DATA ACC...	0	OKAY	SEQUENTIAL	1	0
6	0x0000023C	-	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	0	0
7	0x0000023C	-	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	0	0
8	0x0000023C	0x00000008	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
9	0x0000023C	0x12BFFFF5	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
10	0x0000023C	0x12BFFFF5	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
11	0x0000023C	0x12BFFFF5	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0
12	0x00000240	0x12BFFFF5	-	READ	32	INCR	OPCODE F...	0	OKAY	IDLE	1	0

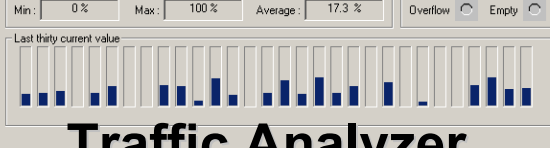
HDL Fault Finder Stack

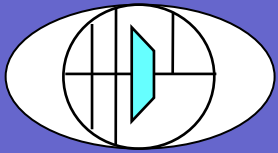
```
119 when "00" => null; -- idle
114 when "01" =>
117 if r.hwwrite = '0' then v.penable := '1';
118 else v.pndata := abhi.hwdata; end if;
119 v.psel := '1'; v.state := "10";
120 when others =>
121 if r.penable = '0' then v.psel := '1'; v.penable := '1'; end if;
122 v.state := "00"; v.hready := '1';
123 end case;
```

Bus Tracer Stack

Traffic Analyzer

Traffic analyzer statistics: Min: 0% Max: 100% Average: 17.3%

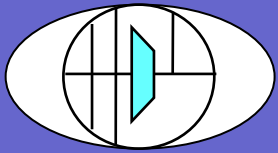




What can instruments do

•Types of on-chip instruments available

Logic Debug	Debug Switches	Interactive control and setting signals to desired values
Logic Debug	Virtual Debug LEDs	Interactive observing of internal signal values
Logic Debug	Debug Logic Module	Custom triggering on specific event (FSM sequences, assertions...)
Trigger/Analysis	Glitch Detector	Signal glitch tracking and system synchronization
Trigger/Analysis	Logic Equation Module	Monitor/Trigger on real-time adjustable logic equation of two signals
Trigger/Analysis	Parallel Trigger	Monitor/Trigger on Pattern recognition of parallel signals
Trigger/Analysis	Serial Trigger	Monitor/Trigger on pattern recognition of serial/sequential signals
Trigger/Analysis	Counter Marker	Trigger after a fixed duration or fixed occurrence of an event
Trace	History Register	Trigger based sequential recording into RAM or registers
Trace	Transaction Register	Trigger based selective recording of key events into RAM/registers
System Analysis	Bus Range Monitoring	Capturing signal value inside or outside a given range
System Analysis	Bus Trace Analyzer	Monitoring and compressed data trace compression computing.
System Analysis	Bus Protocol Monitor	Check bus transactions based on defined rules of bus behaviors
System Analysis	Traffic Analyzer	Analyze bus traffic metrics (DMA, stalls, latencies) over long periods
Verify Properties	Assertion Checker	Hardware verification of PSL/SVA system properties
Logic Debug	HDL Fault Finder	Monitoring events by inserting Watchpoints and Breakpoints
Stimulate Signals	Pseudo Random Gen.	Inject pseudo random signal sequences for robustness checking



“Simple” Bus Architecture

- Debug a complex design composed of processors and IPs
- Different levels of verification

