Event Driven Sensor Systems: Tiny OS, Tiny GALS, galsC and nesC

Group Members:
Chetan Fegade
Nikhil Mascarenhas

Mentor: Dr. Yann-Hang Lee
Outline

1. Introduction
2. TinyGALS programming model
3. TinyOS
4. NesC
5. Middleware
6. Conclusion
7. References
8. Q & A
Introduction

- Event driven sensor systems/networks ???
- Two types of OS models followed in embedded systems programming –
  1. Event driven programming
     Useful for computing devices with scarce resources, energy efficient
  2. Multi-threaded programming
     Classical approach, polling is done, resource use is intense
- WSN ?
Wireless Sensor Networks

» The basic unit which integrate to form a Wireless Sensor Network is known as “Mote” or ‘Node’
TinyGALS

» A programming model for event driven embedded systems (Sensor networks)

» **GALS** – Globally Asynchronous Locally Synchronous (related to Computing systems)

» Synchronous -> Computation that takes place very quickly as compared to EVENT arrival rate and program flow transfers immediately to calling code

» Asynchronous -> Control flow does not transfer immediately and hence execution is decoupled
Why different programming model?

• Increase in the complexity of applications

• Implementing concurrency with event driven approach is not feasible with current high level sequential embedded programming languages like C, C++
Important Assumptions :-

- Application would be developed to work on a Single processor systems
- A complete application runs in one thread
- Hardware interrupts is the only way of preemption
- Interrupts are non-entrant
Hierarchy of programming model
Elements of programming model

» Components – {COUNTER, CLOCK, INT_TO_LEDS}
» Modules – {count, leds}
» Application
Component ‘C’

» $C = (V_C, X_C, I_C)$

» $V_C$ – Internal variables

» $X_C$ – External variables

» $I_C$ – Group of methods

» $I_C$ is further divided into 2 groups as,

   * ACCEPTS$_C$ – Inputs to components and can be called by other components

   * USES$_C$ – Outputs of components and may belong to other components

Following EVENTS are responsible for activation of component $C$:

1. Abstracted hardware generates interrupt – Source component
2. Event triggered on input port of module module to which component is connected via method call - Triggered component
3. Activated by another component connected before it - Called components

» Locally Synchronous
Module ‘M’

- \( M = (\text{COMPONENTS}_M, \text{INIT}_M, \text{INPORTS}_M, \text{OUTPORTS}_M, \text{PARAMETERS}_M, \text{LINKS}_M) \)

- COMPONENTS\(_M\) – Integrated components
- INIT\(_M\) - Initialization methods
- INPORTS\(_M\) - List of input ports
- OUTPORTS\(_M\) - List of output ports
- PARAMETERS\(_M\) - Global variable mappings
- LINKS\(_M\) – Interconnections within module

A specific module may be activated in following cases ->
1. One of the Source components in it activated
2. One of the Triggered component is activated

- Scheduling Algorithm is First In First Out (FIFO)
- Globally Asynchronous
Application ‘A’

A = (MODULES_A, GLOBALS_A, VARMAPS_A, CONNECTIOSN_A, START_A)

- MODULES_A - Integrated modules
- GLOBALS_A - List of global variables used by application
- VARMAPS_A - List of global variable mappings with respect to the parameters
- CONNECTIOSN_A - Interconnections between modules
- START_A - Entry method
TinyGUYS (Global Yet Synchronous)

» This technique allows the Global variables to be shared between modules

» Modules are decoupled from each other via message passing

» But if there is a shared variables between modules which needs to be updated very frequently, it will overcrowd the scheduler queue and being FIFO scheduling other tasks would be unnecessarily delayed and this will make the whole programming model slow and inefficient.

» For this purpose, Write to a global variable takes place asynchronously to a buffer maintained for that specific variable, Read from this variable can be done Synchronously.
Tiny OS
Component Based operating system and platform for wireless sensor networks – Wiki

What makes Tiny OS suitable for Event Driven WSN

- Fits well into the physical constraints the WSN
- Reactive
- Flexible
- Application specific OS
OS Design

» Basic building block: Component

» Components are wired via interfaces to form a graph.
Commands, Events and Tasks

» Commands are request for services

» Events are signals of completion

» Split Phase operation

» Tasks represents functions to be executed later.

» Task can be either from a ‘command and event’ or ‘interrupt’
Interfaces

- StdControl
- Timer
- TimerM
- HWClock
Application is a graph of components.
Interface is either *provides* services or *uses* service.
The entire tiny OS code is written in nesC language.

Code for TimerM component

```nesC
module TimerM
{
 provides interface StdControl;
 provides interface Timer;
 uses interface Clock;
}
```
What does an interface look like

interface StdControl {
    command result_t init();
    command result_t start();
    command result_t stop();
}

interface Timer {
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event result_t fired();
}

interface Clock {
    command result_t setRate(char interval, char scale);
    event result_t fire();
}
configuration TimerC {
  provides {
    interface StdControl;
    interface Timer;
  }
  implementation {
    components TimerM, HWClock;
    TimerM.Clk -> HWClock.Clock;
  }
}
More on NesC

» No pointers allowed in nesC
» No dynamic memory allocation.
» NesC Compiler detects race condition at compile time.
» It generates a single executable file
“middleware is computer software that provides services to software applications beyond those available from the operating systems” - Wiki
Design Challenges

» Abstraction Support
» Data Compatibility
» Resource Constraints
» Dynamic Topology
» Scalability
» Security
Hence we conclude by saying Tiny OS may not be the best solution for WSN but it is one of the optimum solutions.
References

» http://en.wikipedia.org/wiki/TinyOS
» http://en.wikipedia.org/wiki/Middleware
» http://ptolemy.eecs.berkeley.edu/papers/03/TinyGALSreport/TinyGALSreport.pdf
THANK YOU!