FPGA Based CPU Instrumentation for Hard Real-Time Embedded System Testing

Richard Fryer, Department of Computer Science
California Polytechnic State University, San Luis Obispo, CA 93407
rfryer@calpoly.edu

Abstract

The addition of system instrumentation features have been sporadically incorporated into processor architectures over the last several decades. Particular emphasis areas of high performance, embedded and real-time computing are reviewed in terms of software and hardware measurements and approaches representing active research directions. A novel approach is described that may be readily used with recent advances in Field Programmable Gate Array technology using embedded processors. The approach and preliminary results are described using a Xilinx device with a MicroBlaze™ 32 bit architecture. Some system level problems are outlined and examined.

1. Real Time Non-Intrusive (RTNI) monitoring

The concept of RTNI instrumentation of embedded systems, whether the goal is system debug or system performance measurement depends on key requirements. The instrumentation mechanisms must be *transparent* to the behavior of the software. The goal of 'transparency' is to maintain critical aspects of program behavior. First, the flow of addresses in program execution should be identical between instrumented and 'native' code. Secondly, time relationships of all software detectable events should be equivalent (i.e. order of interrupts as well as location in program execution when interrupts occur; sequence of procedures run, time spent in each process, etc.). Kirshon et. al. state [1] that the term 'consistent' is used in this way, and dictates that monitoring must '*not • • • change program behavior*.'

There is strong focus in embedded system engineering in testing, integrating and validating the code that will be used operationally. Instrumentation code unquestionably changes program behavior, and for this reason, when special code is used for testing, it is also usually required to be part of the operational system [2,3].

A number of special hardware approaches have been either proposed and implemented [4, 5, 6, 7]. Methods described include both those that add new functionality (visible to the programmer) to processor architectures. For example, the method described in [4] adds special output hardware that may be programmed to emit desired measurement signals. This attractive approach reduces the instrumentation overhead by using this special hardware to carry out most instrumentation activities. A disadvantage of this approach is the rather (procedurally) difficult need to recompile instrumented code to change test conditions and scenarios and the lack of provision to also correlate system hardware or controller features *outside* the software on the target processor.

Much progress has been made using variations on LeBlanc's REPLAY concept [8] has been extensively developed [9, 10]. The approach described herein owes much to REPLAY.

2. Software Instrumentation Goals

Beyond determining cause of errors, however, there are several other needs for Real-Time system designers. Modeling software as a component of a detailed hardware/software co-design requires a fairly detailed model of software behavior [6]. A persistent experience in embedded system design is that interaction of software and hardware components are sources of many surprises to system designers and implementers. This experience motivates even more careful modeling of the software/hardware components as well as their

interactions. Careful validation of these models is essential.

RTNI features must be designed to provide the required data for successful software model construction and validation [5]. Parameters that should be measured (when possible) or estimated to develop models include as examples: [Task initiation times, lifetimes & invocation rates; Delay (between events) and distribution of delays; Signaling Distributions; Comparison of Alternate Actions; Frequency & distributions of interrupts; I/O block distributions and sizes; Distribution of values of selected variables; and Coordination delays between multiple processors].

Fortunately, support for system testing – both validation and anomaly identification[1][11] calls for measuring many of these same parameters. The major differences between these broad goals for instrumentation are *Tracing* and *Complex Triggers*. Tracing here refers to the need for debugging equipment to be able to adequately store execution history information as well as variable accesses histories to provide clues to the cause of improper behavior. 'Complex triggering' is the notion of allowing a tester to use various pieces of information that he has about the system behavior and clues derived from ongoing test activities and traces to narrow down the search space to trap problem causes. To complete anomaly detection, we use a process we identify as *re-execution*' which, similar to REPLAY, allows for ambiguities in execution to be resolved.

Only the needs for history data and code coverage in tests places requirements on the processor instrumentation support needed beyond that needed for performance measurements and modeling – and in fact, these are the driving requirements. The other characteristics listed can be implemented in external hardware.

