

EE 533
Lab 3: Mini Network Intrusion
Detection System (NIDS)
Professor Young Cho
Melvin Houston
1/31/26

Objective: Design and simulate a mini Network Intrusion Detection System (Mini-NIDS) via schematics and Verilog coding, to successfully filter packets containing attack patterns.

GitHub code: <https://github.com/mlhoustoniii/EE533.git>

Hierarchy of the system:

Ids_sim - top file

- **Input_fifo** - Fifo with fall through to receive the packets
- **Matcher** - Detects if input data matches with the 7-Byte stored pattern
 - Busmerge
 - Wordmatch
 - reg9B
- **Drop_fifo** - Removes a 7-Byte packet from memory, if the stored sequence is detected
 - reg9B
 - Comp8
- **module_regs** - Contains the logic for hardware- and software-specific register signals
 - Generic_entr_regs
 - Generic_sw_regs
 - Generic_hw_regs

Any modules that were user-developed were done via schematic entry. The schematics were then converted to Verilog files before behavioral simulations were performed. The resulting verilog code is structural and is difficult to infer its behavior in an efficient manner. For a lab such as this, I believe schematic entry is simpler, and allows one to visually interpret the design's intent. However, for a larger and more complex design, I would prefer to write mostly verilog, and ideally have modules which are easily reusable; with very large schematics, debugging proves more difficult than verilog coding.

In the below figure is the output waveform of the Mini-IDS. The takeaway of this system's functionality is that data received from outside of the network is parsed through and searching for a match between the input and a known pattern, which is stored. If the pattern is detected within the packets of incoming data, the specific, those specific bits will be filtered out and prevented from reaching the other side of the IDS.

There are several key modules that enable the functionality of this NIDS. The *busmerge* module takes the original input data of 72 bits, and concatenates it with a copy of 56 bits (7 Bytes) from the previous cycle, combined with 56 bits from the current cycle. This provides a 112-bit window of data in a single cycle, to ensure there is an opportunity to find the pattern and match the sequence, if present. Another key module is *comparator*. Besides the two inputs which are to be compared, the *amask* input allows for the explicit inclusion or exclusion of one of the seven

comparison units, within the comparator macro. For example, if the upper 8-bits in a sequence are not cared for in a detection window, they may be excluded by setting *amask[6]* to a value of zero. Within the *comparator* macro, the *comp8* module's function is to output a logic high if two 8-bit input vectors are of equal value, providing the base of the pattern detection functionality. Additionally, the *comp8* module is used within the *dropfifo* module, and is responsible for the comparison between the read address and the write address fed into the fifo. The last important block, *reg9B*, contained within *dropfifo*, served as an intermediate memory block for timing synchronization, as the rest of the system's modules required time to determine if the data held inside *reg9B* was to be filtered out (dropped).

