A Hardware Design Environment for ROS2 Node to FPGA-Integrated SoC

Xingze Li

Graduate School of Informatics
Nagoya University
Nagoya, Aichi 464-8601

Ryota Yamamoto

Department of Engineering for Innovation
National Institute of Technology, Tomakomai College
Tomakomai, Hokkaido 059-1275
r-yamamoto@tomakomai-ct.ac.jp

Shinya Honda

Graduate School of Informatics
Nagoya University
Nagoya, Aichi 464-8601
honda@ertl.jp

Abstract— As robotic applications increasingly demand both high performance and energy efficiency, FPGA-integrated system-on-chip (SoC) platforms have emerged as a promising solution for offloading intensive workloads. Despite their potential, integrating hardware (HW) accelerators with ROS2-based applications remains a significant difficulties, often requiring deep domain expertise. In this paper, we present CWB2ROS, a development framework that enables seamless integration between ROS2 nodes and HW modules synthesized via high-level synthesis (HLS) using Cyber Work Bench (CWB). CWB2ROS supports core ROS2 communication models, including Publish/Subscribe, Service and Parameters, and automates the generation of necessary software wrappers. Experimental results on the Kria KR260 platform demonstrate that CWB2ROS achieves competitive communication latency and offers greater flexibility compared to existing solutions.

I. Introduction

In recent years, FPGA-integrated SoC (System on Chip) devices have been increasingly used to accelerate processing and improve energy efficiency, especially for compute-intensive workloads such as machine learning [1]. In the field of robotics, such heavy computations are also frequently employed for tasks like image recognition and map generation. For example, if map generation is delayed, it can interfere with the robot behavior.

The Robot Operating System (ROS) has become a de facto standard in robotics software development [5], offering modular components and robust communication frameworks. In ROS, functionalities are implemented as "nodes" that exchange data through well-defined interfaces.

Despite the suitability of FPGAs for performance-critical tasks, their adoption in robotics is hindered by the complexity of hardware (HW) design and integration. Developing HW accelerators typically involves hardware description languages (HDLs) and detailed platform knowledge. While high-level synthesis (HLS) offers a more accessible development pathway using languages like C/C++, it still demands expertise in software-hardware interfacing, particularly when integrating with ROS2 environments. Due to this complexity, there is a substantial barrier for both robotics engineers unfamiliar with HW, and HW designers unfamiliar with ROS2.

To address these development problem, we propose CWB2ROS, a development framework that enables seamless integration of C/C++ based HLS modules generated using Cyber Work Bench (CWB) into ROS2 systems. By automatically generating interface nodes that expose HW functionality through ROS2 interfaces, the framework allows HW components to be used as conventional software nodes. The main contributions of this work are:

- A seamless integration flow between CWB-generated HW modules and ROS2 applications
- Support for core ROS2 communication models: Publish/Subscribe, Service, and Parameter
- A comparative evaluation highlighting the performance and development benefits over existing tools

This framework is designed to bridge the gap between HW developers and robotics developers, enabling efficient co-design and prototyping of FPGA-accelerated robotic systems.

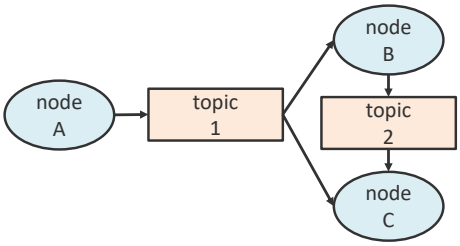


Fig. 1.: Publisher/Subscriber communication.

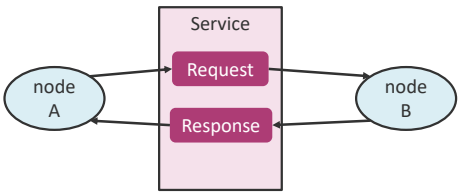


Fig. 2.: Server Client communication.

II. Related Work

A. *ROS2*

ROS2 (Robot Operating System 2) is a modular and open-source middleware framework widely adopted in robotic systems. It introduces several layers of abstraction, such as plumbing, tools, capabilities, and ecosystem. ROS2 provides node-based application design.

