STEPHEN A. EDWARDS
CSEE 4840
Embedded System Design
Spring 2014

FPGA UDP Memcached Server Design

Final Report

May 21st, 2014

Chia-Kai Chou
Meng-Yi Hsu
Chen Wen

1. 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 HDD) must be read. Memcached runs on Unix, Linux, Windows and Mac OS X and is distributed under a new BSD free software license. However, dealing with large amount of ethernet packet (such as 10Gbps), conventional X86 computers has unavoidable execution time consumption caused by the software setting for CPU. For FPGAs, which are directly hardware based computing, the time latency would be eliminated and result in high efficiency in data mining. As a result, we believe that A FPGA Memcached server will best exploit the potentials of Memcached System than building Memcached Server on X86 computers. To exemplify our idea, we decide to use SoCKit FPGA to build a UDP packet Memcached server.

User Datagram Protocol (UDP) is one of the most important members of Internet protocol suite (IPS). In Memcached Server, the decoding UDP packet headers and arranging data storages are the major tasks. FPGA is very flexible in allocating memory and implementing parallel programming. These advantages of FPGA make it ideal to decode UDP packets, generate hash maps and form a Memcache Server.

For massive read/write commands,almost any conventional software programming language has unavoidable execution time consumption in waiting for ALU. Given that building Memcached Server in X86 computer can already speed up execution time for the entire system, building Memcached Server on FPGA can simply avoid this consumption. Unlike any conventional software programming language, Systemverilog programming is a hardware programming language. There will not be any clock cycles wasted on waiting for ALU.

2. Design Goal

The FPGA Memcached Server is able to send and accept any standard Memcached user computer (Memcached Client) command through the Memcached protocol under the UDP header .We assume the Ethernet signals comes from Avalon ST, which is the project deliverable from another design group. Memcached Server decodes IPv4 header, UDP packet and Memcached protocol. With analyzing the commands stores in the Memcached protocol, the Memcached server will store the hash key and hash value into hash table or send proper hash value through a feedback UDP packet upon request.

3. Challenge

- Software:

The protocol in only official document for Memcached UDP is wrong.The document link is shown as the following:
http://memcachedb.googlecode.com/svn/trunk/doc/protocol.txt

The frame header it mentions in the button is wrong: "4-5 Total number of datagrams in this messag." The correct description is "4-5th byte must be hard-coded by '0' and '1' separately," which is the most important part for any kind of UDP request. It is because if we follow the official document's description, we will never get the response from Memcached Server even any error message. Besides, none of us have the background to debug Internet protocol, and it is the starting point for our project to apply the protocol to our hardware design. At first, we try to find the open-source client library for UDP, but there are just a few of them which support UDP and none of them can be compiled without error, which means there is no Memcached client library which support UDP so far. Moreover, when searching the keyword: "Memcached UDP", our project comes in the first two pages in Google Search Engine, which means our team is not only the pioneer in FPGA Memcached server but in Software Memcached UDP debugger. After we look into the source code in great detail, we find the bug and figure out a way to bypass the bug so that we finally can interact with the Memcached server, which means the server will respond any request even some error message, only if we bypass the bug.

- Hardware

When we integrate our final code into FPGA board with the environment to receive Internet package, there is always some bugs appear. Usually, the failure comes from that the Signal Tap couldn't hear any input data from system console by unknown reason. Also, the input pattern from system console isn't totally the same with the real ethernet packet and avalon stream protocol.

3. Design Development

The UDP packet handler (handler.sv) is the Core module in our design. From the Ethernet port of the FPGA board, UDP packets are imported. These UDP packets are consisted of Ethernet Header, IP Header, UDP Header, Data and Ethernet Trailer (Fig. 1). The handler 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 handler module is designed to detect different segments and combine designated data segments. 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.

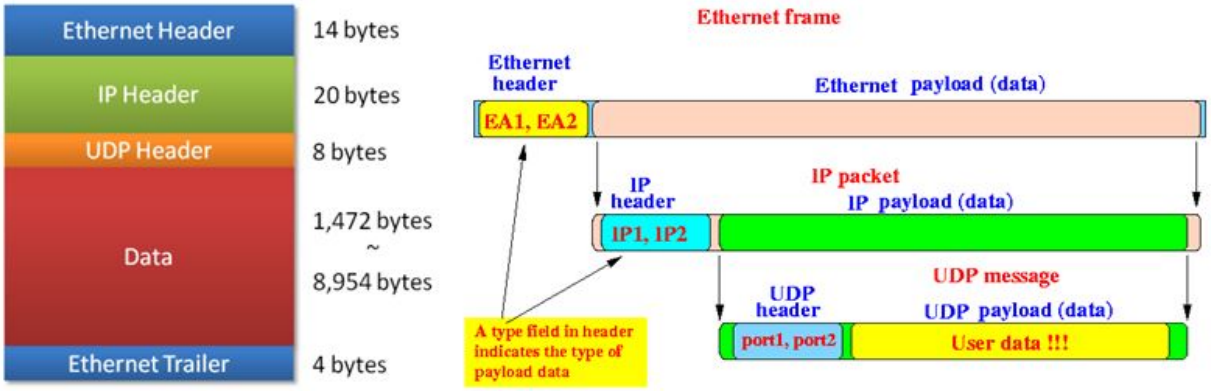


Figure 1, Ethernet Packet and Structure

3.1 Fetching UDP Memcached Protocol

We first implemented a completed UDP Memcached System on our personal computers. Memcached System is primary in used for TCP packet and the information regarding to UDP Memcached System is very limited. To find out the specifications of Memcached Protocol, we built a Memcached System all by ourselves.

First of all, we need to figure out what UDP Memcached Protocol is. Here is the brief introduction. The first of the following three figures shows the whole format of a request and response Memcached UDP packet. The second and the third show the format inside the header in the packet seperately.

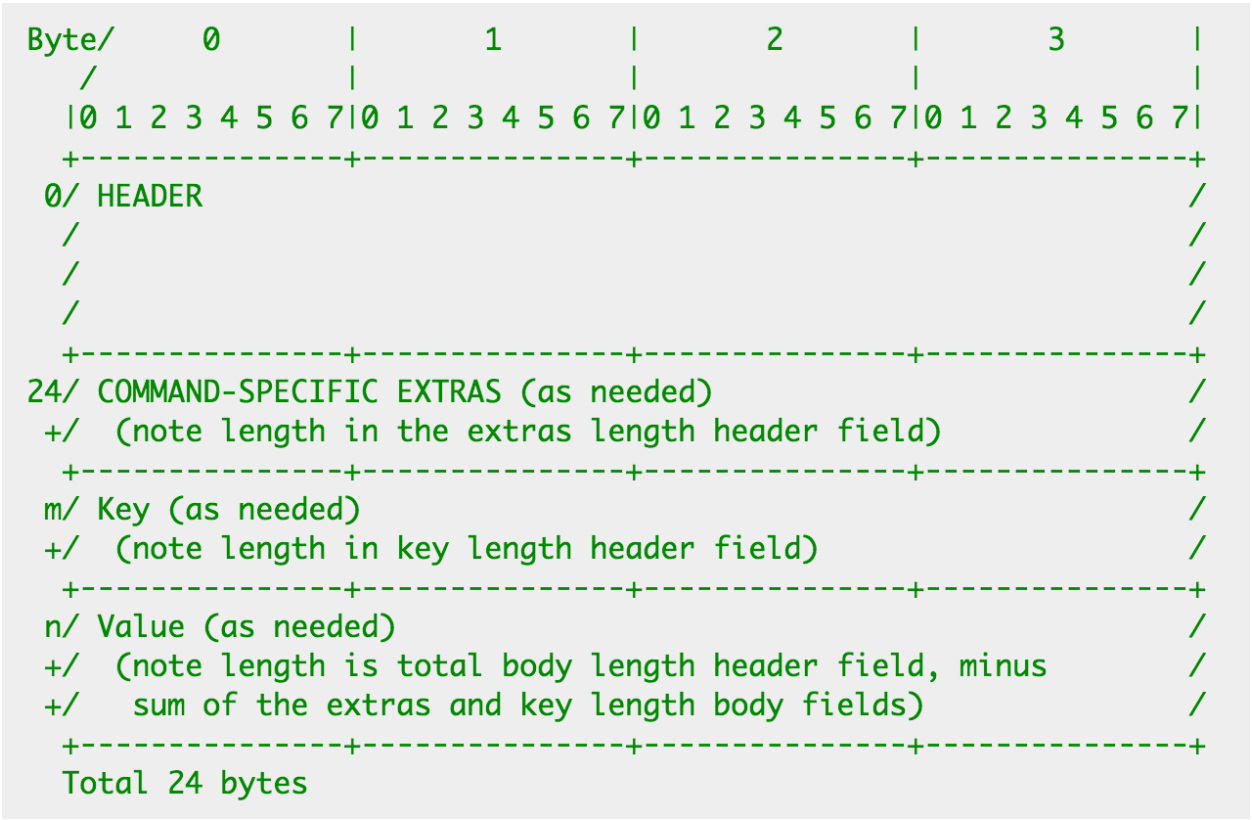


Figure 1.1, General format of a packet

```
Request header:

  Byte/     0       |       1       |       2       |       3       |
     /              |               |               |               |
    |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
    +---------------+---------------+---------------+---------------+
   0| Magic         | Opcode        | Key length                    |
    +---------------+---------------+---------------+---------------+
   4| Extras length | Data type     | Reserved                      |
    +---------------+---------------+---------------+---------------+
   8| Total body length                                             |
    +---------------+---------------+---------------+---------------+
  12| Opaque                                                        |
    +---------------+---------------+---------------+---------------+
  16| CAS                                                           |
    |                                                               |
    +---------------+---------------+---------------+---------------+
    Total 24 bytes
```

