Timed Functional Simulation and Interference Analysis of Mixed-Criticality Applications

Verkehr
Transportation

Kim Grüttner, Philipp Ittershagen,
Frank Oppenheimer

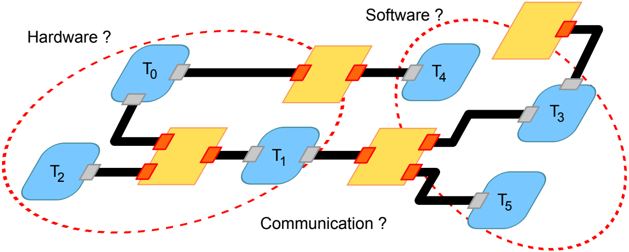
`kim.gruettner@offis.de`

OFFIS – Institute for Information Technology
Hardware/Software Design Methodology Group
R&D Division Transportation

December, 9th 2014
EMC^2 WP2/WP4 Meeting

1 HW/SW Communication and Partitioning

Motivation



- **Communication** between Tasks is a **critical part** of the development of concurrent embedded systems
- **Mapping** of components (Hard-/Software) is often subject to **changes**
  - but require expensive design changes
- **Goal:** Enable inexpensive HW/SW (re-)partitioning
  - **transparent tool-based** support of communication **across HW/SW boundaries**
  - provision of **generic HW/SW interface implementation**

2 Outline

1 Previous Work

2 Planned Extensions in EMC^2

3 Conclusion

3 Outline

Previous Work

1. Previous Work
   - OSSS – Oldenburg System Synthesis Subset
   - Introduction to rmi4linux
2. Planned Extensions in EMC^2
3. Conclusion

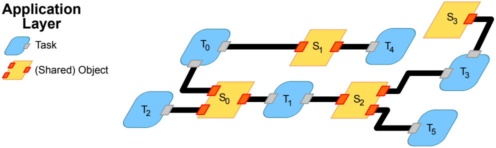
4 Properties of OSSS

Previous Work

- OSSS: Extension of **SystemC**
- Offers **homogeneous** Modeling- and Refinement Methodology, for describing **HW** and **SW components**
    - Hardware Module
    - Software Task
    - **Shared Object**
- Shared Object: Communication objects between HW modules and SW tasks
    - Synthesis/Mapping in dedicated HW or Memory

5 Application and Virtual Target Architecture Layer

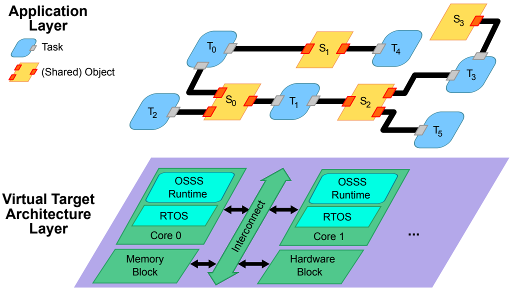
Previous Work



- Initial model: **Application Layer**
    - **Concurrent Tasks**, communicating through **Method Calls** on Shared Objects

5 Application and Virtual Target Architecture Layer

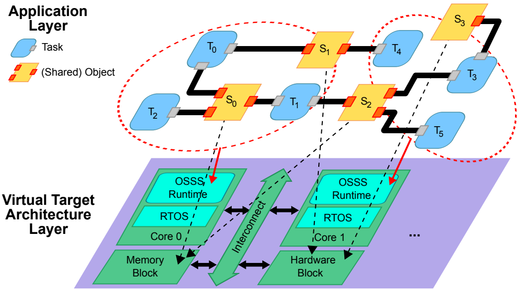
Previous Work



- Initial model: **Application Layer**
  - **Concurrent Tasks**, communicating through **Method Calls** on Shared Objects
- Refinement: **Virtual Target Architecture (VTA) Layer**

5 Application and Virtual Target Architecture Layer

Previous Work



- Initial model: **Application Layer**
  - **Concurrent Tasks**, communicating through **Method Calls** on Shared Objects
- Refinement: **Virtual Target Architecture (VTA) Layer**
  - Shared Objects can be mapped to **HW Blocks** and **Memories**

6 Shared Objects

Previous Work

- **Shared Objects** offer **method-based interface (Service)** for tasks
- Concurrently accessing **tasks (clients) are synchronized transparently**
- Services can have **logical pre-conditions (Guards)**
  - based on **inner state** of **Shared Object**
  - can **block** service call until Guard is fulfilled
  - a blocked service call can only be released by **another unblocked service**, changing the Shared Object's inner state