A ROS2 node represents a unit of functionality. Nodes interact using four main communication mechanisms:

- Publish/Subscribe (Pub/Sub): A many-to-many asynchronous communication model where publisher nodes transmit messages to topics, and subscriber nodes receive them. As illustrated in Fig. 1, node A publishes to topic 1, which is subscribed to by nodes B and C. Separately, node B publishes to topic 2, which is received by node C.
- Service: A synchronous client-server model. A client node sends a request, and the server node responds after processing (see Fig. 2).
- Parameters: Runtime-configurable key-value settings that can influence node behavior.
- Actions: A combination of Pub/Sub and Service used for managing long-running tasks asynchronously.

In this paper, we focuses three communication: Pub/-Sub, Service and Parameters in ROS2 applications.

B. *FPGA-integrated SoC*

FPGA-integrated System-on-Chip (SoC) platforms, such as Kria series (AMD Inc.), integrate reconfigurable logic with general-purpose processing units. These platforms are especially suitable for robotics and computer vision, where tasks like image processing and simultaneous localization and mapping (SLAM) impose both performance and power constraints.

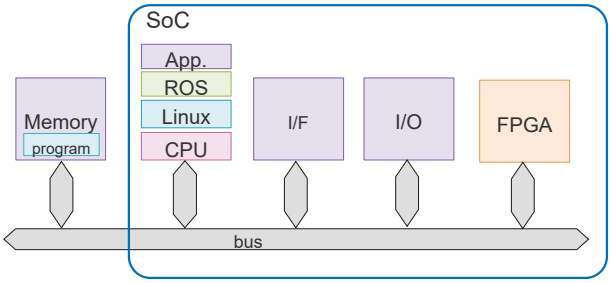


Fig. 3.: An example of FPGA-integrated SoC architecture.

A typical FPGA-integrated SoC architecture is shown in Fig. 3. Here, the CPU and FPGA fabric communicate via system bus, with the CPU typically running a Linux-based OS and orchestrating I/O and application logic. HW modules are used to offload performance-critical operations from the CPU to the FPGA, improving overall system efficiency.

Designing in such environments, however, it is difficult for developers to design low-level memory-mapped interfaces, custom drivers, and synchronization mechanisms. These requirements are often outside the skill set of robotics software developers.

In this study, we targets Kria KR260 (AMD Inc.), a board designed for robotics applications.

C. *High Level Synthesis*

To mitigate the complexity of HW design, HLS tools allow HW functionality to be described using high-level languages such as C or C++. For example, there are two HLS tools:

Vitis HLS (AMD): Offered as part of the Vitis development environment, Vitis HLS is restricted to AMD devices but integrates with the Kria Robotics Stack (KRS).

Cyber Work Bench (CWB, NEC): A commercial tool that is FPGA vendor-agnostic. CWB accepts input in BDL, SystemC, C, or C++, and provides RTL output with optimizations such as loop unrolling and memory mapping. CWB also includes an automatic interface generator that maps function arguments to registers and memory accessible over the AXI bus. It is an essential feature for integration with CPU software [3].

This study adopts CWB to support a wide range of FPGA-integrated SoCs. The design description written in C or C++.

D. ROS Integration for FPGA-integrated SoCs

D.1. Vitis HLS

For Kria Zynq devices targeted in this study, AMD provides a toolchain known as the Kria Robotics Stack (KRS). KRS facilitates the development of ROS2 applications on FPGA-integrated SoCs by leveraging high-level synthesis (HLS). It offers an integrated environment that combines robotics-oriented libraries and utilities, thereby accelerating the prototyping and deployment of industrial robotic applications.

A development with KRS requires a comprehensive understanding of the ROS2 development process. KRS extends the standard ROS2 build system by incorporating HLS-based IP core generation into the ament and colcon build workflows. While this integration aligns with conventional ROS2 application development, it imposes substantial cognitive and technical demands on developers who lack prior experience with ROS2 or its associated build infrastructure.

Addtionally, KRS development has been inactive for approximately two years. It supports only AMD FPGA models compatible with Vivado 2022.1 and does not support other FPGA vendors or newer devices [4]. Furthermore, its performance and flexibility have not been systematically evaluated in comparison with alternative solutions.