Figure 1.2, General format of a Request Header

```
Response header:

  Byte/     0       |       1       |       2       |       3       |
     /              |               |               |               |
    |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
    +---------------+---------------+---------------+---------------+
   0| Magic         | Opcode        | Key Length                    |
    +---------------+---------------+---------------+---------------+
   4| Extras length | Data type     | Status                        |
    +---------------+---------------+---------------+---------------+
   8| Total body length                                             |
    +---------------+---------------+---------------+---------------+
  12| Opaque                                                        |
    +---------------+---------------+---------------+---------------+
  16| CAS                                                           |
    |                                                               |
    +---------------+---------------+---------------+---------------+
    Total 24 bytes

Header fields:
Magic               Magic number.
Opcode              Command code.
Key length          Length in bytes of the text key that follows the
                    command extras.
Status              Status of the response (non-zero on error).
```

Figure 1.3, General format of a Response packet

We discover Memcached Protocol in great detail, shown as below in sequence:

- A extra 64 bit “header” in front of the Memcached header
- 1 byte for “Magic”(i.e. 80 for Request; 81 for Response)
- 1 byte for “Opcode”(i.e. operation, ex. Get, Add, etc)
- A 32 bit “Extras” might be included, which has Flag(i.e. status) and Expiration(i.e. duration)
- A 32 bit for “Opaque”, which is useless
- A 64 bit for "CAS" value, which is kept unique.
- (If necessary) A “Key” (arbitrary string up to 250 bytes in length. No space or new lines)
- (If necessary) A “value” correspond to “Key”

We fetch these protocol fields through the WireShark installed on the client computer. All the protocol fields match with the datasheet we find in the Memcached distributor’s website, except for the first 64 bit, which I have described in detail about this bug in the Challenge part.

3.2 Packet Handler (handler.sv)

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 or update it with new value. The handler encapsulates the data in order to send an UDP packet back to the client computer. The encoding process would be the opposite of the decoding process. The new UDP Header, IP Header, Mac Header and checksum along with the requested data in the Memcached Protocol 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) standard would be followed.

The Avalon Interface (Avalon ST) sends and receives data with 8 bytes (64 bits) every clock cycle with controlling signals startofpacket, endofpacket etc. as Fig. 2. All the headers except for the Memcached protocol are decoded with a FiFo mechanism and store temporary for latter response header’s calculation. Like any

standard UDP encoder, we swap all the necessary header fields (like source MAC/destination MAC) as well as calculate many size fields and command fields.

For our Memcached header implementation, we enable the first two basic functions: get hash value and add hash value.

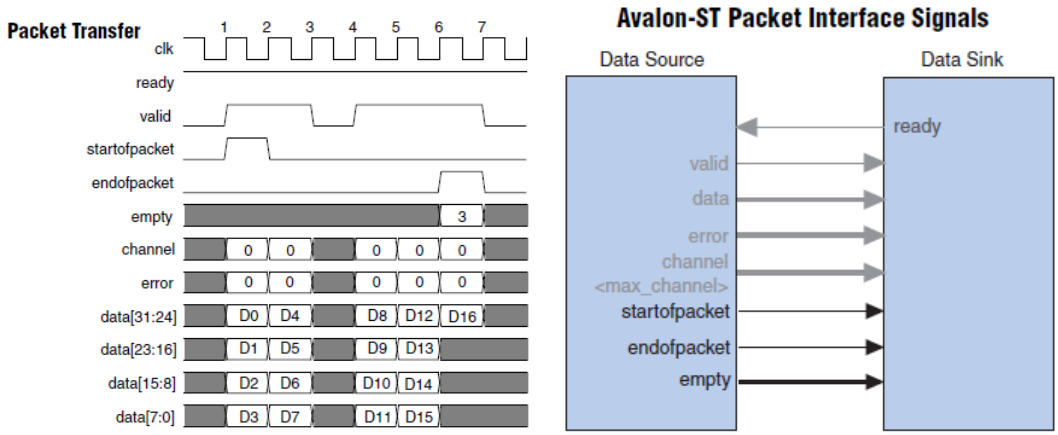
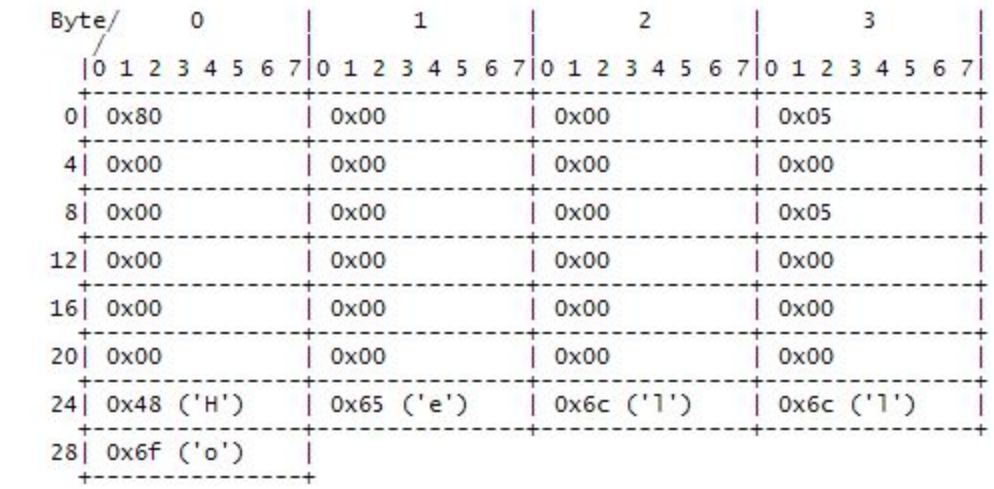


Fig. 2. Avalon-ST packet interface signals and packet transfer schematic.

3.2.1 Get Hash Value

The standard Memcached protocol requires to know the hash key to get the corresponding hash value. The “Get” command in Memcached system has its standard convention. Below is an example request Memcache header from Memcached client, (“hello” is the key)




```
Field        (offset) (value)
 Magic        (0)    : 0x80
 Opcode       (1)    : 0x00
 Key length   (2,3)  : 0x0005
 Extra length (4)    : 0x00
 Data type    (5)    : 0x00
 VBucket      (6,7)  : 0x0000
 Total body   (8-11) : 0x00000005
 Opaque       (12-15): 0x00000000
 CAS          (16-23): 0x0000000000000000
 Extras              : None
 Key          (24-29): The textual string: "Hello"
 Value               : None
```

Figure 2, Get Command Sending Example

and the response header would be look like, (“world” is value)

