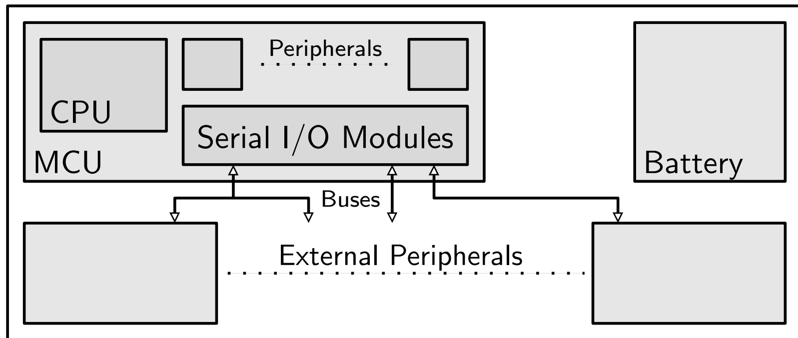
Synchronous Programming of Device Drivers for Global Resource Control in Embedded Operating Systems

Nicolas Berthier   Florence Maraninchi   Laurent Mounier

LCTES 2011 — April 13

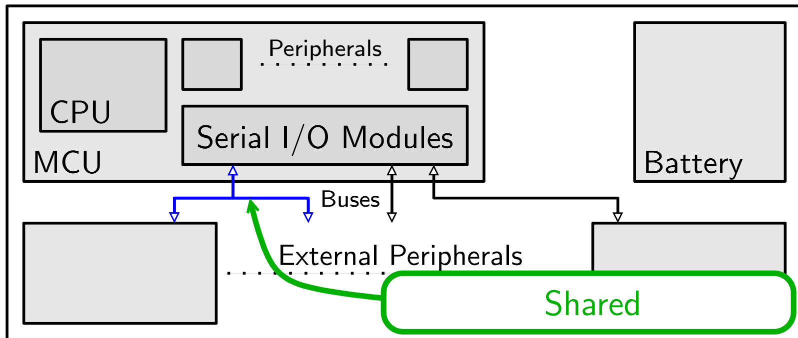
The Need for Global Knowledge & Control

Typical Hardware Platform: Wireless Sensor Network Node



The Need for Global Knowledge & Control

Typical Hardware Platform: Wireless Sensor Network Node

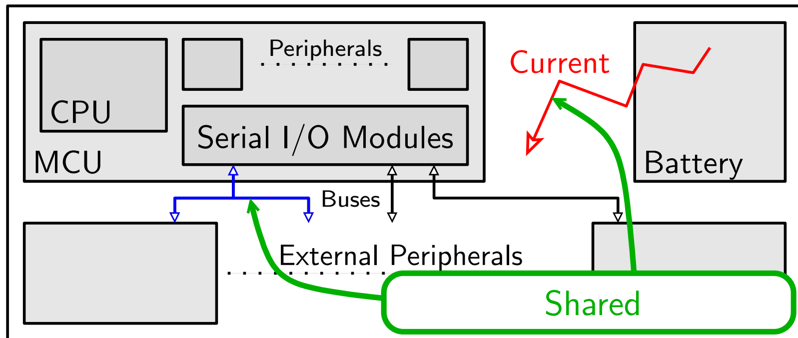


Problems

- Shared Resources (as usual — *e.g.*, Buses)

The Need for Global Knowledge & Control

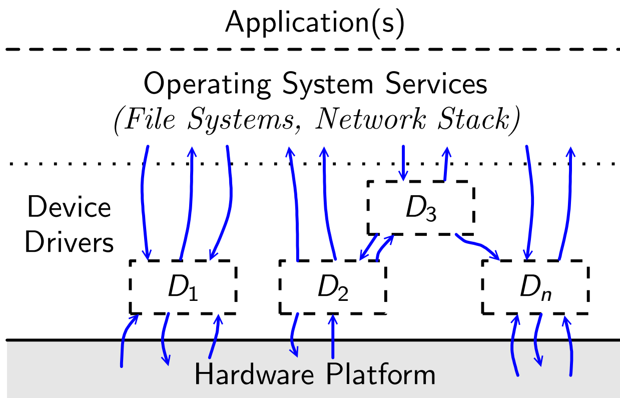
Typical Hardware Platform: Wireless Sensor Network Node



Problems

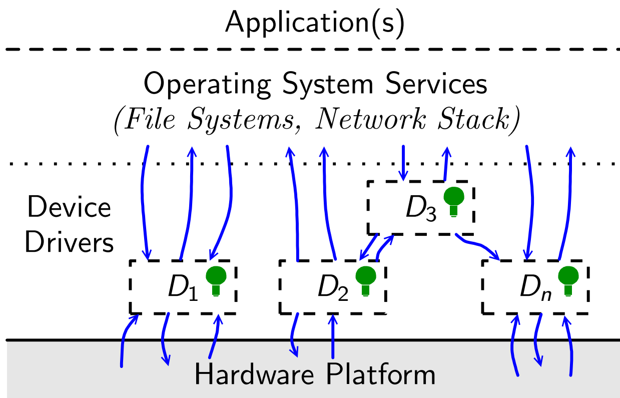
- Shared Resources (as usual — *e.g.*, Buses)
- Avoiding Consumption Peaks (*e.g.*, How to Forbid a Radio Transceiver and an ADC to be Simultaneously in their Highest Consuming State?)