Finally, the approach described can become a resource for use in the SIMPLEX approach [12] to graceful evolution of systems, a technique being integrated into Real-Time systems due to the extensive test requirements for those systems. The SIMPLEX approach requires some significant changes to baseline software, an aspect that can be improved. The RTNI approach described affords the possibility to provide software comparisons required in SIMPLEX with significantly reduced software changes.

[1] Anomaly detection will be loosely equated to 'debugging' in the following.

3. RTNI MEASUREMENT DETAILS

This section addresses the demands placed on a RTNI capability to provide the required data for each of the goal areas outlined above. Several aspects of current RISC technology need to be kept in mind in terms of their impact on these measurement needs. The major concerns include: Pipelining, Speculative Execution, Multiple Execution Units and Register Renaming. These will be extensively treated in a final paper.

The system in development extends the MicroBlaze™ 32 bit RISC architecture developed by Xilinx [13]. Figure 1 below shows the architecture of this 'soft' processor. This system runs in many different FPGA models with speeds currently exceeding 500 Dhrystone MIPS.

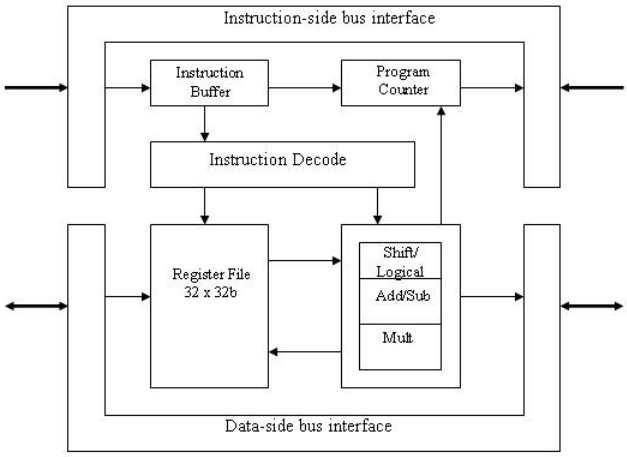

Figure 1 Architecture block diagram of the Xilinx MicroBlaze™

Visibility of all busses, including the bus between the L1 interface and the processor provides excellent access to all needed signals for high fidelity RTNI support and for re-execution in special cases..

Figure 2 below shows this block diagram in context with the rest of the CPU and peripherals, and with the add-on features needed to provide for RTNI, SIMPLEX, and the other features described above.

On-board instrumentation resources are required for these instrumentation instructions to call on. These are outlined the following Figure.

The on-chip counters provide a method of totaling statistics required for measurements as described in sections 2 and 3.

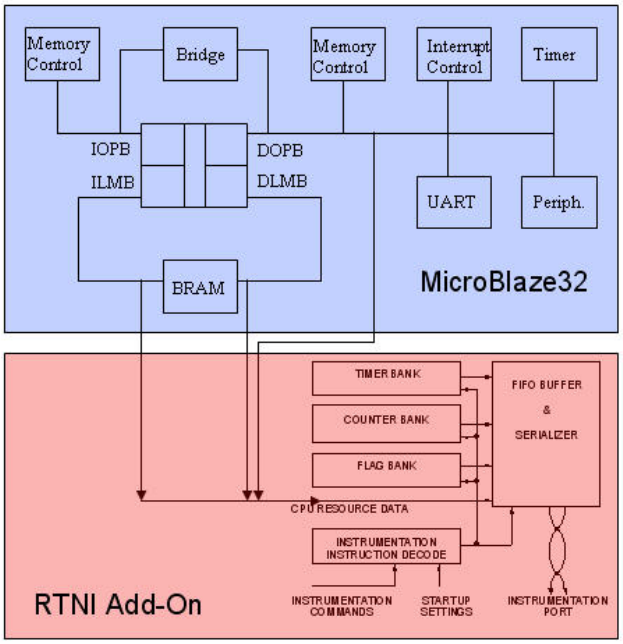

Figure 2 Expanded MicroBlaze™ diagram with RTNI add-on.

