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

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

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

December 7, 2012

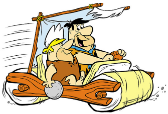
Laziness Is Beautiful: Let the Hardware Do the Work!

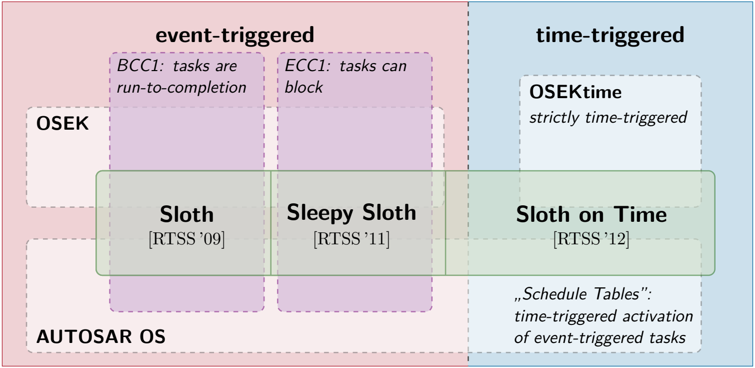
Sloth kernels use hardware for OS purposes, and

- 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

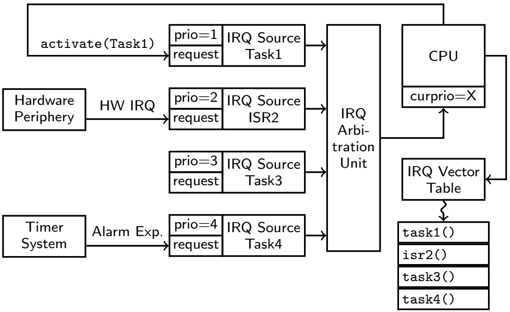




Sloth Recap

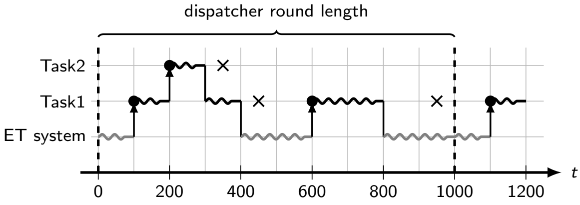
Main Idea

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?

SLOTH on Time: Goals



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)

SLOTH on Time: Idea

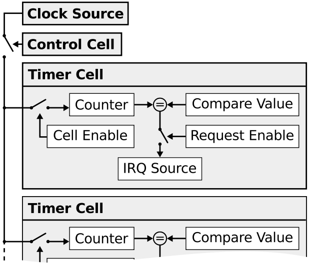
Observation

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



Implementation: Static Analysis and Code Generation

Input

Static Application Configuration:

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

Analysis and Cell Mapping

Input

Timer Hardware Description:

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

Intermediate

Cell and IRQ Map:

```
100 => Cell7  // Activation
200 => Cell8  // Activation
600 => Cell9  // Activation
450 => Cell10 // Deadline
350 => Cell11 // Deadline
950 => Cell12 // Deadline
Cell7 => IRQTask1
Cell8 => IRQTask2
Cell9 => IRQTask1
```

Code Generation

Output

Cell Initialization Code:

```
void initCells(void) {
  Cell7.compare = 1000;
  ...
  Cell7.value = 1000 - 100;
  ...
}
void startDispatcher(void) {
#ifndef CONTROLCELLS
  Cell7.enable = 1;
  ...
#else
  // Control Cell 42 for Cells 7-12
  Cell42.output = 1;
#endif
}
```

Output

IRQ Initialization Code:

```
void initIRQs(void) {
  Cell7.irqPrio =
    triggerPrio;
  ...
  Cell7.handler =
    &handlerTask1;
  ...
  Cell10.handler =
    &deadlineViolationHandler;
  ...
}
```

Output

Task Handler Code:

```
void handlerTask1(void) {
  // Prologue
  savePreemptedContext();
  setCPUPrio(execPrio);
  Cell10.reqEnable = 1;
  Cell12.reqEnable = 1;

  userTask1();

  // Epilogue
  Cell10.reqEnable = 0;
  Cell12.reqEnable = 0;
  restorePreemptedContext();
  iret();
}
```

Implementation: Static Analysis and Code Generation

Input

Static Application Configuration:

```
roundLength = 1000;
expiryPoints = {
  100 => Task1,
  200 => Task2,
  600 => Task1
};
deadlines = {
  450 => Task1,
  350 => Task2
```

Analysis and Cell Mapping

Intermediate

Cell and IRQ Map:

```
100 => Cell7  // Activation
200 => Cell8  // Activation
```

Input

Timer Hardware Description:

```
TimerArray0 = {
  Cell0 = {
    irqSource => 128,
    isMaster  => false,
    controls  => {},
    ...
  },
  ...
  Cell42 = {
```

