System Architecture Directions for Networked Sensors

Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister

University of California, Berkeley

Presented by : Aseem Lalani

Outline

- Introduction
- Paper Contribution
- Network Sensor Characteristics
- Example Design Point
- TinyOS Design Issues
- Illustration
- Evaluation
- Architectural Implications & Conclusions

Introduction

- Trends that need to work together to enable the networked sensor
  - Smaller, Cheaper, Low Power Unit
  - Complete systems on a chip
  - Integrated low-power communication
  - Integrated low-power transducers
- Building Block now includes not just memory and processing but also DAC's, ADC's, UART's, interrupt controller and counters.
- Communication can take the form of wired, short-range RF, infrared, optical etc.
- Networked sensors can be constructed of just square inch size and using fraction of a watt in power.

Paper Contribution

- Initial Exploration of system architecture for networked sensors.
- Development of a tiny microthread operating system called "TinyOS" which addresses the two core issues viz.
  - Concurrency Intensive: Several Flows of data must be kept moving simultaneously
  - Efficient Modularity: Hardware specific and application specific components must snap together with little processing and storage overhead.

Networked Sensor Characteristics

- Small physical size and low power consumption
- Concurrency-intensive operation
- Limited Physical Parallelism and Control Hierarchy
- Diversity in Design and Usage
- Robust Operation

Small physical size and low power consumption & Concurrency-intensive operation

- Size and Power constrain the processing, storage and interconnect capability of the basic device. Reducing size and power thus forms the driving force in hardware design
- Software must make efficient use of processor and memory while enabling low power communication.

- Primary mode of operation for these devices is to flow information from place to place with a modest amount of processing on-the-fly.
- Does not involve accepting commands, stop, think, respond.
- Information is simultaneously captured from sensors, manipulated and streamed onto a network.

Limited Physical Parallelism and Control Hierarchy & Diversity in Design and Usage

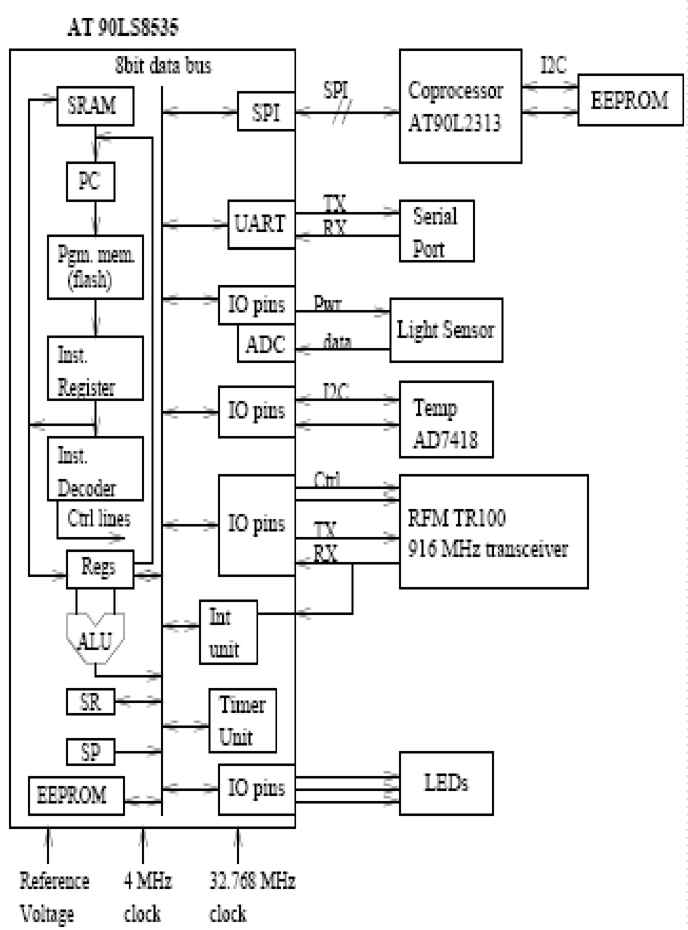
- Number of independent controllers, capabilities of those controllers and the sophistication of the processor-memory-switch level interconnect are much lower than conventional systems.
- Typically, sensor provides primitive interface directly to a single-chip microcontroller.

- Sensor devices are application specific.
- Need to assemble only the software components required to synthesize the application from the hardware components.
- Need for unusual degree of software modularity.

