Kill Switch: Hardware-Based Line-Rate Filtering and Capture of 10Gb/s Ethernet Network
Qiushi Ding qd2119, Bokai Chen bc2526, Yuyang Wang yw2507, Liheng Wang lw2496, Jingshu Fang jf2781


CATALOG
I. Introduction...............................................................................................................2
II. System structure......................................................................................................3
III. Module....................................................................................................................4
IV. Implementation and simulation .............................................................................8
V. Contribution.............................................................................................................20
VI. Source Code.........................................................................................................21
I. Introduction

Our project idea is to build a hardware (FPGA) based high speed filtering system. It selectively passes traffic from one port of the Solarflare AoE card to the second one, filtering specific packets depending on their "session". For traffic that belongs to a particular 'session', it will be captured and written to disk. It is a high speed hardware-based firewall and packet capture device.

A. Motivation

Nowadays financial activity like stock exchange is mainly fulfilled by computer and internet. If we have a fast enough device that has a processing period of 50ns thus making it transparent to the servers and users, can both filtering and capturing the dealing packets, we may conduct early dealing predictions based on the packets we captured. The device is also a firewall to protect the servers from massive packets attack.

B. Platform

We use solarflare AoE card, which contains a dual port 10Gb/s PCI-Express NIC with onboard Stratix V FPGA. Solarflare’s ApplicationOnload™ Engine (AOE) is an open platform that combines a high performance, ultra-low latency server adapter with a tightly-coupled bump-in-the-wire programmable FPGA. The integrated FPGA subsystem provides the capability to run latency-sensitive and high-throughput mission-critical applications directly in the network adapter, accelerating host application performance while reducing overall latency and footprint.

In our project we use Avalon-ST to communicate in the FPGA system.

![Figure 1 Typical Avalon-ST interface signals](image)

C. Tools

We use Qsys and fdk in our project.

The Qsys system integration tool saves significant time and effort in the FPGA design process by automatically generating interconnect logic to connect intellectual property (IP) functions and subsystems. Qsys is the next-generation SOPC Builder tool powered by a new FPGA-optimized network-on-a-chip (NoC) technology delivering higher performance, improved design reuse, and faster verification compared to SOPC Builder. The Solarflare® AOE Firmware Development Kit (FDK) enables customers and developers to create and deploy customized applications for the AOE, which moves application processing to solarflare’s ultra-low latency platform thereby accelerating real-
time network data. Solarflare’s comprehensive FDK accelerates and simplifies deployment for a variety of industries, including financial services, with a rich development environment and complete toolkit. The AOE Firmware Development Kit includes a complete toolkit specifically designed to simplify integration of existing logic directly into the data path. The FDK provides a complete development environment with both inline streaming data path interfaces, and host-based configuration and management interfaces. Additionally, the FDK integrates seamlessly with Altera’s Quartus® II design suite to enable the entire development flow for the AOE FPGA allowing final download to the development platform via either Altera or Solarflare download mechanisms.

II. System structure

A. Top level

![Top Level](image)

**Figure 2 Top Level**

Our design has two domains; one is 32 bit 312.5MHz while the other is 64 bit 200MHz. We put our design in the 200MHz domain and surrounded by converters.

B. Converters

![Converter0](image)

**Figure 3 converter0**

The converter contains a bit conversion unit and a dual clock fifo to provide clock domain conversion.
III. Module

A. Killswitch

Our module has one instream and two outstream. Based on the statistic data, the module determines which way one packet goes - pass or block.

![Figure 5 top_simple](image)

Figure 5 top_simple

The streams are using Avalon streaming interface, which has ready, valid, data, error, startofpacket, endofpacket and empty.

