Performance Optimization of a NIC for SoC Prototyping

M.Tech Thesis Presentation (EE-798)

Bhupendra Sahu (22M1170)

Under the guidance of

Prof. Madhav P. Desai

Department of Electrical Engineering
Indian Institute of Technology Bombay

Background and Objectives

Background

- A custom NIC was initially developed by Harshad Ugale.
- Siddhant Singh Tomar improved memory architecture with fast local memory and L2 cache.
- Despite these efforts, throughput remained below 10% of link speed.

Objectives

- Achieve at least 60% of link speed through NIC performance optimization.
- Develop a reliable scheme for packet transmission and reception.
- Build a scalable NIC platform for future high-throughput applications.

Introduction

- A Network-Centric SoC (N-SoC) is designed using a 32-bit AJIT processor, custom NIC, and DRAM.
- Implemented on Xilinx KC705 FPGA with a target NIC throughput of 600 Mbps.
- Key enhancements: fast memory access, larger FIFOs, and hardware-managed queues.

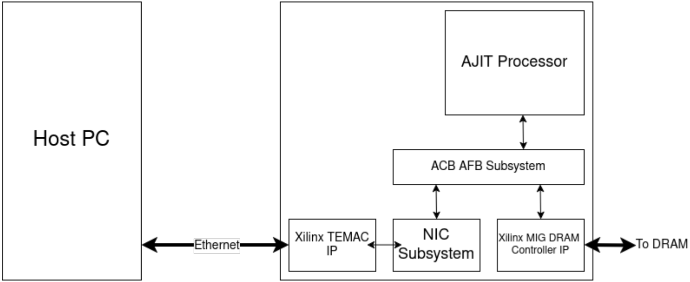

Figure: N-SoC with AJIT processor and custom NIC

Packet Buffer Management

- Buffer pointers circulate through hardware queues for efficient packet flow:
  1. **Free Queue:** All pointers start here.
  2. **NIC Receive:** Pops a pointer, writes packet to memory, pushes to Receive Queue.
  3. **Processor:** Pops from Receive Queue, processes data, pushes to Transmit Queue.
  4. **NIC Transmit:** Pops from Transmit Queue, sends packet, returns pointer to Free Queue.
- This cycle ensures efficient reuse of memory and smooth data transfer between NIC and processor.

SBC Architecture Diagram

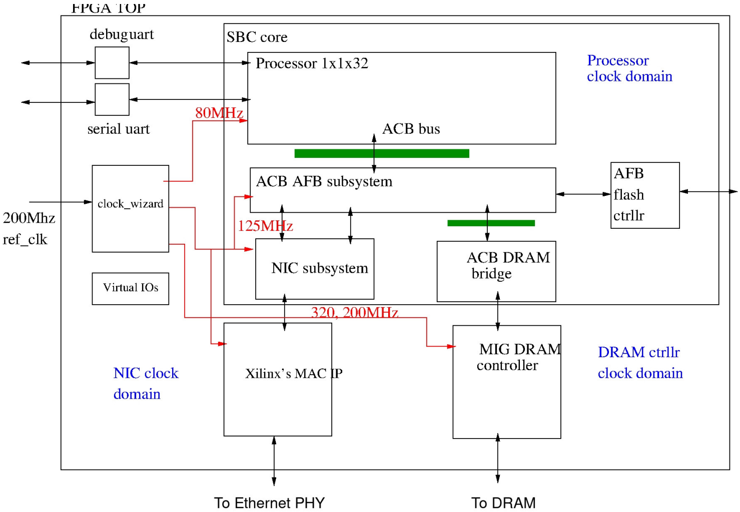

Figure: Architecture of single board computer with AJIT 1x1x32

SBC Architecture Overview

Core Components:

- AJIT Processor (1x1x32)
- Custom NIC Subsystem
- AFB & ACB Bus Complex and DRAM Bridge

Peripherals:

- Xilinx Tri-mode Ethernet MAC IP
- Xilinx MIG DRAM Controller IP

Measurement Setup:

- Processor Clock: 80 MHz
- NIC Clock: 125 MHz
- Ethernet Link Speed: 1 Gbps

NIC Subsystem Architecture

- Slave Register and Master Memory Interfaces
- Direct physical memory access for packet RX/TX
- Integrated with Xilinx Tri-mode Ethernet MAC IP for frame handling

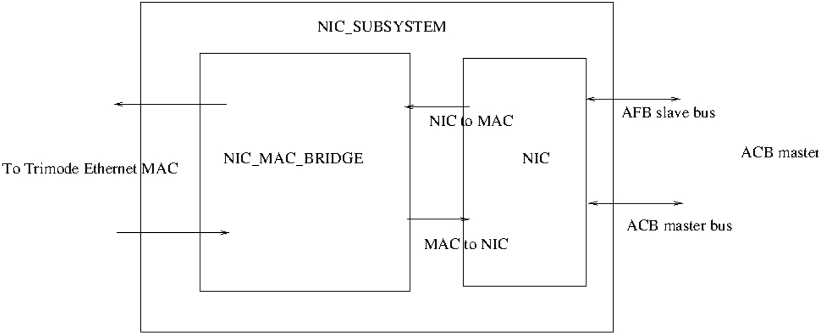

Figure: NIC Subsystem

Optimization A: Shared Fast Local Memory

- 256KB shared SRAM between NIC and processor
- Reduced memory access latency
- Marked non-cacheable to maintain consistency

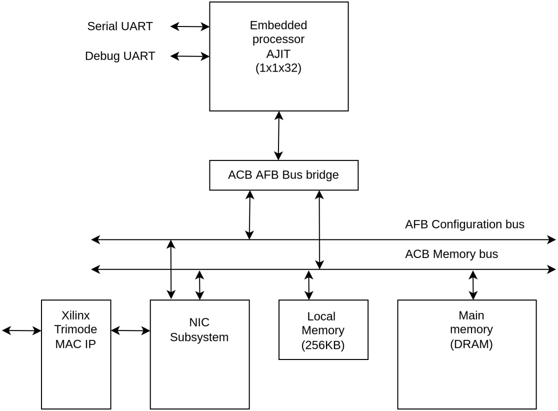

Figure: NIC and processor sharing a fast local packet memory of 256KB [1]

Optimization B: L2 Cache

- Inclusive write-back L2 cache
- Shared access for processor and NIC
- Reduces DRAM access latency

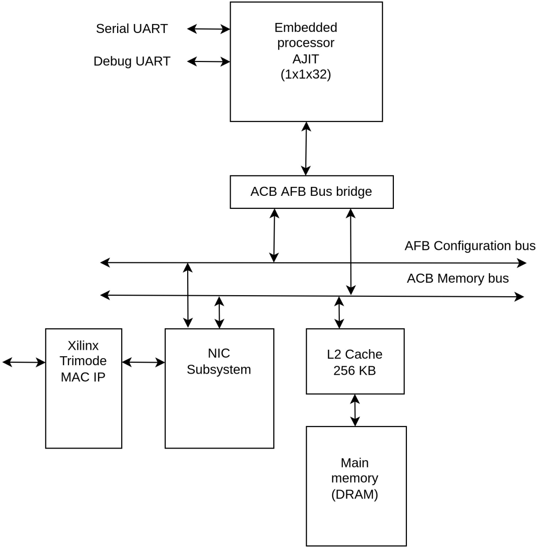

Figure: NIC and processor access main memory via L2 cache [1]

RTT and Throughput Comparison Summary

Payload (B)	Baseline (μs)	Fast Mem (μs)	L2 Cache (μs)
64	230	192	202
128	244	201	213
256	270	220	234
512	323	261	280
1024	409	333	354

