Cornelia Wulf, Sergio Pertuz, Diana Göhringer
Chair of Adaptive Dynamic Systems
TU Dresden

TrustedIoT: Low-power FPGA-based mobile robot with enhanced IoT security

29.08.2024

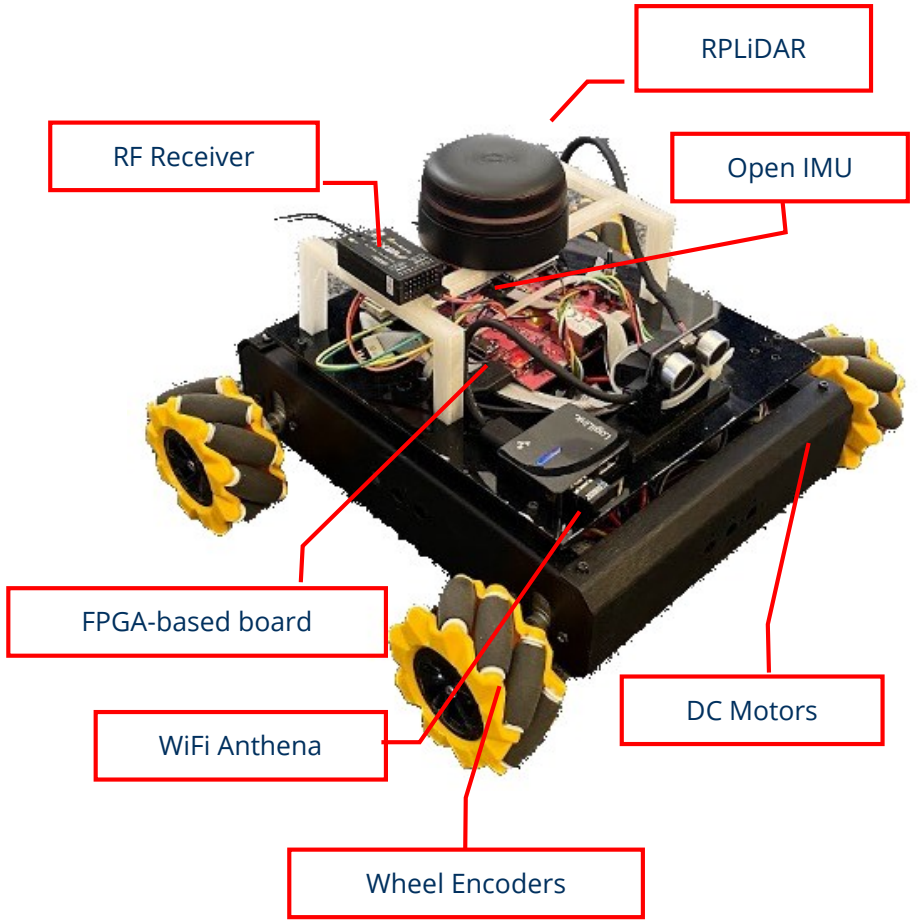
Motivation

Memory isolation

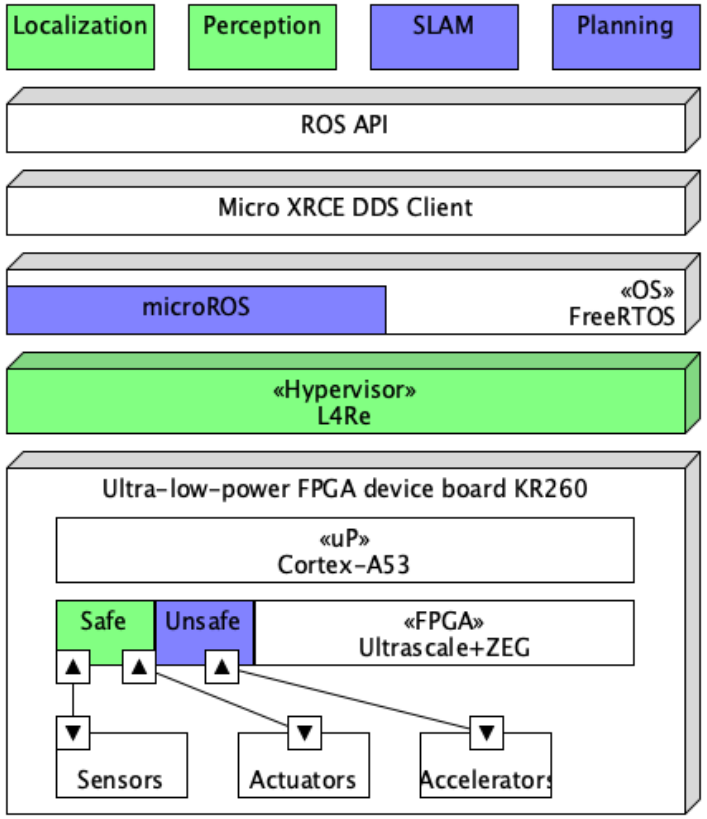
- Software side:
  Hypervisors isolate trusted from untrusted guest operating systems.