Counter and flag logical assignments are made with the instrumentation compiler. Counters and flags are logical and are mapped to physical resources similarly to techniques used within the basic CPU design.

The resource flags may set at one location within instrumentation memory and be tested by later Instrumentation Memory instructions to determine whether to emit an event.

A fairly common debug step is to capture specific internal CPU state information at the occurrence of an event. This is provided for by providing access to these signals for saving into the FIFO.

Complete tracing is almost always an unrealistic approach in instrumentation as processors by their nature press the storage size and speed limits with each processor generation. A successful instrumentation strategy must limit data using complex conditions – or 'instrumentation filters.' In early systems, banks of high speed comparators of various types were used; pairing up to bracket procedures and data structures.

The purpose of using these conditions is to reduce the bandwidth required in the instrumentation bus and to reduce the size of the resulting dataset that must be processed to extract the desired metrics. Recall that a typical embedded system may not recycle all tasks for seconds or minutes, and the target processor is executing at hundreds of mega-operations per second with tens of processors. Finally, we employ a high speed serial link, readily available on this specific (and other vendor's components) FPGA parts.

Some of these data-reduction approaches can be accomplished with the resources shown in Figure 2, but another concept can help even more.

5. INSTRUMENTATION MEMORY CONCEPTS

The final instrumentation concept incorporated has been dubbed the 'Instrumentation Memory' – a concept developed by the author but impractical to incorporate in most COTS processor concepts [5]. The concept was not implemented at that time due to the then expense of the high speed memory required by the architecture. The concept is outlined in Figure 3 below.

In this concept, a second memory fetch stream is matched word-for-word with the operational software execution in each processor. This memory has a small data field, each 'op-code' of which defines instrumentation functions. By monitoring the CPU address register, the Function Memory can execute the 'instrumentation program' in lock-step. In this manner, each location of memory becomes in effect an independent comparator with a measuring function that may be associated with each location. This may be likened to a 'watchpoint' with the option to take any instrumentation step at the pause - though in this case there is no pause in processor execution. The opcode structure for the instrumentation memory must of course allow for functions that may overlap addresses.

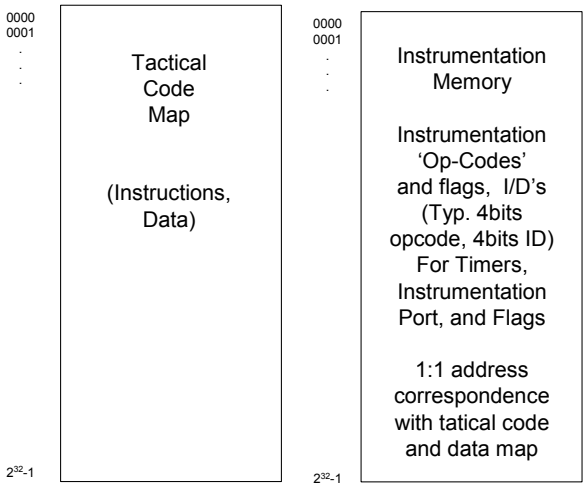

Figure 3 Overview Of Instrumentation Memory

6. SUMMARY

Extracting high data-rate software instrumentation data from a system is equivalent to the basic system I/O problem. Fortunately, emerging processor resources now have real estate available to provide for number of high performance multi-gigabit serial ports - attractive as pin costs may be much higher than silicon costs. The writer thinks that this method of data extraction has significant potential for standardization, as did the now popular JTAG approach. The more recent Nexus standard effort [14] is not incompatible with this approach.

Another approach to reducing the bandwidth requirement is to encode the data using standard compression schemes. The nature of the data indicates that a factor of 2X speed improvement in the data exporting port may be expected from this alone, though studies are incomplete.