6 Shared Objects

Previous Work

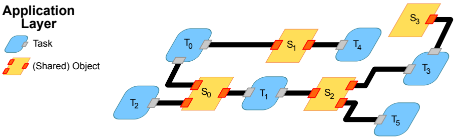
- **Shared Objects** offer **method-based interface (Service)** for tasks
- Concurrently accessing **tasks (clients) are synchronized transparently**
- Services can have **logical pre-conditions (Guards)**
  - based on **inner state** of **Shared Object**
  - can **block** service call until Guard is fulfilled
  - a blocked service call can only be released by **another unblocked service**, changing the Shared Object's inner state

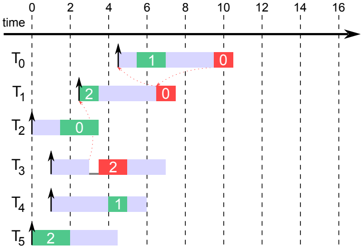
FIFO-Method	Guard
`put(int item)`	`!full`
`int get()`	`!empty`

Table: Example for using Guards

7 Example

Previous Work



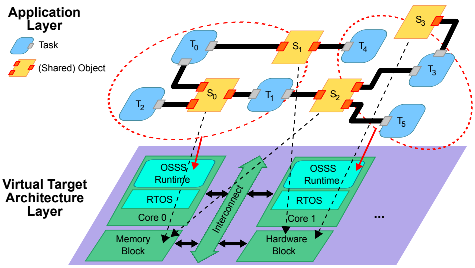


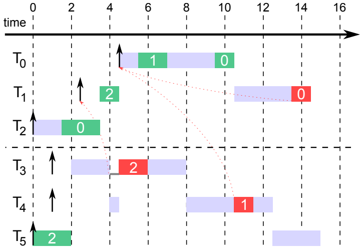
Unscheduled

- ↑ arrival
- 🟪 running
- 🟥 X access res. X (sync required)
- ····· preemption
- ── blocked
- 🟩 X access on res. X (decoupled)

7 Example

Previous Work





No preemption, Suspended

↑ arrival

running

🟥 X access res. X (sync required)

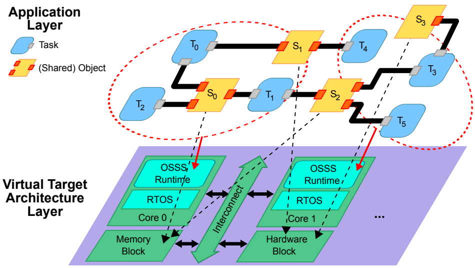
preemption

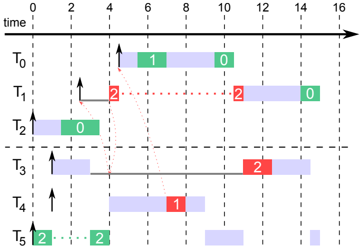
blocked

🟩 X access on res. X (decoupled)

7 Example

Previous Work





Priority Inheritance, Suspended

↑ arrival

running

🟥 X access res. X (sync required)

····· preemption

—— blocked

🟩 X access on res. X (decoupled)

8 Main requirements for rmi4linux

Previous Work

The driver framework **rmi4linux** shall

- offer an **abstract interface** to Shared Objects, **independent from their mappings** (Hardware, Memory, and Processor)
- reduce interferences between Client and (Hardware) Shared Object through **asynchronous protocol**
- **separate platform specific information** from implementation to ease **Porting** between different platforms
- allow execution of **multiple Clients on a CPU**
  - Problem: Serialization of **concurrent access** and **possible blocking** by Guard conditions: **Multi-Level Lock**

9 Communication between SW Tasks and HW Shared Objects

Previous Work

- HW Shared Object has **its own Scheduler** for arbitrating multiple requests, that could be **blocked by Guards**
- Communication of SW Tasks with HW Shared Objects through **Remote Method Invocation** (RMI)

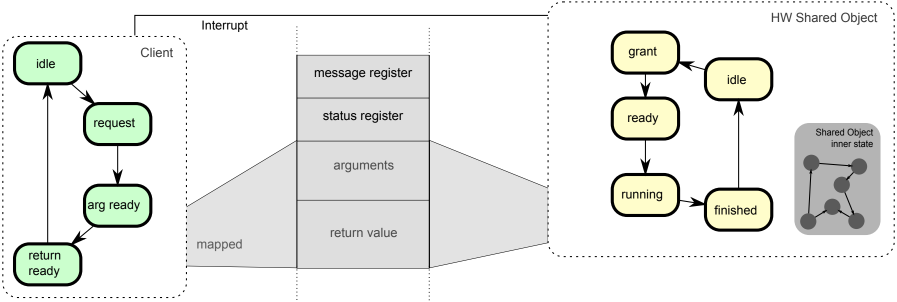
9 Communication between SW Tasks and HW Shared Objects