Usual Programming Practice (with an OS)



- Device Drivers Designed Independently of Each Other

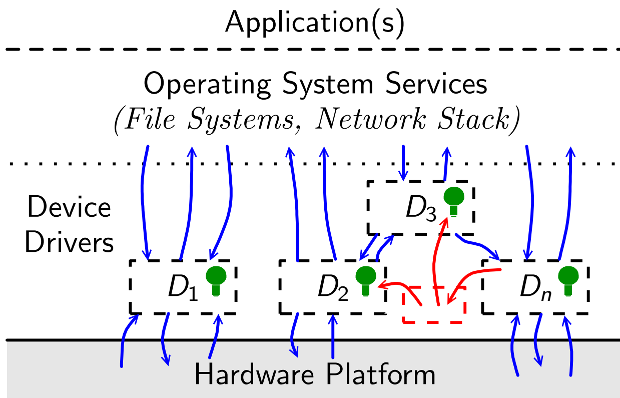
Usual Programming Practice (with an OS)



- Device Drivers Designed Independently of Each Other

$\rightsquigarrow$ **Decentralized Knowledge!**

Usual Programming Practice (with an OS)

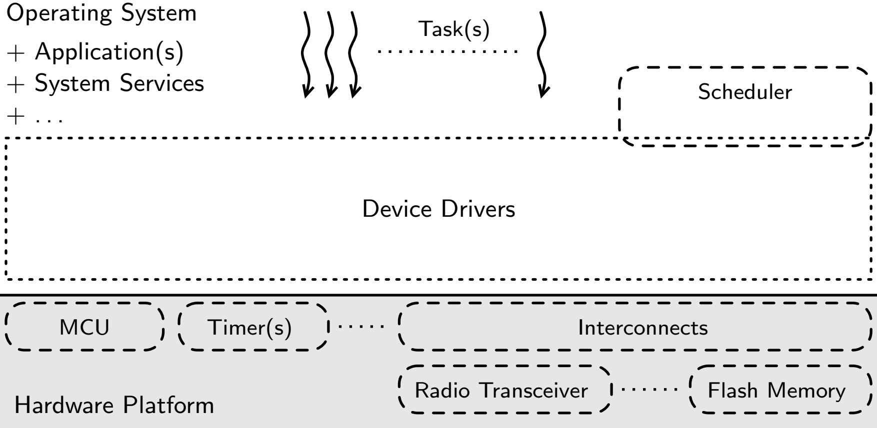


- Device Drivers Designed Independently of Each Other

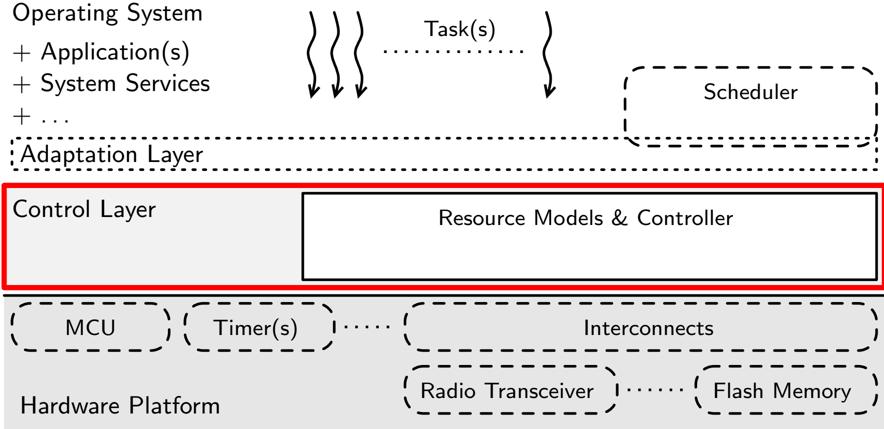
  $\rightsquigarrow$ **Decentralized Knowledge!**

- Ad hoc Solutions for Resource & Power Management

Idea of the Solution



Idea of the Solution



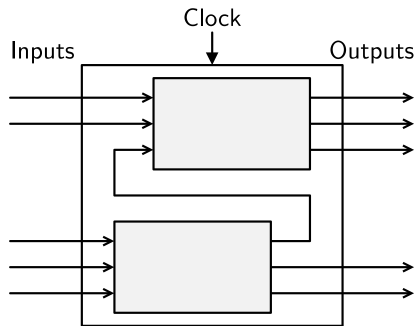
Outline

- Context
- Synchronous Programming in a Nutshell
- Proposal
- Evaluation
- Summary

