Debugging Intermittent Systems

Brief overview of the current debugger landscape

January 24, 2024

Kevin Kollenda

Friedrich-Alexander-Universität Erlangen-Nürnberg
Motivation

Sales and collection of portable batteries and accumulators


Values for Waste collected. Bars in red not represent available data.

Source of data: Eurostat (online data code: env_waspb)
Last update 03/01/2024 23:00
Table of Contents

1. Intermittent Systems
2. Debugging Challenges
3. Intermittent Systems Debugging
4. Existing Solutions & Further Research
5. Conclusion
Intermittent Systems
Intermittent Systems

- Removal of batteries in favor of (super)-capacitors
Intermittent Systems

- Removal of batteries in favor of (super)-capacitors
- Harvest energy from external sources (solar, thermal, ...)

Kevin Kollenda  Debugging Intermittent Systems
Intermittent Systems

- Removal of batteries in favor of (super)-capacitors
- Harvest energy from external sources (solar, thermal, ...)
- Execution is frequently interrupted due to power loss
Intermittent Systems

- Removal of batteries in favor of (super)-capacitors
- Harvest energy from external sources (solar, thermal, ...)
- Execution is frequently interrupted due to power loss
How can we ensure reliable program execution considering the rapidly changing energy inputs?
Reliable Execution

- Checkpointing
  - Save volatile state to non-volatile memory
  - Restore from checkpoint after power loss
Reliable Execution

- Checkpointing
  - Save volatile state to non-volatile memory
  - Restore from checkpoint after power loss

- Task-Based Programming
  - Program is divided into *tasks*
  - Tasks are only run when there is enough energy available
Reliable Execution

- Checkpointing
  - Save volatile state to non-volatile memory
  - Restore from checkpoint after power loss
- Task-Based Programming
  - Program is divided into tasks
  - Tasks are only run when there is enough energy available
- Non-Volatile Systems
  - Conventional DRAM can be replaced by NVRAM
  - Non-volatile microarchitectures for processors
Volatile State Restoration

**Volatile System State**

- Processor State
  - program counter := 0x42

- Memory
  - 0x00 0x01 0x02 0x03
  - 0x04 0x05 0x06 0x07
  - 0x08 0x09 0x0A 0x0B
  - 0x0C 0x0D 0x0E 0x0F

- Processor State
  - program counter := 0x42

- Memory
  - 0x00 0x01 0x02 0x03
  - 0x04 0x05 0x06
  - 0xEE 0xF1 0x32 0x45 0xDA
  - 0x27 0xCF 0x01 0x00

**Peripheral State Restoration**

```
1 sensor = InitializeSensor();
2 Calibrate(sensor);
3 while (data = Read(sensor)) {
4   Checkpoint();
5   // Power failure occurs
6   Transmit(data);
7 }
```
**Volatile State Restoration**

**Processor State**
- program counter := 0x42

**Memory**
- 0x00 0x01 0x02 0x03
- 0x04 0x05 0x06 0x07
- 0x08 0x09 0x0A 0x0B
- 0x0C 0x0D 0x0E 0x0F

**Peripheral State Restoration**

```plaintext
sensor = InitializeSensor();
Calibrate(sensor);
while (data = Read(sensor)) {
    Checkpoint();
    // <Power failure occurs>
    Transmit(data);
}
```

Kevin Kollenda
Debugging Challenges
Common debugging methods *increase* the system’s power draw:
Energy Behaviour

Common debugging methods *increase* the system’s power draw:

- Toggling an LED upon reaching a certain line of code
Common debugging methods *increase* the system’s power draw:

- Toggling an LED upon reaching a certain line of code
- Tracing program flow using `printf`
Common debugging methods increase the system’s power draw:

- Toggling an LED upon reaching a certain line of code
- Tracing program flow using printf
- Streaming log output to external devices (serial, I²C, …)
Common debugging methods *increase* the system’s power draw:

- Toggling an LED upon reaching a certain line of code
- Tracing program flow using `printf`
- Streaming log output to external devices (serial, I²C, …)
- Assertions for simple invariants and complex data structures
Common debugging methods *increase* the system’s power draw:

- Toggling an LED upon reaching a certain line of code
- Tracing program flow using `printf`
- Streaming log output to external devices (serial, I²C, ...)
- Assertions for simple invariants and complex data structures

Established embedded system debuggers do not account for this and require that the device under test is *continuously* powered.
Energy Behaviour

Common debugging methods increase the system’s power draw:

- Toggling an LED upon reaching a certain line of code
- Tracing program flow using printf
- Streaming log output to external devices (serial, I²C, ...)
- Assertions for simple invariants and complex data structures

Established embedded system debuggers do not account for this and require that the device under test is continuously powered.