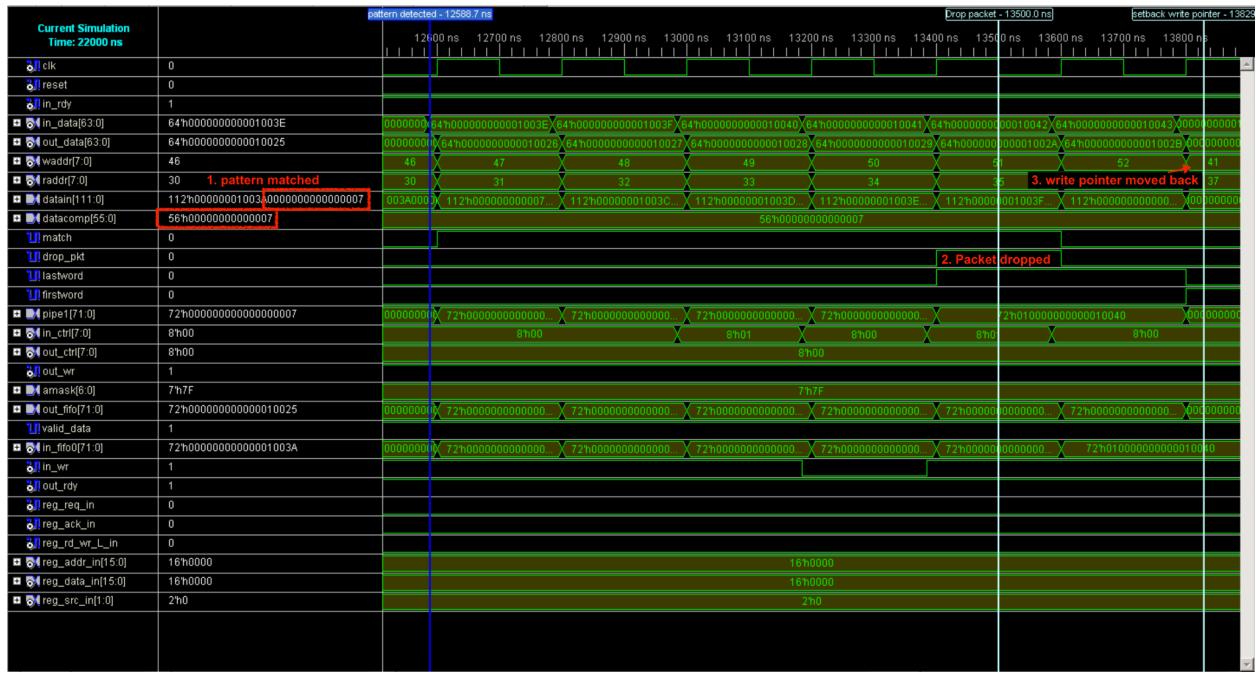


Figure 0: Mini-IDS testbench results

In the figure above, the testbench has found a match of the pattern, contained within the *datacomp* vector, and the *match* signal is latched in the subsequent clock tick. Once the end of the packet is reached, the latched *match* signal is read and causes the entire packet to be dropped. The *drop_pkt* signal is fed and causes the final fifo to overwrite the input data containing the unwanted packet by setting the write pointer back and is now ready to accept new data at that address.

