SLOTH on Time: Efficient Hardware-Based Scheduling for Time-Triggered RTOS

Wanja Hofer, Daniel Danner, Rainer Müller, Fabian Scheler, Wolfgang Schröder-Preikschat, Daniel Lohmann

December 7, 2012
Sloth kernels use hardware for OS purposes, and

- are concise (200–500 LoC)
- are small (300–900 bytes)
- are fast (latency speed-up 2x to 170x)
- implement relevant industry standards
OSEK and AUTOSAR OS in 90 Seconds

- Standards developed by the automotive industry
- Families of statically configured RTOS

![Diagram showing OSEK and AUTOSAR OS]

OSEK:
- **event-triggered**
  - BCC1: tasks are run-to-completion
  - ECC1: tasks can block

OSEKtime: strictly time-triggered

AUTOSAR OS:
- **event-triggered**
  - Sleepy Sloth [RTSS ’11]
- **time-triggered**
  - Sloth on Time [RTSS ’12]

Sloth:
- [RTSS ’09]

“Schedule Tables”: time-triggered activation of event-triggered tasks
**SLOTH Recap**

**Main Idea**

Threads are interrupt handlers, synchronous thread activation is IRQ
⇒ Interrupt subsystem does scheduling and dispatching work

---

Can we transfer this idea to time-triggered systems?
1. Support for time-triggered task activation like in OSEKtime and AUTOSAR schedule tables
2. Deadline monitoring: exception when task is still running
3. Integration of an additional event-triggered system
4. Timing protection with task execution budgets (see paper)
5. Synchronization with a global time base (see paper)
**Observation**

Many microcontrollers have an *abundance* of timer cells available (e.g., 256 cells on Infineon TriCore TC1796)

**Implementation Concept**

Instrumentation of timer cells for specific OS purposes:

1. task activation
2. task deadline monitoring
3. task control
4. task budget monitoring
5. clock synchronization
Static Application Configuration:

- roundLength = 1000;
- expiryPoints = {
  100 => Task1,
  200 => Task2,
  600 => Task1
};
- deadlines = {
  450 => Task1,
  350 => Task2,
  950 => Task1
};
- availableTimerCells = {Cell7, ..., Cell12, Cell42};

Cell and IRQ Map:

- Cell7 = 100, Activation, IRQTask1
- Cell8 = 200, Activation, IRQTask2
- Cell9 = 600, Activation, IRQTask1
- Cell10 = 450, Deadline, deadlineViolationHandler
- Cell11 = 350, Deadline, deadlineViolationHandler
- Cell12 = 950, Deadline, deadlineViolationHandler

Analysis and Cell Mapping:

Timer Hardware Description:

TimerArray0 = {
  Cell0 = {
    irqSource => 128,
    isMaster => false,
    controls => {}
  },
  ...
  Cell42 = {
    irqSource => 170,
    isMaster => true,
    controls => {7, ..., 12},
    ...
  },
  ...
};

Cell Initialization Code:

```c
void initCells(void) {
  Cell7.compare = 1000;
  ...
  Cell7.value = 1000 - 100;
  ...
} 
```

Start Dispatcher Code:

```c
void startDispatcher(void) {
  ifndef CONTROLCELLS
    Cell7.enable = 1;
    ...
  #else
    // Control Cell 42 for Cells 7-12
    Cell42.output = 1;
  #endif
}
```

Timer Array Configuration:

```c
TimerArray0 = {
  Cell0 = {
    irqSource => 128,
    isMaster => false,
    controls => {}
  },
  ...
  Cell42 = {
    irqSource => 170,
    isMaster => true,
    controls => {7, ..., 12},
    ...
  },
  ...
};
```

Task Handler Code:

```c
void handlerTask1(void) {
  // Prologue
  savePreemptedContext();
  setCUPrio(execPrio);
  Cell10.reqEnable = 1;
  Cell12.reqEnable = 1;
  userTask1();
  // Epilogue
  Cell10.reqEnable = 0;
  Cell12.reqEnable = 0;
  restorePreemptedContext();
  iret();
}
```