Synchronous Programming in a Nutshell

Program ≈ Synchronous Circuit

- High-level Description of Parallelism
- Compiled into Sequential Code
- Various Programming Styles



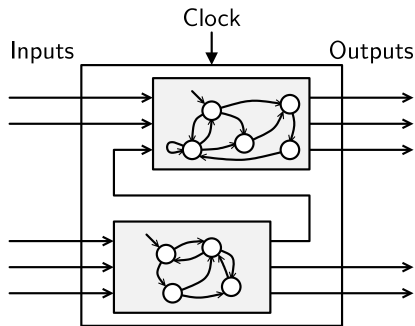
Languages

- LUSTRE, ARGOS, ESTEREL ...
- Easy Integration with C Code
  - Import of C Function
  - Export to C Module

Synchronous Programming in a Nutshell

Program ≈ Synchronous Circuit

- High-level Description of Parallelism
- Compiled into Sequential Code
- Various Programming Styles

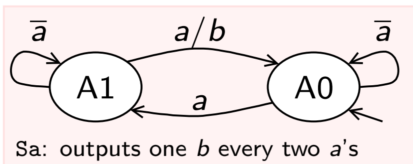


Languages

- LUSTRE, ARGOS, ESTEREL . . .
- Easy Integration with C Code
  - Import of C Function
  - Export to C Module

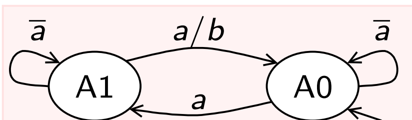
Synchronous Programs as Boolean Mealy Machines

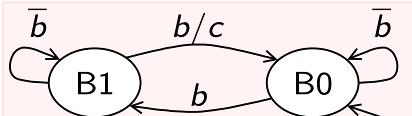
One Boolean Mealy Machine



Synchronous Programs as Boolean Mealy Machines

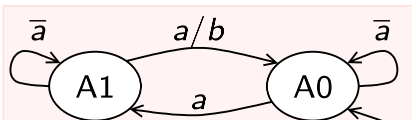
Synchronous Product

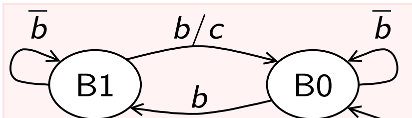


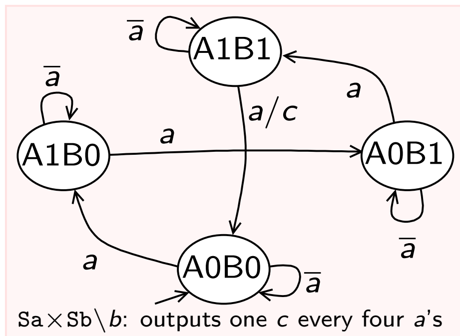


Synchronous Programs as Boolean Mealy Machines

Synchronous Product

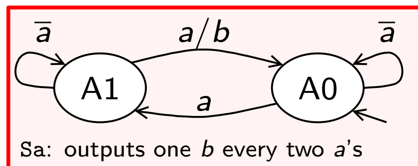


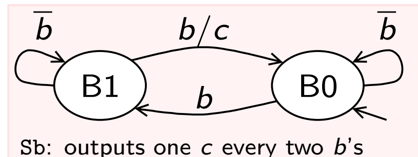


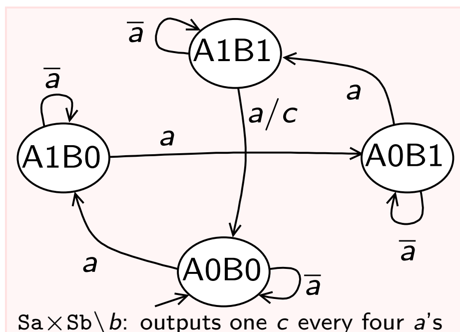


Synchronous Programs as Boolean Mealy Machines

In Lustre



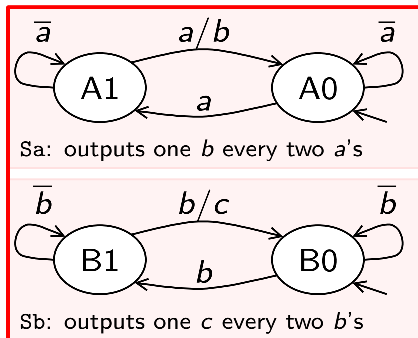


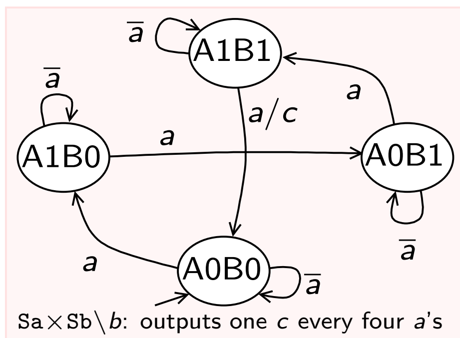



```
node Sa (a: bool) returns (b: bool);
var inA0, inA1, m_A0, A0: bool;
let
  inA1 = not m_A0;      inA0 = m_A0;      b = a and inA1;

  A0 = not a and inA0 or a and inA1; m_A0 = true -> pre A0;

tel;
```

