STEPHEN A. EDWARDS
CSEE 4840
Embedded System Design
Spring 2014

FPGA UDP Memcached Server Deign

Design Document

March 27, 2012

Chia-Kai Chou

Meng-Yi Hsu

Chen Wen

Table of Contents

Introduction

Memcached is a general-purpose distributed memory caching system It is used to speed up any data writing/reading by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read. Memcached runs on Unix, Linux, Windows and Mac OS X and is distributed under a permissive free software license. However, we believe that A FPGA Memcached server will best exploit the potential of Memcached than X86 Servers. FPGA has a high flexibility in memory allocation and parallel computation. To exemplify our idea, we decide to user SoCKit FPGA to build a UDP packet Memcached server.

User Datagram Protocol (UDP) is one of the most popular members of Internet protocol suite (IPS). Ideally, the natural parallelism ability of FPGA suits the need to handle decoding UDP packets. Since System Verilog programming is a logic-level programming, there will not be any clock cycle wasted on waiting for ALU, which is unavoidable execution time consumption in X86 Servers.

Problem Statement

FPGA Memcached Server needs to be able to decoder from and encode to any standard UDP packets from the Internet. Memcached Server stores the decoded UDP as hash table with a proper selection of hash function. The hash function will also be available to use as an API. A test bench C application is needed for demonstrate that the client computer could achieve the proper hash value from the hash key, and change the hash value and store in back to Memcached Server. The test bench will feature the function of monitoring the time consumption of Memcached Server’s decoding/encoding.

Design Concept

The entire design is divided into the following segments,

1. UDP Encoder in Verilog.
2. UDP Decoder in Verilog.
3. Hash function in Verilog.
4. Hash function (API) in C language.
5. Test bench in C language.

UDP Packet Decoder

The UDP packet decoder is the first module in our design. From the Ethernet port of the FPGA board, Ethernet packets are imported. These Ethernet packets consist of Ethernet Header, IP Header, UDP Header, Data and Ethernet Trailer (Fig. 1). The decoder module is designed to parse the packet, extract the user data and output the data payload and the length of the Data. Since the speed limit of Ethernet is about 500 bytes per time, the packet is sent separately. In concert with the fragmental Ethernet packet, the decoder module is designed to detect different segments and combine data segments of Ethernet payload into completed UDP packet before decoder it.

We decide to use on-chip cache to perform as the memory in our Memcached Server System. We made the calculation that for on-chip cache is enough for roughly ten UDP packets. This strategy provides the possible highest speed in memory writing/reading.

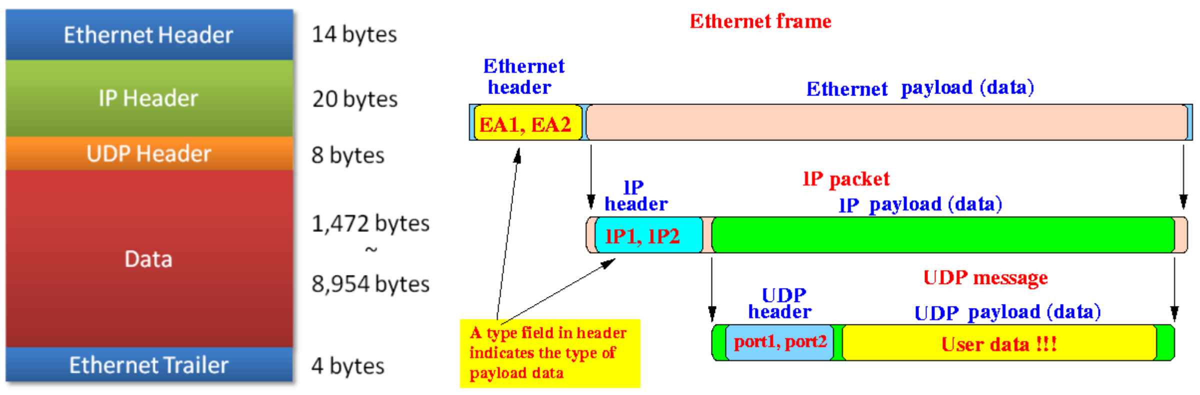

Figure 1, Ethernet Packet and Structure

UDP Packet Encoder

The client computers connect to the memcached system via router in a LAN. In most cases, the client computers only concern to ask for data (hash value) in the hash table and possibly update it with new value. The encoder module encapsulates the data in order to send an Ethernet frame back to the client computer. The encoding process would be the opposite of the Decode module. UDP Header, IP Header, Mac Header and checksum would be sequentially add to the packet and eventually send back to the Ethernet. IEEE 802.3 CSMA/CD (Carrier Sense Multiple Access with Collision Detection) would be followed.