Captured schematics

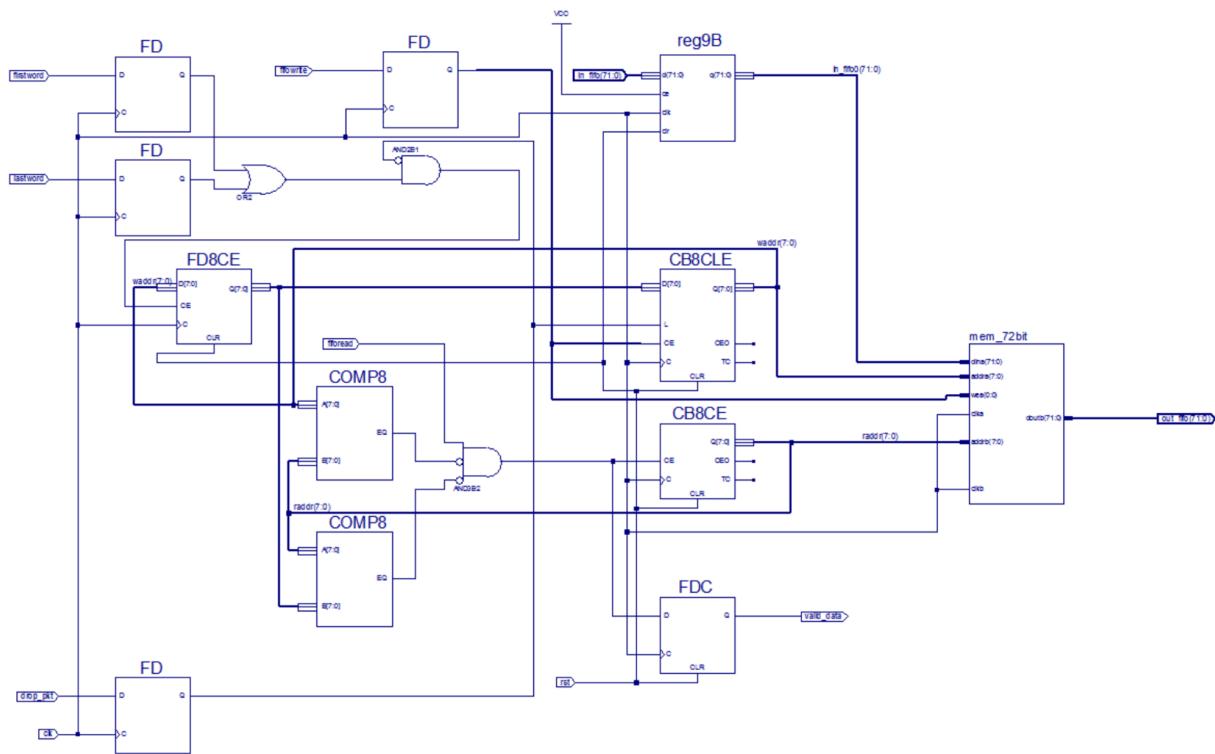


Figure 1: dropfifo schematic

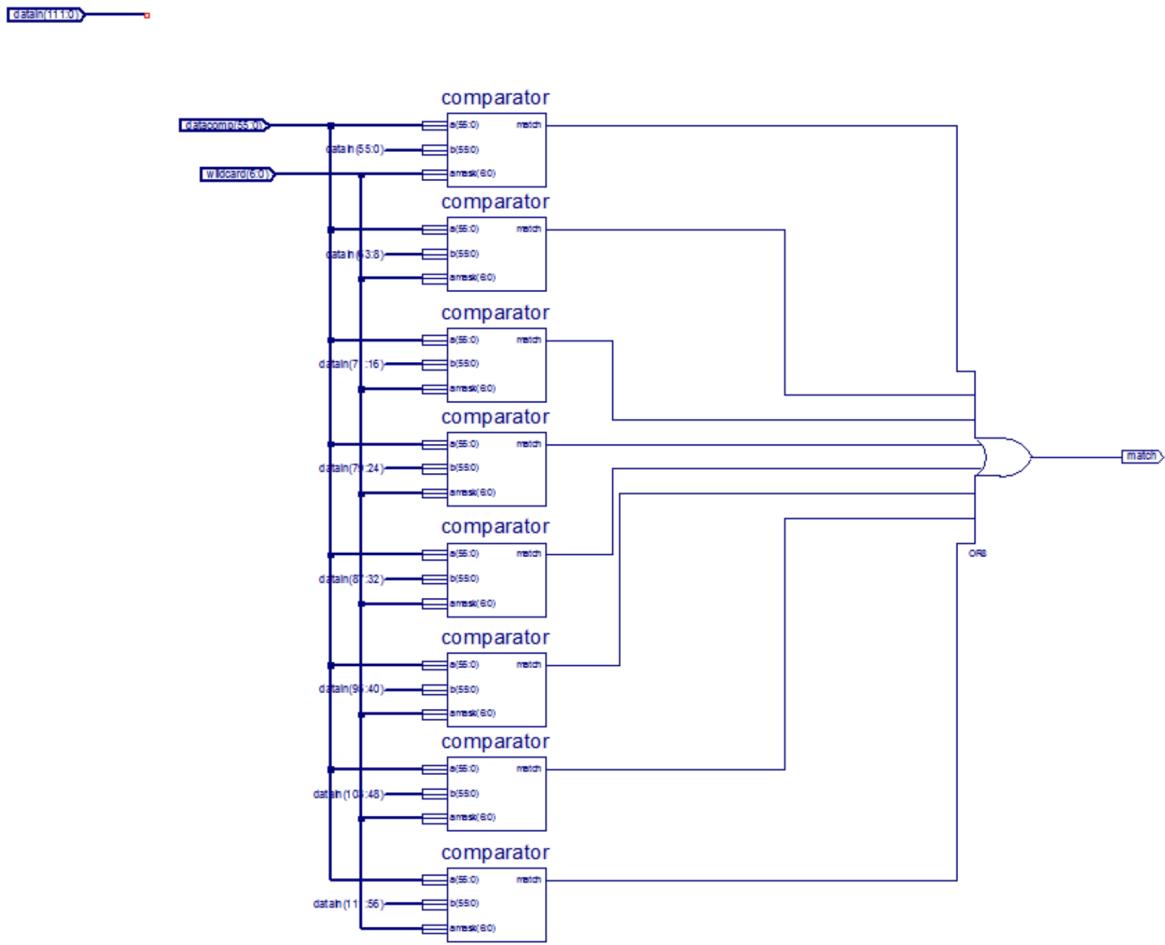


Figure 2: wordmatch schematic

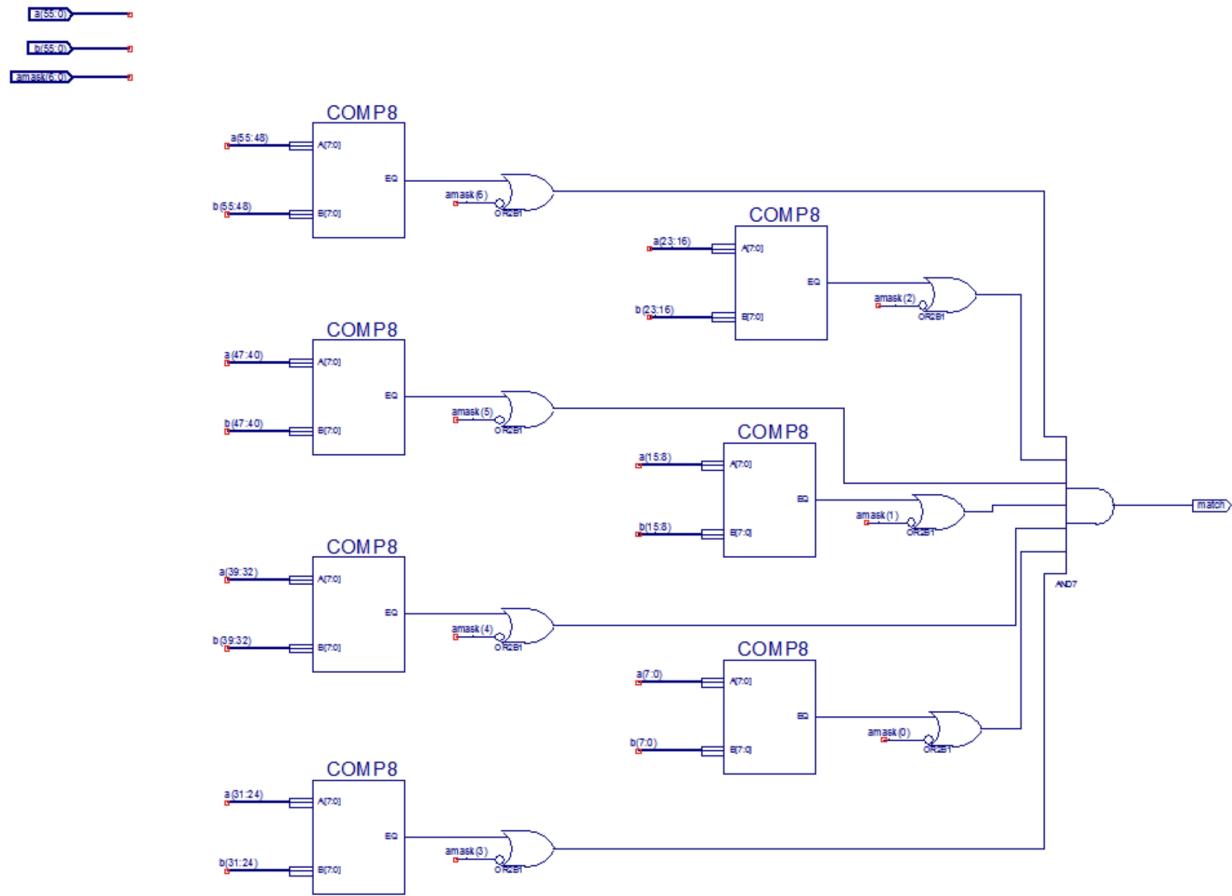


Figure 3: comparator schematic

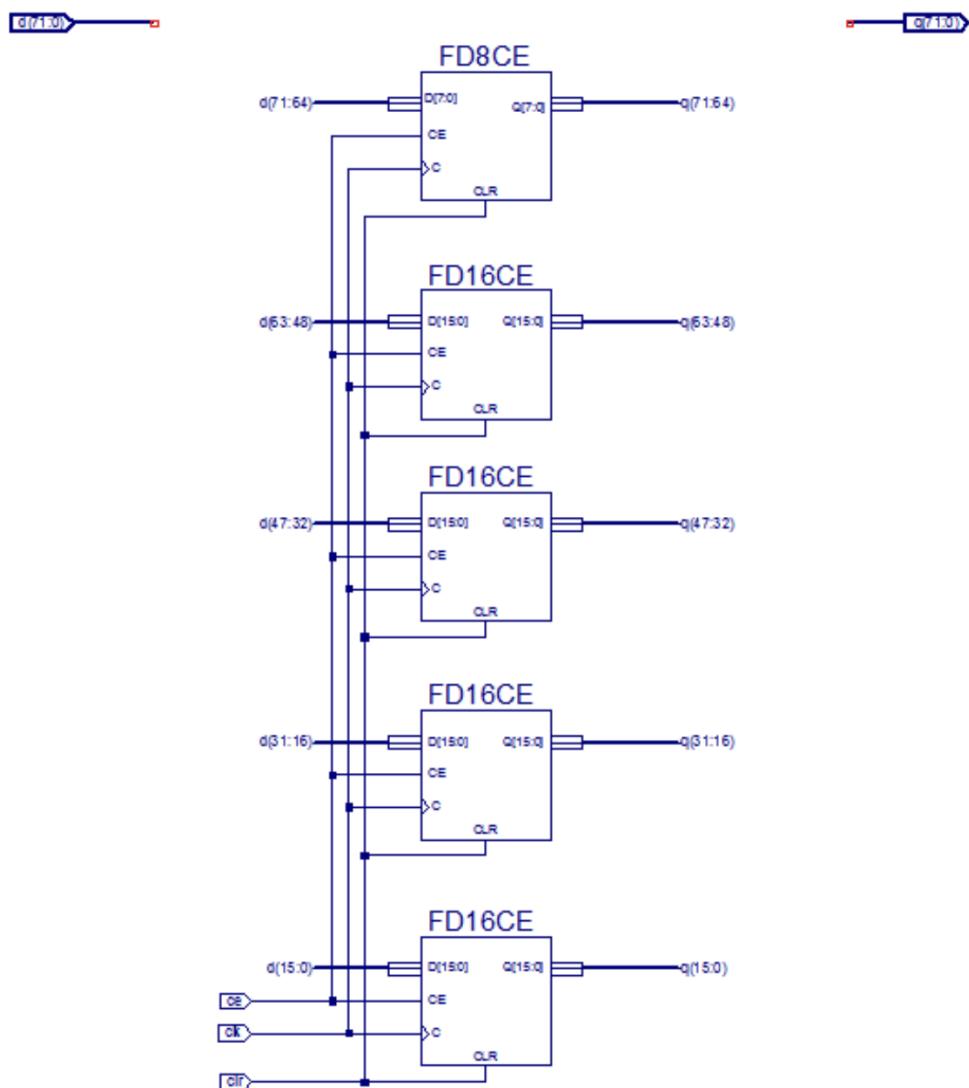


Figure 4: Reg9B schematic

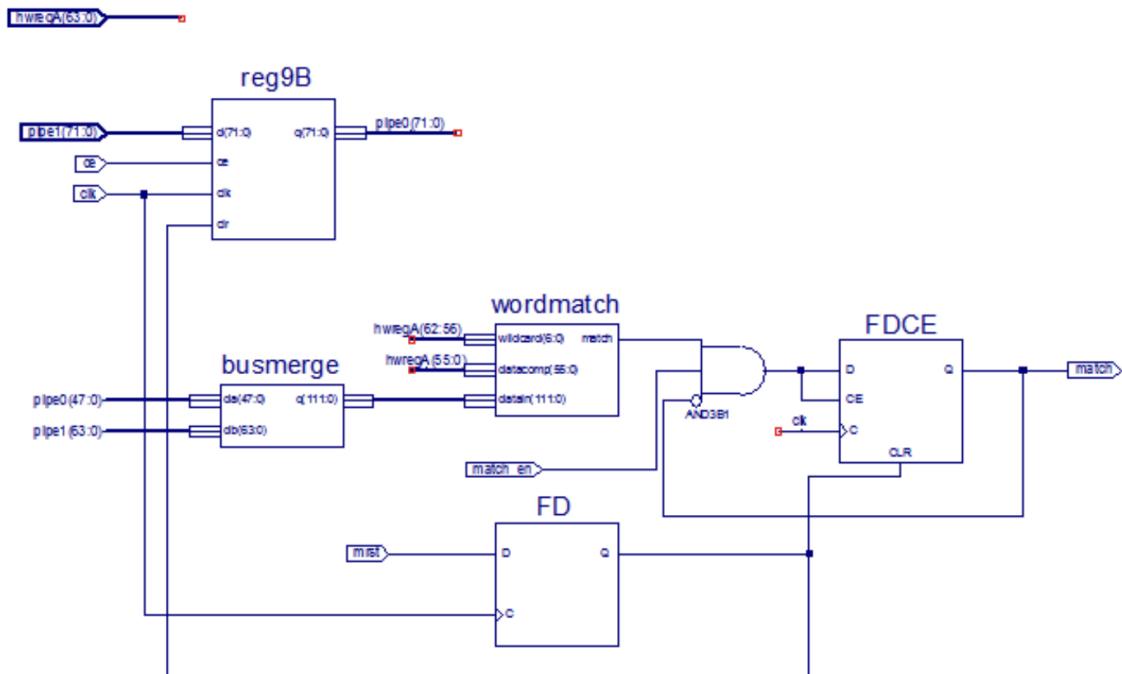


Figure 5: Detect7B schematic

Verilog converted froms schematics

Dropfifo

```
module FD8CE_HXILINX_dropfifo(Q, C, CE, CLR, D);
```

```
    output [7:0] Q;
```

```
    input C;
```

```
    input CE;
```

```
    input CLR;
```

```
    input [7:0] D;
