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

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

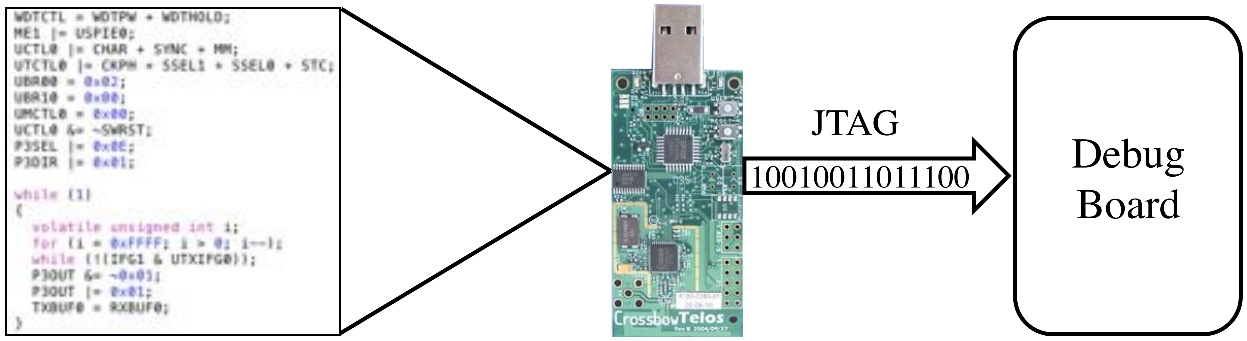
School of Electrical and Computer Engineering
Purdue University

Problem Statement

- Debugging deployed Wireless Sensor Networks (WSN)
  - Software: profilers affect timing and are OS specific
  - Hardware: bench debuggers not suitable for WSN deployment

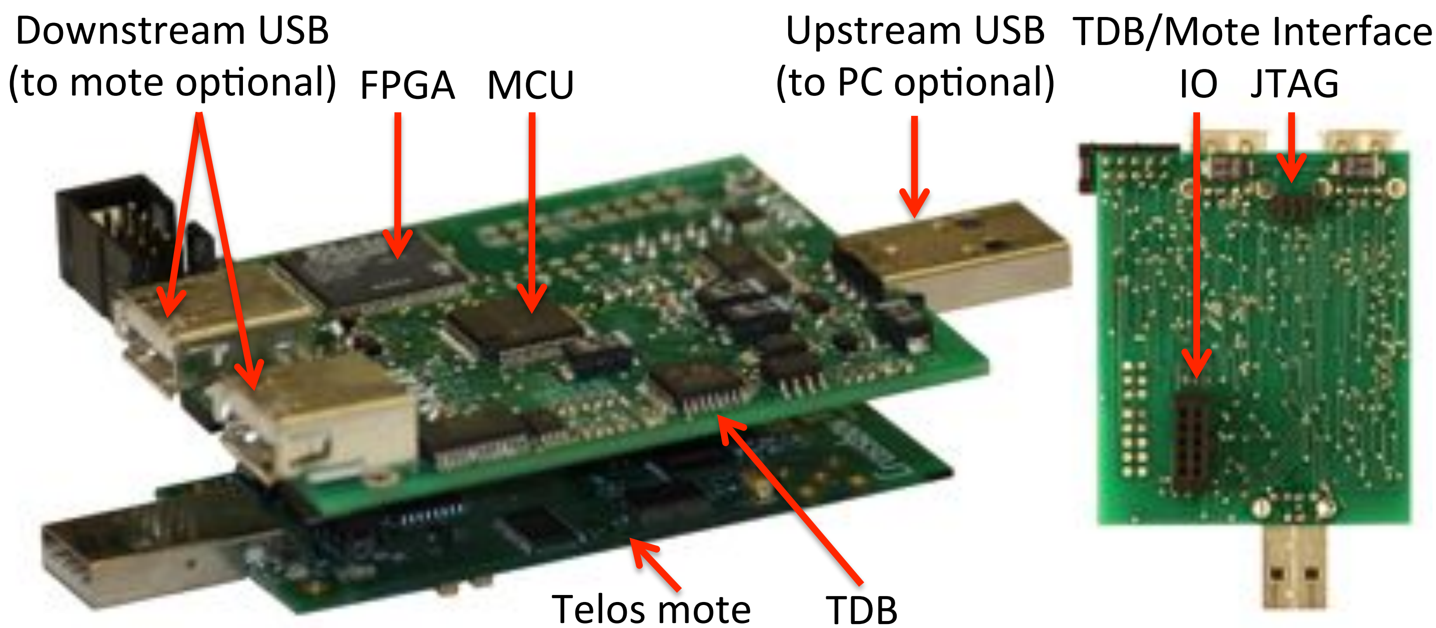
- How to perform tracing and profiling of software
  - Non-intrusively
  - With high spatial and temporal granularity
  - Low energy
  - Low cost
  - Easy to integrate and deploy
- Tracing provides a sequence of events useful for debugging
- Profiling determines energy consumption and time per event