Robust Operation

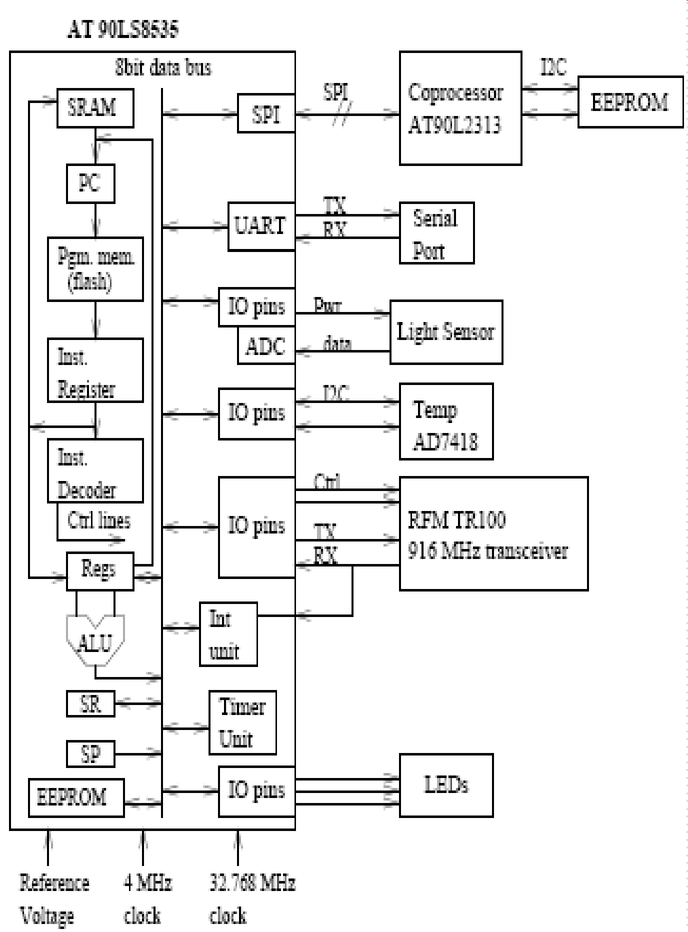
- Sensor networks are generally formed by deploying large numbers of unattended sensors forming an application specific network.
- Reliability is achieved by using redundancy in sensors deployed.
- Such redundancy cannot be used within individual sensors due to lack of space and power.
- However a failure in this case may cost huge communication cost.
- Requirement of the application to tolerate individual device failures (robustness).
- Operating systems running on single node should facilitate the development of reliable distributed applications.

Example Design Point



- 8-bit ATMEL 90LS8535 processor with 16-bit addresses, 32 8-bit general purpose registers and runs at 4 MHz and 3.0 V working in idle, power down and power save modes.
- 8 KB of flash as program memory
- 512 bytes of SRAM as data memory

Example Design Point

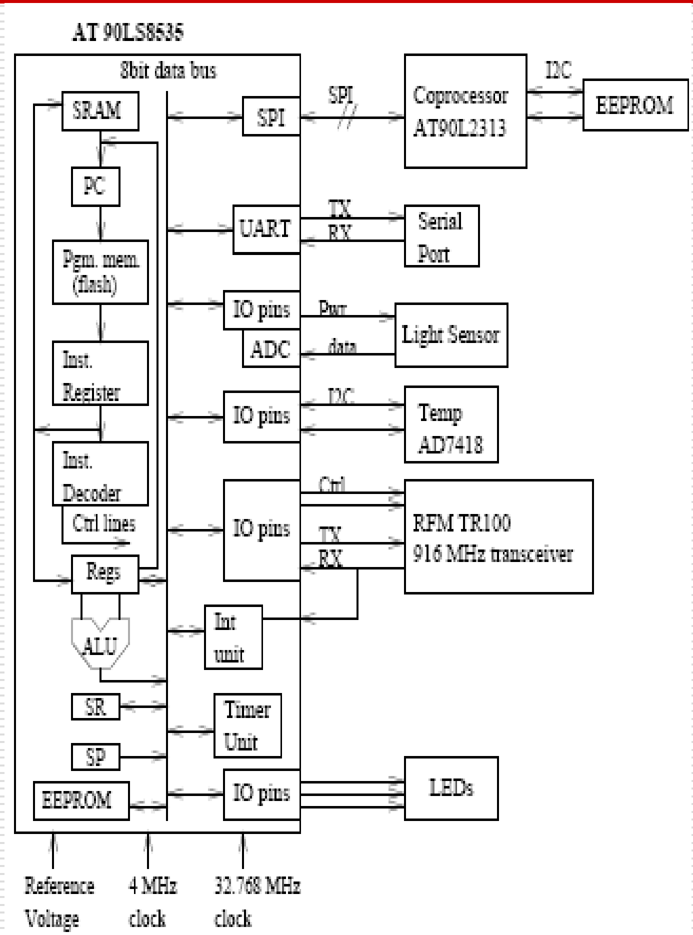


- 3 LED outputs connected through general I/O ports used to display digital values or status.

