Design and Implementation of a Software Interface for an AI Modem

Yunjoo Kim, Jungbo Son, Yuro Lee, JungSook Bae
Terrestrial & Non-Terrestrial Integrated Telecommunications Research Laboratory
ETRI (Electronics and Telecommunications Research Institute)
Daejeon, Korea
{yunjoo, jbson, yurolee, jsbae}@etri.re.kr

***Abstract*—This study focuses on the design and implementation of two system software interfaces. These interfaces reduce data processing latency between a field programmable gate array (FPGA) and a graphics processing unit (GPU) in an artificial intelligence (AI)-based wireless modem system. Experimental results indicate that the shared memory-based interface achieved lower and more stable latency than the transmission control protocol/internet protocol (TCP/IP)-based interface and is well suited for real-time systems.**

***Index Terms*—AI modem, system software interface, intelligent wireless access**

I. Introduction

Modem architectures have evolved into artificial intelligence (AI) modems that include graphics processing units (GPUs) and neural processing units (NPUs) [1] [2]. In these environments, it is important to reduce latency and synchronization issues in data flows between computing resources and network interface cards (NICs). To ensure real-time operation, these systems require low-latency software interfaces [3] [4].

This paper designs software interfaces for real-time operations and evaluates them in a practical system. Their structures and performance are described in Sections II through V.

II. AI Modem System Overview

A. System Architecture and Parameters

The proposed AI modem system consists of a GPU, a central processing unit (CPU), and multiple field programmable gate arrays (FPGAs), as shown in Fig. 1. The GPU, as a computing resource, executes the machine learning-based software modem and performs demodulation and channel estimation. Three FPGAs mounted on two boards handle baseband signal processing and transmission buffering.

TABLE I: System Parameters

Variable	Value
System bandwidth (MHz)	400
Subcarrier spacing (kHz)	120
FFT size	4096
Number of subcarriers	3072
Occupied bandwidth (MHz)	368.64
OFDM symbol duration (μs)	8.33
CP length	288
System clock (MHz)	122.88
DAC/ADC sampling frequency (MHz)	3932.16

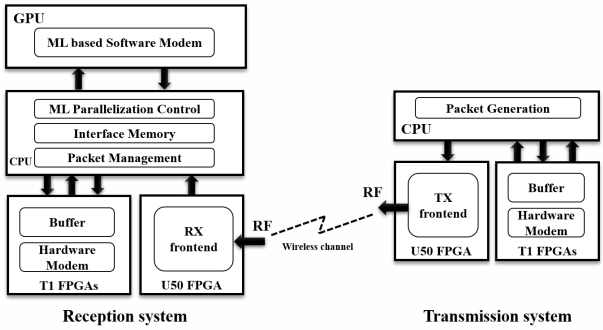


Fig. 1: AI Modem transceiver platform architecture

This system operates in the 28 GHz millimeter-wave band and supports up to 400 MHz bandwidth and a downlink transmission rate of 1 Gbps. The main system parameters are summarized in Table I.

B. Slot-based Reception Flow

The reception system processes data in slot units. As shown in Fig. 2, its overall processing flow consists of a machine learning module implemented with TensorRT and compute unified device architecture (CUDA), referred to as the ML+CUDA module, and a hardware modem. In this process, the software interface exchanges slot data packets between the ML+CUDA module and the hardware modem through interface memory. Multiple instances of the ML+CUDA module operate in parallel.

To support slot-based data transfer between an FPGA and a GPU, the proposed system uses peripheral component interconnect express (PCIe) and data plane development kit (DPDK).

III. Software Interface Design

This chapter describes the software interfaces that are accessible from both Python and C environments.

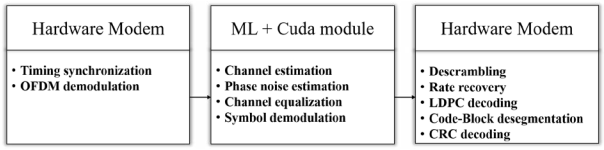


Fig. 2: ML-based reception modem block diagram

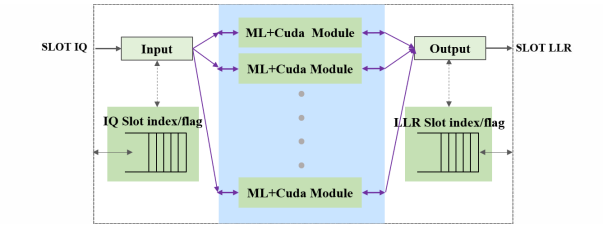


Fig. 3: Shared memory-based software interface

A. TCP/IP based Interface Design

An interface was based on transmission control protocol/internet protocol (TCP/IP) sockets with the loopback function, which is commonly supported in both Python and C environments. This design enabled data transfer between the GPU and the CPU/FPGA components. In the proposed structure, each ML+CUDA module used an independent TCP/IP socket and each instance was identified by a unique port.

