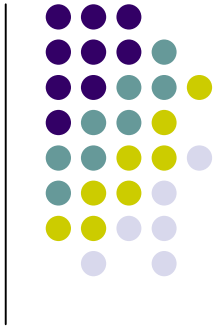
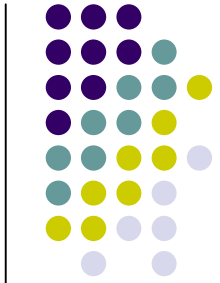


System Architecture Directions for Networked Sensors

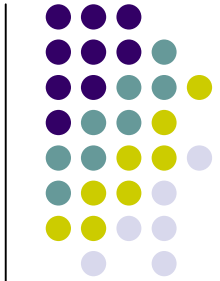
Hrishikesh Thaker

Outline



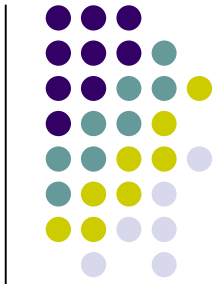
- Introduction
- Network Sensor Characteristics
- Hardware Design
- Tiny OS
- Effectiveness
- Conclusion
- References

Introduction



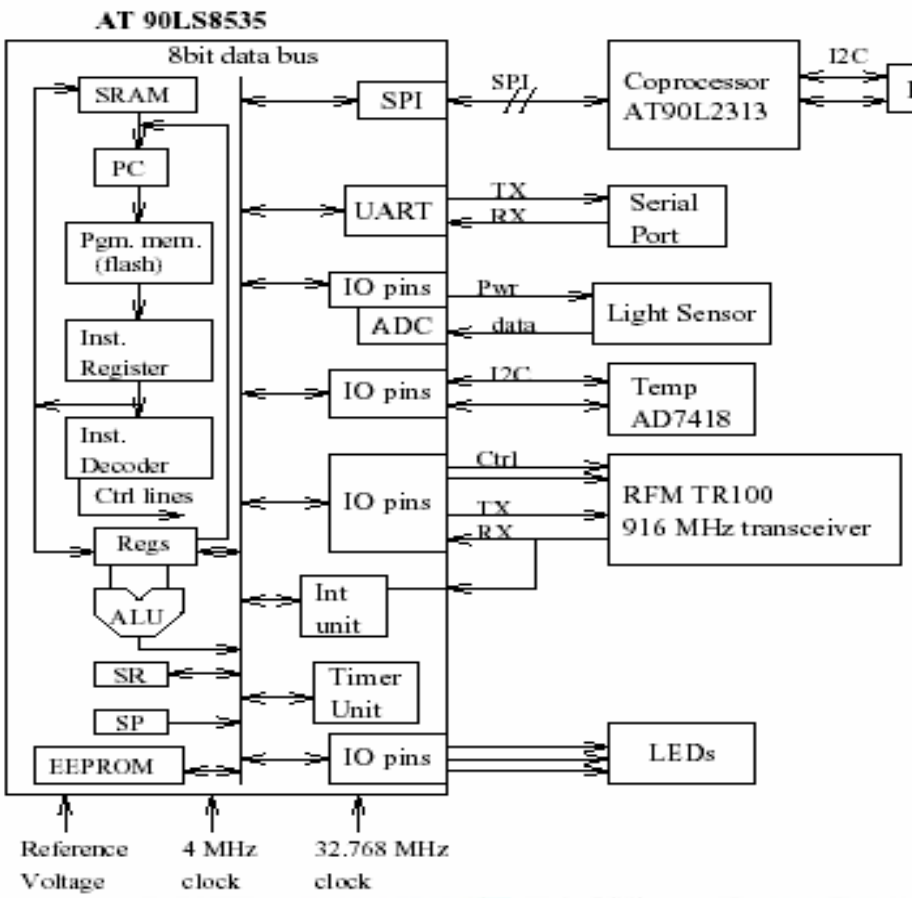
- Technological progress
- Missing Elements
  1. Overall system architecture
  2. Methodology for Systematic advance
- Key Requirements
  1. Develop a small device
  2. Design a Tiny event-driven OS that support efficient modularity and concurrency-intensive operation.

