ReconROS Executor:

Event-Driven Programming of FPGA-accelerated ROS 2 Applications

Christian Lienen and Marco Platzner

Computer Engineering Group
Paderborn University

{christian.lienen,platzner}@upb.de

PADERBORN
UNIVERSITY

Motivation

- High demand for computational power in robotics applications
  - – energy sensitive due to limited battery capacity
  - o e.g., visual navigation or object detection algorithms in mobile robots
- ➢ Platform FPGAs are promising candidates

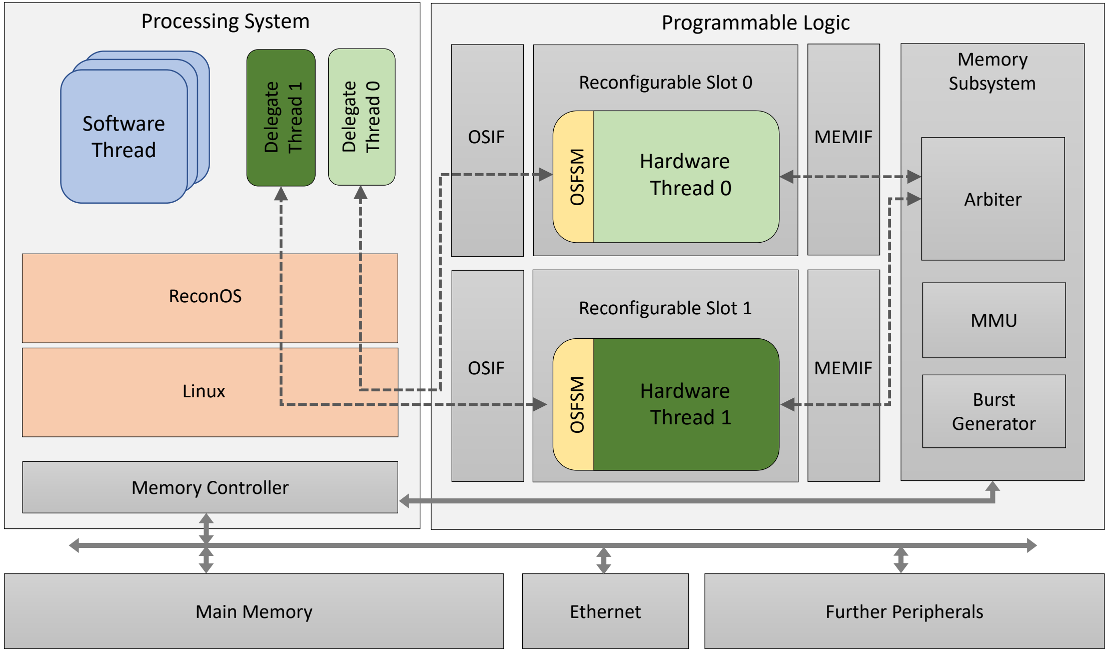
- Reconfigurable fabric limited in resources
  - o Application might not fit as whole
  - o Time sharing of resources needed
- ➢ Modern Platform FPGAs provide dynamic partial reconfiguration properties

❖ How to use hardware acceleration in robotics in combination with partial reconfiguration?

ReconOS | Architecture

ReconOS introduces **HW threads**

- Areas in the reconfigurable fabric
- Combined with a *delegate* software thread
  - Executes SW functionality on behalf of the hardware thread
- Access via the OSIF and MEMIF interface
  - **OSIF** (*Operating System Interface*): Communication channel between hardware thread and delegate
  - **MEMIF** (*Memory Interface*): Allows for access to the shared virtual address space

