Request-Response Trace for Bus Performance Analysis

Dr. Neal Stollon  
neals@fs2.com

Bruce Ableidinger  
brucea@fs2.com

First Silicon Solutions Division of MIPS Technologies  
5440 SW Westgate Dr., Suite 240  
Portland, OR 97221  
(503) 292-6730
Abstract:
The complexities involved in interconnect centric designs and their flexibility and parameterization make sophisticated latency and performance analyses an important consideration, not just to further optimize the current design, but to quickly and easily quantify requirements and enhancements. A Bus Request-Response Trace (RRT) system for on-chip IP and analysis, along with related capabilities and alternatives was developed to provide visibility into the various interconnection points of SoC architectures, by recording request and response bus events and measurements for one or several bus masters simultaneously. This system analysis implementation allows capture of information about core load/store operations and their latency for the different bus masters, and exports them over dual trace ports to an off chip trace probe, along other trace and analysis data captured at the processor level. While this paper discusses specific bus architectures and interfaces to MIPS processor cores, both interconnects and analysis based on other processor cores or bus architectures can be addressed without any significant loss of generality.

Author(s) Biography

Neal Stollon is a systems engineer and Director of Technical Marketing with First Silicon Solutions Division of MIPS Technologies Inc. He has over 25 years digital design and processor development experience at Texas Instruments, LSI Logic, Alcatel, and others. Dr. Stollon has a Ph.D in EE from Southern Methodist University and is a Texas Professional Engineer. He has written over 30 technical papers and holds 7 patents.

Bruce Ableidinger is Director of Business Development at First Silicon Solutions (FS2). He has worked in the instrumentation and development tools business for over 25 years including time at Intel and Tektronix. He has a MSCS from Oregon State University and a BSEE from Washington State University. He is co-inventor of 3 patents.
Request-Response Trace for Bus Performance Analysis

Abstract: A bus traffic monitoring approach, called a Bus Request-Response Trace, (Bus RRT™) uses On Chip Instrumentation (OCI™) IP to provide performance analysis of on bus level latency measurements for complex Systems on Chip (SoCs). In complex SoC, which typically include an increasing large number of embedded processors and other cores, analysis of the interconnect takes on new complexities in order to allow software development that maximizes performance for each processor and at the systems level.

Introduction

In complex SoC platform architectures, one of the more encompassing areas where modification and tuning can provide performance rewards is at the functional interconnect level. On Chip interconnect systems for integrating IP blocks into a SoC solution are an area that has received much attention in recent years, with a number of multi-layer, cross-bar, and network on chip alternatives being developed both by IP vendors and internally by integrating companies themselves. The goal in most cases is to address within reasonable wiring and size constraints, an increasing amount of bandwidth required for complex applications, optimized communications between different block of IP, both with each other and with shared resources such as memory and peripherals. In many cases these interconnect architectures contain sophisticated internal complexity and which have tunable parameters, to allow tradeoffs and optimization of the interconnect features for a given architecture and application. The interfaces to these interconnect systems is typically implemented at a socket level, using one of several bus interface standards (OCP, ABMA AHB, and AXI being among the more prevalent) as a modular, bi-directional socket interface between an IP block and other interconnected blocks.

This document discusses a socket-level traffic monitoring approach, called a Bus Request-Response Trace (Bus RRT™) that uses On Chip Instrumentation (OCI™) IP to provide information on system bus latencies and related measurements for a complex Systems on Chip (SoC) being developed by Mobileye Vision Technologies. In complex SoCs, which typically include an increasingly large number of embedded processors and other cores, the interconnect takes on a new level of complexity in order to enable the maximum performance of each processor. A comprehensive understanding of the interaction and performance of this interconnect is critical for real time performance and synergistic to the understanding of the overall processing operations and interactions that are best analyzed in a physical, rather than simulated environment (1).