Network Sensor Characteristics

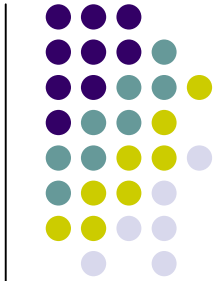


- **Small Physical size**
- **Low Power consumption (Fraction of Watt).**
- **Concurrency intensive operation.**
- **Limited Physical Parallelism and Controller Hierarchy**
- **Diversity in design and Usage**
- **Robust operation**

Schematic Diagram

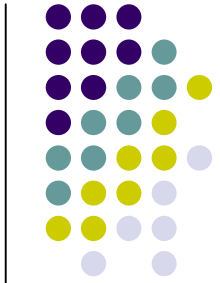


TinyOS Introduction



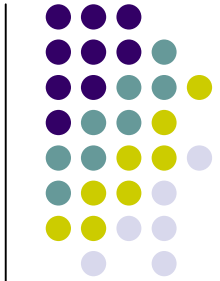
- **TinyOS is an embedded operating system, written in NesC programming language to operate within the severe memory constraints inherent in sensor networks**
- **Small physical size, modest active power load, tiny inactive load hardware design**
- **Designed to support concurrency intensive operations required by network sensors to achieve efficient modularity and robustness**
- **It combines sensing, communication, and computation into single architecture**

System Challenge



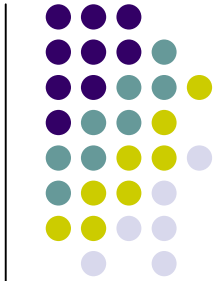
- Concurrency intensive operations
  - Unsynchronized, Multiple , high data flow
  - Low memory and low power consumption
- Bit by Bit interaction with radio
- Small physical size – (no multiple controllers, direct interface with micro controller)
- Modularity – application specific

Tiny OS Overview

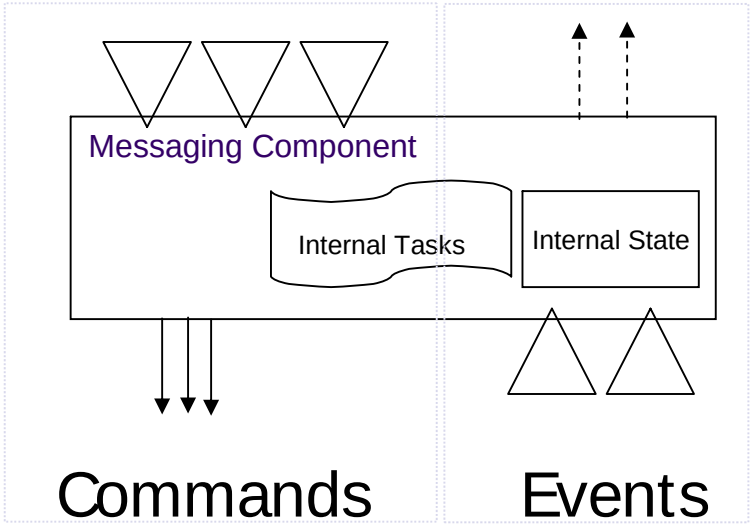


- Event driven model.--- uses CPU efficiently
- Two level scheduling (Event and Tasks)
- Fine-grain multithreading
- Static memory allocation
- System composed of state machines
- Each state machine is a TinyOS “component”
- Provides efficient battery usage