- Photo sensor representing the analog input.

- Radio representing an asynchronous I/O device with hard real time constraints working in transmit, receive or power-off modes with no buffering and no latching.

Example Design Point



- Temperature sensor (analog) having internal A/D converter unlike the Light Sensor.
- Serial to Parallel conversion using UART (Universal Asynchronous Transmitter Receiver)
- Co-processor AT90L2313 connected using SPI (Serial Peripheral Interface)

Tiny MicroThreading Operating System (TinyOS)

- Driving force behind TinyOS
  - Hardware Constraints
    - Small Physical Size
    - Modest Active Power Load
    - Tiny inactive load
  - Software Constraints
    - Manage hardware capabilities effectively
    - Support concurrency-intensive operation
    - Provide efficient modularity and robustness
  - Inability to use existing embedded device operating systems for above constraints.

TinyOS Design

- System Configuration consists of
  - Tiny Scheduler
  - Graph of components
    - Set of Command Handlers
    - Set of Event Handlers
    - Encapsulated fixed-size frame
    - Bundle of simple tasks

- Some Definitions
  - Commands: Non-Blocking Requests made to lower level components
    - Commands deposit request parameters onto its frame and conditionally post a task for later execution
    - It may invoke lower commands
    - Must provide feedback to caller indicating a success or not e.g.: buffer overrun.

TinyOS Design

- Events Handlers: Used to deal with hardware events either directly or indirectly.
  - Lowest level components have handlers connected directly to hardware interrupts e.g.: timer events, counter events.
  - They deposit information into its frame, posts tasks, signal higher level events or call lower level commands.
  - Hardware event may flow upwards by calling higher level events or downwards by calling lower level commands.
  - Cycles are avoided in command/event chain by enforcing that commands cannot signal events (but not vice-versa).
  - Both commands and events perform small fixed amount of work within the context of their component's state.

- Tasks: Primary work is done by tasks
  - Atomic w.r.t. other tasks and run to completion although they can be pre-empted.
  - Tasks can call lower level commands, signal higher level events and schedule other tasks within a component.
  - Simulate concurrency within each component since they execute asynchronously w.r.t. events.

TinyOS Design

- Task Scheduler: FIFO scheduler utilizing a bounded size scheduling data structure.
  - More sophisticated structures like priority-based or deadline-based could be used.
  - Processor is put to sleep when task queue is empty while peripherals remain operating (for efficient battery usage).

Example Component

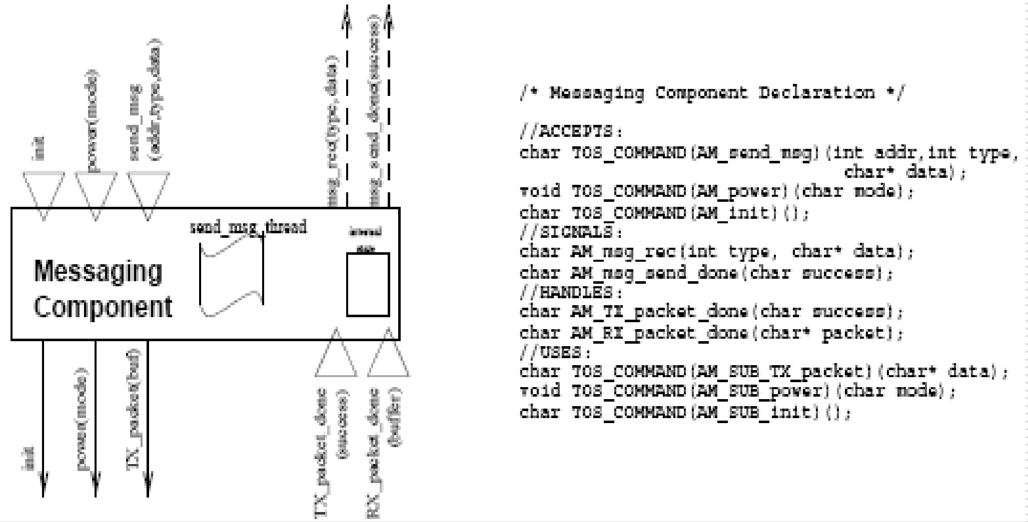
- Block of State (component frame)

