Konfigurierbare Systemsoftware (KSS)

VL 6 – Generative Programming: The SLOTH Approach

Daniel Lohmann

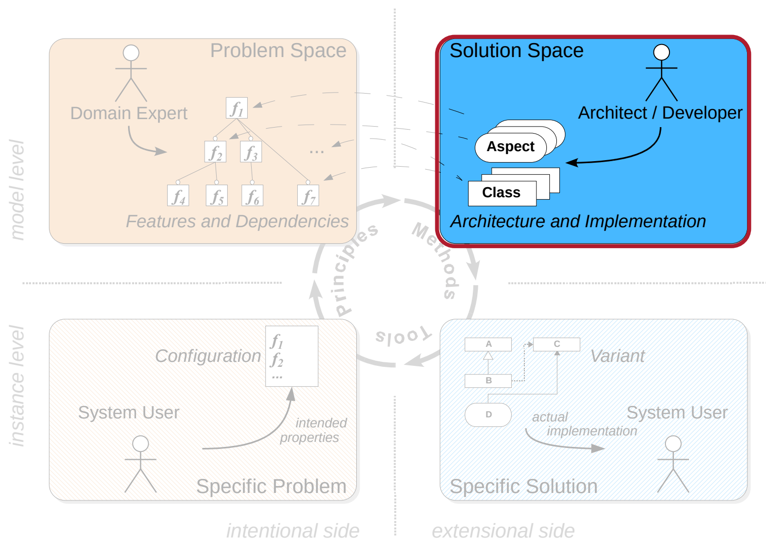
Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 12 – 2012-06-27

http://www4.informatik.uni-erlangen.de/Lehre/SS12/V_KSS

About this Lecture



Implementation Techniques: Classification

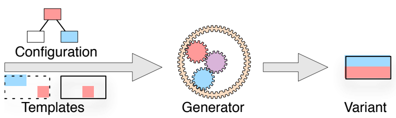
- Decompositional Approaches

  - Text-based filtering (untyped)
  - Preprocessors

- Compositional Approaches

  - Language-based composition mechanisms (typed)
  - OOP, **AOP**, Templates

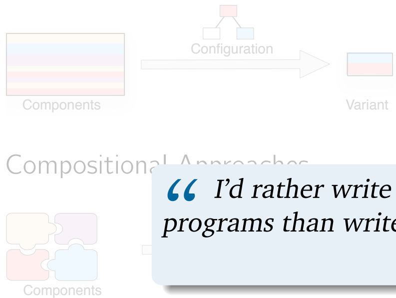
- Generative Approches



  - Metamodel-based generation of components (typed)
  - MDD, C++ TMP, generators

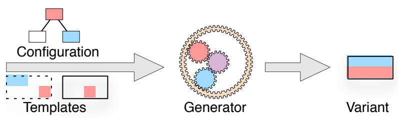
Implementation Techniques: Classification

- Decompositional Approaches
  - Text-based filtering (untyped)
  - Preprocessors
- Compositional Approaches



> *“I'd rather write programs to write programs than write programs.”*
>
> Dick Sites (DEC)

- Generative Approches
  - Metamodel-based generation of components (typed)
  - MDD, C++ TMP, generators



Agenda

6.1 Motivation: OSEK and Co
6.2 SLOTH: Threads as Interrupts
6.3 SLEEPYSLOTH: Threads as IRQs as Threads
6.4 Outlook: SLOTH ON TIME
6.5 Summary and Conclusions
6.6 References

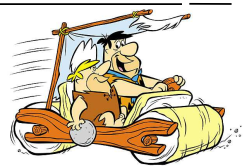
Agenda

6.1 Motivation: OSEK and Co
- Background
- OSEK OS: Abstractions
- OSEK OS: Tailoring and Generation

6.2 SLOTH: Threads as Interrupts
6.3 SLEEPYSLOTH: Threads as IRQs as Threads
6.4 Outlook: SLOTH ON TIME
6.5 Summary and Conclusions
6.6 References