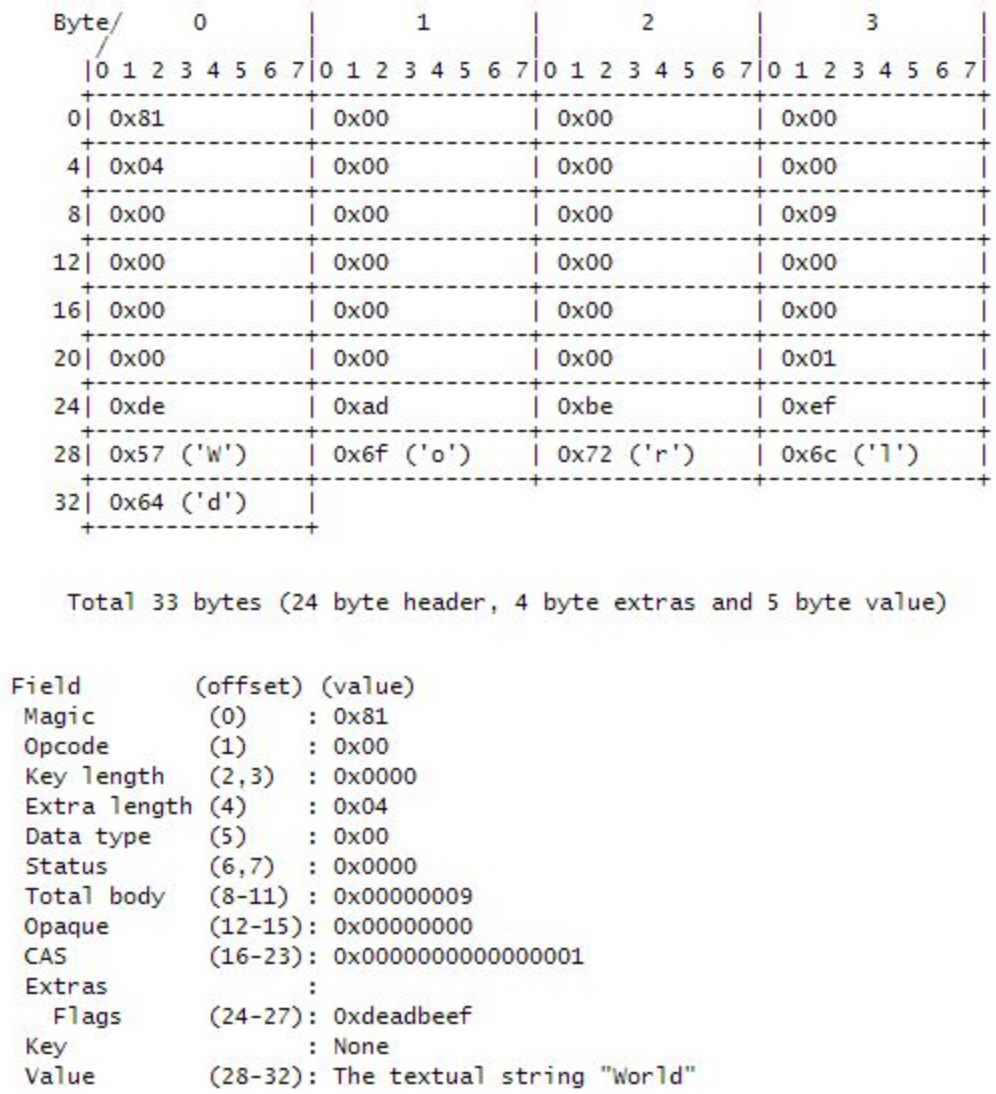


Figure 3, Get Command Responding Example

3.2.2 Add Hash Value

The standard Memcached protocol requires to know both the hash key and hash value to correctly set up data. The “Add” command in Memcached system also has its standard convention. It should be noticed that add command will overwrite any exist value to the corresponding key. Below is an example request Memcache header from Memcached client, (“hello” is the key)

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0| 0x80          | 0x02          | 0x00          | 0x05          |
  +---------------+---------------+---------------+---------------+
 4| 0x08          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
 8| 0x00          | 0x00          | 0x00          | 0x12          |
  +---------------+---------------+---------------+---------------+
12| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
16| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
20| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
24| 0xde          | 0xad          | 0xbe          | 0xef          |
  +---------------+---------------+---------------+---------------+
28| 0x00          | 0x00          | 0x0e          | 0x10          |
  +---------------+---------------+---------------+---------------+
32| 0x48 ('H')    | 0x65 ('e')    | 0x6c ('l')    | 0x6c ('l')    |
  +---------------+---------------+---------------+---------------+
36| 0x6f ('o')    | 0x57 ('W')    | 0x6f ('o')    | 0x72 ('r')    |
  +---------------+---------------+---------------+---------------+
40| 0x6c ('l')    | 0x64 ('d')    |
  +---------------+---------------+

 Total 42 bytes (24 byte header, 8 byte extras, 5 byte key and
                 5 byte value)

Field        (offset) (value)
 Magic        (0)    : 0x80
 Opcode       (1)    : 0x02
 Key length   (2,3)  : 0x0005
 Extra length (4)    : 0x08
 Data type    (5)    : 0x00
 VBucket      (6,7)  : 0x0000
 Total body   (8-11) : 0x00000012
 Opaque       (12-15): 0x00000000
 CAS          (16-23): 0x0000000000000000
 Extras               :
   Flags      (24-27): 0xdeadbeef
   Expiry     (28-31): 0x00000e10
 Key          (32-36): The textual string "Hello"
 Value        (37-41): The textual string "World"
```

Figure 4, Add Command Sending Example

Even though there is one logic in need for an actual feedback. There is still an standard feedback packet to add command,

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0| 0x81          | 0x02          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
 4| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
 8| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
12| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
16| 0x00          | 0x00          | 0x00          | 0x00          |
  +---------------+---------------+---------------+---------------+
20| 0x00          | 0x00          | 0x00          | 0x01          |
  +---------------+---------------+---------------+---------------+

 Total 24 bytes


Field        (offset) (value)
 Magic        (0)    : 0x81
 Opcode       (1)    : 0x02
 Key length   (2,3)  : 0x0000
 Extra length (4)    : 0x00
 Data type    (5)    : 0x00
 Status       (6,7)  : 0x0000
 Total body   (8-11) : 0x00000000
 Opaque       (12-15): 0x00000000
 CAS          (16-23): 0x0000000000000001
 Extras               : None
 Key                  : None
 Value                : None
```

Figure 5, Add Command Responding Example

3.3 Coding Implementation and Algorithm

Ethernet packets consist of fixed 336 bits headers and unfixed-length memcached payloads. In the 336 bits header, several specific bits are restricted to identify the UDP protocol, the total packet length and checksum. In our design, packets that are not in this regulation, which means that it is not UDP protocol or there are missing bits in the packet would be threw away to ‘trash’. The system would then stop and wait until next packet.

If the packet is the ‘correct ’ UDP protocol, then the following 8 bytes would be sequentially assigned to different states in a FSM. Each state gets information we want from the packet and saves it to registers. For fixed length data, it is directly assigned to registers with the same length. However, for data with unfixed length, since we can get the data’s length from the previous 8bytes or 16

bytes, we then count down how many 8 bytes we still need for this data and put arrange them in a relatively long array. For example, for the 'key' in memcached protocol, which has unfixed length, we created a 15 times 8 bytes empty array [63:0]getkey[14:0] to save all the 'keys'. Since we could first know the key length from previous 16 bytes, we could actually count how many 8 bytes are there for the whole key. Whenever we count down 'one' 8 bytes, the key length would be subtracted with 8 and the 8 bytes input data would be save into the big array from the 15th 8 bytes to the last 8 bytes. After we save the whole key into this array, we combine all the sub arrays (getkey[14].....getkey[0]). If this data contains not only 'key'(maybe key+value), then different elements would be parsed from this array and send into different arrays with the same length. In our design, we shift the data in the long array and make the first digit of the data at the left side of the long array. (getkey[319]=first key digit).

When we are sending packet back to the client, the packet is wrapped up and sent as 8 bytes per clock. The idea is almost the same as receive package but is opposite this time. It would be easy to deal with fixed length data. For the unfixed-length data, the data would be taken out from the large array which we created to save them. The data would be shifted 'right' and make only 8 bytes at the right. As a result, every time we just send the 8 bytes data [63:0]. (Remember, we save our data to the left.)

For example, [ABC00...]>>>>[000A], First to send, [A]
[ABC00...]>>>>[000B], Second to send, [B]
[ABC00...]>>>>[000C], Third to send, [C]

3.4 Test Result

We test the entire Memcached system separately in a simulation environment in ModelSim and on the real FPGA board. For the FPGA on-board demonstration depends on the completion of another design group for their success of establishing communication between Ethernet port and Avalon ST interface. To avoid the delay in testing, we are given and a test signal given from the TA. This signal allows us to manually input signals in Quartus's System Console through the Signal Tab. We believe that this signal gives a sufficient test simulation and a validation of the success of our design.

The ModelSim always give excellent results. The feedbacks of both the set and add command`s are in concord with the expectation.

For the on-board testing, we had many times of success but suffer from bugs in the environment which the TA set up for us. The known bugs at our knowledge contains situation such as:

1. We cannot set “0x70”, “0x71”, “0x72”, “0x73”, “0x74”, “0x75”, etc.
2. It is not working with certain signal length.

The simulation signal is consisted with a texture file which contains the packet and a stream.tcl. We discover that this stream.tcl modifies many system files which happening at the compiling time. After some compiling times, the simulation signal stops behaving stable.

One exemplified test result is attached as below. Figure 6.2 shows the simulation of Get Command, which gets the “Key” from the input signal and matches the corresponding value. Figure 6.1 shows the simulation of Add Command. The two highlighted signal(i.e. “addKeyout” and “addValueout”) shows both the “Key” and “Value”, which comes from the input signal, has been saved in FPGA. Figure 6.3 shows one of the successful deployment on FPGA board. We can clearly see the state transfer smoothly as expected.