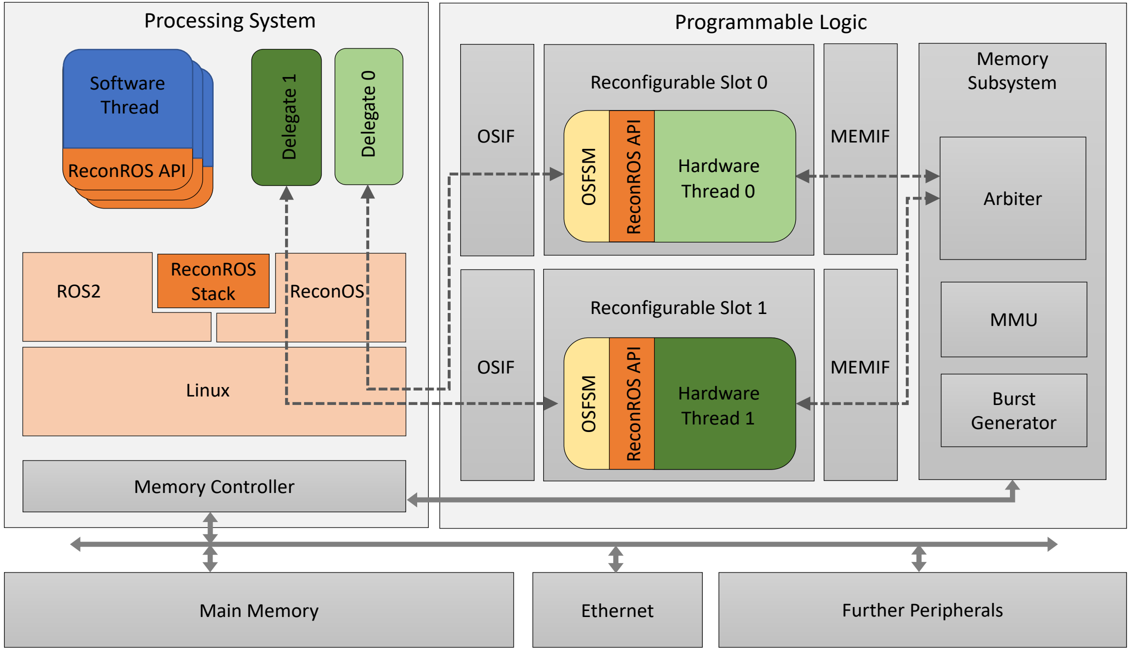


ReconROS | Architecture

ReconROS extends ReconOS by ROS 2 support

- Usage of Pub/Sub Communication, ROS 2 Services and Actions in hardware threads
- Consistent programming model for hardware and software nodes
- Support of VHDL and high-level synthesis (C/C++)
- Tool flow supports predefined and user-defined message types

➢ Hardware nodes can be shifted easily between hardware and software domain



Design Considerations about Hardware Callbacks

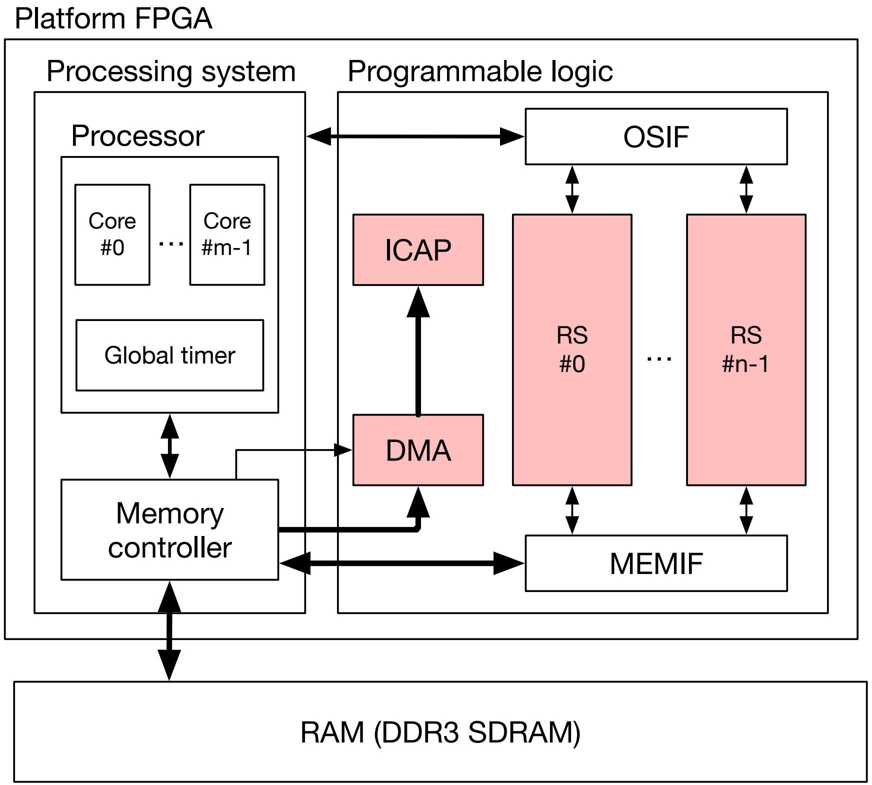
- Open question: how to leverage partial reconfiguration in robotics applications?
  - Exchange of which parts of the system?
  - When to start the reconfiguration process?