![Figure 6 Packet transfer](image)

Figure 6 Packet transfer
We put the packet data, command data and decisions into 3 fifos. Data are from the instream while decisions are generated by the statistic module based on the session tag (source IP, destination IP, port, packet type etc.).
B. **FIFOs**

The Avalon® Streaming (Avalon-ST) Single Clock and Avalon-ST Dual Clock FIFO cores are FIFO buffers which operate with a single clock and separate clocks for input and output ports, respectively. You can configure the cores to include Avalon Memory-Mapped (Avalon-MM) status interfaces to report the FIFO fill level. The Avalon-ST Single Clock and Avalon-ST Dual Clock FIFO cores are SOPC Builder-ready and integrates easily into any SOPC Builder-generated systems.
Figure 9 Dual clock FIFO block diagram

Figure 10
IV. Implementation and simulation

Meaning of signal
1. Ready signal: mark the cycles where transfers may take place.
2. Valid signal: The valid signal qualifies valid data on any cycle where data is being transferred from the source to the sink.
3. Data signal: carries the bulk of the information being transferred from the source to the sink.
4. Error signal: Errors are signaled with the error signal.
5. Start of packet: marks the active cycle containing the start of the packet. This signal is only interpreted when valid is asserted.
6. End of packet: marks the active cycle containing the end of the packet. This signal is only interpreted when valid is asserted.
7. Empty: indicates the number of symbols that are empty during the cycles that mark the end of a packet.

A. Part 1:
The first part of simulation is done on the modelsim.
Experiment expectation: For sender is not in the blacklist, Data in data_sink goes to the sfp terminal, otherwise go to nfc terminal.

Scenario 1: Packet whose information is in the blacklist.
Analyze:

At 300 ps, read_sink turn to 1. At 400 ps, valid_sink turn to 1 and data start to flow in to the component. Thereby the startofpacket_sink turns to 1. **After certain time**, end of packet turn to 1 indicate the transformation is complete. Since it is in the blacklist, decision_q change to 1. All the signal concerns with sfp remain unchanged. valid_nic, startofpacket_nic and endofpacket_nic follow the same pattern as those for sink. We can see the data in data_sink and data in data_nic is identical, indicates the transformation is correct.
We can also the time difference between pfifo_q and data_sink for the same value is 400ps, which is equal with the time difference between endofpacket_sink and startofpacket_nic.

Scenario 2: Packet whose information is not in the blacklist.
Analyze: Compare with scenario 1, since it is not in the blacklist, the difference is decision_q change to 0. All the signal concerns with nic remain unchanged. valid_sfc, startofpacket_sfc and endofpacket_sfc follow the same pattern as those for sink. We can see the data in data_sink and data in data_sfc is identical, indicates the transformation is correct.

**Figure 15**

---

Scenario 3: One packet is not in the blacklist followed by one packet is in the blacklist.

Analyze:
Since first packet is not in blacklist, decision_q is 0. All the signal concerns with nic remain unchanged. valid_sfc, startofpacket_sfc and endofpacket_sfc follow the same pattern as those for
sink. We can see the data in data_sink and data in data_sfc is identical, indicates the transformation is correct.

![Figure 19](image19.png)

Then for the second packet, since it is in the blacklist, decision_q change to 1. All the signal concerns with sfp remain unchanged. valid_nic, startofpacket_nic and endofpacket_nic follow the same pattern as those for sink. We can see the data in data_sink and data in data_nic is identical, indicates the transformation is correct.

![Figure 20](image20.png)

The statistic module:

Meaning of signal
1. session: input containing the session passed from the parser
2. Sel: output to indicate if the session is selected to be blocked, if sel is 1, dump the session, 0, let the session pass.
3. Lut: Look up table to check if the input session is the same with anyone stored in it.
4. Blacklist: Used to store the eligibility of being blocked for the session in the corresponding position in the look up table.
Scenario 1: Packet which is already in the blacklist and packet which is not in the blacklist but appears too many times, they both get blocked.

For this scenario, we pre-stored a session in the look up table and make it to be blocked, so the corresponding position in the black list is 1. We can see this pre-setting in the graph below that at the beginning of the simulation, at the 0th position of the look up table, there is already a session stored which is “c0a80101c0a801fffffffff” and the corresponding position (i.e 0th position) in the black list is set to 1.
Now we start to feed the statistic module with a repeated session of “c0a80101c0a80000ffffffff” followed by “c0a80101c0a801fffffff”, the latter is already in the blacklist and former is not.

