Synchronous Programming of Device Drivers for Global Resource Control in Embedded Operating Systems

Nicolas Berthier

Supervisors: Florence Maraninchi & Laurent Mounier

Synchrone Team

CITI — January 27, 2011

Nicolas BERTHIER

Supervisors: Florence MARANINCHI & Laurent MOUNIER

Synchrone Team

CITI — January 27, 2011
Context: Wireless Sensor Networks

Components

- $\mu$-Controller (MCU)
- Radio Transceiver(s)
- Sensors
- Battery
- ...

Constraints, Problems

- Slow Computations
- Small Memory
- Battery-Awareness
Context: Wireless Sensor Networks

Components

- µ-Controller (MCU)
- Radio Transceiver(s)
- Sensors
- Battery
- ...

Constraints, Problems

- Slow Computations
- Small Memory
- Battery-Awareness
- ...

Constraints, Problems

- Slow Computations
- Small Memory
- Battery-Awareness
- ...

Components

- µ-Controller (MCU)
- Radio Transceiver(s)
- Sensors
- Battery
- ...

Context: Wireless Sensor Networks
Example WSN Hardware Platform

Wsn430

- CPU
- MCU
- ROM
- RAM
- Timers

Shared Resources (Hardware Modules, Buses, ...
Example WSN Hardware Platform

Wsn430
Example WSN Hardware Platform

Wsn430

- CPU
- MCU
- ROM
- RAM
- Timers
- I/O
- GPIO
- USART0
- USART1
- SPI
- UART
- \( I^2C \)

Shared Resources (Hardware Modules, Buses...)
Example WSN Hardware Platform

Wsn430

- MCU
- ROM
- RAM
- Timers
- I/O
- GPIO
- USART0
- USART1
- I2C
- SPI
- UART
- I2C
- SPI
- UART
- RS232
- LEDs
- Humid.
- SID
- Radio
- Flash
Example WSN Hardware Platform

Wsn430
Example WSN Hardware Platform

Wsn430
Hardware Behavior: MCU Automaton

TI MSP430 Operating Modes

Discrete States
Hardware Behavior: MCU Automaton

TI MSP430 Operating Modes

- **Discrete States**
- **Power Consumption**

### Operating Modes

The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 2−10. The operating modes take into account three different needs:

- **Ultralow-power**
- **Speed and data throughput**
- **Minimization of individual peripheral current consumption**

The MSP430 typical current consumption is shown in Figure 2−9.

<table>
<thead>
<tr>
<th>Mode</th>
<th>CPU and Clocks Status</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>CPU is active, all enabled clocks are active</td>
<td>Low</td>
</tr>
<tr>
<td>LPM0</td>
<td>CPU, MCLK are disabled</td>
<td>Low</td>
</tr>
<tr>
<td>LPM1</td>
<td>CPU, MCLK, DCO osc. are disabled</td>
<td>Low</td>
</tr>
<tr>
<td>LPM2</td>
<td>CPU, MCLK, SMCLK, DCO osc. are disabled</td>
<td>Low</td>
</tr>
<tr>
<td>LPM3</td>
<td>CPU, MCLK, SMCLK, DCO osc. are disabled</td>
<td>Low</td>
</tr>
<tr>
<td>LPM4</td>
<td>CPU and all clocks disabled</td>
<td>Low</td>
</tr>
</tbody>
</table>

When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. The peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.
Hardware Behavior: Radio Automaton

Chipcon CC1100 Simplified Control State Diagram

- Default state when the radio is not receiving or transmitting. Typ. current consumption: 1.6 mA.
- Used for calibrating frequency synthesizer upfront (entering receive or transmit mode can then be done quicker). Transitional state. Typ. current consumption: 8.2 mA.
- Frequency synthesizer is on, ready to start transmitting. Transmission starts very quickly after receiving the STX command strobe. Typ. current consumption: 8.2 mA.
- Typ. current consumption: 13.5 mA at -6 dBm output, 16.9 mA at 0 dBm output, 30.7 mA at +10 dBm output.
- In FIFO-based modes, transmission is turned off and this state entered if the TX FIFO becomes empty in the middle of a packet. Typ. current consumption: 1.6 mA.
- In FIFO-based modes, reception is turned off and this state entered if the RX FIFO overflows. Typ. current consumption: 1.6 mA.