- Set of commands (upside-down triangles)

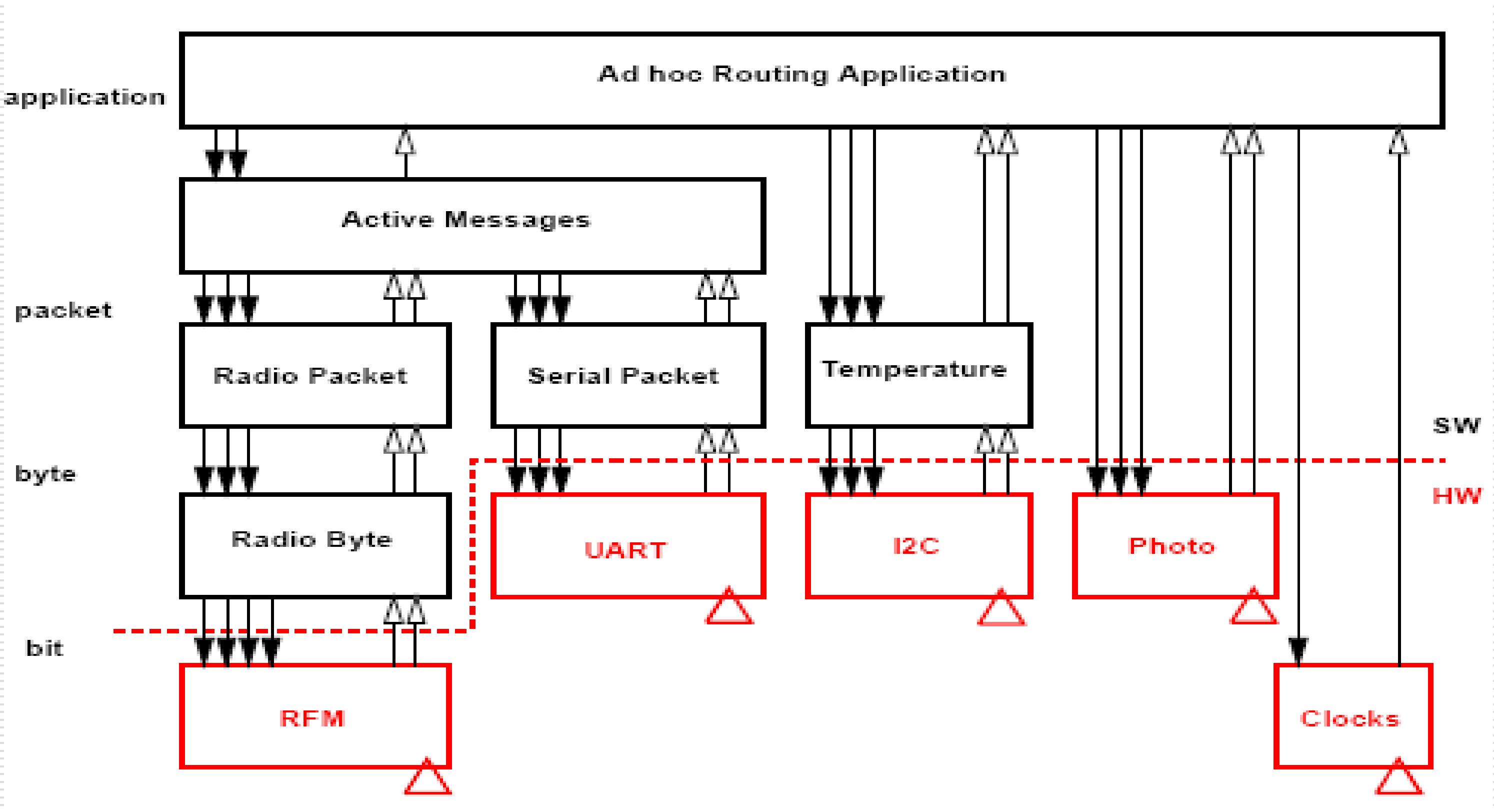
- Set of Handlers (triangles)

- Commands used by Handlers (Solid downward arcs)