- Introduction of ***hardware callbacks***
  - Event-driven programming de-facto standard in the ROS world
  - Hardware threads which start after certain events
    - Like software callbacks using rclcpp / rclpy
    - Message Object available at callback start time (except for timer events)
    - Callback terminate after execution
      - Termination signaled using OSIF interface

ReconROS Executor Hardware Architecture

- Platform FPGA provides m CPU cores for software callback execution
  - During design time, n reconfigurable slots (RS) are specified
  - Slots can differ in size / resources and can have additional interfaces as well

- Direct Memory Access Unit for bitstream transfer
  - Configuration data is transferred from main memory to ICAP (Internal Configuration Access Port)
  - Fast (up to 400 MBit/s) transmission without CPU load

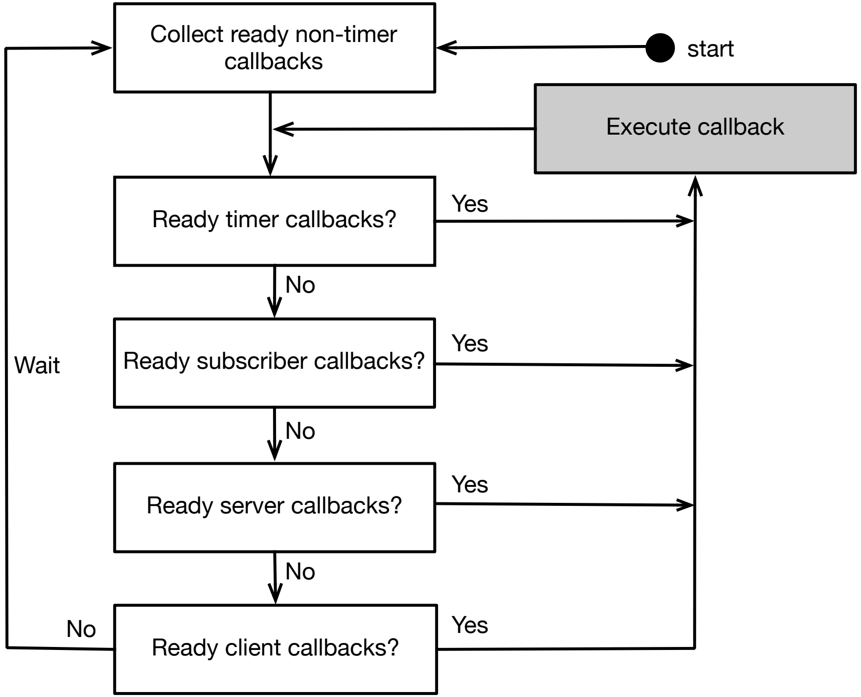


ROS 2 Standard Executor

- Event-Driven programming common methodology in ROS 2 rclcpp / rclpy applications
  - ➢ callbacks are triggered by timers, subscribers, service servers or service clients

- Standard scheduling of callbacks by the ROS 2 Standard Executor
  - – ROS specific order of execution
  - – ROS timers privileged (at least until “dashing”)

