

Improving Energy-Efficiency in Sensor Networks by Raising Communication Throughput Using Software Thread Integration

Ramnath Venugopalan
and Alexander Dean



Center for Embedded Systems Research
Department of Electrical and Computer Engineering
NC State University

Motivation

- Sensor networks
 - Small size and low cost are very important
 - Small size => small batteries => less energy available
 - Un-tethered and unattended => cannot replace batteries for all
 - Energy conservation is of utmost importance
 - Largest consumers of energy - CPU and RF Module (RFM)
- Current Research
 - Hardware Optimization - low-power CMOS circuits for CPU and RFM
 - Software Optimization
 - Energy aware applications, communication (MAC, routing, etc.)
 - **Compiler optimizations to reduce cycle count**
 - Loop unrolling, elimination of dead code, register allocation, etc.
 - *Fast code == energy-efficient code*

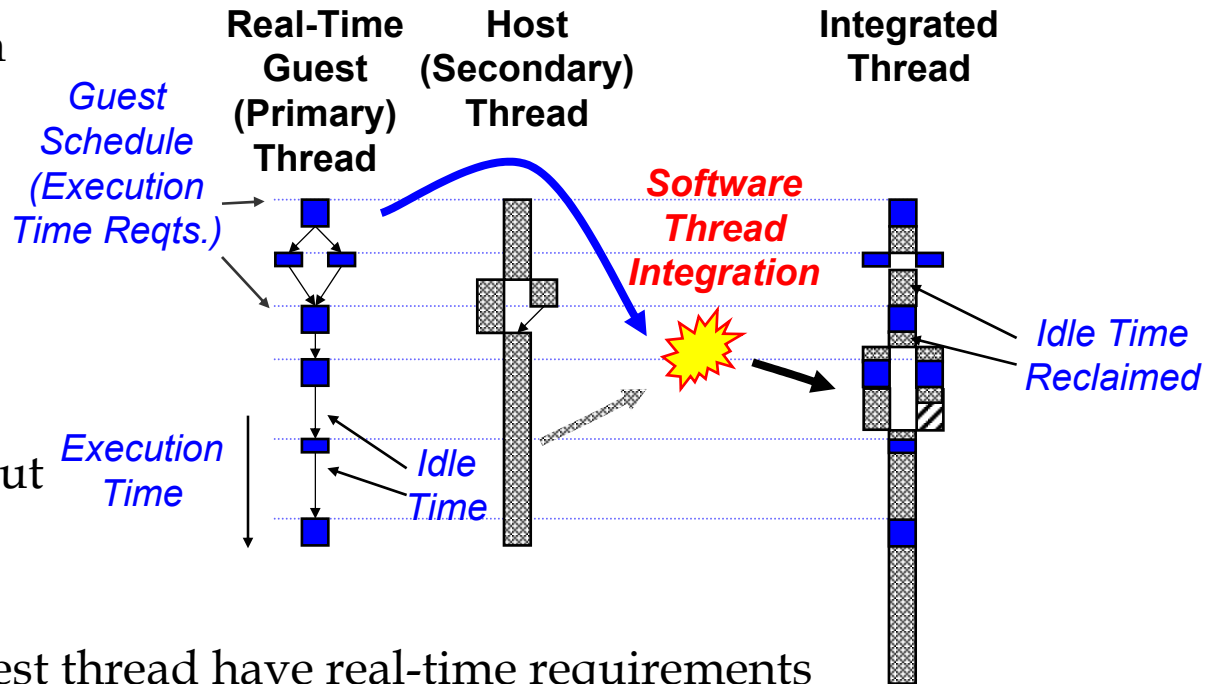
Problem - Interrupt Overhead

- Interrupts
 - Byte-level protocol controllers: 1 ISR/byte
 - Interrupt overhead for communication increases energy consumption:
 - CPU stays in active mode longer
 - There will be a reduction in maximum bit-rate => RFM will be ON for a longer time
 - May need to raise CPU clock frequency to meet required bit-rate
- Solution
 - A packet consists of bytes transmitted back-to-back
 - After start symbol, we know *when* each byte will be received
 - Use Software Thread Integration (STI) to get rid of ISR overheads and lower energy use

Software Thread Integration

- Software Thread Integration

- Compiler method to interleave assembly language threads at a fine-grain level.
- Provides low-cost concurrency even without fast context switches.



- Thread-Level - Basic STI

- Some instructions in guest thread have real-time requirements
- Provides a set of transforms which interleave code freely based on those sub-thread timing requirements
 - Replicate code into conditionals, fuse loops, etc.

- Result is greater efficiency

- *Integrated threads* more efficient
 - Can squeeze much more performance from cheap MCU
- *Integration process* automated
 - Can write/schedule code much faster