Table: Average RTT (μs) for Various Payload Sizes

Payload (B)	Baseline (Mbps)	Fast Mem (Mbps)	L2 Cache (Mbps)
64	9.28	9.46	9.41
128	15.84	17.37	16.95
256	18.56	32.42	28.64
512	27.92	49.97	43.96
1024	36.24	57.33	51.58
1486	40.72	62.37	56.46

Table: Raw Ethernet Throughput (Mbps) for Various Payload Sizes

Tri-mode Ethernet MAC on KC705

Processor-free Ethernet evaluation:

- Uses Xilinx TEMAC IP core on KC705.
- MAC runs in programmable logic, no CPU.
- Built-in packet generator/checker.
- AXI-lite control state machine initializes MAC & PHY.
- Loopback via MAC address swap enables RTT testing.

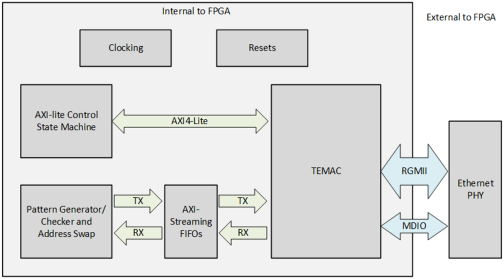

Figure: Tri-mode MAC IP Example Design [4]

Ideal for raw link performance benchmarking.

Performance Results

Average RTT (μs)

Payload (bytes)	RTT (μs)
64	99
128	109
256	120
512	137
1024	156

Throughput (Mbps)

Payload (bytes)	Throughput (Send/Receive)
64	144.09
128	170.36
256	487.85
512	867.51
1024	852.00
1486	888.19

Near 1 Gbps max throughput with very low jitter — efficient hardware-only MAC.

Optimization 1: Queue Relocation in NIC_1.2

- NIC_1.1 stored Rx/Tx/Free queues in DRAM, causing:
  - High latency from ACB bus memory access
  - Contention between processor and NIC for DRAM bandwidth

- NIC_1.2 shifts queues to on-chip BRAM FIFOs, enhancing:
  - Latency via fast memory-mapped FIFOs
  - DRAM bandwidth reserved solely for packet buffers
  - Efficient polling using NIC status registers

- **Queue Architecture:**
  - **Free Queue:** Single FIFO (depth 64), lock-protected
  - **Rx/Tx Queues:** Four FIFOs each per server (depth 64), lock-free, round-robin scheduled

NIC Subsystem Architecture

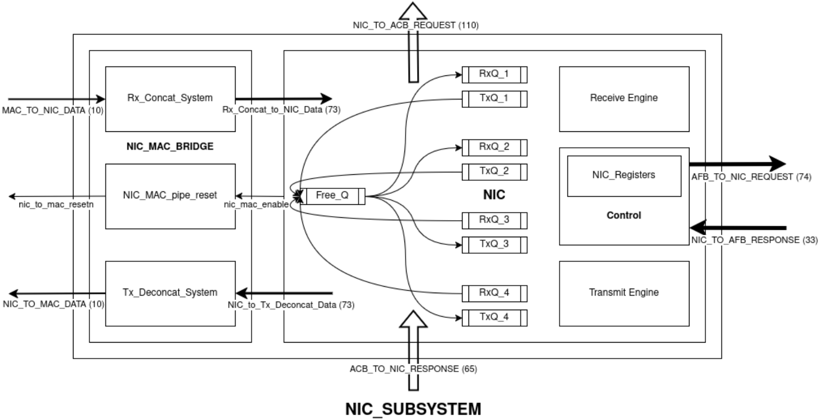

Figure: NIC subsystem architecture highlighting queue relocation

Simulation Timing Summary

- **Register Access:** 15 cycles (150 ns)
- **Queue Operations (push/pop):**
  - NIC_1.1: 286 cycles (2860 ns)
  - NIC_1.2: 20 cycles (200 ns)
- **Full Cycle Sequence (NIC_1.2):** 218 cycles (2180 ns) for:

$$
\begin{tikzcd}
\texttt{pop (free)} \arrow[r] & \texttt{push (Rx)} \arrow[r] & \texttt{pop (Rx)} \arrow[d] \\
\texttt{push (free)} & \texttt{pop (Tx)} \arrow[l] & \texttt{push (Tx)} \arrow[l]
\end{tikzcd}
$$

