Interim Report: Software-Hardware Integration framework for Galassia-5 Subsystems

Executive Summary

This project focuses on building a communication integration backbone for the GALASSIA-5 satellite payload, so that three key subsystems — the camera payload, the AI module, and the Ku-band radio can exchange data reliably through a common, well-defined interface.

In the mission's current development process, each subsystem is designed and tested largely in isolation. The camera team focuses on imaging, the AI team on models and accelerators, and the RF team on Ku-band downlink. There is no unified communication layer or early integration platform, which means that compatibility issues (data formats, timing, throughput, error handling) may only surface very late, when all hardware is assembled. This is risky for a flight system and makes debugging much harder.

The payload computer is based on a Zynq UltraScale+ MPSoC (ZSOM-F01 SoM board), which combines a quad-core ARM Cortex-A53 Processing System (PS) with FPGA Programmable Logic (PL). The long-term goal of this project is to use Zynq platform to:

1. Provide a common communication layer that connects the camera, AI module and Ku-band radio.
2. Allow the PS software to coordinate these subsystems through a software integration layer (command, telemetry, status, data paths).
3. Use a space-grade protocol (SpaceWire) in the PL as the backbone for data movement where required.

.A key constraint is that the camera payload only supports SpaceWire as its primary high-speed digital interface. Therefore, as a first concrete step, this interim stage focuses on:

- Implementing and understanding SpaceWire in the PL using the open-source SpaceWire Light core.

- Prototyping bidirectional communication over SpaceWire between a "camera node" and a "payload computer node" in Vivado simulation.
- Studying the Zynq PS–PL architecture to prepare for AXI-Lite control and future software integration of AI and Ku-band.

A1. Objective and scope

Role and Responsibilities

Within the GALASSIA-5 team, my role is subsystem communication and software integration. Specifically, I am responsible for designing the communication backbone that connects the camera payload, the on-board AI module, and the Ku-band radio to the ZSOM-F01 payload computer.

- Implementing and validating the link-layer and register-level logic required to make these subsystems talk to each other.
- Building testbenches, simulations and (later) hardware tests that can be used to verify integration before full mission hardware is complete.

My work sits at the intersection, making sure that when all subsystems are ready, there is a coherent way for data and commands to flow through the payload computer.

Mission Problem

The core integration problem can be summarised as:

- Each major subsystem such as the camera, AI, Ku-band is currently developed and tested on its own. There is no shared communication layer or early integration platform to ensure they will work together when everything is plugged into the payload computer.

This leads to several risks:

- Late discovery of integration bugs (data format mismatches, timing assumptions, throughput limits).
- Difficult debugging, because once all hardware is stacked together, it is harder to isolate which part is failing.
- Inefficient test cycles, since every subsystem team must work with incomplete assumptions about how they will interact with others

For the camera specifically, the problem is even more concrete:

- Its only high-speed digital interface is **SpaceWire**.
- Without a SpaceWire implementation on the payload computer, the camera cannot even participate in early integration tests

Objective

The main objective of this project is:

- To design and implement a software-driven communication integration layer on the ZSOM-F01 payload computer that can connect the camera (via SpaceWire), AI module, and Ku-band radio into a coherent subsystem network.

This long-term objective is broken down into stages:

1. Stage 1 –
    - Implement and validate a SpaceWire link between two nodes in simulation, representing the camera and the payload computer.
    - Understand how this SpaceWire core will sit in the PL, and how the PS will eventually control it.
2. Stage 2 –
    - Expose SpaceWire control/status via AXI-Lite registers, allowing ARM software to configure, monitor and send test packets.
3. Stage 3 –
    - Bring up this design on real ZSOM-F01 hardware, and test SpaceWire using LVDS physical drivers.
4. Stage 4 –
    - Define and implement higher-level protocols (commands, telemetry, data packets) that the AI module and Ku-band radio can use via this communication backbone.
5. Stage 5 –
    - Demonstrate a prototype integrated system where the payload computer coordinates camera data, AI processing and Ku-band preparation using the communication layer designed in this project.