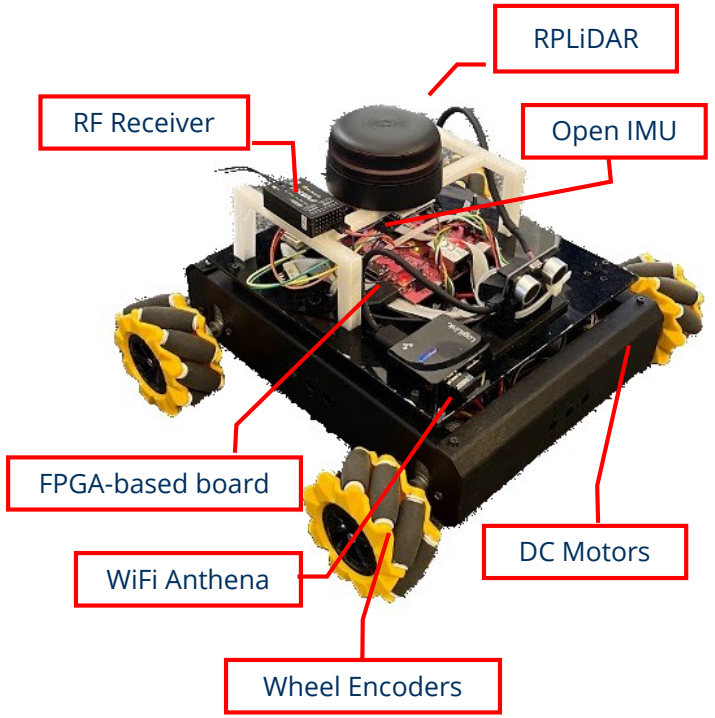
- Hardware side:
  Fine-grained isolation mechanism for shared usage of hardware accelerators is missing.