```

```
    reg [7:0] Q;
```

```
    always @ (posedge C or posedge CLR)
```

```
        begin
```

```
            if (CLR)
```

```
                Q <= 8'b0000_0000;
```

```
            else if (CE)
```

```
                Q <= D;
```

```
        end
```

```
endmodule
```

```
`timescale 100 ps / 10 ps
```

```
module COMP8_HXILINX_dropfifo (EQ, A, B);
```

```
    output EQ;
```

```
    input [7:0] A;
```

```
    input [7:0] B;
```

```
    assign EQ = (A==B) ;
```

```
endmodule
```

```
`timescale 100 ps / 10 ps
```

```

module CB8CLE_HXILINX_dropfifo(CEO, Q, TC, C, CE, CLR, D, L);

parameter TERMINAL_COUNT = 8'b1111_1111;

output      CEO;
output [7:0]  Q;
output      TC;

input       C;
input       CE;
input       CLR;
input [7:0]  D;
input       L;

reg  [7:0]  Q;

always @ (posedge C or posedge CLR)
begin
    if (CLR)
        Q <= 8'b0000_0000;
    else if (L)
        Q <= D;
    else if (CE)
        Q <= Q + 1;
end

assign CEO = TC & CE;
assign TC = Q == TERMINAL_COUNT;

endmodule
`timescale 100 ps / 10 ps

```

```

module CB8CE_HXILINX_dropfifo(CEO, Q, TC, C, CE, CLR);

parameter TERMINAL_COUNT = 8'b1111_1111;

output      CEO;
output [7:0]  Q;
output      TC;