Previous Work

- HW Shared Object has **its own Scheduler** for arbitrating multiple requests, that could be **blocked by Guards**
- Communication of SW Tasks with HW Shared Objects through **Remote Method Invocation** (RMI)

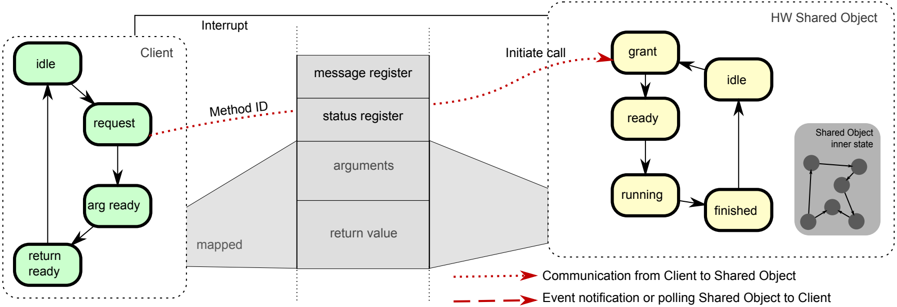
1. Method arguments are serialized
2. Shared Object is notified about requested call:
   - Client sends **Client ID** and **Method ID**
   - Shared Object Scheduler calculates Guard mask, performs **arbitration**, and might **grant requested service call**
3. **Method arguments are received and de-serialized** by Shared Object for **executing Service Call**
4. If service call generates return value, these steps are performed in reverse order

10 Communication between SW Tasks and HW Shared Objects
Previous Work



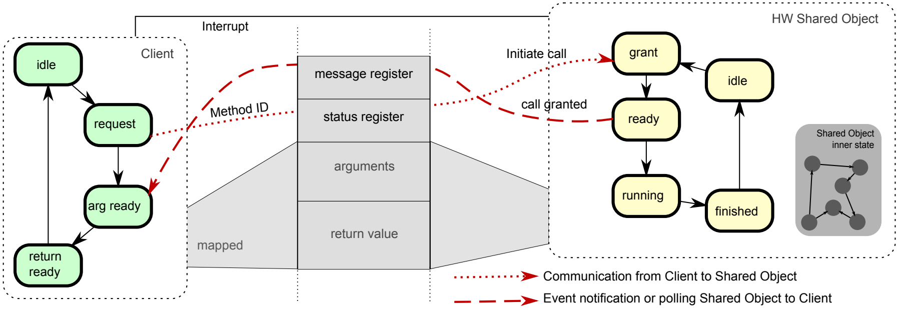
10 Communication between SW Tasks and HW Shared Objects

Previous Work



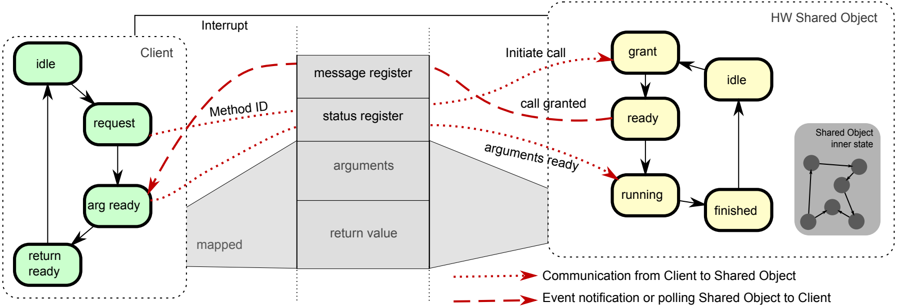
10 Communication between SW Tasks and HW Shared Objects

Previous Work



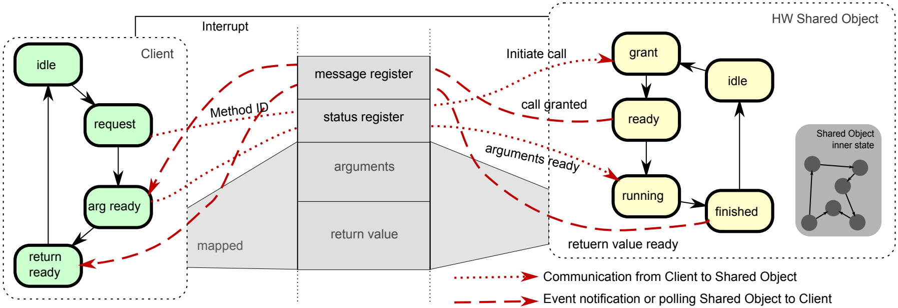
10 Communication between SW Tasks and HW Shared Objects