Signal Debugging HDL Debugging Assertion Debugging

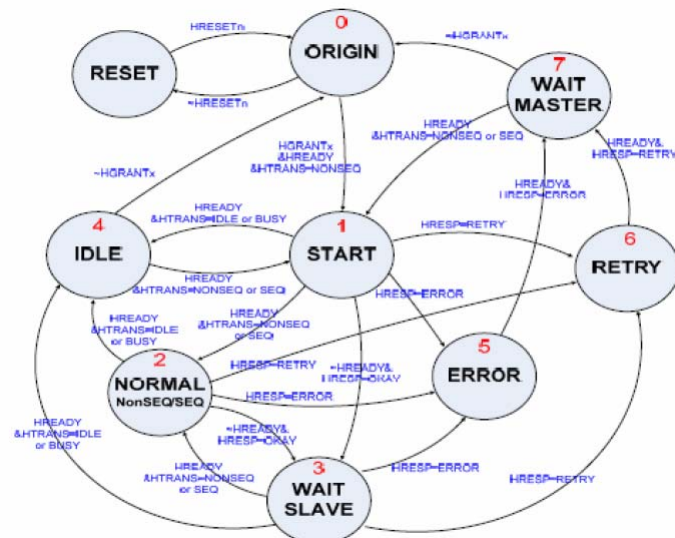
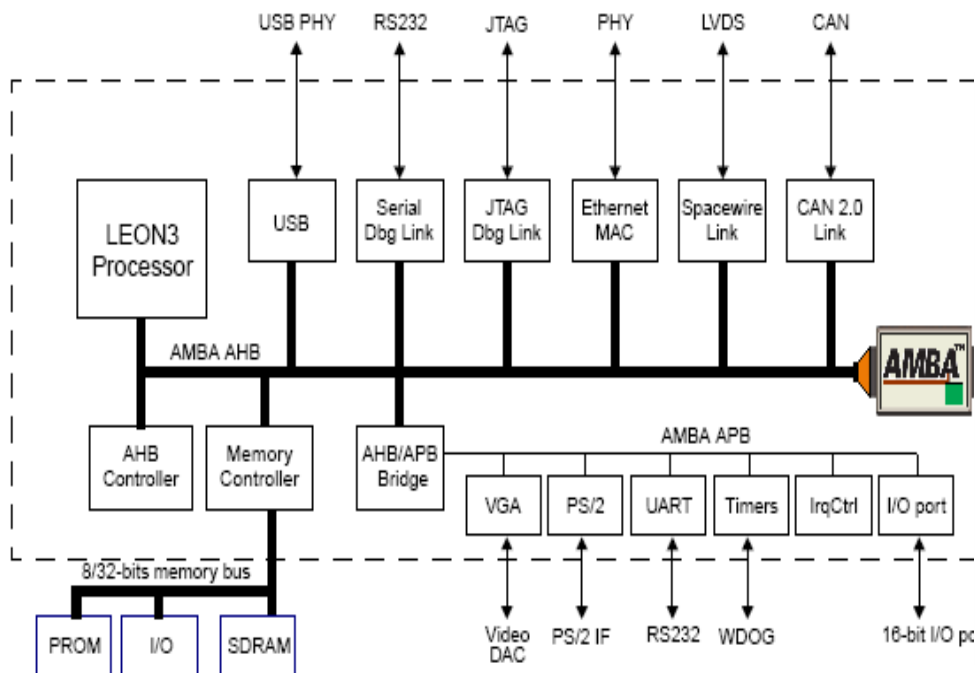
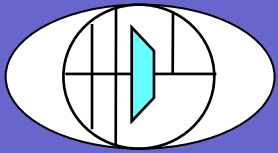
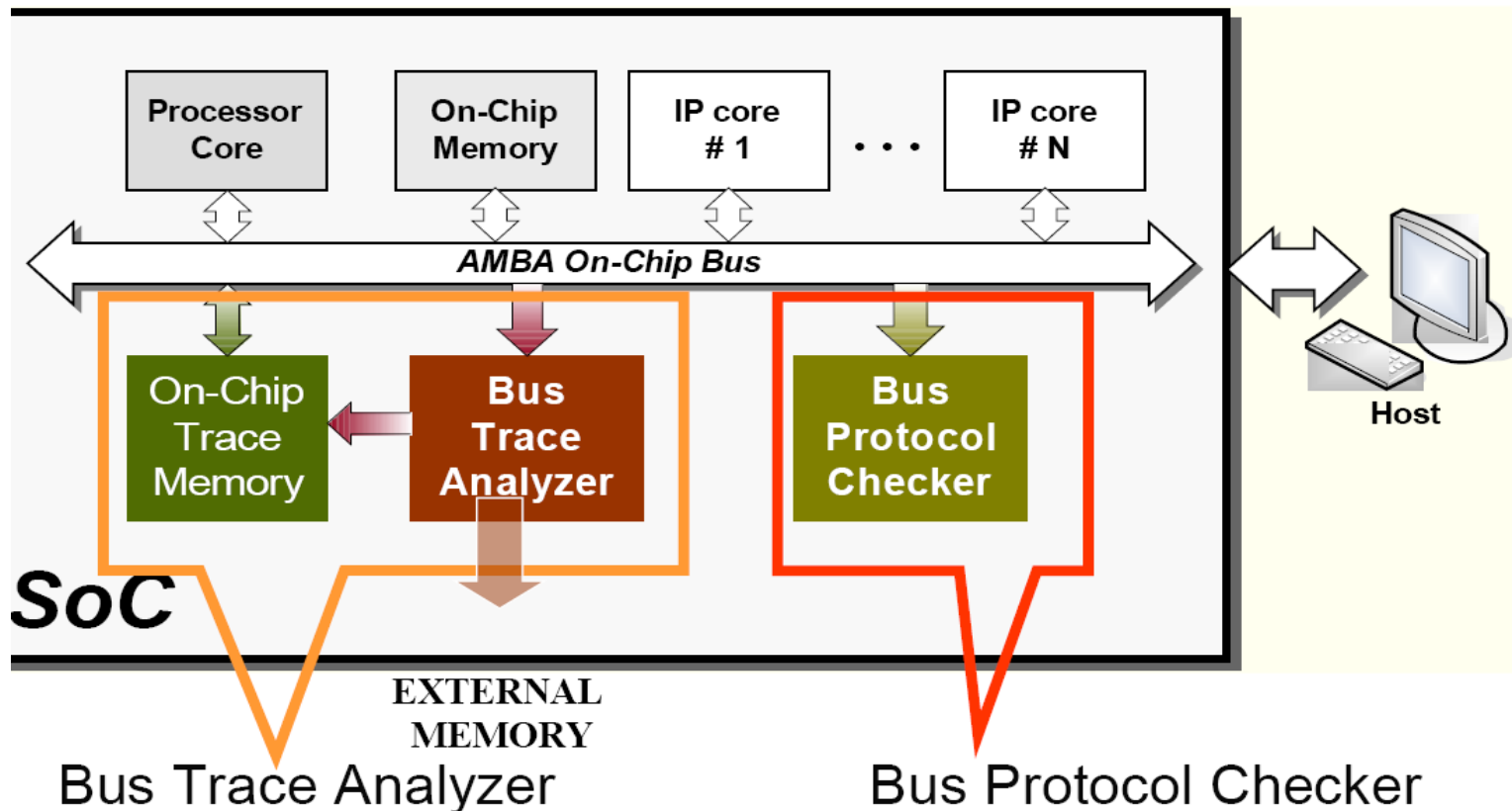