![Figure 22](image22)

Now we start to feed the statistic module with a repeated session of “c0a80101c0a80000ffffffff” followed by “c0a80101c0a801fffffff”, the latter is already in the blacklist and former is not.

![Figure 23](image23)

From the graph above we can see that whenever the unblocked session is fed in, the sel signal is always 0, and whenever the blocked session is fed in, the sel goes to 1. The blacklist stays the same til the unblocked session appears more than a certain number of times in a certain time duration. When that happens, the previously unblocked session gets blocked which can be seen from the graph above, we can see the blacklist vector changes from “100 to 110” and the “sel” signal stays at 1 from that transition because both the input session are in the blacklist and should be blocked. So the above indicates the correctness of the simulation.

**Scenario 2: Packet which is not in the blacklist and does not get blocked.**
Similar to the pre-settings in the last scenario, repeated session of “c0a80101c0a80000fffffffff” followed by “c0a80101c0a801fffffffff” is fed to the statistic for several times. The difference is that the session which is not previously stored in the blacklist does not appear enough times to be blocked. We can see from the graph above that the blacklist vector stays “100” which means the un-blacklisted session is not blocked. So the simulation is correct.

**B. Part2:**

After make sure the function correctness on the modelsim, the next part of the simulation is done on the signaltype. We use the two computers, marvin and trillian in the cs lab. One computer is used to send the pcap (tcp packet we define) by tcpreplay. The other computer is used to receive tcp packet. We monitor the signaltype on the receiver computer.

Experiment expectation: For sender is not in the blacklist, Data in data_sink goes to the sfp terminal, otherwise go to nfc terminal.

**Scenario 1 : computer receive packets which are not in the blacklist**

Since the packet is not in the blacklist, terminal sfp receive the data and nic doesn’t. The data in data_sink and data in data_sfp is identical, which indicate the function works well.
Scenario 2: computer receive packets which are in the blacklist

Since the packet is in the blacklist, terminal nic receive the data and sfp doesn’t. The data in data_sink and data in data_sfp is identical, which indicate the function works well.

Figure 26

Figure 27
**C. Part 3:**

The last part of the simulation is done on the wireshark. Like part 2, one computer is used to send the pcapby tcpreplay. The other computer is used to receive tcp packet. We monitor the wireshark on the receiver computer.

Scenario settings and expected result: There are basically three kinds of packets. One is already in the blacklist and should be blocked all the time, this packet’s destination ends with 255. The other two are not in the blacklist and one of them appears at a normal rate which means it will not be blocked, the other one appears frequently enough to be blocked. These two packets’ destination ends with 111 and 222 respectively.

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>000000</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
</tbody>
</table>

From the picture above we can see that the packet which is pre-stored in the blacklist does not show up at all which means it is blocked from the beginning of the transmitting. The other one which ends with “111” is not blocked because it does not appear frequently enough. But the one ends with “222” appears only 8 times, because in that time duration, it is detected as “too fast” by the statistic module and therefore got blocked. From the graph above, we can verify the correctness of the statistic module.
V. Contribution

Qiushi Ding (30%): Design of whole system structures and interfaces. Build up Qsys environment. Build up debugging environment (modelsim do files & signaltap). Build up testing environment (tcp replay and sfp connection). Coding for the top top level vhdl file for fifo operation.

Liheng Wang (20%): Design and debug a prototype program which can choose the data either go nic terminal or sfp terminal based on it’s ip address. Run its simulation on the modelsim. Run all the simulation without the part concern with statistic of the final program on modelsim, signaltap and wireshark.

Yuyang Wang (18%): Design and debug the all statistic module, run part of the simulations on the modelsim signaltap and wireshark.

Bokai Chen (18%): Debugging the final version of killswitch & designing CAM for look up table.

Jingshu Fang (14%): Wrote the testbench to read in a pcap file in TCP protocol as input and test the correctness of the killswitch system. Wrote a compile.tcl file to compile files of different modules for simulation on modelsim.
VI. Source Code

-- killswitch.vhd

-- This file was auto-generated as a prototype implementation of a module
-- created in component editor. It ties off all outputs to ground and
-- ignores all inputs. It needs to be edited to make it do something
-- useful.