- Available as single-threaded and multi-threaded version



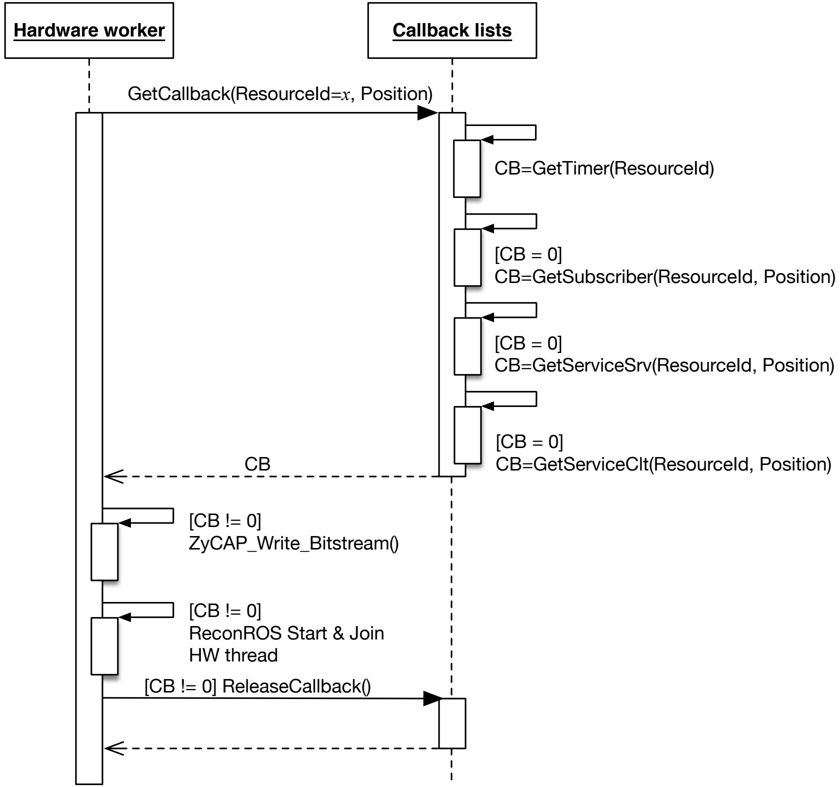
Scheduler

- Scheduling principle is adapted from the standard ROS 2 scheduler
  - Privileged Timer (always checked)
  - Other object are checked in order started at the last handled object (position variable)

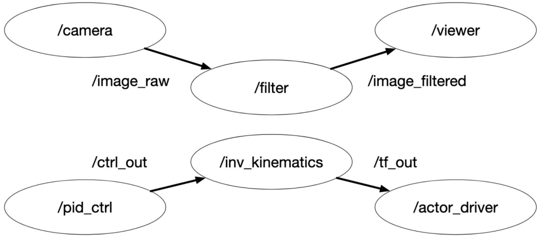
- After callback selection, reconfigurable slot is eventually reconfigured (depends on last executed callback)

- Worker blocks until execution, releases callback instance afterwards

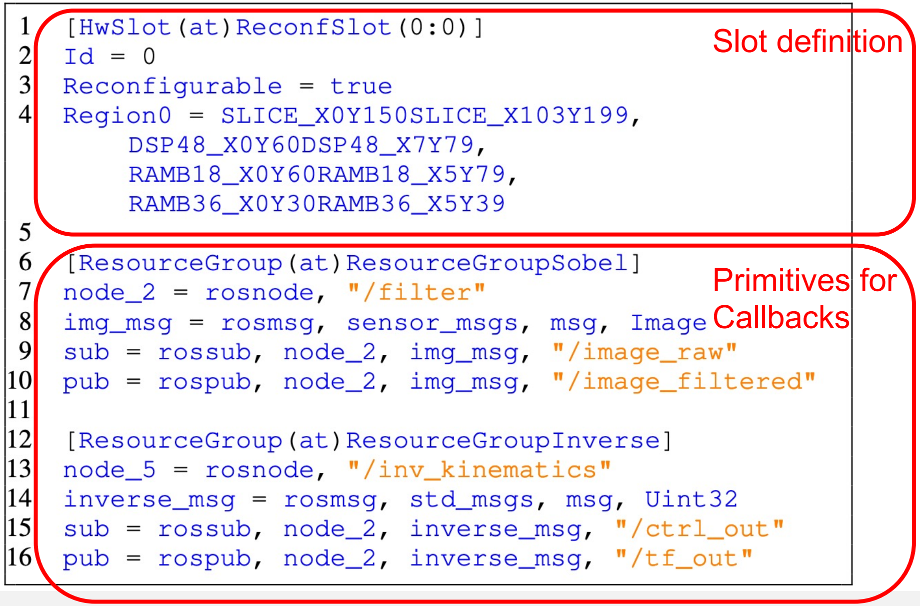
- One Linux thread per reconfigurable slot


