FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

NETWORK SWITCHING SYSTEM DESIGN WITH VHDL

GROUP PA09

Abednego Zebua	**2306161883**
Grace Yunike Margaretha S.	**2306267031**
Naufal Hadi Rasikhin	**2306231366**
Dimas Ananda Sutiardi	**2306250586**

PREFACE

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya, sehingga laporan proyek akhir dengan judul "Network Switching System Design with VHDL" ini dapat diselesaikan dengan baik. Ucapan terima kasih kami sampaikan kepada asisten laboratorium yang telah memberikan bimbingan dan arahan, serta teman-teman yang turut berkontribusi dalam proses pengerjaan laporan ini.

Laporan ini disusun sebagai bagian dari pemenuhan tugas akhir pada mata kuliah Perancangan Sistem Digital. Dalam laporan ini, kami membahas tentang desain dan implementasi replika sistem hardware switch yang berfungsi untuk menerima dan mengirimkan paket data berdasarkan alamat MAC tujuan menggunakan VHDL. Proyek ini dirancang untuk mereplikasi cara kerja switch dalam jaringan komputer yang menggunakan tabel MAC untuk pengiriman data antar port. Desain sistem ini dilengkapi dengan buffer dan menggunakan pendekatan struktural serta FSM dengan microprogramming sebagai inti pengolahan data dalam sistem.

Selama pengerjaan proyek ini, kami menyadari bahwa terdapat keterbatasan baik dari segi pengetahuan maupun pengalaman. Oleh karena itu, kritik dan saran dari pembaca sangat kami harapkan untuk perbaikan di masa yang akan datang. Kami juga memohon maaf apabila terdapat kekurangan dalam penyusunan laporan ini, dan semoga laporan ini dapat memberikan pemahaman lebih dalam mengenai penerapan konsep-konsep VHDL dalam perancangan sistem jaringan komputer.

Depok, 9 Desember 2024

Group PA09

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND

Setelah mempelajari sedikit banyak mengenai Jaringan Komputer, kami menemukan satu perangkat yang cukup menarik dalam ekosistem jaringan dari skala kecil hingga besar, yaitu sebuah switch. Switch adalah perangkat jaringan yang digunakan untuk mensegmentasi jaringan ke dalam sub jaringan yang berbeda yang disebut subnet atau segmen LAN. Switch bertanggung jawab untuk memfilter dan meneruskan paket di antara segmen LAN berdasarkan alamat MAC.

Switch merupakan sebuah perangkat digital yang bekerja secara sekuensial. Berdasarkan informasi tersebut, kelompok kami merasa bahwa dapat melakukan replikasi logic perangkat tersebut dengan VHDL (VHSIC Hardware Description Language).

1.2 PROJECT DESCRIPTION

Project ini bertujuan untuk membuat replika perangkat keras dari sistem switch pada jaringan komputer dengan menggunakan VHDL. Switch yang didesain memiliki kemampuan menerima paket dari beberapa input port yang dimilikinya, kemudian akan menentukan alamat pengiriman packet tersebut berdasarkan mac-address tujuan dalam MAC table yang dimilikinya. Model desain akan mereplikasi switch yang memiliki sistem buffer, jadi frame yang diterima bersamaan pada satu waktu akan dikirimkan berurutan. Jika tidak ditemukan alamat MAC tujuan, maka switch akan melakukan broadcast kepada seluruh switch aktif.

Design akan berupa FSM dengan Microprogramming yang akan berperan sebagai processor dari Switch tersebut. Loop construct akan digunakan didalam Testbench bersamaan dengan Procedure/Functions untuk mereplikasi contoh input berupa paket yang akan diterima oleh switch.

1.3 OBJECTIVES

Tujuan-tujuan dari proyek adalah sebagai berikut:

1. Membuat replika perangkat keras switch jaringan komputer menggunakan VHDL.
2. Memodelkan logika pemrosesan paket data berdasarkan MAC address menggunakan FSM dan *microprogramming*.
3. Mengimplementasikan *structural style* untuk konstruksi entity.
4. Mengintegrasikan Perangkat keras dan lunak.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Ketua	● Membahas dan merancang proyek ● Membuat main code ● Melakukan Bug fixing ● Melakukan integrasi entitas yang ada	Abednego Zebua
Anggota	● Membahas dan merancang proyek ● Membuat laporan PPT ● Melakukan test simulasi dan melaporkan bug	Grace Yunike Margaretha Sitorus
Anggota	● Membahas dan merancang proyek ● Melakukan Bug fixing dan testbench ● Membuat README.md	Naufal Hadi Rasikhin
Anggota	● Membahas dan merancang proyek ● Membuat laporan makalah ● Melakukan test simulasi dan melaporkan bug	Dimas Ananda Sutiardi

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

Alat-alat yang digunakan dalam pembuatan proyek akhir Perancangan Sistem Digital adalah sebagai berikut:

- Visual Studio Code sebagai code editor
- ModelSim sebagai media simulasi
- Quartus sebagai synthesizer
- GitHub sebagai media kolaborasi

2.2 IMPLEMENTATION

Pada proyek ini kami mengimplementasikan beberapa Desain dalam VHDL untuk membuat proyek yang akan kami buat salah satunya yaitu Dataflow untuk mendeskripsikan perilaku sistem digital dengan fokus pada aliran data, Test Bench digunakan untuk memvalidasi dan menguji keseluruhan fungsi dari desain sistem switch jaringan, Structural digunakan untuk menunjukkan hubungan antara komponen perangkat keras secara eksplisit , Looping digunakan untuk menjalankan satu atau beberapa pernyataan berulang kali, FSM berfungsi sebagai pengontrol utama dan Microprogramming digunakan untuk mengatur operasi Finite State Machine (FSM) dengan cara yang modular dan fleksibel. .