Scope and Boundaries

The scope is intentionally focused on:

- Refining the integration problem and value proposition — why a unified communication backbone is needed and why it starts with SpaceWire.
- Studying the Zynq UltraScale+ MPSoC (ZSOM-F01) architecture: what the PS and PL can do, how AXI bridges them, and where the integration logic should live.
- Integrating the SpaceWire Light core into a Vivado project and understanding its interfaces.
- Building a dual-node SpaceWire simulation testbench that represents:
    - a camera node and a payload computer node exchanging realistic test messages.
- Designing a roadmap for how this SpaceWire-based backbone will later carry AI and Ku-band traffic as well.

Actual AI inference and Ku-band RF transmission are out of scope for this interim report, but they are explicitly included in the project roadmap and the design is structured to accommodate them in the next phases.

Novelty & Value Proposition

The novelty of this project lies in integrating three heterogeneous subsystems — camera, AI accelerator, and Ku-band transmitter through a unified, space-protocol–based communication architecture on a Zynq MPSoC. Unlike typical CubeSat payloads that use ad-hoc or subsystem-specific interfaces, this design introduces a reusable, modular communication backbone that can be adopted by later GALASSIA missions. Early simulation of SpaceWire links also provides an integration platform even before camera or radio hardware becomes available, reducing overall mission risk and shortening development cycles.

A2. Context of problem

Mission Setting

GALASSIA-5 is a small satellite mission that aims to push more intelligence on-orbit. Instead of downlinking raw data and doing everything on the ground, the mission intends to:

- Capture imagery from the camera payload,
- Process it using an on-board AI module, and
- Transmit only relevant results or compressed data through a Ku-band radio.

At the heart of this flow sits the ZSOM-F01 payload computer, built around a Zynq UltraScale+ MPSoC that combines:

- A Processing System (PS) — quad-core ARM Cortex-A53 and dual Cortex-R5, external DDR memory, general-purpose I/O, and OS support.
- A Programmable Logic (PL) — FPGA fabric with logic cells, DSP slices, internal RAM and high-speed I/O.

The payload computer is therefore the central hub:

- It receives data or commands from the camera,
- It forwards or pre-processes data for AI,
- It prepares telemetry and data packets for the Ku-band radio.

Integration Challenge

Integrating multiple heterogeneous subsystems into a single payload computer is not a trivial task. Each subsystem exposes different electrical interfaces, protocols, and timing constraints:

- Camera Payload — communicates purely through SpaceWire, a high-speed spacecraft data-link protocol with strict timing, flow-control, and packet formatting requirements.
- AI Module (e.g., Hailo) — typically interfaces via PCIe, Ethernet, or other high-bandwidth digital links, and expects structured input tensors or pre-processed image frames.
- Ku-Band Radio — often requires a framed, deterministic telemetry stream and may enforce its own packetisation rules for uplink/downlink.

Because these subsystems all "speak" differently, the payload computer cannot simply pass bytes around. Without a unifying communication strategy, the software in the Processing System (PS) becomes a patchwork of ad-hoc drivers, each behaving differently. This creates several problems:

- Testing becomes complicated — no single simulation or testbench can exercise all subsystems consistently.
- System reuse becomes difficult — future missions cannot adopt the same design without rewriting large software sections.
- Debugging becomes painful — failures may originate from the subsystem logic, the communication protocol, or the payload computer, with no clean isolation.

Thus, part of the project's novelty lies in establishing a common communication backbone and structured software layer that all subsystems can use. The goal is to make future additions (AI, Ku-Band, future sensors) plug-and-play rather than bespoke one-offs.

A3. Methodology/Concept Development

The methodology for this project is structured around the idea that the camera, AI module and Ku-band radio will eventually share a common communication backbone through the Zynq UltraScale+ payload computer. In the early stages, the focus is on the camera because it mandates the use of SpaceWire as its primary high-speed interface.

The design approach is therefore:

- First, implement and validate SpaceWire in the PL (as it is required for the camera anyway).
- Second, define a clean AXI-Lite control interface so that the PS can manage this link in software.
- Third, use the same architectural pattern and communication layer as the foundation for integrating AI and Ku-band subsystems in later stages.

Rationale for Choosing SpaceWire

SpaceWire offers:

- High throughput (tens to hundreds of Mbps).
- Full-duplex links (simultaneous send/receive).
- Packet-based communication with logical addressing.
- Built-in flow control using credit mechanisms.
- Robustness and widespread adoption in real satellite missions.