Focus on AXI memory-mapped interfaces



Architecture Stack

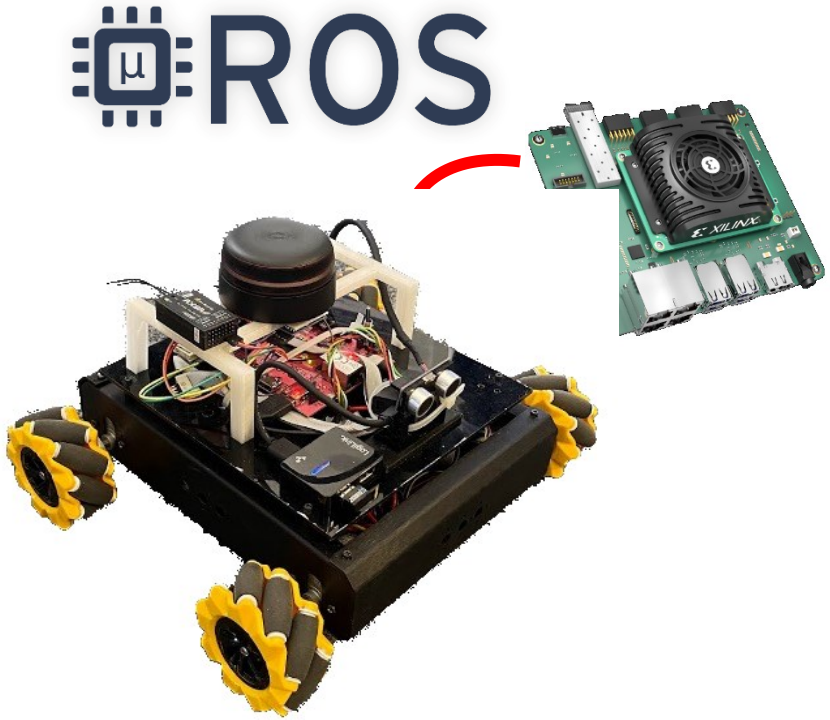




ROS Middleware

TUD implemented

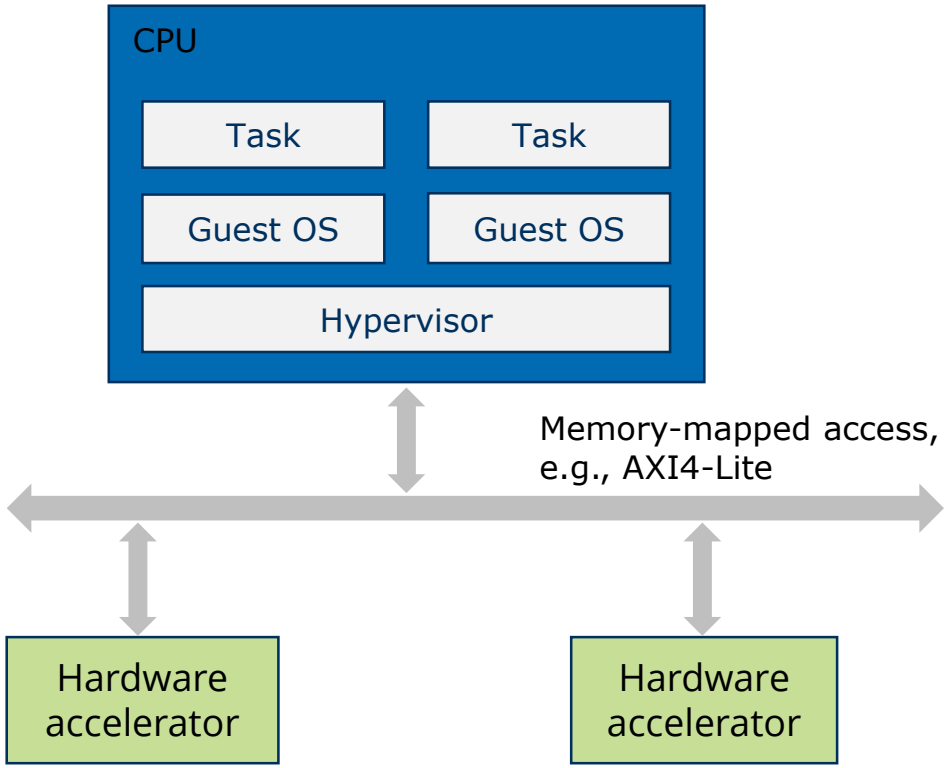
- **ROS2/microROS (novelty)** on a Zynq/ZynqMP device
- and added a **hardware-based secure layer** to the networking and middleware to have improved security in robotics IoT.



Access Control for Hardware Accelerators

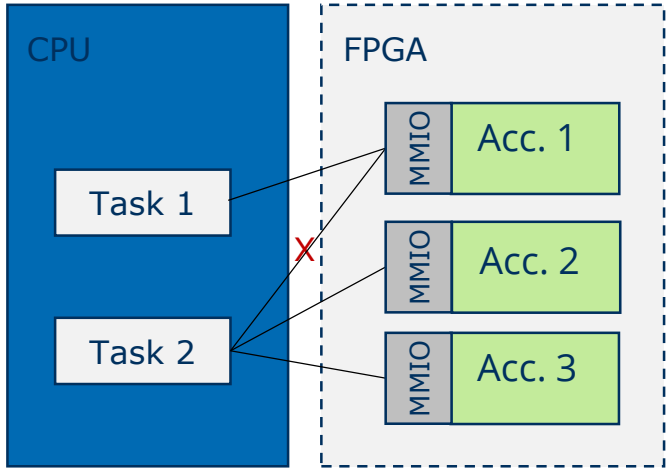
Challenges in embedded virtualized systems:

- Sharing of hardware accelerators
- Protection from unauthorized access
  - Memory-mapped communication (e.g., AXI4-Lite)
  - Adversaries: malicious software tasks that invade address space of a different task



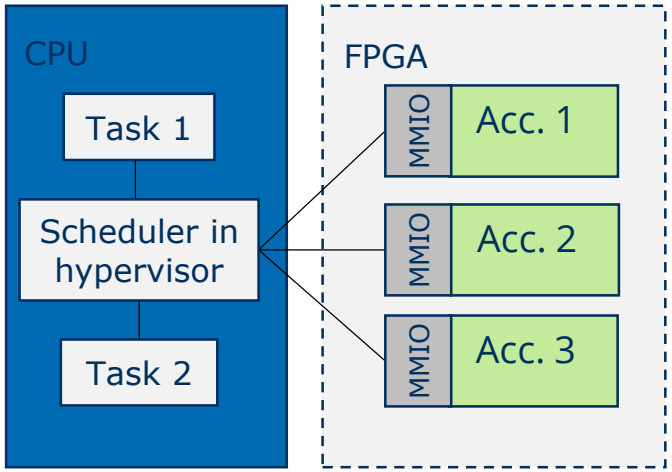
Access Control for Hardware Accelerators