Optimization 2: MAC FIFO Upgrade in NIC_1.2

Problem in NIC_1.1:

- MAC RX and TX FIFOs were only 4kB
- Could not handle bursty or high-speed traffic
- Caused packet drops and reduced reliability

Fix in NIC_1.2:

- FIFO size increased to 16kB for both RX and TX
- Allows more packets to be buffered safely
- Change made by adjusting BRAM and address width

FIFO Upgrade in MAC

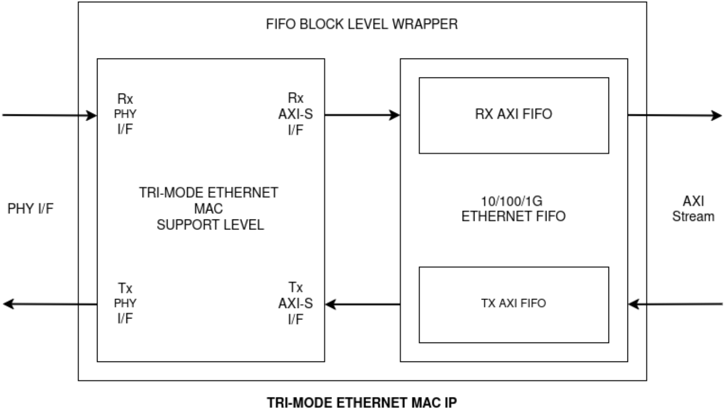

Figure: MAC in NIC_1.2 with 16kB RX/TX FIFOs

Optimization 3: Packet Scheduling in NIC_1.2

- Traditional FIFO transmission leads to head-of-line blocking and poor bandwidth utilization.
- High-throughput applications (e.g., image streaming) need both speed and reliable delivery.
- NIC_1.2 introduces two improved scheduling strategies:
  - **Continuous Fill Strategy** – maximizes throughput by keeping FIFOs full.
  - **ACK-Based Reliable Protocol** – ensures data integrity with selective retransmission.
- Both strategies were implemented and validated on the Xilinx KC705 FPGA.

Continuous Fill Strategy

Mechanism:

- Monitors packet dequeue events to maintain FIFO fullness.
- Continuously sends new packets without waiting, ensuring constant data flow.

Benefits:

- Maximizes throughput by maintaining continuous FIFO occupancy.
- Well-suited for large, steady data streams like images or video.

Limitation:

- Lacks error detection and recovery; any lost packets are not retransmitted.

ACK-Based Reliable Protocol

Mechanism:

- Sender transmits all packets with sequence numbers.
- Sends a summary packet with total count.
- Receiver responds with ACK: "OK" or "MI" (Missing Indexes).
- Sender retransmits only the missing packets.

Advantages:

- Ensures reliability without full TCP/IP stack.
- Maintains high throughput with minimal retransmissions.
- Graceful timeout and clean shutdown if ACKs are lost.