Hardware worker
Callback lists
GetCallback(ResourceId=x, Position)
CB=GetTimer(ResourceId)
[CB = 0]
CB=GetSubscriber(ResourceId, Position)
[CB = 0]
CB=GetServiceSrv(ResourceId, Position)
[CB = 0]
CB=GetServiceClt(ResourceId, Position)
CB
[CB != 0]
ZyCAP_Write_Bitstream()
[CB != 0]
ReconROS Start & Join
HW thread
[CB != 0] ReleaseCallback()

Design Example



ReconROS configuration file

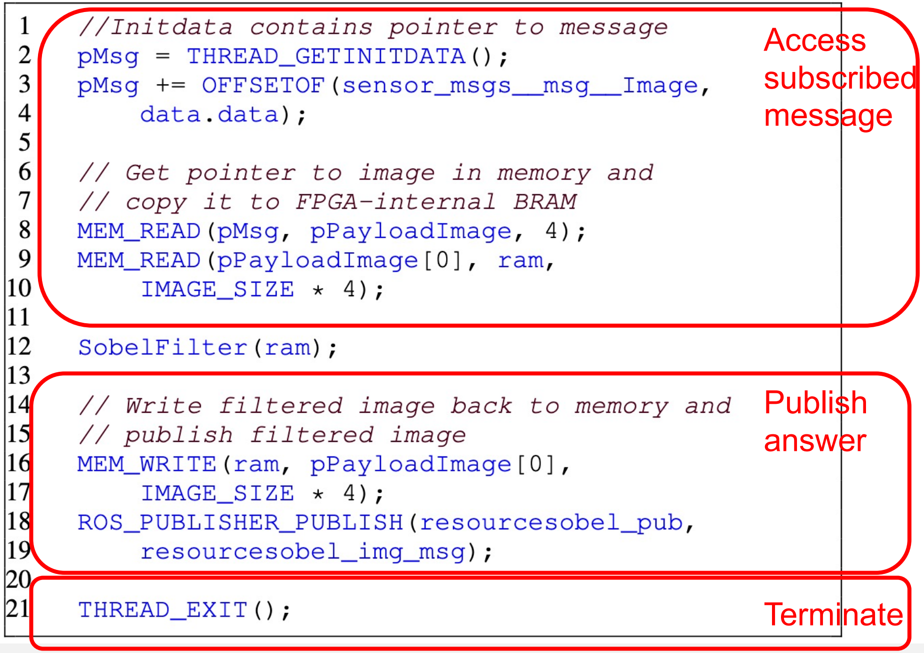


```
[HwSlot(at)ReconfSlot(0:0)]
Id = 0
Reconfigurable = true
Region0 = SLICE_X0Y150SLICE_X103Y199,
    DSP48_X0Y60DSP48_X7Y79,
    RAMB18_X0Y60RAMB18_X5Y79,
    RAMB36_X0Y30RAMB36_X5Y39

[ResourceGroup(at)ResourceGroupSobel]
node_2 = rosnode, "/filter"
img_msg = rosmsg, sensor_msgs, msg, Image
sub = rossub, node_2, img_msg, "/image_raw"
pub = rospub, node_2, img_msg, "/image_filtered"

[ResourceGroup(at)ResourceGroupInverse]
node_5 = rosnode, "/inv_kinematics"
inverse_msg = rosmsg, std_msgs, msg, Uint32
sub = rossub, node_2, inverse_msg, "/ctrl_out"
pub = rospub, node_2, inverse_msg, "/tf_out"
```