Main Idea – Reduce Interrupt Overhead

Before Integration



After Integration

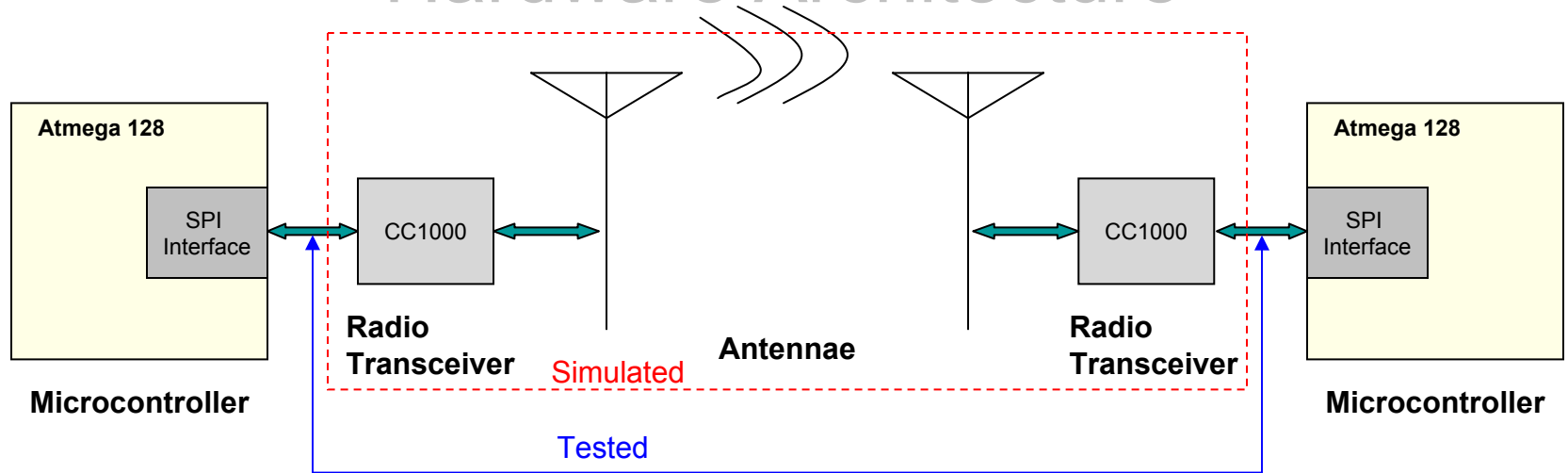


- Interrupt overhead can use up valuable CPU time
- Remove using STI, leveraging static scheduling of interrupt handlers
- Advantages
 - Communication rate can be increased, RFM can transmit quicker => active for less time
 - CPU does more work in fewer cycles => active for less time
 - CPU can operate at lower frequency since fewer execution cycles needed
- *Balance all 3 advantages to achieve optimal energy savings using STI*
- *Gather DSP work from application for later processing*
- *When we need to send or receive a packet, try to run an integrated function to do both communication and DSP efficiently*

Ensuring That Static Scheduling Works

- Timing is critical once packet begins (for both Tx and Rx)
 - All timing is referenced to the start of the packet
 - SPI Tx, Rx functions must write or read SPDR at correct times
- Accurate static schedule depends on
 - Predictable instruction execution speed (μ arch features can hurt)
 - Predictable software
 - Interrupts disabled
 - Conditionals padded to last same amount of time regardless of path
 - All relevant loops in this app. have known iteration counts
 - Matched clock rates between Tx and Rx nodes
 - Crystals typically provide +/- 50 ppm accuracy
 - Comm. system can handle up to $1/(2 * \text{num bits per packet})$ clock difference
 - So normal crystals are good for packets up to 5000 bits
 - Resonators are cheaper and less accurate...

Hardware Architecture

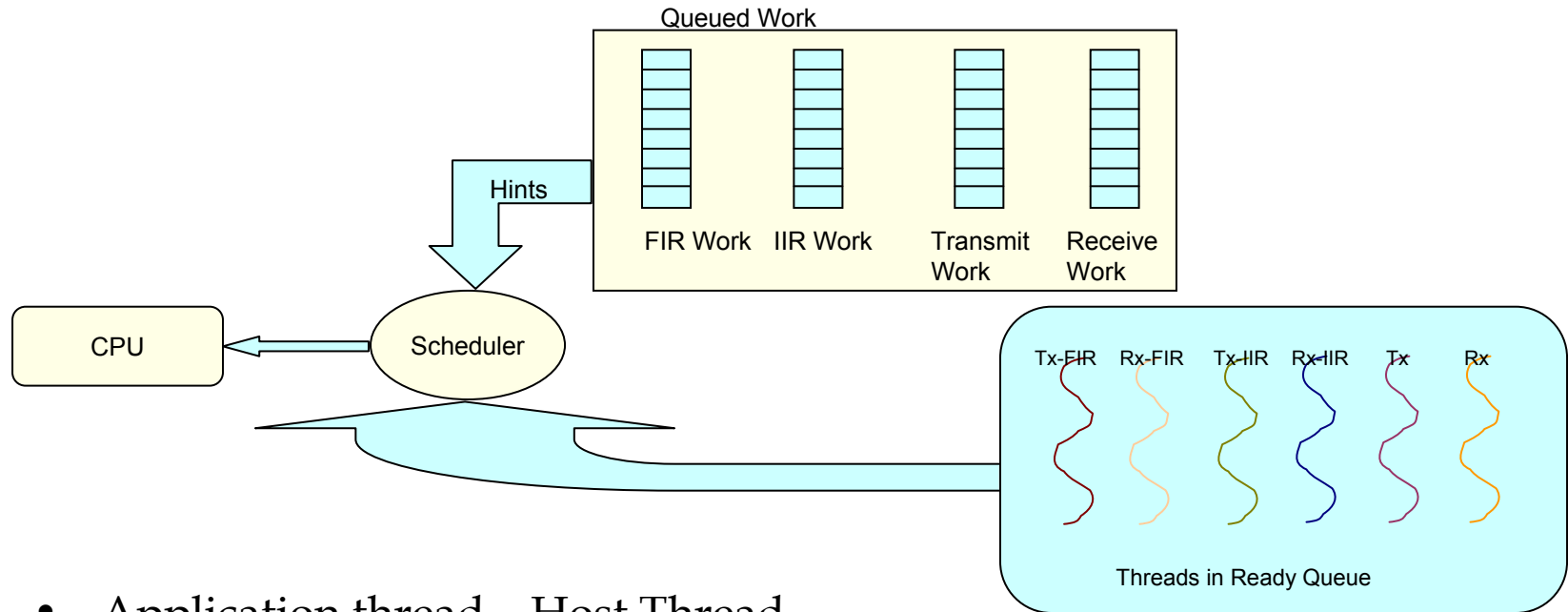


- Atmega128 microcontroller uses SPI to communicate with other nodes
 - Simulate CC1000 radio modules
 - Test with live wired SPI interface
- Serial Peripheral Interface – SPI
 - Can run at speeds of $f_{cpu}/2^1$ to $f_{cpu}/2^7$
- Atmega128 microcontroller
 - AVR Arch.: 8-bit, 32 registers, load-store, 2-stage pipeline, no cache
 - On-chip memory: 4K SRAM, 128K Flash. *No Cache*
 - Clock ≤ 20 MHz
 - Fully predictable hardware
- Atmega128 power-save modes used
 - Active: Executing instructions
 - 2.85 mW/MHz
 - Idle Mode: All peripherals running
 - 1.35 mW/MHz
 - Extended Standby Mode: Only oscillators running
 - 0.06 mW/MHz

