TinyOS: Embedded Software for Wireless Sensor Networks

Sinem Coleri
Anshuman Sharma

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

Motivation for Sensor Networks

- Primary function
  - Sample environment for sensory information
  - Propagate or process data
- Applications
  - Traffic density measurements in highways
  - Determination of duration of traffic lights
  - Car detection in parking garages
  - Environment monitoring
    - Light, temperature

Hardware Platform

- Current networked sensor
  - Two board sandwich
    - Main board with radio comm.
      - 4MHz, 8 bit MCU (ATMEL)
      - 512 bytes RAM, 8K ROM, 512 bytes EEPROM
      - Small co-processor unit, serial port, LED outputs
      - 900 MHz radio (RF Monolithics)
    - Sensor Board
      - Light, temperature, magnetic field
- Future networked sensors
  - Communication, computation and MEMS devices in microscopic scale chips

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication

Motivation for TinyOS

- Requirements shaping the design of networked sensors
  - Small physical size
    - Constrains storage
  - Low power consumption
    - Constrains processing, communication
  - Concurrency intensive operation
    - Sampling sensor, streaming data from or into network, processing data simultaneously

Motivation for TinyOS (cont...)

- Diversity in design and usage
  - Requires efficient modularity
    - Application specific devices, not general purpose
    - Important to assemble just the software components to synthesize app. from hardware components

TinyOS

- Simple component based OS
  - Subset of components used for particular application
  - Components are reentrant cooperating state machines
- Efficient modular composition
  - Overhead of modularity eliminated by static info
- Maintains high level of concurrency in limited space
  - Refusing requests for memory inside the application
- Uses power efficiently
  - Spending unused CPU cycles in sleep
  - Turning radio off when not is use

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

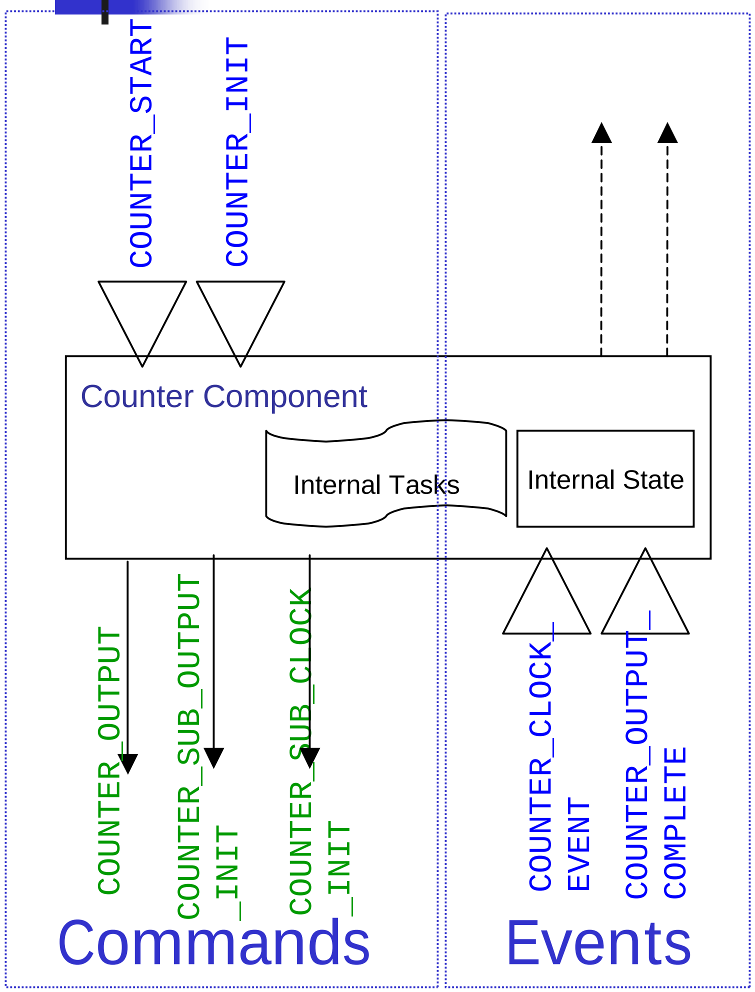
Complete TinyOS application

- Scheduler
- Graph of components
  - Each component has
    - Interface*(.comp)*
    - Internal Implementation*(.c)* (eg. VHDL, Verilog)

Complete TinyOS application:Component

- Interface comprises of synchronous *commands* and asynchronous *events*
  - Upper Interface
    - Commands it implements
    - Events it signals
  - Lower Interface
    - Commands it uses
    - Events it handles
- Internal Storage
  - Structured into a *fixed-size frame*
- Internal Concurrency
  - Light-weight threads – *tasks*