Lowest power mode. Most register values are retained. Current consumption typ 400 nA, or typ 900 nA when wake-on-radio (WOR) is enabled.
Hardware Behavior: Radio Automaton

Chipcon CC1100 Simplified Control State Diagram

**Transmit mode**

1. **IDLE**
   - Default state when the radio is not receiving or transmitting.
   - Typ. current consumption: 1.6 mA.

2. **TX FIFO underflow**
   - Optional transitional state. Typ. current consumption: 8.2 mA.

3. **TXOFF_MODE = 00**
   - Optional transitional state. Typ. current consumption: 8.2 mA.

4. **TXOFF_MODE = 01**
   - Typ. current consumption: 13.5 mA at -6 dBm output, 16.9 mA at 0 dBm output, 30.7 mA at +10 dBm output.

5. **STX**, **STX or RXOFF_MODE = 01**, **STX or RXOFF_MODE = 10**, **STX or RXOFF_MODE = 11**
   - Typ. current consumption: 8.2 mA.

6. **SRX** or **SFSTXON or wake-on-radio (WOR)**
   - Frequency synthesizer startup, optional calibration, settling.
   - Typ. current consumption: 8.2 mA.

7. **SRX** or **STX** or **SFSTXON** or wake-on-radio (WOR)
   - Frequency synthesizer is on, ready to start transmitting. Transmission starts very quickly after receiving the STX command strobe.
   - Typ. current consumption: 8.2 mA.

8. **SRX or STX or SFSTXON** or wake-on-radio (WOR)
   - Frequency synthesizer is turned on, can optionally be calibrated, and then settles to the correct frequency.
   - Transitional state. Typ. current consumption: 8.2 mA.

9. **STX or RXOFF_MODE = 01**
   - Typ. current consumption: 0.16 mA.

**Receive mode**

1. **RX FIFO overflow**
   - In FIFO-based modes, reception is turned off and the state entered if the RX FIFO overflows.
   - Typ. current consumption: 1.6 mA.

2. **RXOFF_MODE = 00**
   - In FIFO-based modes, transmission is turned off and this state entered if the TX FIFO becomes empty in the middle of a packet.
   - Typ. current consumption: 1.6 mA.

3. **RXOFF_MODE = 01**
   - Typ. current consumption: from 14.4 mA (strong input signal) to 15.4 mA (weak input signal).

**IDLE**

- Used for calibrating frequency synthesizer upfront (entering receive or transmit mode can then be done quicker).
- Transitional state. Typ. current consumption: 8.2 mA.

**Sleep**

- Lowest power mode. Most register values are retained.
- Current consumption typ: 400 nA, or typ 900 nA when wake-on-radio (WOR) is enabled.

**Optional freq. synth. calibration**

- Typ. current consumption: 8.2 mA.

**Control of Global Power Consumption?**

+ ADC + MCU + Flash Memory + ...
Hardware Behavior: Radio Automaton

Chipcon CC1100 Simplified Control State Diagram

Transmit mode
- IDLE
  - Manual freq. synth. calibration
  - SFSTXON or RXOFF_MODE = 01
  - Typ. current consumption: 13.5 mA at -6 dBm output, 16.9 mA at 0 dBm output, 30.7 mA at +10 dBm output.
- STX
  - TXOFF_MODE = 01
  - TX FIFO underflow
  - Typ. current consumption: 8.2 mA.
  - Optional transitional state. Typ. current consumption: 8.2 mA.
- TXOFF_MODE = 00
  - Optional transitional state. Typ. current consumption: 8.2 mA.
- RX FIFO overflow
  - Typ. current consumption: from 14.4 mA (strong input signal) to 15.4 mA (weak input signal).
- RXOFF_MODE = 00
  - In FIFO-based modes, reception is turned off and this state entered if the RX FIFO overflows. Typ. current consumption: 1.6 mA.