CC1000 RF transceiver

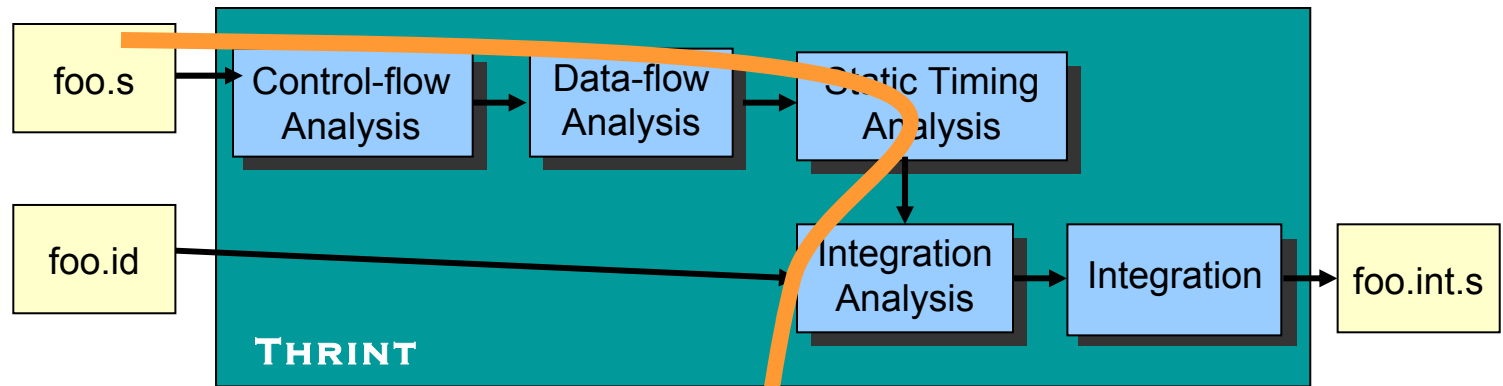
- Single chip low power UHF transceiver for ISM band (e.g. 868 MHz)
 - Supports up to 76.8 kbps (other devices are faster)
- Energy consumption depends on:
 - Output power while transmitting
 - Duration for which it is active
 - Does NOT depend on bit-rate of communication
- Two modes:
 - Active: Transmit 31.2 mW, Receive 28.8 mW
 - Power-down: 0.6 μ W
- Modeled CC1000 energy consumption from datasheet specs

Software Architecture

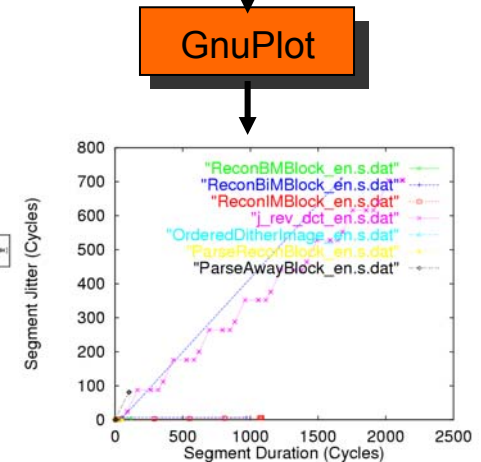
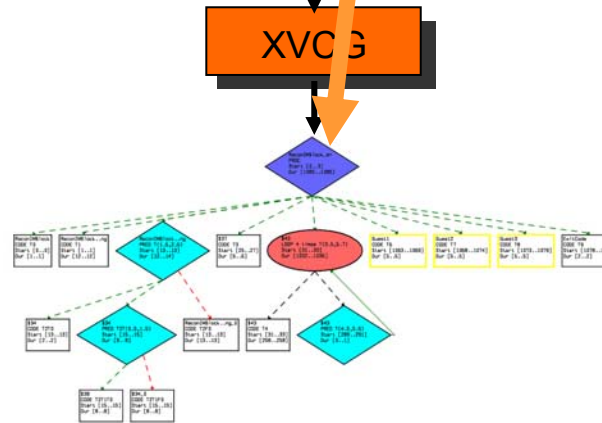


- Application thread - Host Thread
 - Uses fixed-point (16.16) 8th order FIR and 6th order IIR filter threads
- Communication thread - Real-Time Guest Thread
 - Transmit and receive threads use SPI for communication
- Six threads in system - 4 integrated threads (Tx-FIR, Rx-FIR, Tx-IIR, Rx-IIR), 2 discrete threads (Tx, Rx)
- Scheduling appropriate threads
 - Integrated thread runs when application queue has work
 - Otherwise use discrete padded communication threads

Software Thread Integration



- id file marks functions to integrate
- In past used Thrint to integrate Alpha code
- AVR support was under development at time of this project
 - So just used Thrint for control dependence visualization and static timing analysis