```

```

input      C;
input      CE;
input      CLR;

reg [7:0]   Q;

always @(posedge C or posedge CLR)
begin
    if(CLR)
        Q <= 8'b0000_0000;
    else if(CE)
        Q <= Q + 1;
end

assign CEO = TC & CE;
assign TC = Q == TERMINAL_COUNT;

endmodule
`timescale 1ns / 1ps

module dropfifo(clk,
                 drop_pkt,
                 fiforead,
                 fifowrite,
                 firstword,
                 in_fifo,
                 lastword,
                 rst,
                 out_fifo,
                 valid_data);

    input clk;
    input drop_pkt;
    input fiforead;
    input fifowrite;
    input firstword;
    input [71:0] in_fifo;
    input lastword;
    input rst;
    output [71:0] out_fifo;

```

```

output valid_data;

wire [71:0] in_fifo0;
wire [7:0] raddr;
wire [7:0] waddr;
wire XLXN_36;
wire XLXN_37;
wire XLXN_38;
wire [7:0] XLXN_46;
wire XLXN_55;
wire XLXN_56;
wire XLXN_58;
wire XLXN_62;
wire XLXN_70;
wire XLXN_103;
wire [0:0] XLXN_105;

FD XLXI_3 (.C(clk),
    .D(firstword),
    .Q(XLXN_36));
defparam XLXI_3.INIT = 1'b0;
FD XLXI_4 (.C(clk),
    .D(lastword),
    .Q(XLXN_37));
defparam XLXI_4.INIT = 1'b0;
FD XLXI_5 (.C(clk),
    .D(drop_pkt),
    .Q(XLXN_70));
defparam XLXI_5.INIT = 1'b0;
FD XLXI_6 (.C(clk),
    .D(fifowrite),
    .Q(XLXN_105[0]));
defparam XLXI_6.INIT = 1'b0;
CB8CE_HXILINX_dropfifo XLXI_8 (.C(clk),
    .CE(XLXN_62),
    .CLR(rst),
    .CEO(),
    .Q(raddr[7:0]),
    .TC());
// synthesis attribute HU_SET of XLXI_8 is "XLXI_8_2"

```

```

CB8CLE_HXILINX_dropfifo XLXI_9 (.C(clk),
    .CE(XLXN_105[0]),
    .CLR(rst),
    .D(XLXN_46[7:0]),
    .L(XLXN_70),
    .CEO(),
    .Q(waddr[7:0]),
    .TC());
// synthesis attribute HU_SET of XLXI_9 is "XLXI_9_4"
reg9B XLXI_10 (.ce(XLXN_103),
    .clk(clk),
    .clr(rst),
    .d(in_fifo[71:0]),
    .q(in_fifo0[71:0]));
FD8CE_HXILINX_dropfifo XLXI_22 (.C(clk),
    .CE(XLXN_58),
    .CLR(rst),
    .D(waddr[7:0]),
    .Q(XLXN_46[7:0]));
// synthesis attribute HU_SET of XLXI_22 is "XLXI_22_0"
COMP8_HXILINX_dropfifo XLXI_23 (.A(waddr[7:0]),
    .B(raddr[7:0]),
    .EQ(XLXN_56));
// synthesis attribute HU_SET of XLXI_23 is "XLXI_23_1"
COMP8_HXILINX_dropfifo XLXI_24 (.A(raddr[7:0]),
    .B(XLXN_46[7:0]),
    .EQ(XLXN_55));
// synthesis attribute HU_SET of XLXI_24 is "XLXI_24_3"
OR2 XLXI_33 (.I0(XLXN_37),
    .I1(XLXN_36),
    .O(XLXN_38));
AND2B1 XLXI_34 (.I0(XLXN_70),
    .I1(XLXN_38),
    .O(XLXN_58));
AND3B2 XLXI_41 (.I0(XLXN_55),
    .I1(XLXN_56),
    .I2(foread),
    .O(XLXN_62));
FDC XLXI_42 (.C(clk),
    .CLR(rst),

```

```
.D(XLXN_62),
.Q(valid_data));
defparam XLXI_42.INIT = 1'b0;
memory72bit XLXI_46 (.addra(waddr[7:0]),
.addrb(raddr[7:0]),
.clka(clk),
.clkb(clk),
.dina(in_fifo0[71:0]),
.wea(XLXN_105[0]),
.doutb(out_fifo[71:0]));
VCC XLXI_52 (.P(XLXN_103));
endmodule
```

