Project 1 - Replay Buffer

CPE 186- Computer Hardware Design

MWF 8:00-8:50 AM

Vinh Nguyen, Kevin Mai, Vaukee Lee,

Andrew Lee, Petru Malac

Person	Design Contribution	Code Contribution	Validation Contribution	Report Contribution
Vinh Nguyen	N/A	CRC	CRC	Yes
Kevin Mai	Top, Replay Buffer	Top, Replay Buffer,CRC	Top, Replay Buffer	Yes
Vaukee Lee	CRC	CRC	CRC	Yes
Andrew Lee	Replay Buffer	Replay Buffer	Replay Buffer	Yes
Petru Malac	N/A	CRC	CRC	No

Table of Contents

Introduction

In packet based architectures, information is sent from one device to another through packets which contain the information. In most modern computers, the packet based PCIe standard is used to communicate between different devices in our computer, to connect devices like graphics cards, hard drives and wifi cards. Transmission between data isn't perfect however and thus we need to implement logic to ensure data makes it to the destination, and one way we are doing this is a Replay Buffer.

A Replay Buffer stores data as the sending device sends out packets. This information is received and checked for errors at the receiving end of the architecture. If the receiving device detects an error in the packet it sends a non-acknowledgement (NAK) indicating that the transmission has problems, then the Replay Buffer will resend the failed packets to the sender. However, if no error is detected the system will send an acknowledgement (ACK) that includes the packet number for the packet that was last received properly. Following this the replay buffer will purge all packets with a sequence number less than or equal to the ACK sequence value.

Engineering Data

Design Purpose

The purpose of this project is to learn about the Transmission Layer sending, and we had to create 3 components that are involved in that - we had to manually create TLP Packets, we had to create a verilog design of the LCRC, then build the Replay Buffer that takes data from the 8kB replay buffer used inside the PCIe transaction layer.

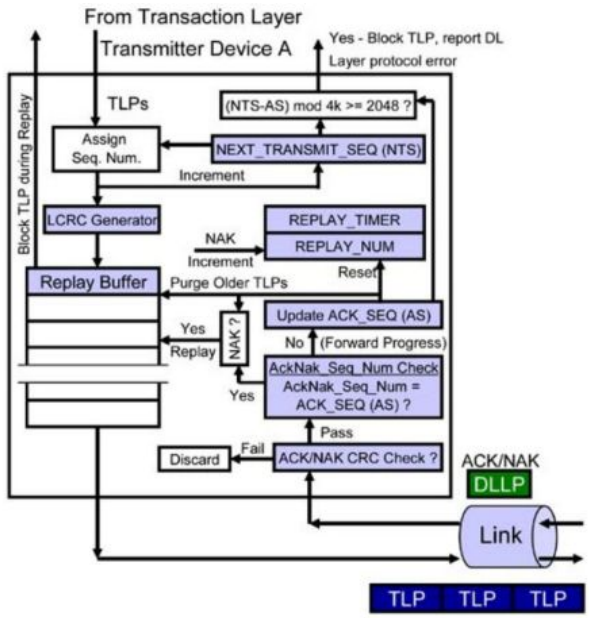
For the TLP Packets, Transaction Layer Packet, each packet is composed of a 3DW header that contains information regarding the purpose of the packet, memory read or write or IO read or write. They also include information regarding the target of the operation as well as the amount of data being sent or expected to be received. They are used to clarify what data is moving across the link, and allow the receiver to work properly with the packets. In this project, we create TLP packets that perform a memory write, a memory read, along with a IO write/read. The data from the headers will be received by LCRC which is appended to the end of the data string. The LCRC is used to detect any errors in the receiving of the raw data from TLP Packet.