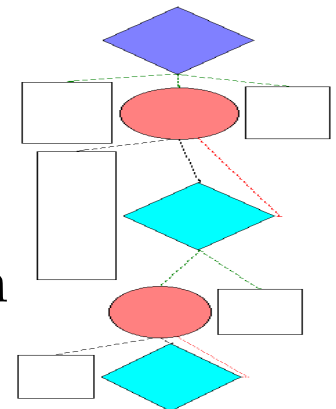
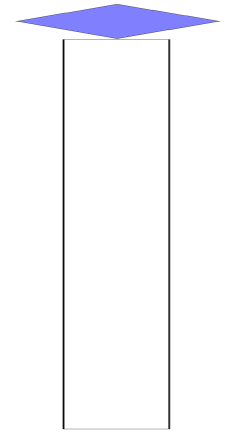
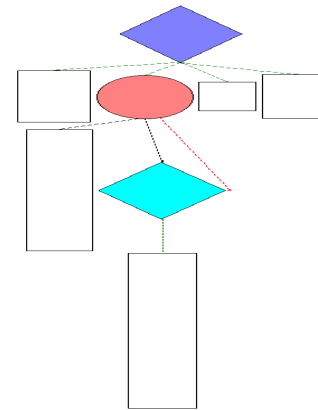


Experimental Method 1

- Software Development
 - Wrote/procured C source code
 - Compiled to asm with avr-gcc 3.2
 - Partitioned register file for functions to be integrated
 - Examined asm code for comm (Tx, Rx) and filtering (FIR, IIR)
 - Analyzed for timing and control structure using Thrint
 - Comm (Tx, Rx) and filtering (FIR, IIR) manually integrated to create four new threads
 - Integrated code compiled/linked with avr-gcc

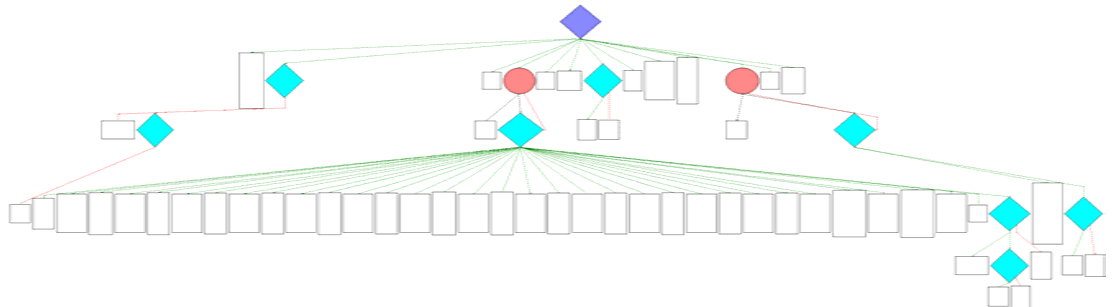
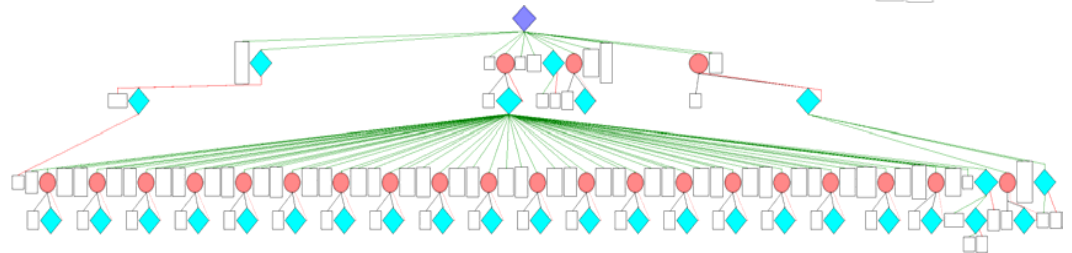
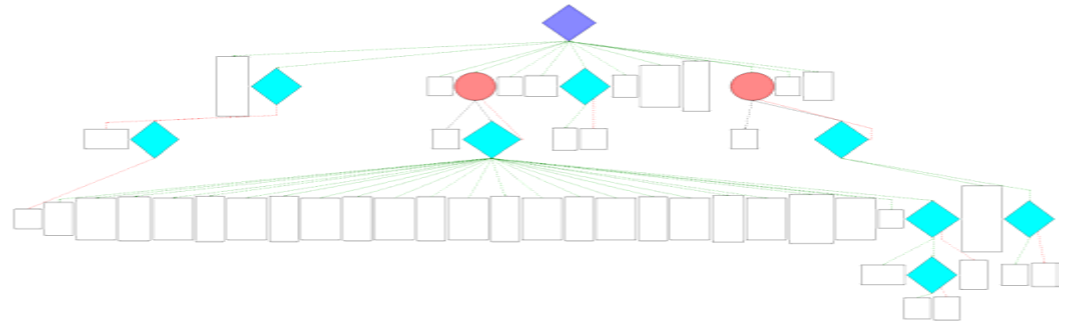
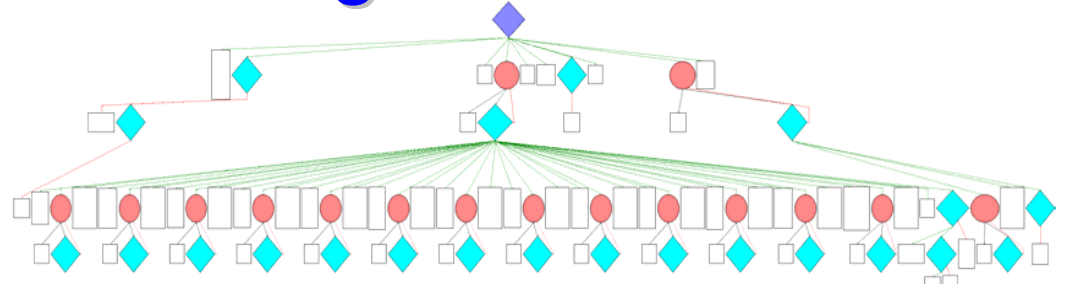
Details of Functions to Integrate