D.2. Other Research Examples

Other notable approaches for integrating ROS2 and FPGA include: Nyboe et al. proposed an FPGA acceleration solution for drones that facilitates seamless data exchange between ROS2 nodes and HW tasks [6]. Vilches et al. developed a platform-independent architecture that improves real-time responsiveness in robotic systems [7]. Lienen et al. demonstrated HW-executed ROS2 nodes using ReconOS, validating coordination between FPGA logic and ROS2 environments [8].

In contrast to these, our proposed tool, CWB2ROS, aims to support a range of FPGA-integrated SoCs and reduce development complexity by automating node integration based on C code.

III. CWB2ROS: ROS Node Development Tool for HW

In this study, we propose a tool called CWB2ROS, which facilitates the integration of ROS2 nodes with HW modules generated by Cyber Work Bench (CWB). This tool automates the generation of ROS2 node definitions and build system files necessary for interacting with the HW, thereby streamlining the development process. To support a broad range of FPGA-integrated SoCs, the tool adopts CWB as its high-level synthesis (HLS) backend.

The tool overview for implementing CWB2ROS are defined as follows:

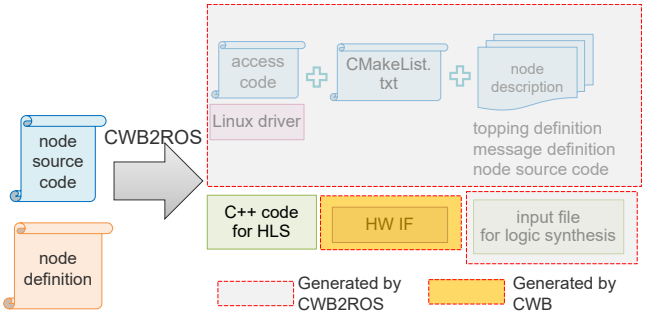


Fig. 4.: CWB2ROS: overview.

Spec. 1: Supports ROS2 communication mechanism.

Spec. 2: Maps inputs and outputs of HW modules synthesized by HLS to ROS2 message interfaces.

Spec. 3: Requires the user to provide only a node definition file and C++ source code for the HW function.

Spec. 4: Utilizes the HW I/O auto-generation feature provided by CWB.

Spec. 5: Follows the standard ROS2 development flow.

Based on these specifications, the proposed architecture of CWB2ROS is shown in Fig. 4.

A. ROS2 Communication Plumbing in CWB2ROS

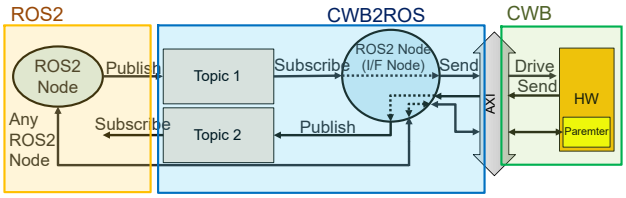
To satisfy the above specifications, CWB2ROS supports two major ROS2 communication models: Publish/-Subscribe and Service.

In Publish/Subscribe communication, data is asynchronously transmitted between publisher and subscriber nodes. A software node (referred to as an interface node), which is automatically generated and performs as a communication bridge between ROS2 and the HW module, is used to handle communication with the HW. This interface node receives a topic from a ROS2 node, forwards it to the HW, waits for the HW to complete processing, and then publishes the result on another topic.

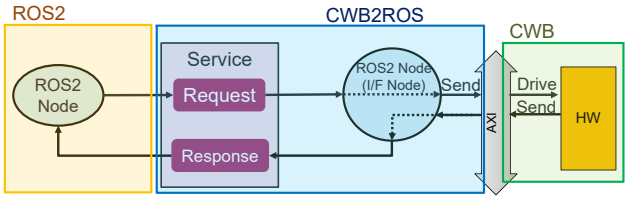
In Service communication, the interface node performs as a ROS2 server node. When a request is received from a client node, it invokes the HW module with the input arguments, waits for the result, and returns it as a service response.

This abstraction allows other ROS2 nodes to interact with HW-backed processing as if they were communicating with standard software ROS2 nodes.