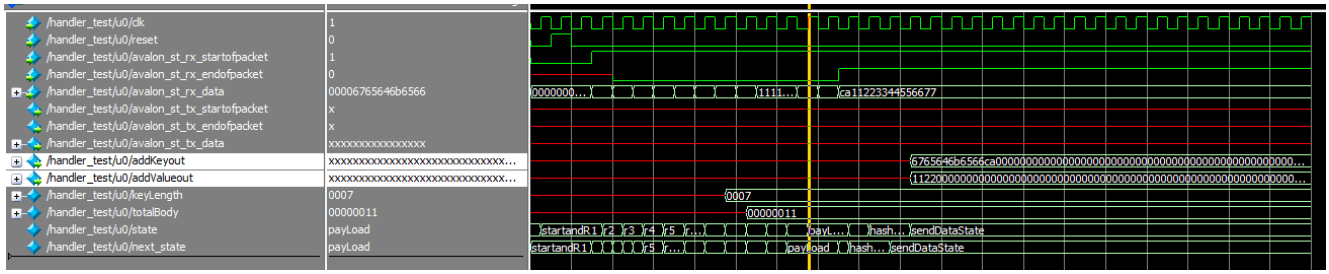


Figure 6.1, The Get Command Result

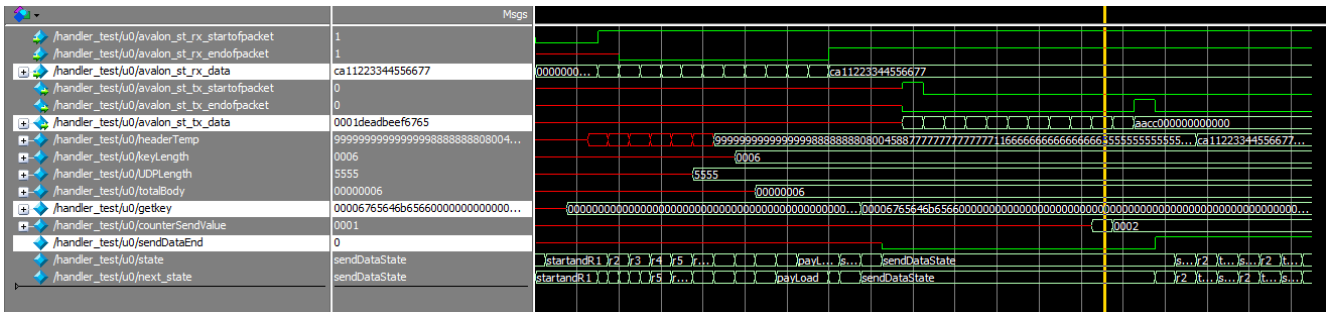


Figure 6.2, The Add Command Result

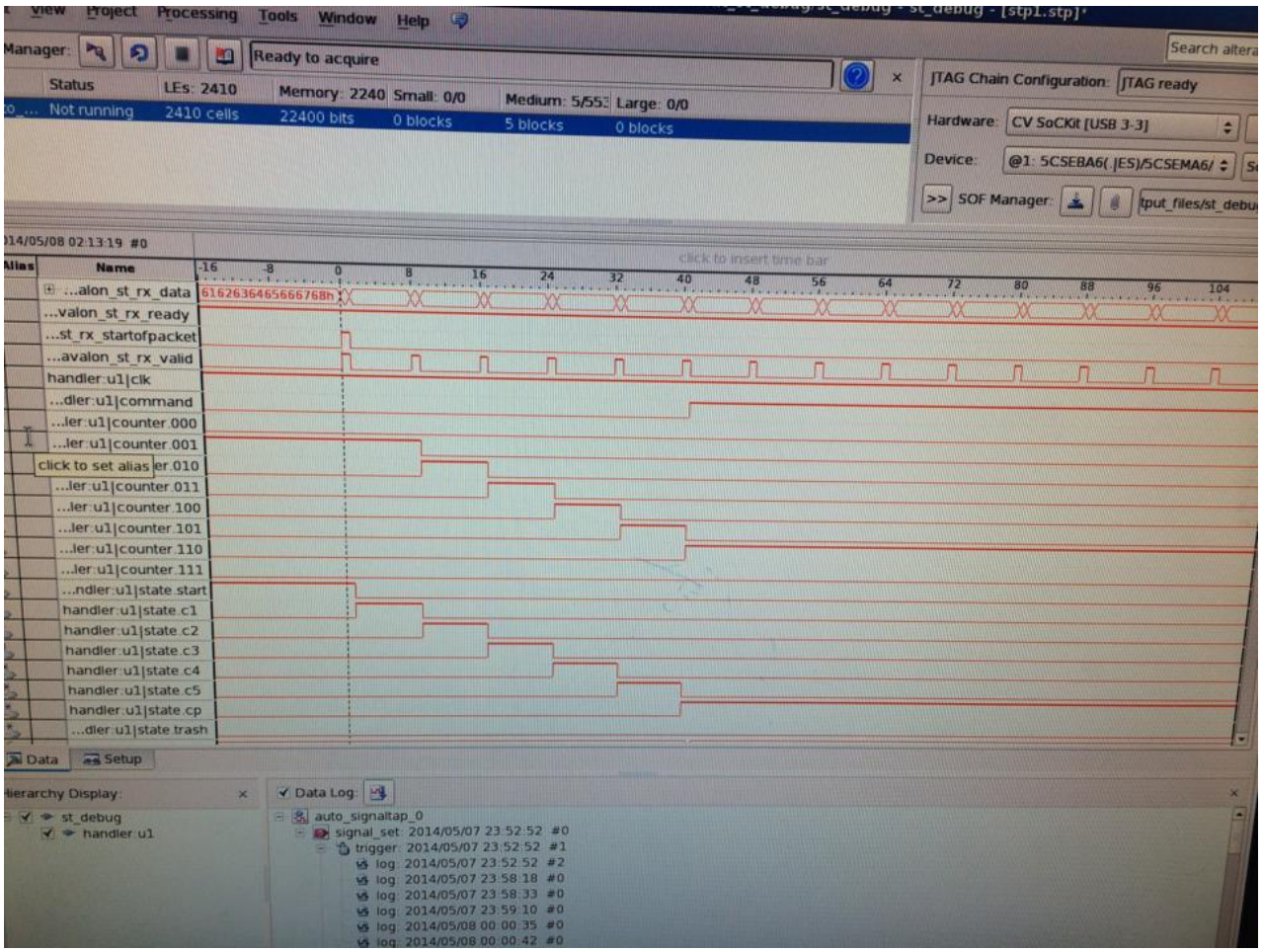


Figure 6.3, The FSM Result on FPGA board

4. System Specification

4.1 Inputs/outputs

Inputs:

- For on-board Test: Send UDP packet manually via System Console.
- For ModelSim: Use the Testbench signal to simulate input.

Outputs:

- For on-board Test: The output from Signal Tab, shown in System Console
- For ModelSim: The output from Testbench.

4.2. System Description

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

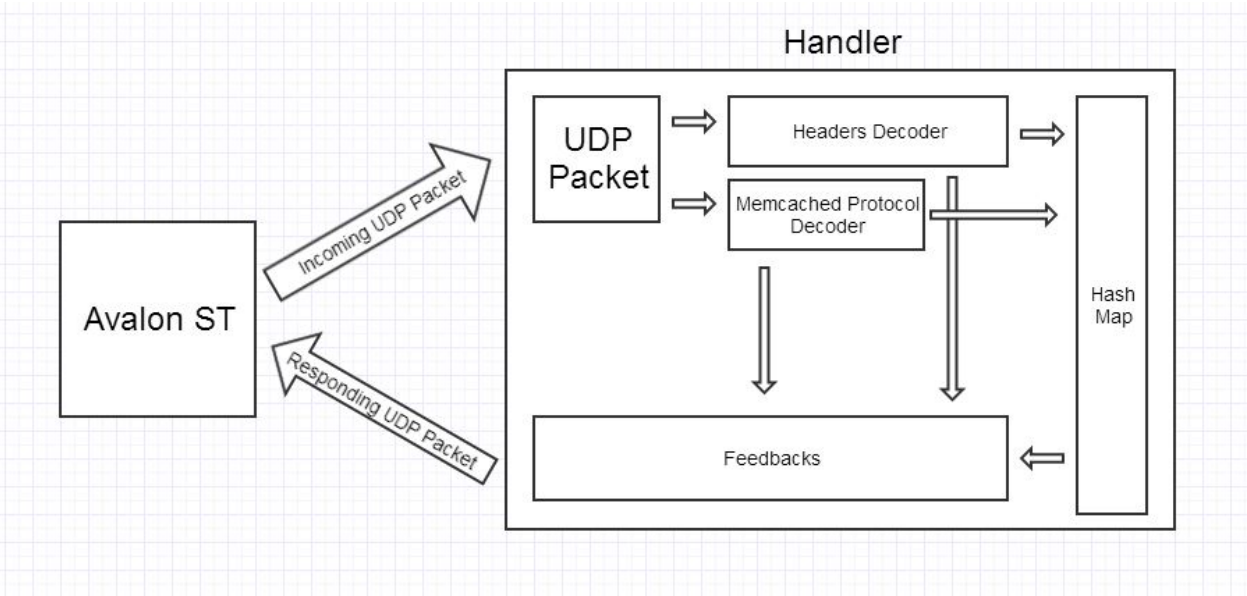


Figure 7, the Entire System

5. Deliverables

All of the code separate into the following three parts :