- FIR and IIR filters are loops with fixed iteration counts
 - Unrolled loops for performance and to simplify integration
 - Resulting code for each is a single basic block
- SPI Transmit function uses loop with one iteration per byte
- SPI Receive function uses loop (1 iter/byte) with nested polling loop to await new SPI data



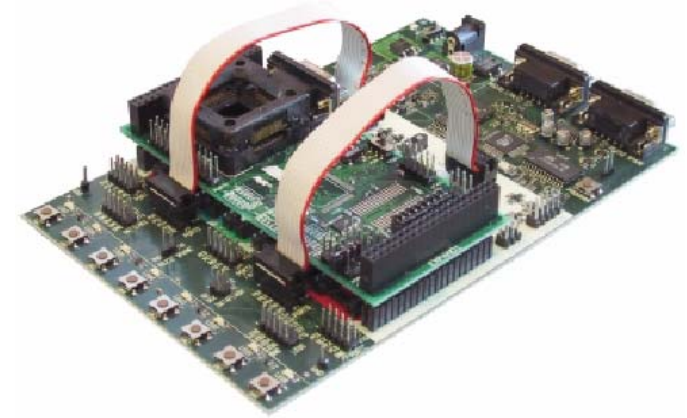
Results of Integration

- Rx+FIR
 - Code 42% larger than sum of original functions
- Tx+FIR
 - Code 42% larger
- Rx+IIR
 - Code 45% larger
- Tx+IIR
 - Code 45% larger



Experimental Method 2

- Functional Verification
 - Executable downloaded to AVR STK500 MCU eval. board
 - Code executed on Atmega128 MCU
 - Verified DSP filters with DAC and scope
 - Verified SPI with live hardware (connected two boards)
- Performance Evaluation
 - Cycle counts measured using on-chip debugger (JTAG-ICE)
 - Resulting energy found using cycle counts and energy models from data sheets ($V_{CC}=3.0\text{ V}$)
- Communication System
 - Assumed traffic of 200 packets/sec, 35 bytes/packet
 - Tested hardwired SPI, rather than RFM



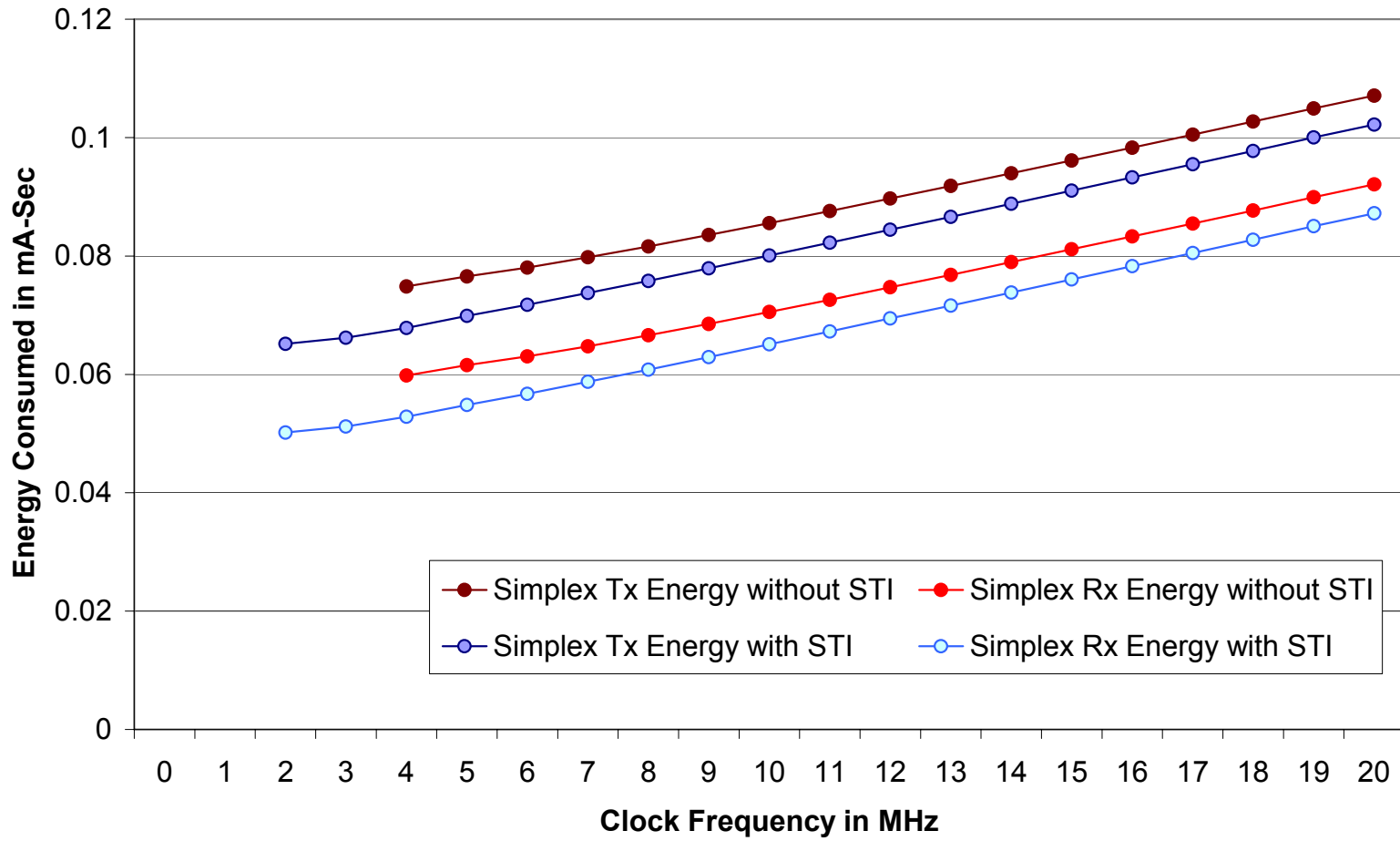
Atmega128 with the STK501/500

Experiment 1 - Behavior with increase in frequency

- Assumption - Fixed bit-rate and packet rate
- RFM on-time is constant
 - Hence, RFM sleep-time is also constant
 - So all gains in energy come from CPU savings
- At any time, the CPU can be:
 - Running communication code
 - Running application code (DSP)
 - In Idle mode, waiting for communication to complete
 - In Standby mode
- ISR-based system requires higher MCU frequency due to overhead and communication deadlines
- STI based system can run at lower frequencies - fewer cycles to execute
 - Energy saved from both factors