SoC Interconnect Complexities

Advanced buses allow a range of high bandwidth implementations and define a number of features and capabilities in addition to baseline data transfer. These features include the extensions for special bus command modes, burst operations and multiple data tags and threads that increase the number of traced signals. The flip side of working with advanced bus architectures is it does present an additional level of complexity in configuring and coordinating operation of large amounts of data. Analysis considerations include issues specific to a given interface and more global issues of how the on-chip bus subsystem is performing. These can include understanding and optimizing bus issues like transmission efficiency, latency, saturation, resource conflicts, and other operational considerations that can have a direct impact on the performance and operation of the processor components.
On chip instruments, (along with simulation) play an important part of SoC development and verification flows, providing the ability to analyze what is happening on the hardware itself, both during prototyping and systems level verification stages, and increasingly on the final products themselves. The problem in analyzing information like embedded buses in hardware in many cases hinges on a problem of visibility – it is difficult to fix what you cannot see. This visibility problem for the embedded SoC is more complex than can be addressed adequately by traditional on-chip test methods such as traditional JTAG scan, for several reasons:

- Bus operations are multi-cycle, with signals in a bus cycle becoming active at different times, requiring sequential tracing, rather than as a single-cycle snapshot that scan typically provides.

- Bus operation problems are interrelated with the operations of at least two communicating blocks (a processor and memory peripheral as an example). Traditional debug methods such a halting part of a system for test can introduce changes and new variables that interfere with the test scenario and process.

- If problems are intermittent or sparse, then trace operations need to operate in a triggered mode, so information for a given range of bus cycles of interest is captured in real-time.

- The problem is, to a large part, a multicore extension of embedded processor analysis, where run control (such as MIPS EJTAG™) and instruction execution and data trace (such as MIPS PDtrace™) are integral parts of processor support. For larger systems with multiple cores, the problem extends beyond processor execution to also understanding system operation and communications.

Figure 1: Processor and Bus Trace OCI for a Simple SoC
All this points to better understanding at the interconnect level being a critical layer of analysis. There are a variety of reasons why new generations of interconnects and analysis tools to support them are increasingly critical and important;

1. Heterogeneous multiprocessing IC integration requires a clean way to efficiently handle complex data flow architectures with inter-communicating cores, with diverse requirements and features - different data feeds, operating speeds, types of data endianness, diverse and dynamic levels of security, and Quality of Service (QoS).

2. Growing awareness that flexible and rapid integration of IP from multiple external sources is needed to reducing time to market, with concurrent requirements for integrating the test, hardware verification, and simulation environments.

3. Growing sophistication of the processors’ data flow requirements, requiring the ability to handle multi-processing and multi-threading in efficient, non-blocking manners. In particular, the multi-threading features of the MIPS32® 34Kf™ cores benefit from both a processor and bus level system analysis environment.

4. Growing appreciation for platform design approaches that efficiently address product upgrades, market segmentation, and product differentiation while maintaining common design infrastructure to keep design efforts manageable.

5. Supporting analysis IP provides a means of tying together pre-silicon and initial physical product verification by providing access and visibility to embedded operations (2). This analysis allows in-depth understanding operation of the design under different conditions.

Industry is addressing these issue by a range of solutions, one of which is more complex interconnect structures and adoption of socket-based interconnect approaches. OCP architectures have pioneered this concept of socket based design, with other bus architectures adopting many of the same principles in order to provide the needed range of design tradeoffs and performance.

At least three commercial companies offer interconnect and bus structure automation and IP tools, with several other SoC centric interconnect approaches being used as proprietary customized solutions by SoC silicon vendors. Sonics (http://www.sonicsinc.com) offers the most mature commercially available solution, with its third generation SMX and related SMART Interconnect architectures. Offering alternative approaches are Arteris (http://arteris.com), offering a Network on Chip (NoC) interconnect architecture and Silistix (http://www.silistix.com) offering a self-timed (clockless) interconnect.

**Socket Based SoC Design**

Socket based interconnect is a standards oriented approach that focuses on adding value to the interface socket between the IP block and the bus fabric. Socket based interconnect is an underlying principal in many OCP based architectures, but can also be applied to other bus architectures. Since many bus architectures allow addition and selection of various bus options that increase the functionality of the bus interconnect, using a socket based interface simplifies addition, removal, or accommodation of the bus interface to the IP blocks, and the development of test suites to address verification, and optimization of the design.

Figures 2 and 3 diagram demonstrate some of the features of a socket based design. The sockets accept initiators (masters) and targets (slaves), with the circuitry of the socket encompassing the necessary state machines, gating and muxing circuitry, and wiring to effect the desired data flow (QoS, multi-threaded non blocking communication, security features, dynamic power gating, etc.)
characteristics. The socket consists of a set of agents that provide the necessary signal and protocol management to address the specific interface needs of a core to the more general resources of the interconnect fabric. Socket based bridges may also be defined to other interconnect environments between OCP, AMBA AHB, and AXI and bridges for other arbitrary existing interconnect structures can also be developed to simplify utilization of legacy hardware.