Receive mode
- IDLE
  - Sleep
    - Lowest power mode. Most register values are retained.
    - Typ. current consumption: 0.16 mA.
  - Crystal oscillator off
    - Typ. current consumption: 0.16 mA.
  - SPWD or wake-on-radio (WOR)
- SRX or TXOFF_MODE = 11
  - Frequency synthesizer is on, ready to start transmitting.
- STX or RXOFF_MODE = 01
  - Frequency synthesizer is turned on, can optionally be calibrated
  - Typ. current consumption: 8.2 mA.
  - Optional transitional state. Typ. current consumption: 8.2 mA.

Control of Global Power Consumption?

+ ADC + MCU + Flash Memory + ...
Programming WSNs: Usual Practice

Applications

Operating System Support / Abstractions

- Multitasking
- System Services
- Hardware Device Drivers
Programming WSNs: Usual Practice

Applications

Operating System Support / Abstractions

- Multitasking
- System Services (Network Stack, File Systems...)
- Hardware Device Drivers
Programming WSNs: Usual Practice

Applications

Operating System Support / Abstractions

- Multitasking
- System Services
- Hardware Device Drivers

Operating Systems Programming for WSNs

- Device Drivers designed *Locally*

(Network Stack, File Systems...)
Programming WSNs: Usual Practice

Applications

Operating System Support / Abstractions
- Multitasking
- System Services
- Hardware Device Drivers

(Network Stack, File Systems...)

Operating Systems Programming for WSNs

- Device Drivers designed Locally
- Ad hoc Solutions for Resource Management & Power-Awareness
Programming WSNs: Usual Practice

Applications

Operating System Support / Abstractions

- Multitasking
- System Services
- Hardware Device Drivers

(Network Stack, File Systems...)

Operating Systems Programming for WSNs

- Device Drivers designed *Locally*
- *Ad hoc* Solutions for Resource Management & Power-Awareness
  ⇒ Decentralized Knowledge!
Problems

Recap
Problems

Recap

▶ Shared Resources
Problems

Recap

► Shared Resources
► Power Management
Problems

Recap

- Shared Resources
- Power Management

➡️ Need for Global Control!
Outline

- Preliminary Remarks
- Proposal
- Implementation
- Summary
Outline

- **Context**

- **Preliminary Remarks**
  - Synchronous Programming
  - Communicating Boolean Mealy Machines
  - From Automata to Device Drivers

- **Proposal**

- **Implementation**

- **Summary**
Synchronous Programming

Main Concept

▶ Communication takes *no time*
▶ Program $\approx$ Synchronous Circuit
Synchronous Programming

Main Concept

- Communication takes *no time*
- Program $\approx$ Synchronous Circuit

Languages

- **Lustre, Signal, Esterel** …
Synchronous Programming

Main Concept

- Communication takes *no time*
- Program $\approx$ Synchronous Circuit

Languages

- Lustre, Signal, Esterel …

Tools

- Model-checking
- Test
- …
Communicating Boolean Mealy Machines

Synchronous Product

\[ \begin{align*}
A_1 & \xrightarrow{\bar{a}} \ x \xrightarrow{a/b} \ x \xrightarrow{a} \ x \xleftarrow{a} \ x \\
A_0 & \xrightarrow{\bar{a}} \\
B_1 & \xrightarrow{\bar{b}} \ x \xrightarrow{b/c} \ x \xrightarrow{b} \ x \xleftarrow{b} \\
B_0 & \xrightarrow{\bar{b}} 
\end{align*} \]
Communicating Boolean Mealy Machines

Synchronous Product
Communicating Boolean Mealy Machines

Synchronous Product
Communicating Boolean Mealy Machines

Synchronous Product
Communicating Boolean Mealy Machines

In Lustre

```lustre
node Sa (a: bool) returns (b: bool);
var inA0, inA1, m_A0, A0: bool;
let
    inA1 = not m_A0;
    inA0 = m_A0;
    b = a and inA1;
    A0 = not a and inA0 or a and inA1;
m_A0 = true -> pre A0;
tel;
```
Communicating Boolean Mealy Machines

In Lustre

```lustre
node SE (a: bool) returns (c: bool);
var inA0, inA1, m_A0, A0, inB0, inB1, m_B0, B0, b: bool;
let
  inA1 = not m_A0; inA0 = m_A0; b = a and inA1;
  inB1 = not m_B0; inB0 = m_B0; c = b and inB1;
  A0 = not a and inA0 or a and inA1; m_A0 = true -> pre A0;
  B0 = not b and inB0 or b and inB1; m_B0 = true -> pre B0;
tel;
```
Communicating Boolean Mealy Machines