Scheduling Strategy Illustration

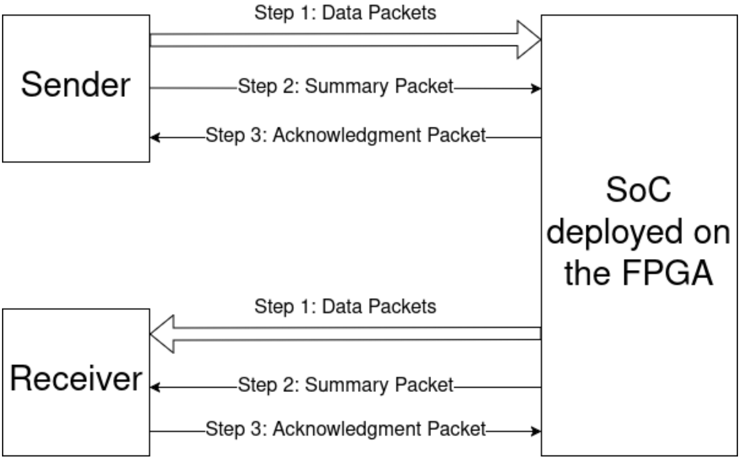

Figure: NIC_1.2 Implementation of Scheme 4 — Acknowledgment-Based Packet Scheduling

RTT after NIC Optimization

Payload Size (B)	Baseline (μs)	Fast Local Mem (μs)
64	143	140
128	146	145
256	151	149
512	165	156
1024	188	180

Table: RTT Comparison after NIC Optimization

Throughput after NIC Optimization (16kB MAC FIFOs)

Payload (B)	Baseline (Mbps)	Fast Local Mem (Mbps)
64	52.60	62.74
128	84.79	100.94
256	151.49	179.19
512	270.42	334.75
1024	477.09	602.45
1486	526.12	731.89

Table: Throughput Comparison after NIC Optimization (16kB MAC FIFOs)

Image Transfer Performance (NIC_1.2 with Fast Local Memory)

Metric	Send (Mbps)	Receive (Mbps)	Images/sec
Small Image (347.5 kB; 680 $\times$ 680) — Continuous Scheduling			
Avg Throughput	659.12	657.66	115.6
Max Throughput	791.18	781.49	–
Small Image (347.5 kB; 680 $\times$ 680) — ACK-Based Protocol			
Avg Throughput	647.79	647.31	113.8
Max Throughput	761.81	750.63	–
Large Image (15.9 MB; 4600 $\times$ 4600) — Continuous Scheduling			
Avg Throughput	653.03	652.56	2.45
Max Throughput	706.40	705.65	–
Large Image (15.9 MB; 4600 $\times$ 4600) — ACK-Based Protocol			
Avg Throughput	638.25	637.37	2.39
Max Throughput	702.61	701.89	–

Table: Image transfer results for NIC_1.2 with fast local memory (measured on FPGA). Images/sec calculated using total Tx + Rx time.

Average RTT (μs) Comparison: NIC_1.1 vs NIC_1.2

Payload (bytes)	NIC_1.1		NIC_1.2	
	Baseline	Fast Mem	Baseline	Fast Mem
64	230	192	143	140
128	244	201	146	145
256	270	220	151	149
512	323	261	165	156
1024	409	333	188	180

Table: Average RTT (μs) Comparison: NIC_1.1 vs NIC_1.2

Throughput (Mbps) Comparison: NIC_1.1 vs NIC_1.2

Payload (bytes)	NIC_1.1		NIC_1.2	
	Baseline	Fast Mem	Baseline	Fast Mem
64	9.28	9.46	52.60	62.74
128	15.84	17.37	84.79	100.94
256	18.56	32.42	151.49	179.19
512	27.92	49.97	270.42	334.75
1024	36.24	57.33	477.09	602.45
1486	40.72	62.37	526.12	731.89