- **Agents provide** …
  - Protocol conversion
  - Agent adapts to IP core
- Decoupling of IP cores from fabric
  - Provide local, isolated environment
- Data flow services
  - QoS-based arbitration
  - Power
  - Access security
  - Error management
  - Burst, width, and command conversion

**Figure 2: Socket based Agents**

These communication complexities require in system information and performance analyses to determine parameters to for optimizing use of the inherent flexibility and parameterization to optimize the design. FS2 Bus Monitoring IP and Analysis Tools to monitor this performance transparently for high complexity interconnect networks to provide optimized system operation.

Supporting the need to monitor bus data for values, latencies, and other data that is relevant for both debug and optimization, interconnect architectures includes features that enable data flow control, access, and monitoring of performance and optimization parameters. As in any aspect of design, there are tradeoffs between the levels of visibility into interconnect and communications operations and the resources required to be able to effectively and sufficiently monitor key information. These resources typically include some tradeoff of on-chip instrumentation resources, IO and trace buffering bandwidths, and the overall gate impact to the design. The specifics of the monitoring function in some cases vary with the size and performance or the interconnect structure.

The Sonics Multi-Service Exchange (SMX™) architecture offers a good example, since is it comprised of a distributed structure of three classes of interconnect structures, Crossbar Exchanges, Shared Link Exchanges, and Extended Link exchanges, each with their own features and analysis requirements. SMX Crossbars allow the fastest unimpeded connectivity, while Shared Links require less overhead of additional gates and also support the optimal data flow by QoS selection. Extended Links support more widely separated IP cores. Related Sonics interconnects also provide connectivity of slower peripheral or optimize scheduling of DRAM controllers and the effective bandwidth of external DRAM.
An architecture like SMX supports several types of interconnect segments that can be optimized for performance and require analysis information from the interconnect structure. Different types of interconnect segments have differing integration and test ports requirements and features but appear common at the socket level.

**Figure 3: Sonics SMX Structure**

An architecture like SMX supports several types of interconnect segments that can be optimized for performance and require analysis information from the interconnect structure. Different types of interconnect segments have differing integration and test ports requirements and features but appear common at the socket level.

**An Overview of Internal and External Alternatives for Bus Trace**

Debug ports simplify controllability and visibility by providing a low overhead access to internal signals. Debug interfaces can be categorized as

a. internal, where most of the instrumentation functions are implemented on chip and the interface uses a low pin count interface, most often JTAG or

b. external, where the instrumentation functionality is shared between an on-chip component and an off-chip component, typically implemented in a probe that are connected by a (typically parallel) trace probe port.

JTAG is the default analysis interface for many IP blocks (since it is there for test purposes, irregardless of debug features the block might support), however JTAG is a relatively slow serial interfaces not designed for trace and other data intensive analysis. Most JTAG based instrumentation relies on on-chip memory to buffer between traced data and the JTAG export bandwidth available. The size of these buffers vs. the amount of trace required is a tradeoff. Large buffer may impact SoC area. Buffers of modest size however are easily overloaded for a large amount of trace data that is generated in cases of multiple IP blocks or internal buses, placing limits on durations of trace that can be supported in some cases.

The underlying advantage of JTAG is that is it ubiquitous, and is a default port implemented in most digital chips for test purposes. JTAG allows user defined instructions to extend the JTAG
instruction set for probe or trace modes, instructions for ICE, and access internal JTAG enabled registers. While in probe mode, the processor ICE can examine and modify the internal and external state of a system’s registers, memory, and I/O space. In a trace mode, it can be used to export arbitrarily large amount of information off chip serially. A rich infrastructure of tools environments and standardized debug schemes have been built on this foundation to provide both JTAG debug of embedded processors and other parts of an embedded system.