⇒ Intermittent systems require purpose-built energy-aware debuggers
Software-Based Debugger Issues

Snippet (a)

```c
Checkpoint();
total = NVM_Load();
for i < N {
    total += Sense();
    NVM_Store(total);
}
// i gets saved
Checkpoint();
```

Software-based debuggers can alter the program's behaviour!

Kevin Kollenda

Debugging Intermittent Systems
Software-Based Debugger Issues

Snippet (a)

```c
Checkpoint();
total = NVM_Load();
for i < N {
    total += Sense();
    NVM_Store(total);
}
// i gets saved
Checkpoint();
```

Snippet (b)

```c
Checkpoint();
total = NVM_Load();
for i < N {
    // i gets saved
    DBG_Breakpoint();
    total += Sense();
    NVM_Store(total);
}
```

Software-based debuggers can alter the program's behaviour!

Kevin Kollenda
Software-based debuggers can alter the program’s behaviour!

Kevin Kollenda

Debugging Intermittent Systems
Requirements for an Intermittent System Debugger

Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like
Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like

- Breakpoints with single-stepping
Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like

- Breakpoints with single-stepping
- Output tracing & assertions
Requirements for an Intermittent System Debugger

Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like

- Breakpoints with single-stepping
- Output tracing & assertions
- Reading from and writing to device memory
Requirements for an Intermittent System Debugger

Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like

- Breakpoints with single-stepping
- Output tracing & assertions
- Reading from and writing to device memory

but also be able to *manipulate the device’s energy input* to enable the
Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like

- Breakpoints with single-stepping
- Output tracing & assertions
- Reading from and writing to device memory

but also be able to *manipulate the device’s energy input* to enable the

- Recording of the device’s energy consumption
Requirements for an Intermittent System Debugger

Intermittent system debuggers must not only provide *energy-neutrality* for existing debugging operations, like

- Breakpoints with single-stepping
- Output tracing & assertions
- Reading from and writing to device memory

but also be able to *manipulate the device’s energy input* to enable the

- Recording of the device’s energy consumption
- Manual injection of power failures
Requirements for an Intermittent System Debugger

Intermittent system debuggers must not only provide energy-neutrality for existing debugging operations, like

- Breakpoints with single-stepping
- Output tracing & assertions
- Reading from and writing to device memory

but also be able to manipulate the device’s energy input to enable the

- Recording of the device’s energy consumption
- Manual injection of power failures
- Replay of previously captured energy traces
Intermittent Systems Debugging
**Full Energy Management**

Full energy emulation:

- Replaces device’s power supply
- Enables energy trace replay
- Simulated components

Partial energy emulation:

- Hooks into existing circuitry
- Closer to real-world conditions
Full energy emulation:
- Replaces device’s power supply
**Full Energy Emulation Comparison**

**Full Energy Management**

- **Energy Emulator** → **Microcontroller**

**Full energy emulation:**
- Replaces device’s power supply
- Enables energy trace replay
**Full Energy Emulation Comparison**

**Full Energy Management**

Energy Emulator ➔ Microcontroller

**Full energy emulation:**
- Replaces device’s power supply
- Enables energy trace replay
- Simulated components
**Full energy emulation:**
- Replaces device’s power supply
- Enables energy trace replay
- Simulated components

**Partial energy emulation:**
- Hooks into existing circuitry
- Closer to real-world conditions
Energy Emulation Comparison

Full Energy Emulator: Microcontroller

Full Energy Management:
- Replaces device’s power supply
- Enables energy trace replay
- Simulated components

Partial Energy Emulator: Power Supply

Partial Energy Management:
- Hooks into existing circuitry

Kevin Kollenda
Debugging Intermittent Systems
Energy Emulation Comparison

**Full Energy Management**
- Energy Emulator
- Microcontroller

**Partial Energy Management**
- Energy Emulator
- Power Supply
- Microcontroller

**Full energy emulation:**
- Replaces device’s power supply
- Enables energy trace replay
- Simulated components

**Partial energy emulation:**
- Hooks into existing circuitry
- Closer to real-world conditions
Software-based debuggers:
**Software**-based debuggers:

- Offer software library
Software-based debuggers:

- Offer software library
- Impact program behaviour
Software-based debuggers:

- Offer software library
- Impact program behaviour
- Low barrier of entry
**Software**-based debuggers:
- Offer software library
- Impact program behaviour
- Low barrier of entry

**Hardware**-based debuggers:
- Connect to processor’s in-built debugging circuitry
- Debugging tasks are offloaded
- Increased energy consumption

Regardless of the debugger’s kind:
- Standalone or built upon existing debuggers (i.e. GDB)
- Energy management interface
Debugger Design

**Software**-based debuggers:
- Offer software library
- Impact program behaviour
- Low barrier of entry

**Hardware**-based debuggers:
- Connect to processor’s in-built debugging circuitry

Regardless of the debugger’s kind:
- Standalone or built upon existing debuggers (i.e. GDB)
- Energy management interface
**Software**-based debuggers:
- Offer software library
- Impact program behaviour
- Low barrier of entry

**Hardware**-based debuggers:
- Connect to processor’s in-built debugging circuitry
- Debugging tasks are offloaded
Debugger Design

Software-based debuggers:
- Offer software library
- Impact program behaviour
- Low barrier of entry

Hardware-based debuggers:
- Connect to processor’s in-built debugging circuitry
- Debugging tasks are offloaded
- Increased energy consumption

Regardless of the debugger’s kind:
- Standalone or built upon existing debuggers (i.e. GDB)
- Energy management interface
Debugger Design

**Software**-based debuggers:
- Offer software library
- Impact program behaviour
- Low barrier of entry

**Hardware**-based debuggers:
- Connect to processor’s in-built debugging circuitry
- Debugging tasks are offloaded
- Increased energy consumption

Regardless of the debugger’s kind:
- Standalone or built upon existing debuggers (i.e. GDB)
- Energy management interface
Intrusive debugging *always* consumes additional energy.
Intrusive debugging always consumes additional energy.

*How can we achieve realistic conditions during debugging?*
Energy-Neutrality

Intrusive debugging *always* consumes additional energy.

*How can we achieve realistic conditions during debugging?*

**Energy-Guards [1]**

Neutralize the energy impact of certain actions or code snippets.
Intrusive debugging always consumes additional energy.

How can we achieve realistic conditions during debugging?

**Energy-Guards [1]**

Neutralize the energy impact of certain actions or code snippets.

In practice:
Bringing it all together

- Energy Emulator
- Device under Test
- Debugger
- Energy Harvesting
- Sensors
- Wireless Modules
- Displays

→ Mask energy footprint of complex assertions
→ Pause energy consumption during breakpoints
⇒ Recreate previously recorded energy environments

Debug intermittent systems like regular embedded systems
Bringing it all together

→ Mask energy footprint of complex assertions
Bringing it all together

- Mask energy footprint of complex assertions
- Pause energy consumption during breakpoints
→ Mask energy footprint of complex assertions
→ Pause energy consumption during breakpoints
→ Recreate previously recorded energy environments
→ Mask energy footprint of complex assertions
→ Pause energy consumption during breakpoints
→ Recreate previously recorded energy environments
⇒ **Debug intermittent systems like regular embedded systems**
Existing Solutions & Further Research
Existing Debuggers

Energy-Interference-Free Debugger (EDB)

2016

Kevin Kollenda
Debugging Intermittent Systems
Existing Debuggers

Energy-Interference-Free Debugger (EDB) 2016

- Hooks into existing energy circuit
Energy-Interference-Free Debugger (EDB)

- Hooks into existing energy circuit
- Provides software library for debugging

Kevin Kollenda
Debugging Intermittent Systems
Existing Debuggers

Energy-Interference-Free Debugger (EDB)

- Hooks into existing energy circuit
- Provides software library for debugging
- First available intermittent system debugger

[1]
Existing Debuggers

Debugger for Intermittently-Powered Systems (DIPS)
Debugger for Intermittently-Powered Systems (DIPS)

- Fully manipulates the device’s energy input
Existing Debuggers

**Debugger for Intermittently-Powered Systems (DIPS)**

- Fully manipulates the device’s energy input
- Utilizes in-built debugging circuitry