1. Fixed assignment:



Disadvantage:
- No flexibility
- No scalability

2. Access via software scheduler:



Disadvantage:
- Latency

Contributions

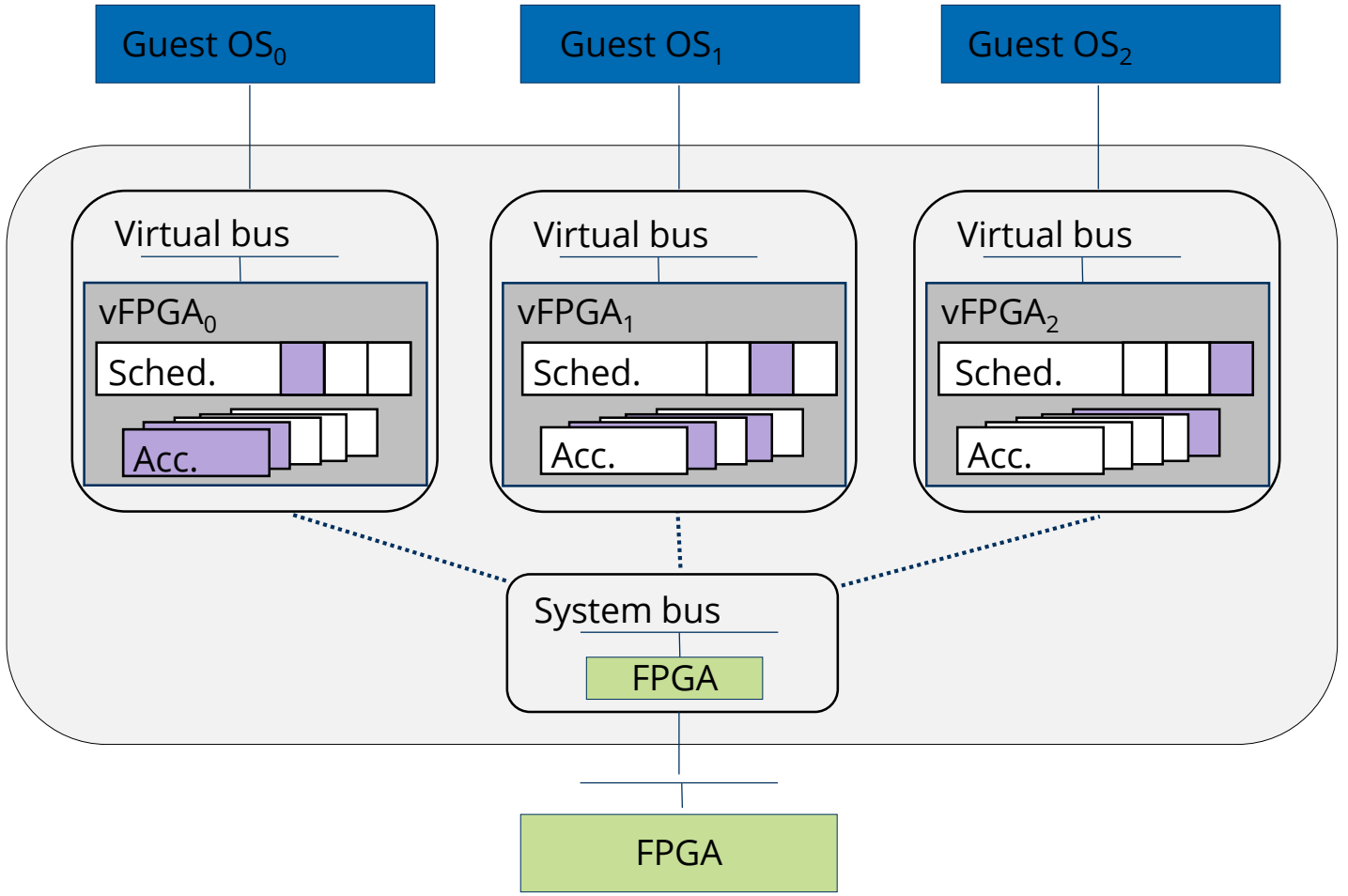
Access control mechanism in hardware:

- Extension of the L4Re isolation mechanisms to memory-mapped communication with hardware accelerators
- Scheduling of hardware tasks in spatial and temporal respect
- Scheduler gives task access rights to hardware accelerator
- Prevention of unauthorized access from malicious software tasks
- Dynamic reaction to changed conditions
- Data isolation enables to share hardware accelerators

Access Control

IO Server of L4Re [1]

- Virtual bus: aggregates resources visible to each IO client
- Virtual FPGA (vFPGA): guest OS's view of FPGA