Solution Approach: AVEKSHA



- 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 triggers for breakpoints and watchpoints, store state on trigger

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

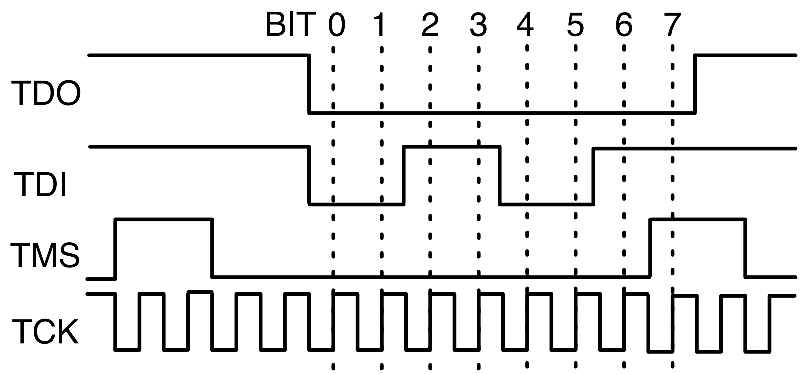
- Reverse engineered important JTAG protocol (MSP430)
  - Common low-power sensor network MCU
  - Enables profiling and tracing for this class of MCU chips
- Designed a HW/SW debugger suitable for deployed WSN
  - Non-intrusive (does not alter software timing)
  - OS and compiler agnostic
  - Low power
  - 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

Presentation Outline

- OCDM Background
- Design
  - Hardware
  - Firmware
  - Energy monitoring and other features
- Case studies
  - Tracing TinyOS tasks and states
  - A TinyOS bug
  - Contiki Processes
  - Profiling Functions
- Conclusions and Future Research

Background: Interfacing to the OCDM over JTAG

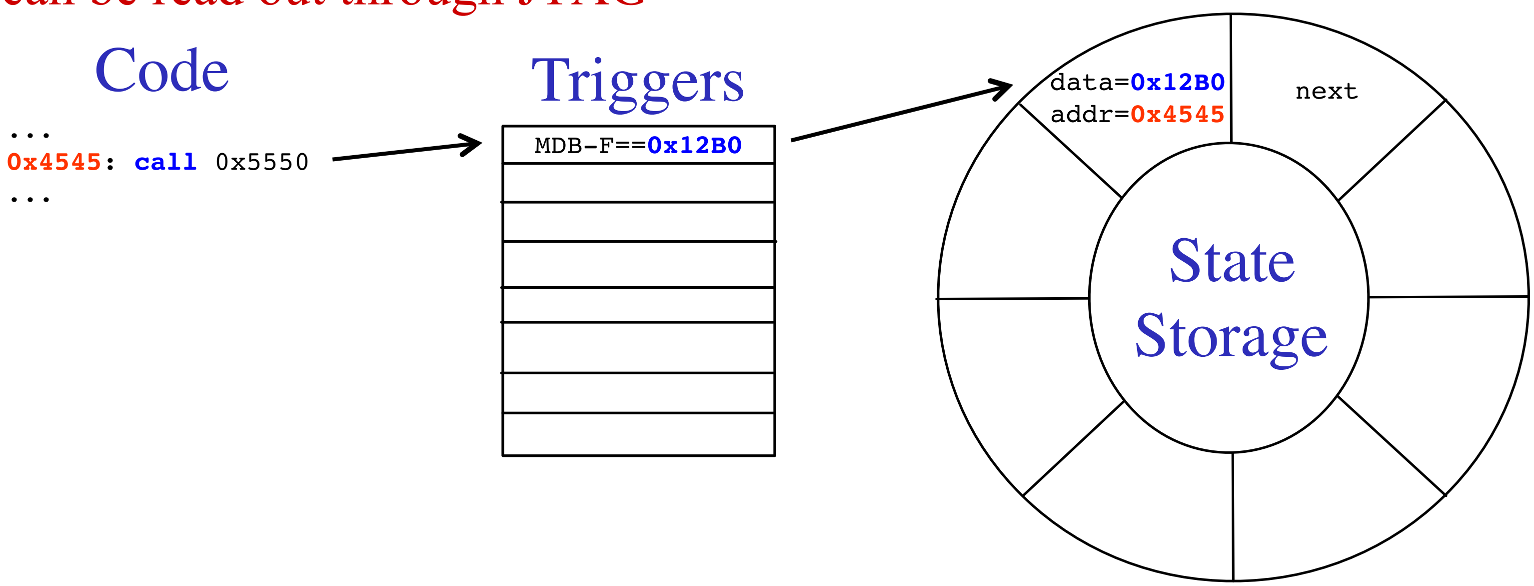
- JTAG interface uses 4 pins
  - TDO data output
  - TDI data input
  - TMS select mode
  - TCK clock