The corresponding communication flows, including key interactions such as “DriveSend” and “Polling” via AXI, are illustrated in Fig. 5a for Pub/Sub and Fig.5b for Service communication.



(a) Pub/Sub Communication



(b) Service Communication

Fig. 5.: Communication Flow

B. *Interface node*

The interface node is automatically generated based on a configuration file and functions as a ROS2 node that communicates with the HW via the AXI bus.

In the Publish/Subscribe model (see Fig. 5a), the interface node subscribes to a topic published by another ROS2 node. Upon receiving a message, it transmits the input data to the HW through the AXI bus using a Linux driver. The HW, upon completing its task, sends the result back via the AXI bus. The interface node then publishes this result on a new topic. Data reception is implemented via polling. Currently, polling is used for simplicity, though interrupt-based or DMA-based implementations are possible and are left as future work.

In the Service model (see Fig. 5b), the interface node receives a service request and passes the input arguments to the HW. After processing, the HW returns the result via the AXI bus, and the interface node responds to the client with this data. As with Pub/Sub, data reception is handled via polling. Currently, polling is used for simplicity, with other methods considered as future improvements.

Although HW-side interface descriptions must be written by the developer, CWB allows this to be done by simply annotating function arguments, making the process manageable even for those with limited HW experience.

C. *Design Flow and Node Definition*

The development flow when using CWB2ROS is illustrated in Fig. 6, which outlines both software and HW build steps such as `make`, `make fpga`, and `colcon` integration. The developer is responsible for preparing two items:

1. A YAML-based node definition file, which describes the interface node and associated topics.
2. A C or C++ source file that defines the HW function, including its input/output interface.

Listing 1: Definition described in YAML

```yaml
SysName: array_pubsub
CWB_WS:
  project: array
  function: AddSub_array #top function
  cpp: true
  arglist: [a, b, sub, res] #args of top function
  file: [add_sub_arr_top.cpp, impl/add_sub_arr.cpp]
  incpath: [impl]
ROS2ComType: PubSub
Node:
  name: array_subscriber
ArgTopic:
  name: Topic_array_arg
  msg_name: ArgArraySubT
  qos: QoS
ReturnTopic:
  name: Topic_array_ret
  msg_name: RetArraySubT
  qos: QoS
Arg:
  - {name: a, type: int8_t, depth: 16}
  - {name: b, type: int16_t, depth: 16}
Return:
  - {name: b, type: int16_t, depth: 16}
  - {name: res, type: int32_t, depth: 16}
Parameter:
  - {name: sub, type: bool, initval: false}
```

Because the direction (input or output) cannot be inferred from the CWB HW description alone, the node definition file must explicitly specify this along with the communication method (Pub/Sub or Service), topic names, and QoS settings.

During the build process:

- For software, CWB2ROS generates the CMakeLists.txt, interface node C++ code, and a Makefile from the YAML node definition. These are built using colcon, the standard ROS2 build tool.
- For HW, the C/C++ description is passed to CWB for HLS, generating RTL files. These RTL files are then passed to logic synthesis tools such as Vivado to generate the final FPGA bitstream for deployment.

Because the HW is described in C/C++, simulation-based verification is also easily performed. This design flow supports seamless integration of HW into ROS2 systems, leveraging familiar development tools and processes.

IV. Evaluation

We evaluated the performance and practicality of CWB2ROS by comparing it with KRS. Our evaluation focuses on two aspects:

- Communication latency between ROS2 nodes and the HW module
- Qualitative comparison regarding development difficulty and supported platforms

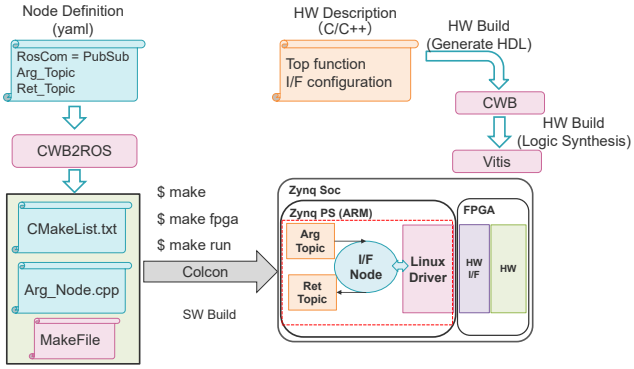


Fig. 6.: CWB2ROS Tool Flow.