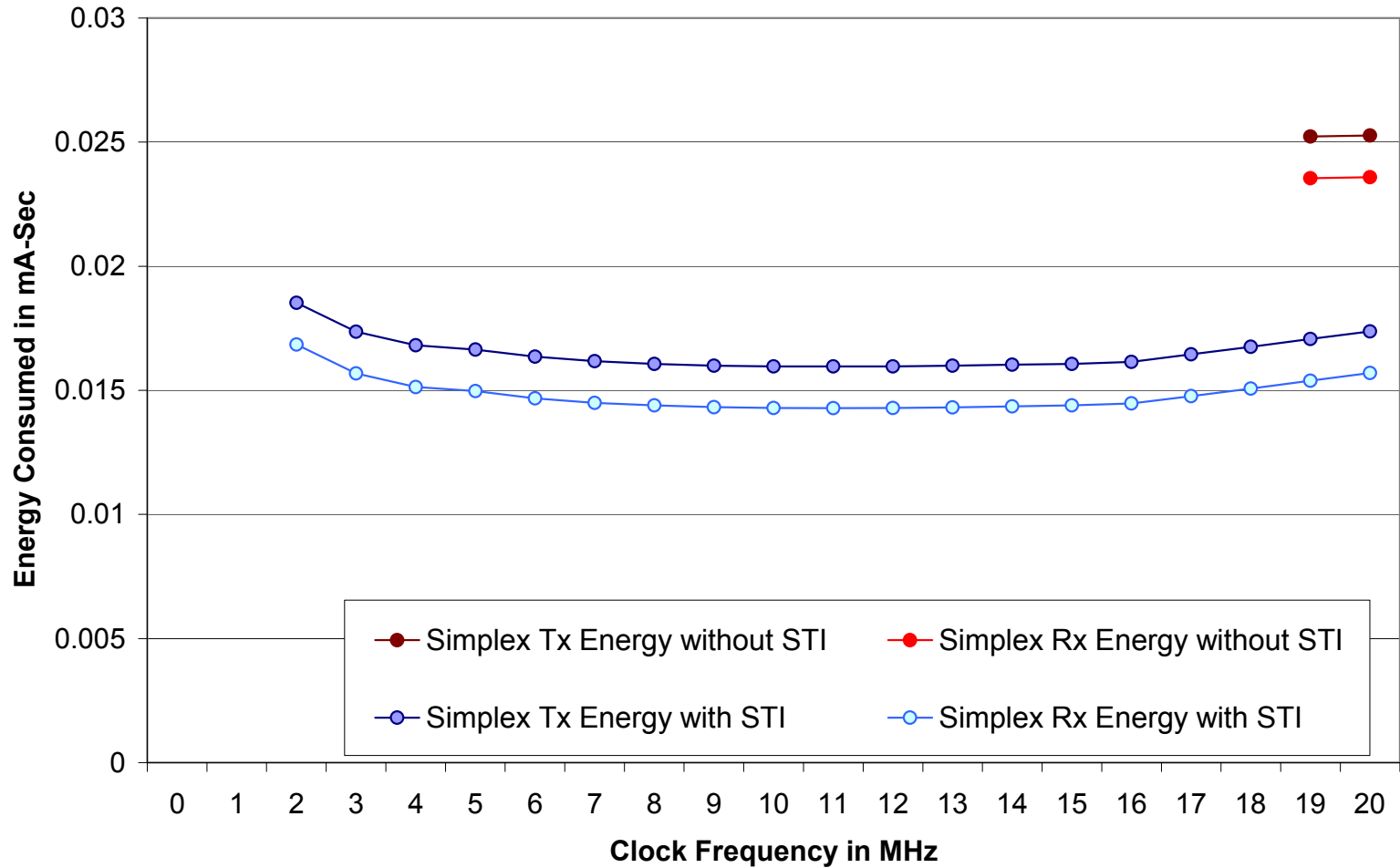
Results – Clock Frequency Vs. Energy Consumed

Clock Frequency Vs. Energy Consumed at 56 kbps



Results – Clock Frequency Vs. Energy Consumed

Clock Frequency Vs. Energy Consumed at 500 kbps



Experiment 1 - Analysis

- Initially CPU active time decreases as frequency rises
- CPU active time continues to decrease until equal to communication time
- After that, CPU needs to stay on until communication finishes.
 - If CPU has idle mode, can save energy with it
- After this point, active time + idle time = constant.