Desain proyek ini terdiri dari beberapa komponen utama yang bekerja agar sistem network switching ini dapat berjalan dengan baik, komponen - komponen utama tersebut meliputi:

- Switch

  Sebagai komponen utama yang menghubungkan serta mengatur logika dari komponen-komponen lainnya.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY switch IS
```

```
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        rw_bit_in : IN STD_LOGIC; --for SwCAM
        sent_frame : OUT STD_LOGIC_VECTOR(167 DOWNTO 0);
--Assuming 168-bit frame
        src_port : OUT STD_LOGIC_VECTOR(3 DOWNTO 0); -- from
decoder
        src_mac : OUT STD_LOGIC_VECTOR(47 DOWNTO 0); -- from
decoder
        dest_port : OUT STD_LOGIC_VECTOR(3 DOWNTO 0); -- from
decoder
        dest_mac : OUT STD_LOGIC_VECTOR(47 DOWNTO 0); -- from
decoder
        output_payload : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        --fa01
        fa01_MAC : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
        fa01_InoutBit : IN STD_LOGIC; --indicator for port
receiving = "0", sending = "1";
        fa01_Payload : IN STD_LOGIC_VECTOR(7 DOWNTO 0); --data to
send
        fa01_DataIn : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); --data
that received
        fa01_DestMac : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
        fa01_FrameOut : OUT STD_LOGIC_VECTOR(167 DOWNTO 0);
        --fa02
        fa02_MAC : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
        fa02_InoutBit : IN STD_LOGIC;
        fa02_Payload : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        fa02_DataIn : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        fa02_DestMac : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
        fa02_FrameOut : OUT STD_LOGIC_VECTOR(167 DOWNTO 0);
        --faa03
        fa03_MAC : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
        fa03_InoutBit : IN STD_LOGIC;
        fa03_Payload : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        fa03_DataIn : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        fa03_DestMac : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
```

```
        fa03_FrameOut : OUT STD_LOGIC_VECTOR(167 DOWNTO 0)
        --add ports later
    );
END ENTITY switch;

ARCHITECTURE rtl OF switch IS
    COMPONENT switchport IS
        PORT (
            clk : IN STD_LOGIC;
            port_id : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            frame_out : OUT STD_LOGIC_VECTOR(167 DOWNTO 0); --
Changed to OUT
            inout_bit : IN STD_LOGIC; -- generate frame when "1",
0 is idle or read mode
            data_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            payload : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            MAC_dest : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            MAC_add : IN STD_LOGIC_VECTOR(47 DOWNTO 0)
        );
    END COMPONENT switchport;

    COMPONENT SwCAM IS --simple cam to keep MAC-Add table
        PORT (
            main_clk : IN STD_LOGIC;
            main_rst : IN STD_LOGIC;
            rw_bit   : IN STD_LOGIC;
            enable   : IN STD_LOGIC;
            mac_find : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            port_out : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
--assuming a switch with 24 ethernet-port, so max bit is 2^5
            hit_flag : OUT STD_LOGIC_VECTOR(1 DOWNTO 0) --if found
'11', not found '10';
        );
    END COMPONENT SwCAM;

    COMPONENT frame_decoder IS -- frame decoder
        PORT (
```

```
            frame_in : IN STD_LOGIC_VECTOR(167 DOWNTO 0); --
Assuming 168-bit frame

            dest_mac : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);

            src_mac : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);

            payload_byte : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)

        );

    END COMPONENT frame_decoder;


    TYPE State_Type IS (LOAD, ACTIVE, DECODE, SEARCH, HOLD,
ASSIGN, FORWARD, RECEIVE, COMPLETE);

    SIGNAL state : State_Type := LOAD;

    SIGNAL rw_bit : STD_LOGIC := '0'; -- 0 = read, 1 = write,
default read

    SIGNAL enable : STD_LOGIC := '0';

    SIGNAL temp_hit_flag : STD_LOGIC_VECTOR(1 DOWNTO 0); --if
found '11', not found '10';

    SIGNAL buffer_frame : STD_LOGIC_VECTOR(2015 DOWNTO 0) :=
(others => '0'); --size  of 12 frame

    SIGNAL mac_find : STD_LOGIC_VECTOR(47 DOWNTO 0); -- decoded

    SIGNAL decode_frame : STD_LOGIC_VECTOR(167 DOWNTO 0); -- to
decode

    SIGNAL temp_src_port : STD_LOGIC_VECTOR(3 DOWNTO 0);

    SIGNAL temp_dest_port : STD_LOGIC_VECTOR(3 DOWNTO 0);
--decoded;

    SIGNAL temp_payload : STD_LOGIC_VECTOR(7 DOWNTO 0); -- decoded


    --temp frame out for ports, after being encoded, add for new
ports later.

    signal temp_fa01_FrameOut : STD_LOGIC_VECTOR(167 DOWNTO 0);

    signal temp_fa02_FrameOut : STD_LOGIC_VECTOR(167 DOWNTO 0);

    signal temp_fa03_FrameOut : STD_LOGIC_VECTOR(167 DOWNTO 0);


    --temp data in for ports, after being encoded, add for new
ports later.

    signal temp_fa01_DataIn : STD_LOGIC_VECTOR(7 DOWNTO 0);

    signal temp_fa02_DataIn : STD_LOGIC_VECTOR(7 DOWNTO 0);

    signal temp_fa03_DataIn : STD_LOGIC_VECTOR(7 DOWNTO 0);


    --for assignments
```

```
    SIGNAL zeros : STD_LOGIC_VECTOR(167 DOWNTO 0) := (OTHERS =>
'0');
    SIGNAL buffer_index : integer := 1;
    SIGNAL hold_count   : integer := 2;
    CONSTANT port_num : integer := 3;
BEGIN
    SwCAM1 : SwCAM PORT MAP(
        main_clk => clk,
        main_rst => reset,
        rw_bit => rw_bit,
        enable => enable,
        mac_find => mac_find,
        port_out => temp_dest_port,
        hit_flag => temp_hit_flag
    );

    FrameDecoder : frame_decoder PORT MAP(
        frame_in => decode_frame,
        dest_mac => mac_find,
        src_mac => src_mac,
        payload_byte => temp_payload
    );

    --instantiation, 3 ports for starters
    fa01 : switchport PORT MAP(
        clk => clk,
        port_id => "0001",
        inout_bit => fa01_InoutBit,
        data_in => temp_fa01_DataIn,
        payload => fa01_Payload,
        MAC_Add => fa01_MAC,
        MAC_Dest => fa01_DestMac,
        frame_out => temp_fa01_FrameOut
    );
```

```
    fa02 : switchport PORT MAP(
        clk => clk,
        port_id => "0010",
        inout_bit => fa03_InoutBit,
        data_in => temp_fa02_DataIn,
        payload => fa02_Payload,
        MAC_Add => fa02_MAC,
        MAC_Dest => fa02_DestMac,
        frame_out => temp_fa02_FrameOut
    );

    fa03 : switchport PORT MAP(
        clk => clk,
        port_id => "0011",
        inout_bit => fa03_InoutBit,
        data_in => temp_fa03_DataIn,
        payload => fa03_Payload,
        MAC_Add => fa03_MAC,
        MAC_Dest => fa03_DestMac,
        frame_out => temp_fa03_FrameOut
    );