Adding an additional probe ports provide IO bandwidth needed for more in depth on-chip instrumentation approaches and is the primary focus of this paper. Internal trace solutions, in particular, FS2 Bus Navigator are discussed in depth in other papers [4.5] Bus Navigator is an on-chip JTAG embedded trace solution, an on-chip bus analyzer that is configurable for bus analysis, but which can also be used for on chip logic analysis. Bus Navigator Monitors and traces up to 256 signals user defined signals selected during synthesis with a trace depth of 64- 64K trace cycles (using on-chip RAM). Trace is controlled by user defined combinatorial or sequential (state-based and counter-based) event triggering on trace or external trigger signals. This triggering can be used to disable trace until interesting events occur or to trigger on sparse or other irregular events or interest. Trace includes optional timestamping (up to 48 bits) for multi-instrument synchronization or time marking for single cycle or extended time traces. The same triggers can be used to drive debug related actions such as cross triggering between bus and processor or other IP operations.

![Figure 4: JTAG based Internal Trace](image)

Navigator can also be used in an off chip mode that streams the bus trace to a high bandwidth debug port. This allows a smaller RAM footprint, by reducing or in some cases (where trace bandwidth is less or equal to the port bandwidth) eliminating the need for on chip buffering. It also adds flexibility in what types and how much of the triggering and other supporting logic for bus trace is placed on chip rather than being in-line or post processed by configurable logic in the probe. In general off chip trace attempts to minimize the amount of on chip logic to essential triggering and filtering, leaving the trace intelligence (complex triggering, performance analysis, etc.) to be implemented in the probe and essentially using the using the trace port to funnel raw data to the probe as expeditiously as possible. The instrumentation features be implemented in the probe allow a trace interface with a smaller, simple trace logic and memory footprint and much deeper trace depth.
In many cases this trace port can be multiplexed with other bus functions. A trace port provides several advantages over JTAG trace, while imposing other limitations. While JTAG trace allows real time trace constrained by the instrumentation and RAM speed, it requires some silicon area for the instrumentation and RAM (the size of which is largely proportional to instrumentation trigger features and trace depth respectively) and has limited export bandwidth. The trace bandwidth capability for off chip trace is typically limited by the number of IO pins dedicated to export of debug information at any given time, and the speed at which these signals can transmit the data.

Since the limitation on streaming trace is it requires more pins (for reasonable IO width) and may have maximum trace speed lower than operating speed (IO may have limited frequency performance compared to internal IP), it does often make sense to be selective on what bus data is being traced. Bus operations in particular are typically bursty in nature, and a bus may spend a significant amount of time in a quiescent state where no information is being transmitted. Simple filtering and buffering can significantly improve the usable bandwidth in an external trace solution,
In a more complex (multicore) instrumentation environment, external trace is limited by selection of critical data from different sources. The complexity of the funnel allows a range of performance tradeoffs in external trace. By selectively choosing trace signals from different subsystems and instruments, an arbitration scheme that funnels the various trace information for export can increase effective trace bandwidth significantly. As shown in the following block diagram, the use of a trace port and JTAG are not exclusive. In many cases, JTAG remains important for control and configuration regardless of the trace mechanism.

![Figure 7: Streaming Trace from multiple on chip Instruments](image)

**Overview of the System Navigator Pro Trace Probe**

Obviously one of the key requirements for external trace is having a trace probe that is designed for this type of application. The System Navigator Pro™ (SNP) is a high performance dual mode (logic trace and generation) trace probe designed for systems analysis and on-chip debug of SoC designs. It provides a high trace bandwidth (over 20 Gbps) and high memory buffer (up to 2 Gbytes of RAM) trace capability to address the high performance needs of SoC platform analysis.

The SNP interfaces to a target via a user defined combination of two 38-pin mictor connector and up to 20 pin JTAG header interfaces. The dual mictor interfaces allow a range of modes of trace operation, including split mode, where each mictor can be used to trace an asynchronous clock domain. This allows concurrent trace of processor execution and bus operations, which may be asynchronous. The two mictor connectors may also be combined into a single higher bandwidth trace interface supporting trace widths of up to 72 signals. The SNP includes a sophisticated triggering system in the probe, allowing event monitoring of all input signals (for up to 16 different events) as well as sequential triggering (up to 16 states) and performance analysis (configurable...
number of counters). Performance analysis information may be captured along with trace. The JTAG interface supports full 1149.1 JTAG operations as well as JTAG extensions such as MIPS EJTAG at up to a 100 MHz rate (provided the target device can meet this timing). Many different JTAG connection standards can be supported including MIPS, ARM, OnCE, and any other system interface requiring 20 signals or less. Non-JTAG targets can be supported with custom configurations. Both mictor channels and the JTAG interface can operate concurrently to allow trace and JTAG control of the SoC.