Synchronous Programs as Boolean Mealy Machines

In 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;
```

Synchronous Programs as C Code

The Compilation Produces a Reactive Kernel in C

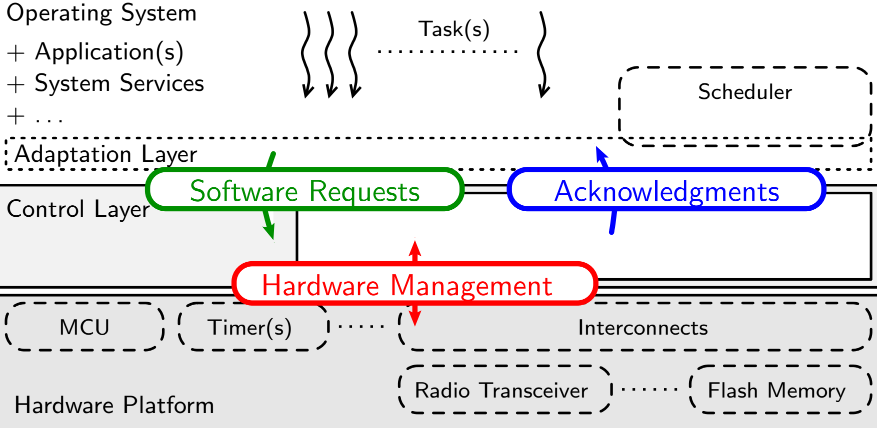
```
bool M1, M2, INIT;              // state variables
void init () { INIT = 1; } // initialization