    --add other ports instantiation later.
    PROCESS (clk, reset)
    BEGIN
        IF reset = '1' THEN
            state <= LOAD;
            sent_frame <= (OTHERS => '0');
            src_port <= (OTHERS => '0');
            dest_mac <= (OTHERS => '0');
            dest_port <= (OTHERS => '0');
            output_payload <= (OTHERS => '0');
        ELSIF rising_edge(clk) THEN
            CASE state IS
                WHEN LOAD =>
```

```
                    buffer_index <= 1;
                    buffer_frame(2015 DOWNTO 1848) <=
temp_fa01_FrameOut;
                    buffer_frame(1847 DOWNTO 1680) <=
temp_fa02_FrameOut;
                    buffer_frame(1679 DOWNTO 1512) <=
temp_fa03_FrameOut;
                    --for future ports
                    --buffer_frame(1511 DOWNTO 1344) <=
fa04_FrameOut;
                    --buffer_frame(1343 DOWNTO 1176) <=
fa05_FrameOut;
                    --buffer_frame(1175 DOWNTO 1008) <=
fa06_FrameOut;
                    --buffer_frame(1007 DOWNTO 840) <=
fa07_FrameOut;
                    --buffer_frame(839 DOWNTO 672) <=
fa08_FrameOut;
                    --buffer_frame(671 DOWNTO 504) <=
fa09_FrameOut;
                    --buffer_frame(503 DOWNTO 336) <=
fa010_FrameOut;
                    --buffer_frame(335 DOWNTO 168) <=
fa011_FrameOut;
                    --buffer_frame(167 DOWNTO 0) <=
fa012_FrameOut;
                    state <= ACTIVE;
                WHEN ACTIVE =>
                    temp_src_port <=
STD_LOGIC_VECTOR(to_unsigned(buffer_index, 4));
                    decode_frame <= buffer_frame(2015 DOWNTO
1848);
                    sent_frame <= buffer_frame(2015 DOWNTO 1848);
                    state <= DECODE;
                WHEN DECODE =>
                    IF (decode_frame = zeros) THEN
                        state <= COMPLETE;
                    ELSIF (decode_frame(167) = 'U') THEN
                        state <= COMPLETE;
                    ELSE
                        state <= SEARCH;
```

```
                    END IF;
                    buffer_frame(2015 DOWNTO 0) <=
buffer_frame(1847 DOWNTO 0) & zeros; --shift left
                WHEN SEARCH =>
                    enable <= '1';
                    if (rw_bit_in = '1') then
                        rw_bit <= '1';
                    else rw_bit <= '0';
                    end if;
                    state <= HOLD;
                WHEN HOLD =>
                    if (hold_count = 1) then
                        state <= ASSIGN;
                    else state <= HOLD;
                    end if;
                    hold_count <= hold_count - 1;
                WHEN ASSIGN =>
                    hold_count <= 2;
                    --assign
                    fa01_FrameOut <= temp_fa01_FrameOut;
                    fa02_FrameOut <= temp_fa02_FrameOut;
                    fa03_FrameOut <= temp_fa03_FrameOut;
                    --add ports in the future
                    src_port <= temp_src_port;
                    dest_port <= temp_dest_port;
                    dest_mac <= mac_find;
                    state <= FORWARD;
                WHEN FORWARD =>
                    output_payload <= temp_payload;
                    IF (temp_hit_flag = "10") THEN --broadcast,
except to source port.
                        IF (temp_src_port /= "0001") THEN
                            fa01_DataIn <= temp_payload;
                            temp_fa01_DataIn <= temp_payload;
                        END IF;
```

```
                        IF (temp_src_port /= "0010") THEN
                            fa02_DataIn <= temp_payload;
                            temp_fa02_DataIn <= temp_payload;
                        END IF;

                        IF (temp_src_port /= "0011") THEN
                            fa03_DataIn <= temp_payload;
                            temp_fa03_DataIn <= temp_payload;
                        END IF;
                        --add another conditions for port id's as
new ports added.
                    ELSIF (temp_hit_flag = "11") THEN
                        CASE temp_dest_port IS
                            WHEN "0001" =>
                                IF (temp_src_port /= "0001") THEN
                                    fa01_DataIn <= temp_payload;
                                    temp_fa01_DataIn <=
temp_payload;
                                END IF;

                            WHEN "0010" =>
                                IF (temp_src_port /= "0010") THEN
                                    fa02_DataIn <= temp_payload;
                                    temp_fa02_DataIn <=
temp_payload;
                                END IF;

                            WHEN "0011" =>
                                IF (temp_src_port /= "0011") THEN
                                    fa03_DataIn <= temp_payload;
                                    temp_fa03_DataIn <=
temp_payload;
                                END IF;
                        --add another conditions for port id's as
new ports added.

                            WHEN OTHERS =>
```

```
                                -- Optionally, you can add a
default action here if necessary
                        END CASE;
                    END IF;
                    state <= RECEIVE;
                WHEN RECEIVE =>


                    state <= COMPLETE;
                WHEN COMPLETE =>
                    enable <= '0';
                    rw_bit <= '0'; --write behavior not set
                    buffer_index <= buffer_index + 1;
                    if (buffer_index = port_num) then state <=
LOAD;
                    else state <= ACTIVE;
                    end if;
            END CASE;
        END IF;
    END PROCESS;
END ARCHITECTURE rtl;
```

Code 1 - Switch.vhd

- Switchport

  Komponen yang bertugas mengirim maupun menerima data frame. Ketika menerima, dapat melakukan decode frame yang diterima tersebut. sedangkan ketika mengirim, dapat menyusun data-data yang ada menjadi sebuah frame dengan bantuan frame_encoder

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE STD.textio.ALL;
USE ieee.std_logic_textio.ALL;

ENTITY switchport IS
    PORT (
```

```
        clk : IN STD_LOGIC;
        port_id : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        frame_out : OUT STD_LOGIC_VECTOR(167 DOWNTO 0); -- Changed
to OUT
        inout_bit : IN STD_LOGIC; -- generate frame when "1", 0 is
idle or read mode
        data_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        payload : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        MAC_dest : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
        MAC_add : IN STD_LOGIC_VECTOR(47 DOWNTO 0)
    );
END ENTITY switchport;

ARCHITECTURE rtl OF switchport IS
    CONSTANT total_port : INTEGER := 12;
    COMPONENT frame_encoder IS -- frame encoder
        PORT (
            frame_out : OUT STD_LOGIC_VECTOR(167 DOWNTO 0); --
Assuming 168-bit frame
            dest_mac : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            src_mac : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            payload_byte : IN STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
    END COMPONENT frame_encoder;

    SIGNAL tempDest : STD_LOGIC_VECTOR(47 DOWNTO 0);
    SIGNAL tempAdd : STD_LOGIC_VECTOR(47 DOWNTO 0);
    SIGNAL tempOut : STD_LOGIC_VECTOR(167 DOWNTO 0);
    SIGNAL tempPayload : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL frame_ready : STD_LOGIC := '0'; -- Trigger for
frame_out update
BEGIN
    -- Instance of frame_encoder
    encoder : frame_encoder PORT MAP(
        frame_out => tempOut,
        dest_mac => tempDest,
        src_mac => tempAdd,
```

```
        payload_byte => tempPayload
    );
    PROCESS (inout_bit, clk)
    BEGIN
        IF rising_edge(clk) THEN
            CASE inout_bit IS
                WHEN '0'=>
                    frame_out <= (others=> '0');
                    frame_ready <= '0';
                WHEN '1' => -- Send state
                    tempDest <= MAC_dest;
                    tempAdd <= MAC_add;
                    tempPayload <= payload;
                    frame_ready <= '1'; -- Mark frame as ready to
update
                WHEN OTHERS =>
                    tempDest <= (others=> '0');
                    tempAdd <= (others=> '0');
                    tempPayload <= (others=> '0');
                    frame_ready <= '0';
            END CASE;
        END IF;
        -- Update frame_out based on frame_ready signal
        IF frame_ready = '1' THEN
            frame_out <= tempOut;
        END IF;
    END PROCESS;
END ARCHITECTURE rtl;
```

