Demo: AVEKSHA – A Hardware-Software Approach for Non-intrusive Tracing and Profiling of Wireless Embedded Systems [*]

Matthew Tancreti
Purdue University
West Lafayette, IN, USA
mtancret@purdue.edu

Mohammad Sajjad Hossain
Purdue University
West Lafayette, IN, USA
sajjad@purdue.edu

Saurabh Bagchi
Purdue University
West Lafayette, IN, USA
sbagchi@purdue.edu

Vijay Raghunathan
Purdue University
West Lafayette, IN, USA
vr@purdue.edu

Abstract

In this demo, we present AVEKSHA, a system for non-intrusive tracing of execution at a high spatial and temporal granularity suitable for an embedded wireless node, i.e., in a low-cost manner and one that can be deployed at a large scale[1]. AVEKSHA is based on an insight that most processors, including low-cost embedded processors, offer visibility into their internal workings through an On-Chip Debug Module (OCDM), whose signals are exposed through a standard JTAG interface. This interface has been used by embedded system engineers primarily for interactive debugging, such as single stepping, showing values of registers, *etc*. We show how this visibility, together with the fact that most OCDMs provide a general-purpose method of setting triggers, can be leveraged in AVEKSHA to perform automated tracing in a deployed setting.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging Aids, Tracing*

General Terms

Design, Measurement

Keywords

Wireless Sensor Network, Debugging, Tracing, JTAG

1 Introduction

It is often important to get an idea of the events occurring in an embedded wireless node when it is deployed in the field in a remote location, away from the convenience of an interactive debugger. Such visibility can be useful for various purposes — for debugging any problem *a posteriori* in the lab, by recreating the exact sequence of events that the node experienced in the deployment (this approach is called "record and replay-based debugging") [1, 2]; for profiling the operation of a node for the performance of its various software components and the energy consumed by different hardware and software components on the node [3, 4, 5]. As an example of the latter use case, a system owner may be interested in figuring out which software component is being invoked most often and which software component is consuming the most energy per invocation. It is often not possible to do these determinations in a lab setting because the events that the node experiences in the deployment cannot be recreated in the lab and the events (and even their sequence) can have a bearing on these questions.

We would like to have visibility at a fine granularity - both spatially and temporally. Spatially fine visibility implies that it should be possible to trace *individual* events of interest as opposed to only bursts of events (clearly, tracing *every* event is likely to be prohibitive) and it should be possible to trace performance and energy at fine code regions, such as a function or a task (using TinyOS terminology). This is desirable because the fine region of code can then be debugged if it is determined through performance profiling that this region is causing a performance bottleneck, through energy profiling that it is consuming unexpectedly large amounts of energy, or through record and replay that it is the source of a bug. Temporally fine visibility implies that it should be possible to do the tracing with a high sampling frequency. Clearly, the two dimensions are not independent. In order to trace small regions of code in a loop, it is necessary to be able to trace at a fine temporal granularity.

While the problem motivation laid out above has been clear to researchers for quite some time [6], it has proved very difficult to provide a solution for low-cost embedded wireless nodes that can operate at a large deployed scale. The first line of attack has been to provide pure software solutions [7, 1, 2]. Such solutions have perturbed the application too much to be useful for many of the use cases indicated above. For one, they change the timing behavior enough that some bugs get suppressed. Else, they cause such a large slowdown in the application execution that it is not possible to employ them in a deployed setting. To get around this problem, a recent software solution [2] has focused on a specific kind of tracing (control flow tracing) and intelligent static analysis and runtime trace collection, compression and storage. Thus, it addresses one of the above usage scenarios. The second line of research has developed hardware solutions for sub-

[*]This work was supported in part by NSF grants CNS-0953468, ECCS-0925851, and CNS-0716271. The views expressed represent those of the authors and do not necessarily reflect the views of the sponsoring agency.

[1]AVEKSHA is a Sanskrit word that means "to monitor".

SenSys'11, November 1–4, 2011, Seattle, WA, USA.
ACM 978-1-4503-0718-5/11/11

sets of the usage scenarios laid out above. For example, [5] has developed a dedicated integrated circuit, implemented using an FPGA, that is tightly integrated with the host processor and its peripherals and can measure energy drawn accurately at millisecond resolution. Quanto [4] is a solution that de-emphasizes sophisticated hardware design. Instead, it measures energy at the node level, uses indication from device drivers about changes in power state, and performs causality tracking to pin down energy usage due to individual activities. Thus, Quanto is a hardware-software solution, and like all prior solutions that have a software part, is OS-specific (in this case, TinyOS). At the high-end, tools such as Green Hills Software's SuperTrace probe and TimeMachine are available, however, such solutions are expensive and not available for many low-end embedded processors [8].