Input/Intermediate/Output:

At run time:

- Task IRQ handler only needs to save context, lowers its priority if applicable, and calls user task
- No further table management and timer reprogramming

Daniel Lohmann The SLOTH on Time Time-Triggered RTOS (December 7, 2012)
Implementation: Static Analysis and Code Generation

**Static Application Configuration:**
- `roundLength = 1000;`
- `expiryPoints = {100 => Task1, 200 => Task2, 600 => Task1};`
- `deadlines = {450 => Task1, 350 => Task2, 950 => Task1};`
- `availableTimerCells = {Cell7, ..., Cell12, Cell42};`

**Cell and IRQ Map:**
- `100 => Cell7 // Activation`
- `200 => Cell8 // Activation`
- `600 => Cell9 // Activation`
- `450 => Cell10 // Deadline`
- `350 => Cell11 // Deadline`
- `950 => Cell12 // Deadline`

**Cell Initialization Code:**
```c
void initCells(void) {
    Cell7.compare = 1000;
    ... Cell7.value = 1000 - 100;
    ...
}
```

**Start Dispatcher Code:**
```c
void startDispatcher(void) {
    #ifndef CONTROLCELLS
    Cell7.enable = 1;
    ...
    #else
    // Control Cell 42 for Cells 7-12
    Cell42.output = 1;
    #endif
}
```

**Timer Hardware Description:**
```c
TimerArray0 = {
    Cell0 = {
        irqSource => 128,
        isMaster => false,
        controls => {};
    ...
    },
    ...
    Cell142 = {
    ...
    }
```

**At run time:**
- Task IRQ handler only needs to save context, lowers its priority if applicable, and calls user task
- No further table management and timer reprogramming

**IRQ Initialization Code:**
```c
void initIRQs(void) {
    Cell7.irqPrio = triggerPrio;
    ...
    Cell7.handler = &handlerTask1;
    ...
    Cell10.handler = &deadlineViolationHandler;
    ...
}
```

**Task Handler Code:**
```c
void handlerTask1(void) {
    // Prologue
    savePreemptedContext();
    setCPUPrio(execPrio);
    Cell10.reqEnable = 1;
    Cell12.reqEnable = 1;
    userTask1();
    // Epilogue
    Cell10.reqEnable = 0;
    Cell12.reqEnable = 0;
    restorePreemptedContext();
    iret();
}
```
Implementation: Task Activation in OSEKtime

CPU Priority

OSEKtime
trigger priority
OSEKtime
execution priority
idle priority

IRQ handler TT1

IRQ handler TT2

Task TT1

Task TT2

Task TT1

Idle Task

Idle Task

\( t_1 \) \( t_2 \) \( t_3 \) \( t_4 \)
Implementation: Task Activation in OSEKtime

⇒ Seamless integration with existing event-triggered SLOTH
Evaluation: Setup

**Evaluation platform:** Infineon TriCore TC1796
- 32-bit RISC μ-Controller, clocked at 50 MHz
  - widely used in the automotive industry (BMW, Audi, ...)
  - IRQ system with 256 priority levels and 181 mem-mapped IRQ sources
- General Purpose Timer Array (GPTA)
  - 256 timer cells, 92 connected IRQ sources
  - cascading cell configuration possible (for control cells)

**Comparison against:** Commercial OSEKtime/AUTOSAR
- Qualitative
  - Implementation
  - Priority obedience
- Quantitative
  - Task activation/termination latency
  - System service latency
Qualitative Evaluation: OSEKtime

Commercial OSEKtime: Spurious IRQs for deadline monitoring (95 cycles each)

Sloth on Time: avoids this by design!

---

Daniel Lohmann  The Sloth on Time Time-Triggered RTOS (December 7, 2012)
Commercial AUTOSAR: **Priority inversion** with time-triggered activation (2,075 cycles each)

**Sloth on Time:** avoids this *by design!*

![Diagram showing priority inversion and Sloth on Time avoiding it](image-url)
Quantitative Evaluation: Task Activation OSEKtime

Sloth on Time vs. commercial OSEKtime

![Bar chart comparing Sloth on Time and OSEKtime for different operations like Dispatch to idle, Dispatch from BT, etc.]