Code 2 - Switchport.vhd

- SwitchCAM

  Komponen yang menyimpan MAC Table untuk menentukan port mana yang akan menerima frame apabila terdapat frame yang dikirim melalui switch. Mengambil inspirasi dari CAM atau Content Addressable Memory. SwCAM akan mengimport

Mac-Address table dari file (MacTable.txt) yang disediakan untuk disimpan sementara seolah mereplika fungsi RAM yang mengambil input dari NVRAM.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

entity SwCAM is --simple cam to buffer;
    port (
        main_clk    :   in  std_logic;
        main_rst    :   in  std_logic;
        enable      :   in  std_logic;
        rw_bit      :   in  std_logic;
        mac_find    :   in  std_logic_vector(47 downto 0);
        port_out    :   out std_logic_vector(3 downto 0);
--assuming a switch with max 24 ethernet-port, so max bit is 2^5
        hit_flag    :   out std_logic_vector(1 downto 0) --if
found '11', not found '10';
    );
end entity SwCAM;

architecture rtl of SwCAM is
    constant max_port : integer := 24;
    type State_Type is (LOAD, ACTIVE, READ, WRITE, ASSIGN,
COMPLETE);
    signal state : State_Type := LOAD;
    type MAC_Arr is array (0 to max_port) of std_logic_vector(47
downto 0);
    signal MAC : MAC_Arr := ( --initial cam value
        0 => "010101010101010101010101010101010101010101010101",
--cam(0) would be the error assigned value
        1 => "000000000000000000000000000000000000000000000000",
        2 => "000000000000000000000000000000000000000000000000",
        3 => "000000000000000000000000000000000000000000000000",
        4 => "000000000000000000000000000000000000000000000000",
        5 => "000000000000000000000000000000000000000000000000",
```

```
        6 => "000000000000000000000000000000000000000000000000",
        7 => "000000000000000000000000000000000000000000000000",
        8 => "000000000000000000000000000000000000000000000000",
        9 => "000000000000000000000000000000000000000000000000",
        10 => "000000000000000000000000000000000000000000000000",
        11 => "000000000000000000000000000000000000000000000000",
        12 => "000000000000000000000000000000000000000000000000",
        13 => "000000000000000000000000000000000000000000000000",
        14 => "000000000000000000000000000000000000000000000000",
        15 => "000000000000000000000000000000000000000000000000",
        16 => "000000000000000000000000000000000000000000000000",
        17 => "000000000000000000000000000000000000000000000000",
        18 => "000000000000000000000000000000000000000000000000",
        19 => "000000000000000000000000000000000000000000000000",
        20 => "000000000000000000000000000000000000000000000000",
        21 => "000000000000000000000000000000000000000000000000",
        22 => "000000000000000000000000000000000000000000000000",
        23 => "000000000000000000000000000000000000000000000000",
        24 => "000000000000000000000000000000000000000000000000"
    );
    -- Procedure to fill CAM
    procedure fill_cam_from_file(
        signal macs  : inout MAC_Arr;                    -- CAM
array to be filled
        file_name  : in string                           -- File
name to read from
    ) is
        variable mac            : MAC_Arr;
        file mac_file           : text open read_mode is
file_name;
        variable line_buffer    : line;
        variable value          : std_logic_vector(47 downto 0);
    begin
        mac(0) :=
"010101010101010101010101010101010101010101010101";
        for i in 1 to max_port loop
            mac(i) := (others=>'0');
```

```
        end loop;
        -- Read the file and fill the MAC
        for i in 1 to max_port-1 loop
            if (not endfile(mac_file)) then
                readline(mac_file, line_buffer);
                read(line_buffer, value);
                mac(i) := value;
            else
                exit;
            end if;
        end loop;
        macs <= mac;
    end procedure;

    signal portOut     : std_logic_vector(3 downto 0);
    signal hitFlag     : std_logic_vector(1 downto 0) := "00";
    signal macIn       : std_logic_vector(47 downto 0);
    -- Procedure to find MAC
    procedure find_MAC(
        signal macs     : in MAC_Arr;
        signal portout  : out std_logic_vector(3 downto 0);
        signal hit      : out std_logic_vector(1 downto 0);
        signal macIn    : in std_logic_vector(47 downto 0)
    ) is
        variable portoutd : std_logic_vector(3 downto 0);
        variable hitd     : std_logic_vector(1 downto 0);
        variable mac : MAC_Arr := macs;
    begin
        for i in 1 to max_port-1 loop
            if (mac(i) = macIn) then
                portoutd := std_logic_vector(to_unsigned(i, 4));
                hitd := "11";
                exit; -- exit the loop if found
            else hitd := "00";
            end if;
```

```
        end loop;
        if (hitd = "00") then
            portoutd := "0000";
            hitd := "10";
        end if;
        portout <= portoutd;
        hit <= hitd;
    end procedure;
    signal k : integer := max_port;
--processes
begin
    process (main_clk, mac_find)
    begin
        if main_rst = '1' then
            state <= LOAD;
            port_out <= (others => '0');
            hit_flag <= "00";
        elsif rising_edge(main_clk) then
            case state is
                when LOAD =>
                    port_out <= (others => '0');
                    hit_flag <= "00";
                    fill_cam_from_file(MAC, "macTable.txt");
                    state <= ACTIVE;
                when ACTIVE => --constantly in read mode if not
write
                    macIn <= mac_find;
                    if (enable = '1') then
                        if (rw_bit = '0') then
                            state <= READ;
                        else state <= WRITE;
                        end if;
                    else state <= ACTIVE;
                    end if;
                when READ =>
```