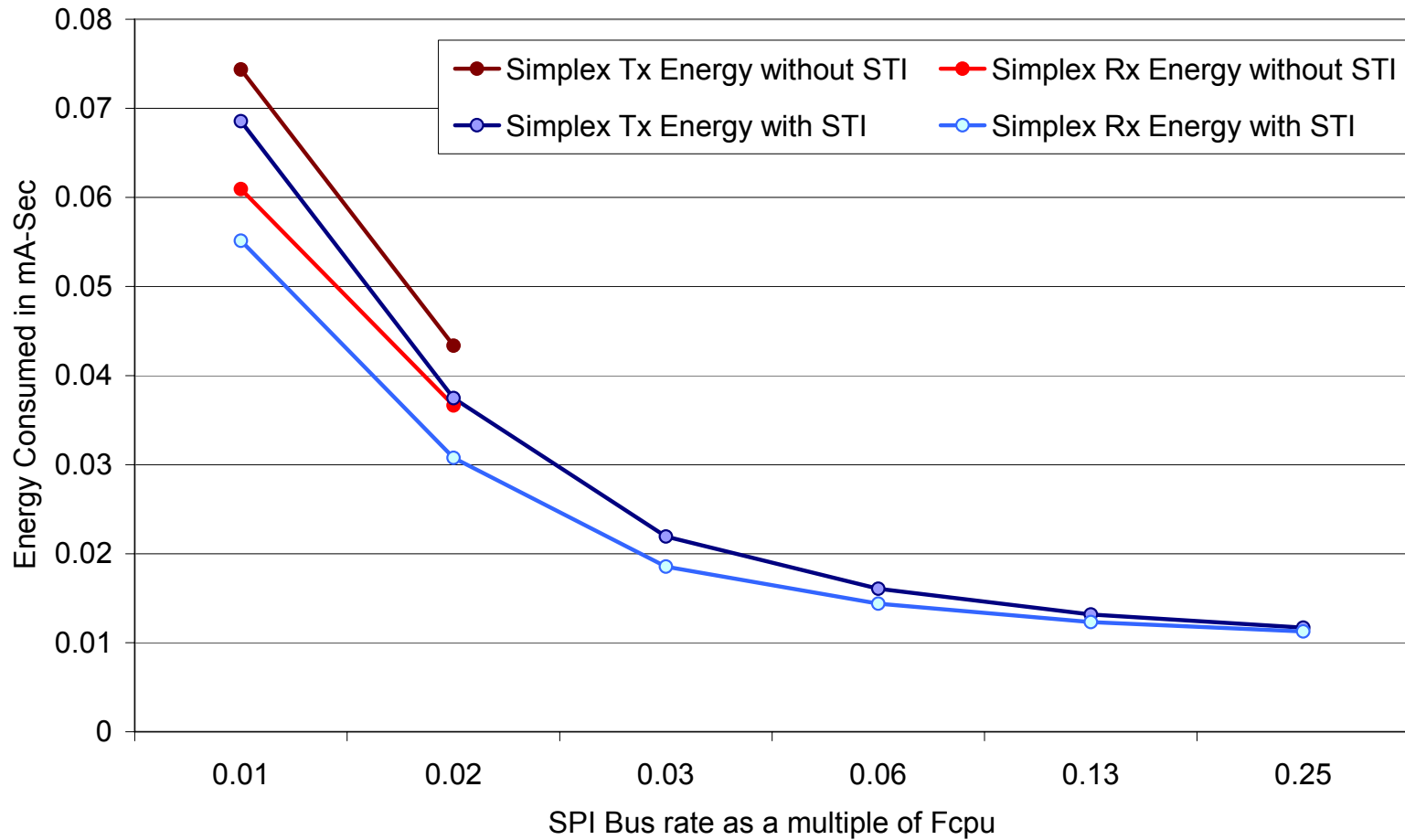
- Currents in idle and active modes increase with frequency
 - Overall energy consumption rises
- Difference in energy consumption for the STI and ISR lines stems from
 - Difference in cycles executed
 - How soon low-power mode can be activated

Experiment 2 - Behavior with rise in bit-rate

- Assumption - Packet rate and clock frequency are constant
- Amount of time taken by CPU to execute code is constant
- ISR-based systems are limited to much lower bit-rates than the STI based systems
- STI-based system can turn off RFM much sooner, saving energy

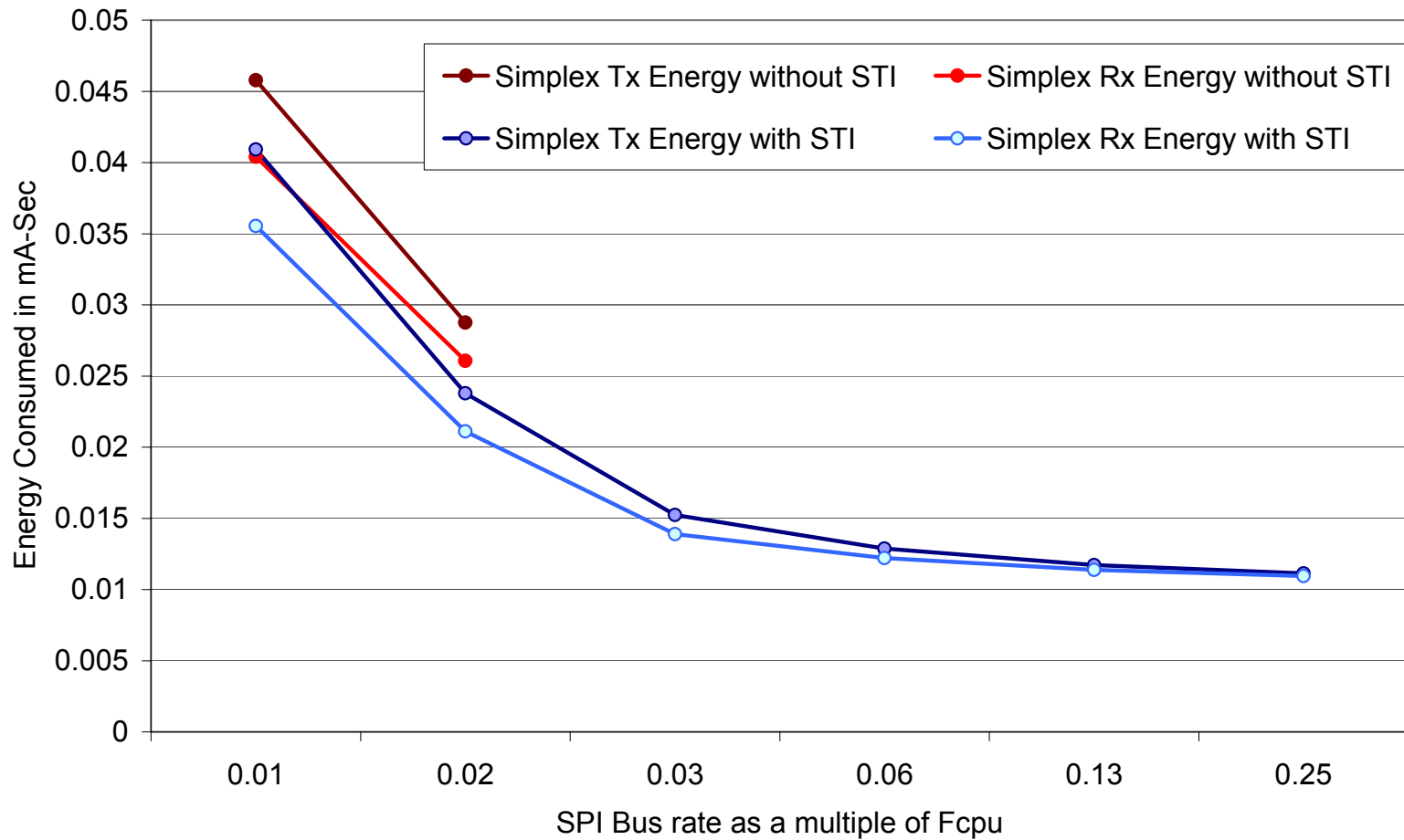
Results - SPI Data Rate Vs. Energy Consumed