- MSP430 OCDM responds to various commands sent over JTAG
- Used 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

WP/BP Mode: MSP430 OCDM Triggers

- Total of 8 triggers can be set on OCDM through JTAG
- Each trigger specifies a condition on
  - Value present on data or address bus (MDB/MAB)
  - Operation type: read, write, or instruction fetch (-R/-W/-F)
- Can combine individual triggers to create complex triggers
- 8 entry state-storage circular buffer stores MDB and MAB when trigger fired; can be read out through JTAG



- **Key design challenge:** read buffer at a fast rate to prevent overwritten data

PURDUE UNIVERSITY

WP/BP Mode: Mapping Software Events to Triggers

- Triggers can be set for generic events e.g., function call and return

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

- A "`nop`" instruction can be used by the programmer to specify arbitrary trigger locations in code

provided to tracer

name value

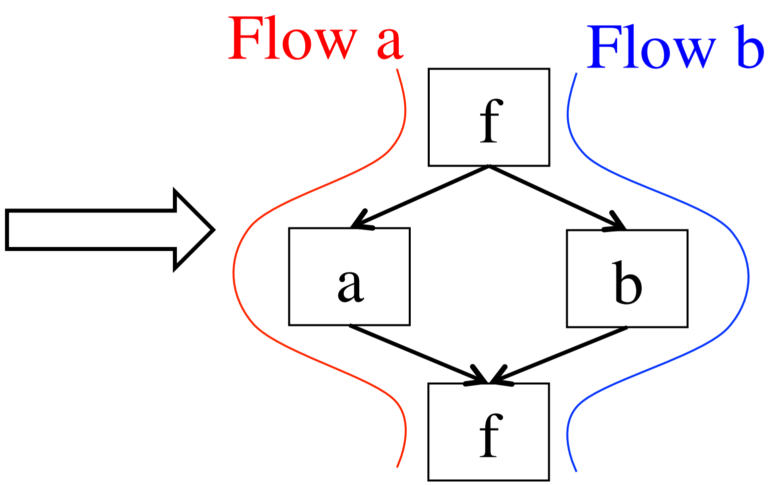
```
if (ready()) {
   NOP;   // state_m ON
   state_m = ON;
} else {
   NOP;   // state_m OFF
   state_m = OFF;
}
```


PURDUE UNIVERSITY

PC Polling Mode

- OCDM allows continuous polling of program counter
- Provides information about program control flow
- PC value can be mapped to a code block or function
  - Need to know the start address of each code block or function
- Basic use of PC polling

```
void f() {
    if (x) {
        a();
    } else {
        b();
    }
    ...
}
```