SNP includes a 400MHz 32-bit processor in the probe. Using the processor, the trace buffer can be searched or preprocessed to reduce host communication overhead. The API includes the capability to download software drivers into the probe so that tasks can be done locally at high speed rather than remotely from the host PC across a communication link. SNP to PC Host interface is via a speed optimized USB interfaces supporting up to 480 megabits per second transfers and a 10/100/1000-Base-T (Gigabit) Ethernet interface supporting remote SNP interface and control.

The SNP architecture is configurable to meet custom needs, with several preloaded configurations available to address combinations of Multi-processor and multi-bus trace based on either FS2 on chip instrumentation (processor execution and/or data trace, embedded bus analysis, etc.) or customer designed on-chip interfaces. Additional customized probe configurations to address specific trace needs or features of a customer design can be rapidly implemented. One of the more unique features of the SNP, is the mictor connections can also be used in a signal injection mode, as well as a signal trace mode, to allow the SNP to be used as a digital signal generator to inject patterns or write information into a SoC target. The SNP and its supporting software provide the integrated functional equivalent of both a high performance buffered logic analyzer and a programmable signal generator.

SNP supported applications include trace of either/or processor, bus, or general on chip systems operations, across either synchronous or asynchronous domains. It also supports a self contained in silicon verification environment, by supporting concurrent generation of stimulus signal inputs and monitoring and trace of response signal outputs for an embedded design.

![SNP Probe Architecture](image)

**Figure 8 : SNP Probe Architecture**

**An RRT Analysis Environment**

RRT was developed in conjunction with Mobileye for analysis of the complex data communications network of their EyeQ2 SoC. This architecture includes several SMX Crossbar and Shared Link
Exchanges connecting the processor masters (MIPS32 34Kf cores, VCE/VMP cores and DMA) and slaves (off-chip DDR memory and on-chip ISRAM, etc.). The analysis is complex as the both the data width and operating speed of the communication links can vary. The MIPS32 34Kf cores, as an example, operate at twice the system clock frequency and interfaces to the SMX over a 64 bit OCP2 bus. Other bus sockets wide vary from 32 to 128 bits, based on connection to specific cores. The SMX allows for resolving the mixing of bus widths and speeds across different blocks, but efficiency and optimization of performance with regards to different data rates, clock rates, etc. of the system cores is not trivial. For the amount of trace required in the EyeQ application, core and system clock speed is reduced so that the trace port can sustain the required trace bandwidth.

The EyeQ2 system analysis environment consists of two major subsystems:

1. PDtrace (a MIPS processor execution and data trace instrumentation) for each 34Kf core. PDtrace interfaces support an aggregated processor trace port for both of the 34Kf core trace outputs.

2. Bus Request-Response Trace (RRT), which is a bus level instrumentation system that ties into the SMX at the socket level and allows the trace of one single bus socket, all masters in the system simultaneously or a selection of masters. The RRT trace buffers each request-response output and includes a trace “funnel” to route the buffered outputs to the off-chip trace port.

![Diagram of RRT and PDtrace subsystem]

**Figure 9 : RRT and PDtrace subsystem**

RRT and PDtrace data are sent off-chip over a dedicated 16 DDR channel trace port. Both PDtrace and RRT trace port interfaces are supported via a single FS2 System Navigator Pro probe, using two Mictor38 connector interfaces, each with its own independent clock source. The probe combines the trace inputs from the two sources and records them in a common memory buffer.
The Mictor includes a common JTAG connection and PDtrace trigger pins for trigger and trigger acknowledge. This trigger may also be used to put one or more cores in debug mode and to communicate with the processor and (in this example) the MIPS32 trace control and visualization tool, PDtrace. On-chip trigger output pins indicate to the probe the status of the processor core(s). The probe and on-chip logic have a common triggering methodology to allow the probe to enable and disable/stall RRT operations in conjunction with PDtrace operations. The triggering scheme also communicates stalling of the trace capture based on processor status. All applicable features of RRT and PDtrace, including the triggering are configured via the JTAG port.

**RRT Operations**

The RRT provides for capture and collection of the following information of the EyeQ2 operations. All Capture is done on chip at the RRT agents and exported via the RRT port:

a) Recording of specifics of master-slave socket transactions and the number of clocks of delay between each request and response.

b) Captured timing and latency of read cycles. Burst reads are reported on arrival of the first requested word or on the arrival of the last word of the burst.

c) Transactions between one (selected) master and all slaves it transacts with, or several masters at the same time. These masters may include any of - the two 34Kf cores, one active VCE/VMP channel (selected as output of the Crossbar) and one active channel of the DMA.