2022
Debugger for Intermittently-Powered Systems (DIPS)

- Fully manipulates the device’s energy input
- Utilizes in-built debugging circuitry
- Scriptable interface for automatic testing
## Existing Debuggers

<table>
<thead>
<tr>
<th>Feature</th>
<th>EDB</th>
<th>DIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDB-Based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy-neutral Debugging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breakpoints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automated Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Stepping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported Architectures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Kevin Kollenda

Debugging Intermittent Systems

16
## Existing Debuggers

<table>
<thead>
<tr>
<th>Feature</th>
<th>EDB</th>
<th>DIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger Design</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Energy Management</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>GDB-Based</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Energy-neutral Debugging</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Breakpoints</td>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Automated Testing</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Single Stepping</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Supported Architectures</td>
<td>MSP430</td>
<td></td>
</tr>
</tbody>
</table>
## Existing Debuggers

<table>
<thead>
<tr>
<th>Feature</th>
<th>EDB</th>
<th>DIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger Design</td>
<td>Software</td>
<td>Hardware</td>
</tr>
<tr>
<td>Energy Management</td>
<td>Partial</td>
<td>Full</td>
</tr>
<tr>
<td>GDB-Based</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Energy-neutral Debugging</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>Software</td>
<td>Software &amp; Hardware</td>
</tr>
<tr>
<td>Automated Testing</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Single Stepping</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Supported Architectures</td>
<td>MSP430</td>
<td>ARM</td>
</tr>
</tbody>
</table>
Future Research

- Support additional architectures
Future Research

- Support additional architectures
- Improve energy emulation hardware
Future Research

- Support additional architectures
- Improve energy emulation hardware
- Incorporate existing testing frameworks (i.e. fuzzing, ...)

Progress in non-volatile technologies lessen impact of intermittency
Future Research

- Support additional architectures
- Improve energy emulation hardware
- Incorporate existing testing frameworks (i.e. fuzzing, ...)
- Progress in non-volatile technologies lessen impact of intermittency
Conclusion
Intermittent systems pose unique challenges to existing debuggers
Intermittent systems pose unique challenges to existing debuggers
- Energy-neutral debugging via energy-guards
Intermittent systems pose unique challenges to existing debuggers

- Energy-neutral debugging via energy-guards
- Real-world energy conditions provided by energy emulator

Bright future for intermittent devices

⇒ Increase IoT sustainability by reducing the need for batteries
Summary

- Intermittent systems pose unique challenges to existing debuggers
  - Energy-neutral debugging via energy-guards
  - Real-world energy conditions provided by energy emulator
- Requires tight integration between energy emulator and debugger
Summary

- Intermittent systems pose unique challenges to existing debuggers
  - Energy-neutral debugging via energy-guards
  - Real-world energy conditions provided by energy emulator
- Requires tight integration between energy emulator and debugger
- Bright future for intermittent devices
Summary

- Intermittent systems pose unique challenges to existing debuggers
  - Energy-neutral debugging via energy-guards
  - Real-world energy conditions provided by energy emulator
- Requires tight integration between energy emulator and debugger
- Bright future for intermittent devices

⇒ Increase IoT sustainability by reducing the need for batteries
Questions?
EDB providing assertions with power using energy-guards [1]
Impact of guarded printf calls [1]
DIPS Latencies

<table>
<thead>
<tr>
<th>Device Under Test</th>
<th>$t_{\text{init}}$ (ms)</th>
<th>$t_{\text{rec}}$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRF52 [Arm-M4]</td>
<td>311.1</td>
<td>72.7</td>
</tr>
<tr>
<td>SAM4L8 [Arm-M4]</td>
<td>324.7</td>
<td>75.8</td>
</tr>
<tr>
<td>MKL05Z [Arm-M0+]</td>
<td>309.6</td>
<td>105.8</td>
</tr>
<tr>
<td>STM32F3 [Arm M4]</td>
<td>318.6</td>
<td>68.2</td>
</tr>
<tr>
<td>Apollo 3 [Arm M4]</td>
<td>331.1</td>
<td>95.6</td>
</tr>
</tbody>
</table>

DIPS initial and reconnection latencies [2]


**Ambient energy harvesting nonvolatile processors: From circuit to system.**  