PC Polling: Extracting Function Start Addresses

- At boot-up, read program binary from mote through JTAG and perform disassembly
  - Discover location of function and interrupt start addresses
  - Used to lookup what function a PC address belongs to
  - Has the advantage of requiring no setup in advance

```
ffe0 <InterruptVectors>:
403a
8536
8c24
8876
8460
40c4
4078
86aa
```

```
4078 <sig_TIMERA0_VECTOR>:
   4078: 0f 12       push r15
   407a: 0e 12       push r14
   407c: 0d 12       push r13
   407e: 0c 12       push r12
   4080: b0 12 94 40 call #0x4094
   4084: 3c 41       pop  r12
   4086: 3d 41       pop  r13
   4088: 3e 41       pop  r14
   408a: 3f 41       pop  r15
   408c: b1 c0 f0 00 bic  #240, 0(r1)
   4090: 00 00
   4092: 00 13       reti

4094 <Msp430TimerCapComP__0__Event__fired>:
   4094: 1f 42 62 01 mov  &0x0162, r15
   4098: 8f 10       swpb r15
   409a: 5f f3       and.b #1, r15
   409c: 02 24       jz   $+6
   409e: 1f 42 72 01 mov  &0x0172,r15
   40a2: 30 41       ret
```

Function start address table
0x4078
0x4094
...

PURDUE UNIVERSITY

Limitations of Watchpoint and PC Polling Modes

- Watchpoint mode
  - State-storage buffer is 8 entries
  - Each poll and read of state buffer takes 122 mote cycles
  - Therefore, 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 task and function call granularity
- Cannot do watchpoint polling and PC polling at the same time

Design Challenge: Speed of JTAG Polling

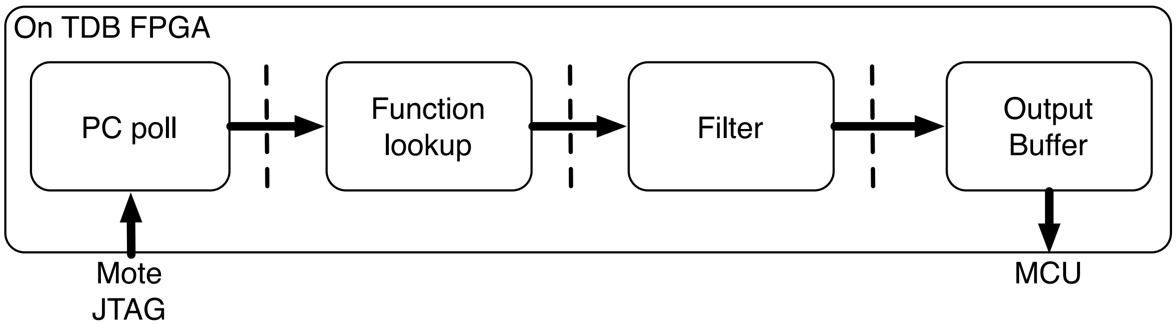
- If JTAG is controlled by software
  - MCU has to generate JTAG clock and process data
  - For example, using another MSP430 running at 8MHz would take the time shown

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*

- Datasheet specifies 10MHz JTAG clock maximum
  - We find we can reliably clock at 12MHz (WP mode) and up to 24MHz (PC poll)
- Using FPGA clocked at 48MHz we can achieve the maximum polling rate
  - FPGA generates 24Mhz JTAG clock

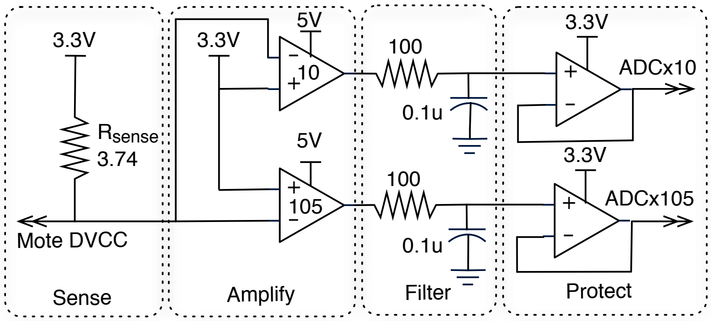
FPGA Pipeline for Function Profiling

- We use a pipelined architecture on the FPGA to improve the throughput of OCDM to TDB communication
- For example, PC polling pipeline stages are
  - Poll PC address
  - Binary search for function pointer
  - Filter block: does PC value indicate entry into a new function
  - Output buffer interfaces to debug board MCU



- The pipelined architecture enables us to keep up with the rate of the OCDM event stream