```
                    find_MAC(MAC, portOut, hitFlag, macIn);
                    if (k > 0 and hitFlag = "00") then
                        state <= READ;
                        k <= k - 1;
                    else state <= ASSIGN;
                    end if;
                when WRITE =>
                    state <= ASSIGN;
                    --not now, future update;
                when ASSIGN =>
                    k <= max_port;
                    port_out <= portOut;
                    hit_Flag <= hitFlag;
                    state <= COMPLETE;
                when COMPLETE =>
                    state <= ACTIVE;
            end case;
        end if;
    end process;
end architecture rtl;
```

Code 3 - SwCAM.vhd

- Frame Encoder

  Komponen yang berfungsi untuk menyusun payload, destination address, dan source address menjadi sebuah frame

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

entity frame_encoder is -- frame encoder
  port (
    frame_out : out std_logic_vector(167 downto 0); -- Assuming
168-bit frame
    dest_mac : in std_logic_vector(47 downto 0);
    src_mac  : in std_logic_vector(47 downto 0);
```

```
    payload_byte : in std_logic_vector(7 downto 0)
  );
end entity frame_encoder;


architecture rtl of frame_encoder is
begin
  process (dest_mac, payload_byte)
  begin
    frame_out <= "0101010110101011" & dest_mac & src_mac &
"0000100000000000" & payload_byte &
"11111111111111111111111111111111"; -- Gathering all the fields of
the frame
  end process;
end architecture rtl;
```

Code 4 - frame_encoder.vhd

- Frame Decoder

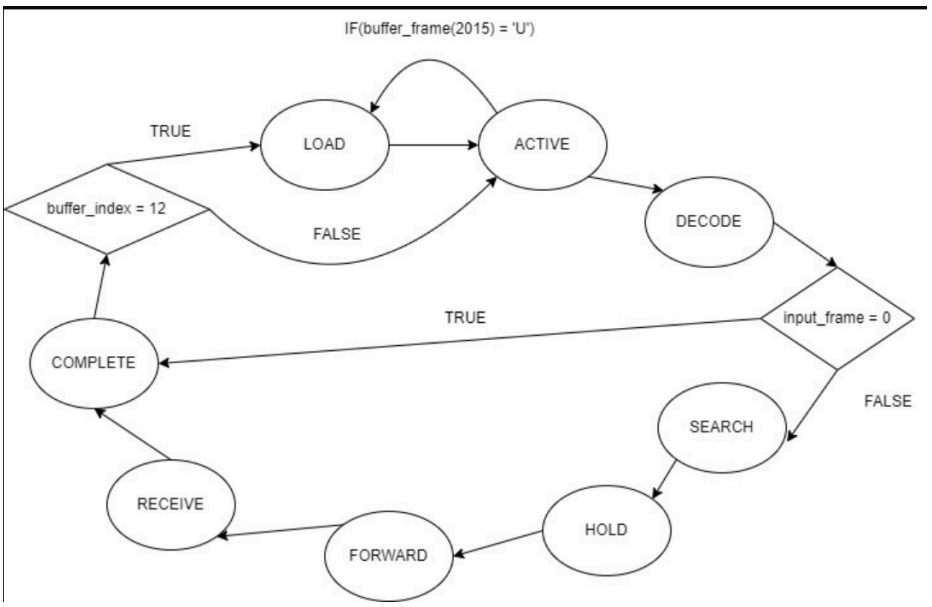
  Komponen yang berfungsi untuk memecah frame yang diterima menjadi data-data seperti payload, dest. Address, dan source address menjadi sebuah frame

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


entity frame_decoder is -- frame decoder
  port (
    frame_in : in std_logic_vector(167 downto 0); -- Assuming
   168-bit frame
    dest_mac : out std_logic_vector(47 downto 0);
    src_mac  : out std_logic_vector(47 downto 0);
    payload_byte : out std_logic_vector(7 downto 0)
  );
end entity frame_decoder;