Component Interface




```
//COUNTER.comp//
TOS_MODULE COUNTER;

ACCEPTS{
    char COUNTER_START(void);
    char COUNTER_INIT(void);
};

USES{
        char COUNTER_SUB_CLOCK_INIT(char
    interval, char scale);
        char COUNTER_SUB_OUTPUT_INIT();
        char COUNTER_OUTPUT(int value);
};

HANDLES{
    void COUNTER_CLOCK_EVENT(void);
    char COUNTER_OUTPUT_COMPLETE(char
    success);
};

SIGNALS{
};
```

Complete TinyOS application:Component Entities

- Command
  - Function call across component boundaries
  - Can post tasks, call commands
  - Returns status
    - Way of managing limited storage
  - Example
    - Sending packet
    - Sampling sensor

Complete TinyOS application:Component Entities

- Event
  - Up-call for notification of action
  - Interrupt at the lowest level
  - Can post tasks, call commands, signal events
  - Example
    - Receiving packet
    - Clock interrupt

Complete TinyOS application:Component Entities

- Task
  - Way to incorporate arbitrary computation
  - Can post tasks, call commands, signal events
  - Example
    - Encoding a byte
    - Performing CRC check

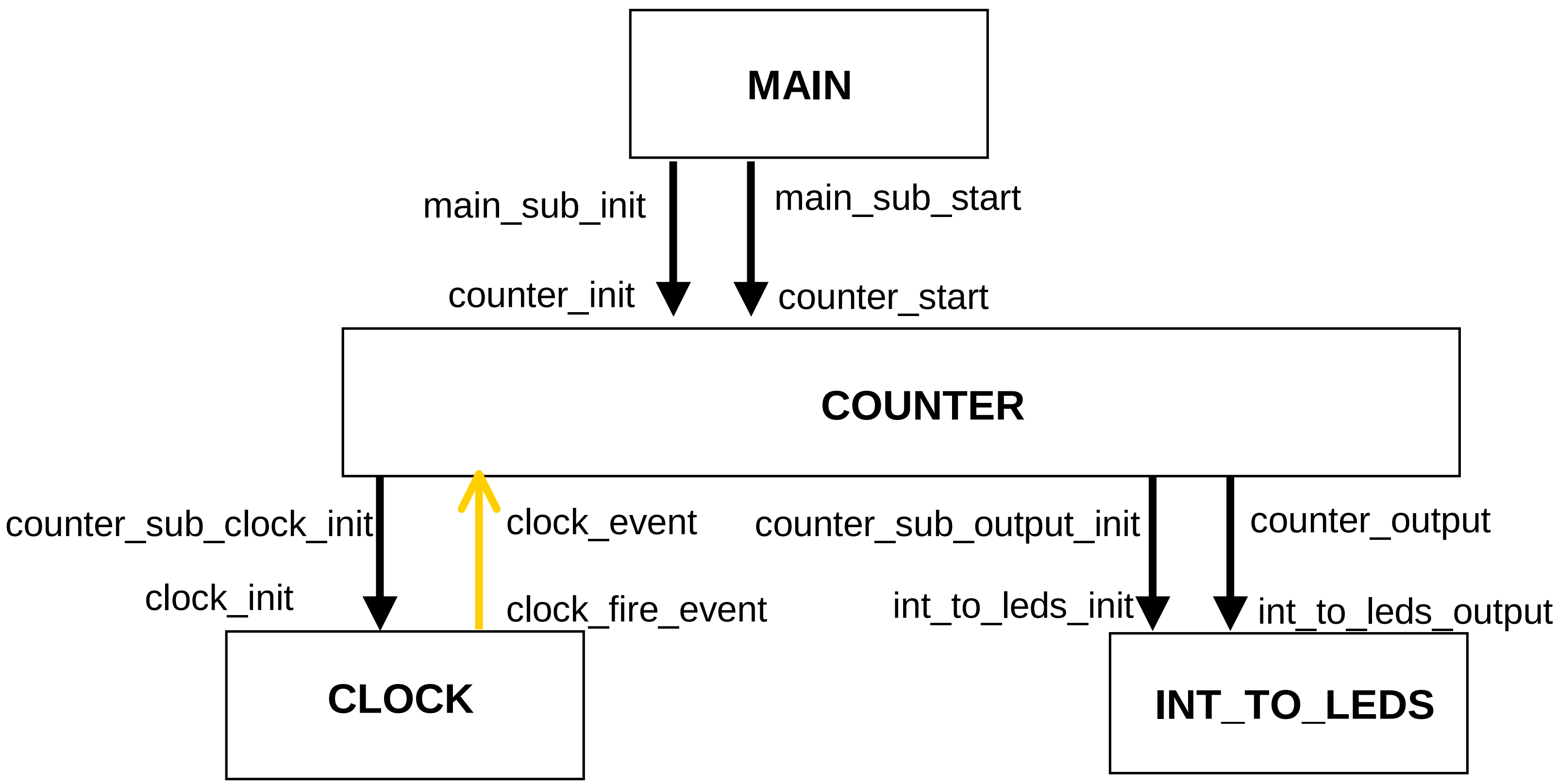
Complete TinyOS application:Component Entities