Previous Work



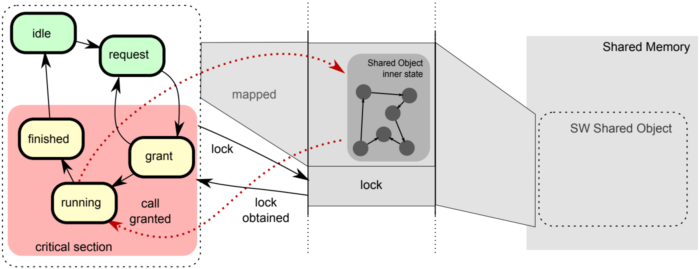
10 Communication between SW Tasks and HW Shared Objects

Previous Work



11 Communication of SW Tasks and SW Shared Objects

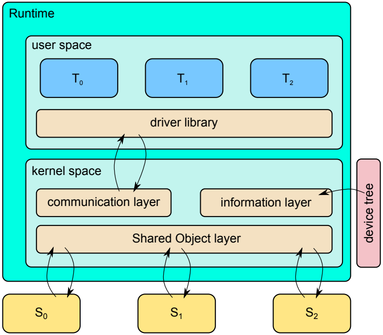
Previous Work



- Methods/Services are executed from within the context of the task
- Modification of inner state requires exclusive access
- Usage of dedicated memory location (SW-Register) for locking access across different CPUs
- Can be relaxed in a single CPU case since Lock can be mapped to OS mechanisms (Semaphore)

12 Integration with Linux
Previous Work

- Kernel Module and Driver Library
- Transfer of Service Arguments and Return data using **dedicated mapped address space**
- Platform information statically available in **device tree**
- Monitoring of Shared Object **State**
  - waiting for **Guard Condition**
  - waiting for **completion** of Service Call



13 Outline

Planned Extensions in EMC^2

1 Previous Work

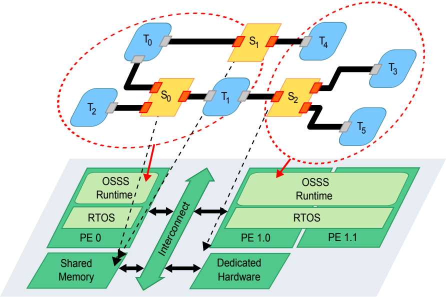
2 Planned Extensions in EMC^2

- Representation of Mixed-Criticality
- Targeted Hardware Platform

3 Conclusion

14 Extension of Virtual Target Architecture Model

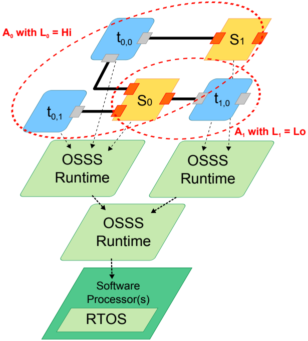
Planned Extensions in EMC^2



- Support for
  - Data and Instruction Cache
  - Memory Management Unit (MMU)
  - Shared Instruction and Data Memory
  - Shared Cache
  - Symmetric Multi-Processing (SMP)
- Provision of
  - hierarchical bus/x-bar interconnect
  - unpredictable bus arbitration
  - predictable (e.g. TDMA) bus arbitration

15 Extensions of Application Layer Model

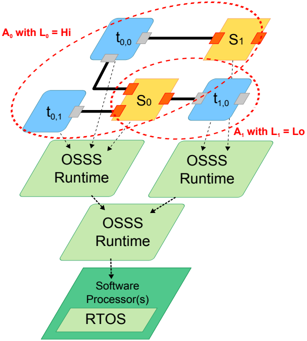
Planned Extensions in EMC^2



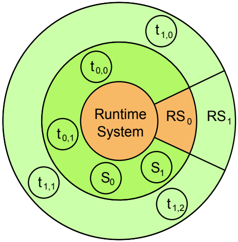
- Mixed-Criticality Model
  - Finite set of Applications A_i with
    - criticality level L_i
    - with set of Tasks T_i
    - with set of Shared Objects S_i
  - Each Task $t_j \in T_i$ is defined by $(P_j, D_j, C_j, SI_j, L_j)$ with
    - period (minimum arrival time) P
    - deadline D
    - workload and memory access graph C
    - ports to Shared Object Interfaces $SI \in Si.I$
    - criticality level L
  - Each Shared Object Si consists of
    - a set of Interfaces I_i with methods $m_j \in i_k \in I_i$ (let M_i be the union of all methods in I_i)
    - a set of side effect free Guards G_i
    - a set of guarded methods $GM_i \in M_i \times G_i$ implementing all interfaces methods M_i
    - a shared resource access arbitration policy

