AVEKSHA: A Hardware-Software Approach for Non-intrusive Tracing and Profiling of Wireless Embedded Systems

Matthew Tancreti, Mohammad Sajjad Hossain, Saurabh Bagchi, Vijay Raghunathan

School of Electrical and Computer Engineering, Purdue University

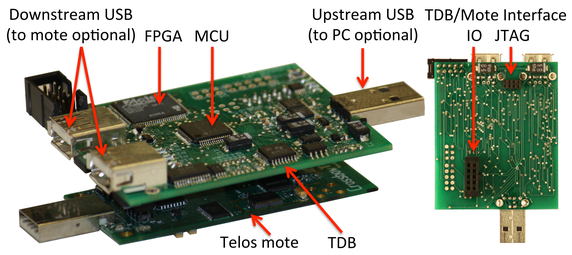
Problem Statement

- Current approaches to debugging deployed Wireless Sensor Networks (WSN) can be improved
  - Software: affects timing and is OS specific
  - Hardware: bench debuggers not suitable for deployment
- How can we perform tracing and profiling of software that is
  - Non-intrusive
  - Able to monitor at high spatial and temporal granularity
  - Low energy
  - Low cost
  - Easy to integrate and deploy

Solution Approach

- AVEKSHA is a hardware/software approach
- Exploit on-chip debug module (OCDM)
  - Comes free on most MCUs (also called EEM on MSP430)
  - Exposed through JTAG interface
  - Asynchronous with MCU operation
  - Advanced features: complex breakpoints and watchpoints, triggered state-storage
- Approach
  1. Poll OCDM state
  2. Filter for important events
  3. Store to flash or stream over USB

What We Built: The Telos Debug Board



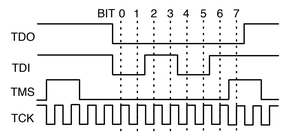
- Connects to mote IO and JTAG
- Has an MCU for initialization and configuration
- Has an FPGA for high speed polling of OCDM state

Our Contributions

- Reverse engineered an important JTAG protocol (MSP430)
  - A common low-power sensor network MCU
  - Enables profiling and tracing for this class of MCU chip
- Designed a HW/SW debugger suitable for deployed WSN
  - Non-intrusive (does not alter software timing)
  - OS and compiler agnostic
  - Low power
  - No significant hardware modification to mote
  - Easy to deploy (does not need to be customized per application)
- Validated design through case studies
  - Tracing and profiling in TinyOS and Contiki
  - Found resource consuming bug in TinyOS low-power-listening radio stack

Interfacing to the OCDM over JTAG

- JTAG interface uses 4 pins
  - TDO data output
  - TDI data input
  - TMS select mode
  - TCK clock



- MSP430 OCDM responds to various commands sent over JTAG
- We use the following command sequences
  - Set watchpoint and breakpoint triggers
  - Poll CPU status (e.g., if halted at a breakpoint)
  - Poll the state-storage buffer (for information stored at watchpoint trigger)
  - Poll the program counter (PC)
- These sequences map to 3 operation modes of the board: watchpoint (WP), breakpoint (BP), and pc polling

Tracing and Profiling Modes

- Watchpoint polling
  - Total of 8 triggers can be set through JTAG
  - Each trigger specifies conditions based on the value of the data or address bus (MDB/MAB), and the operation: read, write, or instruction fetch (-R/-W/-F)
  - Triggers can be set for generic events e.g., function call and return
  - A nop instruction can be used by the programmer to specify arbitrary trigger locations in code

Event	Condition	# Triggers
Function call	MDB-F==0x12B0	1
Function return	MDB-F==0x4130	1
Interrupt	MAB-R≥0xFFE0	1
Interrupt return	MDB-F==0x1300	1
Peripheral read	0x0010≤MAB-R≤0x01FF	2
User defined	MDB-F==0x4404	1

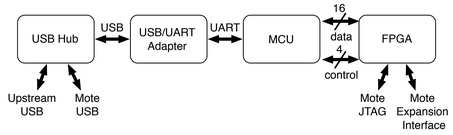
- PC polling
  - OCDM allows continuous polling of program counter
  - Provides information about program control flow
  - PC value can be mapped to a code block or function

Speed of Polling

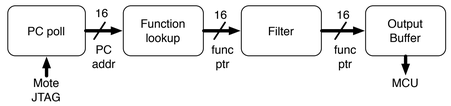
Mode	Operation	Software (μs)	FPGA (μs)
Breakpoint	Test	43	3.8
	Read Addr.	140	12.2
	Resume	77	6.8
	Total	*260*	*22.8*
Watchpoint	Test	200	18.3
	Read Addr.	140	12.2
	Total	*340*	*30.5*
PC Polling	Read PC	48	1.6
	Total	*48*	*1.6*