--

-- This file will not be automatically regenerated. You should check it in
-- to your version control system if you want to keep it.

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity killswitch is
  generic(
    id_length : natural := 10 * 8;
    statistic_width : natural := 96
    -- fixengine_width : natural := 8;
    -- fix_tag_width : natural := 3*8;
    -- fix_value_width : natural := 256
  );
  port ( clk : in std_logic := '0'; -- clk.clk
    reset : in std_logic := '0'; -- reset_n.reset
  );
--Avalon_streaming sink

ready_sink : out std_logic; -- stream_in.ready
valid_sink : in  std_logic := '0'; -- .valid
data_sink : in  std_logic_vector(63 downto 0) := x"0000000000000000"; -- .data
error_sink : in  std_logic := '1'; -- .error
startofpacket_sink : in  std_logic := '0'; -- .startofpacket
endofpacket_sink : in  std_logic := '0'; -- .endofpacket
empty_sink : in  std_logic_vector(2 downto 0) := "100"; - .empty

--Avalon_streaming source to sfp

ready_sfp : in  std_logic := '0'; -- avalon_streaming_source.ready
valid_sfp : out std_logic; -- .valid
data_sfp : out std_logic_vector(63 downto 0); -- .data
error_sfp : out std_logic := '0'; -- .error
startofpacket_sfp : out std_logic; -- .startofpacket
endofpacket_sfp : out std_logic; -- .endofpacket
empty_sfp : out std_logic_vector(2 downto 0); -- .empty

--Avalon_streaming source to nic

ready_nic : in  std_logic := '0'; -- avalon_streaming_source.ready
valid_nic : out std_logic; -- .valid
data_nic : out std_logic_vector(63 downto 0); -- .data
error_nic : out std_logic := '0'; -- .error
startofpacket_nic : out std_logic; -- .startofpacket
endofpacket_nic : out std_logic; -- .endofpacket

empty_nic : out std_logic_vector(2 downto 0)--.

endofpacket

-- --FIX_Engine

-- fix_tag : in std_logic_vector(fix_tag_width-1 downto 0) := (others => '0');

-- fix_tag_v : in std_logic :='0';

-- fix_value : in std_logic_vector(fix_value_width-1 downto 0) := (others => '0');

-- fix_value_v : in std_logic :='0';

-- fix_data_out : out std_logic_vector(fixengine_width-1 downto 0) := (others => '0');

-- fix_data_valid_out : out std_logic := '0'

);
end entity killswitch;

architecture rtl of killswitch is

signal moveout: std_logic :='0';

signal pfifo_empty: std_logic := '1';

signal pfifo_full: std_logic := '0';

signal pfifo_q: std_logic_vector(63 downto 0) := (others => '0');

signal cfifo_empty: std_logic := '1';

signal cfifo_full: std_logic := '0';

signal cfifo_q: std_logic_vector(7 downto 0) := (others => '0');
signal next_packet_req : std_logic := '0';
signal decision : std_logic_vector (0 downto 0) := "0"; -- 1 for pass, 0 for drop
signal decision_v : std_logic := '0';
signal decision_empty : std_logic := '0';
signal decision_full : std_logic := '0';
signal decision_q : std_logic_vector (0 downto 0) := "0'';

signal package_data: std_logic_vector(63 downto 0) := x"0000000000000000";
signal package_data_out: std_logic_vector(63 downto 0) := x"0000000000000000";
signal command_data: std_logic_vector(7 downto 0) := x"00";
signal command_data_out: std_logic_vector(7 downto 0) := x"00";

signal ip_src_addr: std_logic_vector(31 downto 0) := x"00000000";
signal ip_dest_addr: std_logic_vector(31 downto 0) := x"00000000";
signal tcp_src_port: std_logic_vector(15 downto 0) := x"0000";
signal tcp_dest_port: std_logic_vector(15 downto 0) := x"0000";
--signal fix_comp_id: std_logic_vector(id_length-1 downto 0) := (others => '0');
--signal fix_targ_id: std_logic_vector(id_length-1 downto 0) := (others => '0');