- Software Memcached UDP Client

```
import java.net.*;
import java.io.*;
import java.util.LinkedList;
import java.util.zip.DataFormatException;

public class MyUdpSender {

    public static void main (String[] args) {


        // Get "Nollo"
        byte[] raw = {
         (byte)0x00,  // 0, Req ID[0]
         (byte)0x01,  //    Req iD[1]
         (byte)0x00,  //    Seq Nm[2]
         (byte)0x01,  //    Seq Nm[3]
         (byte)0x00,  //    Total Byte[0]  Must be 0
         (byte)0x01,  // 5  Total Byte[1]  Must be 1
         (byte)0x00,  //    Must be 0
         (byte)0x00,  //    Must be 0
         (byte)0x80 ,(byte)0x00, (byte)0x00, (byte)0x05, //Key length
         (byte)0x00 ,(byte)0x00, (byte)0x00, (byte)0x00, //Extras length
         (byte)0x00 ,(byte)0x00, (byte)0x00, (byte)0x05, //Total body length (including Key, Value)
         (byte)0x00 ,(byte)0x00, (byte)0x00, (byte)0x00, //Opaque
         (byte)0x00 ,(byte)0x00, (byte)0x00, (byte)0x00, //CAS
         (byte)0x00 ,(byte)0x00, (byte)0x00, (byte)0x00, //CAS
         (byte)0x4e ,(byte)0x6f, (byte)0x6c, (byte)0x6c, //"Nollo"  (Key)
         (byte)0x6f
        };

        // Add "Nollo"  "World"
        byte[] raw2 = {
         (byte)0x00,  // 0, Req ID[0]
         (byte)0x01,  //    Req iD[1]
         (byte)0x00,  //    Seq Nm[2]
         (byte)0x01,  //    Seq Nm[3]
         (byte)0x00,  //    Total Byte[0]  Must be 0
         (byte)0x01,  // 5  Total Byte[1]  Must be 1
         (byte)0x00,  //    Must be 0
         (byte)0x00,  //    Must be 0
         (byte)0x80 , (byte)0x02 ,(byte)0x00 ,(byte)0x05 , //Key length
         (byte)0x08 , (byte)0x00 ,(byte)0x00 ,(byte)0x00 , //Extras length
         (byte)0x00 , (byte)0x00 ,(byte)0x00 ,(byte)0x12 , //Total body length (including Extras, Expiration, Key, Value)
         (byte)0x00 , (byte)0x00 ,(byte)0x00 ,(byte)0x00 , //Opaque
         (byte)0x00 , (byte)0x00 ,(byte)0x00 ,(byte)0x00 , //CAS
         (byte)0x00 , (byte)0x00 ,(byte)0x00 ,(byte)0x00 , //Extras (Flags and Expiration)
         (byte)0xde , (byte)0xad ,(byte)0xbe ,(byte)0xef , //Flags    Key
         (byte)0x00 , (byte)0x00 ,(byte)0x0e ,(byte)0x10 , //Expiration
         (byte)0x4e , (byte)0x6f ,(byte)0x6c ,(byte)0x6c , //"Nollo"  (Key)
         (byte)0x6f , (byte)0x57 ,(byte)0x6f ,(byte)0x72 , //"World"  (Value)
         (byte)0x6c , (byte)0x64
        };
```

```
        try {
            DatagramPacket packet = new DatagramPacket (raw, raw.length, InetAddress.getLoopbackAddress () , 11211);
            DatagramSocket socket = new DatagramSocket (5566);
            socket.send (packet);
            socket.close ();

            while (true) {
                byte[] receiveData = new byte[64000];
                DatagramPacket receivePacket = new DatagramPacket (receiveData, receiveData.length);
                socket.receive (receivePacket);
                System.out.println ("Receiving ...");
            }

        } catch (Exception e) {
            System.out.println ("Ooops - " + e);
            e.printStackTrace ();
        }

    }
}
```