For the LCRC verilog unit, we are designing an implementation for cyclic redundancy check (CRC) which is error-detecting code commonly used in digital

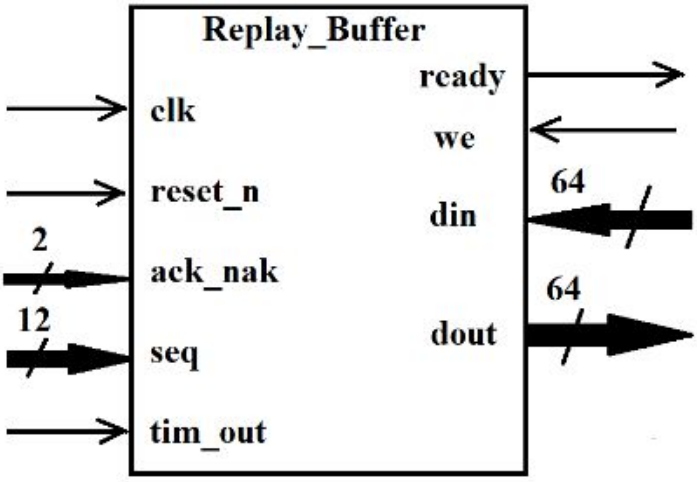
networks and storage devices to detect accidental changes to raw data. We use the CRC algorithm to create the CRC values then concatenate that along with the TLP to create the input that is transmitted that the Replay buffer handles.

For the Replay Buffer, we need to store the concatenated CRC Packets from the LCRC and store them in a FIFO stack. Once information is stored, the buffer needs to start sending out information and depending on how the information is received the replay buffer needs to respond to ACK & NAK signals from the receiver.

Schematics and Diagrams



The Transaction Layer block on the left shows how the sender device looks in regards to the transmission layer. At the input we take in the TLP data packet. First we assign the sequence number and append the bits to that at the front. Then we generate the LCRC and send the information to the replay buffer.



On the right is the Data Block for the replay buffer, showing us all the signals we need to have in the Replay Buffer. The clock and reset is because of the fact it has memory and is a sequential design, the ACK/NAK is a 2 bit control input from the sender that tells the replay buffer if an ACK or NAK happened, sequence number is a 12 bit control input from the sender that tells you where the ACK and NAK is corresponding to, the ready tells if the receiver can send, the write enable allows the TLP to take in inputs,din and doubt are the inputs and outputs of the replay buffer.

TLP

```
0 10 0 0000 0 000 0000 0 0 00 00 00 0000 0001 = h'40000001 memory write 3DW
0 11 0 0000 0 000 0000 0 0 00 00 00 0000 0001 = h'60000001 memory write 4DW

0 00 0 0000 0 000 0000 0 0 00 00 00 0000 0001 = h'00000001 memory read
request

0 10 0 0010 0 000 0000 0 0 00 00 00 0000 0001 = h'42000001 IO write 3DW

DW2 => h'0000000f

DW3 => data address => h'00000032 (d'50)

DW0-1024 => data

Packet 1:  h'40000001, h'0000000f, h'00000014 //memory write 3DW 1 DW
payload

Packet 2: h'00000001, h'0000000f, h'00000024  //memory read no data payload

Packet 3: h'42000001, h'0000000f, h'0000003c  //IO Write 1 DW payload

Packet 4: h'4000001, h'0000000f, h'00000064  //memory write 3DW 1 DW
payload

Packet 5: h'40000001, h'0000000f, h'0000008c //memory write 3DW 1 DW
payload

//adding Sequence numbers and reserve space


Packet 1: h'0, h'000, h'40000000, h'0000000f, h'00000014

Packet 2: h'0, h'001, h'40000000, h'0000000f, h'00000024

Packet 3: h'0, h'002, h'42000000, h'0000000f, h'0000003c
```

```
Packet 4: h'0, h'003, h'40000000, h'0000000f, h'00000064

Packet 5: h'0, h'004, h'40000000, h'0000000f, h'0000008c

//randomly generated data 16 bit data

h'1010

h'1021

h'1110

h'0010

h'1101

appended to packets

Packet 1: h'000, h'40000001, h'0000000f, h'00000014, h'1010

Packet 2: h'001, h'00000001, h'0000000f, h'00000024

Packet 3: h'002, h'42000001, h'0000000f, h'0000003c, h'1110

Packet 4: h'003, h'40000001, h'0000000f, h'00000064, h'0010

Packet 5: h'004, h'40000001, h'0000000f, h'0000008c, h'1101
```