signal start_statistic: std_logic := '0';
--signal start_fixengine: std_logic := '0';
signal session_tag: std_logic_vector(statistic_width-1 downto 0) := (others => '0');
signal drop : std_logic := '0';
signal drop_v: std_logic := '0';
signal stat_addr_out: integer range 0 to 511:=0;
signal release : std_logic := '1';
signal ready_in : std_logic := '0';
signal fix_data : std_logic_vector(63 downto 0) := (others => '0');
signal fix_data_valid : std_logic := '0';

begin
V1: entity work.packetbuffer port map (clk, package_data, moveout, reset, ready_in, pfifo_empty, pfifo_full,pfifo_q);
V2: entity work.commandbuffer port map (clk, command_data, moveout, reset, ready_in, cfifo_empty, cfifo_full,cfifo_q);
V3: entity work.decision_buffer port map (clk, decision, next_packet_req, reset, decision_v, decision_empty, decision_full,decision_q);
V4: entity work.statistic port map (clk,session_tag,start_statistic,reset,stat_addr_out,drop,drop_v);

process (clk)
variable offset: in integer range 0 to 1000 := 0;
variable offset_decision_sfp: integer range 0 to 1000 := 0;
variable offset_decision_nic: integer range 0 to 1000 := 0;
variable offset_decision: integer range 0 to 1000 := 0;
begin
if (rising_edge(clk)) then
if reset = '1' then
ip_src_addr <= (others => '0');
ip_dest_addr <= (others => '0');
tcp_src_port <= (others => '0');
tcp_dest_port <= (others => '0');
--fix_comp_id <= (others => '0');
--fix_targ_id <= (others => '0');
start_statistic <= '0';
offset := 0;

valid_sfp <= '0';
startofpacket_sfp <= '0';
endofpacket_sfp <= '0';
empty_sfp <= "000";
error_sfp <= '0';
data_sfp <= (others => '0');

valid_nic <= '0';
startofpacket_nic <= '0';
endofpacket_nic <= '0';
empty_nic <= "000";
error_nic <= '0';
data_nic <= (others => '0');

else--not reset
  start_statistic <= '0';
  if pfifo_full = '0' and cfifo_full = '0' and decision_full = '0' then
    decision(0)<=drop;
    decision_v<=drop_v;
    ready_sink <= '1';
    if valid_sink = '1' then
      command_data(5) <= startofpacket_sink;
      command_data(4) <= endofpacket_sink;
      command_data(3 downto 1) <= empty_sink;
command_data(0) <= error_sink;
package_data <= data_sink;
ready_in <= '1';

--start to analyze
if startofpacket_sink = '1' then--startofpacket_sink
    offset := 0;
    fix_data_valid <= '0';
    start_statistic <= '0';
    release <= '0';
elsif endofpacket_sink = '1' then--endofpacket_sink
    if release = '0' then
        start_statistic <= '1';
        session_tag <= ip_src_addr & ip_dest_addr & tcp_src_port & tcp_dest_port;
        --session_tag generated
    else
        decision(0) <= '0';
        decision_v <= '1';
    end if;
else--others, during the packet
    start_statistic <= '0';
    offset := offset + 1;--0 will be 1
    case offset is
        when 2 => -- let go all non-ipv4 packets
            if package_data(31 downto 0) /= x"08004500" then
                release <= '1';
            end if;
        when 3 => -- let go all non-tcp packets
            ..
        ..
if package_data(7 downto 0) /= x"06" then
    release <= '1';
end if;

when 4 => -- take record of ip source address
    -- & first 16 bits of destination address
    if release = '0' then
        ip_src_addr <= package_data(47 downto 16);
        ip_dest_addr(31 downto 16) <= package_data(15 downto 0);
    end if;

when 5 => -- take record of remaining 16 bits of destination address,
    -- TCP source port & destination port
    if release = '0' then
        ip_dest_addr(15 downto 0) <= package_data(63 downto 48);
        tcp_src_port <= package_data(47 downto 32);
        tcp_dest_port <= package_data(31 downto 16);
    end if;