Reactive Kernel

```c
bool M1, M2, INIT;  // state variables
void init () { INIT = 1; }  // initialization
```
Communicating Boolean Mealy Machines

Reactive Kernel

```c
bool M1, M2, INIT; // state variables
void init () { INIT = 1; } // initialization

void run_step (bool a) {
    bool L1, L2, L3, L4, L5, L6;

    L2 = INIT | M1;
    L5 = INIT | M2;
    L4 = ~L5 & a;
    L1 = ~L2 & L4;
    L6 = L5 & ~a;
    L3 = L2 & ~L4;

    main_O_c (L1);
    M1 = L3 | L1;
    M2 = L6 | L4;
    INIT = 0;
}
```
From Automata to Device Drivers

$cmd_x()$  Sequencial code sending command $x$ to the device
$\text{goto X}$  Software request
$irq_i$  Hardware request / signal
$int_j$  Internal device event (e.g. end of transmission, etc.)
### From Automata to Device Drivers

#### Preliminary Remarks

From Automata to Device Drivers

Device Peripheral datasheet

- Sequencial code sending command $x$ to the device
- Internal device event (e.g. end of transmission, etc.)

**cmd**

**goto X**

**irq**

**inta** / **irqaa**

**intca** / **irqca**

**intcb** / **irqcb**

**cmdc**

**cmdb**

**cmdbb**

**cmdb()**

**cmdbb()**

**cmdc()**

**goto B / cmdb()**

**goto B / cmdbb()**

**goto C / cmdc()**

**Software Request**

**Software**

** interrupts**

**cmdc()**

**cmdbb()**

**cmdb()**

**goto X**

**Software request**

**Hardware request / signal**

**Internal device event (e.g. end of transmission, etc.)**
Outline

- Context
- Preliminary Remarks
- Proposal
  - Overview
  - Structure
    - Adaptation Layer
    - Control Layer
    - Device Driver Machines
    - Controller
  - Further Possibilities
    - Best Low-Power Mode
    - Other Possibilities
- Implementation
Principles of the Solution

*(Para-*)Virtualization Concept

- Interception and Control of Software Operations
- *Global* Resource Control ⇒ *Centralized Knowledge*
- May Forbid (or Enforce) Operations
Principles of the Solution

(Para-)Virtualization Concept

- Interception and Control of Software Operations
- *Global* Resource Control
- May Forbid (or Enforce) Operations

⇒ Centralized Knowledge

![Diagram showing the relationship between software requests, hardware platform, global properties, and acknowledgments.](diagram.png)
Overview

Hardware Platform

- MCU
- Timer(s)

Interconnects

- Radio Transceiver
- Flash Memory
Overview

Hardware Platform

- MCU
- Timer(s)
- Interconnects
- Radio Transceiver
- Flash Memory

Operating System

+ Application(s)
+ Network stack
+ ...

Task(s)

- OS Scheduler

Device Drivers
Overview

Hardware Platform
- Flash Memory
- Radio Transceiver
- Interconnects
- Timer(s)
- MCU

Adaptation Layer
- Operating System
  + Application(s)
  + Network stack
  + ...
- Task(s)
- OS Scheduler

Control Layer
- Resource Models & Controller
Structure: Adaptation Layer

Modified part of the Operating System

- Simplified Device Drivers
Structure: Adaptation Layer

Modified part of the Operating System

- Simplified Device Drivers

Interacts with the Control Layer

- Emitting *software requests* to the Control Layer
Structure: Adaptation Layer

Modified part of the Operating System

▶ Simplified Device Drivers

Interacts with the Control Layer

▶ Emitting *software requests* to the Control Layer

▶ Receiving *output events* from the Control Layer

▶ Notifications (Hardware Events, Acknowledgments)
Structure: Adaptation Layer

Modified part of the Operating System

- Simplified Device Drivers

Interacts with the Control Layer

- Emitting *software requests* to the Control Layer
  - Using `on_sw()`

- Receiving *output events* from the Control Layer
  - Notifications (Hardware Events, Acknowledgments)

```c
turn_adc_on ()
if (on_sw (adc_on) = ack_a)
  return success;
timer_wait (some time);  // Consider we can
turn_adc_on ();           // try again later
```
Structure: Adaptation Layer