In summary, our problem statement is the following: *How to perform non-intrusive tracing of execution at a high spatial and temporal granularity suitable for an embedded wireless node, i.e., in a low-cost manner and one that can be deployed at a large scale?*

We make the following claims to novelty and practical feasibility from AVEKSHA:

1. We present the first technique for non-intrusive tracing of a wide variety of events, including arbitrary user-defined events, in embedded wireless nodes.
2. Our tracing technique is agnostic to the operating system, compiler infrastructure, or language in which the application is implemented.
3. Our hardware is built using off-the-shelf components and requires little effort in integrating with the application board, which is modified only very slightly for enabling the tracing.
4. Our solution is suitable for deployment at a large scale because it is low cost, can operate on battery power, and extracts program information directly from the application processor.

2 Design

We develop a debug board formed of standardized components – a microcontroller unit (MCU), which in our case happens to be the same as the application processor, MSP430F1611 from Texas Instruments, and an Actel FPGA, both of which interact with the OCDM on the target application processor over the JTAG interface. We refer to our debug board as the Telos Debug Board (TDB) because it is intended to be used with the Telos wireless sensor node (however, our solution is not restricted to the Telos and can easily be adapted to other embedded platforms based on the MSP430 microcontroller, and with some effort to other embedded platforms). The MSP430 OCDM (also referred to by the microcontroller datasheets as the Enhanced Emulation Module or EEM) allows AVEKSHA unprecedented visibility into the state of the application processor. Further, the OCDM has a small circular buffer where events of interest can be stored and subsequently drained by the FPGA on the TDB. The triggering mechanism of the OCDM is very flexible and is therefore attractive for AVEKSHA. For example, the OCDM can be triggered to indicate when the application processor has accessed a certain memory region or a peripheral device, such as a sensor. We find that the triggering mechanism can be combined with thoughtful design to trace all the events of interest for our three usage scenarios – performance profiling, energy profiling, and record-and-replay.

AVEKSHA operates in one of three modes: breakpoint, watchpoint, and program counter (PC) polling. Breakpoint is useful in some contexts, however, it is intrusive and therefore does not meet our solution requirements. The watchpoint mode has AVEKSHA set triggers, where each trigger unambiguously maps to an event of interest (such as when a sensor is read). When a trigger fires, the application processor is not stopped, but the state is dumped to a buffer on the OCDM, which is then emptied out by AVEKSHA. This is a rate-limited operation and if events of interest happen with a high enough frequency, the buffer overflows and AVEKSHA misses some events of interest. In the PC polling mode, the TDB tracks the program counter values of the application processor without interrupting it. Then, it processes the PC values to determine events of interest, such as when control flow has entered a particular function. These three modes reveal different tradeoffs in terms of intrusiveness, the flexibility in defining which events to collect, and the rate at which collection can be done.

3 Demonstration

We will show how the TDB can be used to trace events and measure energy consumption using two representative applications, one in TinyOS and one in Contiki. For the TinyOS application, we will display a live trace of the tasks and state transitions of the radio stack, as packets are being received by a Telos mote. This is useful in understanding the operation and energy consumption of the radio stack. The Contiki application uses light readings to track movement. We will display a trace of application processes, and indicate how they correlate to tracking. The demonstration shows that the TDB is OS agnostic, no reprogramming is required when switching between the TinyOS and Contiki applications.

4 References

[1] M. Wang, Z. Li, F. Li, X. Feng, S. Bagchi, and Y.-H. Lu, "Dependence-based multi-level tracing and replay for wireless sensor networks debugging," in *Proceedings of the 2011 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, LCTES '11, (New York, NY, USA), pp. 91–100, ACM, 2011.

[2] V. Sundaram, P. Eugster, and X. Zhang, "Efficient diagnostic tracing for wireless sensor networks," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, (New York, NY, USA), pp. 169–182, ACM, 2010.

[3] X. Jiang, P. Dutta, D. Culler, and I. Stoica, "Micro power meter for energy monitoring of wireless sensor networks at scale," in *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, (New York, NY, USA), pp. 186–195, ACM, 2007.

[4] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: tracking energy in networked embedded systems," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, (Berkeley, CA, USA), pp. 323–338, USENIX Association, 2008.

[5] T. Stathopoulos, D. McIntire, and W. J. Kaiser, "The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes," in *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, (Washington, DC, USA), pp. 383–394, IEEE Computer Society, 2008.

[6] H. Thane, D. Sundmark, J. Huselius, and A. Pettersson, "Replay debugging of real-time systems using time machines," in *Parallel and Distributed Processing Symposium*, IPDPS '03, (Washington, DC, USA), pp. 288.2–, IEEE, 2003.

[7] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic, "Achieving repeatability of asynchronous events in wireless sensor networks with envirolog," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1 –14, april 2006.

[8] "Green hills software inc." http://www.ghs.com/.