Given that GALASSIA-5 aspires to follow realistic space engineering practices, adopting SpaceWire is not just technically convenient; it is aligned with industry standards and improves the transferability of this work to future satellite projects.

Choice of IP Core

For simulation and academic work, I use the SpaceWire-Light open core because it is license-free, small in resource footprint, RTL-visible for debugging, and has an

AXI-style wrapper that eases PS–PL integration. This keeps the project flexible for future hardware substitutions.

Zynq UltraScale+ Architecture and PS–PL Integration Concept

Communication between PS and PL is done via AXI buses, primarily AXI-Lite for control and AXI-Full for data if needed. In this project: The SpaceWire core is instantiated in the PL and the PS interacts with it through memory-mapped AXI-Lite registers, enabling:

- Link configuration (e.g. enable, reset, speed divisor).
- Status monitoring (e.g. link running, error flags).
- Possibly, simple transmit/receive FIFOs.

. Refer to Appendix 3 for more details on Zynq.

Conceptual Design of the Communication Layer

The communication layer is conceptualised as three tiers:

1. Physical & Link Layer (PL / SpaceWire Core)
    - Encodes and decodes SpaceWire signals.
    - Manages low-level data/strobe lines and flow control.
    - Provides flags for link state (started, running, errors, etc.).
2. Register & Control Layer (AXI-Lite interface)
    - Exposes control registers to the PS.
    - Allows software to bring the link up/down, set parameters, and write payload data.
    - Provides read access to status and received data.
3. Application Layer (PS Software)
    - Implements subsystem-specific packet formats for camera, AI and Ku-band.
    - Defines commands (e.g. "start capture", "send frame X", "acknowledge frame").
    - Handles retries, logging, and integration with other software modules.

In this interim phase, the focus is mainly on Tier 1 (SpaceWire core) and Tier 2 (conceptual AXI-Lite design) with Tier 3 being planned for later implementation.

A4. Prototypes

Simulation Environment and SpaceWire Core

The project uses Vivado 2025.1 as the development environment. Refer to Appendix 1 for Design Steps.

Dual-Node SpaceWire Testbench

A key prototype is the dual-node testbench, which instantiates SpaceWire core A and core B:

- Node A simulates the camera payload side.
- Node B simulates the payload computer side.

The testbench includes:

- Clock generation at a defined system frequency (e.g. 100 MHz).
- Reset sequencing to bring the cores into a known state.
- Link startup control using autostart and link disable signals.
- A simulated "cable" using delayed connections between data and strobe lines:
    - A_do → B_di, B_do → A_di
    - A_so → B_si, B_so → A_si

Refer to Appendix 2 to view the code for controlling input signals in simulation

Test Messages and Scoreboard

To verify that the communication is behaving correctly, the testbench uses string-based test payloads:

- From camera (A) to payload computer (B):

CAMERA_FRAME (representing a frame identifier or small metadata packet).

- From payload computer (B) back to camera (A):

  PC_RETURNOK (representing an acknowledgement or completion code).

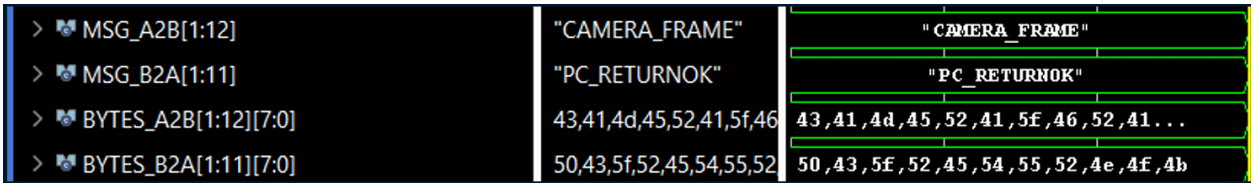

Figure 2. Output of Dual Core Communication using 2 Spacewire Cores

These strings are converted into bytes and transmitted over the SpaceWire link. On each side, a scoreboard process reconstructs the string from received bytes and compares it to the expected value. When the received payload matches the expected string, the testbench reports a PASS message; otherwise, it flags a failure.