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

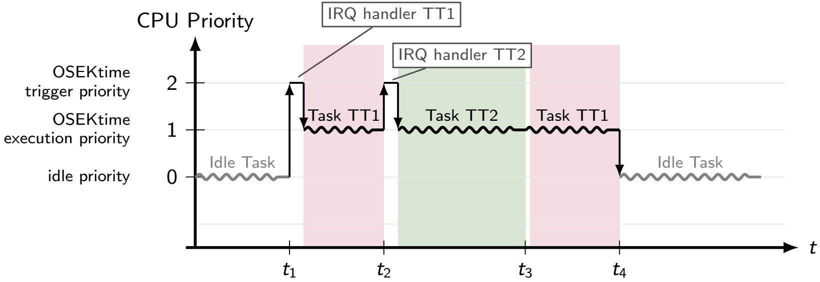
```
  ...
}
void startDispatcher(void) {
#ifndef CONTROLCELLS
  Cell7.enable = 1;
  ...
#else
  // Control Cell 42 for Cells 7-12
  Cell42.output = 1;
#endif
}
```

IRQ Initialization Code:

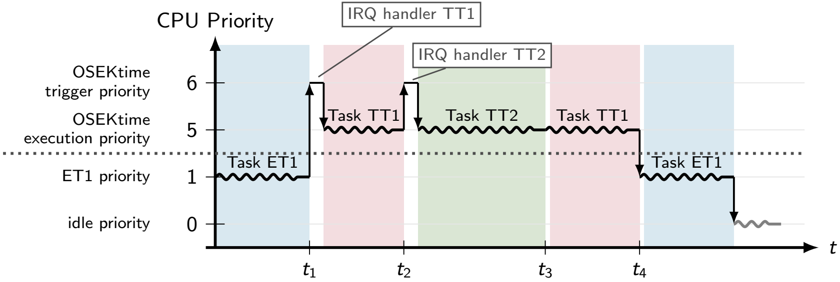
```
void initIRQs(void) {
  Cell7.irqPrio =
    triggerPrio;
  ...
  Cell7.handler =
    &handlerTask1;
  ...
  Cell10.handler =
    &deadlineViolationHandler;
  ...
}
```

```
  Cell10.reqEnable = 1;
  Cell12.reqEnable = 1;

  userTask1();

  // Epilogue
  Cell10.reqEnable = 0;
  Cell12.reqEnable = 0;
  restorePreemptedContext();
  iret();
}
```

Implementation: Task Activation in OSEKtime



Implementation: Task Activation in OSEKtime



⇒ Seamless integration with existing event-triggered SLOTH

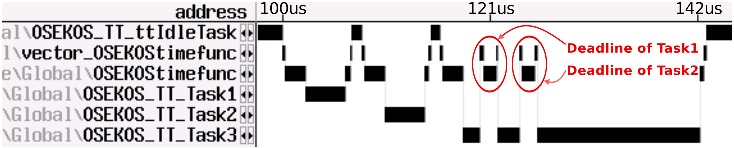
Evaluation: Setup

- **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)

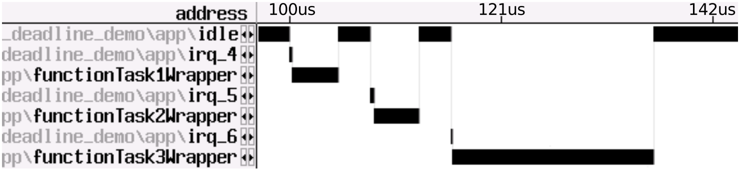
- **Comparison against:** Commercial OSEKtime/AUTOSAR
  - Qualitative
    - Implementation
    - Priority obedience
  - Quantitative
    - Task activation/termination latency
    - System service latency

Qualitative Evaluation: OSEKtime

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

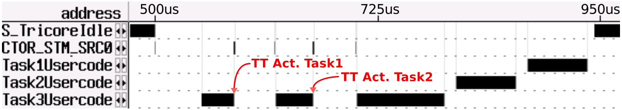


Sloth on Time: avoids this *by design!*

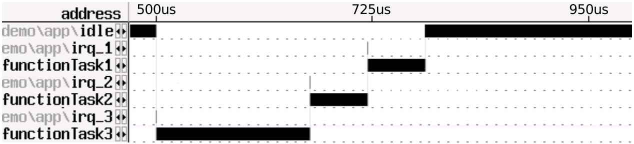


Qualitative Evaluation: AUTOSAR

Commercial AUTOSAR: **Priority inversion** with time-triggered activation (2,075 cycles each)