Slot definition

Primitives for Callbacks

Exemplary callback code



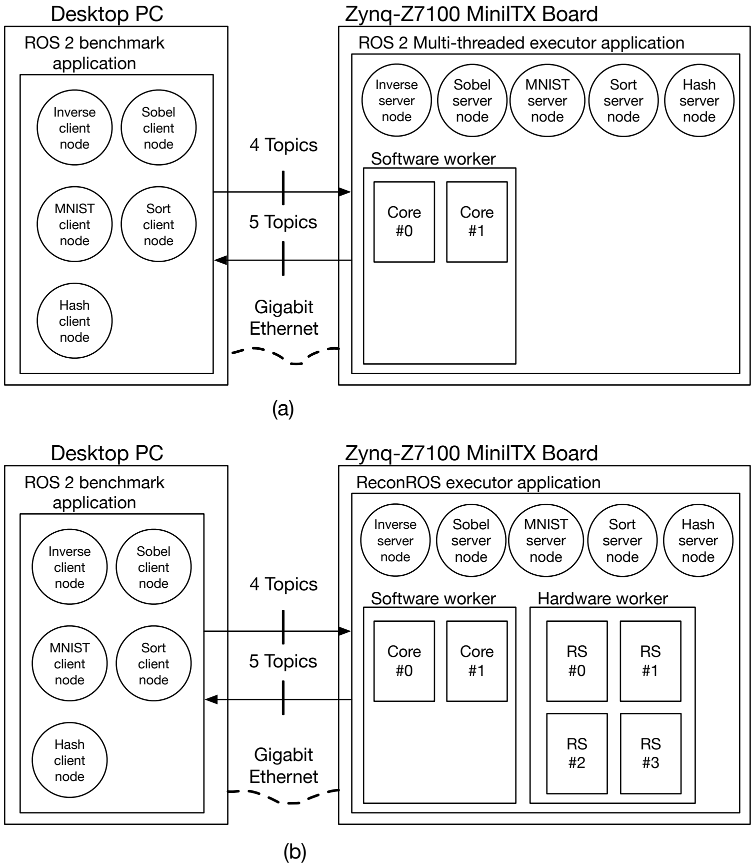
```
//Initdata contains pointer to message
pMsg = THREAD_GETINITDATA();
pMsg += OFFSETOF(sensor_msgs__msg__Image,
    data.data);

// Get pointer to image in memory and
// copy it to FPGA-internal BRAM
MEM_READ(pMsg, pPayloadImage, 4);
MEM_READ(pPayloadImage[0], ram,
    IMAGE_SIZE * 4);

SobelFilter(ram);

// Write filtered image back to memory and
// publish filtered image
MEM_WRITE(ram, pPayloadImage[0],
    IMAGE_SIZE * 4);
ROS_PUBLISHER_PUBLISH(resourcesobel_pub,
    resourcesobel_img_msg);

THREAD_EXIT();
```

Access subscribed message

Publish answer

Terminate

Evaluation



- Client application sends requests to server application, waits for the response
  - Procedure is repeated 1000 times

- Reference server application based on five nodes
  - Each node has one callback
  - Timer, Pub/Sub and Service Node

- Two versions of server application implemented
  a) standard ROS 2 multi-threaded executor
  b) ReconROS executor with four reconfigurable slots

Callback Speedups / Reconfiguration Costs

ROS 2 Callback	Speedup
Sobel filter	2.5
Number sorting	48.2
MNIST Classifier	1.4
Inverse Kinematics	4.3
Hash Calculation	1.2