A5. Testing

Purpose of Testing

The primary purpose of the current testing is:

- To confirm that SpaceWire cores can establish a stable link (A ↔ B) under simulation.
- To verify that bidirectional packets can be transmitted and reconstructed correctly.
- To gain confidence in the timing behavior (e.g. reset, autostart, txready, rxvalid).
- To understand the waveforms so they can later be compared to hardware measurements (e.g. logic analyzer, ILA).

Types of Tests Conducted

The tests so far are simulation-based in Vivado and focus on:

1. Link Initialization Test
    - Reset is asserted and de-asserted.
    - Autostart is enabled.
    - Link enable/disable is toggled to observe transition into a "running" state.

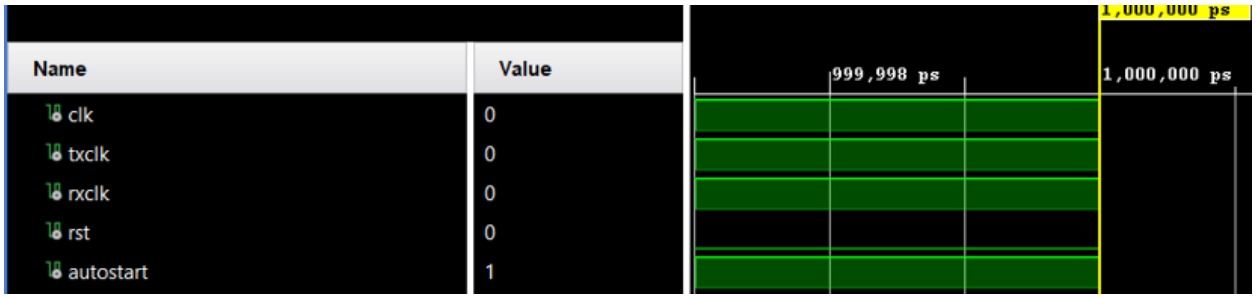

Figure 4. Simulation Output Example of Autostart being enabled

2. Unidirectional Transmission Test (A → B)
    - Node A sends CAMERA_FRAME with correct end-of-packet flag.
    - Node B receives the bytes and the scoreboard checks correctness.
3. Bidirectional Transmission Test (A ↔ B)
    - A and B both send their respective messages (CAMERA_FRAME, PC_RETURNOK).
    - Proper interleaving of traffic is observed in the waveforms.
    - Scoreboards on both sides verify that the complete messages are reconstructed.
4. Timing and Flow-Control Observations
    - Inspecting txrdy, txwrite, rxvalid, rxread signals.
    - Ensuring the testbench respects the protocol (e.g. only writing when ready).