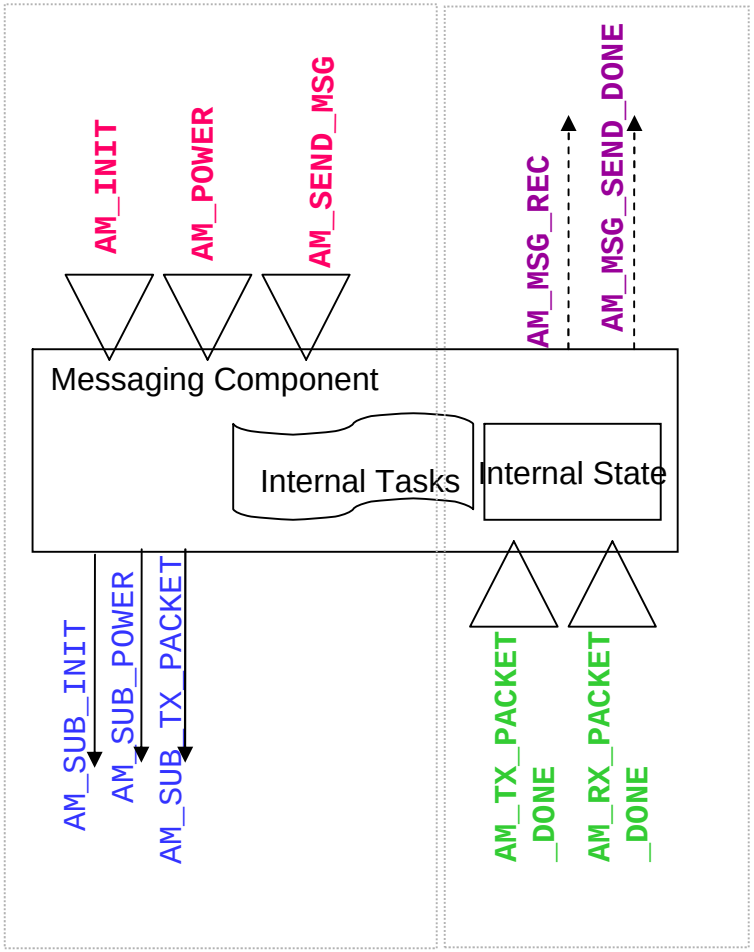
Tiny OS Component Design



- **Every component has**
  - **Frame**
  - **Tasks**
  - **Command Handler Interface**
  - **Event Handler Interface**
- **Frame: static storage model**
- **Command and events are function calls**

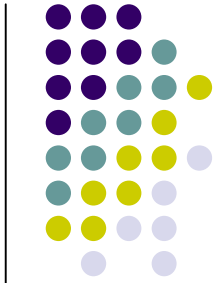


TOS Component



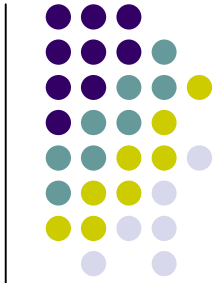

```
                                    //AM.comp//
AM TOS_MODULE;
//ACCEPTS{
  char AM_SEND_MSG(char addr, char type,
char* data);
  void AM_POWER(char mode);
  char AM_INIT();
};
//SIGNALS{
  char AM_MSG_REC(char type,          char*
data);
  char AM_MSG_SEND_DONE(char success);
};
//HANDLES{
  char AM_TX_PACKET_DONE(char success);
  char AM_RX_PACKET_DONE(char* packet);
};
//USES{
  char AM_SUB_TX_PACKET(char* data);
  void AM_SUB_POWER(char mode);
  char AM_SUB_INIT();
};
```