Trace collection allows overall capture for an extended (for example, at least one video frame) processing period using the Memory buffer in the probe. Concatenation of multiple frames may be done as a post processing stage on exported RRT trace files.

Post-trace software provides post processing and views of transactions and delay times over varying periods of time for both single and multiple cores. RRT data is correlated and used in conjunction with PDtrace data to provide a picture of system operation.

**RRT Implementation**

The on chip component of RRT consists of 3 primary On-Chip Instrumentation (OCI) IP blocks, all of which are implemented in synthesizable Verilog code.

a) RRT agents, specific to processor or core level interface to capture and buffer relevant trace information based on system operations and trace configuration.

b) The RRT “trace funnel”, which provides the aggregation of trace information from all RRT agents and combines and schedules the trace information for export, and

c) The RRT Navigator Trace Port, which handles communications with the off chip probe.

Configuration of each block is performed via JTAG, over a common JTAG chain.
A user defined set of Bus RRT fields may be captured based on connection to the socket. For the bus transaction analysis, signals trace included:

- the master ID (only required if multiple masters are being recorded at one time)
- Slave ID based on unique address bits that identify one slave from another. Hardware in the agents can recognize the memory mapped areas and encode them into the slave ID field
- protocol and traced bits that determine the alignment of a read response cycle to its “parent” request cycle
- request and/or response cycle type (or encoded in other fields)
- Cycle type – read vs. write, single access vs. burst.
- buffer overflow indicator bit, indicating if the RRT record has lost synchronization with the processor operations
- Trace of upper address bits to determine code vs. data memory mapped regions. Two defined modes: Fast (partial) address field and Full (complete address field) are a user selectable options.
- trigger signal – to allow on-chip subsystems to send a trigger signal to the probe.

In order to conserve trace bandwidth, the Bus RRT records are further broken down into two modes – Fast and Full. Fast mode is limited to a single cycle frame and includes socket level control signals characterizing the bus transfer along with buffer overflow and/or trigger indicators. Full mode includes control signals as well as full address trace, based on a memory map of necessary upper addresses, and typically is transmitted over multiple trace clock cycles. The capture of this data via RRT allows the following to be performed during chip level operation:

a. Measurement of a processing loop such as frame time. The SysNavPro trace depth handles 2 Gbyte of trace data (sufficient for example for the trace of at least one video frame).

b. Capturing available information for aligning socket measurements with core processor execution to correlate cause-effect of code execution to socket traffic based on coordinated recording of trace from both sources.

c. Capturing available information on aligning socket measurements to correlate each hardware thread to the data transfers that each processor generates.

d. Extraction of thread information extractable from socket address bits traced. Post-trace software can display per-thread socket transaction information providing valuable information to users on the density of transactions over time and the delays associated with those memory accesses, generated for each hardware thread.

e. Post-processing of the trace matches up requests and responses (using the socket protocol and possibly ID bits), and calculate the delay between them based on timestamp values stored along with trace. Sysnav Pro supports a trace timestamp that provides an accurate timeline of each request-response frame.

A RRT triggering system is implemented within the Probe (off-chip) and includes event monitoring of all captured control and address signals to control start/stop pf capture of
trace information in the probe. This trigger may also be used to put one or more cores in Debug mode and to communicate with the processor and PDtrace subsystems. On-chip trigger output pins indicate to the probe status of the processor cores.

The probe and on chip logic have a common triggering communication to allow the probe to enable and disable/stall RRT operations in conjunction with PDtrace operations. The triggering scheme also communicates stalling of trace capture based on processor status.

**Post-Trace Analysis Tools for RRT and PDtrace**

RRT is supported by a set of control and display views and utilities to support analysis of RRT and PDtrace data. Additional visualization can be supported via export of trace to third-party tools. Control setup includes the setting of master trace priorities and selecting which masters are to be in the trace; trigger setup for precise post-trigger positioning and reading trace and formatting data for additional analysis views. Additional views include:

*Raw State View for RRT* - basic acquisition is displayed as a state display that shows one line per trace frame with columns correspond to the trace fields: transaction type (read/write, request or response), master name, slave name, transaction ID or outstanding request count, buffer overflow and probe generated trace timestamp values.