CRC

Sequence

```
module seq(input [95:0] tlp_in,
      output [127:0] data_o);

reg [11:0] seq_num = 12'b0;
reg [15:0] seq_out;

always@(*)begin
```

```
      seq_num = seq_num + 1;
      seq_out = {4'b0,seq_num};
end

assign data_o = {seq_out,tlp_in,16'b0};

endmodule
```

Verilog Model

```
module crc #(parameter NBITS=16)
      (input clk,
            input rst,
            input we,
            input [95:0] data,
            output reg [NBITS-1:0] q,
            output reg [127:0] dataOut,
            output reg rdy);

reg [NBITS:1] d;
reg xorin;
integer count = 0;
reg [11:0] seq;


always@(posedge clk, negedge rst)
begin
    if(!rst)
    begin
       d <= 16'b1;
       seq <= 0;
    end
    else begin
       if(we)begin
             d <= data;
             if(rdy)begin
             rdy <= 0;
             end
       end
       else begin
```

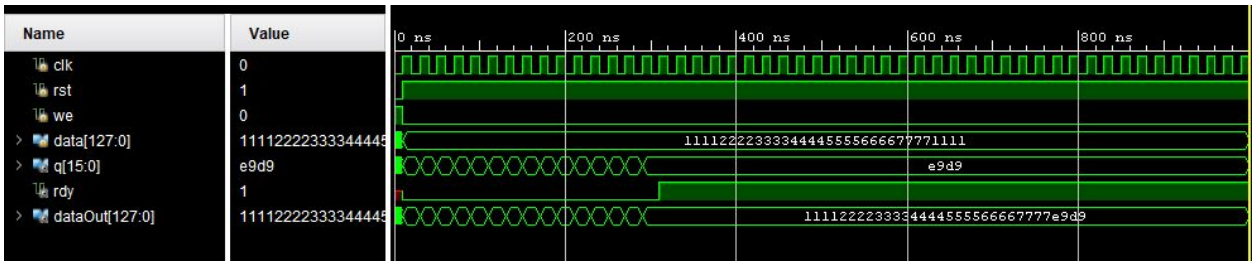
```
            if(count < 15)begin
            d <= {d[NBITS-1:1], xorin};
            rdy <= 0;
            count <= count + 1;
            end
            else begin
            rdy <= 1;
            seq <= seq + 1;
            end
       end
    end
end

always@(*)begin
    xorin = (d[12] ^ d[3] ^ d[1]);
    if(rdy)begin
    q = d[NBITS:1];
    dataOut[15:0] = q;
    dataOut[95:16] = data[95:16];
    dataOut[127:96] = seq;
    end
end

endmodule
```

Model Validation

```
module crc_tb;

    //parameter NUMB = 16;

    reg clk;
    reg rst;
    reg we;
    reg [95:0] data;
    wire [15:0] q;
    wire rdy;
      wire [127:0] dataOut;

crc #(16)
rr1(.clk(clk),.rst(rst),.we(we),.data(data),.q(q),.dataOut(dataOut),.rdy(rd
y));
```

```
initial
      $monitor($time, " we = %b rst = %b finish = %b\n\t\t\t data =
%h\n\t\t\t d = %b\n\t\t\t dataOut = %h\n", rst, we, rdy, data, crc.d,
dataOut);


initial
begin
      rst = 1'b0; we = 1;
#10    rst = 1'b1; we = 0;
    data = 96'h111122223333444455556666;
/*
#340    rst = 1'b0;
#10    rst = 1'b1; we = 1;
#10    we = 0;
    data = 96'h11112222000044445555AAAA;

#200    data = 128'h1111222233334444ffff666677770000;
#200    data = 128'h1111eeee33334444555566667770000;
*/

#300 $finish;
end


initial
begin
      clk = 1'b0;
      forever #10 clk = ~clk;
end

endmodule
```