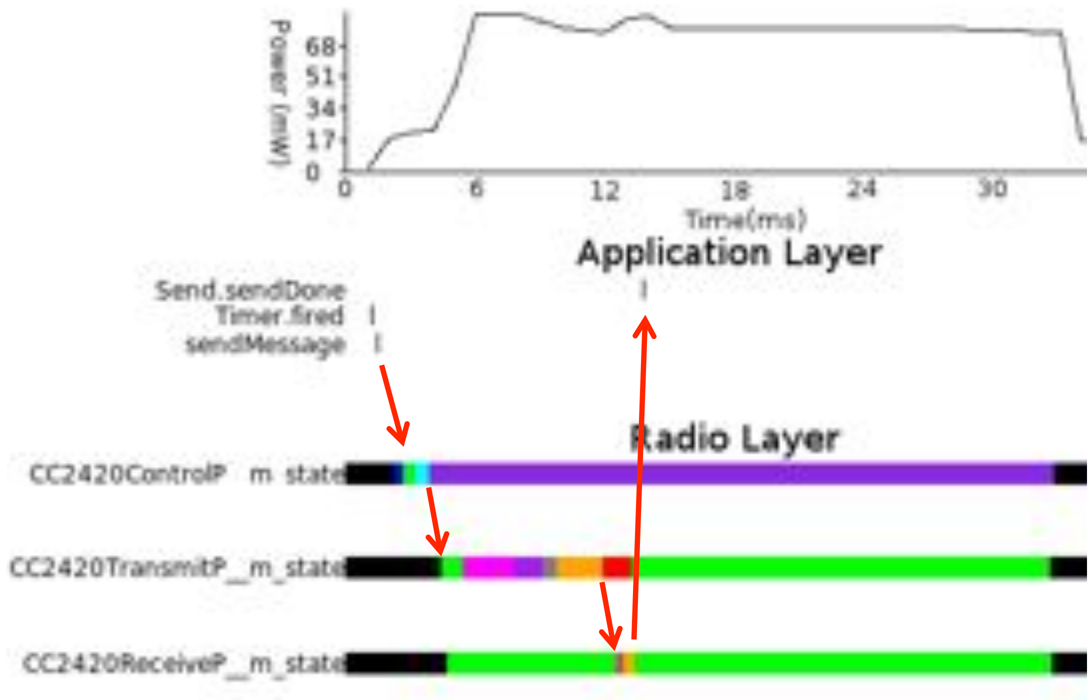
Other Features of the Board



- Energy monitoring
  - Range of 18uA to 30mA
  - Two amplifiers x10 and x105
  - Inexpensive 12bit ADC (on MSP430)
  - Samples at 20kHz
- Streaming data over USB
  - Can also be used as a bench-top debugger
- Can be powered through USB or battery
  - Provides power to the mote
- Low power, board enters sleep when mote is in sleep

Case Study 1: Using Watchpoints to Trace TinyOS States

- Inserted "nop" instruction for
  - Each state transition (indicated by an assignment to variables *m_state)
  - Each task handler