Speed-Up:
- **14** for Dispatch
- **2.7** for Terminate
- **8.6** for Dispatch from idle
- **8.6** for Dispatch from BT
- **4.0** for Dispatch from ET
- **2.7** for Terminate to idle
- **2.7** for Terminate to BT
- **1.3** for Terminate to ET

**purely time-triggered OSEKtime**

**Mixed OSEKtime/OSEK**
**Sloth on Time vs. commercial AUTOSAR**

<table>
<thead>
<tr>
<th>Event</th>
<th>Sloth on Time</th>
<th>AUTOSAR</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatch idle → BT</td>
<td>167.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dispatch BT → BT</td>
<td>171.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dispatch ET → ET</td>
<td>31.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminate BT → idle</td>
<td>27.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminate BT → BT</td>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminate ET → ET</td>
<td>6.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quantitative Evaluation: System Services

Sloth on Time vs. commercial AUTOSAR

<table>
<thead>
<tr>
<th>Function</th>
<th>Sloth on Time</th>
<th>AUTOSAR</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartScheduleTableRel()</td>
<td>10.2</td>
<td>20</td>
<td>108</td>
</tr>
<tr>
<td>StopScheduleTable()</td>
<td>37.6</td>
<td>20</td>
<td>37.6</td>
</tr>
</tbody>
</table>

Cycles

0 200 400 600 800 1000 1200

StartScheduleTableRel()   StopScheduleTable()
**SLOTH on Time: Conclusions**

**Being a Sloth Pays Off!**

- Small and concise kernel
  - 500 LoC
  - 900 bytes ROM
  - 8 bytes RAM
- Excellent real-time characteristics
  - minimal latencies (speed-up 2x to 170x)
  - no unnecessary IRQs
  - no priority inversion
  ⇒ higher schedulability
- 3 RTSS papers
  ⇒ thesis complete :-)

Daniel Lohmann  The SLOTH on Time Time-Triggered RTOS (December 7, 2012) 15
Quantitative Evaluation: Deadline Monitoring

**Sloth on Time vs. commercial OSEKtime**

![Bar chart](image)

- **Dispatch 1 deadline**: Sloth on Time 4.6, OSEKtime 120
- **Dispatch 2 deadlines**: Sloth on Time 3.5, OSEKtime 120
- **Dispatch 3 deadlines**: Sloth on Time 2.7, OSEKtime 120
- **Terminate 1 deadline**: Sloth on Time 1.6, OSEKtime 120
- **Terminate 2 deadlines**: Sloth on Time 1.1, OSEKtime 120
- **Terminate 3 deadlines**: Sloth on Time 0.9, OSEKtime 120
- **Dispatch w/muxing**: Sloth on Time 8.6, OSEKtime 120
- **Terminate w/muxing**: Sloth on Time 1.3, OSEKtime 120

**Speed-Up:**
- Dispatch 1: 4.6
- Dispatch 2: 3.5
- Dispatch 3: 2.7
- Terminate 1: 1.6
- Terminate 2: 1.1
- Terminate 3: 0.9
- Dispatch w/muxing: 8.6
- Terminate w/muxing: 1.3

**One cell per deadline**

**Multiplexed deadlines**
Time-Triggered Activation with Local Timer Cells

Dispatcher Table
(length = 200)

<table>
<thead>
<tr>
<th>Act. Task1</th>
<th>Act. Task2</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset=60</td>
<td>offset=170</td>
</tr>
</tbody>
</table>

0 50 100 150 200

t

1st dispatcher round
2nd dispatcher round

0 50 100 150 200

counter

compare value (LTC00) → 200
initial counter (LTC01) → 100

compare value (LTC04) → 200
initial counter (LTC05) → 100

IRQ Task1
IRQ Task1
 IRQ Task2
IRQ Task2

Daniel Lohmann  The SLOTH on Time Time-Triggered RTOS (December 7, 2012)