-- when 8 => -- let go all non-fix packets
-- if data_sink(47 downto 0) /= x"383d4649582e" then --try to find "8=FIX."
--     release <= '1';
--     else
--         release <= '0';
--     end if;

when others => -- do nothing for non-fix packets, forward data to
    FIX parser
    if release = '0' and offset > 9 then
fix_data(15 downto 0) <= data_sink(63 downto 48);
fix_data(63 downto 16) <= package_data(47 downto 0);
fix_data_valid <= '1';
end if;
end case;
end if;
else--valid_sink = '0'
  ready_in <= '0';
end if;
else --any fifo is full
  ready_sink <= '0';
  ready_in <= '0';
  if decision_full = '0' then
    decision(0)<=drop;
    decision_v<=drop_v;
  end if;
end if;--
--
--
if (ready_sfp = '1' or ready_nic = '1') and (pfifo_empty = '0' and cfifo_empty = '0' and decision_empty = '0') then
  if (ready_sfp = '1' or ready_nic = '1') and decision_empty = '0'then
    if decision_q = "0" and ready_sfp = '1' then -- pass
      offset_decision := offset_decision + 1;
      offset_decision_sfp := offset_decision_sfp + 1;
      data_nic <= (others => '0');
      startofpacket_nic <= '0';
      endofpacket_nic <= '0';
empty_nic <= "000";
error_nic <= '0';
valid_nic <= '0';
--offset_decision_nic := 0;

case offset_decision is
when 1 => -- let go all non-fix packets
    moveout <= '0';
    next_packet_req <= '0';
    valid_sfp <= '0';
when 2 =>
    moveout <= '1';
when others => -- do nothing for non-fix packets, forward data to FIX parser
    data_sfp <= pfifo_q;
    valid_sfp <= '1';
    startofpacket_sfp <= cfifo_q(5);
    endofpacket_sfp <= cfifo_q(4);
    empty_sfp <= cfifo_q(3 downto 1);
    error_sfp <= cfifo_q(0);
    valid_nic <= '0';
    if cfifo_q(4) = '1' then
        next_packet_req <= '1';
        moveout <= '0';
        offset_decision := 0;
    else
        next_packet_req <= '0';
        moveout <= '1';
endcase;
end if;
end case;

elsif decision_q = "1" and ready_nic = '1' then -- drop

-- offset_decision_nic := offset_decision_nic + 1;
-- offset_decision_sfp := 0;
offset_decision := offset_decision + 1;

data_sfp <= (others => '0');
startofpacket_sfp <= '0';
endofpacket_sfp <= '0';
empty_sfp <= "000";
error_sfp <= '0';
valid_sfp <= '0';
offset_decision_sfp := 0;
case offset_decision is

when 1 => -- let go all non-fix packets

moveout <= '0';
next_packet_req <= '0';
valid_nic <= '0';

when 2 =>

moveout <= '1';

when others => -- do nothing for non-fix packets, forward data to FIX parser

moveout <= '1';
data_nic <= pfifo_q;
valid_nic <= '1';
startofpacket_nic <= cfifo_q(5);
endofpacket_nic <= cfifo_q(4);
empty_nic <= cfifo_q(3 downto 1);
error_nic <= cfifo_q(0);
valid_sfp <= '0';
if cfifo_q(4) = '1' then
    next_packet_req <= '1';
    moveout <= '0';
    offset_decision := 0;
else
    next_packet_req <= '0';
    moveout <= '1';
end if;
end case;
else
    offset_decision := 0;
data_nic <= (others => '0');
data_sfp <= (others => '0');
startofpacket_nic <= '0';
endofpacket_nic <= '0';
startofpacket_sfp <= '0';
endofpacket_sfp <= '0';
empty_nic <= "000";
error_nic <= '0';
empty_sfp <= "000";
error_sfp <= '0';
moveout <= '0';
valid_sfp <= '0';
valid_nic <= '0';
next_packet_req <= '0';
offset_decision_sfp := 0;
offset_decision_nic := 0;
end if;
else
offset_decision := 0;
data_nic <= (others => '0');
data_sfp <= (others => '0');
startofpacket_nic <= '0';
endofpacket_nic <= '0';
startofpacket_sfp <= '0';
endofpacket_sfp <= '0';
empty_nic <= "000";
error_nic <= '0';
empty_sfp <= "000";
error_sfp <= '0';
moveout <= '0';
valid_sfp <= '0';
valid_nic <= '0';
next_packet_req <= '0';
offset_decision_sfp := 0;
offset_decision_nic := 0;
end if;
end if;
end if;
end process;
end architecture rtl; -- of killswitch
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity statistic is