- Fixed-size frame
  - Internal storage
  - Eliminates the overhead of dynamic memory
  - Determines memory requirement in compile time
  - Example
    - State of component
    - Packet to be sent

Description of Application

- Describes the wiring of the interfaces
  - 1-1 wiring
  - Events to multiple components
  - Multiple components to the same command
- Efficient modularity
  - Optimization by static info

Example Application Desc.



Example Application Desc.

include modules{

MAIN;

COUNTER;

INT_TO_LEDS;

CLOCK;

};

MAIN:MAIN_SUB_INIT COUNTER:COUNTER_INIT

MAIN:MAIN_SUB_START COUNTER:COUNTER_START

COUNTER:COUNTER_CLOCK_EVENT CLOCK:CLOCK_FIRE_EVENT

COUNTER:COUNTER_SUB_CLOCK_INIT CLOCK:CLOCK_INIT

COUNTER:COUNTER_SUB_OUTPUT_INIT INT_TO_LEDS:INT_TO_LEDS_INIT

COUNTER:COUNTER_OUTPUT INT_TO_LEDS:INT_TO_LEDS_OUTPUT

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

Scheduling

- Events have higher priority
  - Events preempt tasks
  - Almost instantaneous event execution
    - Not wait for long latency actions
    - Small amount of work related to component state

Scheduling

- Tasks have lower priority
  - Tasks do not preempt events or other tasks
  - Scheduled by FIFO scheduler
    - Circular buffer keeping pointer to posted tasks
  - Handled rapidly without blocking or polling
    - Unused CPU cycles in sleep state

Scheduling

- Initialize scheduler (init buffer that keeps tasks)
- Issues init command (initialize all other components)
- While(1){
  - While(task buffer non-empty){
    - Take the next task
    - Execute it
    - Remove the corresponding entry from buffer
    - }
  - sleep
  - nop
- }

Puts the processor to sleep but leaves the peripherals operating so that any of them can wake up the system

Hardware interrupt comes

Interrupt handler{
……
Signal event
…
Post task to buffer
}

Event handler{
…
…}

Put task to circular buffer

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

Event-Driven Sensing

- Clock interrupt
- Clock event handler starts ADC conversion
- CPU continues execution or sleeps if nothing else to do
- Sensor interrupt at the end of conversion

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

Communication

- Application Level Communication
- Lower Layer Communication

Application Level Communication

- Use TinyOS to construct a networking infrastructure
  - Self-organized collection of devices
  - Application Level Messaging
  - Stack implementation
    - Requires little storage and power

Tiny Active Messages

- Uses the Active Messages (AM) paradigm
- Overlapping communication and computation through lightweight procedure calls
- Message contains handler name to be invoked on a target node
- Handler does two things
  - Extracts message from network
  - Integrate data into computation or send response

Managing Packet Buffers

- Traditional OS do this in kernel
- Three main issues
  - Encapsulation
  - Data storage reuse
  - Provision of input buffer

Managing Packet Buffers (cont...)

- Buffer provides holes for system specific encapsulation
- The only pointers carried across boundaries
- Send command causes transmit buffer to be "owned"
- Ownership tracking is app. specific
- The "done" event is sent to all components
- Buffer exchange between message handler and system

Lower Layer Communication

- Challenge is to move message from app storage to phy modulation of channel
- We need a cross layer “data pump”
- In the stack
  - The upper component partitions data into subunits
  - The lower component acks this and signals for the next delivery when ready
- Message layer is the packet pump
- byte-by-byte vs. bit-by-bit abstraction

Lower Layer Communication(cont...)

- No controller hierarchy
- Higher level functions can still continue in parallel
- At the base we have a state machine that does bit timing
- RFM component abstracts the real time deadlines from higher layers
- Encoding of data at the same time as transmission
- Reception requires detection of data on channel

Outline

- Motivation for Sensor Networks
- Motivation for TinyOS
- Development Environment for TinyOS
- Scheduling in TinyOS
- Event-driven Sensing
- Communication
- Conclusion

Conclusion

- Effective approach for highly constrained devices
- Non-blocking, event driven model facilitates interleaving processor among multiple flows
- Incremental processing of messages at different levels
- Event and task logical concurrency used everywhere but in the hardware
- Extremely modular design gives way to experimentation