Hash Function in Verilog

In Memcached Server, the hash table is used to save and stream data. The hash table implements an associative array, a structure that can table keys to values. In our design, the decoded UDP packet data is the value we want to store. Like the general hash table, a key (or keys) is assigned to the input data as their identification by a hash function. This hash function will present in Verilog file so the decoder/encoder could use it.

Ideally, the hash function will assign a unique key to each data value, but this situation is rarely achievable. Since Memcached Systems are usually used for speeding up because they help to save certain or target information on the internet, we hope to build a hash function especially for our target UDP packets. By focusing on target information in the input data, we hope to build a perfect hash function, which has no collisions. That is to say, we plan to analyst our input data first and then design how to store the data in the hash table.

Hash Function (API) in C language

We try to implement all the function by hardware but this time we need to communicate with users. The hash function will be presented in a C language file as API. Third-party's client software uses this API to access hash table stored in FPGA Memcached Server.

Test Bench

The test bench consists with one or multiple c language and Qsys files as a complete software application. This C language application simulates the actual third-party's client software on client computer. It provides a direct access for us and our design user to test and evaluate our Memcached Server. The test bench contains the following specific functions:

1. Communicating with FPGA and Fetching Hash Table

   Test bench validates the hash function API by using it to acquire hash values from FPGA Memcached Server.

2. User Hash Table Update

   It is essential that the client computer is able to update the new hash table value back to Memcached Server accurately. The hash table values in Memcached Server will be able to update by the Test Bench. Given that we are only implementing the Memcached Server, we still decide to use this test bench as a demonstration to show that the hash table updating is reachable for the users and their technicians.

3. Estimate the data decoding/encoding time

The Test Bench will calculate or acquire (calculate in the decoding/encoding phase in FPGA) the time usage of FPGA's decoding/encoding, or acquire from

System Specifications

Inputs/outputs

- Inputs: Designated UDP packet inputs.
- Outputs: Designated UDP packet outputs, Memcached Server Time consumption, Hash Table.