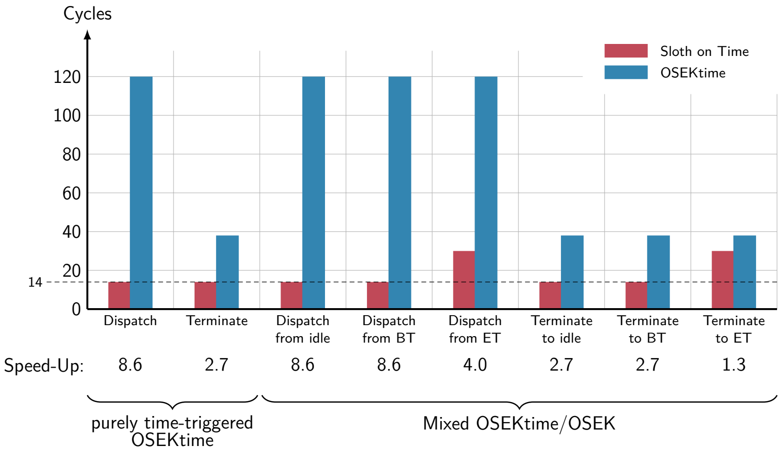


SLOTH on Time: avoids this *by design!*



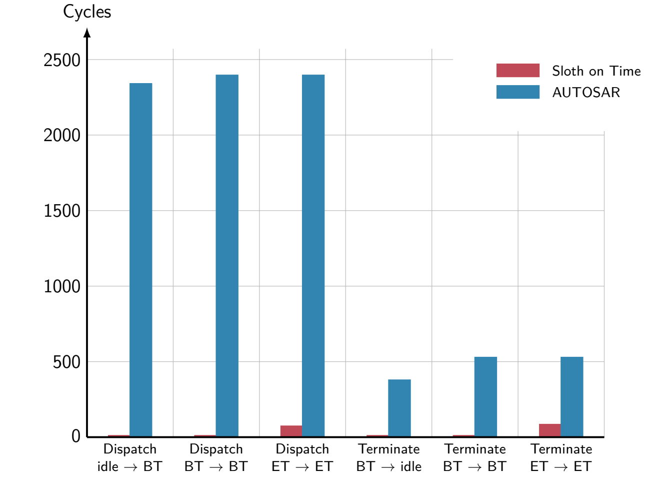
Quantitative Evaluation: Task Activation OSEKtime

Sloth on Time vs. commercial OSEKtime



Quantitative Evaluation: Task Activation AUTOSAR

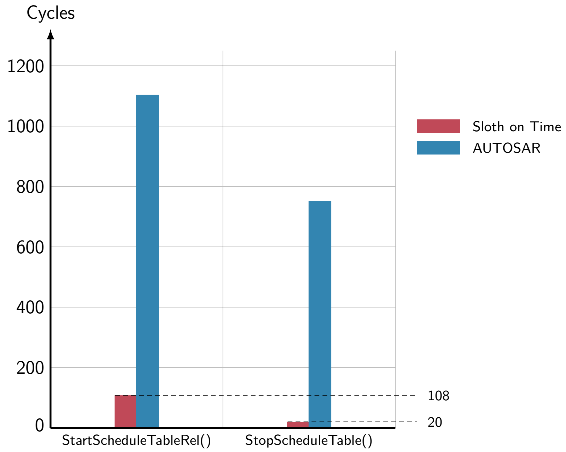
Sloth on Time vs. commercial AUTOSAR



Speed-Up: 167.4 171.4 31.2 27.3 38 6.0

Quantitative Evaluation: System Services

Sloth on Time vs. commercial AUTOSAR



Speed-Up: 10.2 37.6

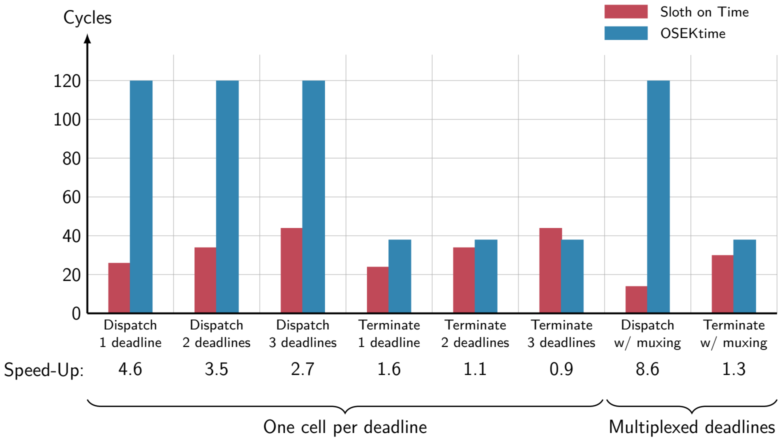
SLOTH on Time: Conclusions

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
  - $\Rightarrow$ higher schedulability
- **3 RTSS papers**
  $\Rightarrow$ thesis complete :-)

Quantitative Evaluation: Deadline Monitoring

Sloth on Time vs. commercial OSEKtime



Time-Triggered Activation with Local Timer Cells