SPI Data Rate Vs. Energy Consumed at Fcpu= 8Mhz



Results - SPI Data Rate Vs. Energy Consumed

SPI Data Rate Vs. Energy Consumed at Fcpu= 20Mhz



Experiment 2 - Analysis

- Initially, energy consumption falls sharply since RFM and CPU active time decreases as bit-rate rises.
- Also, CPU work finishes much earlier than RFM
 - CPU can go into idle mode if available
- Continues to fall sharply until communication time = computation time
- After this point, the CPU must remain active until computation ends
 - RFM goes to sleep earlier with increase in bit-rate.
- Difference in STI and ISR lines stems from
 - Lower number of cycles to execute for STI
 - Lower active time for STI CPU
 - Higher standby time for STI CPU

Conclusions

- STI allows for communication at high bit-rates so that RFM can turn off earlier
- STI allows nodes to run at lower frequencies
- STI eliminates extra CPU cycles of interrupt overhead
- Optimum operating frequency
 - Frequency vs. Energy graphs show energy minimum for given bit-rate (is where communication time = computation time).
 - Much lower for STI than ISR
- Faster bit-rates save energy
 - Knee of the curve in Bit-rate vs. Energy graphs indicates diminishing returns
 - Knee is much higher for STI, total energy is much lower
- Combination of optimum bit-rate and operating frequency for a node for least energy consumption
 - Least energy much lower for STI
- Increase in packet size => increase in ISR overhead, hence energy consumption with STI much lower

Future Work

- Sensor nodes run many other tasks
 - Routing, processing signals from sensors, etc.
 - Sampling signals from sensors
- STI can be applied to this whole combination of tasks to be run on a sensor node
- More accurate evaluation of energy consumption and longevity with STI possible in this case.
- Results of this work can help

References

- Compiling for low power and energy
 - Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Compilation techniques for low energy: An overview. In Proceedings of the 1994 IEEE Symposium on Low Power Electronics, October 1994.
 - Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2(4):437-445, 1994.
 - Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Lee. Instruction level power analysis and optimization of software. Journal of VLSI Signal Processing, pages 1-18, 1996.

References

- Software Thread Integration
 - P. Ganesan and A. Dean, "Efficiently Adding Secure Communications to Networked Low-End Embedded Systems using Software Thread Integration" RTAS 04. May 2004
 - B. Welch, S. Kanaujia, A. Seetharam, D. Thirumalai, A. G. Dean. "Extending STI for Demanding Hard-Real-Time Systems," CASES 2003, San Jose, CA.
 - A. Dean. "Compiling for Concurrency: Planning and Performing Software Thread Integration," 23rd IEEE Real-Time Systems Symposium, December 3-5, 2002, Austin, TX
 - R. Venugopalan, "Improving Energy-efficiency in Sensor Networks through Raising Communication Throughput using STI". Masters thesis, North Carolina State University, July 2003.
 - A. Dean, J. P. Shen, "Hardware to Software Migration with Real-Time Thread Integration". EuroMicro Workshop on Digital System Design, Vasteras, Sweden, August 25-27, 1998

References

- Sensor Networks

- J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Networked Sensors". ASPLOS 2000.
- J. Hill, "A Software Architecture Supporting Networked Sensors". Masters thesis, University of California at Berkeley, December 2000.
- K. S. J. Pister, J. M. Kahn and B. E. Boser, "Smart Dust: Wireless Networks of Millimeter-Scale sensor Nodes", Highlight Article in 1999 Electronics Research Laboratory Research Summary
- Mohamed Younis, Moustafa Youssef, Khaled Arisha, "Energy-Aware Routing in Cluster-Based Sensor Networks", IEEE/ACM MASCOTS 2002.

References

- Data Sheets and Application Notes
 - Atmel, Application Note AVR223.
 - Atmel, Atmega128 Datasheet, 2003.
 - Chipcon, CC1000 Datasheet, 2002.