TOS Component



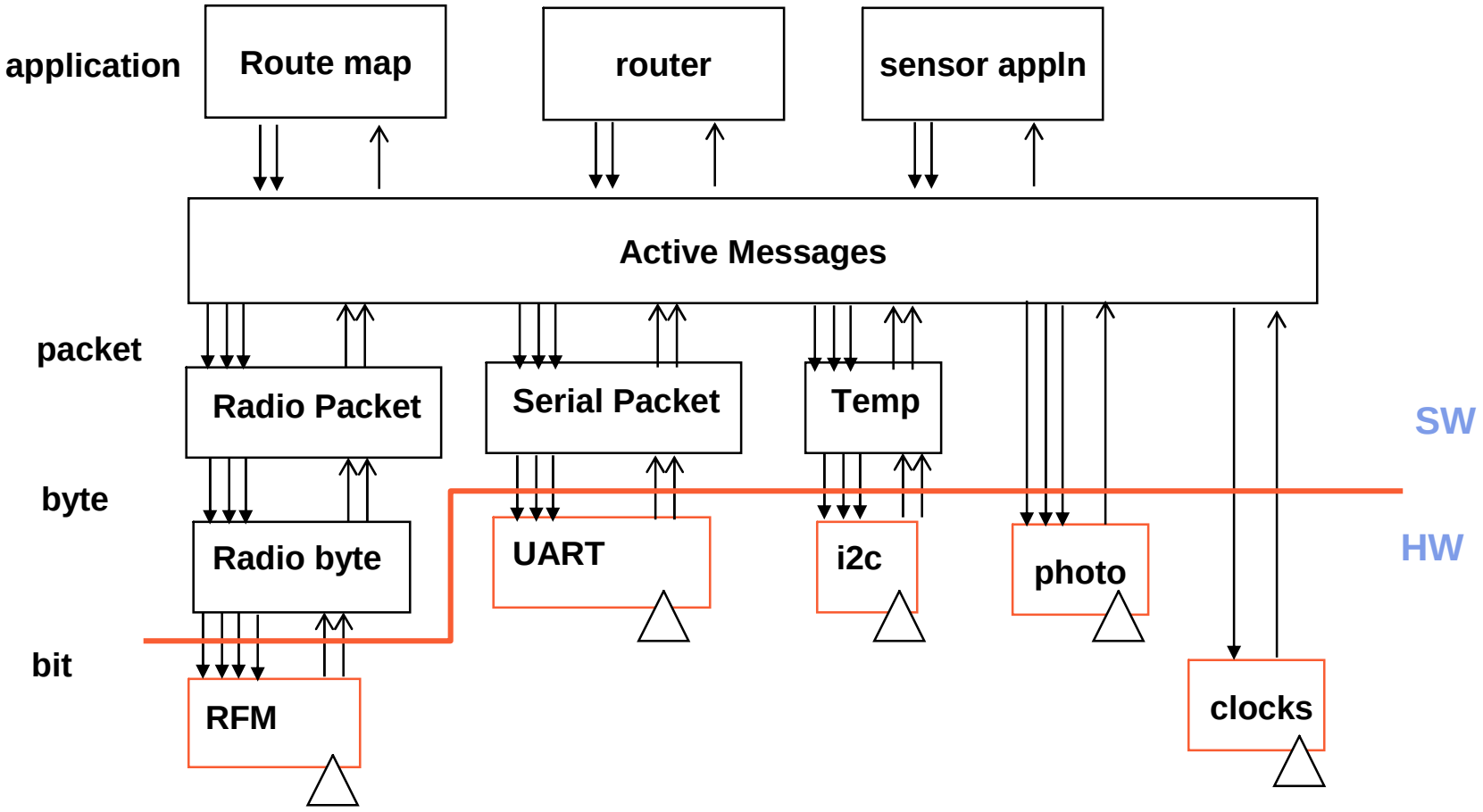
- **Two level scheduling structure (events and tasks)**
- **Scheduler is simple FIFO**
- **Bound on the number of pending tasks.**
- **Tasks cannot preempt other tasks.**
- **Scheduler is power aware**
  - ***Puts processor into* Sleep mode *when queue is empty*.**

TOS Component Types

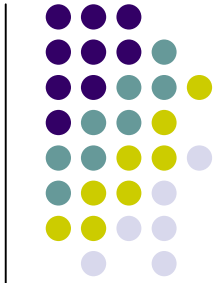


- **Hardware Abstractions**
  - **RFM radio component**
- **Synthetic Hardware**
  - Radio Byte component
- **High Level Software**
  - **Messaging Module**
  - **Performs control, routing and all data transformations**
  - **Filling in a packet buffer and dispatches received messages**