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;
}
```

- One Call to `run_step()` = One Transition in the Product

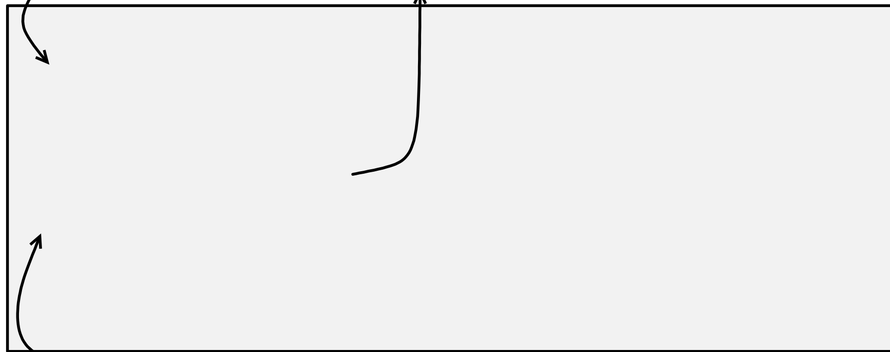
Outline

- Context
- Synchronous Programming in a Nutshell
- **Proposal**
- Evaluation
- Summary

Principles of the Solution

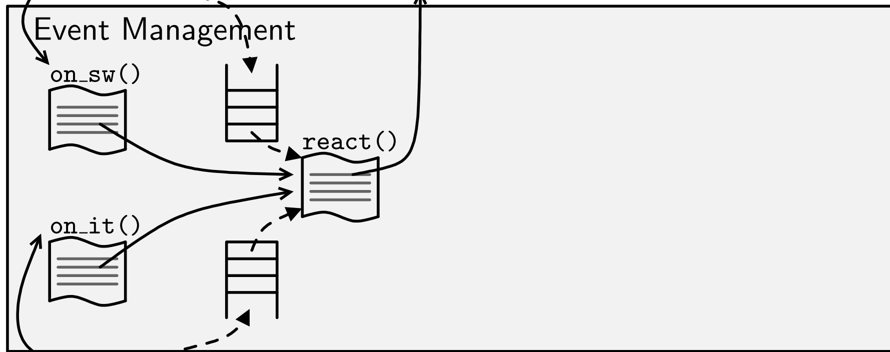


Architecture: Control Layer



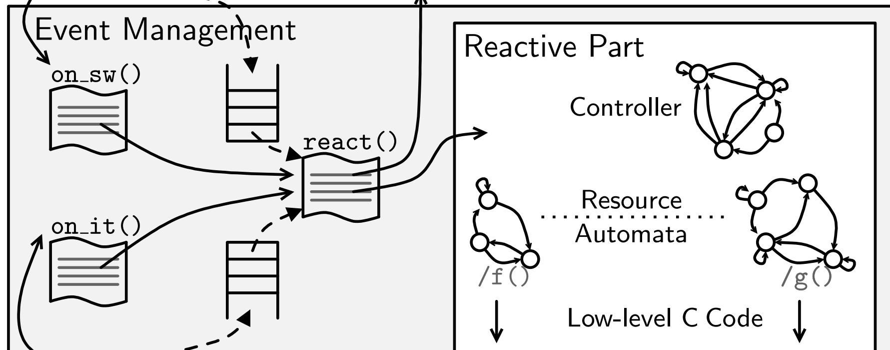
“function call”   push / pop request

Architecture: Control Layer



`react()`: Combines Software and Hardware Requests to be Submitted to the Reactive Part

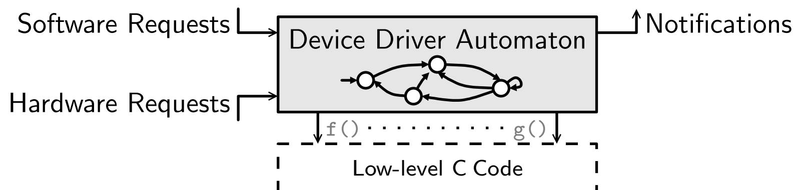
Architecture: Control Layer



`react()`: Combines Software and Hardware Requests to be Submitted to the Reactive Part

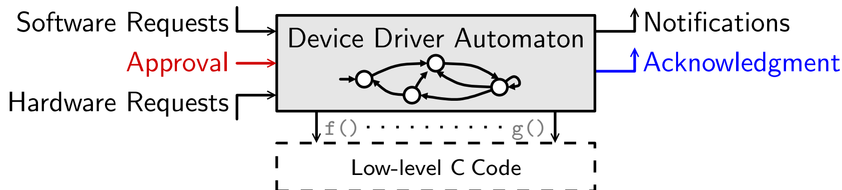
Automata Involved

One Driver Automaton per Device



Automata Involved

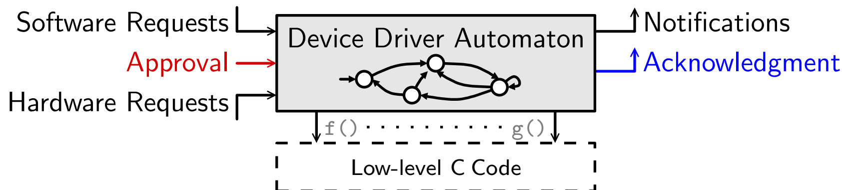
One Driver Automaton per Device



- Can be made Controllable

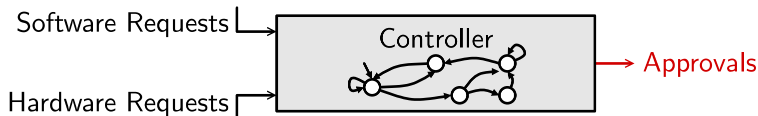
Automata Involved

One Driver Automaton per Device



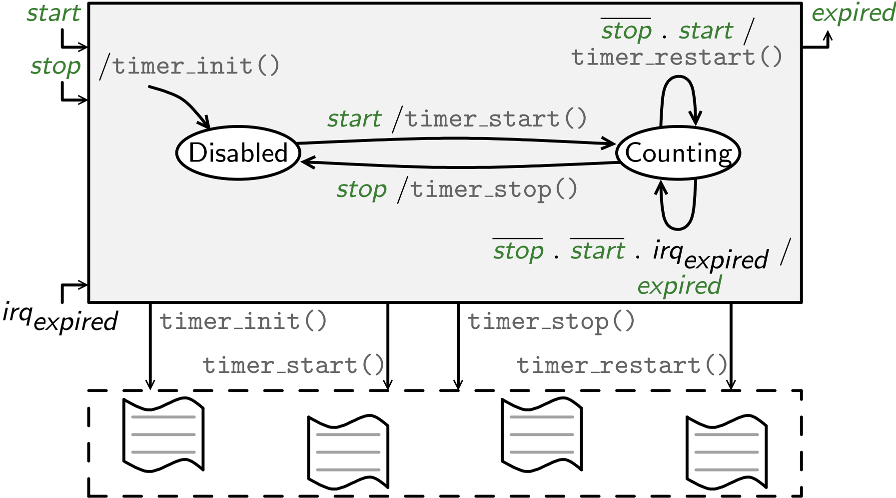
- Can be made Controllable