    generic(
        cam_width: integer := 96-1
    );

    port(
        clk: in std_logic;
        session: in std_logic_vector(95 downto 0) := (others => '0');--session passed from parser
        valid: in std_logic;
        rst: in std_logic;--global reset
        hit_addr_out: out integer range 0 to 8 := 0;
        sel: out std_logic := '0';--select value, pass the seesion when 0, block it when 1
        sel_v: out std_logic := '0'
    );

end statistic;

architecture func of statistic is

    type ram_stat_data is array(0 to 8) of integer range 0 to 8;
    type ram_lut is array(0 to 8) of std_logic_vector(cam_width downto 0);

type next_state_type is(s0,s1);
signal statistic_stat_data: ram_stat_data;--store the statistic data
signal lut: ram_lut;--look up table for comparing
signal statistic_session: std_logic_vector(cam_width downto 0);--store the session
--signal statistic_lut: ram_lut;--store the session for outputing
signal next_state: next_state_type;--current next_state
signal count: integer range 0 to 8;--indicate the last position in the look up table

--signal ram_u: std_logic;--indicating the transfer action when it is set to 1;
--signal update: std_logic;--flag to indicate time to update the blacklist
signal blacklist: std_logic_vector(0 to 8);

begin

process(clk,rst)

variable j: integer := 0;--counter, used as a time period
variable hit: std_logic := '0';--indicate that a certain session has appeared before
variable hit_addr: integer range 0 to 8;
begin

if(rising_edge(clk)) then

if(rst = '1') then--reset settings

next_state <= s0;
statistic_session <= (others => '0');

for c in 0 to 8 loop
if c=0 then
    lut(c) <= \texttt{x"c0a80101c0a801ff0fffff"};
    blacklist(c) <= '1';
else
    lut(c) <= (others => '0');
    blacklist(c) <= '0';
end if;
end loop;

count <= 1;
else
    j := j + 1;
    if (j > 200000000) then
        for a1 in 0 to 8 loop
            statistic_stat_data(a1) <= 0;
        end loop;
        j := 0;
    end if;
end if;

case next_state is
    when s0 =>
        if valid = '0' then
            next_state <= s0;
        else
            next_state <= s1;
            statistic_session <= session; -- store the session to a vector
        end if;
    for further use
for addr in 0 to 8 loop -- check the look up table with the port value which lasts for one cycle

    if (session = lut(addr) and hit = '0') then
        hit := '1'; -- match found

        -----------------------------
        hit_addr := addr;
        hit_addr_out <= addr;
        -----------------------------

    else
        hit := hit;
    end if;
end loop;

if hit = '1' then
    sel <= blacklist(hit_addr);
else
    sel <= '0';
end if;

sel_v <= '1';
end if;

when s1 =>
    next_state <= s0;
    sel_v <= '0';
    if (hit = '1') then
        statistic_stat_data(hit_addr) <= statistic_stat_data(hit_addr) + 1;
    end if;
hit := '0';

--sel <= blacklist(hit_addr);

if (statistic_stat_data(hit_addr) > 5) then
    blacklist(hit_addr) <= '1';
end if;

else
    lut(count) <= statistic_session;
    statistic_stat_data(count) <= 1;

    hit_addr_out <= count;
    count <= count + 1;
end if;
end case;
end if;
end if;
end if;
end process;
end func;