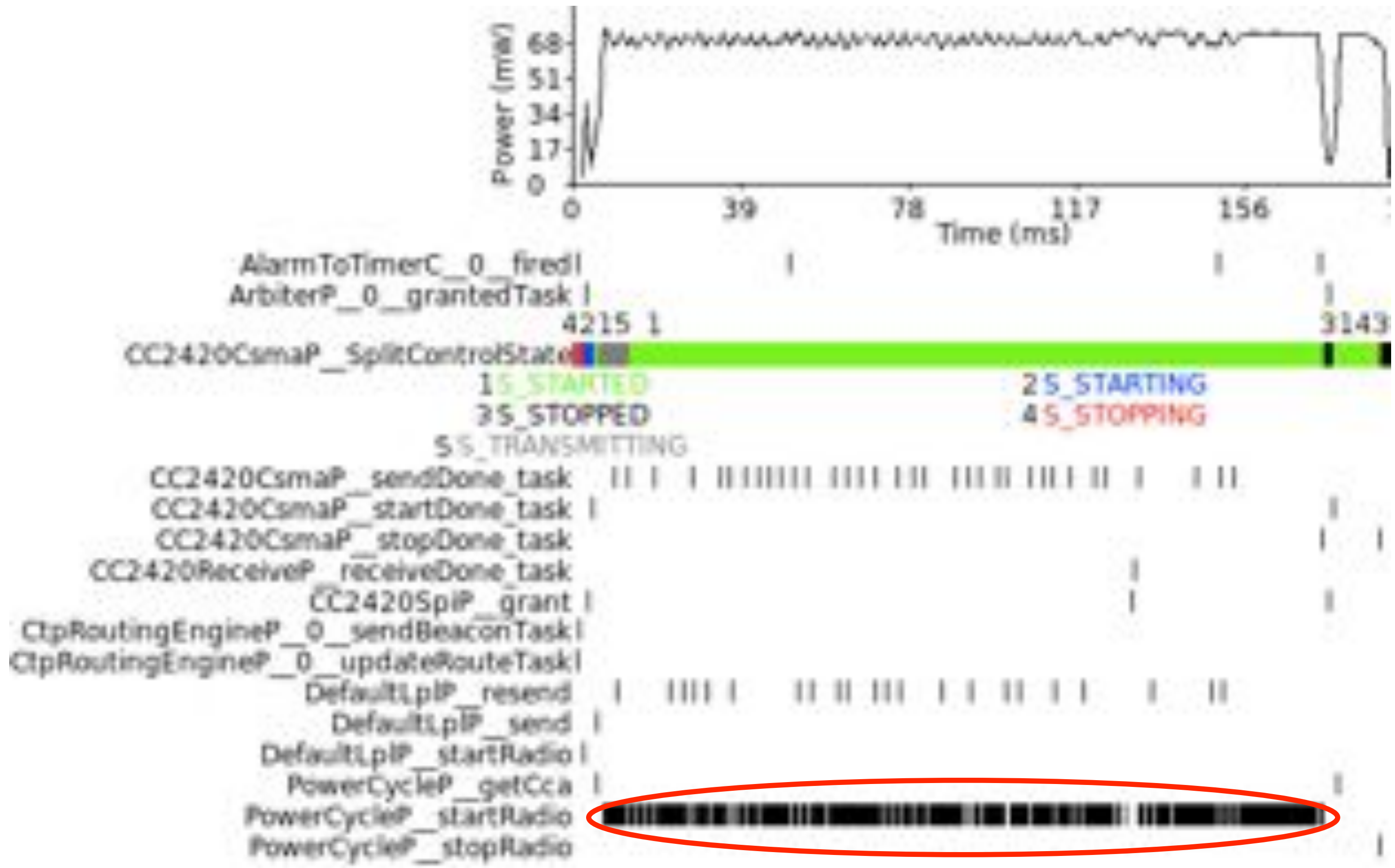


- Gives visibility of fine grained events

PURDUE UNIVERSITY

PowerCycleP Re-post Bug

- Discovered a bug in TinyOS when tracing tasks
- `PowerCycleP__startRadio` re-posts itself



PURDUE
UNIVERSITY

PowerCycleP Bug Explained

- `startRadio` task re-posts if `SubControl__start() != SUCCESS`
- When radio is already started, `SubControl__start() = EALREADY`

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

- Re-post could be permanent with following hypothetical code

```
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);
}
```