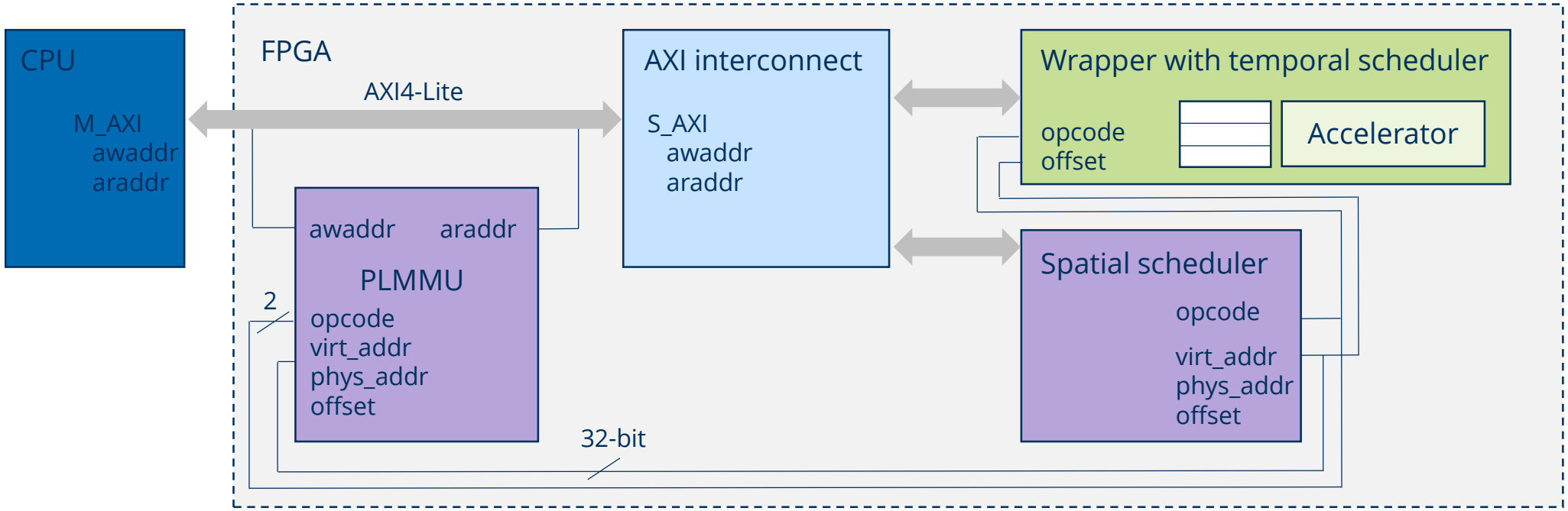
[1] https://l4re.org/

Memory-mapped access of hardware accelerators

l4rec.io

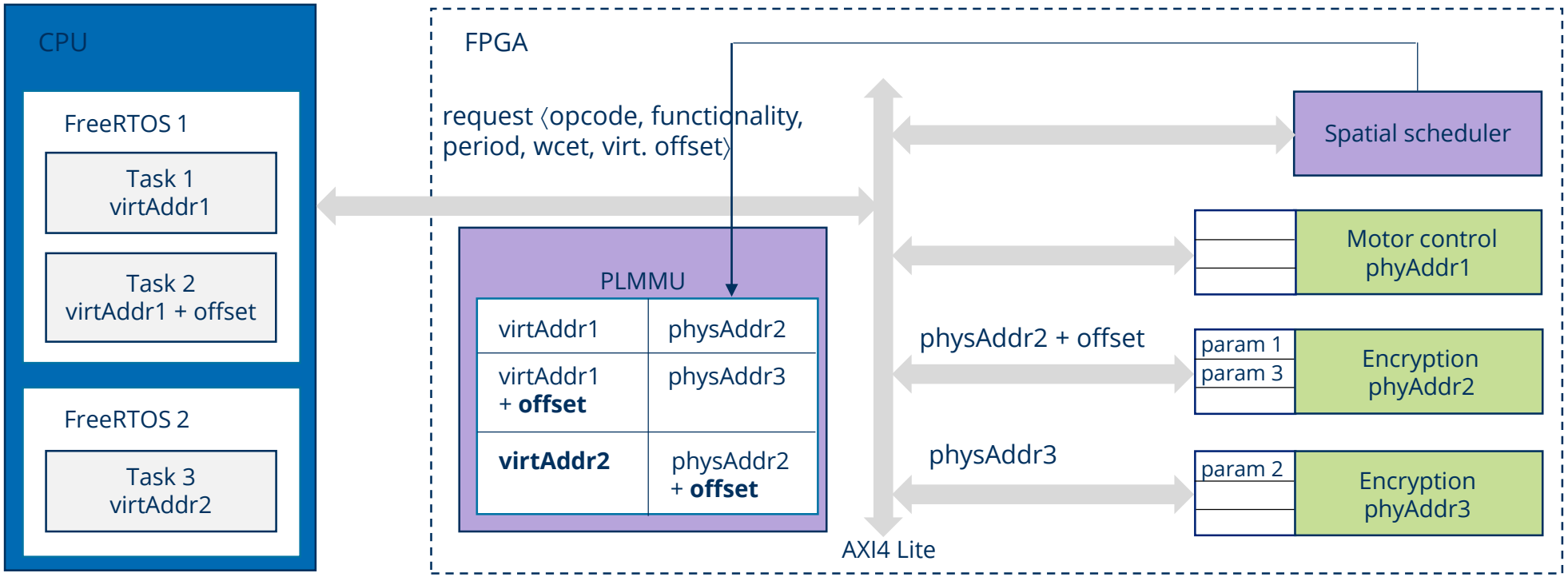
```
Io.hw.add_devices(function()

        os_1 = Io.Hw.Device(function()
                Resource.regs = Io.Res.mmio(BASE_SCHEDULER, BASE_SCHEDULER + OFFSET)
                Resource.regs = Io.Res.mmio(BASE_VIRT1, HIGH_VIRT1)
        end);

        os_2 = Io.Hw.Device(function()
                Resource.regs = Io.Res.mmio(BASE_SCHEDULER + OFFSET,
                                            BASE_SCHEDULER + 2 * OFFSET)
                Resource.regs = Io.Res.mmio(BASE_VIRT2, HIGH_VIRT2)
        end);
        …
End)
```