Model Validation Results



```
270 we = 1 rst = 0 finish = 0
        data = 111122223333444455556666
        d = 0111010011101100
        dataOut = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

290 we = 1 rst = 0 finish = 0
        data = 111122223333444455556666
        d = 1110100111011001
        dataOut = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

310 we = 1 rst = 0 finish = 1
        data = 111122223333444455556666
        d = 1110100111011001
        dataOut = 00000001111122223333444455555e9d9
```

Replay Buffer

Verilog Model

```
module replayBuffer(clk, reset, we, oe, tim_out, full, empty, w_addr,
r_addr, din, dout, ack, nak);

input  [128:0] din;
input  clk, reset;
input  we, oe;
input  ack, nak;
input  tim_out;
output full, empty;
output [2:0] w_addr, r_addr;
output [128:0] dout;

integer i;

reg [2:0]  w_addr, r_addr;
reg [128:0] ram_reg;
reg [128:0] memory [512:0];

always@(posedge clk)
begin
   if(!ack && !nak)
   begin
       if(reset)
       begin
           w_addr <=0;
```

```
          r_addr <=0;
      end
      else if(tim_out)
      begin
       for(i=0; i<=w_addr; i=i+1)
       begin
              ram_reg <= memory[i];
       end
      end
      else
        if( we && !full)
           if( w_addr >= 4)
               w_addr <= 0;
           else
               w_addr <= w_addr + 1;

        if( oe && !empty)
           if( r_addr >= 4)
               r_addr <= 0;
           else
               r_addr <= r_addr + 1;

     if (we)
         memory[w_addr] <= din;

     ram_reg <= memory[r_addr];
   end
   if(ack && !nak)
    begin
       for(i=0; i<=r_addr; i=i+1)
       begin
              memory[i] <= memory[r_addr+i];
              memory[r_addr+i] <= 0;
              w_addr=0;
              r_addr=0;
       end
    end
    if(!ack && nak)
    begin
       for(i=0; i<=r_addr; i=i+1)
       begin
              ram_reg <= memory[i];
```

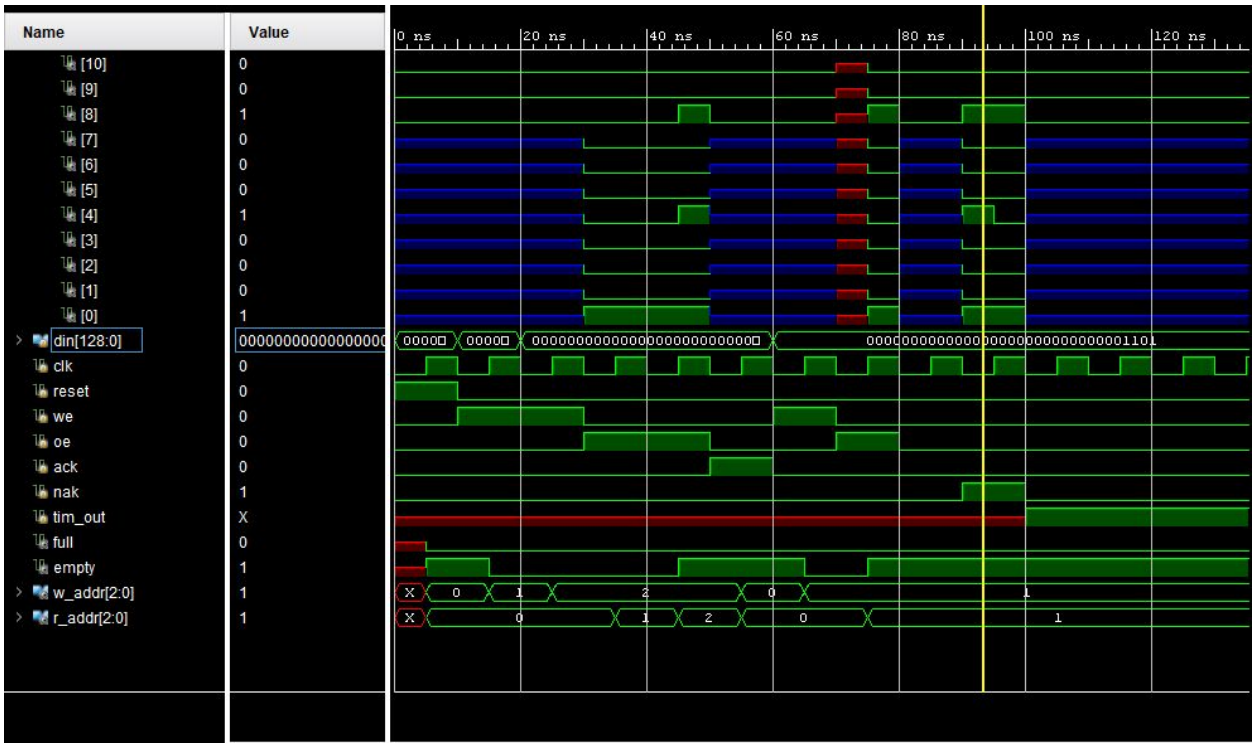
```
                #10;
        end
     end
end

assign dout = ((!we && oe) || nak) ? ram_reg:8'hzz;

assign full = ((r_addr[2]!=w_addr[2]) &&
(r_addr[1:0]==w_addr[1:0]))?1'b1:1'b0;

assign empty= (r_addr == w_addr) ? 1'b1 : 1'b0;
endmodule
```