architecture rtl of frame_decoder is
begin
  process (frame_in)
```

```
  begin
    dest_mac <= frame_in(151 downto 104);
    src_mac <= frame_in(103 downto 56);
    payload_byte <= frame_in(39 downto 32);
  end process;
end architecture rtl;
```

Code 5 - frame_decoder.vhd

Program ini memiliki 8 State. Antara lain LOAD, yang berfungsi untuk mengisi buffer dengan frame di switchport. ACTIVE merupakan state ketika switch sudah dalam keadaan aktif. DECODE merupakan state ketika switch menerima input frame dan memecahkan frame tersebut menjadi data-data seperti payload, dest. address, dan source address. SEARCH merupakan state ketika switch mencari port yang mac addressnya sesuai dengan mac address tujuan pada frame yang masuk. HOLD merupakan state ketika switch menahan data dalam signal buffer. ASSIGN merupakan state setelah HOLD untuk melakukan assign pada port setelah search dan decode, sebenarnya hanya untuk *passing state* saja. FORWARD merupakan state ketika ada port yang ingin mengirim frame pada port lain. RECEIVE merupakan state ketika terdapat port yang menerima input dari port lain. COMPLETE merupakan state ketika seluruh operasi telah selesai.



Gambar 1. State Diagram

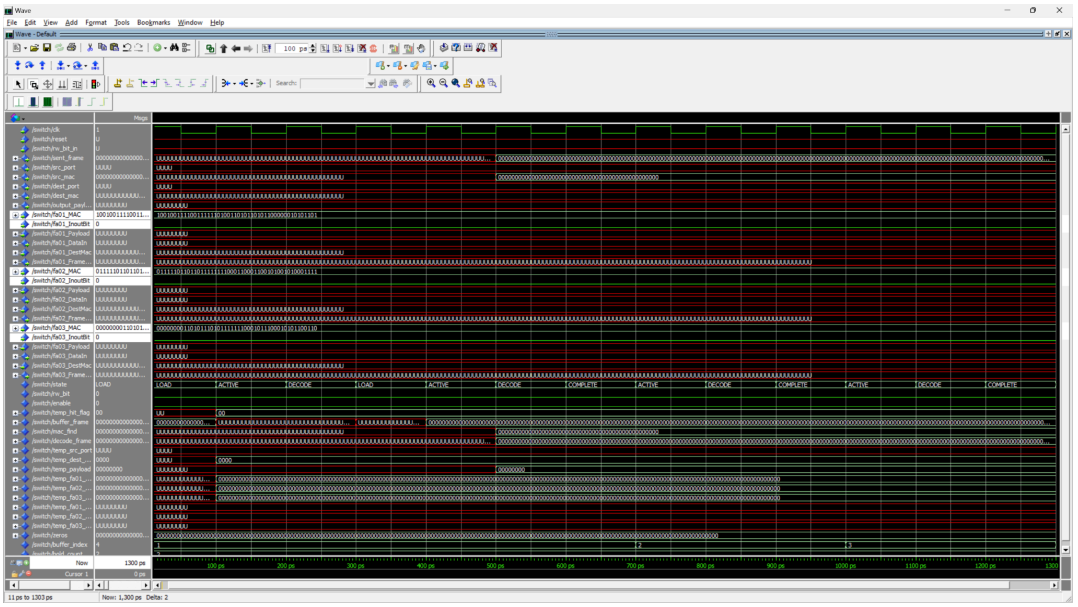
CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

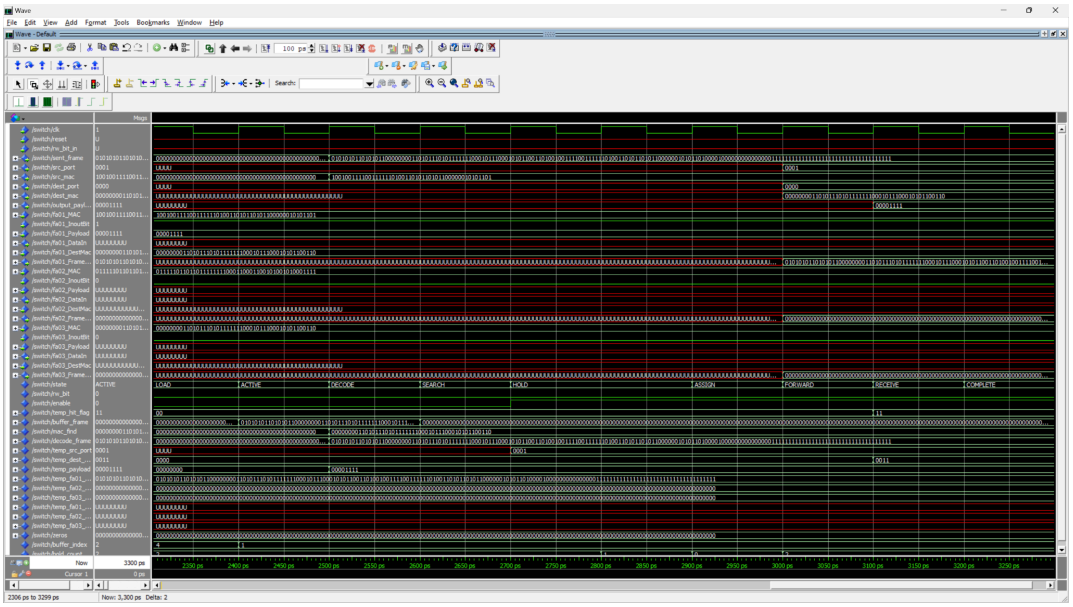
Case 1 - MAC Address Assigned, idle state

Purpose : Initial Assignment Test



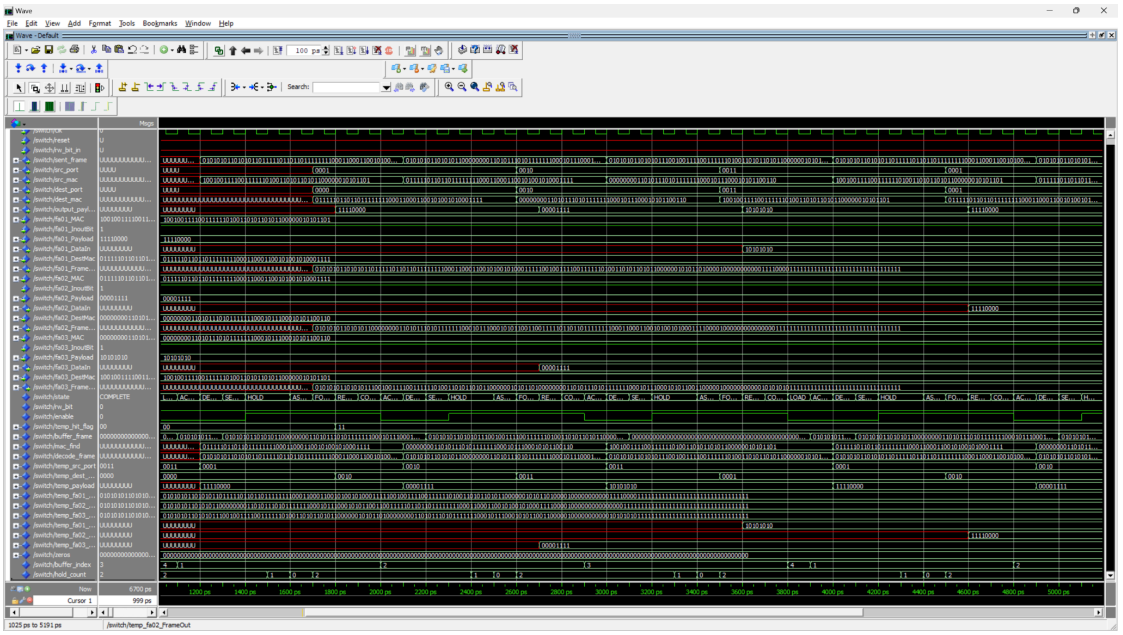
Case 2 - Single Send, Fa01 Send to Fa03

Purpose : Frame Sending Test



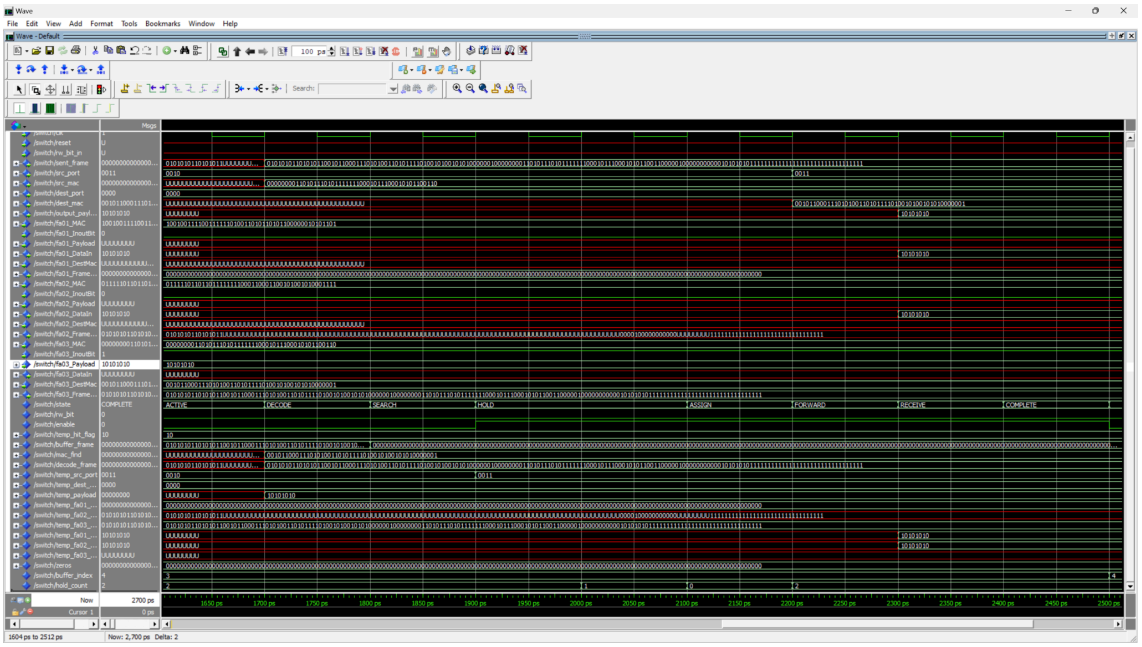
Case 3 - Triple Send, Fa01 to Fa02, Fa02 to Fa03, and Fa03 to Fa01,

Purpose : Check Buffer System



Case 4 - Fa03 Send to Unknown MAC

Purpose : Test Broadcast



3.2 RESULT

Untuk kemudahan design awal, switch didesain hanya menggunakan 3 switchport yakni Fa01, Fa02 dan Fa03. Input yang diterima adalah Clock, Reset, Read/Write bit, serta MAC Address, Inout Bit dan Destination Mac Address untuk masing-masing switchport.

Case 1 :

Pada case satu dilakukan assignment hanya kepada Mac Address masing-masing port dan Inoutbit-nya, yaitu 0 untuk semua switchport. Hasil yang diharapkan adalah inisialisasi value pada temporary signal yang ada

Case 2 :

Case dua dilakukan untuk menguji pengiriman oleh satu port dengan memastikan frame encoder dan Switch CAM bekerja dengan baik. Output yang diharapkan yaitu Frame_Out yang tepat dengan assignment value yang tepat.

Case 3 :

Case tiga dilakukan untuk menguji buffer pengiriman pada frame dengan memastikan buffer bekerja dengan baik, output yang diharapkan yaitu Decode Frame yang sesuai urutan dan ter-decode dengan tepat.