B. Shared-Memory based Interface Design

A shared memory-based interface in Fig. 3 was designed to reduce the latency introduced by the previously applied TCP/IP socket-based interface. Since the shared memory can be accessed from both Python and C environments, the number of memory copy operations is reduced. This reduction leads to lower data transfer latency between the GPU and the FPGA.

IV. EVALUATION AND DISCUSSION

A. Experimental Setup

The experiment used a reception system with an AMD Threadripper PRO 7965WX CPU, an NVIDIA RTX 6000 Ada GPU, and Xilinx U50 and T1 FPGAs (Fig. 4). The software ran on Ubuntu 22.04 LTS with CUDA 12.1. ML-based demodulation was executed on the GPU. Two interfaces—TCP/IP sockets and shared memory—were applied for data transfer between the FPGA and the ML+CUDA module. The evaluation measured average slot latency as the number of module instances increased from 4 to 8.

B. Experimental Results and Analysis

Fig. 5 shows the difference in slot processing latency between the two interface types. In Fig. 5 (a), the processing latency significantly increased as the number of instances increased. In contrast, in Fig. 5 (b), the processing latency remained nearly constant despite the increased number of instances. This result shows that the simple interface structure reduces the overhead related to multiple instances.

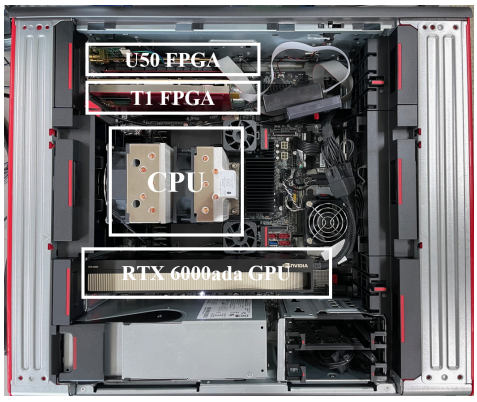
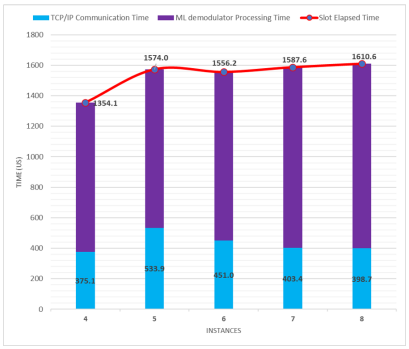
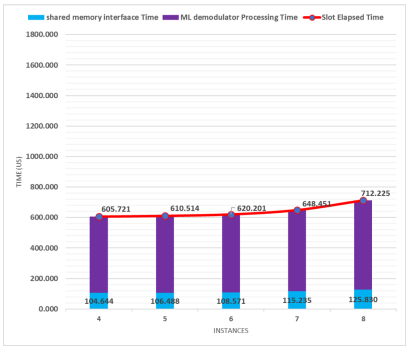


Fig. 4: AI-modem reception server



(a) TCP/IP-based interface



(b) Shared memory-based interface

Fig. 5: Slot elapsed latency comparison

V. CONCLUSIONS

This paper presented the design of two software interfaces for real-time communication with an AI modem. One was a TCP/IP-based design and the other was a shared memory-based design. Experimental results showed that the shared memory-based method was more suitable for real-time processing because of its lower and more stable latency, and the proposed interface can be extended to complex operational environments such as large-scale 5G/6G base stations.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00972, Development of Intelligent Wireless Access Technologies).

REFERENCES

[1] E. Jeong, *et al.*, "Deep Learning Inference Parallelization on Heterogeneous Processors With TensorRT," *IEEE Embedded Systems Letters*, vol. 14, no. 1, pp. 15–18, Mar. 2022.
[2] Y. Zhou and K. Yang, "Exploring TensorRT for Real-Time Deep Learning," in Proc. IEEE HPCC, 2022, pp. 2011–2018.
[3] Y. Xiang and H. Kim, "Pipelined CPU/GPU Scheduling for Multi-DNN Inference," in Proc. IEEE RTSS, 2019, pp. 392–405.
[4] E. F. Kfoury, *et al.*, "A Comprehensive Survey on SmartNICs," IEEE Access, vol. 12, pp. 107297–107336, 2024.