Modified part of the Operating System

▶ Simplified Device Drivers

Interacts with the Control Layer

▶ Emitting *software requests* to the Control Layer
  ▶ Using `on_sw()`

▶ Receiving *output events* from the Control Layer
  ▶ Notifications (Hardware Events, Acknowledgments)
    
    ```c
    turn_adc_on ()
    if (on_sw (adc_on) == ack_a)
      return success;
    timer_wait (some time);  // Consider we can
    turn_adc_on ();           // try again later
    ```
  ▶ Callbacks (Virtual IRQs)
Overview

Operating System
+ Application(s)
+ Network stack
+ ...

Task(s)

OS Scheduler

Control Layer
Resource Models & Controller

Hardware Platform

MCU
Timer(s)
Interconnects
Radio Transceiver
Flash Memory
Structure: Control Layer

- Receives
  - software requests
  - hardware requests (IRQs)

- Emits notifications

- Manages the Peripheral Devices

**Diagram:**
- Software requests (from the adaptation layer)
- Software callbacks
- Hardware events from interrupt controller
- "function call"
- push / pop request
Structure: Control Layer

- Receives
  - software requests
  - hardware requests (IRQs)
- Emits notifications
- Manages the Peripheral Devices

Event Management Part

- Handle request queues
Structure: Control Layer

- Receives
  - software requests
  - hardware requests (IRQs)
- Emits notifications
- Manages the Peripheral Devices

Event Management Part

- Handle request queues
- Executes the Reactive Part

Reactive Part

- Device Drivers Machines
  - Reactive Kernel
- Resource Operational Code
Principles of the Solution (cont’d)

(Para-)Virtualization Concept

- Interception and Control of Software Operations
- Global Resource Control $\Rightarrow$ Centralized Knowledge
- May Forbid (or Enforce) Operations

Key Elements (Boolean Mealy Machines)

- Resource Automata
  - Inputs: Software Requests...
  - Outputs: Low-Level Code, Notifications...
- Controller
Example of Uncontrollable Automaton

Timer

disable . enable / timer_restart()

disable . enable . irq_timer_expired /
timer_expired

timer_init()

Disabled

enable / timer_start()

disable / timer_stop()
Example of Controllable Automaton

Timer

timer_init() → Disabled

enable . ok_t / ack_t, timer_start()

disable . ok_t / ack_t, timer_stop()

disable . enable . irq_t timer_expired / timer_expired

disable . enable . ok_t / ack_t, timer_restart()
Exclusion of Energy-greedy States: Example

Radio Transceiver || ADC
Exclusion of Energy-greedy States: Example

Radio Transceiver || ADC || Controller
Principles of the Solution *(cont’d)*

**(Para-)Virtualization Concept**

- Interception and Control of Software Operations
- *Global* Resource Control
- May Forbid (or Enforce) Operations

⇒ Centralized Knowledge

**Key Elements (Boolean Mealy Machines)**

- Resource Automata
  - Inputs: Software Requests...
  - Outputs: Low-Level Code, Notifications...
- Controller
Principles of the Solution (cont’d)

(Para-)Virtualization Concept

▶ Interception and Control of Software Operations
▶ Global Resource Control ⇒ Centralized Knowledge
▶ May Forbid (or Enforce) Operations

Key Elements (Boolean Mealy Machines)

▶ Resource Automata
  ▶ Inputs: Software Requests & Approval Signals
  ▶ Outputs: Low-Level Code, Notifications & Acknowledgments
▶ Controller
  ▶ Inputs: Software & Hardware Requests
  ▶ Outputs: Approval Signals

Enforcing Global Properties → Designing the Controller
How to: Selecting the Best MCU Low-Power Mode
How to: Selecting the Best MCU Low-Power Mode

- Reducing Energy Consumption
- ...as usual...