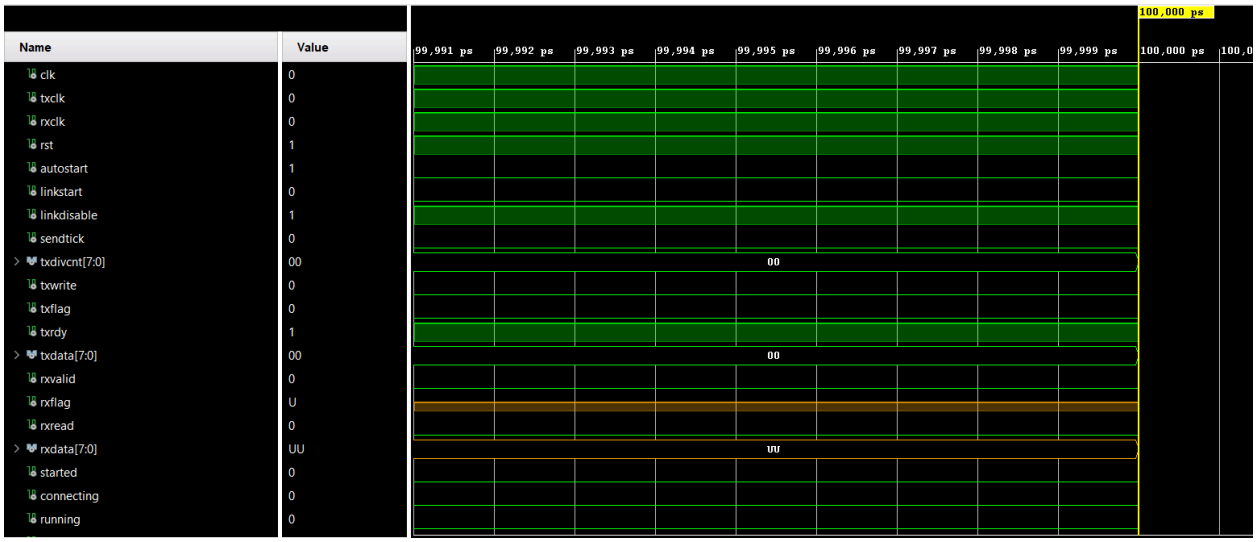

Figure 5. Sample output of the waveform generated from the input signals

Even when some aspects are still being debugged, these tests help clarify where timing or configuration issues occur and what is needed for stable operation.

Test Design and Method

The tests are designed to be:

- Deterministic – clocks, resets, and stimuli are scripted so waveforms are reproducible.
- Incremental – starting from link bring-up, then simple one-direction traffic, then full bidirectional flows.
- Instrumented – relevant signals are added to waveforms (running, txdata, rxdata, flags, clocks) for interpretation.

Figure 6. Sample output of decoder successfully receiving input bytes and assembling into a 4096 bit frame to extract 9 output characters from it

A6. Analysis

Interpretation of Simulation Behavior

From the simulation:

- The SpaceWire cores respond to reset and autostart signals as expected, transitioning through initialization phases.
- The testbench successfully injects payload data into the transmit interface (using txwrite, txdata, txflag).
- On the receive side, rxvalid and rxflag pulses indicate incoming data and end-of-packet boundaries.

These behaviours indicate that:

- The basic structure of the testbench and SpaceWire setup is correct.
- The communication pattern between A and B is visible and analysable.

Even at this stage, examining these waveforms gives insight into:

- Required timing margins between txwrite and txrdy.
- How long it takes for the link to move into a "running" state after reset.
- What the expected sequence of control signals should look like in hardware.

Why SpaceWire on Zynq Is Operationally Sound

From a system perspective, choosing SpaceWire on Zynq UltraScale+ is operationally sound because:

1. Protocol Suitability – SpaceWire is designed for exactly this kind of high-reliability, on-board satellite communication.
2. Architectural Fit – The PL can handle time-critical serial communication; the PS can manage higher-level control and integration logic.
3. Scalability – Once the camera link is proven, the same communication layer can be reused for AI and Ku-band subsystems.
4. Reusability – The SpaceWire implementation and testbenches can be reused or adapted for future satellite missions, not just GALASSIA-5.

A7. Fulfilment of Deliverables

What Is Proven at Interim

At this interim point, the project has:

- Refined the problem statement – clearly identifying the lack of a unified communication integration layer as a risk for GALASSIA-5.
- Justified the use of SpaceWire – conceptually and technically, compared to other protocols.
- Studied the Zynq UltraScale+ architecture – especially PS vs PL roles and how AXI-Lite will be used.
- Created a SpaceWire-based simulation environment – including a dual-node testbench for camera ↔ payload computer communication.

- Defined a clear roadmap – for extending from simulation to hardware and eventually to AI and Ku-band integration.

A8. Awareness of Shortcomings

Current Limitations

The project is currently limited by:

1. Simulation-Only Validation
    - No hardware SpaceWire link has been tested yet.
    - Electrical aspects (signal integrity, LVDS behavior) are not yet validated.
2. Camera-Only Focus
    - AI and Ku-band subsystems are still at conceptual planning level.
    - No end-to-end multi-subsystem test has been carried out yet.

Mitigation Strategies

To mitigate integration risks, the project follows an incremental validation strategy: simulation → PS–PL integration → hardware loopback → subsystem-level testing. If SpaceWire-Light shows timing issues on hardware, the design remains compatible with replacing it with a vendor-provided, timing-verified core without redesigning higher-level software.

A9. Project Plan (Next Stages)

Stage Roadmap and Milestones

Stage 1 (Completed)

- Problem definition and value proposition.
- Choice of SpaceWire and justification.
- Study of Zynq UltraScale+ architecture.

- Initial SpaceWire Light integration in simulation.
- Dual-node testbench and prototype message exchanges.