The OSEK Family of Automotive OS Standards

- **1995** OSEK OS (OSEK/VDX) [6]
- **2001** OSEKtime (OSEK/VDX) [8]
- **2005** AUTOSAR OS (AUTOSAR) [1]



- **OSEK OS** ↦ "Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen"
  - **statically configured**, event-triggered real-time OS
- **OSEKtime**
  - **statically configured**, time-triggered real-time OS
  - can optionally be extended with OSEK OS (to run in slack time)
- **AUTOSAR OS** ↦ "Automotive Open System Architecture"
  - **statically configured**, event-triggered real-time OS
  - real superset of OSEK OS ⇝ backwards compatible
  - additional time-triggered abstractions (schedule tables, timing protetion)
  - intended as successor for both, OSEK OS and OSEKtime

OSEK OS: Abstractions [6]

- Control flows
  - `Task`: software-triggered control flow (strictly priority-based scheduling)
    - Basic Task (BT) — run-to-completion task with strictly stack-based activation and termination
    - Extended Task (ET) — may suspend and resume execution (↦ coroutine)
  - `ISR`: hardware-triggered control flow (hardware-defined scheduling)
    - Cat 1 ISR (ISR1) — runs below the kernel, may not invoke system services (↦ prologue without epilogue)
    - Cat 2 ISR (ISR2) — synchronized with kernel, may invoke system services (↦ epilogue without prologue)
  - `Hook`: OS–triggered signal/exception handler
    - `ErrorHook` — invoked in case of a syscall error
    - `StartupHook` — invoked at system boot time
    - ...

OSEK OS: Abstractions [6] (Cont'd)

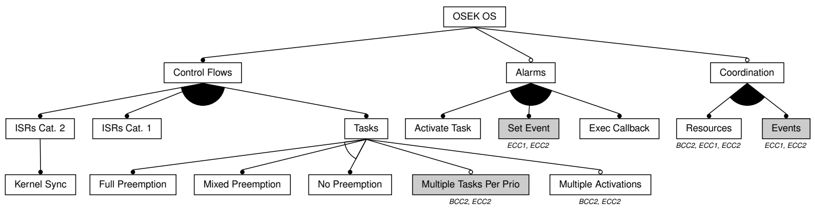
- Coordination and synchronization
  - `Resource`: mutual exclusion between well-define set of tasks
    - stack-based priority ceiling protocol ([9]): `GetResource()` ⤳ priority is raised to that of highest participating task
    - pre-defined `RES_SCHED` has highest priority (⤳ blocks preemption)
    - implementation-optional: task set may also include cat 2 ISRs
  - `Event`: condition variable on which ETs may block
    - part of a task's context
  - `Alarm`: asynchronous trigger by HW/SW counter
    - may execute a callback, activate a task or set an event on expiry

OSEK OS: System Services (Excerpt)

- `Task`-related services
  - `ActivateTask(`*task*`)` ⤳ *task* is active (↦ ready), counted
  - `TerminateTask()` ⤳ running task is terminated
  - `Schedule()` ⤳ active task with highest priority is running
  - `ChainTask(`*task*`)` ↦ atomic { `ActivateTask(`*task*`)`, `TerminateTask()` }
- `Resource`-related services
  - `GetResource(`*res*`)` ⤳ current task has *res* ceilinig priority
  - `ReleaseResource(`*res*`)` ⤳ current task has previous priority
- `Event`-related services (extended tasks only!)
  - `SetEvent(`*task*`, `*mask*`)` ⤳ events *mask* for *task* are set
  - `ClearEvent(`*mask*`)` ⤳ events *mask* for current task are unset
  - `WaitEvent(`*mask*`)` ⤳ current task blocks, until event from *mask* has been set
- `Alarm`-related services
  - `SetAbsAlarm(`*alarm*`, ...)` ⤳ arms *alarm* with absolute offset
  - `SetRelAlarm(`*alarm*`, ...)` ⤳ arms *alarm* with relative offset

OSEK OS: Conformance Classes [6]