![Diagram of MSP430x1xx Operating Modes For Basic Clock System](image)

- **Active Mode**
  - CPU is Active
  - Peripheral Modules Are Active

- **LPM0**
  - CPU Off, MCLK Off, SMCLK On, ACLK On
  - CPUOFF = 1
  - SCG0 = 0
  - SCG1 = 0

- **LPM1**
  - CPU Off, MCLK Off, SMCLK On, ACLK On
  - DC Generator Off if DCO not used in active mode
  - CPUOFF = 1
  - SCG0 = 1
  - SCG1 = 0

- **LPM2**
  - CPU Off, MCLK Off, SMCLK On, ACLK Off
  - DCO Generator Off
  - CPUOFF = 1
  - SCG0 = 0
  - SCG1 = 1

- **LPM3**
  - CPU Off, MCLK Off, SMCLK Off, DCO Off, ACLK On
  - DC Generator Off
  - CPUOFF = 1
  - SCG0 = 1
  - SCG1 = 1

- **LPM4**
  - CPU and all clocks disabled
  - CPUOFF = 1
  - OSCOFF = 1
  - SCG0 = 1
  - SCG1 = 1

Figure 2−10. MSP430x1xx Operating Modes For Basic Clock System
How to: Selecting the Best MCU Low-Power Mode

- Reducing Energy Consumption
  - ...as usual...

- Wake-up time
  - Latency property
How to: Selecting the Best MCU Low-Power Mode

► Reducing Energy Consumption
  ► ...as usual...

► Wake-up time
  ► Latency property

► To be sure to wake up!
  ► Potential IRQs?
How to: Other Possibilities

- Mutual Exclusion of Accesses to Shared Resources
  - Safety Property
How to: Other Possibilities

- Mutual Exclusion of Accesses to Shared Resources
  - Safety Property

- Controlling Guest Tasks / Resources
  - Modification of the Guest Scheduler
  - Allowing Direct Access to the Resources
How to: Other Possibilities

- Mutual Exclusion of Accesses to Shared Resources
  - Safety Property

- Controlling Guest Tasks / Resources
  - Modification of the Guest Scheduler
  - Allowing Direct Access to the Resources

- Booking Controller
  - Slightly more complex (to use)... fits in the model however
Outline

- Context
- Preliminary Remarks
- Proposal
- Implementation
- Summary
Implementation

Proof of Concept

- Rough Implementation
- Resource Automata and Controller encoded in Argos (~ Lustre)
- Multithreaded, Contiki
- Targetting Wsn430 Platform
Implementation

Proof of Concept

▶ Rough Implementation
▶ Resource Automata and Controller encoded in Argos (≈ Lustre)
▶ Multithreaded, Contiki
▶ Targetting Wsn430 Platform

Practicable?

▶ Extra Memory Footprint: 1.5 to 2.5 KB
▶ Timing Overhead: One Reaction ≈ 1,600 CPU cycles

Comparable to Solutions using Decentralized Control
Outline

- Context
- Preliminary Remarks
- Proposal
- Implementation
- Summary
Summary

Global Resource Control

- Synchronous Programming... in Wireless Sensor Networks!
Summary

Global Resource Control

- Synchronous Programming... in Wireless Sensor Networks!
- Many Possible Extensions 〜 Powerful
Summary

Global Resource Control

- Synchronous Programming... in Wireless Sensor Networks!
- Many Possible Extensions ～ Powerful
- Para-Virtualization Concept ～ Flexible Framework
Summary

Global Resource Control

- Synchronous Programming... in Wireless Sensor Networks!
- Many Possible Extensions
  ≈ Powerful
- Para-Virtualization Concept
  ≈ Flexible Framework

Implementation

- Proof of Concept
- Practicable
- Device Drivers Revealed “easier” to Develop
Perspectives

Evaluation

▶ Efficiency to Reduce Power Consumption?

Soon in the Senslab Testbed...
Perspectives

Evaluation

▶ Efficiency to Reduce Power Consumption?
Soon in the Senslab Testbed...

Automated Control

▶ Using Controller Synthesis
Perspectives

Evaluation

▷ Efficiency to Reduce Power Consumption?
   Soon in the Senslab Testbed...

Automated Control

▷ Using Controller Synthesis

Synchronous Approach

▷ “More-Lustre” Solutions?
▷ Monitoring
▷ Other Domains (Real-Time, OS...)
Thank you

Questions?
Outline

- Example Execution
Example Execution

Example Execution