Case 4 :

Case empat dilakukan untuk menguji fungsi broadcast pada switch dengan mengirimkan frame dengan destination MAC Address yang tidak terdaftar pada MacTable.txt. Sehingga frame akan di sebarkan secara broadcast kecuali ke port datangnya frame.

3.3 ANALYSIS

Desain frame encoder dan decoder menunjukkan pemahaman yang kuat tentang struktur dasar frame jaringan. Sementara decoder dapat memisahkan kembali elemen-elemen tersebut, encoder dapat menggabungkan elemen penting seperti destination MAC address, source MAC address, dan payload byte dalam format yang konsisten. Namun, fitur keamanan seperti checksum, yang sangat penting untuk menjamin bahwa data tetap aman selama pengiriman, tidak ada dalam desain ini. Selain itu, meskipun implementasinya mudah,

ketidakmampuan untuk menangani frame dengan panjang variabel dapat menjadi masalah dalam situasi yang lebih kompleks.

Pengujian yang dilakukan dalam empat kasus menunjukkan fungsi inti switch dan efektivitas desain yang dibuat. Case 1 memastikan nilai awal untuk setiap switchport. Ini sangat penting untuk mencegah error yang disebabkan oleh nilai acak pada sinyal internal. Dalam kasus 2, proses pengiriman frame melalui encoder dan integrasi dengan Switch CAM menunjukkan bahwa desain dapat menangani situasi pengiriman data sederhana. Dalam kasus 3, buffer diuji untuk memastikan kemampuan menangani aliran frame secara konsisten, dan hasilnya menunjukkan bahwa mekanisme buffering berfungsi dengan baik. Ketika MAC address tujuan tidak ditemukan dalam tabel CAM, kasus 4 menguji mekanisme broadcast. Hasilnya menunjukkan bahwa desain dapat menangani situasi ini dengan benar, meskipun fitur broadcast berdampak pada efisiensi lalu lintas jaringan.

CHAPTER 4

CONCLUSION

Desain frame encoder dan decoder telah memenuhi persyaratan utama untuk manajemen frame jaringan dengan format tetap. Komponen seperti MAC address, payload, dan header frame yang digabungkan menunjukkan implementasi yang terstruktur. Selain itu, decoder mengekstrak komponen dengan tepat, menunjukkan bahwa desain ini cukup untuk situasi pengiriman data sederhana. Namun, masih ada ruang untuk perbaikan karena elemen seperti keamanan dan integritas data, yang biasanya diatasi dengan checksum atau validasi tambahan, belum diterapkan.