- OSEK offers predefined tailorability by four **conformance classes**
  - **BCC1** only basic tasks, limited to one activation request per task and one task per priority, while all tasks have different priorities
  - **BCC2** like BCC1, plus more than one task per priority possible and multiple requesting of task activation allowed
  - **ECC1** like BCC1, plus extended tasks
  - **ECC2** like ECC1, plus more than one task per priority possible and multiple requesting of task activation allowed for basic tasks
- The OSEK feature diagram



OSEK OS: System Specification with OIL [7]

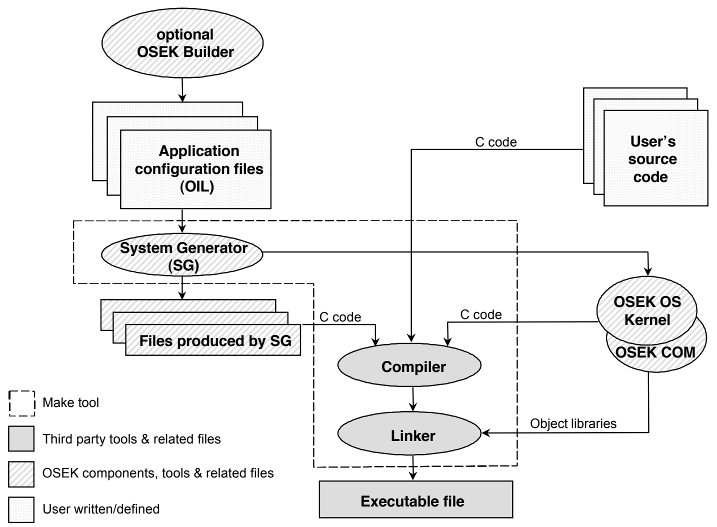
- An OSEK OS instance is configured **completely statically**
  - all general OS features (Hooks, ...)
  - all instances of OS abstractions (Tasks, ...)
  - all relationships between OS abstractions
  - described in a domain-specific language (DSL)
- OIL: The OSEK Interface Language [7]
  - standard types and attributes (`TASK`, `ISR`, ...)
  - vendor/plattform-specific *attributes* (ISR source, priority, triggering)
  - task types and conformance class is deduced

OIL File for Example System (BCC1)

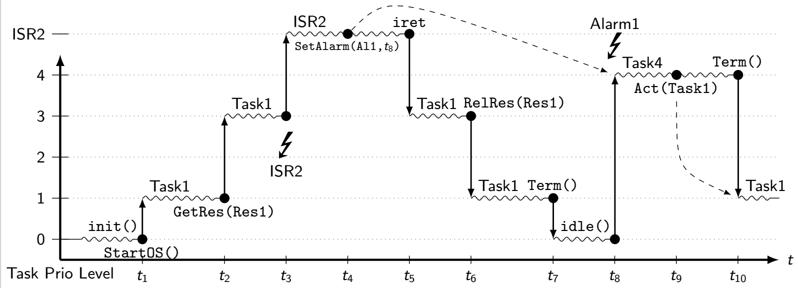
- Three basic tasks: `Task1`, `Task3`, `Task4`
- Category 2 ISR: `ISR2` (platform-spec. source/priority)
- `Task1` and `Task3` use resource `Res1` ⤳ ceiling pri = 3
- Alarm `Alarm1` triggers `Task4` on expiry

```
...
OS ExampleOS {
  STATUS      = STANDARD;
  STARTUPHOOK = TRUE
};
TASK Task1 {
  PRIORITY    = 1;
  AUTOSTART   = TRUE;
  RESOURCE    = Res1;
};
TASK Task3 {
  PRIORITY    = 3;
  AUTOSTART   = FALSE;
  RESOURCE    = Res1;
};
TASK Task4 {
  PRIORITY    = 4;
  AUTOSTART   = FALSE;
};
RESOURCE Res1 {
  RESOURCEPROPERTY = STANDARD;
}
ISR ISR2 {
  CATEGORY    = 2;
  PRIORITY    = 2;
}
ALARM Alarm1 {
  COUNTER     = Timer1;
  ACTION      = ACTIVATETASK {
    TASK      = Task4;
  };
  AUTOSTART   = FALSE;
}
```

OSEK OS: System Generation [7, p. 5]



OSEK OS: Example Control Flow



- Basic tasks behave much like IRQ handlers
  (on a system with support for IRQ priority levels)
  - priority-based dispatching with run-to-completion
  - LIFO, all control flows can be executed on a single shared stack
- So why not dispatch tasks as ISRs?
  ⇝ Let the hardware do all scheduling!
  ⇝ Let's be a SLOTH!