*Aligned State View for RRT* – alignment concatenates two frames – a request cycle and its matching response cycle and a delta timestamp between current and next transactions.

![Figure 10: RRT Navigator Graphical Bus Trace](image)

*Graphical Display* - FS2 trace solutions are supported by a multi-view (Navigator) GUI, which is customized for RRT data display as captured by the probe. The Analyzer GUI allows complex triggering of capture and display of RRT information as waveform and
state views the. The GUI includes utilities for control of bus event monitoring and template based triggering based on captured trace information

*Correlated view of RRT and PDtrace* - allows viewing of common PDtrace and RRT data captured at a common timestamp with a known or defined offset. It also allows RRT and PDtrace data to be locally correlated based on address values, or common triggers, markers and instruction (read/write/burst) types captured in both the PDtrace and RRT. Correlating socket traffic with instructions defines a processor to bus level relationship, such as determining which thread caused a read or write socket cycle.

Integration of the Bus level and processor tools is integrated via a FS2 proprietary Multicore API layer, which allows user transparent sharing at both the JTAG and trace port resources

![Figure 11: Integrated Bus and Processor Trace environments](image)

**Mobileye EyeQ2™ System Application**

The EyeQ2 architecture was designed for intensive parallel processing targeting the challenges of vehicle vision applications; capable of concurrent detection of vehicles, motorcycles, pedestrians, traffic lights and signs, etc.

The Bus RRT system consists of both on-chip IP and analysis and software communicating over a high performance trace port probe and was developed to provide visibility into the various interconnection points of the Mobileye EyeQ2 architecture.

Central to the SoC design are two instances of MIPS32 34Kf processor cores. The 34Kf core is a multi-threaded RISC architecture designed to exploit multi-threading in embedded applications by processing multiple software latency and allowing the user to allocate dedicated processing bandwidth to real-time tasks. The MIPS32 34Kf cores work in conjunction with an array of Mobileye VCE/VMP processors that are optimized to address...
automotive vision processing applications.

Mobileye EyeQ2 uses two types of proprietary vector processors: VCE (Vision Computing Engine) and VMP (Vector Microcode Processor). VCE DSPs have a fixed logic architecture to address image preprocessing and search operations, including object classification, object tracking, disparity identification (for applications using stereo image) and filtering for vision analysis. The VMP engines provide further image and object processing using a high bandwidth architecture utilizing parallel vector, scalar and look-up table units.

Two MIPS 34Kf cores, Mobileye VCE/VMP processors, the DMA, and peripheral cores are connected by a 128bit 166MHz SMX (OCP) interconnect that services the imaging and processing traffic between these SoC components. An additional low bandwidth 32bit bus connects peripherals and VCE/VMP DMAs’ configuration register via one of SMX ports.

![Diagram of Mobileye EyeQ2 SoC](image)

**Figure 12 : Mobileye EyeQ2 SoC**

The IP developed for the Bus RRT system is designed to record request and response bus events at the socket interface and measurement of one processor (bus master socket) with expandability to allow concurrent viewing of key parameters of all masters simultaneously. This system analysis implementation allows capture of information about core load/store operations and their latency for the different socket masters, and exports them over dual trace ports to the SNP probe, along other trace and analysis data, in particular MIPS EJTAG and PDtrace interfaces that are used in providing a complementary run control and trace analysis views of the MIPS processor operations.

While in this paper we focus on a specific architecture and bus interfaces (MIPS and VCE processor cores), both the on chip interconnect and analysis systems discussed can be applied to other processor cores or bus architectures under a similar generic scheme without any significant loss of generality.

**Other SoC Applications**

The RRT tool chain has general application in several areas of SoC performance analysis and debug. We have discussed how RRT allows the real-time measurement of frame processing...
time in a SoC video processing system that used multiple cores and the Interconnect bus IP. RRT was used to improve the utility of processor trace by capturing information for traces performed at the bus connection point and aligning with core processor execution, enabling correlation of the cause-effect of code execution to bus/socket traffic based on the time-based coordination of recorded traces from both sources.