Table: Throughput (Mbps) Comparison: NIC_1.1 vs NIC_1.2

RTT Comparison Plot

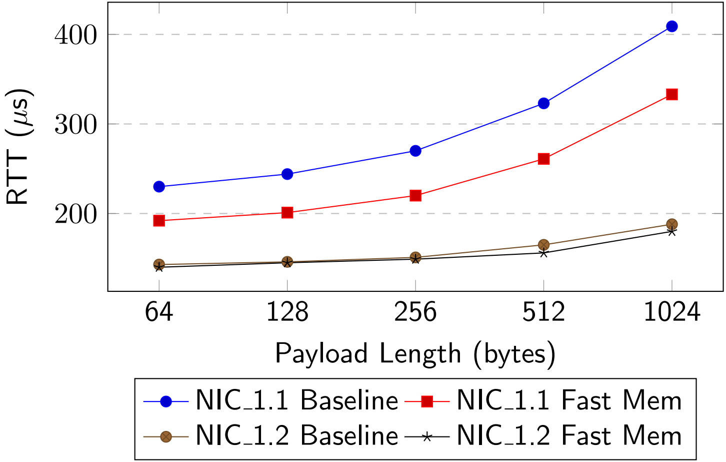

Figure: RTT Comparison between NIC 1.1 and NIC 1.2 (Baseline and Fast Local Mem)

Throughput Comparison Plot

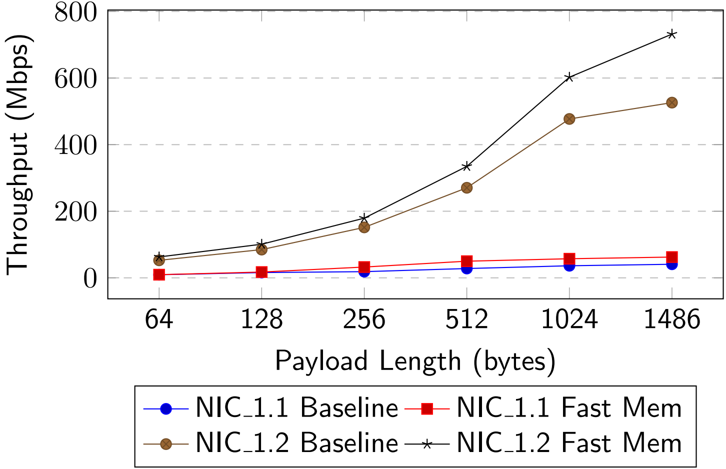

Figure: Throughput Comparison between NIC_1.1 and NIC_1.2 (Baseline and Fast Local Mem, with 16-kB MAC FIFO)

Performance Improvements Summary

- **RTT:** NIC_1.2 reduces latency by up to **2x** versus NIC_1.1.
- **Throughput:** NIC_1.2 achieves up to **12x** higher throughput for small payloads and **11x** for larger ones.
- Enlarged 16 KB MAC FIFOs support more efficient and smoother continuous scheduling.
- Offloading queue management to internal FIFOs reduces DRAM bandwidth usage and overhead.
- Image transfers approach near-saturation throughput on 1 Gbps Ethernet ($\sim$ 659 Mbps).

Future Work

1. **64-bit NIC:** Upgrade NIC_1.2 to 64-bit for improved bandwidth and DRAM efficiency.
2. **Custom Protocols:** Design application-specific network protocols for optimized communication.
3. **Multi-core SoC:** Extend to a multi-core architecture with separate control and data cores.
4. **Beyond 1 Gbps:** Achieve throughput over 1 Gbps through further architectural enhancements.

Toward a scalable, high-performance platform for specialized networking applications.

References

[1] S. S. Tomar, "Towards an soc architecture for software-defined networking (sdn)," m.tech thesis, IIT Bombay, 2024.

[2] M. P. Desai, *The AJIT Processor, IIT Bombay*.

[3] *The SPARC Architecture Manual, Version 8*.

[4] J. Johnson, "Driving ethernet ports without a processor." `https://www.fpgadeveloper.com/driving-ethernet-ports-without-a-processor/`, 2016.

Thank You!

KC705 FPGA Utilization Report

Resource	Available	Baseline	Fastmem	Baseline_4x	Fastmem_4x
LUTs	203800	52.60%	53.01%	52.61%	53.02%
Registers	407600	21.47%	21.84%	21.48%	21.84%
RAM	445	14.04%	29.66%	15.62%	31.24%
DSP	840	1.55%	1.55%	1.55%	1.55%

Table: Resource Utilization of KC705 FPGA