Agenda

6.1 Motivation: OSEK and Co
6.2 SLOTH: Threads as Interrupts
  Basic Idea
  Design
  Results
  Limitation
6.3 SLEEPYSLOTH: Threads as IRQs as Threads
6.4 Outlook: SLOTH ON TIME
6.5 Summary and Conclusions
6.6 References

"SLOTH: Threads as Interrupts" [3]

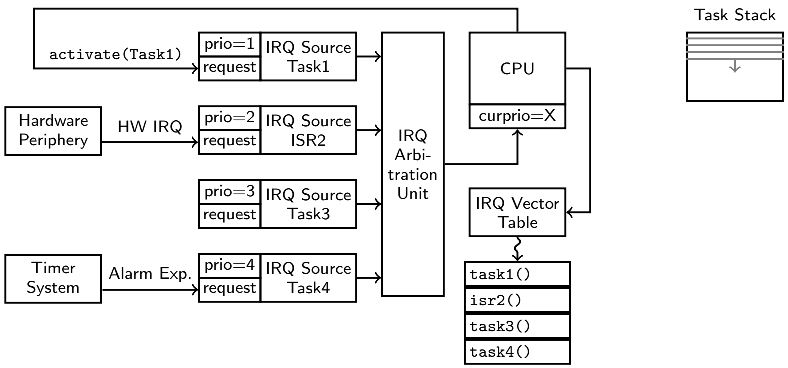
- **Idea: threads are interrupt handlers, synchronous thread activation is IRQ**

Paper title of [3] is a pun to the approach taken by SOLARIS: "Interrupts as Threads", ACM OSR (1995) [5]

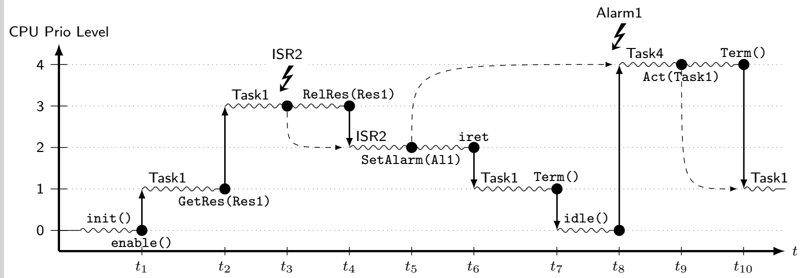
- Let interrupt subsystem do the scheduling and dispatching work
- Applicable to priority-based real-time systems
- Advantage: small, fast kernel with unified control-flow abstraction