Model Validation

```
`timescale 1ns/100ps

module replayBuffer_tb;

    reg  [128:0] din;
    reg  clk, reset;
    reg  we, oe, ack, nak;
    reg tim_out;
    wire full, empty;
    wire [2:0] w_addr, r_addr;
    wire [128:0] dout;

    replayBuffer uut(clk, reset, we, oe, tim_out, full, empty, w_addr,
r_addr, din, dout, ack, nak);

    initial begin
    din=16'h0000;
    clk=0;
    reset=0;
    we=0;
    oe=0;
    ack=0;
    nak=0;
    end

    always
    #5 clk=~clk;
```

```
    initial begin
        reset = 1;
        #10;
        reset = 0;
        we=1;oe=0;
        din=16'h0001;
        #10;
        we=1;oe=0;
        din=16'h1111;
        #10;
        we=0;oe=1;
        #20
        we=0;oe=0;
        ack = 1;
        #10
        we=1;oe=0;
        ack=0;
        din=16'h1101;
        #10
        we=0;oe=1;
        #10
        we=0;oe=0;
        #10
        nak=1;
        #10
        nak=0;
        tim_out = 1;
        #10;
    end

endmodule
```

Model Validation Results



Top

Verilog Model

```
module top(input clk,rst, input [127:0] TLP_in);
//For CRC
wire  we;
wire  rdy;
wire [15:0] lfsr_o;


//for Replay Buffer
wire [127:0] din;
wire oe, full, empty;
wire [2:0] w_addr, r_addr;
wire [127:0] dout;
```

```
wire  ack, nak;

//generate CRC, output data
crc #(16)
crc1(.clk(clk),.rst(rst),.we(we),.data(data),.q(lfsr_o),.dataOut(din),.rdy(
rdy));

//replaybuffer
replayBuffer rp1(.clk(clk), .reset(rst), .we(we), .oe(oe), .full(full),
.empty(empty), .w_addr(w_addr), .r_addr(r_addr), .din(din), .dout(dout),
.ack(ack), .nak(nak));//fifo + replay buffer logic

endmodule
```

Results Discussion

Overall, our program with each component run well with their own validation. We have a total of 4 components in this design. There are many obstacles during the project but we make it work as it should according to the requirement. The CRC work with the valid testbench and result. TLP accept the data in to validate. The Reply Buffer validate the data transfer from the CRC to it as it should. We didn't finish top but everything else works as expected.

Conclusion

Overall this project helped us learn more about how PCIe handles data loss through implementing the logic in Verilog. In this Project we built a Replay Buffer which stores data as the sending device sends out packets depending on the information given back by the receiver. This information is received and checked for errors at the receiving end of the architecture. This lab was a big challenge, mainly because of the large scope of the project and because the instructions were rather vague. After a lot of research and work though, we got the Replay Buffer and CRC to work.