Hardware Architecture



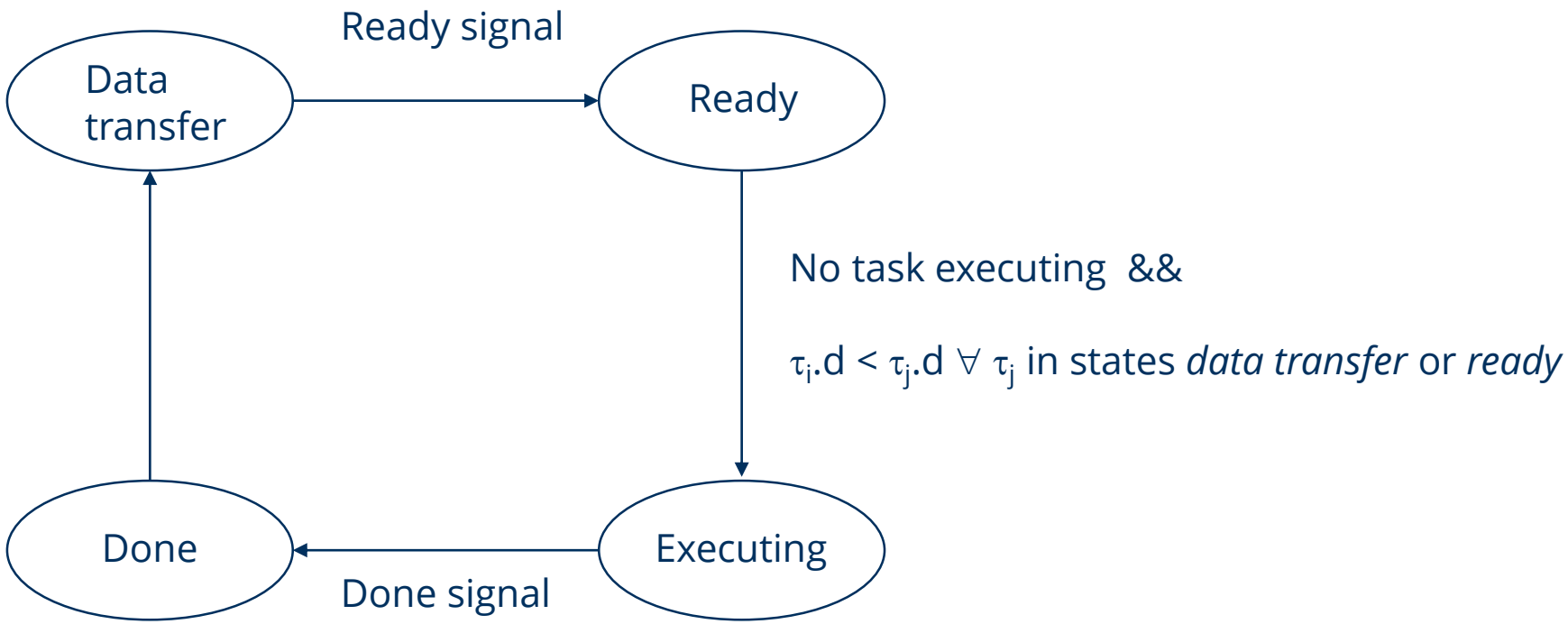
Spatial Scheduling

Utilization U = $\sum \frac{wcet_i}{period_i}$



Temporal Scheduling

Earliest Deadline First (EDF)



Software Tasks

```
// Request access from scheduler
WriteReg(BASE_SCHEDULER_0, INSERT)
WriteReg(BASE_SCHEDULER_0 + 4, ENCRYPTION)
...

// Use accelerator
While(1){      // periodic execution
    WriteReg(BASE_VIRT_0 + 8, deadline);
    WriteReg(BASE_VIRT_0 + 12, parameter);
    ...
    WriteReg(BASE_VIRT_0, 1);    // start
    ...
}
```

```
WriteReg(BASE_SCHEDULER_1, INSERT)
```

```
WriteReg(BASE_VIRT_1 + 8, deadline);
```

Page Fault!