Sloth Design

- IRQ system must support priorities and software triggering



Sloth: Example Control-Flow



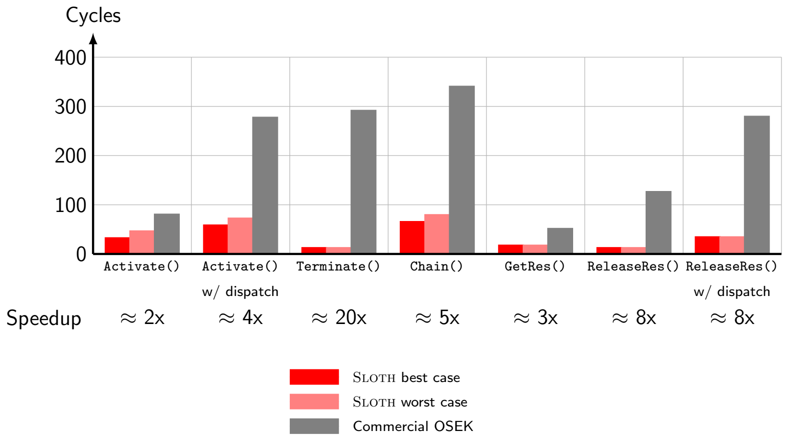
Sloth: Qualitative Results

- Concise kernel design and implementation
  - < 200 LoC, < 700 bytes code memory, very little RAM
- Single control-flow abstraction for tasks, ISRs (1/2), callbacks
  - Handling oblivious to how it was triggered (by hardware or software)
- Unified priority space for tasks and ISRs
  - no rate-monotonic priority inversion [2]
- Straight-forward synchronization by altering CPU priority
  - Resources with ceiling priority (also for ISRs!)
  - Non-preemptive sections with `RES_SCHEDULER` (highest task priority)
  - Kernel synchronization with highest task/cat.-2-ISR priority

Performance Evaluation: Methodology

- Reference implementation for Infineon TriCore
  - 32-bit load/store architecture
  - Interrupt controller: 256 priority levels, about 200 IRQ sources with memory-mapped registers
  - Meanwhile also implementations for ARM Cortex M3 (SAM3) and x86
- Evaluation of task-related system calls:
  - Task activation
  - Task termination
  - Task acquiring/releasing resource
- Comparison with commercial OSEK implementation and CiAO
- Two numbers for Sloth: best case, worst case
  - Depending on number of tasks and system frequency

Performance Evaluation: Results

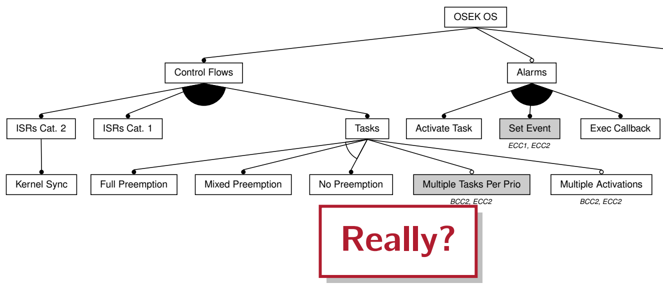


Performance Evaluation: Comparison with CiAO

	Act() w/o dis-patch	Act() w/ dispatch	Term() w/ dispatch	Chain() w/ dispatch	GetRes() w/o dis-patch	RelRes() w/o dis-patch	RelRes() w/ dispatch
SLOTH best case	34	60	14	67	19	14	36
SLOTH worst case	48	74	14	81	19	14	36
CiAO	75	206	107	139	19	66	204

Limitations of the SLOTH Approach

- No extended tasks (that is, events, ↦ OSEK ECC1 / ECC2)
  ↩ **impossible with stack-based IRQ execution model**
- No multiple tasks per priority (↦ OSEK BCC2 / ECC2)
  ↩ execution order has to be the same as activation order



Agenda

6.1 Motivation: OSEK and Co
6.2 SLOTH: Threads as Interrupts
6.3 SLEEPYSLOTH: Threads as IRQs as Threads
- Motivation
- Design
- Results
- SLOTH∗ Generation

6.4 Outlook: SLOTH ON TIME
6.5 Summary and Conclusions
6.6 References

Control Flows in Embedded Systems

	Activation Event	Sched./Disp.	Semantics
ISRs	HW	by HW	RTC
Threads	SW	by OS	Blocking
SLOTH [3]	HW or SW	by HW	RTC
SLEEPYSLOTH [4]	HW or SW	by HW	RTC or Blocking

(RTC: Run-to-Completion)