- Events signaled by Handlers (dashed upward arcs)



Component Types



Sample Configuration of a networked sensor

Component Types

- Hardware Abstractions
  - Map physical hardware to the component model.
  - e.g.: RFM radio component
    - Exports commands to manipulate the individual pins connected to RFM transceiver and posts events informing other components about the transmission and reception of bits.
    - RFM consumes hardware interrupts to transform into either RX_bit_evt or TX_bit_evt.
- Synthetic Hardware
  - Simulate behavior or advanced hardware.
  - e.g.: Radio Byte component
    - It shifts data into or out of underlying RFM module and signals when an entire byte has completed.
    - May involve simple encoding and decoding of data.
    - At a higher level, it may be thought as UART hardware abstraction since it provides similar interface and functionalities.

Component Types

- High Level Software Components
  - Perform control, routing and all data transformation.
  - e.g.: Messaging module explained in earlier section
    - Performs filling in a packet buffer prior to transmission and dispatches received messages to their appropriate place.
    - Components that perform calculations on data or data aggregation also fall into this category.

Illustrate interaction of components

- Sensors are distributed in a localized area.
- They monitor temperature and light conditions and periodically transmit measurements to central base station (data source).
- Forwards data for sensors which are out of range of the base station (multi-hop).
- Basic Idea:
  - Broadcast application data in the form of fixed action messages.
  - If receiver is intermediate hop en-route to base station, the message handler initiates retransmission of message to the next recipient.
  - At the base station, the handler forwards the packet to the attached computer.
  - Base station periodically broadcasts route information.
  - At each device, the three significant events are: arrival of route update, arrival of message that needs to be forwarded and collection of new data.

Illustrate interaction of components

- Thousands of events are flowing through each sensor.
- Timer event defines the start of data collection.
- Application uses messaging layer’s send_message command to initiate transfer.
- Command records message location and schedules a task to handle the transmission.
- Task composes the packet and initiates a downward chain of commands by calling TX_packet command in the packet component
- In turn the command calls TX_byte within the Radio Byte component to start byte-by-byte transmission.
- Radio Byte Component prepares for transmission by putting RFM component into transmission state and schedules encode_task to prepare byte for transmission.

Illustrate interaction of components

- encode_task encodes the data and sends first bit of data to the RFM component for transmission.
- Radio Byte acts as data drain providing bits to RFM in response to TX_bit_evt event.
- On completion of byte transmission, TX_bit_evt signal is propagated to the packet level controller through TX_byte_done event.
- When all bytes of the packet have been drained, the packet level will signal TX_packet_done to the application level which will signal the msg_send_done event.
- Transmission process reversed for reception purposes.

Evaluation

- Small Physical Size:

- TinyOS Scheduler occupies only 178 bytes and the complete network sensor application requires only about 3 KB of Instruction memory

- Data Size of Scheduler is only 16 bytes which utilizes only 3% of available Data memory.

- Entire application comes in at 226 bytes which is still under 50% of the 512 bytes available.

Component Name	Code Size (bytes)	Data Size (bytes)
Multihop router	88	0
AM_dispatch	40	0
AM_temperature	78	32
AM_light	146	8
AM	356	40
Packet	334	40
RADIO_byte	810	8
RFM	310	1
Photo	84	1
Temperature	64	1
UART	196	1
UART_packet	314	40
I2C_bus	198	8
Processor_init	172	30
TinyOS scheduler	178	16
C runtime	82	0
Total	3450	226

Evaluation

- Concurrency Intensive Operations:

- Network sensors need to handle multiple simultaneous flows.

- Table shows context switch speed calibrated against intrinsic hardware cost for moving bytes in memory

Operations	Cost (cycles)	Time (μs)	Normalized to byte copy
Byte copy	8	2	1
Post an Event	10	2.5	1.25
Call a Command	10	2.5	1.25
Post a task to scheduler	46	11.5	6
Context switch overhead	51	12.75	6
Interrupt (hardware cost)	9	2.25	1
Interrupt (software cost)	71	17.75	9

- Posting a task and switching context costs about as much as moving 6 bytes of memory