Sifat switch yang diuji dalam empat skenario menunjukkan kemampuan sistem untuk memenuhi kebutuhan dasar jaringan. Pada Kasus 1, proses inisialisasi memastikan nilai awal stabil, yang penting untuk mencegah kesalahan. Pada Kasus 2 dan 3, pengiriman frame dan pengelolaan buffer berjalan dengan baik, menunjukkan kerja sama yang telah dirancang antara encoder, decoder, dan mekanisme perpindahan. Dalam Kasus 4, fungsi broadcast juga berhasil; namun, dampak pada efisiensi lalu lintas jaringan harus dipertimbangkan lebih lanjut dalam pengembangan.

Secara keseluruhan, desain ini berhasil menampilkan kemampuan dasar yang diperlukan untuk mengelola lalu lintas jaringan, seperti pengkodean, dekode, dan fungsi switching. Namun, untuk memenuhi kebutuhan jaringan modern yang lebih kompleks, diperlukan pengembangan tambahan, seperti penerapan pengelolaan tabel MAC yang dinamis, pengoptimalan mekanisme broadcast, dan peningkatan efisiensi pengelolaan buffer. Dengan menambahkan fitur-fitur ini, desain akan lebih mampu menangani berbagai skenario jaringan yang membutuhkan skalabilitas dan keandalan yang lebih tinggi.

REFERENCES

[1] GeeksforGeeks, “What is a network switch, and how does it work?,” GeeksforGeeks, Oct. 02, 2021. https://www.geeksforgeeks.org/what-is-a-network-switch-and-how-does-it-work/

[2] GeeksforGeeks, “VHDL Very High Speed Integrated Circuit Hardware Description Language,” GeeksforGeeks, Jun. 25, 2024. https://www.geeksforgeeks.org/vhdl-very-high-speed-integrated-circuit-hardware-description-language/

[3] D. Williams, “Implementing a Finite State Machine in VHDL,” Allaboutcircuits.com, Dec. 23, 2015. https://www.allaboutcircuits.com/technical-articles/implementing-a-finite-state-machine-in-vhdl/

[4] Å. Braathen, “Designing FSMs in VHDL - EmLogic AS,” EmLogic AS - The Norwegian Embedded Systems & FPGA Design Centre, Dec. 18, 2023. https://emlogic.no/2023/12/designing-fsms-in-vhdl/

[5] GeeksForGeeks, “Encoder in Digital Logic,” *GeeksforGeeks*, Oct. 24, 2017. https://www.geeksforgeeks.org/encoder-in-digital-logic/

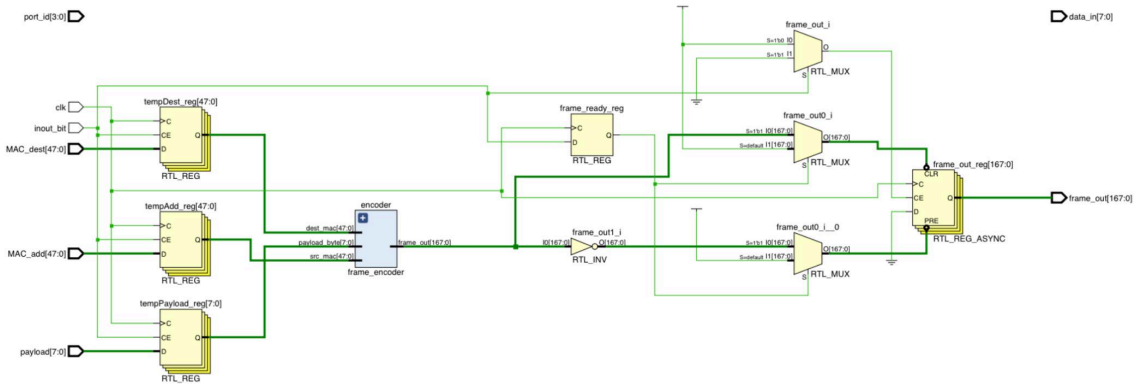
[6] “Encoders and Decoders in Digital Logic - GeeksforGeeks,” *GeeksforGeeks*, May 08, 2018. Available: https://www.geeksforgeeks.org/encoders-and-decoders-in-digital-logic/

[7] “Virtual Machine State Diagram,” Microsoft.com, May 31, 2018. https://learn.microsoft.com/id-id/previous-versions/windows/desktop/msvs/virtual-machine-state-diagram

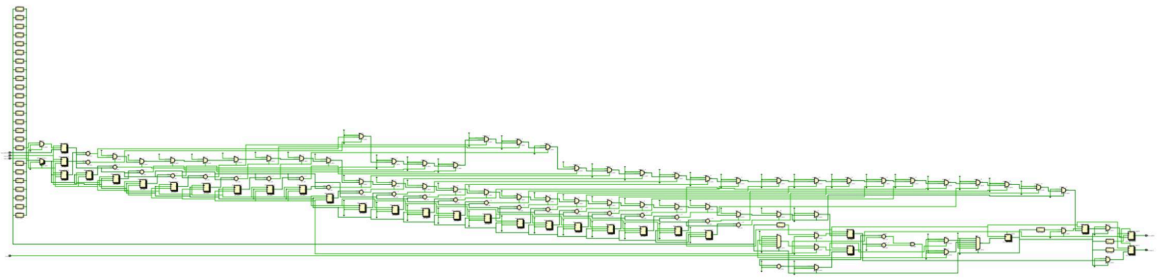
APPENDICES

Appendix A: Project Schematic

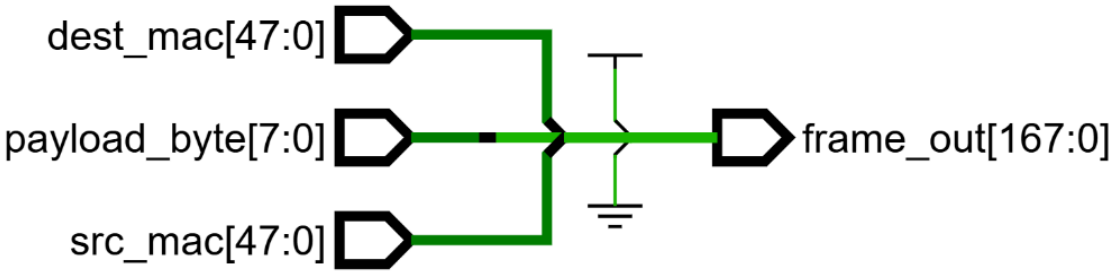
Put your final project latest schematic here



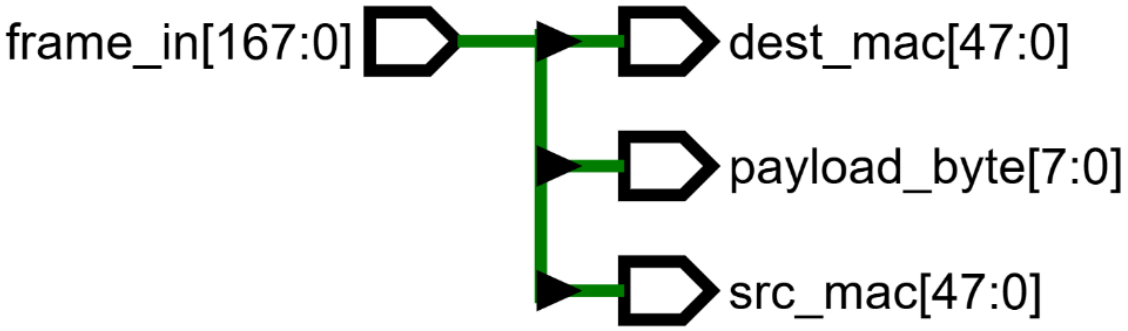
Gambar 2.Switchport Synthesize



Gambar 3. SwCAM Synthesize



Gambar 4. Frame Encoder Synthesize



Gambar 5. Frame Decoder Synthasize

Appendix B: Documentation