- Memcached UDP Handler

```
module handler (
    input logic clk, reset,
    input logic avalon_st_rx_startofpacket,
    input logic avalon_st_rx_endofpacket,
    input logic avalon_st_rx_valid,
    input logic [63:0] avalon_st_rx_data,
    input logic [2:0] avalon_st_rx_empty,
    input logic [5:0] avalon_st_rx_error,
    input logic avalon_st_tx_ready, // if Avalon-ST is ready to send packet

    output logic avalon_st_rx_ready, // tell Avalon-ST ready to accept packet
    output logic avalon_st_tx_startofpacket,
    output logic avalon_st_tx_endofpacket,
    output logic avalon_st_tx_valid,
    output logic [63:0] avalon_st_tx_data,
    output logic [2:0] avalon_st_tx_empty,
    output logic avalon_st_tx_error,
    output logic [959:0]addKeyout,
    output logic [959:0]addValueout
    );

    logic [335:0] headerTemp; //temporary store the incoming header
    logic [15:0] keyLength; // store the key length, 1 means 1 char
    logic [31:0] valueLength;
    logic [16:0] kL;
    logic [31:0] keyValueLen;
    logic [31:0] kvL;
    logic [1:0] command; // 0 for get, 1 for set
    logic [15:0] UDPLength;
    logic sendData;
    logic [3:0]sendDataCounter;
    logic flag;
    logic [31:0] totalBody;
    logic [63:0] getkey1[4:0];
    logic [63:0] addKeyValue1[14:0];
    logic [2:0] countkL;
    logic [3:0] countkvL;
    logic [319:0] getkey;
    logic [959:0] addKeyValue;
    logic [15:0] sendvaluelength;
    logic [959:0] sendvalue;
    logic [959:0] sendvalue2;
    logic [2:0] counterhash;
    logic [959:0] addKey;
    logic [959:0] addValue;
    logic [15:0]counterSendValue;
    logic sendDataEnd;

    enum logic [4:0] {
        startandR1 = 5'd1,
        trash = 5'd2,
        r2 = 5'd3,
        r3 = 5'd4,
```

```
    r4 = 5'd5,
    r5 = 5'd6,
    r6andMemProt1 = 5'd7,
    memProt2 = 5'd8,
    memProt3 = 5'd9,
    memProt4 = 5'd10,
    memProt5 = 5'd11,
    memProt5Pa=5'd12,
    payLoad=5'd13,
    splitData=5'd14,
    hashtable=5'd15,
    sendDataState=5'd16
    } state, next_state;

always_ff @ (posedge clk) begin
    if (reset) begin
            state <= startandR1;
    end

    else begin
            state <= next_state;
    end
end

always_comb begin
    next_state = state;
    case (state)

  startandR1 : begin
                    if(avalon_st_rx_valid && avalon_st_startofpacket)    next_state = r2;
                    else  next_state = startandR1;
                end

     r2    :    begin
                    if (avalon_st_rx_valid && avalon_st_rx_data[31:8]==24'h080045)      next_state = r3;
                    else if (avalon_st_rx_valid && avalon_st_rx_data[31:8]!=24'h080045)   next_state = trash;
                    else if (avalon_st_rx_valid==0)               next_state = r2;
                end

     r3    :    begin
                    if (avalon_st_rx_valid && avalon_st_rx_data[7:0]==8'h11)   next_state = r4;
                   else if (avalon_st_rx_valid && avalon_st_rx_data[7:0]!=8'h11)  next_state = trash;
                   else if (avalon_st_rx_valid==1'b0)   next_state = r3;
                end

     r4    :    begin
                    if (avalon_st_rx_valid)     next_state = r5;
                    else  next_state = r4;
                end

     r5    :   begin
                    if (avalon_st_rx_valid)       next_state = r6andMemProt1;
                    else  next_state = r5;
```

```
                end

    r6andMemProt1 : begin
                    if (avalon_st_rx_valid)  next_state = memProt2 ;
                   else   next_state = r6andMemProt1;
                end

    memProt2   :   begin
                    if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b0)
                    next_state = memProt3;
                else if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b1)
                next_state = startandR1;
                    else
                    next_state = memProt2;
                end

    memProt3   :   begin
                    if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b0)
                next_state = memProt4;
                else if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b1)
                next_state = startandR1;
                    else
                    next_state = memProt3;
                end

   memProt4   :    begin
                    if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b0) begin
                    case(command)
                         2'b01:
                          next_state = memProt5Pa;
                         2'b10:
                          next_state = memProt5;
                    endcase
                    end
                    else if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b1)
                        next_state = startandR1;
                    else
                        next_state = memProt4;
                end

   memProt5   :    begin
                    if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b0)
                        next_state = memProt5Pa;
                    else if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b1)
                        next_state = startandR1;
                    else
                        next_state = memProt5;
                end
   memProt5Pa  :    begin
                    if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b0 && flag==1)
                next_state = payLoad;
                else if (avalon_st_rx_valid==1'b1 && avalon_st_rx_endofpacket==1'b1 && flag==1)
```

```
                    next_state = startandR1;
                        else if (avalon_st_rx_valid==1'b0 && flag==1)
                      next_state = memProt5Pa;
                        else next_state = trash;
                       end
    payLoad     :  begin
                        if (avalon_st_rx_valid && avalon_st_rx_endofpacket==1'b0)
                        next_state = payLoad;
                       else if (avalon_st_rx_valid && avalon_st_rx_endofpacket==1'b1)
                        next_state = splitData;
                       else
                        next_state = payLoad;
                   end

     splitData:    begin
                        next_state = hashtable;
                    end

    hashtable:      begin
                        case(command)
                            2'b01:begin
                                    next_state = sendDataState;
                                    end
                            2'b10:begin
                                    if (counterhash==3'd2) begin
                                    next_state = sendDataState;
                                    end
                                    else next_state = hashtable;
                                    end
                        endcase
                    end

    sendDataState: begin
                        if (sendDataEnd==1'b0) begin
                            next_state = sendDataState;
                        end
                        else next_state =  startandR1;
                    end

     trash :        begin
                        if(avalon_st_rx_endofpacket==1'b1)  next_state = startandR1;
                        else if (avalon_st_rx_endofpacket==1'b1) next_state = startandR1;
                    end

    default :       begin
                        next_state = startandR1;
                    end
    endcase
end
```

```
always_ff @ (posedge clk) begin  //read in two floating numbers from master in READ state
    if  (reset) begin
        getkey1[4]<=64'd0;
        getkey1[3]<=64'd0;
        getkey1[2]<=64'd0;
        getkey1[1]<=64'd0;
        getkey1[0]<=64'd0;
        getkey[319:0]<=320'd0;
        addKeyValue1[14]<=64'd0;
        addKeyValue1[13]<=64'd0;
        addKeyValue1[12]<=64'd0;
        addKeyValue1[11]<=64'd0;
        addKeyValue1[10]<=64'd0;
        addKeyValue1[9]<=64'd0;
        addKeyValue1[8]<=64'd0;
        addKeyValue1[7]<=64'd0;
        addKeyValue1[6]<=64'd0;
        addKeyValue1[5]<=64'd0;
        addKeyValue1[4]<=64'd0;
        addKeyValue1[3]<=64'd0;
        addKeyValue1[2]<=64'd0;
        addKeyValue1[1]<=64'd0;
        addKeyValue1[0]<=64'd0;
        addKeyValue[959:0]<=960'd0;
        avalon_st_rx_ready <= 1'b1;
    end

    else begin
        case(state)
    startandR1  : begin
                    headerTemp[335:272] <= avalon_st_rx_data[63:0];
                  end

        r2   : begin
                  if (avalon_st_rx_valid==1'b1 && avalon_st_rx_data[31:8]==24'h080045) begin
                      headerTemp[271:208] <= avalon_st_rx_data[63:0];
                   end
               end



        r3   : begin
                   if (avalon_st_rx_valid==1'b1 && avalon_st_rx_data[7:0]==8'h11) begin
                    headerTemp[207:144] <= avalon_st_rx_data[63:0];
                  end
               end

        r4   : begin
                  if (avalon_st_rx_valid==1'b1) begin
                  headerTemp[143:80] <= avalon_st_rx_data[63:0];
                end
               end
```

```
        r5   : begin
                 if (avalon_st_rx_valid==1'b1) begin
                    headerTemp[79:16] <= avalon_st_rx_data[63:0];
                    UDPLength <= avalon_st_rx_data[15:0];
                 end
               end

r6andMemProt1: begin
                 if (avalon_st_rx_valid==1'b1) begin
                     headerTemp[15:0] <= avalon_st_rx_data[15:0];
                 end
               end

    memProt2  : begin
                    if (avalon_st_rx_valid==1'b1) begin
                    case (avalon_st_rx_data[47:32])
                          16'h8000: begin // get command
                                      keyLength[15:0] <= avalon_st_rx_data[31:16];
                                      command <= 2'b01;
                                    end

                          16'h8002: begin // add command
                                      keyLength[15:0] <= avalon_st_rx_data[31:16];
                                      command <= 2'b10;
                                    end
                    endcase
                    end
                 end

    memProt3  : begin
                    if (avalon_st_rx_valid==1'b1) begin
                        case (command)
                        2'b01: begin // get command
                                    totalBody[31:0] <= avalon_st_rx_data[47:16];
                                    kL <= keyLength;
                                end

                        2'b10: begin // add command
                                    totalBody[31:0] <= avalon_st_rx_data[47:16];
                                    keyValueLen[31:0] <= avalon_st_rx_data[47:16]-32'd8;
                                    kvL[31:0] <= (avalon_st_rx_data[47:16]-32'd8);
                                end
                        endcase
                    end
                 end

    memProt4  : begin
                 if (avalon_st_rx_valid==1'b1) begin
                   case (command)
                   2'b01: begin
                                flag <= 1'd1;
                          end
```

```
                2'b10: begin
                          if (totalBody[31:0]<32'd8) begin
                           flag <= 1'd0;
                          end
                          else if (totalBody[31:0]>=32'd8 && keyValueLen[31:0]<{16'd0,keyLength[15:0]})
                           flag <= 1'd0;
                          else flag <= 1'd1;
                         end
                endcase
              end
              end

 memProt5  : begin          //wait for a clock (for add)
                         end

memProt5Pa: begin
              if (avalon_st_rx_valid==1'b1) begin
                case (command)
                  2'b01: begin //get
                      if (keyLength > 16'd5) begin
                          getkey1[4] <= avalon_st_rx_data[47:0];
                          kL <= kL-16'd6;
                          countkL <= 3'd1;
                      end
                      else begin
                          case (keyLength)
                              16'd1: getkey1[4] <= avalon_st_rx_data[47:40];
                              16'd2: getkey1[4] <= avalon_st_rx_data[47:32];
                              16'd3: getkey1[4] <= avalon_st_rx_data[47:24];
                              16'd4: getkey1[4] <= avalon_st_rx_data[47:16];
                              16'd5: getkey1[4] <= avalon_st_rx_data[47:8];
                          endcase
                      end
                  end

                  2'b10: begin //add
                      if (keyValueLen > 32'd5) begin
                          addKeyValue1[14] <= avalon_st_rx_data[47:0];
                          kvL <= kvL-32'd6;
                          countkvL <= 4'd1;
                      end
                      else begin
                          case (keyValueLen)
                              32'd1: addKeyValue1[14] <= avalon_st_rx_data[47:40];
                              32'd2: addKeyValue1[14] <= avalon_st_rx_data[47:32];
                              32'd3: addKeyValue1[14] <= avalon_st_rx_data[47:24];
                              32'd4: addKeyValue1[14] <= avalon_st_rx_data[47:16];
                              32'd5: addKeyValue1[14] <= avalon_st_rx_data[47:8];
                          endcase
                      end

                  end
```

```
                endcase
             end
            end

payLoad: begin //payLoad would be la continuing loop until meet endofpacket
            if (avalon_st_rx_valid==1'b1) begin
                counterhash <= 3'd1;
                case (command)
                    2'b01: begin //get
                        if (kL > 16'd7) begin
                            getkey1[3'd4-countkL]<= avalon_st_rx_data[63:0];
                            kL <= kL-16'd8;
                            countkL <= countkL+3'd1;
                        end
                        else if (kL <= 16'd7 && keyLength > 16'd6)begin
                            case (kL)
                                16'd1: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:56],56'd0};
                                16'd2: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:48],48'd0};
                                16'd3: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:40],40'd0};
                                16'd4: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:32],32'd0};
                                16'd5: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:24],24'd0};
                                16'd6: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:16],16'd0};
                                16'd7: getkey1[3'd4-countkL]<= {avalon_st_rx_data[63:8],8'd0};
                            endcase
                        end

                       end
                    2'b10: begin
                        if (kvL > 32'd7) begin
                            addKeyValue1[4'd14-countkvL]<= avalon_st_rx_data[63:0];
                            kvL <= kvL-32'd8;
                            countkvL <= countkvL+15'd1;
                        end
                        else if (kvL <= 32'd7 && keyValueLen > 32'd6) begin
                            case (kvL)
                                32'd1: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:56],56'd0};
                                32'd2: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:48],48'd0};
                                32'd3: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:40],40'd0};
                                32'd4: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:32],32'd0};
                                32'd5: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:24],24'd0};
                                32'd6: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:16],16'd0};
                                32'd7: addKeyValue1[4'd14-countkvL]<= {avalon_st_rx_data[63:8],8'd0};
                            endcase
                        end
                       end
                    endcase

                end
            end
```

```
splitData: begin
          sendDataCounter <= 4'b0;
          case(command)
               2'b01: begin
                         getkey[319:0] <={getkey1[4],getkey1[3],getkey1[2],getkey1[1],getkey1[0]};
                         end
                   2'b10: begin
                         addKeyValue[959:0] <= {addKeyValue1[14],addKeyValue1[13],addKeyValue1[12],addKeyValue1[11],
                         addKeyValue1[10],addKeyValue1[9],addKeyValue1[8],addKeyValue1[7],addKeyValue1[6],addKeyValue1[5],
                         addKeyValue1[4],addKeyValue1[3],addKeyValue1[2],addKeyValue1[1],addKeyValue1[0]};
                    end
               endcase
           end

hashtable: begin
           sendDataEnd <= 1'b0;
           case(command)
                2'b01:begin
                      case (getkey[319:0]) //get (counterhash==1)
                        {16'd0,48'h6765646b6566,256'd0}: begin
                                                         sendvaluelength <= 16'd12;
                                                         sendvalue <= {96'h67657476616c75650102aacc,864'b0};
                                                         end
                        {16'd0,48'h6765746b6578,256'd0}: begin
                                                         sendvaluelength <= 16'd12;
                                                         sendvalue <= {96'h67657476616c756502020201,864'b0};
                                                         end
                        {16'd0,48'h6765746b6579,256'd0}: begin
                                                         sendvaluelength <= 16'd12;
                                                         sendvalue <= {96'h67657476616c756503030301,864'b0};
                                                         end
                        endcase
                       end
           2'b10:begin
                case (counterhash)
                3'd1:
                    begin
                        addKey<= (addKeyValue>>(16'd943-16'd8*keyLength[15:0])) <<(16'd959-16'd8*keyLength[15:0]); //addkey
                        addValue<=addKeyValue<<(16'd16+16'd8*keyLength[15:0]);//addvalue
                        counterhash <= 3'd2;
                    end
                3'd2:
                    begin
                        addKeyout<=addKey;
                       addValueout<=addValue;
                        counterhash <= 3'd2;
                    end
                endcase
               end
           endcase
        end
```

```
sendDataState: begin
               if (avalon_st_tx_ready==1) begin //client ready for sending back value
               case (sendDataCounter)
                        4'd0: begin
                                avalon_st_tx_error <= 1'd0;
                                avalon_st_tx_valid <= 1'd1;
                                avalon_st_tx_startofpacket <= 1'd1;
                                avalon_st_tx_endofpacket <= 1'd0;
                                avalon_st_tx_data [63:16] <= headerTemp[287:240];
                                avalon_st_tx_data [15:0] <= headerTemp[335:320];
                                sendDataCounter <= 4'd1;
                              end

                        4'd1: begin
                                avalon_st_tx_error <= 1'd0;
                                avalon_st_tx_valid <= 1'd1;
                                avalon_st_tx_startofpacket <= 1'd0;
                                avalon_st_tx_endofpacket <= 1'd0;
                                avalon_st_tx_data [63:32] <= headerTemp[319:288];
                                avalon_st_tx_data [31:0] <=  {24'h080045,headerTemp[215:208]};
                                sendDataCounter <= 4'd2;
                              end

                        4'd2: begin
                                avalon_st_tx_error <= 1'd0;
                                avalon_st_tx_valid <= 1'd1;
                                avalon_st_tx_startofpacket <= 1'd0;
                                avalon_st_tx_endofpacket <= 1'd0;
                                case (command)
                                    /*ipv4 Length setting, "4" is the "deadbeef"*/
                                    2'b01: avalon_st_tx_data [63:48] <= (16'd624+8*sendvaluelength[15:0]); // get
                                    2'b10: avalon_st_tx_data [63:48] <= 16'd592; // add
                                endcase
                                avalon_st_tx_data [47:8] <= headerTemp[191:152];
                                avalon_st_tx_data [7:0] <= 8'h11;
                                sendDataCounter <= 4'd3;
                              end

                        4'd3: begin
                                avalon_st_tx_error <= 1'd0;
                                avalon_st_tx_valid <= 1'd1;
                                avalon_st_tx_startofpacket <= 1'd0;
                                avalon_st_tx_endofpacket <= 1'd0;
                                avalon_st_tx_data[63:48] <= 16'd0;//IPV4CHECKSUM;
                                avalon_st_tx_data[47:16] <= headerTemp[95:64];
                                avalon_st_tx_data[15:0] <= headerTemp[191:176];
                                sendDataCounter <= 4'd4;
                               end
```

```
4'd4: begin
        avalon_st_tx_error <= 1'd0;
        avalon_st_tx_valid <= 1'd1;
        avalon_st_tx_startofpacket <= 1'd0;
        avalon_st_tx_endofpacket <= 1'd0;
        avalon_st_tx_data[63:48] <= headerTemp[175:160];
        avalon_st_tx_data[47:32] <= headerTemp[47:32];
        avalon_st_tx_data[31:16] <= headerTemp[63:48];
        sendDataCounter <= 4'd5;
        case (command)
            /*udp Length setting, "4" is the "deadbeef"*/
            2'b01: avalon_st_tx_data [15:0] <= (16'd560+16'd8*sendvaluelength[15:0]); // get
            2'b10: avalon_st_tx_data [15:0] <= 16'd528; // add
        endcase
       end

4'd5: begin //end of sending header in here
        avalon_st_tx_error <= 1'd0;
        avalon_st_tx_valid <= 1'd1;
        avalon_st_tx_startofpacket <= 1'd0;
        avalon_st_tx_endofpacket <= 1'd0;
        avalon_st_tx_data[63:48] <= 16'b0;//UDPCHECKSUM; //end of sending header
        avalon_st_tx_data[47:0] <= 48'h000100000001; //send the "extra 8 bytes" get from WireShark
        sendDataCounter <= 4'd6;
      end

4'd6: begin
        avalon_st_tx_data[63:48] <= 16'd0;  //send the "extra 8 bytes" get from WireShark
        sendDataCounter <= 4'd7;
        case (command) //start of memcache protocol
            2'b01: avalon_st_tx_data [47:0] <= 48'h810000000400; //get
            2'b10: avalon_st_tx_data [47:0] <= 48'h810200000000; //add
        endcase
      end

4'd7: begin
        avalon_st_tx_error <= 1'd0;
        avalon_st_tx_valid <= 1'd1;
        avalon_st_tx_startofpacket <= 1'd0;
        avalon_st_tx_endofpacket <= 1'd0;
        case (command)
            2'b01: begin//get
                avalon_st_tx_data [63:48] <= 16'd0;
                avalon_st_tx_data [47:16] <= (16'd4 +{16'b0,sendvaluelength}); //"4" -> deadbeef
                avalon_st_tx_data [15:0] <= 16'd0;
                    end
            2'b10: avalon_st_tx_data [63:0] <= 16'd0; //add
        endcase
        sendDataCounter <= 4'd8;
      end
```

```
4'd8: begin
        avalon_st_tx_error <= 1'd0;
        avalon_st_tx_valid <= 1'd1;
        avalon_st_tx_startofpacket <= 1'd0;
        avalon_st_tx_endofpacket <= 1'd0;
        case (command) //get and add are the same but keep for better future modification
            2'b01: avalon_st_tx_data [63:0] <= 64'h0000000000000000; //get
            2'b10: avalon_st_tx_data [63:0] <= 64'h0000000000000000; //add
        endcase
        sendDataCounter <= 4'd9;
      end

4'd9: begin
        avalon_st_tx_error <= 1'd0;
        avalon_st_tx_valid <= 1'd1;
        avalon_st_tx_startofpacket <= 1'd0;

        case (command)
            2'b01: begin     //get
                avalon_st_tx_data [63:16] <= 48'h0001deadbeef;
                if  (sendvaluelength > 16'd1) begin
                    avalon_st_tx_data [15:0] <= sendvalue[959:944]; //start to send data
                  sendvaluelength <= (sendvaluelength-16'd2);
                  sendvalue <= {sendvalue[943:0],16'd0};
                  counterSendValue <= 16'd1;
                  sendDataCounter <= 4'd10;
                end
                else  begin
                    avalon_st_tx_endofpacket <= 1'd1;
                    avalon_st_tx_data [15:0] <= {sendvalue[959:952],8'd0};
                    avalon_st_tx_empty <= 3'b001;//1 byte is empty
                    sendDataCounter <= 4'd11;
                end
                end

            2'b10: begin
                    avalon_st_tx_endofpacket <= 1'd1;//add
                    avalon_st_tx_data [63:0] <= {16'h0001,48'd0};
                    avalon_st_tx_empty <= 3'd6; //6 bytes are empty
                    sendDataCounter <= 4'd11;
                   end
            endcase
         end
```

```
                    4'd10: begin// only for get
                            avalon_st_tx_error <= 1'd0;
                            avalon_st_tx_valid <= 1'd1;
                            avalon_st_tx_startofpacket <= 1'd0;
                            if(sendvaluelength>16'd7) begin
                                avalon_st_tx_endofpacket <= 1'd0;
                                avalon_st_tx_data[63:0] <= sendvalue[959:0]>>(16'd960-(16'd64*counterSendValue));
                                sendvalue2[959:0] <= sendvalue[959:0]>>(16'd960-(16'd64*counterSendValue)-16'd8*(sendvaluelength-16'd8));
                                counterSendValue <= (counterSendValue+16'd1);
                                sendvaluelength <= (sendvaluelength - 16'd8);
                                //send to the next 64 bits value, valueLengthCP is not in use any more
                                sendDataCounter <= 4'd10; //continues sending data
                            end else begin
                                avalon_st_tx_endofpacket <= 1'd1;
                                sendDataCounter <= 4'd11;
                                case (sendvaluelength)
                                    16'd1: begin
                                        avalon_st_tx_data <= {sendvalue2[7:0],56'd0};
                                        avalon_st_tx_empty <= 3'b111; //7 bytes empty
                                    end
                                    16'd2: begin
                                        avalon_st_tx_data <= {sendvalue2[15:0],48'd0};
                                        avalon_st_tx_empty <= 3'b110; //6 bytes empty
                                    end
                                    16'd3: begin
                                        avalon_st_tx_data <= {sendvalue2[23:0],40'd0};
                                        avalon_st_tx_empty <= 3'b101; //5 bytes empty
                                    end
                                    16'd4: begin
                                        avalon_st_tx_data <= {sendvalue2[31:0],32'd0};
                                        avalon_st_tx_empty <= 3'b100; //4 bytes empty
                                    end
                                    16'd5: begin
                                        avalon_st_tx_data <= {sendvalue2[39:0],24'd0};
                                        avalon_st_tx_empty <= 3'b011; //3 bytes empty
                                    end
                                    16'd6: begin
                                        avalon_st_tx_data <= {sendvalue2[47:0],16'd0};
                                        avalon_st_tx_empty <= 3'b010; //2 bytes empty
                                    end
                                    16'd7: begin
                                        avalon_st_tx_data <= {sendvalue2[55:0],8'd0};
                                        avalon_st_tx_empty <= 3'b001; //1 bytes empty
                                    end
                                    16'd8: begin
                                        avalon_st_tx_data <= sendvalue2[63:0];
                                    end
                                endcase
                            end
                           end
```

```
                            4'd11: begin //wrap up
                                    avalon_st_tx_error <= 1'd0;
                                    avalon_st_tx_valid <= 1'd0;
                                    avalon_st_tx_startofpacket <= 1'd0;
                                    avalon_st_tx_endofpacket <= 1'd0;
                                    sendDataEnd <= 1'd1; //tell it is end to send data
                                   end
                        endcase
                    end
                end
        endcase
    end
  end
endmodule
```