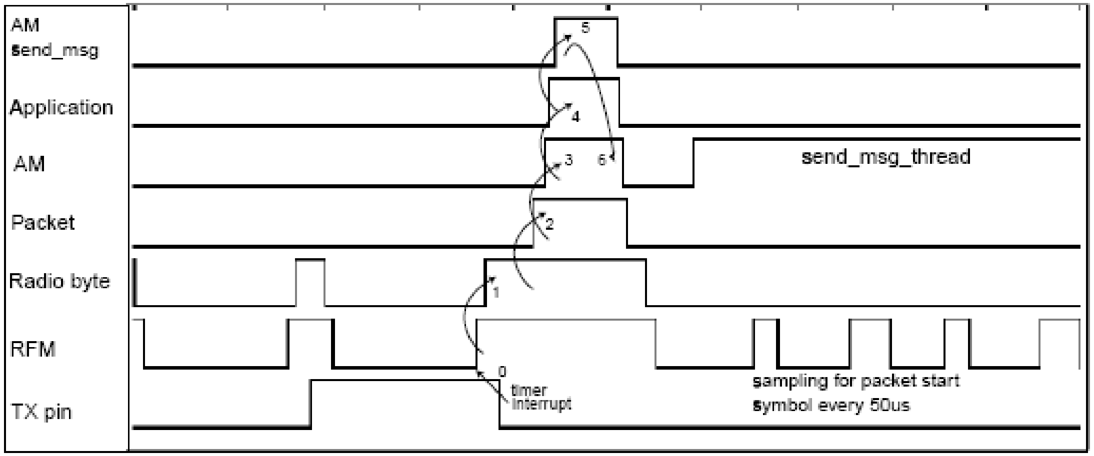
- Most expensive are interrupts particularly software operations which involve saving and restoring of registers.

Evaluation

- Efficient Modularity:

Events and commands propagate through components quickly.

Figure shows dynamic composition of the crossings.



0. Hardware interrupt timer at step 0.
1. TX_bit_evt propagates event into byte level processing.
2. Handler issues command to transmit final bit and fire TX_byte_ready.
3. TX_packet_done.
4. send_msg command has finished.
5. Post a task send_msg_task to send the packet
6. Prepare message and send.

Evaluation

- Limited physical parallelism and controller hierarchy:

Components	Packet reception breakdown	Percent CPU Utilization	Energy (nJ/bit)
AM	0.05%	0.02%	0.33
Packet	1.12%	0.51%	7.58
Radio handler	26.87%	12.16%	182.38
Radio decode task	5.48%	2.48%	37.2
RFM	66.48%	30.08%	451.17
Radio Reception	-	-	1350
Idle	-	54.75%	-
Total	100.00%	100.00%	2028.66
Components	Packet transmission breakdown	Percent CPU Utilization	Energy (nJ/bit)
AM	0.03%	0.01%	0.18
Packet	3.33%	1.59%	23.89
Radio handler	35.32%	16.90%	253.55
Radio encode task	4.53%	2.17%	32.52
RFM	56.80%	27.18%	407.17
Radio Transmission	-	-	1800
Idle	-	52.14%	-
Total	100.00%	100.00%	4317.89

- Processor is still idle 50% of the time under highly active periods.
- e.g.: low level bit and byte processing utilize significant CPU resources but the CPU is not the system bottleneck.
- If bit-level functions were implemented on separate microcontroller, a performance gain is not expected because of radio bandwidth limitations.
- Additional power/time incurred in transferring data between microcontrollers.
- If components were implemented by dedicated hardware, we would be able to make several power saving design choices including sleeping or lowering the frequency of the processor.

Evaluation

- Diversity in usage and robust operation:

- Versatility of architecture tested by creating sample applications that exploit the modular structure of the system.
- Ability to target multiple CPU architectures in future systems by developing the system in C.
- Multi-hop routing application automatically reconfigures itself to withstand individual node failures so that sensor network as a whole is robust.

Architectural Implications & Conclusions

- Authors show that its possible to have multiple flows of data using single microcontroller and hence using multiple microcontrollers is a option and not a requirement.
- The interconnect between multiple microcontrollers will need to support an efficient event based communication model.
- Authors conclude that Bit level processing cannot achieve same data transfer rates as Bluetooth and hence the Radio Byte component needs to become a hardware abstraction rather than synthetic hardware.
- Inclusion of additional hardware support for events would make a significant performance impact. E.g.: additional set of registers for the execution of events would save about 20% of CPU load (meaning lower power consumption or higher performance).
- Reconfigurable computing in the future will require implementing interconnect and controller hierarchy (integrated onto a chip) using FPGA’s trivially rather than UART’s and DMA’s.
- Authors thus attempt to provide a systematic analysis architectural alternatives in the network sensor regime.