Simple Configuration



Evaluation



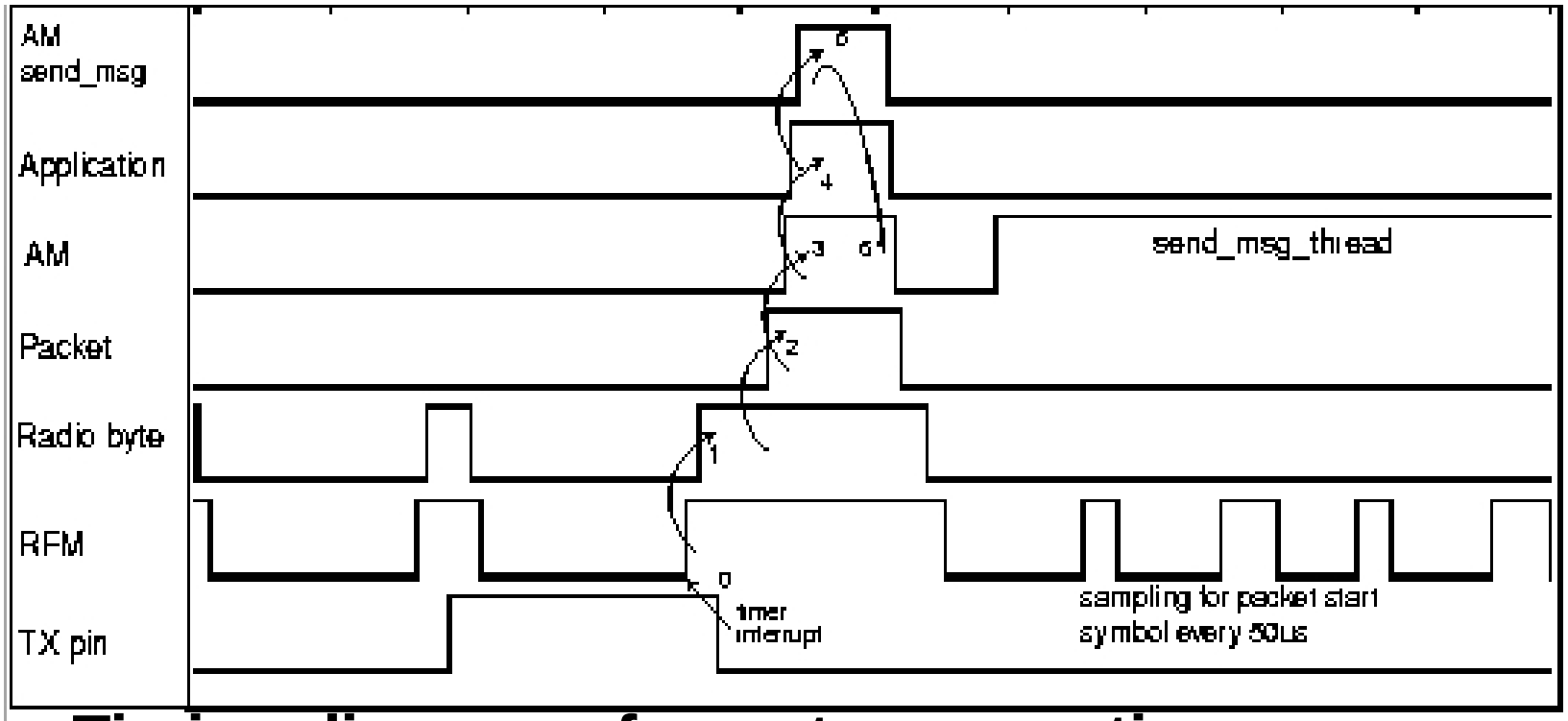
1. **Small Physical Size**

- **Entire application takes 226 bytes of data, under 50% of 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
Procesor_init	172	30
TinyOS scheduler	178	16
C runtime	82	0
Total	3450	226

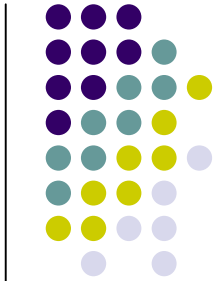
Efficient Modularity

- Events and Commands propagates quickly
- Total propagation delay up the 5 layer radio communication stack is about 40 micro-seconds



Timing diagram of event propagation

Power Breakdown



	Active	Idle	Sleep
MCU	5 mA	2 mA	5 µA
LED	4.6 mA each	0	0
EE-Prom	3 mA	0	0
Radio	7 mA (TX)	4.5 mA (RX)	5 µA
Photo Diode	200 µA	0	0
Temperature	200 µA	0	0

- **When active, power consumption of LED and Radio reception are about equal to processor**
- **Processor, radio, sensors at peak load consumes 19.5mA at 3.0 V or about 60 mW**

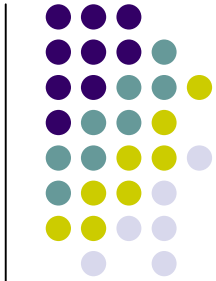
Work and Energy breakdown

Components	Packet reception work breakdown	CPU Utilization	Energy (nj/Bit)
AM	0.05%	0.20%	0.33
Packet	1.12%	0.51%	7.58
Radio handler	26.87%	12.16%	182.38
Radio decode thread	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

- Successfully demonstrated a system with multiple flows of data running through a single microcontroller

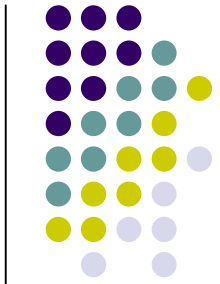
Slide courtesy Jason Hill et al

Conclusions



- Small memory size
- Power efficient
  - Put micro controller and radio to sleep
- Efficient modularity
- Concurrency-intensive operations
  - Event-driven architecture
  - Efficient interrupts/events handling
- Real-time
  - Non-preemptable FIFO task scheduling
  - No real-time guarantees or overload protection

References



- **“A wireless embedded sensor architecture for system-level optimization” by Jason hill and David Culler**