- Comprises all operations done by the ReconROS Executor

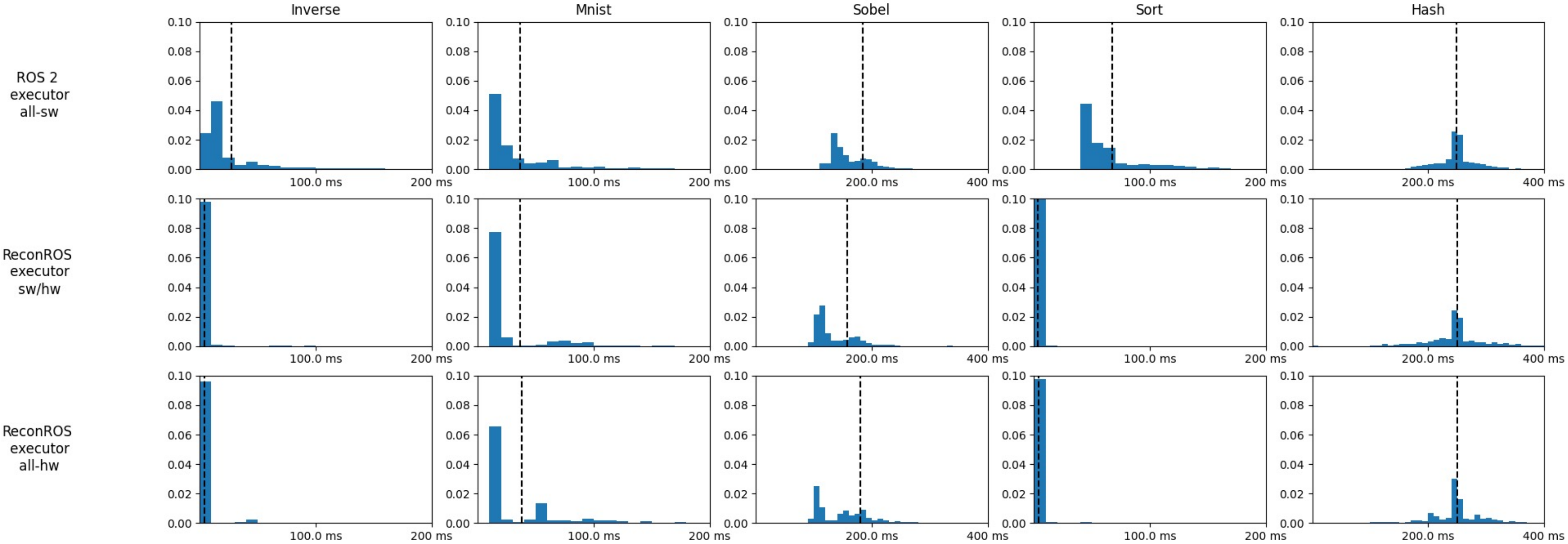
➢ Speedup for all five application examples

Slot	Size [MB]	Reconfiguration Time
0	2.8	24.0 ms
1	2.8	24.0 ms
2	5.2	38.4 ms
3	4.8	36.9 ms

➢ Reconfiguration time higher then expected

- 160 MB/s instead of 400 MB/s
- DMA driver improvements required

Results



- Speedup for overall ROS 2 node follows the trends for the callback's measurements in isolation
- Distribution of the time triggered callback (Hash) like ROS 2 standard executor
- Probably, the DDS layer remains as the bottleneck in this case
  - Reconfiguration times in all-hw configuration close the sw/hw mixture

Conclusion

Summary

- ReconROS Executor provides a solution for hardware accelerator scheduling
- Brings the event-driven programming paradigm to hardware

Future Work

- More advanced scheduling strategies
  - E.g., including migrations between hardware and software
- Integration in the standard ROS 2 executor

Thank you for your attention!
If you have any questions or feedback / comments,
feel free to contact me via mail:
christian.lienen@upb.de

https://github.com/Lien182/ReconROS

(Code, Demos, Board Images,…)

[1] Christian Lienen, Marco Platzner and Bernhard Rinner. "ReconROS: Flexible Hardware Acceleration for ROS2 Applications", International Conference on Field Programmable Technology (ICFPT), 2020.

[2] Christian Lienen and Marco Platzner. "Design of Distributed Reconfigurable Robotics Systems with ReconOS“, ArXiv paper under review, 2021.

[3] Christian Lienen and Marco Platzner. "ReconROS Executor: Event-Driven Programming of FPGA-accelerated ROS 2 Applications." arXiv preprint arXiv:2201.07454 (2022).