One Controller Makes Decisions



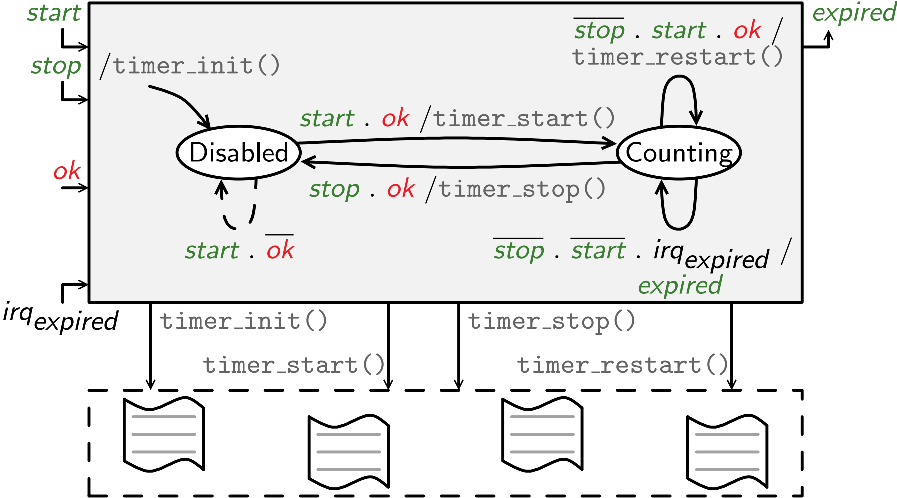
Example of Uncontrollable Driver Automaton

Timer



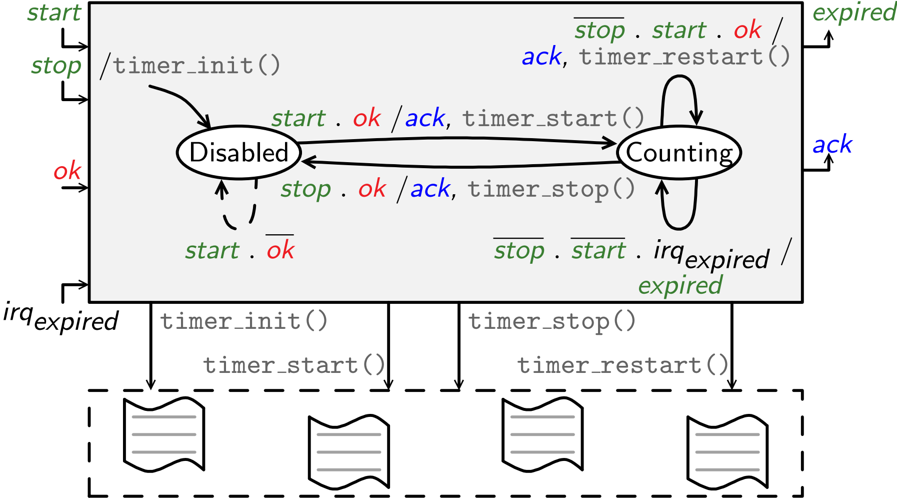
Example of Controllable Driver Automaton

Timer



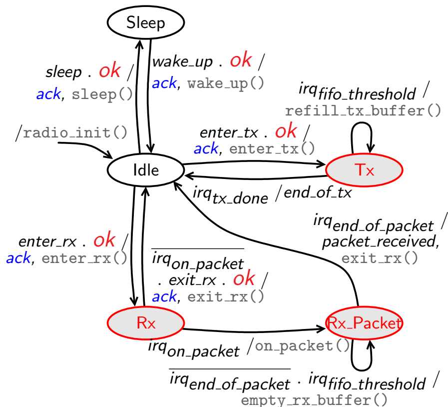
Example of Controllable Driver Automaton

Timer

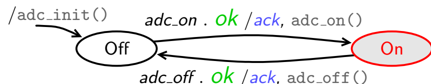


Exclusion of Energy-greedy States: Example

Radio Transceiver

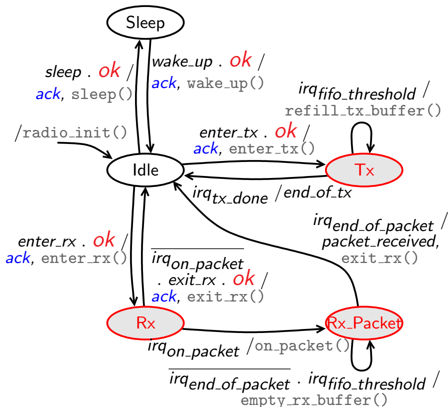


ADC

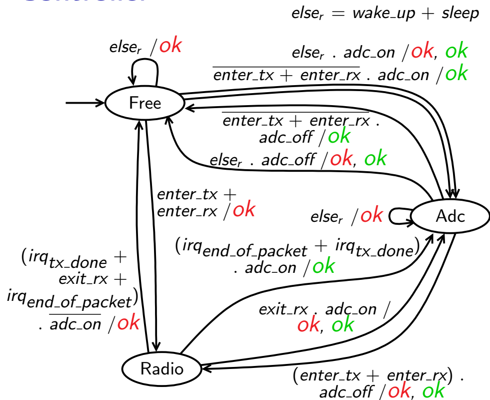


Exclusion of Energy-greedy States: Example

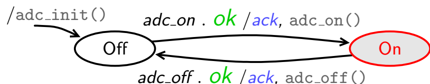
Radio Transceiver



Controller



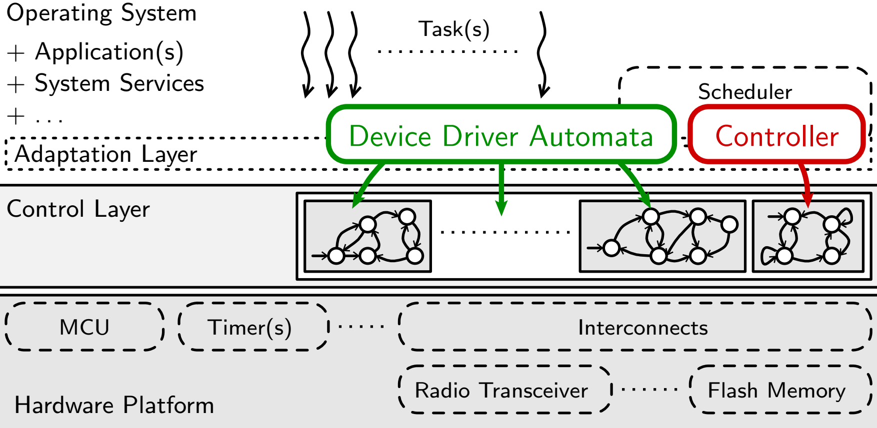
ADC



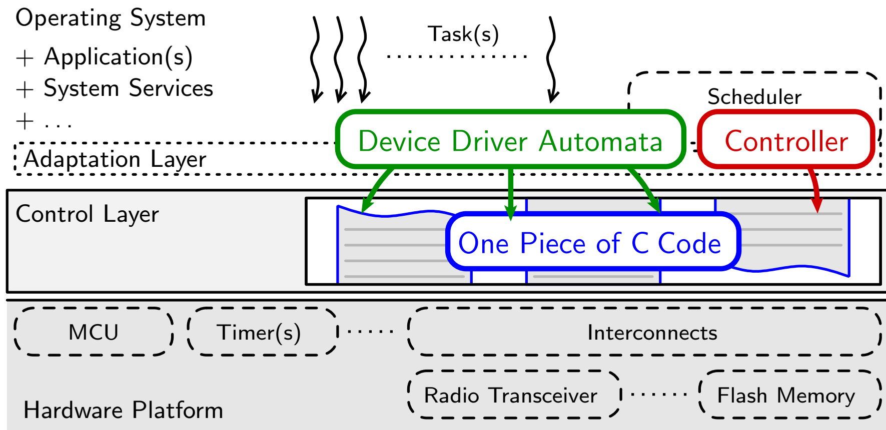
Guarantee:

- States Tx×On, Rx×On and Rx_Packet×On are Unreachable

Putting it All Together



Putting it All Together



Outline

- Context
- Synchronous Programming in a Nutshell
- Proposal
- **Evaluation**
- Summary

Evaluation

Proof-of-Concept Implementation

- Targeting Wireless Sensor Network Nodes (Wsn430)
- Device Drivers & Controller Encoded in ARGOS ($\leadsto$ LUSTRE)
    - $\sim$ 10 Device Drivers