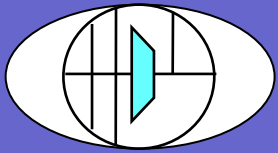
Figure 4. AMBA AHB master transition states



Bus Trace example

Goal : Record as many AMBA Bus Transactions as possible depending on a trade-off between the accuracy and the depth of the recording.



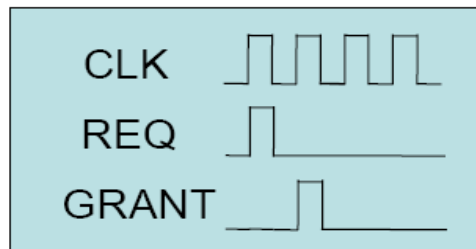


Assertion Based Instrumentation

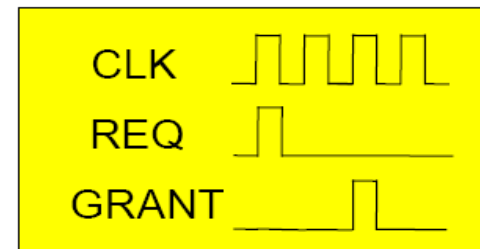
PSL assertion example

```
assert always req -> next grant;
```

This assertion means that whenever the HDL signal req is true, the HDL signal grant must be true in the following cycle. A cycle is specified using a given clock signal.



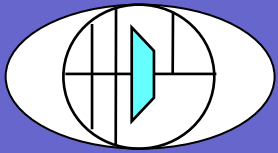
Transaction OK



Error (grant arrives too late)

```
assert always req -> next (ack until grant);
```

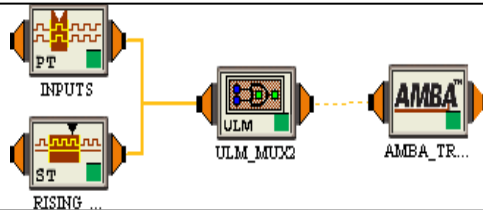
This asserts that whenever req is true, ack is true in the following cycle and ack remains true until the first subsequent cycle in which grant is true.



Instrumentation Features



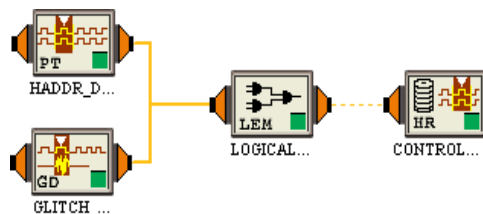
PSL Assertion Checker : Check bus transactions
Switch/Leds : Check current state of external IO



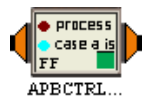
Capture bus transactions during normal execution and boot of the processor after a hard reset



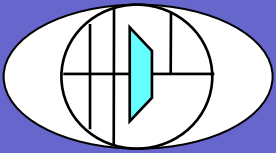
Isolate specific transactions which access to a defined part of the memory (instruction fetching, write to AHBRAM, read/write to APB registers)



Concentrate on control signals associated to each AMBA transaction (write strobe, protection signal, selection signal, ...)