- Watchpoint mode
  - State-storage buffer is 8 entries
  - Each poll and read of state buffer takes 122 mote cycles
  - Cannot exceed burst of 8 events in 976 mote cycles
  - For example, suitable for monitoring task execution and state transitions in TinyOS but not function calls
- PC polling mode
  - Only provides PC values, cannot get MDB and MAB values
  - Each PC poll takes 7 mote cycles
  - Suitable for function call granularity

Design

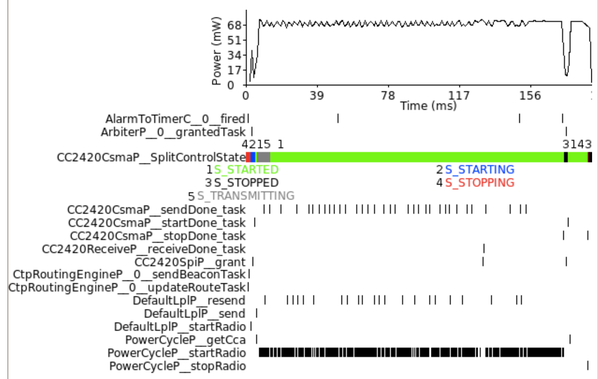


- FPGA for polling JTAG and filtering events
  - Would require MCU at much higher clock frequency
- MCU for initialization and coordination
- USB connectivity to stream data to host PC



- Goal is to poll JTAG at maximum possible speed
- For PC polling pipeline stages are: poll PC address, binary search for function pointer, filter: is this a new function, output buffer

A Bug Discovered in TinyOS



- While testing we discovered a bug in TinyOS
- PowerCycleP_startRadio task re-posts itself during startup when SubControl_start() != SUCCESS
- But, when radio already started SubControl__start() = EALREADY

```
static inline void PowerCycleP__startRadio__runTask(void) {
  if (PowerCycleP__SubControl__start() != SUCCESS) {
    PowerCycleP__startRadio__postTask();
  }
}
```

- Following hypothetical code shows how re-post can go on indefinitely

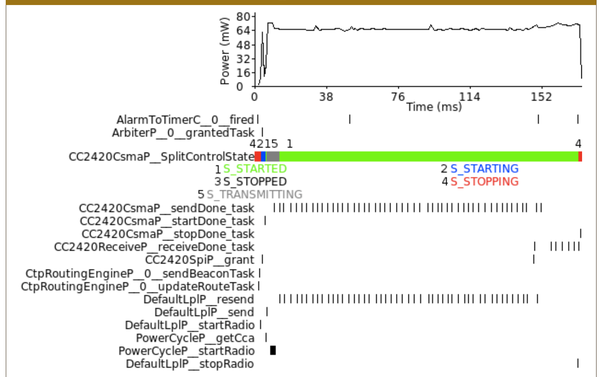
```
event void RadioControl.startDone(error_t err) {
  sendMessage();
  // Schedule the timer to fire while
  // PowerCycleP__startRadio is spinning
  call Time.startOneShot(100);
}
event void Timer.fired() {
  call LowPowerListening.setLocalWakeupInterval(0);
}
```

- A simple fix is to add a condition for EALREADY

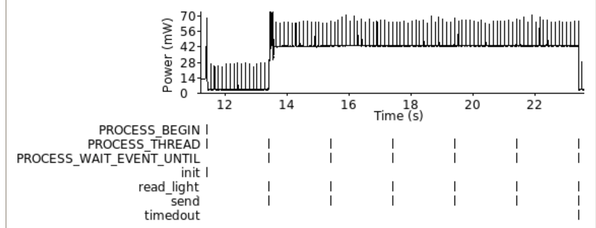
```
static inline void PowerCycleP__startRadio__runTask(void) {
  if (PowerCycleP__SubControl__start() != SUCCESS
  && PowerCycleP__SubControl__start() != EALREADY) {
    PowerCycleP__startRadio__postTask();
  }
}
```

- Now patched in TinyOS repository (bug tracker issue 51)

Bug Fixed

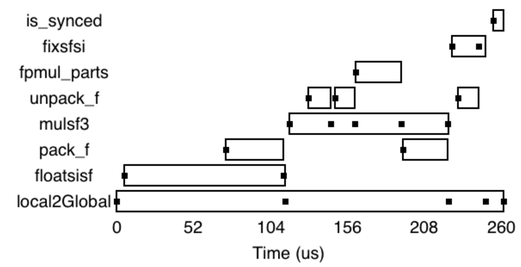


OS Agnostic



- Figure shows processes of a Contiki light tracking application
- We can change from debugging TinyOS tasks to Contiki processes without reconfiguring the board

Profiling with PC Polling



- PC polling down to granularity of 7 mote cycles
- Usually enough to catch every function transition
- Knowledge of call graph required to convert polled PC data into profile

Acknowledgements

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.

Reference

M. Tancreti, M. S. Hossain, S. Bagchi, V. Raghunathan, “AVEKSHA: A Hardware-Software Approach for Non-intrusive Tracing and Profiling of Wireless Embedded Systems,” in Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys ’11, ACM, 2011.