**Zynq Networking Dataplane**

Revision

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 1.0 | 01/23/2026 | Jesse Oinonen | Created the document |
| 1.1 | 02/08/2026 | Jesse Oinonen | PL architecture |

Table of contents

Contents

[2 Introduction 3](#_Toc221458742)

[3 Architecture 4](#_Toc221458743)

[3.1 Programmable Logic (PL) 5](#_Toc221458744)

[3.1.1 axi\_lite\_slave.sv 5](#_Toc221458745)

[3.1.2 axi\_addr\_decode.sv 5](#_Toc221458746)

[3.1.3 csr.sv 6](#_Toc221458747)

[3.1.4 axi\_rx.sv 6](#_Toc221458748)

[3.1.5 eth\_parser.sv 6](#_Toc221458749)

[3.1.6 ipv4\_parser.sv 6](#_Toc221458750)

[3.1.7 udp\_tcp\_parser.sv 7](#_Toc221458751)

[3.1.8 flow\_key\_gen.sv 7](#_Toc221458752)

[3.1.9 flow\_table.sv 7](#_Toc221458753)

[3.1.10 action\_stage.sv 7](#_Toc221458754)

[3.1.11 axi\_tx.sv 8](#_Toc221458755)

# Introduction

This project implements a Linux-controlled, FPGA-accelerated networking dataplane on the Zynq-7000 platform. The system combines an ARM-based 650MHz dual-core Cortex-A9 Processing System (PS) running embedded Linux with a high-performance Programmable Logic (PL) datapath optimized for deterministic throughput and low latency.

The dataplane is built around an AXI-based architecture, utilizing AXI-Stream for high-speed data movement and AXI4-Lite for control and configuration. A zero-copy DMA mechanism enables efficient data transfer between software and hardware without unnecessary memory copies. In addition, the design includes reusable observability logic for measuring throughput, latency, and error statistics at runtime.

The project is intended as a reusable hardware/software co-design reference, demonstrating best practices in FPGA datapath design, Linux kernel integration, and AXI-based system architecture.

# Architecture

The architecture of this project is divided into processing unit (PS) and programmable logic (PL).

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, diagrammi

Tekoälyllä luotu sisältö voi olla virheellistä.

*Figure 1. Block diagram.*

## Programmable Logic (PL)

The programmable logic (PL) of this project is Arty Z7 project’s XC7Z010-1CLG400C FPGA which contains 28k logic cells, 80 DSP slices and 2.1 Mbits of block RAM.

### axi\_lite\_slave.sv

The axi\_lite\_slave module provides the control-plane interface between the Processing System (PS) and the Programmable Logic (PL). AXI4-Lite is selected as the communication bus due to its simplicity and suitability for low-bandwidth control register and table access.

The module implements an AXI4-Lite slave interface and handles read and write transactions issued by the PS. Write transactions are forwarded to the internal PL address decoding logic, while read transactions are supported only for readable control and status registers.

The AXI4-Lite interface operates with a fixed 32-bit data bus width. All transactions are aligned to the data bus width and follow the AXI4-Lite protocol timing and handshaking requirements.

### axi\_addr\_decode.sv

The axi\_addr\_decode module decodes write transactions received from the axi\_lite\_slave module and routes them to the appropriate PL submodules. Address decoding is required only for write operations, as only the control and status registers are readable, while the flow table and action stage tables are write-only structures.

The upper address bits (waddr[31:30]) determine the destination module for each write transaction. Based on the decoded address, the write enable, address, and data signals are forwarded to the selected module.

This centralized address decoding approach simplifies the AXI4-Lite interface logic and allows each destination module to remain decoupled from AXI-specific signaling.

The addresses are decoded according to Table 1.

|  |  |
| --- | --- |
| **waddr[31:30]** | **Module** |
| 2’b00 | csr.sv |
| 2’b10 | flow\_table.sv |
| 2’b01 | action\_stage.sv |

*Table 1. Write address decoding.*

### csr.sv

The csr module implements the control and status registers of the dataplane. These registers are accessible by the PS via AXI4-Lite and are used to configure dataplane behavior, enable or disable functionality, and monitor runtime status information.

The CSR block is the only PL module that supports AXI read operations. All registers are memory-mapped and use a fixed 32-bit data width. The module exposes configuration outputs to the dataplane pipeline and receives status inputs from various PL modules.

### axi\_rx.sv

The axi\_rx module receives incoming network traffic from the Ethernet MAC over an AXI4-Stream interface. The received data corresponds to Ethernet Layer-2 frames at the MAC client interface level, including the destination MAC address, source MAC address, EtherType field, and payload.

Physical layer fields such as the preamble, start-of-frame delimiter (SFD), and frame check sequence (FCS) are terminated at the PHY/MAC layer and are not visible to the PL.

The axi\_rx module forwards the AXI4-Stream data and associated sideband signals to the parsing stage of the dataplane pipeline without modifying the packet contents.

### eth\_parser.sv

The eth\_parser module parses Ethernet Layer-2 headers from the incoming AXI4-Stream packet data. It extracts relevant header fields such as source MAC address, destination MAC address, and EtherType.

Based on the EtherType field, the module determines whether the packet should be forwarded to higher-layer parsers or dropped. Unsupported or non-IPv4 packets may be flagged for dropping while still maintaining proper AXI4-Stream frame termination.

The extracted header fields are forwarded as metadata to subsequent pipeline stages.

### ipv4\_parser.sv

The ipv4\_parser module processes IPv4 headers following successful Ethernet parsing. It extracts IPv4-specific fields such as source IP address, destination IP address, protocol field, and header length.

The module validates basic IPv4 header properties and determines whether the packet payload should be forwarded to transport-layer parsing. Packets with unsupported protocols or invalid headers may be marked for dropping.

Parsed IPv4 metadata is forwarded alongside the packet stream to the next pipeline stage.

### udp\_tcp\_parser.sv

The udp\_tcp\_parser module parses transport-layer headers for UDP and TCP packets. Based on the IPv4 protocol field, it extracts source and destination port numbers and protocol-specific information required for flow identification.

Packets using unsupported transport protocols bypass this stage or are marked for dropping depending on the system configuration. The extracted transport-layer metadata is forwarded to the flow key generation stage.

### flow\_key\_gen.sv

The flow\_key\_gen module generates a fixed-width flow key based on parsed packet metadata. The flow key is constructed from selected Layer-2, Layer-3, and Layer-4 header fields, such as IP addresses, transport ports, and protocol identifiers.

The generated flow key uniquely identifies a traffic flow and is used as the lookup key for the flow table. The module outputs the flow key together with packet metadata to the match-action pipeline.

### flow\_table.sv

The flow\_table module implements a programmable lookup table that maps flow keys to flow identifiers or action descriptors. The table is populated and updated by the PS via AXI4-Lite write transactions.

During packet processing, the flow table performs a lookup using the generated flow key and outputs the corresponding match result. The table is write-only from the PS perspective and does not support readback.

### action\_stage.sv

The action\_stage module applies actions to packets based on the match result received from the flow table. Actions may include packet forwarding, modification, dropping, or metadata updates.

Action entries are configured by the PS via AXI4-Lite writes. The module operates entirely in the dataplane and does not expose readable state to the PS.

### axi\_tx.sv

The axi\_tx module transmits processed packets from the dataplane to the Ethernet MAC using an AXI4-Stream interface. It receives packet data and metadata from the action stage and forwards them while preserving frame boundaries and AXI4-Stream protocol signaling.

The module serves as the final stage of the dataplane pipeline before packets exit the PL toward the network interface.