wordmatch

```
module OR8_HXILINX_wordmatch (O, I0, I1, I2, I3, I4, I5, I6, I7);
```

```
    output O;
```

```
    input I0;
    input I1;
    input I2;
    input I3;
    input I4;
    input I5;
    input I6;
    input I7;
```

```
assign O = (I0 || I1 || I2 || I3 || I4 || I5 || I6 || I7);
```

```
endmodule
```

```
`timescale 1ns / 1ps
```

```
module wordmatch(datacomp,
```

```
    datain,
    wildcard,
    match);
```

```
    input [55:0] datacomp;
    input [111:0] datain;
    input [6:0] wildcard;
    output match;
```

```
    wire XLXN_17;
    wire XLXN_21;
    wire XLXN_33;
    wire XLXN_45;
    wire XLXN_77;
    wire XLXN_80;
    wire XLXN_81;
    wire XLXN_82;
```

```
comparator XLXI_1 (.a(datacomp[55:0]),
    .amask(wildcard[6:0]),
```

```

.b(datain[55:0]),
.match(XLXN_17));
comparator XLXI_2 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[63:8]),
.match(XLXN_21));
comparator XLXI_4 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[71:16]),
.match(XLXN_82));
comparator XLXI_5 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[79:24]),
.match(XLXN_33));
comparator XLXI_6 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[87:32]),
.match(XLXN_81));
comparator XLXI_7 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[95:40]),
.match(XLXN_80));
comparator XLXI_8 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[103:48]),
.match(XLXN_45));
OR8_HXILINX_wordmatch XLXI_16 (.I0(XLXN_77),
.I1(XLXN_45),
.I2(XLXN_80),
.I3(XLXN_81),
.I4(XLXN_33),
.I5(XLXN_82),
.I6(XLXN_21),
.I7(XLXN_17),
.O(match));
// synthesis attribute HU_SET of XLXI_16 is "XLXI_16_0"
comparator XLXI_17 (.a(datacomp[55:0]),
.amask(wildcard[6:0]),
.b(datain[111:56]),
.match(XLXN_77));

```

endmodule

Comparator

```
module COMP8_HXILINX_comparator (EQ, A, B);
```

output EQ;

```
input [7:0] A;  
input [7:0] B;
```

assign EQ = (A==B);

endmodule

`timescale 100 ps / 10 ps

```
module AND7_HXILINX_comparator (O, I0, I1, I2, I3, I4, I5, I6);
```

```
output O;

input I0;
input I1;
input I2;
input I3;
input I4;
input I5;
input I6;

assign O = I0 && I1 && I2 && I3 && I4 && I5 && I6;

endmodule
`timescale 1ns / 1ps

module comparator(a,
                  amask,
                  b,
                  match);

    input [55:0] a;
    input [6:0] amask;
    input [55:0] b;
    output match;

    wire XLXN_16;
    wire XLXN_19;
    wire XLXN_22;
    wire XLXN_25;
    wire XLXN_46;
    wire XLXN_49;
    wire XLXN_65;
    wire XLXN_82;
    wire XLXN_83;
    wire XLXN_84;
    wire XLXN_86;
    wire XLXN_87;
    wire XLXN_88;
    wire XLXN_90;
```

```

COMP8_HXILINX_comparator XLXI_1 (.A(a[55:48]),
    .B(b[55:48]),
    .EQ(XLXN_16));
// synthesis attribute HU_SET of XLXI_1 is "XLXI_1_3"
COMP8_HXILINX_comparator XLXI_3 (.A(a[47:40]),
    .B(b[47:40]),
    .EQ(XLXN_19));
// synthesis attribute HU_SET of XLXI_3 is "XLXI_3_4"
COMP8_HXILINX_comparator XLXI_4 (.A(a[39:32]),
    .B(b[39:32]),
    .EQ(XLXN_22));
// synthesis attribute HU_SET of XLXI_4 is "XLXI_4_2"
COMP8_HXILINX_comparator XLXI_5 (.A(a[31:24]),
    .B(b[31:24]),
    .EQ(XLXN_25));
// synthesis attribute HU_SET of XLXI_5 is "XLXI_5_1"
COMP8_HXILINX_comparator XLXI_11 (.A(a[7:0]),
    .B(b[7:0]),
    .EQ(XLXN_65));
// synthesis attribute HU_SET of XLXI_11 is "XLXI_11_6"
COMP8_HXILINX_comparator XLXI_12 (.A(a[23:16]),
    .B(b[23:16]),
    .EQ(XLXN_46));
// synthesis attribute HU_SET of XLXI_12 is "XLXI_12_0"
COMP8_HXILINX_comparator XLXI_13 (.A(a[15:8]),
    .B(b[15:8]),
    .EQ(XLXN_49));
// synthesis attribute HU_SET of XLXI_13 is "XLXI_13_7"
OR2B1 XLXI_18 (.I0(amask[6]),
    .I1(XLXN_16),
    .O(XLXN_90));
OR2B1 XLXI_19 (.I0(amask[5]),
    .I1(XLXN_19),
    .O(XLXN_87));
OR2B1 XLXI_20 (.I0(amask[4]),
    .I1(XLXN_22),
    .O(XLXN_84));
OR2B1 XLXI_21 (.I0(amask[3]),
    .I1(XLXN_25),