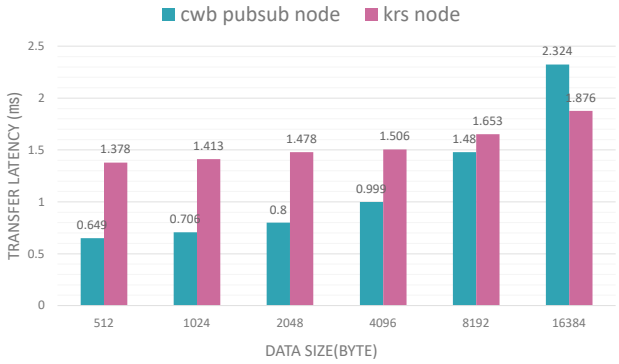


Fig. 7.: Communication latency vs. data size for CWB2ROS and KRS.

TABLE I
: Exprimental Setup

Target board	Kria KR260
KRS version	2022.1
HLS Tool	CWB
Logic Synthesis	Vivado 2022.1
Linux (KRS)	Petalinux
Linux (CWB2ROS)	Ubuntu 22.04

A. *Experimental Setup*

The evaluation environment is configured as TABLE I. Note that the choice of Vivado version (2022.1) was determined by KRS compatibility constraints.

For latency measurements, we used a ROS2 node that sends a topic to initiate processing and another node that subscribes to the result. We measured the time elapsed from when the first node publishes a message to when the subscriber node receives the response topic after HW processing.

The HW module used for this test simply copies the input data and returns it without any additional computation, to isolate the communication latency.

B. *Latency Result*

Figure 7 shows the measured communication latency as a function of data size (in bytes). The x-axis is plotted on a linear scale, and the y-axis represents the end-to-end transfer latency in milliseconds. CWB2ROS shows better performance than KRS for data sizes of 8KB or less. However, for data sizes larger than 8KB, KRS achieves lower latency.

We attribute this to the fact that KRS uses burst-mode transfers via the Xilinx Runtime (XRT) library, which enables efficient communication via burst-mode transfer. In contrast, CWB2ROS currently uses a single data transmission mechanism via AXI, which becomes less efficient as data size increases.

These results suggest that for small-to-moderate data sizes, CWB2ROS provides lower communication latency. To improve performance for larger data sizes, we plan to support DMA or interrupt-based transfer mechanisms in the future.

C. *Qualitative Comparison*

A qualitative comparison between KRS and CWB2ROS is shown in Table II, focusing on required knowledge and toolchain compatibility.

As shown in Table II, KRS requires developers to be familiar with the internal workings of the ROS2 build system, including the use of extended CMake configurations specific to KRS. This makes the development process more complex, particularly for HW engineers who are not accustomed to ROS environments. In contrast, CWB2ROS abstracts away most of the ROS2-specific configurations and requires only a simple YAML-based node definition and C/C++ source code, lowering the entry barrier significantly.

In terms of implementation complexity, CWB2ROS significantly reduces the development burden. For instance, integrating a HW module with one input and one output requires only 25 lines of YAML-based configuration, compared to approximately 54 lines in KRS. More importantly, KRS demands advanced knowledge of C++ templates, ROS2 node structure, and OpenCL-based memory management. For example, defining a publisher in KRS entails invoking create_publisher with appropriate type parameters, which may be challenging for HW developers. By contrast, our approach allows users to specify only essential elements—such as communication type, topic names, and argument data types—without requiring in-depth expertise in ROS2 internals or OpenCL semantics. This substantially lowers the barrier to entry for integrating HW into ROS2 systems.