SLEEPYSLOTH: Main Goal and Challenge

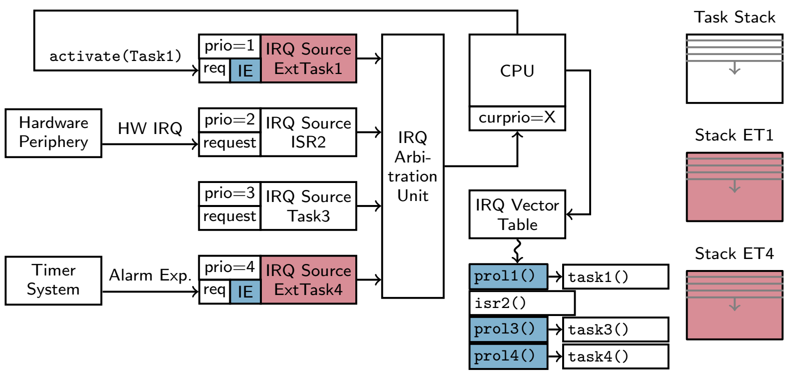
Main Goal

Support extended blocking tasks (with stacks of their own), while preserving SLOTH's latency benefits by having threads run as ISRs

Main Challenge

IRQ controllers do not support suspension and re-activation of ISRs

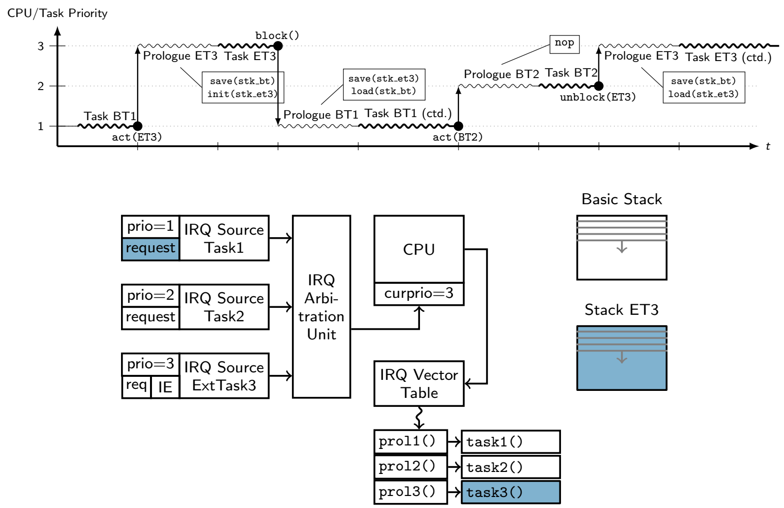
SLEEPYSLOTH Design: Task Prologues and Stacks



SLEEPYSLOTH: Dispatching and Rescheduling

- Task prologue: switch stacks if necessary
  - Switch *basic task* ↪ *basic task* omits stack switch
  - On job start: initialize stack
  - On job resume: restore stack
- Task termination: task with next-highest priority needs to run
  - Yield CPU by setting priority to zero
  - (Prologue of *next* task performs the stack switch)
- Task blocking: take task out of "ready list"
  - Disable task's IRQ source
  - Yield CPU by setting priority to zero
- Task unblocking: put task back into "ready list"
  - Re-enable task's IRQ source
  - Re-trigger task's IRQ source by setting its pending bit

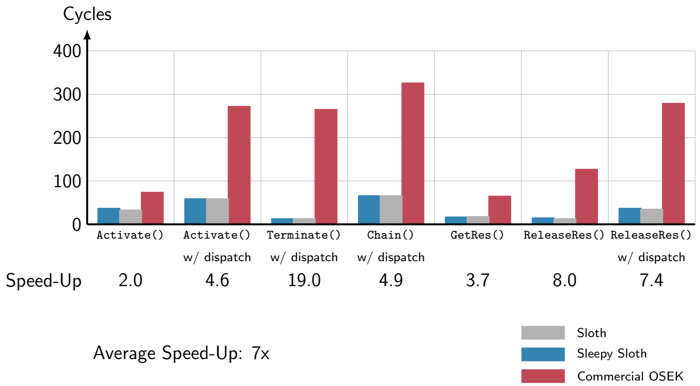
SleepySloth: Example Control Flow



SleepySloth: Evaluation

- Reference implementation on Infineon TriCore microcontroller
- Measurements: system call latencies in 3 system configurations, compared to a leading commercial OSEK implementation
  1. Only basic run-to-completion tasks
  2. Only extended blocking tasks
  3. Both basic and extended tasks