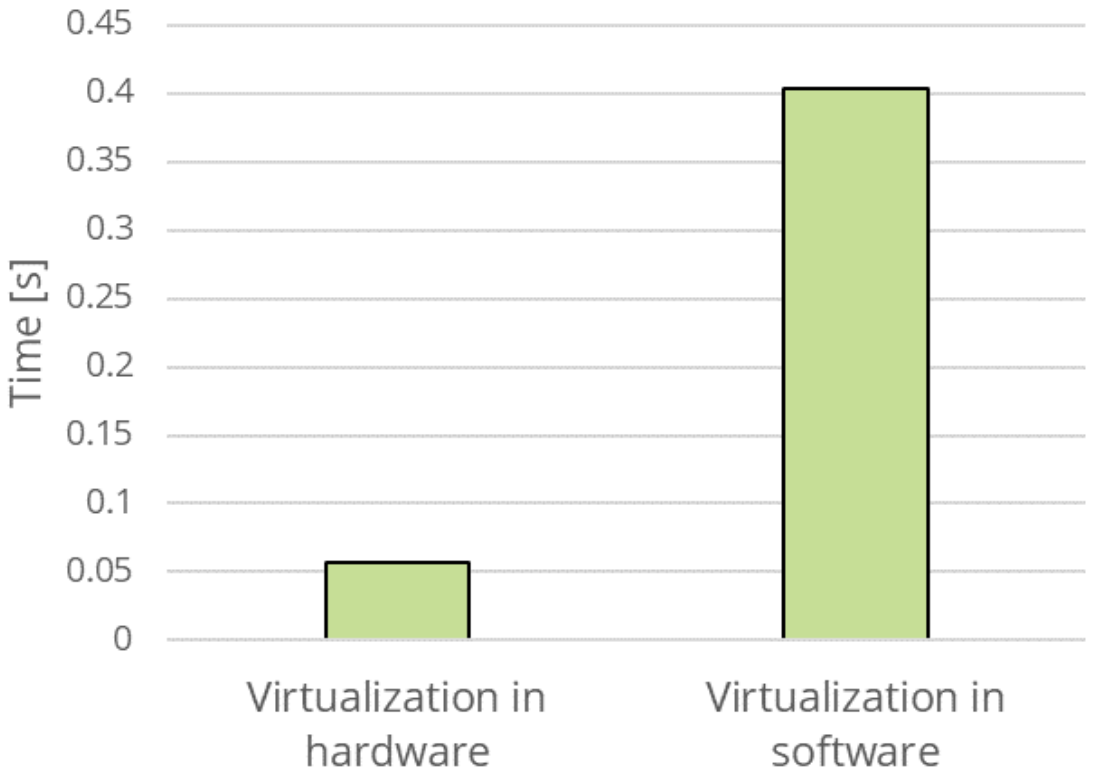
Evaluation: Possible Use Case

- Robotics:
  - Two software tasks that process workpieces in parallel: Exclusive access to motor control IP-core for a certain duration
  - Monitoring task
    All tasks compete for an encryption accelerator
- Zynq UltraScale+ MPSoC ZCU104
- Hardware accelerators
  - Motor Control
  - 4 Prince encryption cores
- ARM Cortex-A53:
  L4Re and 3 paravirtualized FreeRTOS instances, each running 4 hardware tasks

Evaluation: Benefit

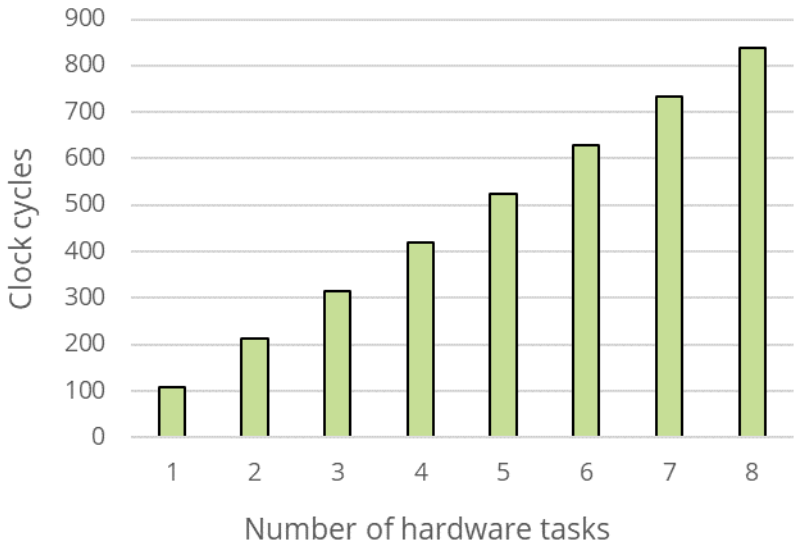
- Unauthorized access:
  Page fault
- Virtualization in hardware
  versus software:

