Peripheral State Persistence for Transiently Powered Systems

Gautier Berthou, Tristan Delizy, Kevin Marquet, Tanguy Risset, Guillaume Salagnac

Citi Lab, INSA Lyon, France

IoENT Workshop, june 7th 2017
Context: *Transiently Powered Systems*

Internet of Tiny Things
- Internet of Things ➤ networked embedded systems
- no battery ➤ must harvest power from the environment

- smart cards
- RFID tags
- wearable sensors

➤ wearable computing, home automation, environment monitoring, parking assistance, supply chain control...
Transient power = frequent power failures
SW baseline: bare-metal application programming

Program=app+libs+drivers

ISR  deviceA_interrupt()
{ ... }
ISR  deviceB_interrupt()
{ ... }

void main(void){
    hardware_init();
    ___enable_interrupts();
    hardware_access_1();
    compute_step_1();
    hardware_access_2();
    compute_step_2();
    ...
}

Application must run to completion within one “lifecycle”

Software architecture
- no OS support
- main() + interrupt routines + hardware accesses
Problem statement

Industrial Approach:
- Application software must run to completion in one lifecycle
- SW and HW are codesigned: one platform per application

How to run arbitrary code despite frequent, unexpected reboots?

Academic approach:
- Spread execution across multiple lifecycles
**State of the art: program checkpointing**

**Program Checkpointing:**
- **Anticipate** power failures
- Save program state to a **non-volatile** memory
- **Restore state** on next boot

▶ Idea: add “OS” code to hide checkpointing to the application

---

**Diagram**

- **Energy Harvester + buffer**
- **Power in**
- **Power failures detection**
- **CPU**
- **RAM**
- **NVM**

---

**Graph**

- **Energy**
- **Vboot**
- **Vsave**
- **Vdeath**
- **Off time**
- **Lifecycles**
- **Time**

---

T. Delizy 6/21
Checkpointing for Transiently Powered Systems

[Ransford et al '13]

CPU

RAM

NOR Flash

[Jayakumar et al '14]

CPU

FeRAM

[Bhatti & Mottola '16]

CPU

RAM

FC

[NAND Flash]

[Balsamo et al '15, '16]

CPU

RAM

FeRAM

[Lucia & Ransford '15]

CPU

FeRAM

[Bhatti & Mottola '16]

CPU

RAM

NAND Flash

[Ait Aoudia et al '14] (previous work)
Outline

Introduction: Context and State of the Art

Peripheral State Persistence
  Peripheral State: Volatility Problem
  Peripheral Access: Atomicity Problem

Experimental Results

Conclusion and Perspectives
This paper: Making peripherals persistent, too

Non trivial initialization
- timing, polling, ordering constraints

Non trivial access
- not mapped in memory

Most peripherals don’t support "resuming"

Program checkpointing is not enough
The Peripheral State Volatility Problem

Application code

```c
void main(void){
    sensor_init();
    rf_init(myconfig);

    for(;;){
        v = sensor_read();
        rf_send(v);
        ...
    }
}
```

Restoring memory content will not restore device state

T. Delizy 10/21
Our approach: distinct roles for OS and drivers

Each driver:
- Adds a `restore()` function
- `init()` + transitions to saved state
- Put its variables into a device context

Description of a “restore()-able” state

Operating System:
- Persists device context
- Calls every `restore()` functions
- Persists application state
The Peripheral Access Atomicity Problem

In most cases, resuming execution in the middle of a hardware access does not make sense.
Our approach: make driver calls atomic

encapsulate driver functions into OS wrappers.

On wrapper **entry**:  
- save arguments + function called  
- switch to volatile stack

On wrapper **exit**:  
- save device contexts  
- clear arguments  
- switch back to main stack

Interrupted driver calls are **retried** and not just **resumed**.
Our approach: make driver calls atomic

capsulate driver functions into OS wrappers.

On wrapper entry:
- save arguments + function called
- switch to volatile stack

On wrapper exit:
- save device contexts
- clear arguments
- switch back to main stack

Interrupted driver calls are retried and not just resumed.
Outline

Introduction: Context and State of the Art

Peripheral State Persistence
  Peripheral State: Volatility Problem
  Peripheral Access: Atomicity Problem

Experimental Results

Conclusion and Perspectives
Sytare Evaluation Setup

- Energy source
- Harvester
- Energy buffer
- Power Manager
- Microcontroller
- Sensors
- Radio

- μcontroller: 16-bit CPU 24MHz, 1kB SRAM, 15kB FeRAM
- RF transiever: 2.4 GHz transciever, 64B packets
### Sense and send example application

#### Original Application

```c
void main(void)
{
    sensor_init();
    rf_init(myconfig);

    for(;;){
        v = sensor_read();
        compute();
        rf_send(v);
        ...
    }
}
```

#### Adapted Application

```c
void main(void)
{
    syt_sensor_init();
    syt_rf_init(myconfig);

    for(;;){
        v = syt_sensor_read();
        compute();
        syt_rf_send(v);
        ...
    }
}
```
Evaluation methodology

Experimental setup
- Lifecycle: ON for a duration T, then OFF (and then repeat)
- Measure efficiency for various values of T

Performance metrics
- Duration of shortest usable lifecycle
- Execution temporal efficiency w.r.t. bare-metal baseline
Results for Sense and Send scenario

Figure: WSN performance over lifecycle duration (milliseconds)
Results: Driver calls overhead

- led toggle
  - Wrapper entry: 3.4µs
  - Driver function: 1.6µs
  - Wrapper exit: 17.8µs
  - Overhead x 14.25

- ADC read
  - Wrapper entry: 2.4µs
  - Driver function: 75.2µs
  - Wrapper exit: 17.6µs
  - Overhead x 1.27

- radio sleep
  - Wrapper entry: 2.4µs
  - Driver function: 23µs
  - Wrapper exit: 29.2µs
  - Overhead x 2.37

- radio wake up
  - Wrapper entry: 2.4µs
  - Driver function: 428µs
  - Wrapper exit: 31.4µs
  - Overhead x 1.08

- radio send
  - Wrapper entry: 2.4µs
  - Driver function: 3.4ms
  - Wrapper exit: 20.6µs
  - Overhead x 1.01
Outline

Introduction: Context and State of the Art

Peripheral State Persistence
  Peripheral State: Volatility Problem
  Peripheral Access: Atomicity Problem

Experimental Results

Conclusion and Perspectives
Conclusion and Perspectives

Peripheral State Persistence for Transiently Powered Systems

- **Volatility**: device contexts + `restore()` methods
- **Atomicity**: retry VS resume

Project sources available at: [https://gitlab.inria.fr/citi-lab/sytare](https://gitlab.inria.fr/citi-lab/sytare)

**Perspectives**: Look at programming abstractions for transient power

- Expose interrupts to application code
- Add delay-tolerance to driver calls
- Energy based decision making
- Design networking stacks and protocols
- Reduce overhead on driver calls
System boot sequence

Power-up

Hardware boot: 0.75 ms

App state restoration: 45 µs

Device context restoration: 27 µs

Peripheral state restoration: 1.17 ms

Next checkpoint initialization: 30 µs

Application execution
The ST23ZL48 product is a serial access microcontroller specially designed for secure smartcard applications. It is based on an enhanced STMicroelectronics 8/16-bit CPU core offering 16 Mbytes of linear addressing space. It is manufactured using an advanced highly reliable ST CMOS EEPROM technology. Moreover, an ISO 7816-3 EMV-compliant asynchronous receiver transmitter (IART) communication peripheral is available.