Modules Organization

The modules Organization is showed as below,

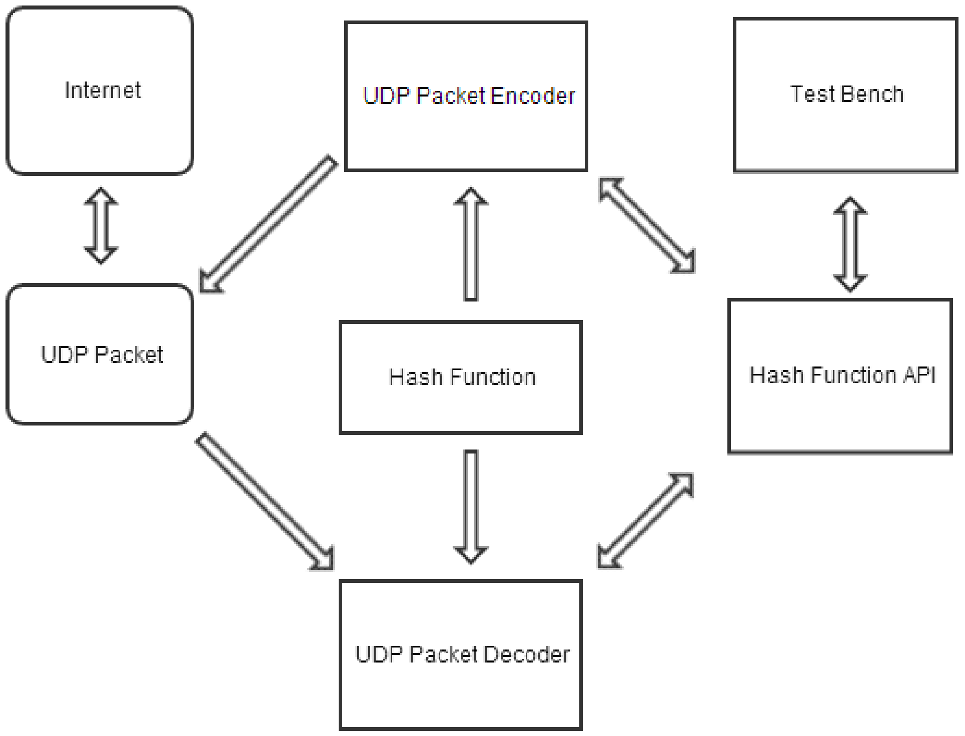

Figure 2 Modules Organization

System Description

The entire system looks like as below. The communication between Memcached Server and client computer is through the LAN.

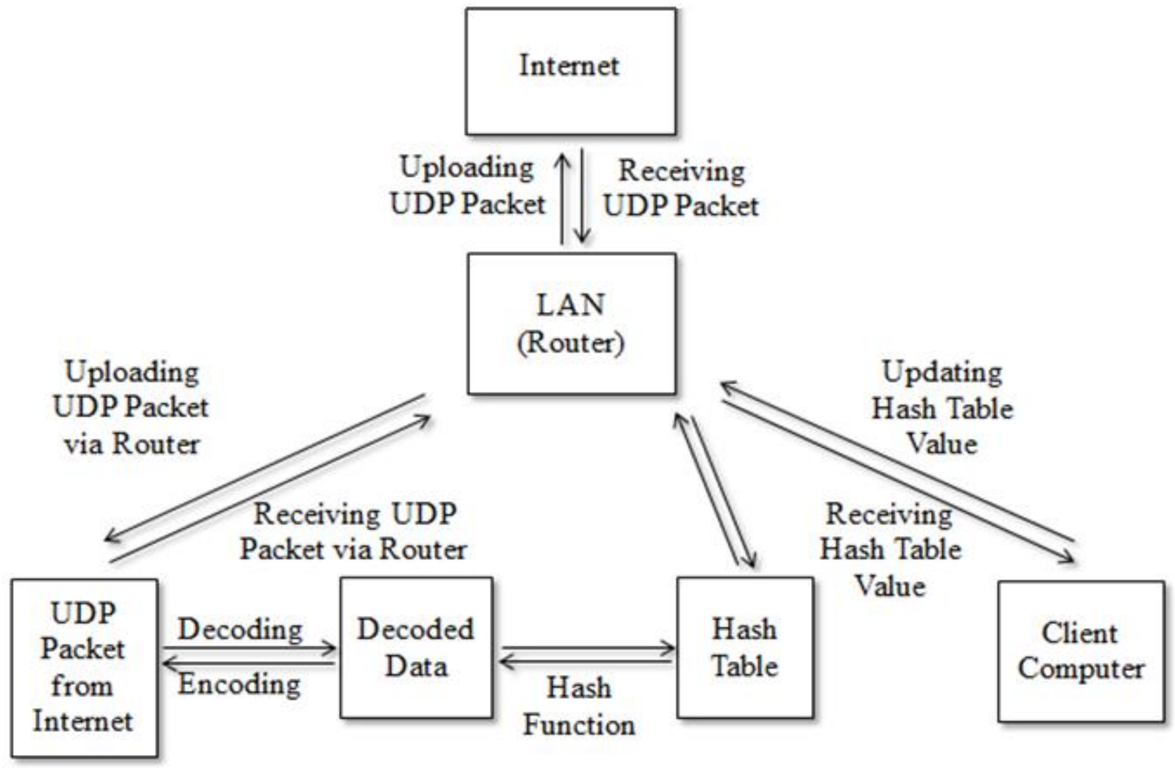

Figure 3 Design Scope

Deliverables

- Memcached Server: SoCKit Cyclone V FPGA.
- An exemplary client computer: Linux Computer

Milestones

- (1) Design Decode.sv
- (2) Feasible Hash Table
- (3) Test Bench Application in C language
- (4) Debugging, Wrapping up