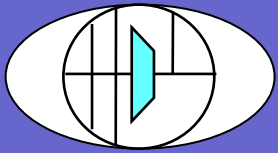
Check internal signal behaviour according to HDL code in order to find casual errors or bugs



Instrumented Assertions

Example of supported properties

```
    assert always {sig1} ==> {sig2};
    assert always {sig1;sig2} -> {sig3; sig4};
    assert always {sigA} ==> {req; busy; gnt};
assert always {sigA} ==> {req & !busy & !gnt; !req & busy & !gnt; !req & !busy &
                        gnt};
    assert always {sigA} ==> {req; busy; busy; busy; gnt};
    assert always {sigA} ==> {req; busy[*3]; gnt};
    assert never (sig1 -> (sig2 before sig3));
    assert never (sig1 -> sig2 );
    assert always req -> next ack;
    assert always req -> (req until_ grant);
    assert always sig -> (sig1 before sig2);
    assert always sig -> (sig1 before_ sig2);
    assert always {sig1} -> {sig2[*2]; sig3};
    assert always (grant -> prev(req));
sequence s1 = {req; !req; req; !req}; assert always req -> s1;
```

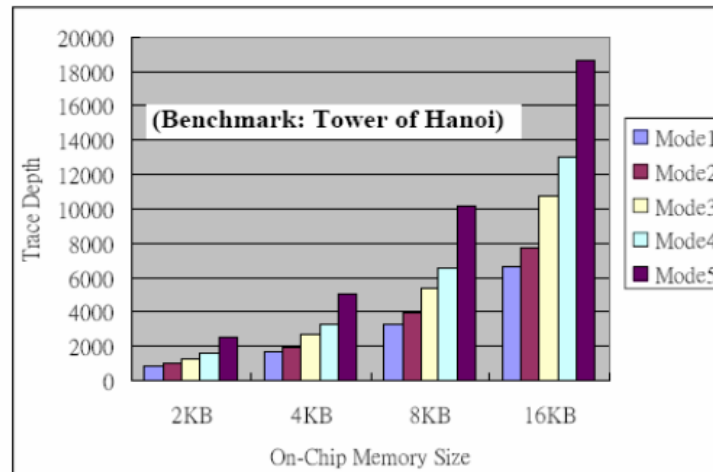
Bus Trace Compression

Data Trace Compression Ratio in Different Tracing Modes

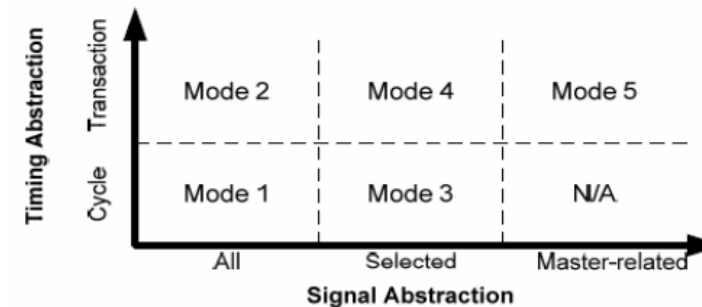
	Original trace size (88bits/cycle)	Compression Ratio After Signal/Timing Abstraction and Trace Reduction				
		@ mode 1	@ mode 2	@ mode 3	@ mode 4	@ mode 5
Perpetual Calendar	880000	77.58%*	83.00%	90.05%	94.67%	96.99%
Fibonacci Sequence	880000	78.32%*	80.05%*	87.91%	90.45%	94.73%
G. C. D.	880000	83.22%	83.45%	89.14%	92.76%	95.68%
Towers of Hanoi	880000	78.64%*	80.61%*	85.99%	88.58%	92.56%
Knight Problem	880000	78.85%*	80.51%*	96.39%	97.16%	98.32%
Quick Sort	880000	78.27%*	79.99%*	99.64%	99.71%	99.80%
Geometric mean	-	79%	81.26%	91.39%	93.81%	96.32%

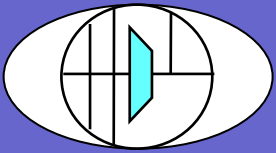
(*): the star sign means the trace buffer (256 bits) overflow

Trace Depth Vs Different Tracing Modes



Tracing Modes of Combinations of Different Signal and Timing Abstraction Levels





Summary

- ESL and Instrumentation are important analysis/verification trends
- Need and value for synergy of EDA and Debug tools integration
- Growing recognition of value of standards based integration
- Design flows need to include
 - Value from consistency, reuse in debug and verification
 - Standards based integration makes the difference
- Wide range of On Chip Instrumentation IP and solutions
 - Instrumentation Tools, Interfaces for chip event monitoring, setting triggers, etc.
 - Automation for simplifying use and analysis
- HDL Dynamics is working with Temento and others (GreenSocs) to develop ESL and Instrumentation compatible Debug Cockpit