```

```
.O(XLXN_83));
OR2B1 XLXI_22 (.I0(amask[2]),
    .I1(XLXN_46),
    .O(XLXN_88));
OR2B1 XLXI_23 (.I0(amask[1]),
    .I1(XLXN_49),
    .O(XLXN_86));
OR2B1 XLXI_24 (.I0(amask[0]),
    .I1(XLXN_65),
    .O(XLXN_82));
AND7_HXILINX_comparator XLXI_25 (.I0(XLXN_83),
    .I1(XLXN_82),
    .I2(XLXN_84),
    .I3(XLXN_86),
    .I4(XLXN_87),
    .I5(XLXN_88),
    .I6(XLXN_90),
    .O(match));
// synthesis attribute HU_SET of XLXI_25 is "XLXI_25_5"
endmodule
```

reg9B

```
module FD16CE_HXILINX_reg9B(Q, C, CE, CLR, D);
```

```
    output [15:0] Q;
```

```
    input C;
```

```
    input CE;
```

```
    input CLR;
```

```
    input [15:0] D;
```

```
    reg [15:0] Q;
```

```
    always @(posedge C or posedge CLR)
```

```
        begin
```

```
            if(CLR)
```

```
                Q <= 16'b0000_0000_0000_0000;
```

```
            else if(CE)
```

```
                Q <= D;
```

```
        end
```

```
endmodule
```

```
'timescale 100 ps / 10 ps
```

```
module FD8CE_HXILINX_reg9B(Q, C, CE, CLR, D);
```

```
    output [7:0] Q;
```

```
    input C;
```

```
    input CE;
```

```
    input CLR;
```

```
    input [7:0] D;
```

```
    reg [7:0] Q;
```

```
    always @(posedge C or posedge CLR)
```

```
        begin
```

```
            if(CLR)
```

```

        Q <= 8'b0000_0000;
    else if (CE)
        Q <= D;
    end

endmodule
`timescale 1ns / 1ps

module reg9B(ce,
    clk,
    clr,
    d,
    q);

input ce;
input clk;
input clr;
input [71:0] d;
output [71:0] q;

FD8CE_HXILINX_reg9B XLXI_1 (.C(clk),
    .CE(ce),
    .CLR(clr),
    .D(d[71:64]),
    .Q(q[71:64]));
// synthesis attribute HU_SET of XLXI_1 is "XLXI_1_0"
FD16CE_HXILINX_reg9B XLXI_2 (.C(clk),
    .CE(ce),
    .CLR(clr),
    .D(d[63:48]),
    .Q(q[63:48]));
// synthesis attribute HU_SET of XLXI_2 is "XLXI_2_1"
FD16CE_HXILINX_reg9B XLXI_3 (.C(clk),
    .CE(ce),
    .CLR(clr),
    .D(d[47:32]),
    .Q(q[47:32]));
// synthesis attribute HU_SET of XLXI_3 is "XLXI_3_2"

```

```
FD16CE_HXILINX_reg9B XLXI_4 (.C(clk),
    .CE(ce),
    .CLR(clr),
    .D(d[31:16]),
    .Q(q[31:16]));
// synthesis attribute HU_SET of XLXI_4 is "XLXI_4_3"
FD16CE_HXILINX_reg9B XLXI_5 (.C(clk),
    .CE(ce),
    .CLR(clr),
    .D(d[15:0]),
    .Q(q[15:0]));
// synthesis attribute HU_SET of XLXI_5 is "XLXI_5_4"
endmodule
```

Detect7B

```
module detect7B(ce,
    clk,
    hwregA,
    match_en,
    mrst,
    pipe1,
    match);

input ce;
input clk;
input [63:0] hwregA;
input match_en;
input mrst;
input [71:0] pipe1;
output match;

wire clr;
wire [71:0] pipe0;
wire [111:0] XLXN_1;
wire XLXN_10;
wire XLXN_16;
wire match_DUMMY;

assign match = match_DUMMY;
busmerge XLXI_1 (.da(pipe0[47:0]),
    .db(pipe1[63:0]),
    .q(XLXN_1[111:0]));
wordmatch XLXI_5 (.datacomp(hwregA[55:0]),
    .datain(XLXN_1[111:0]),
    .wildcard(hwregA[62:56]),
    .match(XLXN_10));
reg9B XLXI_6 (.ce(ce),
    .clk(clk),
    .clr(clr),
    .d(pipe1[71:0]),
    .q(pipe0[71:0]));
FD XLXI_8 (.C(clk),
    .D(mrst),
    .Q(clr));
```

```
defparam XLXI_8.INIT = 1'b0;
FDCE XLXI_9 (.C(clk),
    .CE(XLXN_16),
    .CLR(clr),
    .D(XLXN_16),
    .Q(match_DUMMY));
defparam XLXI_9.INIT = 1'b0;
AND3B1 XLXI_10 (.I0(match_DUMMY),
    .I1(match_en),
    .I2(XLXN_10),
    .O(XLXN_16));
endmodule
```