- Testbench for Memcached Get Command

```
/**********************
Name: Testbench for Memcached Get Command
Author: FPGA Memcached UDP Team
Date: 2/28/2014
Description:
This testbench is for addition only
**********************/

`timescale 1ns / 100ps
`include "./handler.sv"
module handler_test ();

// inputs
reg clk;
reg reset;
reg avalon_st_rx_valid;
reg avalon_st_rx_startofpacket;
reg avalon_st_rx_endofpacket;
reg [63:0] avalon_st_rx_data;

// output
wire ready;
//wire [319:0] getkey;
//wire [63:0] avalon_st_tx_data;

//using named instantiation
handler u0(
    .clk(clk),
        .reset(reset),
    .avalon_st_rx_valid(avalon_st_rx_valid),
    .avalon_st_rx_startofpacket(avalon_st_rx_startofpacket),
    .avalon_st_rx_endofpacket(avalon_st_rx_endofpacket),
    .avalon_st_rx_data(avalon_st_rx_data),
    .avalon_st_rx_ready(avalon_st_rx_ready),
    .avalon_st_tx_data(avalon_st_tx_data)
);

always
#5 clk = ~clk; // create a 50Mhz clock

initial
begin


    $display($time,"<<starting the simulation>>");
```

```
    clk = 1'b0;
    reset=1'b0;
    avalon_st_rx_valid = 1'b0;       //reset is low
    avalon_st_rx_startofpacket = 1'b0;       // do nothing
    avalon_st_rx_data = 64'b0;
@ (negedge clk);
    reset = 1'b1;
@ (negedge clk);
    reset = 1'b0;
@ (negedge clk);     avalon_st_rx_valid = 1'b1;          //1
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h9999999999999999};

@ (negedge clk); avalon_st_rx_valid = 1'b1;              //2
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h8888888808004588};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;              //3
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h7777777777777711};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;              //4
    avalon_st_rx_startofpacket = 16'b1;
    avalon_st_rx_data = {64'h6666666666666666};
    avalon_st_rx_endofpacket = 1'b0;


@ (negedge clk); avalon_st_rx_valid = 1'b1;              //5
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h5555555555555555};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;              //6
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h4444444444444444};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;              //7
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h0000800000060000};
    avalon_st_rx_endofpacket = 1'b0;
```

```
@ (negedge clk); avalon_st_rx_valid = 1'b1;             //8
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h0000000000060000};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //9
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h1111111111111111};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //10 key: 6765646b6566
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h00006765646b6566};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //(for testing purpose)
    avalon_st_rx_startofpacket = 1'b1;                  // 11 will be truncated
    avalon_st_rx_data = {64'habcd123456789011};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //(for testing purpose)
    avalon_st_rx_startofpacket = 1'b1;                  // 12 will be truncated
    avalon_st_rx_data = {64'hca11223344556677};
    avalon_st_rx_endofpacket = 1'b1;

#200
    $display($time,"<<simulation complete>>");
    $finish;


    $monitor ($time,"clk=%b,reset=%b,avalon_st_rx_valid=%b,avalon_st_rx_data=%b
        ,avalon_st_rx_startofpacket=%b",clk,reset,avalon_st_rx_valid,avalon_st_rx_data,avalon_st_rx_startofpacket);

end

endmodule
```