System modeling and testing - especially for embedded hardware/software intensive systems - requires accurate and trustable models of hardware, software and hardware / software interactions and measured behavior.

The RTNI approach outlined, enhanced with the Instrumentation Memory design provides a number of means of acquiring the needed data from such systems using new processors. True non-intrusive data capture for REPLAY along with enhanced SIMPLEX operation are direct benefits of the approach.

Specific hardware impact (FPGA resource requirements) are in detailed assessment at present.

5. REFERENCES

[1] Kishon, A., P. Hudak and C. Consel. "Monitoring Semantics: A Formal Framework for Specifying, Implementing, and Reasoning about Execution Monitors", *Proceedings of the ACM Sigplan '91 Conference on Programming Language Design and Implementation*, Toronto. ACM, New York, New York, June 26-28, 1991. pp 338-352.

[2] Xu, M. Bodik, R. and Hill, M. "A 'Flight Data Recorder' for Enabling Full-system Multiprocessor Deterministic Replay", 30th International Symposium on Computer Architecture (ISCA 2003), 9-11 June 2003, San Diego, California, USA. IEEE Computer Society 2003, ISBN 0-7695-1945-8 pp. 122-135.

[3] Harelick, M. and Stoyen, A. "Concepts from Deadline Non-Intrusive Monitoring", 24th *IFIP Workshop on Real-Time Programming*, Saarland, Germany, May 1999.

[4] Carpenter, R.. "Performance Measurement Instrumentation at NBS", *Proceedings of the Workshop on Instrumentation for Future Parallel Computing Systems,* Sante Fe, New Mexico. ACM Press, New York, New York. May 1989. pp. 159-184.

[5] Lemon, L. M., "Hardware system for developing and validating software", *Proceedings of the 13th Asilomar Conference on Circuits, Systems and Computers.* Pacific Grove, CA, November 1979. IEEE Piscataway, N.J. pp. 455-459.

[6] Cannon, W.J., M. T. Michael, and D. D. Beeson, "Real Time, Non-Intrusive Instrumentation of Reduced Instruction Set Computer (RISC) Microprocessors", *Proceedings of the National Aerospace and Electronics Conference (NAECON).* Dayton, OH. IEEE Piscataway, N.J. May 1992. pp. 550-557.

[7] Shobaki, M.E. and Lindh, L., "A Hardware and Software Monitor for High-Level System-on-Chip Verification," in International Symposium on Quality Electronic Design (ISQED), March 2001 pp56-61.

[8] LeBlanc, T.J. and J.M. Mellor-Crummey, "Debugging Parallel Programs with Instant Replay," in IEEE Transactions on Computers, Vol. C-36, No. 4, Apr. 1987 pp. 78-86.

[9] Thane, H., Sundmark, D., Huselius, J. and Pettersson, A., "Replay Debugging of Real-Time Systems using Time Machines", International Parallel and Distributed Processing Symposium (IPDPS '03), IEEE Piscataway, N.J., April 2003,pp 288-295.

[10] Ronsse, M. et. al. "Record/Replay for Nondeterminstic Program Executions", Communications of the ACM, V46 #9, September 2003, pp. 62-67

[11] Telles, Matt, and Y. Hsieh, 2001 *The Science of DEBUGGING,* Coriolis, Scottsdale, AZ.

[12] Sha, L. "Dependable System Upgrade", Proceedings of the IEEE Real-Time Systems Symposium, December 02-04, 1998 pp. 440-449.

[13] Spencer Isaacson, Doran Wilde: *"The Task-Resource Matrix: Control for a Distributed Reconfigurable Multi-Processor Hardware"* RTOS. ERSA 2004: pp. 130-136

[14] O'Keeffe, H. "IEEE-ISTO-1999, the Nexus 5001 Forum Standard," in *IEEE-ISTO Forum, (January 2000).*