With respect to toolchain compatibility, KRS is limited to AMD FPGAs that are supported by Vivado 2022.1, and thus cannot be used with newer FPGA models or alternative synthesis tools. CWB2ROS, however, is designed to be toolchain-agnostic and can support a wider

TABLE II
: Comparison of KRS and CWB2ROS

Aspect	KRS	CWB2ROS
Required Knowledge	Familiarity with ROS 2, custom build configurations, and KRS-specific CMake.	Basic YAML node definition and C/C++ hardware description.
Lines of Code for Node Definition and Communication (1 input, 1 output)	54 lines total: 12 for OpenCL init, 9 for memory mapping, 13 for callback, 12 for publishing, 8 for main.	25 lines total: 8 for HW top function, 1 for communication type, 7 for args, 6 for return, 2 for node.
Toolchain Compatibility	Only supports AMD FPGAs with Vivado 2022.1.	Supports various SoCs depending on the logic synthesis tool.

Example Code (KRS) is refered from `https://github.com/ros-acceleration/acceleration_examples/blob/main/nodes/doublevadd_publisher/src/doublevadd_component_fpga.cpp`

range of FPGA-integrated SoCs, provided that the logic synthesis tool can handle RTL generated by CWB. While some adaptation may still be necessary for non-AMD platforms, the proposed tool offers much greater flexibility in this regard.

V. Summary and Conclusions

This paper proposed CWB2ROS, a development environment that enables seamless integration of HW modules synthesized via HLS into ROS2-based robotic applications. By leveraging CWB, the tool allows developers to describe HW behavior in C/C++ and automatically generate the necessary ROS2 interface components, including communication nodes and build configurations.

CWB2ROS supports Pub/Sub, Client-Server and Parameter communication mechanisms and abstracts the complexity of hardware-software integration through auto-generated interface nodes. The proposed development flow reduces the barrier to entry for developers with either HW or software backgrounds and improves reusability across FPGA platforms.

In our evaluation using the Kria KR260 board, CWB2ROS demonstrated lower communication latency than KRS for data sizes up to 8KB, while maintaining a significantly simpler development process. Although latency increased for larger data sizes due to the polling-based transfer mechanism, this limitation is addressable by supporting DMA in future work.

Compared to KRS, CWB2ROS also offers broader platform compatibility, as it is not tied to a specific version of Vivado or a particular FPGA vendor. This makes it suitable for use with newer and more diverse FPGA-integrated SoCs.

In summary, CWB2ROS Simplified HW/ROS2 integration through automatic node generation and lower communication latency. Furthermore, CWB2ROS moderate data transfers, and enhanced portability across FPGA platforms.

As future work, we plan to extend the tool to support burst-mode and DMA-based communication to further reduce latency for large data transfers. Additionally, expanding support for other ROS2 communication models (e.g., Actions) is also under consideration.

Acknowledgements

This presentation is based on results obtained from a project, JPNP23015, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma S. Vrudhula J. Seo, Jae-sun, Y. Cao "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," *Proceedings of 24th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2016)*, pp. 16–25, 2016.

[2] S. Mittal "A survey of FPGA-based accelerators for convolutional neural networks," *Neural computing and applications, Springer*, Vol. 32, No. 4, pp. 1109–1139, 2020.

[3] NEC "CyberWorkBench: Products — NEC, " Available: `https://www.nec.com/en/global/prod/cwb/` (cited 2025–05–27).

[4] AMD "Kria Robotics Stack (KRS) – KRS 1.0 documentation", Available: `https://xilinx.github.io/KRS/sphinx/build/html/index.html` (cited 2025–05–27).

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, 2022.

[6] F. F. Nyboe, N. H. Malle, E. Ebeid "MPSoC4Drones: An open framework for ROS2, PX4, and FPGA integration," *The 2022 International Conference on Unmanned Aircraft (ICUAS '22), IEEE*, pp. 1246–1255, 2022.

[7] V. Mayoral-Vilches, S. M. Neuman, B. Plancher V. J. Reddi "Robotcore: An open architecture for hardware acceleration in ROS 2," *The 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022), IEEE/RSJ*, pp. 9692–9699, 2022.

[8] C. Lienen, M. Platzner, B. Rinner "Reconros: Flexible hardware acceleration for ros2 applications," *The 21st International Conference on Electronics Packaging Technology (ICEPT 2020), IEEE*, pp. 268–276, 2020.