RRT is also used in more general systems based on the multithreading MIPS32 34K processor. By tracing the bus interconnect to memory, RRT gives the developer real-time bus latency information and shows whether the interconnect is stalled at the precise time of a requested load/store operation. However, this does not provide information on which 34K hardware thread is running nor what part of the application code is running at that time. By correlating the trace information from the RRT bus socket(s) and the CPU trace obtained from PDtrace it is possible to get a complete picture of how load/stores from each thread of execution are impacting overall system operation.

Leveraging this information one step further, optimization of the QoS system used to schedule threads in a multi-threaded MIPS32 34K core operating within a SoC design can be achieved. For example, RRT can be used to extract thread information from socket address bits traced in a system using a MIPS32 34K core. Post-trace software can then display per-thread bus/socket transaction data, providing valuable information to users on the density of transactions over time and the delays associated with those memory accesses, generated for each hardware thread. This detail of information is extremely useful for performance tuning the application of software threads of execution to the hardware threaded capability of a multithreaded processor, allowing system developers to optimize bus utilization and throughput in such a complex SoC design.

Summary:

Many applications, Mobileye EyeQ2 SoC compute-intensive vision processing being only one, place a high priority on analysis of communications between processing resources. Internal bus traffic visibility and measurement capabilities are an important component of this system analysis since in complex SoC environment, software performance can be dramatically impacted by a range of problems caused by poor bus traffic balancing, bad QoS management, unprotected concurrent memory access, incorrect cache coherency management, etc. Poor on-bus traffic balancing causes latencies that can seriously impact the real-time capabilities and for applications that are data dependent, a real time analysis environment with in-depth on-chip data access is valuable. One major driver for a high performance bus traffic monitoring tool is enabling statistical measures of bus latencies. The Bus RRT approach provides this by combining flexibility of fast (partial)/full bus tracing at almost real-time rates with powerful post-trace analysis correlating on-bus and on-CPU events.

The Bus RRT system consists of both on-chip IP and analysis and software communicating over a high performance trace probe and was developed to provide visibility into the various interconnection points of SoC architectures. The IP developed for the Bus RRT system is designed to record request and response bus events and measurement of one processor (bus master) with expandability to allow concurrent viewing of key parameters of all masters simultaneously. This system analysis implementation allows capture of information about
core load/store operations and their latency for the different bus masters, and exports them over dual trace ports to the SNP probe, along other trace and analysis data, in particular MIPS EJTAG and PDtrace interfaces that are used in providing a complementary run control and trace analysis views of the MIPS processor operations. The complexities involved in these interconnect centric designs and their flexibility and parameterization make embedded features for sophisticated latency and performance analyses an important consideration, not just to further optimize the current design, but to quickly and easily quantify requirements and enhancements for a systems optimization.

An important feature of RRT is ability to correlate and resolve operations in the bus fabric and processor architectures and processes to provide understanding of system operations. For integrated MIPS applications using PDtrace and RRT, a bus event that can be matched per CPU and even per TC thread, with zero-impact on processing timing, by selectively and passively tracing, compressing, and transferring data at both bus and processor levels through a common Navigator probe. Together with PDtrace, RRT gives a serious advantage for system analysis compared with other monitoring alternatives.

Productivity of this solution was also an important criterion for focusing on trace based system analysis. Mobileye estimation for RRT productivity compared with a SystemC model alternative has showed up to 100 times speed advantage. From a business model, RRT is much simpler than SystemC model based analysis approaches, once third party IP integration overhead and all expenses of licenses for external developers are factored in.

Comprehensive RRT control by a flexible triggering system is an additional “plus”. And tracing control enables optimal utilization of on-probe memory buffer and conditional debugging control.

The RRT system is a result of close cooperation between Mobileye and FS2 and the support of MIPS Technologies and Sonics, as IP providers. The system provides a full, cheap and efficient tracing/debugging solution for a complex dual-MIPS32 34Kf SoC environment. While this paper discusses one specific bus architecture and its interfaces to MIPS processor cores, both the interconnect and analysis systems based on other processor cores or bus architectures can be applied under a similar generic scheme as discussed without any significant loss of generality.

References:


MIPS, MIPS32, 34Kf, OCI, RRT, Bus Navigator and FS2 are trademarks or registered trademarks of MIPS Technologies, Inc. in the United States and other countries. SMX and SMART Interconnects are trademarks of Sonics Inc. EyeQ2, VCE, VMP are trademarks of Mobileye Vision Technologies. Ltd. All other trademarks referred to herein are the property of their respective owners.