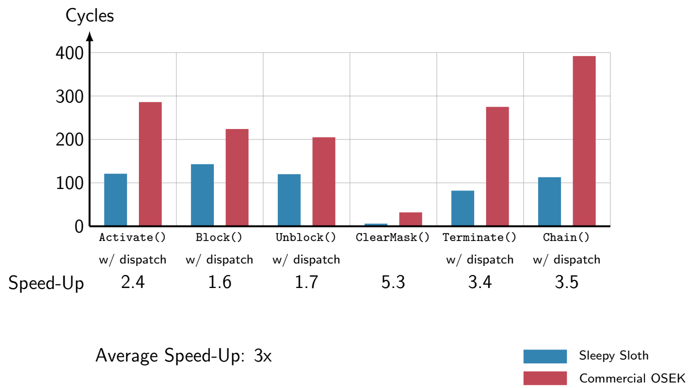
Evaluation: Only Basic Tasks



Average Speed-Up: 7x

- SleepySloth outperforms commercial kernel with SW scheduler
- SleepySloth as fast as original Sloth

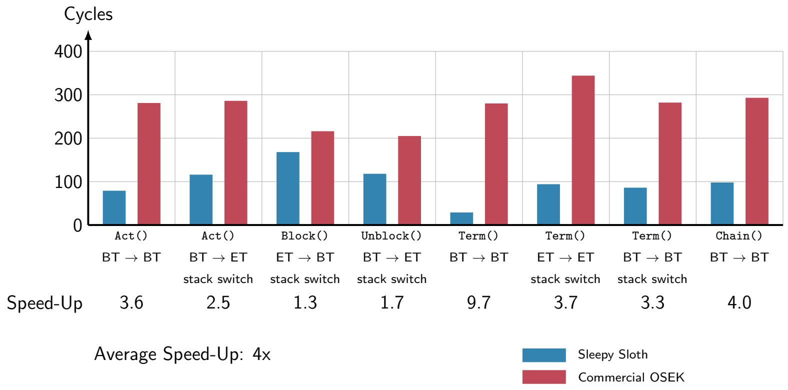
Evaluation: Only Extended Tasks



Average Speed-Up: 3x

- Still faster than commercial kernel with SW scheduler
- SleepySloth: Extended switches slower than basic switches

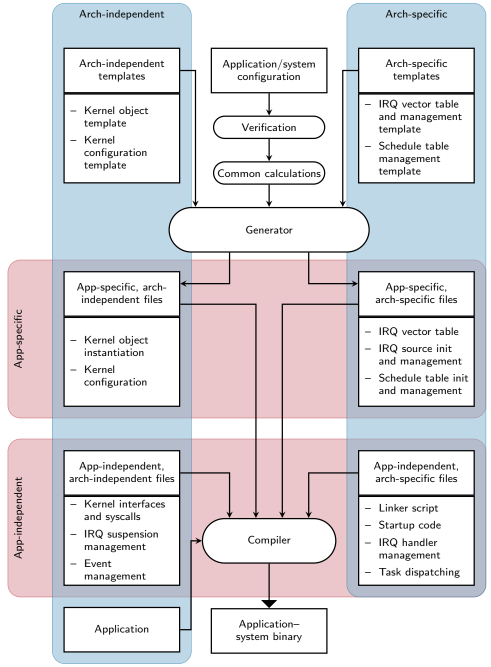
Evaluation: Extended *and* Basic Tasks



- Basic switches in a mixed system only slightly slower than in purely basic system

Sloth∗ Generation

- Two generation dimensions
  - Architecture
  - Application
- Generator is implemented in Perl
  - templates
  - configuration

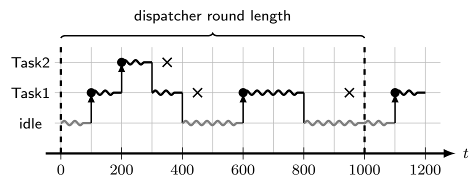


Agenda

6.1 Motivation: OSEK and Co
6.2 Sloth: Threads as Interrupts
6.3 SleepySloth: Threads as IRQs as Threads
6.4 Outlook: Sloth on Time
6.5 Summary and Conclusions
6.6 References