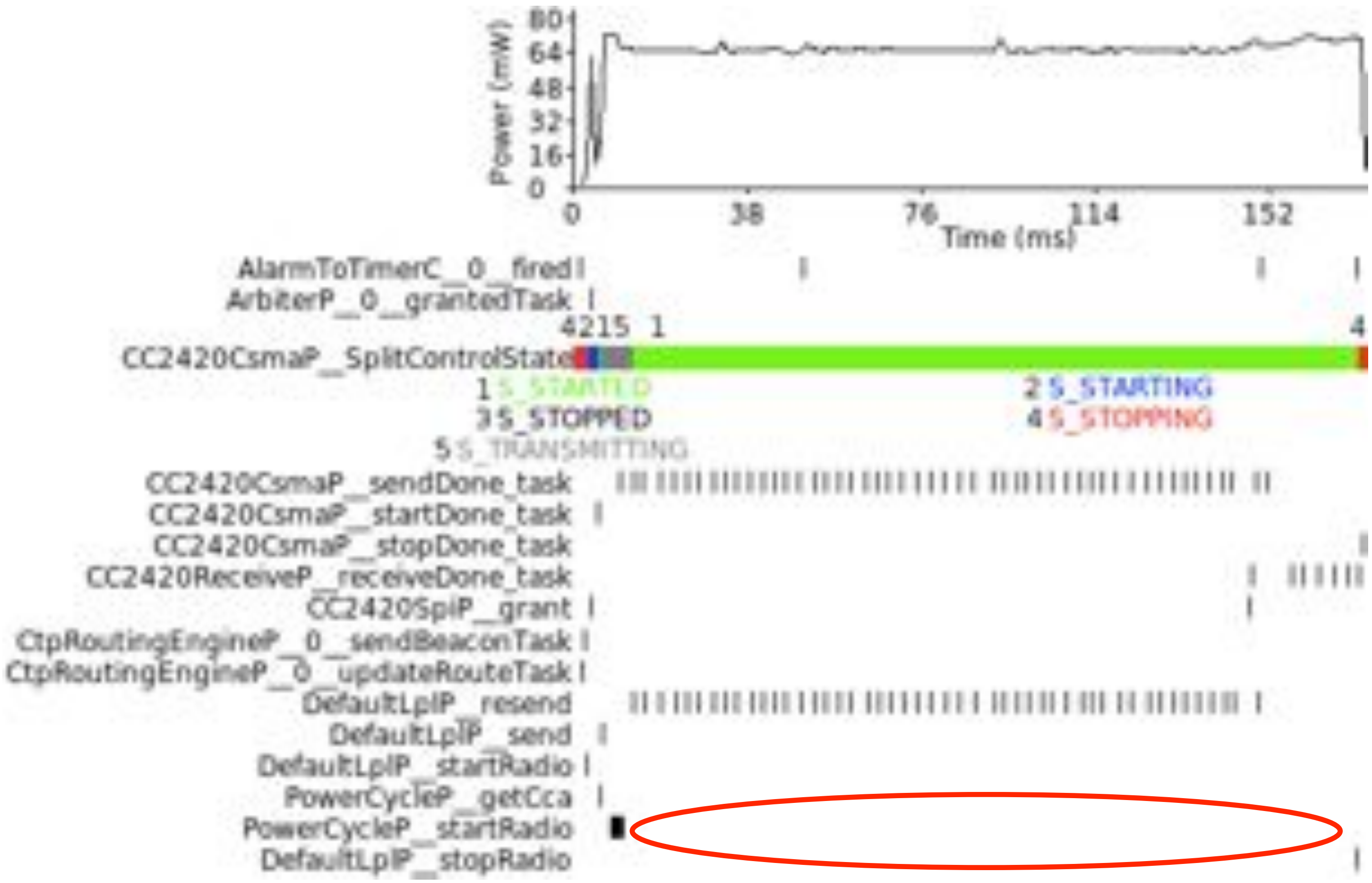
- Simple fix

```
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)

PURDUE
UNIVERSITY

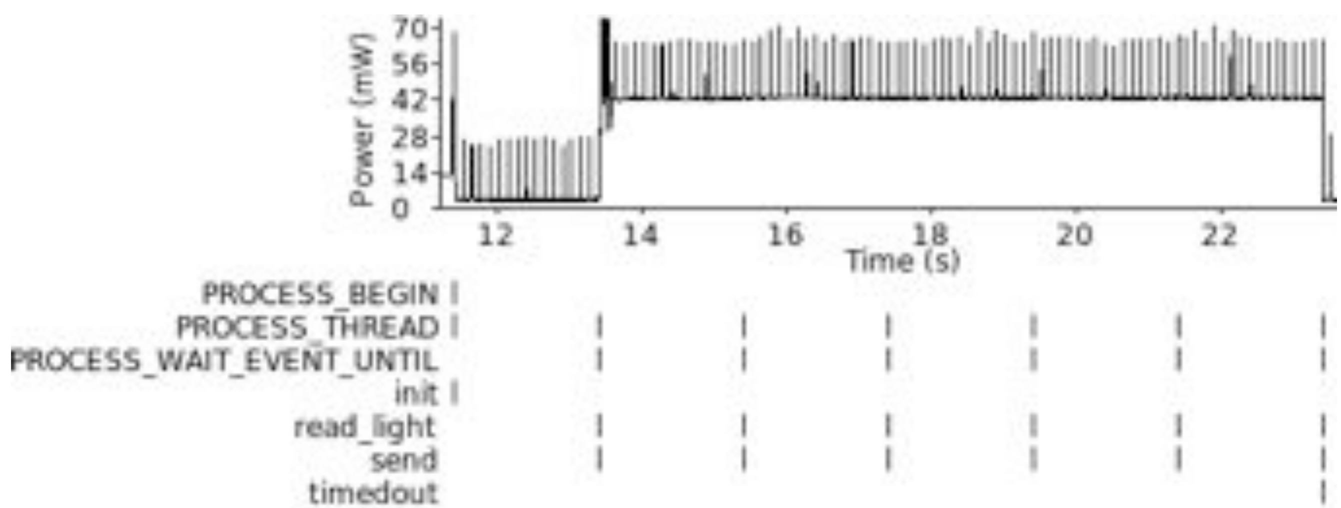
With Bug Fixed



PURDUE UNIVERSITY

Case Study 2: Monitoring Contiki Processes

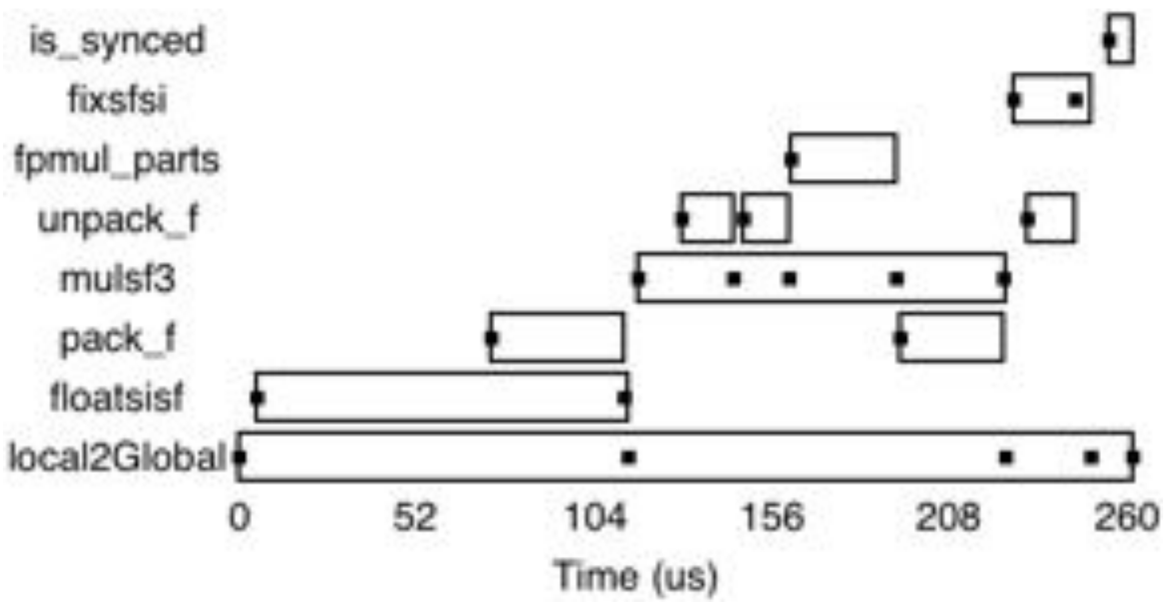
- Light tracking application [IPSN'11]
- Set "nop" instructions for Contiki processes
- Verifies OS agnostic nature of our architecture
  - Required no change of TDB switching from TinyOS to Contiki application



PURDUE UNIVERSITY

Case Study 3: Profiling with PC Polling

- PC polling down to granularity of 7 mote cycles
- Usually enough to catch every function transition
- Can be combined with call graph information to generate a profile



PURDUE
UNIVERSITY

Conclusions

- We proposed a hardware software approach for tracing and profiling of sensor network software
- Designed, implemented, and tested the Telos Debug Board
  - Non-intrusive (does not change timing)
  - OS/compiler agnostic
  - No significant hardware modification to mote
  - Easy to deploy (does not need *a priori* knowledge of application)
- Future design improvements
  - Reduce power in sleep mode (fast wakeup)
  - Improve energy monitoring accuracy
  - Add additional flash storage
- Future research directions
  - Debugging a deployed network
  - Applications such as record-and-replay

Thank You

Matthew Tancreti (mtancret@purdue.edu)