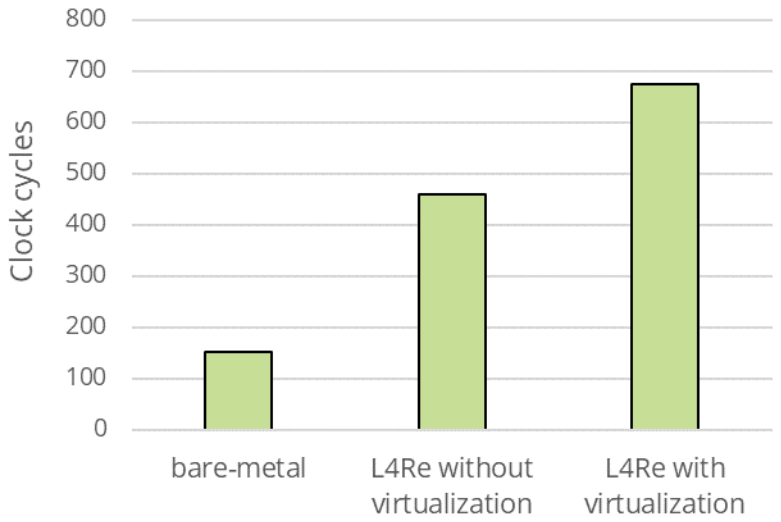
  7.02 x performance improvement



Execution time per 100 hardware task instances

Evaluation: Overhead





Overhead induced by spatial scheduling

Spatial scheduling: once per task

Overhead induced by temporal scheduling

Temporal scheduling: once per task instance

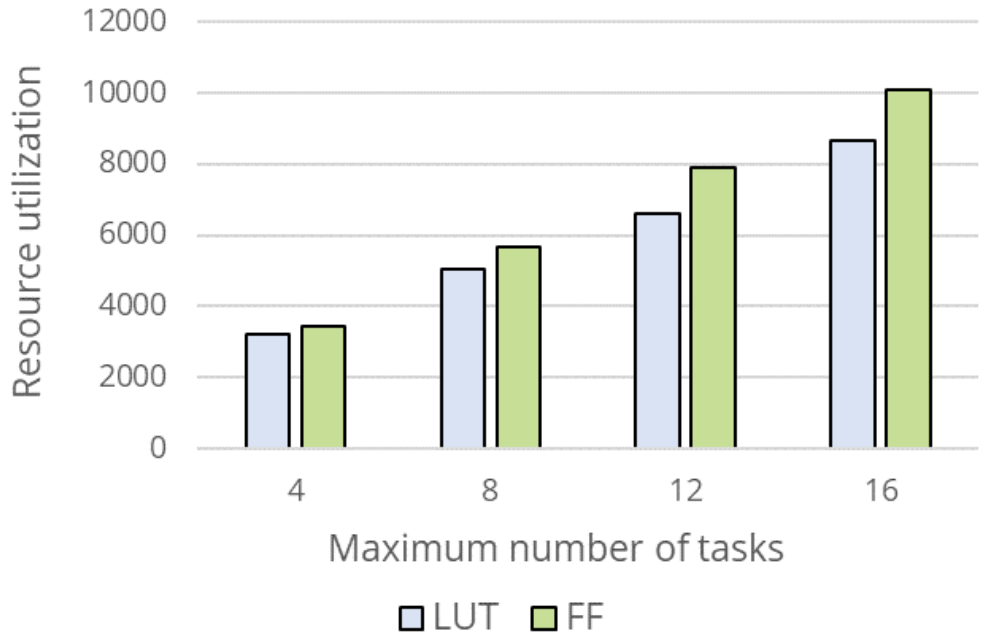
Virtualization increases overhead by 1.47 times.

Evaluation: Resource Consumption

Resource consumption of spatial scheduler
- Grows proportional to maximum number of tasks
- 4 DSPs

PLMMU (max. 16 tasks):
- 2966 LUTs
- 1345 FFs

EDF (per accelerator, max. 3 tasks):
- 470 LUTs
- 628 FFs



Conclusion

Hardware-level control mechanism for memory-mapped communication

- Prevention of unauthorized access
- Shared usage of hardware accelerators with spatial and temporal scheduling
- Preservation of priorities
- Latency reduction compared to a software approach by 7.02 times

Future work

- Virtualization of interrupts
- Exchange of hardware accelerators (dynamic function exchange)
- More complex use cases

Questions ?

Remarks

Discussion