Stage 2 – PS–PL Integration via AXI-Lite (Completed)

- Design AXI-Lite register map for SpaceWire control and status.
- Implement register interface in PL, map to SpaceWire core signals.
- Develop simple PS-side software (bare-metal or minimal OS) to:
    - Bring link up and down.
    - Trigger test transmissions.
    - Monitor status flags.

Stage 3 – Hardware Bring-Up on ZSOM-F01

- Configure clocking and I/O constraints for the Zynq UltraScale+ on the ZSOM-F01.
- Synthesize and implement the design with SpaceWire core and AXI-Lite.
- Perform loopback tests using LVDS lines, initially on a single board.
- Capture hardware signals using in-chip logic analyzers or external measurement tools.

Stage 4 – Subsystem Protocol Design (AI & Ku-band)

- Define packet types for AI (e.g. "send tile", "return inference result").
- Define packet types for Ku-band (e.g. telemetry frames, compressed data packets).
- Implement encoding/decoding logic in PS software.
- Simulate and then test basic AI and Ku-band message flows over the same SpaceWire link.

Stage 5 – Final Validation and Documentation

- Measure throughput and latency under realistic traffic patterns.
- Demonstrate end-to-end test where camera-like data is forwarded through SpaceWire to the payload computer and then prepared for AI or radio.

- Finalise documentation: architecture diagrams, test descriptions, results, and reflection

Interim Conclusion

The interim phase demonstrates that a SpaceWire-based communication backbone on the Zynq UltraScale+ platform is feasible and technically sound. The completed simulation environment establishes a foundation for reliable subsystem communication, and the architectural plan provides a clear path toward PS–PL integration and hardware implementation. As the project moves into Semester 2, focus will shift toward register-level control, hardware bring-up, and extending the communication layer to the AI and Ku-band subsystems. The results so far indicate strong potential to meet all final deliverables.

Appendix:

Appendix 1

Creating a Vivado project representing the SpaceWire simulation environment.

- Importing the SpaceWire Light open-source VHDL files into the project.
- Setting up design sources for the SpaceWire core and simulation sources for testbenches.
- Ensuring that all VHDL files are compiled with correct standards (e.g. VHDL-2008) and libraries.

Appendix 2

```
-- clocks & control
signal clk, rxclk, txclk : std_logic := '0';
signal rst               : std_logic := '1';
signal autostart         : std_logic := '0';
signal linkstart         : std_logic := '0';
signal linkdisable       : std_logic := '1';
signal senddata          : std_logic := '0';
signal sendtick          : std_logic := '0';
```

Figure 3. Sample Code Used to control the clocks signals in the Simulation

Appendix 3

The Zynq UltraScale+ MPSoC divides the chip into:

PS (Processing System)

- Quad ARM Cortex-A53 for general computation.
- External DDR memory interface.
- Peripherals such as UART, USB, Ethernet, etc.

PL (Programmable Logic)

- FPGA fabric where the SpaceWire Light core will live.
- High-speed I/O, including LVDS transceivers, ideal for physical SpaceWire signals..

-

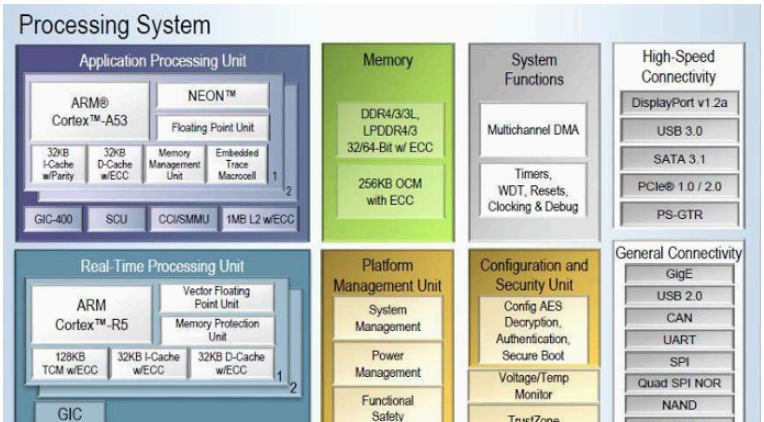

Figure 1. Overall Block Diagram of the Zynq ZU3EG