Sloth on Time: Time-Triggered Laziness

- **Idea: user hardware timer arrays to implement schedule tables**
- TC1796 GPTA: 256 timer cells, routable to 96 interrupt sources
  - use for task activation, deadline monitoring, execution time budgeting
- Sloth on Time implements OSEKtime [8] and AUTOSAR OS schedule tables [1]
  - combinable with Sloth or SleepySloth for mixed-mode systems
  - up to 170x lower latencies compared to commercial implementations



Agenda

6.1 Motivation: OSEK and Co
6.2 Sloth: Threads as Interrupts
6.3 SleepySloth: Threads as IRQs as Threads
6.4 Outlook: Sloth on Time
6.5 Summary and Conclusions
6.6 References

Summary: The Sloth* Approach

- Exploit standard IR/timer hardware to delegate core OS functionality to hardware
  - scheduling and dispatching of control flows
  - OS needs to be tailored to application *and* hardware platform
    - ⇝ generative approach is necessary
- Benefits
  - tremendous latency reductions, very low memory footprints
  - unified control flow abstraction
    - hardware/software-triggered, blocking/run-to-completion
    - no need to distinguish between tasks and ISRs
    - reduces complexity
  - less work for the OS developer :-)

We are sloth

Referenzen

[1] AUTOSAR. *Specification of Operating System (Version 4.1.0)*. Tech. rep. Automotive Open System Architecture GbR, Oct. 2010.

[2] Luis E. Leyva del Foyo, Pedro Mejia-Alvarez, and Dionisio de Niz. "Predictable Interrupt Management for Real Time Kernels over conventional PC Hardware". In: *Proceedings of the 12th IEEE International Symposium on Real-Time and Embedded Technology and Applications (RTAS '06)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2006, pp. 14–23. DOI: `10.1109/RTAS.2006.34`.

[3] Wanja Hofer, Daniel Lohmann, Fabian Scheler, et al. "Sloth: Threads as Interrupts". In: *Proceedings of the 30th IEEE International Symposium on Real-Time Systems (RTSS '09)*. IEEE Computer Society Press, Dec. 2009, pp. 204–213. ISBN: 978-0-7695-3875-4. DOI: `10.1109/RTSS.2009.18`.

[4] Wanja Hofer, Daniel Lohmann, and Wolfgang Schröder-Preikschat. "Sleepy Sloth: Threads as Interrupts as Threads". In: *Proceedings of the 32nd IEEE International Symposium on Real-Time Systems (RTSS '11)*. IEEE Computer Society Press, Dec. 2011, pp. 67–77. ISBN: 978-0-7695-4591-2. DOI: `10.1109/RTSS.2011.14`.

[5] Steve Kleiman and Joe Eykholt. "Interrupts as Threads". In: *ACM SIGOPS Operating Systems Review* 29.2 (Apr. 1995), pp. 21–26. ISSN: 0163-5980.

Referenzen (Cont'd)

[6] OSEK/VDX Group. *Operating System Specification 2.2.3*. Tech. rep. `http://portal.osek-vdx.org/files/pdf/specs/os223.pdf`, visited 2011-08-17. OSEK/VDX Group, Feb. 2005.

[7] OSEK/VDX Group. *OSEK Implementation Language Specification 2.5*. Tech. rep. `http://portal.osek-vdx.org/files/pdf/specs/oil25.pdf`, visited 2009-09-09. OSEK/VDX Group, 2004.

[8] OSEK/VDX Group. *Time Triggered Operating System Specification 1.0*. Tech. rep. `http://portal.osek-vdx.org/files/pdf/specs/ttos10.pdf`. OSEK/VDX Group, July 2001.

[9] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". In: *IEEE Transactions on Computers* 39.9 (1990), pp. 1175–1185. ISSN: 0018-9340. DOI: `10.1109/12.57058`.