- Testbench for Memcached Add Command

```
/**********************
Name: Testbench for Memcached Add Command
Author: Memcached UDP team
Date: 2/28/2014
Description:
This testbench is for addition only
**********************/

`timescale 1ns / 100ps
`include "./handler.sv"
module handler_test ();

// inputs
reg clk;
reg reset;
reg avalon_st_rx_valid;
reg avalon_st_rx_startofpacket;
reg avalon_st_rx_endofpacket;
reg [63:0] avalon_st_rx_data;

// output
wire ready;

//using named instantiation
handler u0(
        .clk(clk),
        .reset(reset,
        .avalon_st_rx_valid(avalon_st_rx_valid),
        .avalon_st_rx_startofpacket(avalon_st_rx_startofpacket),
        .avalon_st_rx_endofpacket(avalon_st_rx_endofpacket),
        .avalon_st_rx_data(avalon_st_rx_data),
        .avalon_st_rx_ready(avalon_st_rx_ready),
        .avalon_st_tx_data(avalon_st_tx_data)
);

always
#5 clk = ~clk;                              // create a 50Mhz clock

initial
begin
    $display($time,"<<starting the simulation>>");
    clk = 1'b0;                             // at time 0
    reset=1'b0;
    avalon_st_rx_valid = 1'b0;              //reset is low
    avalon_st_rx_startofpacket = 1'b0;      // do nothing
```

```
    avalon_st_rx_data = 64'b0;
@ (negedge clk);
    reset = 1'b1;
@ (negedge clk);
    reset = 1'b0;
@ (negedge clk);     avalon_st_rx_valid = 1'b1;          //1
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h9999999999999999};

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //2
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h8888888808004588};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //3
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h7777777777777711};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //4
    avalon_st_rx_startofpacket = 16'b1;
    avalon_st_rx_data = {64'h6666666666666666};
    avalon_st_rx_endofpacket = 1'b0;


@ (negedge clk); avalon_st_rx_valid = 1'b1;             //5
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h5555555555555555};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //6
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h4444444444444444};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //7
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h0000800200070000};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //8
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h0000000000110000};
    avalon_st_rx_endofpacket = 1'b0;
```

```
@ (negedge clk); avalon_st_rx_valid = 1'b1;             //9
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h1111111111111111};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //10
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h1111111111111111};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //11 key: 6765646b6566
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'h00006765646b6566};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //12 key(continue): ab
    avalon_st_rx_startofpacket = 1'b1;                  //    value: cd123456789011
    avalon_st_rx_data = {64'habcd123456789011};
    avalon_st_rx_endofpacket = 1'b0;

@ (negedge clk); avalon_st_rx_valid = 1'b1;             //13 value(continue): ca1122
    avalon_st_rx_startofpacket = 1'b1;
    avalon_st_rx_data = {64'hca11223344556677};
    avalon_st_rx_endofpacket = 1'b1;

#200
    $display($time,"<<simulation complete>>");
    $finish;

    $monitor ($time,"clk=%b,reset=%b,avalon_st_rx_valid=%b,avalon_st_rx_data=%b,avalon_st_rx_startofpacket=%b"
        ,clk,reset,avalon_st_rx_valid,avalon_st_rx_data,avalon_st_rx_startofpacket);
end

endmodule
```

7 Reference

- Memcached Binary Protocol
https://code.google.com/p/memcached/wiki/MemcacheBinaryProtocol

- Memcached Binary Protocol
http://memcachedb.googlecode.com/svn/trunk/doc/protocol.txt