16 Extensions of OSSS Runtime

Planned Extensions in EMC²



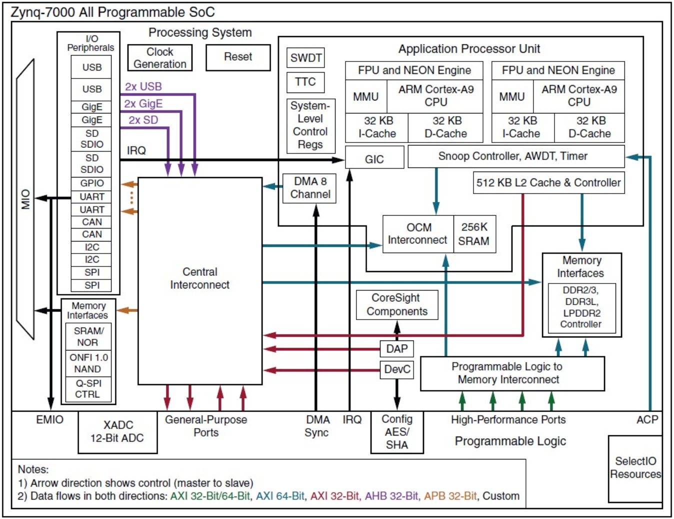
- Hierarchical Scheduling, e.g.
  - ring-based scheduling with different scheduler per ring
  - tree-based scheduling
  - ...



RS_0, RS_1 : Runtime System Scheduler
$t_{0,0}$, $t_{0,1}$: Tasks of RS_0
$t_{1,0}$, $t_{1,1}$, $t_{1,2}$: Tasks of RS_1
S_0, S_1 : Shared Objects of RS_0

17 Targeted Hardware Platform

Planned Extensions in EMC²



Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32-Bit/64-Bit, AXI 64-Bit, AXI 32-Bit, AHB 32-Bit, APB 32-Bit, Custom

18 Outline

Conclusion

1 Previous Work

2 Planned Extensions in EMC^2

3 Conclusion

- **OSSS** offers. . .
  - homogeneous **Modeling- and Refinement Methodology**
  - and executable model for parallel HW/SW SoCs
- Driver framework **rmi4linux**. . .
  - offers **homogeneous interface** of OSSS SW Tasks for accessing Shared Objects under Linux
  - **abstracts** Shared Object **specific mapping** (CPU-local, -global and HW)
- **Outlook:**
  - Representation of **different criticalities** at Application Layer
  - Support for **hierarchical scheduling** to support mixed-criticality scheduling
  - Extension of Virtual Target Architecture Layer to **support Xilinx Zynq**

Thank you very much for your attention!

Questions?

21 Bibliography (I)

Conclusion

P. A. Hartmann, K. Grüttner, P. Ittershagen, and A. Rettberg.
A framework for generic HW/SW communication using remote method invocation.
In *Electronic System Level Synthesis Conference (ESLsyn), 2011*, pages 1–6, June 2011.

P. A. Hartmann, K. Grüttner, A. Rettberg, and I. Podolski.
Distributed Resource-Aware Scheduling for Multi-core Architectures with SystemC.
In Mike Hinchey, Bernd Kleinjohann, Lisa Kleinjohann, Peter A. Lindsay, Franz J. Rammig, Jon Timmis, and Marilyn Wolf, editors, *Distributed, Parallel and Biologically Inspired Systems*, volume 329 of *IFIP Advances in Information and Communication Technology*, pages 181–192. Springer Berlin Heidelberg, 2010.

P. A. Hartmann, H. Kleen, P. Reinkemeier, and W. Nebel.
Efficient Modelling and Simulation of Embedded Software Multi-Tasking using SystemC and OSSS.
In *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, pages 19–24, Sept 2008.

P. Ittershagen, P. A. Hartmann, K. Grüttner, and A. Rettberg.
Hierarchical Real-Time Scheduling in the Multi-Core Era – An Overview.
In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, pages 1–10, June 2013.

P. Reinkemeier, P. Ittershagen, I. Stierand, P. A. Hartmann, S. Henkler, and K. Grüttner.
Seamless Segregation for Multi-Core Systems.
Technical report, OFFIS, 9 2013.