- Adapted Operating Systems:
    - Home-made Multithreaded
    - CONTIKI

Evaluation

Proof-of-Concept Implementation

- Targeting Wireless Sensor Network Nodes (Wsn430)
- Device Drivers & Controller Encoded in ARGOS ($\rightsquigarrow$ LUSTRE)
  - $\sim$ 10 Device Drivers
- Adapted Operating Systems:
  - Home-made Multithreaded
  - CONTIKI

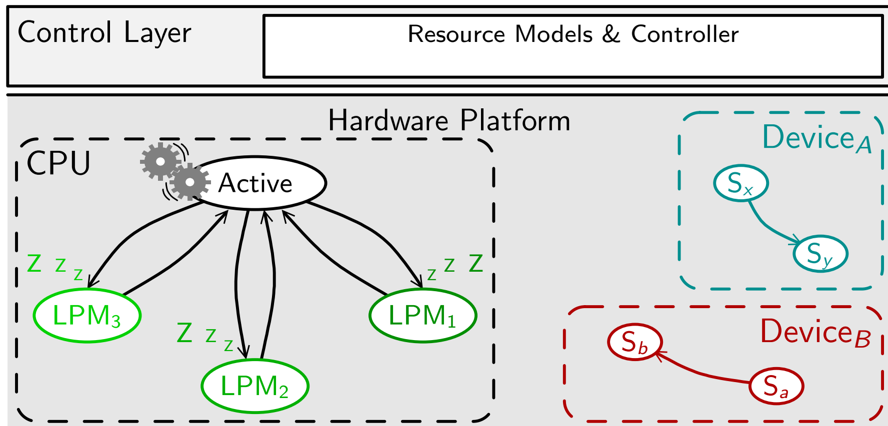
Results

- Extra Memory Footprint: 1.5 to 2.5 KB
- Timing Overhead:

	TINYOS (ICEM) with n Arbiters & m Power Managers	One Execution of Our Reactive Kernel
CPU Cycles	$\lesssim 350n + 400m$	$\approx 1,600$

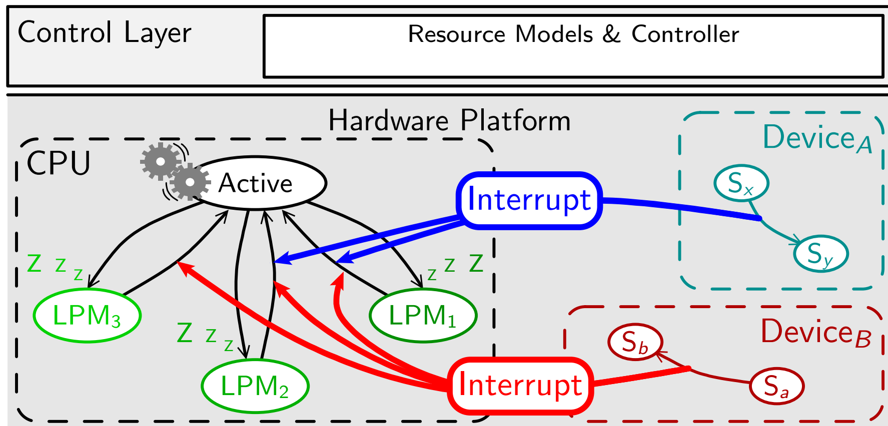
Other Controllable Properties

Selecting the Best Low-Power Mode



Other Controllable Properties

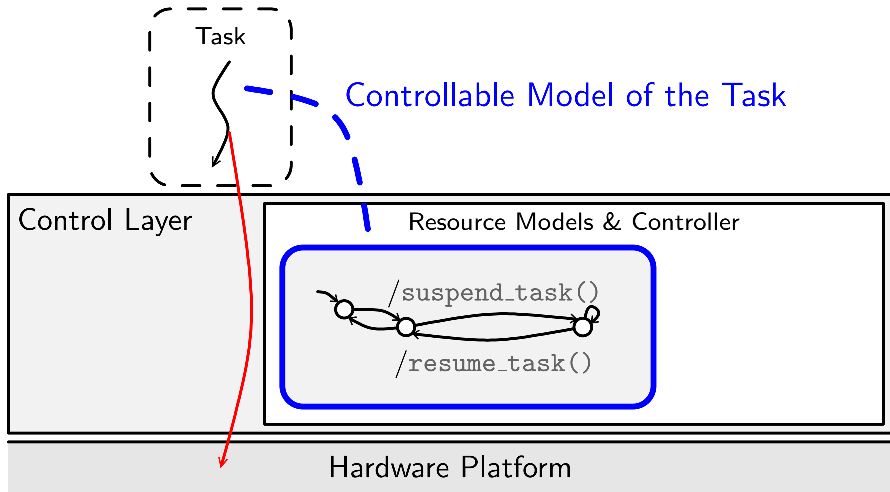
Selecting the Best Low-Power Mode



- Wake-up Guarantee (*e.g.*, State LPM$_3$ Should not be Reachable if Device$_B$ is not in State S$_a$)

Other Controllable Properties *(cont'd)*

Allowing Direct Access to the Devices



Outline

- Context
- Synchronous Programming in a Nutshell
- Proposal
- Evaluation
- **Summary**

Summary

Global Resource Control

- Synchronous Programming... in Wireless Sensor Networks!
- Para-Virtualizaton Concept $\rightsquigarrow$ Flexible Framework
- Many Possible Extensions $\rightsquigarrow$ Powerful

Evaluation

- Proof-of-Concept Implementation
- Practicable
- Device Drivers Revealed "*easier*" to Develop

Summary

Global Resource Control

- Synchronous Programming... in Wireless Sensor Networks!
- Para-Virtualizaton Concept $\leadsto$ Flexible Framework
- Many Possible Extensions $\leadsto$ Powerful

Evaluation

- Proof-of-Concept Implementation
- Practicable
- Device Drivers Revealed "*easier*" to Develop

Prospects

- Efficiency to Reduce Power Consumption? (in the Senslab Testbed)
- Automation: Using Controller Synthesis (by using SIGALI)

Thank You!

Outline

- **Adaptation Layer**
- **Example Execution**

Architecture: 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 Outputs from the Control Layer
  - Notifications (Hardware Events, Acknowledgments)

```
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)

Outline

- Adaptation Layer
- **Example Execution**

Example Execution